University of Southampton

University of Southampton Research Repository

Copyright © and Moral Rights for this thesis and, where applicable, any accompanying data are retained by the author and/or other copyright owners. A copy can be downloaded for personal non-commercial research or study, without prior permission or charge. This thesis and the accompanying data cannot be reproduced or quoted extensively from without first obtaining permission in writing from the copyright holder/s. The content of the thesis and accompanying research data (where applicable) must not be changed in any way or sold commercially in any format or medium without the formal permission of the copyright holder/s.

When referring to this thesis and any accompanying data, full bibliographic details must be given, e.g.

Thesis: Author (Year of Submission) "Full thesis title", University of Southampton, name of the University Faculty or School or Department, PhD Thesis, pagination.

Data: Author (Year) Title. URI [dataset]

UNIVERSITY OF SOUTHAMPTON

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Interpretable Multiple Instance Learning

by

Joseph Arthur Early

ORCiD: 0000-0001-7748-9340

A thesis submitted for the degree of
Doctor of Philosophy

June 2024

University of Southampton

Abstract

Faculty of Engineering and Physical Sciences
School of Electronics and Computer Science

Doctor of Philosophy

Interpretable Multiple Instance Learning

by Joseph Arthur Early

With the rising use of Artificial Intelligence (AI) and Machine Learning (ML) methods, there comes an increasing need to understand how automated systems make decisions. Interpretable ML provides insight into the underlying reasoning behind AI and ML models while not stifling their predictive performance. Doing so is important for many reasons, such as facilitating trust, increasing transparency, and providing improved collaboration and control through a better understanding of automated decision-making. Interpretability is very relevant across many ML paradigms and application domains.

Multiple Instance Learning (MIL) is an ML paradigm where data are grouped into bags of instances, and only the bags are labelled (rather than each instance). This is beneficial in alleviating expensive labelling procedures and can be used to exploit the underlying structure of data. This thesis investigates how interpretability can be achieved within MIL. It begins with a formalisation of interpretable MIL, and then proposes a suite of model-agnostic post-hoc methods. This work is then extended to the specific application domain of high-resolution satellite imagery, using novel inherently interpretable MIL approaches that operate at multiple resolutions.

Following on from work in the vision domain, new methods for interpretable MIL are developed for sequential data. First, it is explored in the domain of Reward Modelling (RM) for Reinforcement Learning (RL), demonstrating that interpretable MIL can be used to not only understand a model but also improve its predictive performance. This is mirrored in the application of interpretable MIL to Time Series Classification (TSC), where it is integrated into state-of-the-art methods and is able to improve both their interpretability and predictive performance. The integration into existing models to provide inherent interpretability means these benefits are delivered with little additional computational cost.

Contents

List of Figures

List of Tables

List of Acronyms

AI Artificial Intelligence

AUPC Area Under the Precision Curve

AOPCR Area Over the Perturbation Curve to Random

AURC Area Under the Recall Curve

AUROC Area Under the Receiver Operating Characteristic

CAM Class Activation Mapping

CRC Colorectal Cancer

CSC Concatenated Skip Connection

DL Deep Learning

EO Earth Observation

FE Feature Extractor

GAP Global Average Pooling

GNN Graph Neural Network

GPU Graphics Processing Unit

HN Head Network

HPC High Performance Computing

LCC Land Cover Classification

LIME Local Interpretable Model-agnostic Explanations

LSTM Long Short-Term Memory

MAE Mean Absolute Error

MIL Multiple Instance Learning

ML Machine Learning

MIOU Mean Intersection Over Union

MRMO Multi-Resolution Multi-Out

MRSO Multi-Resolution Single-Out

MSE Mean Square Error

NDR Natural Disaster Response

NDCG@n Normalised Discounted Cumulative Gain at n

RL Reinforcement Learning

RM Reward Modelling

RMSE Root Mean Square Error

RS Remote Sensing

S2P Scene-to-Patch

SHAP Shapley Additive Explanations

SIVAL Spatially Independent, Variable Area, and Lighting

SMIL Standard MIL

SOTA State-Of-The-Art

SR Single Resolution

TOM Theory Of Mind

TSC Time Series Classification

XAI Explainable AI

Mathematical Notation

General

Class space $\mathbb{Z}_C$

d-dimensional space $\mathbb{R}^d$

Dataset $\mathcal{D}$

Dataset (train) $\mathcal{D}_{Train}$

Dataset (validation) $\mathcal{D}_{Val}$

Dataset (test) $\mathcal{D}_{Test}$

Model output (logits) $\mathbf{z}$

Number of classes C

Predictive model M

Supervised Learning

Input $\mathbf{x}$

Input space $\mathcal{X}$

Label $\mathbf{y}$

Label space $\mathcal{Y}$

Prediction $\hat{\mathbf{y}}$

Multiple Instance Learning

Bag $\mathbf{X}$

Bag label $\mathbf{Y}$

Bag label space $\mathcal{Y}_B$

Bag prediction $\hat{\mathbf{Y}}$

Bag size k

Bag-level function F

Instance $\mathbf{x}_i$

Instance interpretability value ϕ_i

Instance interpretability rank order $\mathbf{r}$

Instance interpretability ranking r_i

Instance label $\mathbf{y}_i$

Instance label space $\mathcal{Y}_I$

Instance prediction $\hat{\mathbf{y}}_i$

Instance-level function f

`MILLI` Interpretability

Coalition vector $\mathbf{v}$

Coalition matrix $\mathbf{V}$

`MILLI` **ranking weight kernel** π_R

`MILLI` **weight kernel** π_M

Surrogate model G_c

Scene-to-Patch Models

Number of extracted patches b

Number of scales (resolution) s_n

Patch size p

Scale (resolution) s

Scale 0 (lowest resolution) $s = 0$

Scale 1 (middle resolution) $s = 1$

Scale 2 (highest resolution) $s = 2$

Scale main (combined resolution) $s = m$

Reinforcement Learning and Reward Modelling

Agent action (at t) $\mathbf{a}_t$

Agent action space $\mathcal{A}$

Discrete time step t

Environment state (at t) $\mathbf{s}_t$

Environment state space $\mathcal{S}$

Hidden state dynamics function δ

Hidden state dynamics func. (learnt) δ'

Hidden state (at t) $\mathbf{h}_t$

Hidden state (initial) $\mathbf{h}_0$

Hidden state (initial learnt) $\mathbf{h}'_0$

State dynamics function D

Return function G

Reward function R

Reward function (learnt) R'

Trajectory ξ

Trajectory class space $\mathcal{C}_{Task}$

Trajectory classification function C_{Task}

Trajectory length T

Trajectory space Ξ

Time Series Classification

Attention module ψ_{ATTN}

Attention value a

Classifier module ψ_{CLF}

Discrete time step t

Feature embedding $\mathbf{z}$

Feature embedding length d

Feature embedding matrix $\mathbf{Z}$

Feature extractor module ψ_{FE}

Number of time series channels h

Time series length T

Declaration of Authorship

I declare that this thesis and the work presented in it is my own and has been generated by me as the result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a research degree at this University;
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
3. Where I have consulted the published work of others, this is always clearly attributed;
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
5. I have acknowledged all main sources of help;
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself;
7. Parts of this work have been published as:
   (a) Joseph Early, Christine Evers, and Sarvapali Ramchurn. Model agnostic interpretability for multiple instance learning. In *International Conference on Learning Representations*, 2022c. URL https://openreview.net/forum?id=KSSfF5lMIAg
   (b) Joseph Early, Tom Bewley, Christine Evers, and Sarvapali Ramchurn. Non-Markovian reward modelling from trajectory labels via interpretable multiple instance learning. *Advances in Neural Information Processing Systems*, 35, 2022a. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b157cfde6794e93b2353b9712bbd45a5-Paper-Conference.pdf

(c) Joseph Early, Ying-Jung Deweese, Christine Evers, and Sarvapali Ramchurn. Scene-to-patch Earth observation: Multiple instance learning for land cover classification. *NeurIPS Workshop: Tackling Climate Change with Machine Learning*, 2022b. URL https://arxiv.org/abs/2211.08247

(d) Joseph Early, Ying-Jung Chen Deweese, Christine Evers, and Sarvapali Ramchurn. Extending scene-to-patch models: Multi-resolution multiple instance learning for Earth observation. *Environmental Data Science*, 2, 2023. URL https://doi.org/10.1017/eds.2023.30

(e) Joseph Early, Gavin KC Cheung, Kurt Cutajar, Hanting Xie, Jas Kandola, and Niall Twomey. Inherently interpretable time series classification via multiple instance learning. *Internation Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xriGRsoAza

Signed:..... Date:.....

Acknowledgements

First and foremost, I would like to thank my University of Southampton supervisors Gopal Ramchurn and Christine Evers. Their input through regular meetings and feedback on my work was invaluable. They gave a great deal of insight and lent their expertise during all stages of my PhD. In addition, thank you to all those in the AIC group, the IRIDIS support team, and the Alan Turing Institute's Academic Services.

Throughout my PhD, I have been fortunate enough to collaborate with many researchers from different institutions and academic backgrounds. In particular, I would like to thank Tom Bewley, whom I met through the Alan Turing Institute and collaborated with on a paper at the intersection of our two PhDs, and Ying-Jung Chen Deweese, a co-author on two of my published works who supported me through the Climate Change AI mentorship programme. In addition, I am grateful to have had the opportunity to work alongside and publish a paper with Gavin Cheung, Kurt Cutajar, Hanting Xie, Jas Kandola, and Niall Twomey from the Anomaly Detection and Insights team during my internship with Amazon Prime Video.

My close friends, many of whom are fellow PhD students, have been a continuous source of inspiration and motivation. From Southampton, I would like to thank Tom, Leonard, Callum, Loki, Harry, Anna, Spencer, Ella, Eve, Rich, Alex, Greg, and Andrei. From London, I would like to thank Giulia and Jennifer for joining me in founding and co-leading the Alan Turing Institute's Entrepreneurship Interest Group, and all those who were part of the Turing's 2019 Doctoral Cohort. I am hugely appreciative of my oldest friends Matt, Elliot, Cameron, Jamie, and Barney for their 'pastoral' support and for reminding me that the world exists outside of academia. My greatest thanks goes to Amy Jackson, my partner and best friend throughout both of our PhDs — seeing you become Dr. Jackson a year before me was wonderful to experience and also a source of great motivation.

I would certainly not be at this stage without the love and inspiration of my wonderful parents Claire and Lawrence, and my sister Holly. They have always been there for me: starting with my early adventures in programming (Lego Mindstorms, BASIC Stamp electronics, and Minecraft modding), through my undergraduate degree, and finally nodding along when I ramble about my PhD research. Thank you for all your support and kindness.

To Mum and Dad, for your inspiration and unwavering support.

Chapter 1

Introduction

Tasks we take for granted, for example differentiating between images of cats and dogs, are very difficult to program. The difficulty in writing software for certain tasks comes from the fact that it is difficult to define a concrete set of rules that produce a desired solution. Instead, it is easier to design a system that learns the rules itself. However, we cannot be certain what such a system will learn, nor is it easy to understand what it has learnt. Are these systems making the correct decisions, and are they making those decisions for the right reasons?

The last decade has shown an unprecedented increase in the use of Artificial Intelligence (AI) and Machine Learning (ML) in research and industry. Owing to improvements in computing power and the greater availability of large, rich datasets, it is now possible to develop powerful autonomous systems that can complete an ever-increasing number of complex tasks. Due to the increased autonomy of the systems and their inherent complexity, it becomes more difficult (and more important) for human supervisors to ascertain the system's intentions and decision-making process. This leads to the need for autonomous systems to be able to explain their decisions in order to facilitate human understanding of a system's decision-making process.

The difficulty in understanding how an autonomous system makes decisions is partly due to the complexity of the system. With simpler models, such as linear regression, it is possible to understand the role of each individual parameter and how they affect the overall decision-making process. However, with complex systems such as neural networks, this inherent understanding is lost. Therefore, in order to utilise these more complex, powerful models to solve difficult tasks, Explainable AI (XAI) and interpretable methods are required. This trade-off between predictive performance (how capable the model is of achieving strong performance on difficult tasks) and interpretability (how readily the model can be inherently understood) is summarised in Figure 1.1.

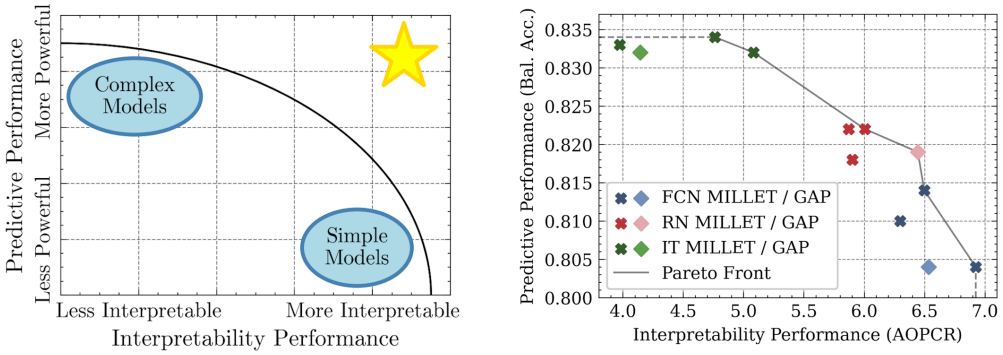

FIGURE 1.1: The predictive power vs interpretability trade-off.

Left: A general overview of the trade-off. As model complexity increases, models become more powerful (greater predictive performance) but sacrifice their interpretability. For example, it is more difficult to interpret a neural network than a linear regression model, but the former can achieve strong performance on more difficult tasks. Ideally, we want models that are both highly interpretable and powerful (indicated with the star in the top right).

Right: An actual example of this trade-off from the research presented in this thesis. Forgoing the specific details for now, the general trend is that as model complexity increases across the three models (FCN to RN to IT, with FCN being the least complex), predictive performance improves while interpretability decreases. However, the `MILLET` method (proposed in this thesis) is able to boost interpretability whilst maintaining predictive performance for the IT and FCN models, moving towards the desired location in the top right. For more details on this diagram, see Chapter 7.

The terms interpretability and explainability are often used interchangeably, but strictly there is a difference between the two. Model interpretability involves understanding the inner workings of a system, i.e., 'opening the black box', whereas explainability focuses on producing explanations that are similar to those a human would use. Interpretability often produces technical evaluations of decision-making, which can only be properly understood by domain experts. Whilst these interpretations can be useful in some contexts, further steps are required to produce explanations that are appropriate for a non-technical audience.

Multiple Instance Learning (MIL) is a specific paradigm in the wider scope of ML. Unlike traditional supervised learning, where each piece of data is given a label, MIL uses data that is grouped into bags, where each bag has a label. This makes labelling data more efficient, as each of the instances within a bag does not need to be labelled, only the bag as a whole — see Figure 1.2 for an overview. This is particularly important with the large-scale datasets that are often used in modern machine learning — developing methods that work with weaker labels (such as those in MIL) reduces both labelling time and cost.

Beyond reducing labelling costs, an additional benefit of MIL is that it is able to exploit underlying structure in data. For example, as explored in Chapter 7, additional information such as temporal relationships and structure is often discarded during training/inference. MIL is able to utilise such structure to achieve improved predictive

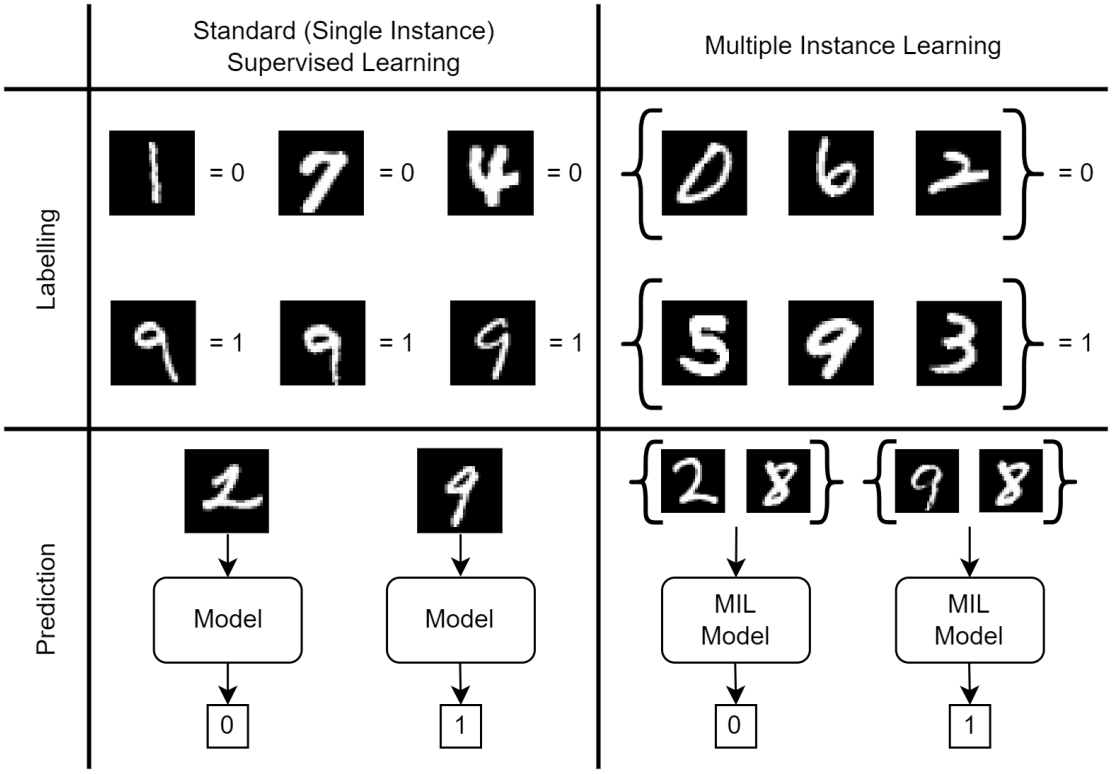

FIGURE 1.2: In this overview of MIL, two versions of a binary classification problem are presented. In the standard (single instance) supervised learning case, images representing the digit 9 are labelled as class 1, and all other digits are labelled as class 0. The model takes a single image as input as predicts class 0 or 1. For the MIL case, images are grouped into bags, and a bag is given the label of class 1 if it contains a 9, otherwise, it is labelled as class 0. This represents the same underlying classification problem, but labelling only occurs at the bag level, i.e., the individual images are unlabelled. One advantage of the MIL, from a labelling perspective, is that a bag can be labelled once a single 9 is observed, i.e., the human labeller does not need to label every single 9. The model then takes a bag as input and makes a prediction of class 0 or 1. In the example shown here, replacing the 2 with a 9 changes the model's prediction from class 0 to class 1. Note MIL bags can be much larger than the sizes (2 or 3 instances) shown here. Images of digits taken from MNIST (Lecun et al., 1998).

performance. Furthermore, as discussed in Chapter 4, MIL problems can be used capture relationships between instances, such as co-occurrence, and through interpretability, these relationships can be uncovered.

In the context of MIL, interpretability is about understanding which of the instances within a bag were used to make decisions. The number of key instances in a bag (those that determine the bag label) is dependent on the specific MIL problem and domain. Many instances with complex relationships could determine the bag label, or it could be that a single instance in a bag is all that matters. Interpretable MIL approaches need to be flexible enough to deal with this variation, along with several other requirements (as explored in Chapter 4).

In the remainder of this introduction, the research questions explored in this thesis are

outlined (Section 1.1), followed by the research contributions (Section 1.2). Finally, the structure of the remainder of the thesis is presented (Section 1.3).

1.1 Research Questions

The research presented in this thesis answers three open research questions:

1. **What is an appropriate definition of MIL interpretability, and how can it be evaluated?**
   While prior works (Ilse et al., 2018; Liu et al., 2012; Tu et al., 2019; Wang et al., 2018) had some notion of MIL interpretability, it was not rigorously defined or the main focus of the research. Rather, interpretability was considered a secondary outcome of new methods and often not formally evaluated. Typically, interpretability in existing MIL work was presented as visualisations of decision-making, which, while somewhat insightful, lacked the rigour and consistency to facilitate comparisons between methods.

2. **How can the labelling benefits of MIL be further utilised, and what does this mean for interpretability?**
   As stated above, MIL does not require labelling for all data instances. Instead, data are grouped into bags, and only the bags need to be labelled. This means labelling data becomes faster and cheaper. A prime example of this is in MIL for histopathology, where each image only requires a high-level label rather than annotations for every nuclei (Hashimoto et al., 2020; Ilse et al., 2018; Javed et al., 2022; Li et al., 2021a; Li and Zhang, 2020; Marini et al., 2021). Much of the existing MIL research is focused on vision classification, so an open question is whether similar MIL benefits can be found in other domains.

3. **As well as facilitating understanding of automated decision-making, can MIL interpretability be utilised to improve underlying model performance?**
   While gaining insight into how a model makes decisions is the primary goal of interpretability, it has further use cases. One such extension is leveraging interpretability to improve the performance of the underlying model, i.e., by improving interpretability, can we also improve performance? Answering this question aims to overcome the view that improving interpretability must always come at the detriment of predictive performance.

1.2 Research Contributions

To answer **RQ1**, the notion of MIL interpretability is formalised, which involves considering the interactions between instances and how they affect the bag label (a common assumption in existing literature is that instances could be treated independently). Through a novel concrete evaluation with respect to these new requirements, existing approaches were found lacking, and as such a suite of new approaches to MIL interpretability is proposed — these are presented in Chapter 4.

An alternative ML paradigm to classification is semantic segmentation, where the objective is to predict a class for every pixel in an image. This typically requires the use of segmentation masks during training. Creating these masks is a time-consuming and expensive labelling process. As such, the use of interpretable MIL for semantic segmentation of high-resolution images is investigated, which does not require these segmentation maps, and contributes towards answering **RQ2**. This was done specifically in the domain of Earth Observation (EO) for climate change applications. Included is a proposal of novel MIL model architectures to utilise multi-resolution inputs, which was shown to improve the quality of visual interpretations. This research is presented in Chapter 5.

To further answer **RQ2** and also answer **RQ3**, MIL methods are developed for use with sequential data. In Reward Modelling (RM) (Reinforcement Learning (RL) from human feedback), interpretable MIL models are proposed that not only demonstrated improved insight into the effect of reward in RL but also produced interpretations that can be fed back into the agent training process to improve performance. This research is presented in Chapter 6. This idea was further explored during a research internship at Amazon, through the creation of new interpretable MIL methods for Time Series Classification (TSC) — a domain where MIL is readily applicable but only has a limited use in prior work. In both RM and TSC, the sequential nature of data lends itself to a MIL representation. For this next phase of research, a key focus was leveraging the ability of existing models and developing '*plug-and-play*' MIL interpretability that could be applied to a wide range of existing State-Of-The-Art (SOTA) methods. This is presented in Chapter 7.

Each of the above research contributions has been published in peer-reviewed conferences and/or journals:

1. Joseph Early, Christine Evers, and Sarvapali Ramchurn. Model agnostic interpretability for multiple instance learning. In *International Conference on Learning Representations*, 2022c. URL https://openreview.net/forum?id=KSSfF5lMIAg (Chapter 4).

2. Joseph Early, Ying-Jung Deweese, Christine Evers, and Sarvapali Ramchurn. Scene-to-patch Earth observation: Multiple instance learning for land cover classification. *NeurIPS Workshop: Tackling Climate Change with Machine Learning*, 2022b. URL https://arxiv.org/abs/2211.08247 (Chapter 5).

3. Joseph Early, Ying-Jung Chen Deweese, Christine Evers, and Sarvapali Ramchurn. Extending scene-to-patch models: Multi-resolution multiple instance learning for Earth observation. *Environmental Data Science*, 2, 2023. URL https://doi.org/10.1017/eds.2023.30 (Chapter 5).

4. Joseph Early, Tom Bewley, Christine Evers, and Sarvapali Ramchurn. Non-Markovian reward modelling from trajectory labels via interpretable multiple instance learning. *Advances in Neural Information Processing Systems*, 35, 2022a. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b157cfde6794e93b2353b9712bbd45a5-Paper-Conference.pdf (Chapter 6).

5. Joseph Early, Gavin KC Cheung, Kurt Cutajar, Hanting Xie, Jas Kandola, and Niall Twomey. Inherently interpretable time series classification via multiple instance learning. *Internation Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xriGRsoAza (Chapter 7).

In addition, I have collaborated on other papers during my PhD that are beyond the scope of this thesis:

1. Saqib Rahman, Joe Early, Matt De Vries, Megan Lloyd, Ben Grace, Gopal Ramchurn, and Timothy Underwood. Predicting response to neoadjuvant therapy using image capture from diagnostic biopsies of oesophageal adenocarcinoma. *European Journal of Surgical Oncology*, 47(1), 2021a. URL https://www.sciencedirect.com/science/article/pii/S0748798320309197.

2. Chris Reed, Keri Grieman, and Joseph Early. Non-Asimov explanations regulating AI through transparency. *Nordic Yearbook of Law and Informatics*, 2021. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3970518.

3. Saqib Rahman, Joseph Early, Ben Sharpe, Megan Lloyd, Matt De Vries, Ben Grace, Sarvapali Ramchurn, and Timothy Underwood. Predicting survival and response to therapy using diagnostic biopsies: A machine learning approach to facilitate treatment decisions for oesophageal adenocarcinoma. *British Journal of Surgery*, 108, 2021b. URL https://doi.org/10.1093/bjs/znab430.185.

4. Sarah Ahmed, Tayyaba Azim, Joseph Early, and Sarvapali Ramchurn. Revisiting deep Fisher vectors: Using Fisher information to improve object classification. In *33rd British Machine Vision Conference BMVC*. BMVA Press, 2022. URL https://bmvc2022.mpi-inf.mpg.de/0900.pdf.

5. Gregory Everett, Ryan J. Beal, Tim Matthews, Joseph Early, Timothy J. Norman, and Sarvapali D. Ramchurn. Inferring player location in sports matches: Multi-agent spatial imputation from limited observations. In *International Foundation for Autonomous Agents and Multiagent Systems*, AAMAS '23, 2023. ISBN 9781450394321. URL https://dl.acm.org/doi/10.5555/3545946.3598821.

6. Keri Grieman and Joseph Early. A risk-based approach to AI regulation: System categorisation and explainable AI practices. *SCRIPTed*, 20, 2023. URL https://script-ed.org/?p=4109.

1.3 Thesis Structure

The remainder of this thesis is split into three parts:

Part I: Preliminaries The first part of this thesis covers the necessary required knowledge prior to presenting the novel research. Chapter 2 introduces general ML concepts for supervised learning, such as data, models, training, and the practicalities of ML (e.g., data cleaning and hyperparameter selection). These general concepts are continually revisited in the remaining chapters. Following the background, Chapter 3 is a literature review of the two key topics covered in this work: interpretability and MIL. General interpretability literature is first explored, followed by a review of MIL and a discussion of how interpretability is applied in MIL.

Part II: Interpretable Multiple Instance Learning for Vision: The next two chapters present research on interpretable MIL for applications in vision. Chapter 4 details a suite of novel approaches to general MIL interpretability, which is applicable to all MIL models. Chapter 5 then dives into a more focused problem, looking at the application of interpretable MIL to EO datasets. This includes the development of novel interpretable multi-resolution methods.

Part III: Interpretable Multiple Instance Learning for Sequential Data: Following the focus on computer vision applications, the next research section focuses on a different domain of problems: sequential data. First, Chapter 6 explores the application of interpretable MIL to reward modelling in RL problems. This is followed by Chapter 7 which proposes new methods for TSC (again using interpretable MIL). In both of these chapters, a key difference to the vision research is the notion of temporal ordering and dependency in the data — RM and TSC both involve sequential data representations through time, which impacts the type of predictive models that are used and has implications for MIL interpretability.

An overall concluding discussion is given in Chapter 8, providing a retrospective analysis of the research presented in this thesis and discussing remaining open challenges in the field of interpretable MIL. Appendices A to D contain additional information organised by research chapter.

Part I

Preliminaries

Chapter 2

Background

This chapter introduces the relevant background for this thesis. It focuses on the key concepts of supervised Machine Learning, laying the foundation for a literature review of Multiple Instance Learning and interpretability in the next chapter. Specifically, this background covers models and data (Section 2.1), training (Section 2.2), neural networks (Section 2.3), and the practicalities of Machine Learning (Section 2.4). Throughout this chapter, a running example of classifying images of huskies and tigers is used to aid understanding.

2.1 Models and Data

In supervised Machine Learning (ML), a piece of labelled data is an item $\mathbf{x}$ (an image, feature vector, video, time series, etc.) with an associated label $\mathbf{y}$. An ML model receives $\mathbf{x}$ as an input and processes it to produce a prediction $\hat{\mathbf{y}}$. Without loss of generality, both $\mathbf{x}$ and $\hat{\mathbf{y}}$ can be considered vectors of some arbitrary size. Later, we will use more precise notation for domain-specific definitions. Given the space of all possible inputs $\mathcal{X}$ and possible predictions/labels $\mathcal{Y}$, an ML model M is a function $M : \mathcal{X} \rightarrow \mathcal{Y}$. The objective of ML is to learn the function that provides the correct mapping for a particular task.

> ***Running Example: Supervised Learning***
>
> Consider an ML model that is intended to classify pictures of huskies and tigers. An input $\mathbf{x} \in \mathcal{X}$ is a picture of a husky or a picture of a tiger, and a prediction $\hat{\mathbf{y}} \in \mathcal{Y}$ is a binary indicator of 0 for husky and 1 for tiger (this is a simplification of model output; see Section 2.2.1). Therefore, the input space $\mathcal{X}$ consists of all possible images of huskies and tigers, while the label space is defined as $\mathcal{Y} \in \mathbb{Z}_2$.

In supervised learning settings, learning the correct function for M (also known as *'fitting'* or *'training'* the model) is achieved by leveraging labelled data. A training dataset $\mathcal{D}_{Train}$ contains $(\mathbf{x}, \mathbf{y})$ pairs — each $\mathbf{x} \in \mathcal{X}$ has a (correct) label $\mathbf{y} \in \mathcal{Y}$. The process used to create these labels could be manual (e.g., requiring expert human knowledge) or generated by some other automated process (e.g., web scraping).

As a minimum, supervised ML problems are set up with two dataset splits: a training set and a test set. The testing dataset $\mathcal{D}_{Test}$ is similar to $\mathcal{D}_{Train}$ but it is not used when fitting the model M. Instead, it is only used for evaluation. This helps identify model overfitting, where M performs very well on $\mathcal{D}_{Train}$ but not on $\mathcal{D}_{Test}$, i.e., it does not generalise beyond the data it has already seen. Typically, overfitting means M has learnt spurious rules that do not accurately capture the relationship between $\mathcal{X}$ and $\mathcal{Y}$. This is also something that can be identified through interpretability.

The actual train/test split used is somewhat arbitrary. However, it is essential to ensure that there is no data leakage, where information from the test dataset is used during training. The original data can be considered as one monolithic collection of $(\mathbf{x}, \mathbf{y})$ pairs, with the division into training and testing data later chosen at random. A typical two-way split is 80/20, where 80% of the data is used for training and 20% is used for testing. Stratified sampling over $\mathcal{Y}$ is often used to preserve the label distribution. Assuming all the data is unique, there is no overlap between the training and testing datasets, and thus no data leakage.

A further extension to train/test splits involves using a validation dataset $\mathcal{D}_{Val}$. The test dataset $\mathcal{D}_{Test}$ should only be used for evaluating the final model performance.

Therefore, the validation dataset $\mathcal{D}_{Val}$ is useful for additional tasks such as hyperparameter tuning (see Section 2.4.3). Using $\mathcal{D}_{Test}$ for such a task would be a form of data leakage. A typical division of the original monolithic dataset when using a validation dataset is 80/10/10, but this could vary depending on the amount of data.

Various extensions to dataset sampling exist. One such method is k-fold cross-validation, which partitions the monolithic dataset into k groups. Each group acts in turn as $\mathcal{D}_{Test}$, with the remaining $k-1$ groups combining to form $\mathcal{D}_{Train}$ — this results in k training repeats. As such, it helps measure the variation in performance due to changes in training data.

Running Example: Datasets

For the huskies and tigers problems, imagine we have 500 images of huskies and 500 images of tigers, and that we have the correct label for each image. For use with most ML models, the images would need to be the same size (which can be achieved through resizing/cropping). These images form our original monolithic dataset of 1,000 $(\mathbf{x}, \mathbf{y})$ pairs, where each $\mathbf{x}$ is one of the images, and each $\mathbf{y}$ is the binary label of 0 for husky and 1 for tiger. An 80/20 train/test split would mean training on 800 $(\mathbf{x}, \mathbf{y})$ pairs and evaluating on 200. If we did this we stratified sampling, $\mathcal{D}_{Train}$ would contain 400 husky images and 400 tiger images, while $\mathcal{D}_{Test}$ would contain 100 husky images and 100 tiger images.

2.2 Training

Below we discuss the different processes involved in training ML models. We first introduce training for classification problems (Section 2.2.1), and then for regression problems (Section 2.2.2), as both are explored in the research chapters of this thesis. We then discuss evaluation metrics in Section 2.2.3.

2.2.1 Classification

The label space $\mathcal{Y}$ is defined by the problem that is being solved. For classification problems, each piece of data belongs to one of C classes, so $\mathcal{Y} \in \mathbb{Z}_C$. However, the model output space may not align with the label space. The output of a predictive model, $\mathbf{z} \in \mathbb{R}^C = M(\mathbf{x})$, is a vector of unnormalised values known as logits. These logits can be normalised to a set of probabilities using the softmax function $\sigma : \mathbb{R}^C \to (0,1)^C$.

The softmax normalised output has the property that the sum over classes is equal to one, i.e., $\sum_{c=0}^{C-1} \sigma(\mathbf{z})_c = 1$. While these probabilities are more easily comparable than the raw logits (as they are normalised), they are not guaranteed to be a robust measure of

model confidence. However, for both logits and probabilities, the class with the greatest value is considered the model's top prediction for the input $\mathbf{x}$. As such, the predicted class $\hat{\mathbf{y}} \in \mathcal{Y} = \text{argmax}(M(\mathbf{x}))$.

> ***Running Example: Model Outputs***
>
> In our huskies and tigers example, the output of our model M is a vector of length two: $M(\mathbf{x}) \in \mathbb{R}^2$. If the model output (logits) are $M(\mathbf{x}) = [1.5, -2.1]$, then the softmax probabilities are $[0.973, 0.027]$. The predicted class from these outputs, $\hat{\mathbf{y}} = \text{argmax}(M(\mathbf{x})) = 0$, which corresponds to the husky class.

2.2.2 Regression

For regression problems, each piece of data is associated with a single real-valued number, $\mathcal{Y} \in \mathbb{R}$, or d real-valued numbers $\mathcal{Y} \in \mathbb{R}^d$. Regression outputs are not typically normalised, so the model's output is in the same space as the labels. However, in some domains, it can be appropriate to consider the upper and lower bounds of the space, for example strictly positive values.

> ***Running Example: Animal Weight***
>
> Our running example is a binary classification problem. As an example of a regression problem, imagine the input data remains the same (pictures of tigers and huskies), but the data are now labelled with the weight of the animal. The label space is now $\mathcal{Y} \in \mathbb{R}_+$ (positive real values). It is possible to extend this to any number of output dimensions. For example, the labels could be the animal's height and weight, which would make the label space $\mathcal{Y} \in \mathbb{R}^2_+$.

2.2.3 Metrics

Metrics are used for two purposes in ML: for training (optimisation) and as measures of performance. One metric that is typically used for training classification models is cross-entropy loss. As this is a loss, lower values are better. Cross-entropy loss is applied to the unnormalised logits, i.e., the raw model outputs $\mathbf{z} = M(\mathbf{x})$.[1] For a piece of data with label $\mathbf{y}$ (the class index), the cross-entropy loss is:

$$CrossEntropyLoss(\mathbf{z}, \mathbf{y}) = -\log \frac{e^{\mathbf{z}_\mathbf{y}}}{\sum_{c=0}^{C-1} e^{\mathbf{z}_c}}. \tag{2.1}$$

[1] This follows the convention used by PyTorch, which is the ML library used in this work. PyTorch makes a distinction between cross-entropy loss and negative log-likelihood loss: the former uses unnormalised logits, and the latter uses log probabilities. See `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html` and `https://pytorch.org/docs/stable/generated/torch.nn.NLLLoss.html`.

When using the cross-entropy loss, a model is rewarded for producing a high logit value for the correct class $\mathbf{y}$ and low logit values for the other classes. A value close to zero when averaged over an entire dataset suggests the model is performing well. However, it remains difficult to understand exactly what certain loss values mean (other than lower is better).

Other metrics are more readily understandable than cross-entropy loss. Such metrics include accuracy and balanced accuracy — the latter weights the performance by the class distribution, meaning it is a better measure for unbalanced datasets. Other metrics include precision, recall, and the F1 score. The best choice of metric is dependent on several factors such as dataset imbalance and the importance of avoiding false positives.

Running Example: Evaluation

We can evaluate a model trained on our tiger and huskies example using cross-entropy loss and accuracy. Before training, we would expect the model to have a high cross-entropy loss and an accuracy of around 0.5 as it is making predictions at random. After training, if the model has learnt to classify the images correctly, we would expect to see a cross-entropy loss close to zero and accuracy nearing 1. In this case, we are using a balanced dataset (an equal number of tigers and huskies), so balanced accuracy will be the same as accuracy. However, if our dataset contained 90% tigers, the model could achieve a high accuracy of 90% by always predicting tigers (without having to learn anything), but the balanced accuracy would be 50%, showing the model's true performance.

For regression problems, metrics such as accuracy cannot be applied. Instead, alternative measures are to evaluate the difference between true and predicted values. Example metrics include Mean Absolute Error (MAE), Mean Square Error (MSE), and Root Mean Square Error (RMSE). These are typically applied directly to the network outputs.

2.3 Neural Networks

Thus far in this chapter, we have remained agnostic to the model M that is used for performing classification or regression. While many different types of models can be used, here we introduce neural networks as they make up the vast majority of models used in this thesis. As we later explore, neural networks are widely used in Multiple Instance Learning (MIL) and are often State-Of-The-Art (SOTA) MIL methods.

Neural networks are constructed in layers, with information flowing from the input layer, through one or many hidden layers, and finally through an output layer which produces logits (see Section 2.2.1). The networks can be considered a collection of

weights (parameters) that determine how the input is transformed into the output. Initially, these weights are set to random values, but through the course of the learning process, they converge to a set of values that produces the desired output. At the heart of the learning is backpropagation, where the predictive error (e.g., cross-entropy loss Equation 2.1) is used to adjust the network's weights such that future predictions will predict the correct label. With sufficient labelled data and epochs of training (where one epoch is an iteration through the entire training dataset), the weights are sufficiently adjusted such that the correct predictions occur. For details on the history of neural networks, see Schmidhuber (2015).

Many different flavours of neural networks exist. However, the same principles such as backpropagation and optimisation of a loss function underpin the vast majority of supervised learning models. The neural network architectures/modules that have relevance to this thesis are convolutional layers (neural networks for images), recurrent layers (neural networks with a form of internal memory), and attention (increasing/decreasing the importance of certain inputs). These will be discussed in greater detail in the relevant later chapters.

Running Example: Neural Network Architecture

For the tigers and huskies example, a simple neural network architecture could be designed as follows. Firstly, as the input data are images, the first few layers would be convolutional. These function as a Feature Extractor (FE) to reduce the input image to a feature vector. Several fully connected layers could be used to perform further processing and reduce the number of features. Finally, the output layer would be of length two as we are performing binary classification: one output gives the logit for a husky prediction, and the other for a tiger prediction.

2.4 Practicalities of Machine Learning

There are several other factors to consider when designing, training, and evaluating ML models. These are the more practical or engineering elements of ML that ensure it works in practice. Below we discuss four key areas of practicality that reoccur in this thesis. A brief overview is provided in each case; the realisation of these details is provided in later chapters as they are domain-specific.

2.4.1 Data Preparation

The performance of any model is constrained by the data it is trained on. Poor quality (e.g., blurry images), insufficient quantity, and incorrect labels are all detrimental to the training process and overall performance of the model. Fortunately, many high-quality

datasets for a wide variety of domains already exist, but data preparation is still an important part of all ML pipelines.

Insufficient data is a common occurrence in ML, but there are approaches that facilitate dataset expansion without having to gather more data and label it, such as augmenting the data via transformations (e.g., image cropping and rotation). There can also be a lack of data for certain classes, meaning these classes are underrepresented in the data and thus the model may struggle to learn them correctly — this is known as class imbalance. Data augmentation and balancing approaches can be used to overcome this problem.

Data normalisation is another key aspect of data cleaning. Neural networks (see Section 2.3) work best when data is normalised, i.e., with a mean of zero and standard deviation of one. This is also beneficial in ensuring consistency across features that may have different value ranges.

> ***Running Example: Data Normalisation and Augmentation***
>
> A typical data normalisation approach in ML is to subtract the dataset mean and divide by the dataset standard deviation. For the huskies and tigers example, the mean and standard deviation of the pixel values would be calculated and used to normalise each image. The training dataset size could also be expanded through image augmentation techniques such as horizontal flipping and random cropping.

2.4.2 Training Improvements

In addition to improving the quality of data that an ML model is trained on, further performance gains can be achieved by modifying the training procedure. A common change is to only update the model's weight via backpropagation after processing a 'batch' of data. This smoothes out the weight updates performed during training, as the gradients used in backpropagation are averaged over the batch. This also benefits training speed, as fewer weight updates are used and the data within a batch can typically be processed by the model in a single pass. A further training improvement is early stopping, when the model performance on a validation set is measured to avoid overfitting. If the validation set performance starts increasing (typically the training set performance would still be decreasing), the model has begun to overfit, and training should be terminated.

Running Example: Training Configuration

The tigers and huskies problem could be configured with an 80/10/10 dataset split, which would facilitate the use of early stopping. Rather than pick a maximum number of epochs, we train the model until the validation set accuracy begins to decrease (at which point the model is overfitting the training data). A patience value (number of epochs) should also be included to verify that this is a consistent decrease, otherwise training may terminate prematurely. We would also need to decide on a batch size — typically this is determined by the amount of memory available, the model size, and the size of each piece of data. Values such as 16, 32, 64, or 128 are usually appropriate for these types of computer vision problems.

2.4.3 Hyperparameter Selection

The ML training process has many hyperparameters. These are different to model parameters which are learnt during training. Instead, hyperparameters are specified prior to training. Examples include the learning rate (how much the model's parameters are adjusted for each training pass) and the number of epochs (how long the model is trained for). While these parameters can be manually set (or have suggested default values), hyperparameter optimisation methods can be used to find ideal values that lead to the best overall model performance. For example, training for too few epochs means the model performance has not converged (i.e., its overall performance could be improved), but training for too many epochs means the model is likely to overfit to the training dataset and perform worse on the test dataset. Hyperparameter optimisation can also be applied to the other aspects mentioned above, such as finding the ideal batch size and what type of data augmentation/normalisation is most appropriate.

Running Example: Hyperparameters

Once some level of performance has been achieved in our tigers and huskies example, i.e., once it is demonstrated that the model is learning something, hyperparameter optimisation can be used to further improve the model. Tuning values such as the learning rate, batch size, and early stopping patience should lead to better performance. However, these values typically cannot be treated independently, so a hyperparameter optimisation framework is required — less expensive than a complete search over all possible parameter combinations, but likely to find better combinations than tuning parameters individually. Additional elements of the ML solution could also be changed, such as altering the network architecture.

Given this background to supervised ML, in the next chapter, we conduct an extensive literature review on interpretability and MIL.

Chapter 3

Literature Review

This chapter provides a comprehensive literature review of interpretability and Multiple Instance Learning. We begin with interpretability (Section 3.1), covering topics such as motivation, causality, and methods. This is followed by Multiple Instance Learning (Section 3.2), discussing data structure, paradigms, assumptions, and methods. Of particular note is Section 3.2.5, which explores the intersection of these two areas and lays the foundations for the research chapters presented later in this thesis.

3.1 Interpretability

Understanding how Machine Learning (ML) systems make decisions is crucial in facilitating the widespread adoption of Artificial Intelligence (AI). Without an understanding of how automated systems make decisions, especially when those decisions affect the end user, there can be little or no trust in the systems. Explainable AI and interpretable ML are two research areas that aim to address the difficulties in understanding the automated decision-making process and how best to communicate said understanding to different audiences. In the following sections, we first review the difference between interpretability and explainability (Section 3.1.1), and then provide the key motivators for interpretability (Section 3.1.2). Next, we discuss interpretability from a social sciences perspective (Section 3.1.3) and a causal perspective (Section 3.1.4). Finally, we explore existing interpretability methods (Section 3.1.5).

3.1.1 Interpretability vs Explainability

The concept of explainability is not new. Philosophers have reasoned about how humans explain things, and ideas from the social sciences can be used as a guide for explainability in AI — both for working out how to get automated systems to explain things, but also understanding how humans perceive their artificial counterparts, allowing human-machine communication to be more effective.

From a social sciences perspective, (Miller, 2019) suggests there are two processes in an explanation: a *cognitive process* and a *social process*. The cognitive process identifies the causes of an event and selects an appropriate subset to form some sort of explanation. Using the product of the cognitive process as the basis for communication, the social process involves the transfer of knowledge between the explainer and the explainee, using some form that is most appropriate for the target audience and the conditions under which the explanation must be provided (e.g., time constraints). An advantage of this framework is that it separates the method of explanation from how it is communicated; different audiences (e.g., domain experts vs children) require different explanations, and the explanations provided may also change based on context. However, the underlying causes and reasoning that were used to make a decision remain the same, so all that needs to change is the communication method, not the underlying causes. This definition is also flexible enough to apply to both humans explaining their actions and AI systems explaining theirs.

Interpretability can be considered the first step of this explanation process (the cognitive process). The outputs of interpretability methods are often technical and likely removed from human reasoning, i.e., depending on the context, they may only be useful to domain experts or AI practitioners. While they do represent a system's decision-making,

the interpretations only form part of the process of explaining a system's actions (Rosenfeld and Richardson, 2019). The social process is then required to transform these technical interpretations into something more digestible and appropriate for a particular audience, i.e., interpretability + social process = explainability.

In this thesis, we focus on the interpretability rather than explainability. While some effort is made to communicate the resulting interpretations (e.g., through visualisation), thorough consideration and rigorous evaluation of the efficacy of said outputs is not undertaken. This would require further research and likely involve evaluation with human participants, see Section 8.2. In the next section, we outline why it is important to develop interpretations of AI systems.

Running Example: Interpretability vs Explainability

In Figure 3.1, we provide a demonstrative example of the difference between interpretability and explainability using the tigers vs wolves example. It begins with the data and model prediction (in this case a correct prediction). The interpretability method (cognitive process) then produces a saliency mask which highlights the regions that were used in the model's decision-making process. The social process then provides a textual explanation alongside this interpretation for two different audiences (technical and non-technical).

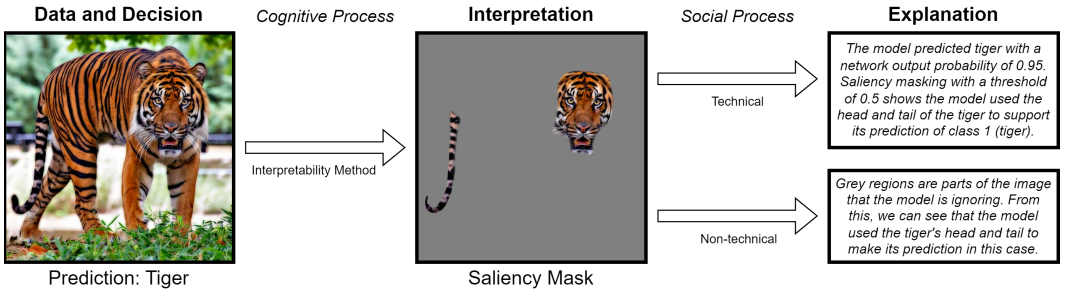

FIGURE 3.1: An example of the difference between interpretability and explainability. The interpretability method in this example is a mock saliency mask, which can be imagined as a pixel or super-pixel output that indicates the importance of different image regions for the model's decision on a scale of between 0 (not important) and 1 (important). Tiger image sourced from https://www.pexels.com/ under an open license.

3.1.2 Motivation

At its core, the main motivation for interpretability is understanding how and why an automated system makes decisions, especially those that have a large impact on the real world. As the system learns to make its own decisions, the reasoning behind its actions is not necessarily explicit, i.e., the system's actions cannot be inherently understood without additional information. This is especially true in complicated domains, where it can be hard to understand why a certain decision was made given a complex input. Some further key drivers of explainability research have previously been identified

(Anjomshoae et al., 2019; Fox et al., 2017). We outline several key motivations below, with a focus on facilitating better relationships between automated systems and humans, as well as improving the efficacy of the systems themselves:

Transparency Some simple ML models can be considered transparent or inherently interpretable, i.e., it is possible to understand how they make decisions without requiring any extra information or processing (Guidotti et al., 2018). For example, linear models and small decision trees are inherently transparent, assuming the user has a good understanding of the input data. Transparency is lost when the input data doesn't represent coherent, understandable features, for example, if it has been pre-processed in such a way that the original features are lost (e.g., certain types of dimensionality reduction). It is also possible to argue that these models do not guarantee transparency unless they are sufficiently small (Lipton, 2018). Rosenfeld and Richardson (2019) state a model is transparent only if it is inherently understandable without the need for another tool. Such transparency is also audience- and context-dependent, i.e. it may be transparent for some end users but not others.

Trust The systematic literature review by Anjomshoae et al. (2019) found that transparency and trust were two of the most prominent drivers in explainability research. They are closely related, as both are designed to increase the user's confidence that a system is working as intended by understanding the reasoning behind its decisions. The importance of fostering trust is that it allows human operators to reduce their workload by giving AI systems more autonomy. This is especially crucial in scenarios where the automated systems are better suited than humans for the task at hand (Wang et al., 2016).

Collaboration When humans and automated systems collaborate, there is some common goal that they are working towards. However, each member of the team may have their own individual sub-goals which must be reconciled in order to achieve efficient performance towards the overall goal. One key element of collaboration is communication — it allows each team member to understand the other's sub-goals and is the medium through which the other motivators for explainability can be achieved. However, communication is not as simple as sharing all the information a system has and assuming the end user can disentangle and make sense of it — this could overload the human if there is a large amount of complex data (Li et al., 2016). To achieve strong collaboration, the communication between automated systems and humans must be focused and appropriate given various constraints such as time and cost of communication. By providing coherent explanations about reasoning, explainability decreases the burden on the end user to interpret the actions of the automated system, which makes collaboration easier.

Running Example: The Need for Interpretability

Figure 3.2 presents a motivating example of why interpretability is required in ML systems. A method may have a high level of predictive performance but have learnt a spurious correlation between the input images and their labels; in this example using the background to identify the class rather than using the animal in the foreground. The model would then fail on valid images where this spurious correlation does not hold; in this example when the animals are presented in different backgrounds to the training data. Without interpretability, it is difficult to verify and therefore trust that the model is not using a spurious correlation such as the background colour.

FIGURE 3.2: A motivating example for interpretability in the tigers and huskies example. If the images in the training data are all similar to the images shown in the first three columns, then it is valid (and likely) for an ML approach to simply learn the colour of the background: white indicates husky, and green/brown indicates tiger. However, if the test data contains images similar to those in the final column, the model will make incorrect predictions as the backgrounds are flipped. Tiger and husky images sourced from https://www.pexels.com/ under an open license.

Intent Communication As an extension of communication to facilitate collaboration, another motivation for explainability is intent communication (Fukuchi et al., 2017; Hayes and Scassellati, 2013). When working in teams, people construct mental models of how other members of their team will act in certain situations, i.e., they build up a 'picture' of how they expect their teammates to act. This is true even for robotic or virtual team members. An artificial system that enables human co-workers to understand its actions through explanations leads to a more accurate mental model, facilitating better decision-making and avoiding harmful misunderstandings (Hayes and Scassellati, 2013). The use of explanations goes beyond providing an understanding of a recent decision — it improves the ability of human collaborators to predict the actions of an automated system in the future, allowing better collaboration and planning (Fukuchi et al., 2017).

Control Despite acting autonomously, ML systems must still provide some degree of control to human users, whether it be completing tasks assigned to them or allowing a user to override the system — they are never acting completely independently

(Holliday et al., 2013). Some level of control is retained by human users in order to ensure the goals of the system are aligned with the human's. Adjustable autonomy takes strides towards providing different levels of user control, but explainability allows it to go a step further. If an automated system makes a mistake, the user may feel they have a lack of control as they do not understand why the system got something wrong. However, if the system explains its reasoning, the user can better understand why the mistake was made, which restores a sense of control.

Debugging A final theme is using explanations as a development tool on top of traditional debugging. In ML development, debugging is more advanced than simply removing errors in the software — the software could be error free but the system still may not function as intended due to the nature of what it has learnt and how it makes decisions. Interpretations allow for insight that goes beyond the software implementation itself, i.e., looking at the system's reasoning rather than the mechanism (Hindriks, 2012). This could lead to adjustments in training or exposure to new data that improves the system's functionality. Explanations also make it possible to test systems without having to run them in all possible scenarios, something that is often impossible in practice (e.g., self-driving cars could find themselves in an infinite number of situations; there is no way to test them all).

Beyond the key themes outlined above, there are some additional motivations for interpretability and explainability. In the EU's GDPR legislation, the end users and those affected by autonomous systems have a "right to explanation", indicating explainability is also motivated by legal requirements (Voigt and von dem Bussche, 2017). However, under the GDPR legislation, this is a non-binding requirement, and a simple overview of the system is likely to be sufficient as far as legality is concerned (Wachter et al., 2017). The more recent EU AI Act includes notions of transparency and human oversight, implying a need for explainability (Panigutti et al., 2023). Regulation of AI is still an ongoing process, and additional legislation further down the line will likely include more specific requirements for interpretability/explainability.

The use of autonomous systems in industry presents additional motivations beyond research-based objectives. In consumer markets where automated systems are used (e.g., virtual assistants and advertising), interpretability can lead to an increased likelihood of use and therefore better sales (Haynes et al., 2009). The competitive edge granted by explainable systems also helps industry conform to fair and ethical standards by proving that the reasons behind the decision-making process do not discriminate (Lipton, 2018).

The motivation for explainability and interpretability given above have an AI heavy focus. However, additional motivation and information about the requirements of interpretability can be gained by utilising knowledge from the social sciences on how

humans communicate and understand each other. In the next section, we delve into explainability and interpretability from a social sciences perspective.

3.1.3 A Social Sciences Perspective

Humans tend to observe others and attribute mental states to them in order to understand their reasoning and better predict their future actions. The internal mentalisation of other's thoughts is known as Theory Of Mind (TOM) or *folk psychology* (Goldman, 2012). TOM represents a formalisation of mental states, including emotions and internal attitudes such as beliefs, desires, hopes, and intentions. This extends to automated systems (both virtual and physical); humans are known to anthropomorphise non-human objects and attribute human-based mental models to them (Lee et al., 2005). This is especially prominent when the system possesses some human-like traits, for example as a robot or a virtual interface depicting a person (Parise et al., 1999).

There is a strong suggestion that effective communication to facilitate good team performance is dependent on the ability of individuals to utilise TOM. This correlation has been demonstrated in human teams — a greater collective intelligence (a measure of group performance) is linked to higher utilisation of TOM skills (Woolley et al., 2010). Further work has shown this occurs both in face-to-face situations as well as in online (i.e., text only) interactions (Engel et al., 2014).

Even with people's innate ability to apply TOM models to non-human objects, there is still a disparity in applying TOM to automated systems. Despite impressive increases in the performance of ML systems on complex tasks, research has shown that humans and automated systems perform the same tasks differently, for example, by looking at different regions of an image during visual question answering (Das et al., 2017). These discrepancies can lead to failed interactions due to ill-assigned TOM models (Chandrasekaran et al., 2017). If the human users assume the systems are acting with similar reasoning and beliefs as themselves (or humans in general), then they might under or overestimate the ability to which the system can perform certain tasks, leading to communication breakdowns and a reduction in performance. This is especially prominent with the rise of large language models, although there is some evidence that recent large language models demonstrate some initial levels of TOM (Kosinski, 2023).

The responsibility of effective communication that utilises TOM lies with all parties involved. The communication between automated systems and humans should facilitate better TOM models on behalf of humans, allowing human users to more accurately understand their artificial teammates. Chandrasekaran et al. (2017) posit this to mean that automated systems have to utilise TOM models of their human team members in order to communicate most effectively (i.e., in a form of communication

which the humans best understand) and allow the humans to build better TOM models for the automated systems. Their work goes on to show that some off-the-shelf interpretability techniques do not help improve a user's TOM model of an ML system, suggesting explainability techniques that communicate in human terms (i.e., concepts and ideas that people are familiar with) and TOM based analogies may be more effective. This agrees with our discussion above about the difference between interpretability and explainability — the social process is necessary to transform technical interpretations into human terms to provide complete explanations that facilitate TOM models.

There are two possible ways to rectify the disparity between the perceived TOM of automated systems and their true reasoning. The first is to make automated reasoning more 'human', i.e., decision-making that utilises similar lines of reasoning to that of a human. Whilst the human decision-making process is effective in tackling a wide variety of problems at a high level of performance, distilling this into ML systems in a way that achieves a high level of performance and is generalisable is a difficult task. There is also the concern that imitating human decision-making may actually limit the abilities of an automated system — by restricting it to the reasoning that we a familiar with, what the system learns may be constrained such that new strategies or relationships in the data are not discovered. The second solution is to ensure that the system explains its actions in such a way that helps users build a sound and accurate TOM model of the system, regardless of its internal implementation. Just as we do not fully understand how the brain functions but can describe our intentions through TOM, it may be sufficient for an automated system to just explain its intentions in a TOM manner without actually utilising human-style reasoning. This emphasises the role of explainability, not on the underlying learning process, making the goals of explainability independent of the implementation of the automated system underneath. This leads to potential long-term benefits as new ML techniques are developed in the future — one only needs to adapt existing explainability methodologies to the underlying techniques without having to completely redevelop the explainability and TOM model.

Both solutions to producing better explanations are dependent on some form of reasoning that facilitates TOM models. For this, we can turn to causality, which we discuss in the next section.

3.1.4 Causality

The real world is inherently based on cause-and-effect relationships; actions in an environment have direct consequences on the environment itself. Even from an early age, humans learn this relationship — knock a glass off a table, and it will fall to the floor and likely smash into many pieces. A lot of human reasoning is based on cause-and-effect interactions, and thus our explanations involve cause-and-effect

relationships. One possible avenue for explaining the actions of an automated system utilises causal relationships — this satisfies the use of TOM and provides effective communication in human-machine teams.

The familiar statement "correlation does not imply causation" has weight when considering how many modern ML systems make decisions. In a supervised learning scenario, a system is typically just learning a correlation between elements of the input data and the output label. In our running example, identifying the correlation between certain groups of pixels and the label tiger is effective at achieving a high level of performance. However, causality can also be explicitly applied in ML. Despite causal relationships being a staple of the world in which we inhabit, there was much resistance to the mere mention of causality in statistics throughout the 20th century. However, modern research is much more open to the inclusion of causality, and advancements in causal inference have begun to pervade the AI and ML world. Pearl and Mackenzie (2018) define a 'ladder of causation' with three different stages, where each 'rung' on the ladder utilises a higher level of causality than the previous stage. At the first level, where Pearl considers modern AI systems to reside, the decision-making process only utilises correlations and associations, i.e., what does x tell me about y? Associative relationships give no guarantee of causality (i.e., does A cause B or vice versa) and there could be unobserved causes responsible for both (Lipton, 2018). From an explainability perspective, correlations offer an initial insight into a system's decision-making, but they fail to capture the true causal relationships and do not enable humans to create a causal TOM mode.

Rung two of the ladder progresses to the inclusion of intervention — if I *do* x, what is the effect on y? This goes beyond an associative relationship — the outcome y is explicitly determined by the cause x, rather than being a correlation that may or may not represent a casual relationship. Without a causal model, it is impossible to perform any reasoning above rung one of the ladder, i.e., above the level of correlation and association. The final rung of the ladder takes the use of interventions a step further and considers counterfactual scenarios — if instead of doing x, I had done x', what would have happened to y? The distinction from rung two of the ladder is that x has already happened, and the effect on y has been observed. The question now revolves around what the effect on y would have been had x been different. This progresses to the inclusion of imaginary worlds, and by utilising a causal model, it becomes possible to unravel the degree to which x affected y (and how x' would have affected y). A depiction of this conceptualisation of causality is given in Figure 3.3.

A further distinction between counterfactual prediction and associative prediction (i.e., prediction that utilises causality rather than just correlation) is that counterfactual reasoning can incorporate expert knowledge (Hernán et al., 2019). In current ML systems, the use of domain expertise typically begins and ends with the curation of the relevant data — once this has been done, the ML system is left to process the inputs and

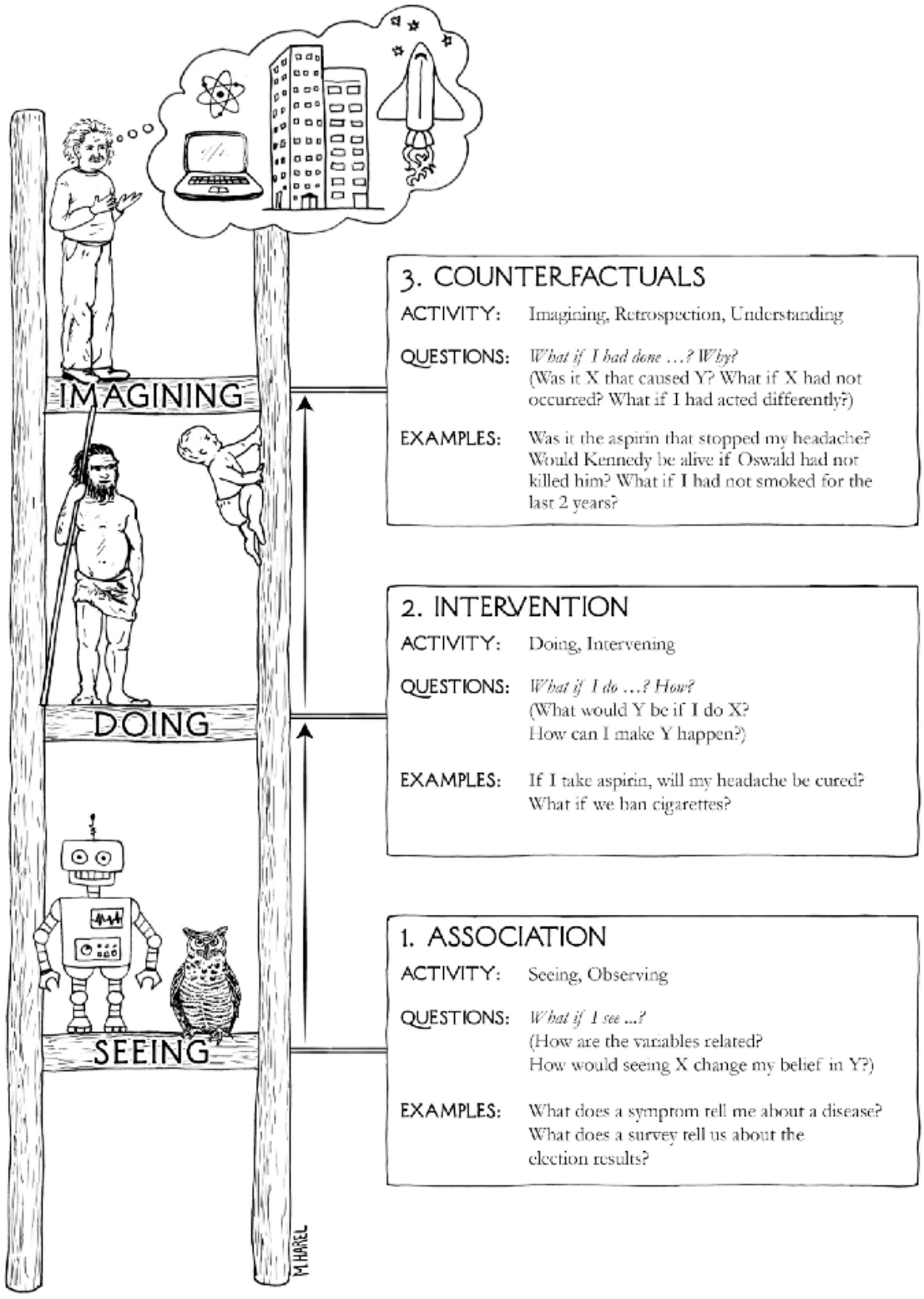

FIGURE 3.3: The ladder of causation; figure copied from (Pearl and Mackenzie, 2018). Each rung adds a level of causal understanding, with many AI systems residing on the bottom rung (i.e., purely associative relationships).

outputs and make its own decisions. In counterfactual reasoning, the domain expert has a greater role in the learning process by providing the causal model and adjusting their

relevant causal assumptions if the results of making inferences about the data do not reflect the causal model. Whilst this line of reasoning can seem fairly abstract, it is fundamental to how humans reason about situations. Providing answers to these sorts of questions is currently beyond the capabilities of AI systems that operate at an associative level, leading to a disparity between the way humans reason about the world and the decision-making process of AI systems. Without addressing these differences, automated systems are severely handicapped in their attempts to communicate their reasoning with humans.

Fortunately, work on causal inference research has led to concrete mathematical models for reasoning about cause-and-effect relationships. Beginning with the development of causal diagrams (Pearl, 1995), and progressing to structural causal models, causal inference provides a coherent, accessible way for defining causal relationships, and can be utilised to perform causal reasoning based on observed, associative data (Halpern and Pearl, 2005). The use of structural causal models allows for the progression from rung one of the ladder (correlations and associative relationships) to rungs two and three of the ladder by defining assumptions regarding causal relationships, providing a solution for transforming system explanations into human-orientated, TOM based reasoning.

As associations cannot identify the true underlying mechanisms that describe the changes that can occur, a causal model is required in order to guarantee robustness (Pearl, 2019). From an explainability perspective, causal explanations are more likely to represent a realistic relationship rather than a spurious correlation. The sorts of queries we want to be able to ask of AI systems, and the answers with which we expect them to respond, go beyond associative reasoning, therefore requiring climbing to rung two or rung three of the ladder of causality, meaning causal relationships must be utilised in explainability. This is reinforced by the consideration of how humans reason and explain using counterfactuals — this is well studied and can be applied to guide how explainability should utilise counterfactual reasoning for explanations (Byrne, 2019).

The role of causation in explainability is an ongoing area of research (Baron, 2023; Beckers, 2022). However, in order to generate casual explanations, is it necessary for systems to utilise causal relationships in their decision-making process? An automated system must either be based on a causal model, or its explanations must be mapped to one (Holzinger et al., 2019). The use of causal models may increase system performance, in which case the premise of explainability becomes easier as the explanations can be directly derived from the system's internal causal model.

With these considerations of TOM and causality, we now specifically focus on interpreting the decision-making of ML systems.

3.1.5 Interpretability Methods

Before we can discuss Multiple Instance Learning (MIL)-specific methods, it is necessary to understand the broad elements of interpretability. Three aspects are discussed below.

3.1.5.1 Global vs Local

Interpretability can be split into *global* and *local* scopes (Molnar, 2020). Global interpretability focuses on how the model makes predictions for any input, whereas local interpretability focuses on one particular input.

In the global case, interpretations give an overview of general trends that hold across an entire dataset. For example, in a medical setting, it could be that a particular patient feature increases the risk of developing some disease across all patients. Such global explanations give a general overview of how an automated system makes decisions, and are useful in ensuring it acts correctly across all possible inputs.

For local outputs, the interpretations are specific to a particular input. Again using the medical case, it could be that the value of a particular feature with respect to the other features increases risk, despite this feature not having a global trend across all patients. This facilitates understanding of decision-making for individual cases and highlights casual relationships learnt by the model which may not be apparent from a global perspective.

3.1.5.2 Inherent vs Post-hoc

In Section 3.1.2, we discussed the idea of model transparency (the concept that some models are inherently interpretable). This means they provide some type of interpretation (e.g., feature importance) alongside their output. This could be either global (e.g., the weights in a linear regression model) or local (e.g., a saliency map over an image). While certain models can provide these interpretations, not all models do so. As discussed previously, inherent interpretability or transparency is typically associated with simpler models.

To understand the decision-making of models that do not provide their interpretations (i.e., are not transparent), post-hoc approaches can be used. These are separate methods (distinct from the decision-making models) that query a model to understand how it makes decisions. This could be a global interpretation, e.g., understanding the representations learnt by convolutional or recurrent models (Du et al., 2019). It could also be a local interpretation, e.g., perturbing the inputs of a specific input to see how a model's decision-making changes and thus revealing the extent to which elements of the inputs affect the prediction. Ribeiro et al. (2016a) proposed Local Interpretable Model-agnostic Explanations (LIME) and Lundberg and Lee (2017) proposed Shapley Additive Explanations (SHAP) — both are examples of popular post-hoc local interpretation methods.

Running Example: Global vs Local Interpretations

Returning to the example edge case images previously shown, in Figure 3.4 we present some mock local interpretations. If we were presented with interpretations in the left example (background important), we could infer that the model has not learnt the correct relationship — it is looking at the background colour (a spurious correlation) rather than the animal itself. However, if we were presented with interpretations similar to those on the right, we could infer that the model is actually using the correct information (the animal) and is ignoring the background, so we could be more confident and trusting in its decisions as it is using a causal relationship. Furthermore, this would also suggest that the model will be robust to other background types. In contrast to these local interpretations, an example global interpretation would be: *"the model identifies huskies by the fact that they are mostly observed in snowy climates (white backgrounds), and identifies tigers by the fact that they are mostly observed in jungle climates (green/brown backgrounds)"*.

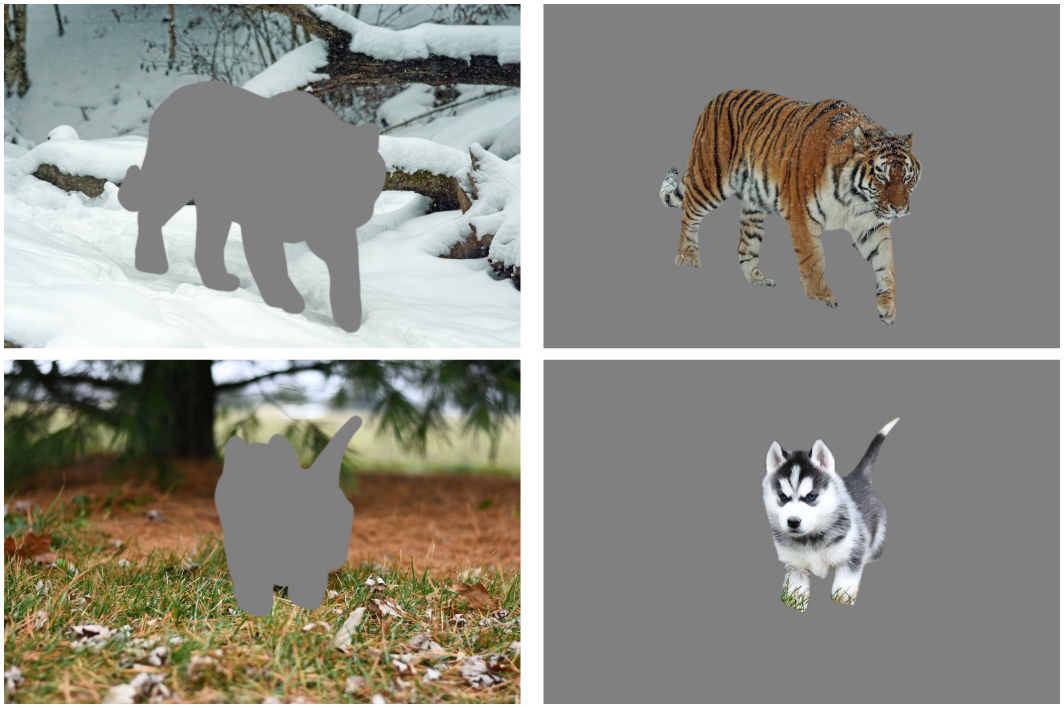

FIGURE 3.4: interpretations allow us to understand model decision-making. In these mock examples, unimportant areas are masked in grey while important areas are unchanged. In the left examples, the animal itself is masked, implying the model is only focusing on the background. In the right examples, the background is masked, implying the model is (correctly) focusing on the animal in the foreground, which is a true causal relationship. Tiger and husky images sourced from https://www.pexels.com/ under an open license.

3.1.5.3 Model-Specific vs Model-Agnostic

A final distinction that is important to discuss with respect to the research presented in this thesis is the concept of model-specific vs model-agnostic approaches. Some interpretability methods can only be applied to certain types of models, making them

model-specific. An example of this is the Class Activation Mapping (CAM) family of methods (originally proposed by Zhou et al., 2016), which examine the activation of weights in neural networks for particular inputs (a post-hoc local method). As this approach analyses the internal workings of neural networks, it is only applicable to that family of models, i.e., it is not applicable to non-neural network approaches.

Alternative approaches, for example, LIME (Ribeiro et al., 2016a) and SHAP (Lundberg and Lee, 2017) as mentioned above, are model-agnostic as they are not tied to one particular family of models. These approaches typically make no assumptions about the model that is being interpreted. Rather they only observe how its outputs change with respect to different inputs. In this sense, the interpretation is only dependent on the input data and the model's outputs — the interpretability method does not care what the underlying model is or how it is implemented.

Given this overview of interpretability, in the next section, we explore MIL in more detail, including the core concepts of interpretability for MIL.

3.2 Multiple Instance Learning

Building on the background to supervised learning given in Chapter 2, we now provide further details on MIL. Many of the concepts previously introduced are relevant in MIL; the significant difference is in how the data is structured. We first cover this in Section 3.2.1, and then explore the different learning paradigms within MIL in Section 3.2.2. We next cover MIL assumptions in Section 3.2.3, which determine the relationship between MIL instances and bags. Specific MIL models are next explored (Section 3.2.4), and then we link the previous discussions of interpretability with respect to MIL in Section 3.2.5. Finally, we discuss MIL as a whole and outline the research chapters in the remainder of this thesis (Section 3.3).

3.2.1 Bags of Instances

For standard supervised learning, each piece of data $\mathbf{x} \in \mathcal{X}$ is considered a complete unit of independent information. In MIL, data is structured differently. Rather than being organised as individual items, data is structured into groups, known in MIL terms as a *bag*. Each piece of data within a bag is then termed an *instance*. Formally, a bag is denoted $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_k\}$, where each $\mathbf{x}_i \in \mathbf{X}$ is an instance and the bag size is k (the number of instances). Following standard MIL notation, bag-level data is denoted with uppercase variables and instance-level data is denoted with lowercase.

As MIL only requires labels at the bag level, it reduces the burden of labelling (Carbonneau et al., 2018). This makes MIL an attractive solution in domains where

manual annotation of instance-level data is expensive, for example when using textual (Angelidis and Lapata, 2018; Bunescu and Mooney, 2007), medical (Hashimoto et al., 2020; Javed et al., 2022; Li et al., 2021a; Li and Zhang, 2020; Marini et al., 2021; Quellec et al., 2017; Sudharshan et al., 2019; Xu et al., 2014), or audio data (Kong et al., 2019; Wang et al., 2019b). MIL has also been applied to problems such as image annotation (Wu et al., 2015), object tracking (Babenko et al., 2011), video event detection (Phan et al., 2015), and Time Series Classification (TSC) (Zhu et al., 2021).

In the most general sense, each instance can be considered a feature vector of some fixed length d, and the space of possible instances can be defined as $\mathbf{x}_i \in \mathbb{R}^d$. As such, a general MIL bag can be represented as a matrix $\mathbf{X} \in \mathbb{R}^{k \times d}$. While a 1D feature vector is appropriate for some use cases, instances can also be represented in other ways, typically dependent on the application domain or data modality:

- **1D feature vector:** An instance can be an encoding or representation of a piece of data, for example, a vector encoding of a word.
- **Images:** For computer vision applications, the instances represent images. These could be patches of a larger image, different representations of the same object (e.g., a car photographed from multiple angles), or stills taken from a video.
- **A single scalar value:** In some cases, an instance could simply represent a single value. For example, a univariate time series could be considered a MIL bag, where the value at each timestep is its own MIL instance.

For the most part, within a particular dataset, instances come from the same space (e.g., the same sized image patches or same length feature vectors) and are all of the same modality (i.e., bags are not comprised of both image patches and feature vectors). However, the size of bags k (the number of instances per bag), can vary across a dataset. For example, sentences can be represented as MIL bags, where each word (or some representation of it) is an instance. Sentences do not have a fixed length, and as such the bags will have different sizes.

MIL is a *weakly supervised* learning paradigm. Under weak supervision, labelling occurs at a higher level of abstraction or is of lower quality than fully supervised learning. Specifically, MIL is an *inexact* supervision paradigm, where only coarse-grained label information is provided, which alleviates expensive labelling costs. This is in contrast to other forms of weak supervision, such as *incomplete* supervision, where only some of the data is labelled (the remainder is completely unlabelled) and *inaccurate* supervision, where the labels are not guaranteed to be correct. See Zhou (2017) for more details on weakly supervised learning.

Recall from Section 2.1 that in standard supervised learning, each piece of data $\mathbf{x} \in \mathcal{X}$ is associated with a single label $\mathbf{y} \in \mathcal{Y}$. In MIL, only the bags are labelled. As such,

models can only learn from the bag-level labels. Formally, each bag $\mathbf{X}$ has an associated label $\mathbf{Y} \in \mathcal{Y}_B$, where $\mathcal{Y}_B$ is the space of bag-level labels.

While only bag-level labels are used during training, there is also the concept of instance-level labels, where an instance $\mathbf{x}_i \in \mathbf{X}$ has an associated label $\mathbf{y}_i \in \mathcal{Y}_I$. Here, $\mathcal{Y}_I$ is the space of instance labels, which could be the same or different to the space of bag-level labels $\mathcal{Y}_B$. The vast majority of MIL datasets do not provide instance labels (Carbonneau et al., 2018), and for those that do, the instance labels remain unused during training. However, they form an important part of evaluating MIL interpretability (which is discussed later in this chapter).

The objective of MIL is to learn some function $F(\mathbf{X})$ that takes a bag $\mathbf{X}$ as input and maps to a bag-level prediction $\hat{\mathbf{Y}} \in \mathcal{Y}_B$. This is sometimes facilitated through an instance-level function $f(\mathbf{x}_i)$ that operates on individual instances as opposed to the bag as a whole. The output of $f(\mathbf{x}_i)$ is an instance-level prediction $\hat{\mathbf{y}}_i \in \mathcal{Y}_I$. Again, we use notational case to differentiate between bag-level (F) and instance-level functions (f).

3.2.2 Paradigms

There are three main paradigms into which methods for solving MIL problems can be sorted:

1. *Instance-space methods*: A model learns the instance function f, allowing it to make predictions for each of the instances in a bag. These predictions are then aggregated to give an overall bag label. While the bag label could be used as a proxy for the (unknown) instance labels, this introduces error and uncertainty into the learning process as the bag label is not guaranteed to match the label of each individual instance. Instead, a better approach is to train the model end to end, using the predicted bag label generated through aggregation (Wang et al., 2018).

2. *Bag-space methods*: Rather than considering the individual instances within a bag, bag space methods instead consider the whole bag as a non-vectorial entities (due to the potential for different bag sizes). A bag can be represented as a set of points in some space, so any distance function that can compares two sets of points in this space can used. Using this distance function, classifiers such as k-nearest neighbours or a kernel-based classifier such as a support vector machine can be used to classify the bags (Amores, 2013).

3. *Embedding-space methods*: As bags can have different sizes, the distance functions for bag space methods operate on non-vectorial entities. Embedding-space methods take an alternative approach by creating a fixed-size feature vector that represents a bag, which can then be classified to give a final bag label (Ilse et al.,

2018; Wang et al., 2018; Zhou et al., 2009). Typically, each instance in a bag is transformed into a low-dimensional embedding using an embedding function, and the set of embeddings is then pooled to give an overall bag representation.

For both bag space and embedding-space methods, the instance prediction function f is not learnt. Instead, the function F is learnt explicitly rather than inferring the bag label from the instance predictions.

Instance-space methods and embedding-space methods are the two most popular and high-performing paradigms (Carbonneau et al., 2018; Wang et al., 2018). A commonality between the two methods is MIL pooling (sometimes called MIL aggregation). For instance-space MIL, the pooling function takes instance predictions and produces a bag prediction. In contrast, embedding-space pooling creates a single bag embedding by aggregating the instance embeddings, and then classifies the bag embedding to give the final bag label. This is what allows embedding-space methods to sometimes be more effective than instance-space methods — the bag embedding captures global bag level information that determines the bag label (Wang et al., 2018; Zhou et al., 2009). MIL pooling is typically implemented as a mean or max function, i.e., the pooling method is a fixed, non-learnable function.

3.2.3 Assumptions

The label of a bag, $\mathbf{Y}$, is determined by the labels $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_k\}$ of the instances contained in the bag. The particular mapping of instance labels to bag labels is known as the MIL assumption, of which several have been proposed for different use cases. The Standard MIL (SMIL) assumption, which is also the original definition of MIL (Dietterich et al., 1997; Maron and Lozano-Pérez, 1997), is a binary problem with positive (class 1) and negative (class 0) bags where the instance-label space is the same as the bag-label space, i.e., $\mathcal{Y}_B \in \mathbb{Z}_2$ and $\mathcal{Y}_I \in \mathbb{Z}_2$. A bag is positive if any of its instances are positive, otherwise, it is negative, i.e.,

$$\mathbf{Y} = \begin{cases} 1, & \text{if any } \mathbf{y}_i = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

This assumption correctly captures the nature of some real-world problems. For example, in histopathology, one might want to classify a tissue sample (the bag) as cancerous or non-cancerous. If any region (an instance) in the sample is cancerous, then the overall label for the sample is cancerous. Only if there are no cancerous regions is the sample labelled as non-cancerous.

However, many problems do not conform to the SMIL assumption. Under SMIL, there is only a single concept to be learnt that distinguishes positive instances from negative instances. It is possible to think of scenarios where classes can only be separated by examining multiple concepts. Foulds and Frank (2010) give the example of classifying images of *oceans*, *beaches*, and *deserts*. If using a one-versus-all classifier for *beaches*, then there is no single concept that is unique to beaches (water appears in *ocean* and sand appears in *desert*). Instead, *beach* is defined by the co-occurrence of water and sand. This has led to a generalisation of SMIL to fit different assumptions. For example, Weidmann et al. (2003) define a hierarchy of assumptions that define the relationship between instance labels and bag labels.

The purpose of defining additional assumptions is to allow learning algorithms to exploit the relationship between instance labels and bag labels, i.e., they can exploit the prior knowledge of the problem domain (Foulds and Frank, 2010). For example, under the SMIL assumption, any single positive instance is sufficient to give a positive bag label, therefore max pooling with an instance-space classifier is able to give correct classifications. Conversely, if using a count-based assumption (Weidmann et al., 2003), two or more positive instances are required, therefore max pooling is no longer appropriate.

SMIL also assumes there is no interaction between the instances, which is not necessarily true for all MIL problems. In fact, learning the relationships between instances is what makes MIL more powerful than standard supervised learning, and indeed, previous work has shown modelling the relationships between instances is beneficial for performance in bag-level prediction (Tu et al., 2019; Zhou et al., 2009). As an example, consider classifying an image (the bag) by splitting it into patches (the instances). Patches that occur next to each other are more likely to come from the same class than patches that are far apart, therefore incorporating the spatial relationships between instances can be used to improve performance. For example, with patches extracted from satellite images, patches containing houses would be expected to be near patches containing roads.

For certain types of MIL models, understanding the MIL assumption is essential for setting up the model in the correct way. However, as we explore in the next section, neural networks can be used as general-purpose approaches that can be applied to any MIL assumption. In the research presented in this thesis, we use neural networks for general MIL classification and regression problems. Nevertheless, understanding the mapping from instance to bag labels remains important, particularly when it comes to interpretability.

Running Example: Tigers and Huskies as MIL

An example of a MIL problem based around tigers and huskies could appear as follows. Each bag is a collection of images of random animals. For an SMIL problem, we could consider tigers to be the positive class, and all other animals (including huskies) to be the negative class. Therefore, any bag containing an image of a tiger is positive, and all other bags (containing no tigers) are negative. In more a complex case, we could define the assumption that positive bags must contain an image of a tiger *and* an image of a husky. In such a case, a bag containing *only* an image of a tiger or an image of a husky is not positive, so the model needs to learn that both have to co-occur for a positive prediction.

3.2.4 Multiple Instance Learning Methods

Many MIL methods have been developed for different domains and MIL assumptions. This section is not intended to give an exhaustive list of all techniques, but rather highlight some of the more recent methods and give an insight into the State-Of-The-Art (SOTA). For a review of older MIL techniques, please see Amores (2013). In the remainder of this section, we review MIL neural networks, graph-based methods, and attention methods, as these are the current SOTA methods.

3.2.4.1 Multiple instance neural networks

In conventional supervised learning, deep neural networks are a widely used and successful method (see Section 2.3). They can be adapted for use in MIL as general learners for any MIL assumption. Two early implementations by Chen and Narendra (2001) and Zhou and Zhang (2002) both utilised the SMIL assumption and operated in the instance-space. A more recent study by Wang et al. (2018) proposed MIL neural networks that are able to exploit modern advances in Deep Learning (DL) such as residual connections (He et al., 2016) and deep supervision (Lee et al., 2015). Two different architectures were proposed: mi-Net and MI-Net. mi-Net operates in the instance space as it makes a prediction for every instance and then aggregates those predictions to an overall bag-level prediction. MI-Net operates in the embedding space: it aggregates instance representation into a single bag representation (typically with mean or max pooling) and then classifies the bag representation. As such, it does not make instance predictions. To aid clarity, in the remainder of this work, we refer to mi-Net and MI-Net as `Instance` and `Embedding` respectively.

Wang et al. (2018) found `Embedding` outperformed `Instance`, highlighting that learning global bag-level information is beneficial for performance (as opposed to just learning to make instance predictions). Although the initial experiments of Wang et al. (2018)

used the SMIL assumption, later works (as well as the research given later in this thesis) showed that neural networks can easily be generalised to multi-class problems (Ilse et al., 2018). Note, in this neural network setup, `Instance` does not use the bag label as a proxy for the instance labels. Instead, the entire network is trained end-to-end, which is possible as the network's final out is a bag-level prediction. The instance-level predictions can be seen as a byproduct of the network architecture, but, as we discuss in Section 3.2.5, they are a form of inherent interpretability.

3.2.4.2 Graph-based methods

Considering the relations between instances is important for some types of MIL problems. Zhou et al. (2009) proposed a bag space method that explicitly captures the relationships between instances. They represented bags as graphs and designed a graph kernel to capture the similarity among graphs, and then used a support vector machine to classify the bags. Using the graph-based representation meant better information about the bag as a whole was captured — information that is not apparent when only looking at the instances without the context of the bag. This work was further developed by Tu et al. (2019) to use Graph Neural Network (GNN) pooling. Instead of using a graph kernel to capture the similarity among graphs, they use an end-to-end graph representation learning algorithm to create fixed-size embeddings for bags, making this an embedded-space method as opposed to a bag-space method. We refer to MIL GNN pooling as `Graph` in the remainder of this work.

3.2.4.3 Attention methods

Typical MIL pooling methods, whether in an instance-based or embedding-based context, treat all instances equally, i.e., they assume all of the instances contribute equally to the overall bag label. However, in reality, some instances will be more important than others, e.g., background regions in images are likely to be less important than foreground regions. Ilse et al. (2018) proposed a novel MIL neural network (see Section 3.2.4.1) approach that utilises attention-based pooling to weight the importance of instances. The attention mechanism used is a learnable function (as opposed to mean and max aggregation that are fixed), which is beneficial in adapting to the given task and data. This attention pooling was shown to give superior performance to conventional pooling methods. We refer to it as `Attention` in the remainder of this work.

One downside of `Attention` is that it does not provide complete instance predictions. Instead, it provides a measure of importance for each instance (the attention scores) — these are not class-specific. One solution that overcomes this issue is additive pooling (Javed et al., 2022). This pooling approach adapts attention pooling such that it makes

instance predictions while also utilising attention. As such, it can be considered more interpretable than attention pooling as instance predictions are class-specific. We refer to additive pooling as `Additive` in the remainder of this work.

In this thesis, we focus on developing and utilising interpretable MIL neural networks. In Figure 3.5 we provide a visual summary of the existing MIL pooling methods that re-occur throughout this work (borrowing from a figure presented in Chapter 7). In the next section, we discuss MIL interpretability in detail.

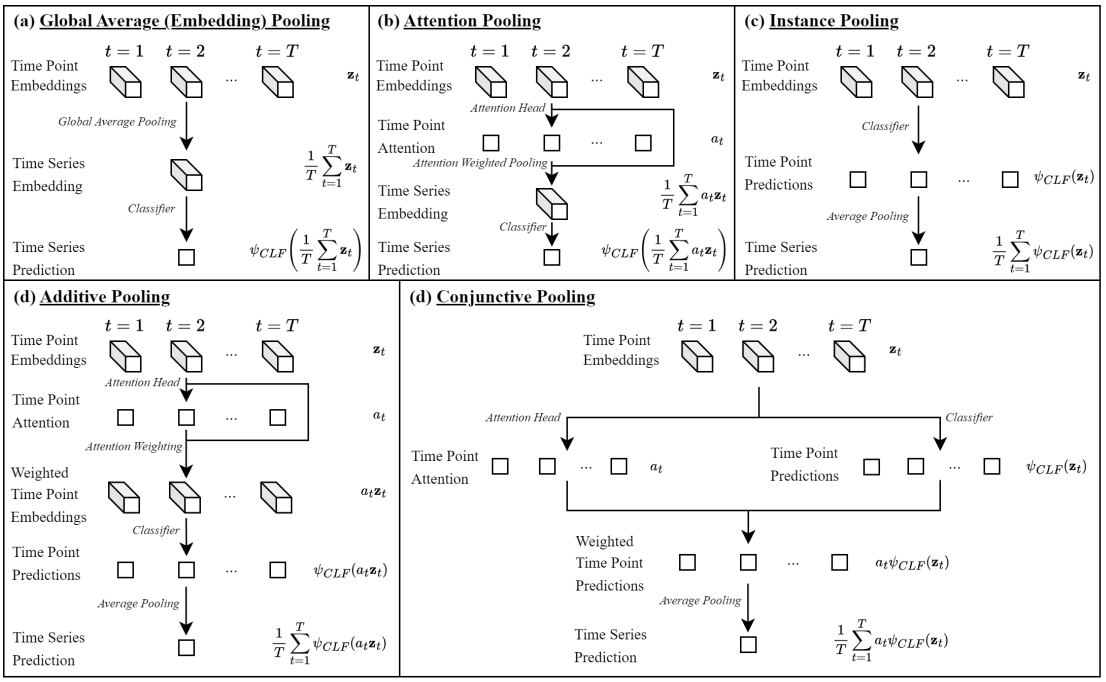

FIGURE 3.5: An overview of five different MIL pooling methods used. `Embedding`, `Attention`, `Instance`, and `Additive` are existing methods that are introduced above, and `Conjunctive` is a novel method proposed in Chapter 7. Foregoing specific detail for now, each pooling approach takes the same input (a collection of embedded vectors) and produces a bag-level prediction. However, they have different approaches for aggregating the information contained in the embeddings, and produce different interpretability outputs.

3.2.5 Interpretability for Multiple Instance Learning

In MIL, there are often only a few key instances within a bag that determine the bag label. For example, in SMIL, the bag label is only determined by the positive instances within a bag, which could be only a single instance in a bag containing over a hundred instances. To interpret the decision-making process of an MIL system, it is necessary to identify these key instances (Liu et al., 2012). The key instances have a causal influence on the bag label — there may be spurious associative links between the other instances and the bag label, but these will not be true causal relationships. This also means the resulting interpretations provided move up from rung one to rung two of the causal

ladder. Furthermore, in the case of key instance detection, the global scope would identify the concepts that differentiate key instances from other instances, and the local scope would identify the key instances for a particular input bag.

Interpreting the decision-making of MIL systems is complicated by noise in the data. As only some of the instances in a bag will be discriminatory key instances, a significant number of the instances in a bag will be non-discriminatory or even related to other bag classes (Amores, 2013). Not only does this confusing information make the learning problem more difficult, but it also complicates the interpretation of the model, as there could be a large amount of non-discriminatory data that is not relevant to the model's decision-making process. An important part of communicating appropriate interpretations of a model's decision-making process is not overloading the explainee with too much information (Li et al., 2016). Identifying the key instances and presenting those as the interpretation of model decision-making filters out the non-discriminatory information and provides the explainee with a reduced and relevant interpretation. Along with an appropriate social process (i.e., the context and meaning of the key instances), this interpretation facilitates human understanding as the causal relationship becomes apparent: these are the instances that determine the bag label, and the other instances are irrelevant.

In Chapter 4, we identify two questions that interpretability methods for MIL should be able to answer: 1) Which are the key instances for a bag? 2) What outcome (class/value) does each key instance support? We refer to these two questions as the *which* and the *what* questions respectively. As we explore below, existing interpretability methods can answer question one, but can only answer question two under the SMIL assumption. From a causal perspective, *which* questions are equivalent to identifying that causal relationships exist between the key instances and the bag label, and *what* questions are about understanding the nature of said relationships.

Under the SMIL assumption, *which* and *what* questions are equivalent — there are only two classes (positive and negative), and the only key instances are the positive instances, therefore once the key instances are identified, it is also known what outcome they support. However, in MIL problems beyond the SMIL assumption, answering *which* and *what* questions becomes two distinct problems. For example, if there is more than one positive class, some key instances will support one positive class, and some key instances will support another. Therefore, solely identifying the key instances does not answer the second question of which class they support. Below we outline previous methods that provide some level of interpretability for MIL systems and consider how well they are able to answer these *which* and *what* questions.

3.2.5.1 Inherent interpretability

Models that produce instance-level predictions, such as `Instance` (Wang et al., 2018) and `Additive` (Javed et al., 2022), are local inherently interpretable methods. Instance-level predictions can answer both the *which* and *what* questions. However, as the prediction is often made independently for each instance, they do not take account of interactions between the instances, so sometimes provide inaccurate predictions. Additional drawbacks of inherently interpretable methods are a lack of flexibility in model choice and a potential sacrifice in predictive power.

3.2.5.2 Key instance detection

Liu et al. (2012) proposed a model-specific method for identifying key instances directly, rather than as a by-product of learning bag labels. They developed a voting framework that utilises a network-based representation of the data to learn to identify key instances. This was done under the SMIL context, where the only key instances are positive instances, so this method only satisfies the two questions but only under the SMIL assumption.

3.2.5.3 Attention

In the context of MIL, attention can be used to learn to attend to important instances within bags (Ilse et al., 2018). The model does not produce instance-level predictions, but the attention values are a measure of how important each instance is for a particular input (a form of local interpretation). These values can be used to identify the key instances, but cannot say which class they support, i.e., attention can only answer *which* questions, not *what* questions. An additional drawback is that the method is model-specific as it can only be used for neural networks, and those networks must use `Attention`.

3.2.5.4 GNNs

GNNs can be used in MIL to capture the structural relationships between instances in bags. By collapsing the learnt graph representation into clusters, the key instances can be identified by how likely they are to belong to a given cluster. Tu et al. (2019) give examples of key instance identification using `Graph` under the SMIL assumption, but it has not been demonstrated for more general cases. It is also a model-specific method, as it relies on the assignment matrices produced as part of the GNN learning process.

In general, existing methods are mostly local approaches focused on the SMIL assumption, and either cannot generalise to other MIL problems or can only answer *which* questions in more general cases. To answer *what* questions, interpretations need to be produced with respect to a particular class. We summarise the inherent interpretability of MIL neural network approaches in Table 3.1.

TABLE 3.1: A summary of the inherent interpretability of MIL neural network approaches. *which* and *what* indicate the interpretability questions the models are inherently able to answer for general MIL problems.

Method	Inherent Interpretability
`Embedding` (Wang et al., 2018)	None.
`Instance` (Wang et al., 2018)	Instance predictions (*which* and *what*).
`Attention` (Ilse et al., 2018)	Instance attention scores (*which*).
`Additive` (Javed et al., 2022)	Instance predictions and attention (*which* and *what*).
`Graph` (Tu et al., 2019)	Graph representation clustering (*which*).

3.3 Discussion

The more recent methods proposed for MIL problems follow a similar theme: they all take older MIL ideas from before the emergence of DL and update them to utilise new methods from supervised learning. Many of the problems that are being solved are the same, but there is now an arsenal of new methods that can be used to tackle them. Identifying the relationships between instances is a prime example of this — the graph kernel method of Zhou et al. (2009) and the GNN method of Tu et al. (2019) are both tackling the same issue with similar approaches, yet the latter method benefits from an additional decade of knowledge on graph representation learning.

The inclusion of DL techniques is also beneficial in scaling up the algorithms to more complex datasets. Techniques such as convolutional layers can be easily integrated into existing MIL algorithms by upgrading the Feature Extractor (FE) or classifiers, whilst still using the same pooling techniques as previous methods. This allows MIL to be applied to much larger datasets such as gigapixel whole slide images in histopathology research (Hashimoto et al., 2020; Javed et al., 2022; Li et al., 2021a; Li and Zhang, 2020; Marini et al., 2021).

It is also important to consider how well these methods generalise to any MIL problem. Previous techniques often focused solely on the SMIL problem, and while a multi-class problem can be transformed into a binary problem using one-versus-all classifiers, this does not scale well when the training time of models and the number of classes increases (as is seen with modern ML models and datasets). The flexibility provided by modern approaches means the methods no longer need to be designed to exploit the existing assumptions about the problem, rather they can be used out of the box and

adapt to any MIL problem. This highlights a trend where the development of MIL methods no longer focuses so heavily on SMIL or other such assumptions. This also makes MIL applicable to other problems where these assumptions do not hold, and maybe even as an alternative approach for some standard supervised learning problems. We give our own example of this in Chapter 5, where MIL is used for semantic segmentation of high-resolution Earth Observation (EO) imagery — an approach that typically requires segmentation labels that are expensive and time-consuming to obtain.

MIL is an active area of research. Beyond the applications mentioned above, additional topics are being explored. One such topic is multi-MIL, where the instances within a bag are arranged into further bags, giving a hierarchical bags-of-bags structure (Fuster et al., 2022; Tibo et al., 2017, 2020). Another topic is multi-modal MIL, where two or more modalities are used within bags (Li et al., 2021b,c; Wang et al., 2023). For example, this could be using tabular data alongside image data.

In the next chapter, we discuss the first piece of novel research, which is focused on developing a MIL post-hoc model-agnostic interpretability method that can answer both *which* and *what* questions for general MIL classification problems with multiple classes. In Chapter 5, this is extended to a MIL regression problem specifically focused on interpretability for high-resolution images in EO applications.

Chapters 4 and 5 are mostly focused on computer vision applications of MIL. Therefore in Chapters 6 and 7, we explore the development of interpretable MIL methods for sequential data, investigating Reward Modelling (RM) and TSC. Investigating interpretable MIL in different application domains and ML paradigms naturally leads to the development of different models and techniques. However, there are common themes that occur across these different areas. Underpinning each of them is the core of MIL interpretability, which we explore next.

Part II

Interpretable Multiple Instance Learning for Vision

Chapter 4

Model Agnostic Interpretability for Multiple Instance Learning

This chapter contains work completed for a paper published at the International Conference of Learning Representations (ICLR) 2022, which was a virtual conference due to the COVID-19 pandemic. The content presented is mostly unchanged from its original format: elements of the original appendix have been brought into the main body, certain sections have been slightly reworded, and some terminology/notation has been modified for consistency with the rest of this thesis.

Paper

Joseph Early, Christine Evers, and Sarvapali Ramchurn. Model agnostic interpretability for multiple instance learning. In *International Conference on Learning Representations*, 2022c. URL https://openreview.net/forum?id=KSSfF5lMIAg

GitHub: https://github.com/JAEarly/MILLI

4.1 Introduction

In Multiple Instance Learning (MIL), data is organised into bags of instances, and each bag is given a single label. This reduces the burden of labelling, as each instance does not have to be assigned a label, making it useful in applications where labelling is expensive, such as healthcare (Carbonneau et al., 2018). However, there are often only a few key instances within a bag that determine the bag label, so it is necessary to identify these key instances in order to interpret a model's decision-making (Liu et al., 2012). Directly interpreting the decision-making process of Machine Learning (ML) methods is difficult due to the complexity of the models and the scale of the data on which they trained, so there is a need for methods that allow insights into the decision-making processes (Gilpin et al., 2018). In MIL problems, only some of the instances in each bag will be discriminatory with respect to a particular class, i.e., a significant number of the instances in a bag will background (non-key) instances or key instances that are discriminatory for other bag classes (Amores, 2013). Identifying the important instances and presenting those as the interpretation of model decision-making filters out the non-discriminatory information and provides the explainee with a reduced and relevant interpretation, which will avoid overloading the user (Li et al., 2016). This work provides the following novel contributions:

1. **MIL interpretability definition** We identify two questions that interpretability methods for MIL should be able to answer: 1) Which are the key instances for a bag? 2) What outcome (class/value) does each key instance support? In the rest of this work, we will refer to these two questions as the *which* and the *what* questions respectively. As we explore in Section 4.2, existing interpretability methods are able to answer *which* questions with varying degrees of accuracy, but can only answer *what* questions under certain assumptions.

2. **MIL model-agnostic interpretability methods** Existing interpretability methods are often model-specific, i.e., they can only be applied to certain types of MIL models. To this end, we build upon the State-Of-The-Art (SOTA) MIL inherent interpretability methods by developing model-agnostic interpretability methods that produce interpretations that are up to 30% more accurate than approaches such as LIME (Ribeiro et al., 2016b) and SHAP (Lundberg and Lee, 2017). The model-agnostic interpretability methods that we propose can be applied to any MIL model, and are able to answer both *which* and *what* questions.

3. **MIL interpretability method comparison** We compare existing model-specific methods with our model-agnostic methods on several MIL datasets. Our experiments are also carried out on four types of MIL models, giving a comprehensive comparison of interpretability performance. To the best of the authors' knowledge, this is one of the first studies to compare different interpretability methods for MIL.

The remainder of this work is laid out as follows. Section 4.2 provides relevant background knowledge and reviews existing MIL interpretability methods. Section 4.3 outlines the requirements for MIL interpretability and details our approaches that meet these requirements. Next Section 4.4 provides our experimental setup, and Section 4.5 provides results. Section 4.6 discusses our findings, and Section 4.7 concludes.

4.2 Background and Related Work

The Standard MIL (SMIL) assumption is a binary problem with positive and negative bags (Dietterich et al., 1997; Maron and Lozano-Pérez, 1997). A bag is positive if any of its instances are positive, otherwise, it is negative. The assumptions of SMIL can be relaxed to allow more generalised versions of MIL, e.g., through extensions that include additional positive classes (Scott et al., 2005; Weidmann et al., 2003). SMIL also assumes there is no interaction between instances. However, recent work has highlighted that modelling relationships between instances is beneficial for performance (Tu et al., 2019; Zhou et al., 2009). Existing methods for interpreting MIL models often rely on the SMIL assumption, so cannot generalise to other MIL problems. In addition, existing methods are often model-specific, i.e., they only work for certain types of MIL models, which constrains the choice of model (Ribeiro et al., 2016b). Identifying the key instances in MIL bags is a form of local interpretability (Molnar, 2020), as the key instances are detected for a particular input to the model. Two of the key motivators for interpreting the decision-making process of a MIL system are reliability and trust — identifying the key instances allows an evaluation of the reliability of the system, which increases trust as the decision-making process becomes more transparent. In this work, we use the term interpretability rather than explainability to convey that the analysis remains tied to the models, i.e., these methods do not provide non-technical explanations in human terms.

Under the SMIL assumption, *which* and *what* questions are equivalent — there are only two classes (positive and negative), and the only key instances are the positive instances, therefore once the key instances are identified, it is also known what outcome they support. However, when there are multiple positive classes, if instances from different classes co-occur in the same bags, answering *which* and *what* questions becomes two distinct problems. For example, some key instances will support one positive class, and some key instances will support another. Therefore, solely identifying *which* are the key instances does not answer the second question of *what* class they support. Existing methods, such as key instance detection (Liu et al., 2012), MIL attention (`Attention`; Ilse et al., 2018), and MIL Graph Neural Network (GNN) (`Graph`; Tu et al., 2019) do not condition their output on a particular class, so it is not apparent what class each instance supports, i.e., they can only answer *which* questions. One existing method that can answer *what* questions is mi-Net (`Instance`; Wang et al., 2018), as it produces

instance-level predictions as part of its processing. However, these instance-level predictions do not take account of interactions between the instances, so are often inaccurate. In this work, we aim to overcome the limitations of existing methods by developing model-agnostic methods that can answer both *which* and *what* questions.

In single-instance supervised learning, model-agnostic techniques have been developed to interpret models. Post-hoc local interpretability methods, such as Local Interpretable Model-agnostic Explanations (LIME) proposed by Ribeiro et al. (2016a) and Shapley Additive Explanations (SHAP) proposed by Lundberg and Lee (2017), work by approximating the original predictive model with a locally faithful surrogate model that is inherently interpretable. The surrogate model learns from simplified inputs that represent perturbations of the original input that is being analysed. In this work, one of the proposed methods is a post-hoc local approach similar to LIME and SHAP but specifically designed for MIL.

4.3 Methodology

At the start of this section, we outline the requirements for MIL interpretability (Section 4.3.1). In Section 4.3.2, we propose three methods that meet these requirements under the assumption that there are no interactions between instances (independent-instance methods). In Section 4.3.3 we remove this assumption and propose our local surrogate model-agnostic interpretability method for MIL.

4.3.1 Multiple Instance Learning Interpretability Requirements

A general MIL classification problem has C possible classes, with one class being negative and the rest being positive. This is a generalisation of the SMIL assumption, which is a special case when $C = 2$. An interpretability method that can only provide the general importance of an instance (without associating it with a particular class) can only answer *which* questions. In order to answer *what* questions, the method needs to state which classes each instance supports and refutes. Formally, for a bag of instances $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ and a bag classification function F, we want to assign a value to each instance that represents whether it supports or refutes a class $c \in \{0, \ldots, C-1\}$:

$$\mathcal{I}(\mathbf{X}, F, c) = \{\phi_1, \ldots, \phi_k\}$$

where $\mathcal{I}$ is the interpretability function and $\phi_i \in \mathbb{R}$ is the interpretability value for instance i with respect to class c. Here, we assume that $\phi_i > 0$ means instance i supports class c, $\phi_i < 0$ implies instance i refutes class c, and the greater $|\phi_i|$, the greater the

importance of instance i. For some existing methods, such as attention, ϕ_i is the same for all classes and $\phi_i \geq 0$ in all cases, meaning these requirements are not met and performance is reduced (see Section 4.6). In the next two sections, we propose several model-agnostic methods that satisfy these requirements.

4.3.2 Determining Instance Attributions

In this section, we propose three model-agnostic methods for interpreting MIL models under the assumption that the instances are independent. This means we can observe the effects of each instance in isolation without worrying about interactions between the instances. We exploit a property of MIL models: the ability to deal with different-sized bags. As MIL models can process bags of different sizes, it is possible to remove instances from the bags and observe any changes in prediction, allowing us to understand what instances are responsible for the model's prediction. Below, we propose three methods that use this property to interpret a model's decision-making.

Single Given a bag of instances $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ and a bag classification function F, we can take each instance in turn and form a single instance bag $\{\mathbf{x}_i\}$ for $i \in \{1, \ldots, k\}$. We then observe the model's prediction on each single instance bag $\phi_i = F_c(\{\mathbf{x}_i\})$, where F_c is the softmax-normalised output of F for class c, i.e., the c^{th} entry in the softmax output vector of $F(\{\mathbf{x}_i\})$. A large value for ϕ_i suggests instance $\mathbf{x}_i$ supports c, and a value close to zero suggests it has no effect with respect to class c (note this is using softmax-normalised model outputs). If we repeat this over all instances and all classes, we can build a picture of the classes that each instance supports, allowing us to answer *what* questions. However, this method cannot refute classes (i.e., $\phi_i \geq 0$ in all cases). It should be noted that this method gives the same outputs as the inherent interpretability of `Instance` (Wang et al., 2018), but here it is a model-agnostic rather than a model-specific method.

One Removed A natural counterpart to the Single method is the One Removed method, where each instance is removed from the complete bag in turn, i.e., we form bags $\mathbf{X} \setminus \{\mathbf{x}_i\}$. For a particular class c, we can then observe the change in the model's prediction caused by removing $\mathbf{x}_i$ from the bag: $\phi_i = F_c(\mathbf{X}) - F_c(\mathbf{X} \setminus \{\mathbf{x}_i\})$. If the prediction decreases, x_i supports c, and if it increases, $\mathbf{x}_i$ refutes c, i.e., this method can both support and refute different classes. However, if there are other instances in the bag that support or refute class c, we may not observe a change in prediction when $\mathbf{x}_i$ is removed, even if $\mathbf{x}_i$ is a key instance.

Combined To access the benefits of both the Single and One Removed methods, we can combine their outputs. A simple approach is to take the mean, i.e., $\phi_i = \frac{1}{2}[F_c(\{\mathbf{x}_i\}) + F_c(\mathbf{X}) - F_c(\mathbf{X} \setminus \{\mathbf{x}_i\})]$. This method can identify the important instances revealed by the Single method, and also refute outcomes as revealed by the One Removed method.

With these three methods, it is assumed that there are no interactions between the instances. This is not true for all datasets, therefore, in the next section, we remove this assumption and propose a further method that can deal with the interactions between instances.

4.3.3 Dealing with Instance Interactions

To calculate instance attributions whilst accounting for interactions between instances, we have to consider the effect of each instance within the context of the bag. With instance interactions, the co-occurrence of two (or more) instances changes the bag label from what it would be if the two instances were observed independently. The instances have different meanings depending on the context of the bag, i.e., on their own they mean something different to what they mean when observed together. One way to uncover these instance interactions is to perturb the original input to the model and observe the outcome. In the case of MIL, these perturbations can take the form of removing instances from the original bag. By sampling coalitions of instances, and fitting a weighted linear regression model against the coalitions and their respective model predictions, it is possible to construct a surrogate locally faithful interpretable model that accounts for the instance interactions in the original bag.

Below, we propose `MILLI`**: M**ultiple **I**nstance **L**earning **L**ocal **I**nterpretations. At the core of our approach is a new method for sampling and weighting coalitions based on an initial ranking of instance importances $\mathbf{r} = \{r_1, \ldots, r_k\}$, i.e., focusing on the content of coalitions (initial instance importances) and their size, not solely by their size as in the comparable approaches of SHAP and LIME. We summarise this approach in Algorithm 1, and then provide further details for each step of the algorithm.

Algorithm 1 `MILLI`

Require: Bag of instances $\mathbf{x}_i \in \mathbf{X}$, class c, number of samples n.

1. Calculate initial instance importance scores for class c: $\mathbf{r} = \{r_1, \ldots, r_k\}$.
2. Sample n coalitions $\mathbf{V}$:

while $|\mathbf{V}| < n$ **do**

Sample coalition $\mathbf{v}$ with $P(\mathbf{v}_i = 1) = p_i$ and $P(\mathbf{v}_i = 0) = 1 - p_i$, where $p_i = \pi_R(r_i)$.

Append $\mathbf{v}$ to $\mathbf{V}$.

end while

3. Fit a surrogate local model $G_c(\mathbf{v}) = \phi_0 + \sum_{i=1}^{k} \phi_i \mathbf{v}_i$ using the sampled set of coalitions $\mathbf{V}$. The learnt weights $\phi_i \in \{\phi_1, \ldots, \phi_k\}$ of G_c are the interpretability scores for $\mathbf{X}$ with respect to class c.

4.3.3.1 `MILLI`: Initial ranking

The determine an initial ranking of instance importance, we use the Single method from Section 4.3.2, which produces initial importance values $\{\phi'_1, \ldots, \phi'_k\}$. This is converted to an instance ranking: the new values $\{r_1, \ldots, r_k\}$ represent the position of instance i in $\{\phi'_1, \ldots, \phi'_k\}$ ordered from largest to smallest (e.g., the instance with the greatest value for ϕ'_i has $r_i = 0$).

4.3.3.2 `MILLI`: Coalition sampling

A coalition is a binary vector $\mathbf{v} \in \{0,1\}^k$ that represents a sub-bag $\mathbf{S_X} = \{\mathbf{x}_i | \mathbf{v}_i = 1\}$, so the number of ones in the coalition, $|\mathbf{v}|$, is equal to the length of $\mathbf{S_X}$. To fit a local surrogate model, we first need to sample a collection of n coalitions $\mathbf{V}$ (effectively a matrix of shape $\mathbb{R}^{n \times k}$). We can consider the problem of sampling as a repeated coin toss, where $P(\mathbf{v}_i = 1) = p_i$ and $P(\mathbf{v}_i = 0) = 1 - p_i$. In equal random sampling, every value $p_i \in \{p_1, \ldots, p_k\}$ is equal to 0.5, meaning $\mathbb{E}[|\mathbf{v}|] = 0.5k$. However, it is possible to improve upon equal random sampling by changing the value of p for each instance in the bag, i.e., we can change the likelihood of each instance being involved in a coalition. To sample more informative coalitions, we set $p_i = \pi_R(r_i)$:

$$\pi_R(r_i) = \begin{cases} (2\alpha - 1)(1 - \frac{r_i}{k})e^{-\hat{\beta} r_i} + 1 - \alpha, & \text{if } \hat{\beta} \geq 0, \\ (1 - 2\alpha)(1 + \frac{r_i - k}{k})e^{|\hat{\beta}|(r_i - k)} + \alpha, & \text{otherwise,} \end{cases}$$
$$\text{and } \hat{\beta} = \begin{cases} \beta, & \text{if } \alpha < 0.5, \\ -\beta, & \text{otherwise.} \end{cases} \tag{4.1}$$

The resulting expected coalition size $\mathbb{E}[|\mathbf{v}|] = \int_1^k \pi_R \, dr$:

$$\mathbb{E}[|\mathbf{v}|] = \begin{cases} \frac{2\alpha - 1}{k\hat{\beta}^2}(e^{-\hat{\beta}k} + \hat{\beta}k - 1) + k(1 - \alpha), & \text{if } \hat{\beta} \geq 0, \\ \frac{1 - 2\alpha}{k\hat{\beta}^2}(e^{-\hat{\beta}k} + \hat{\beta}k - 1) + k\alpha, & \text{otherwise.} \end{cases} \tag{4.2}$$

π_R is a function that weights instances based on their order in the ranking. Its two hyperparameters, $\alpha \in [0,1]$ and $\beta \in (-\infty, \infty)$ define the shape of the function. The value of α dictates whether the sampling should be biased towards instances that are highly ranked or not: $\alpha > 0.5$ means π_R is biased towards instances higher in the ordering, and $\alpha < 0.5$ means π_R is biased towards values lower in the ordering. If $\beta < 0$, the sampling is biased towards smaller coalitions, and if $\beta > 0$, the sampling is biased

towards larger coalitions. The maximum and minimum $\mathbb{E}[|\mathbf{v}|]$ is controlled by α. When $\alpha = 0.5$ or $\beta = 0$, every value $p_i \in \{p_1, \ldots, p_k\}$ is equal to 0.5, meaning we have equal random sampling as described above. We provide an illustration of π_R in Figure 4.1, and an overview of how $\mathbb{E}[|\mathbf{v}|]$ changes in Figure 4.2.

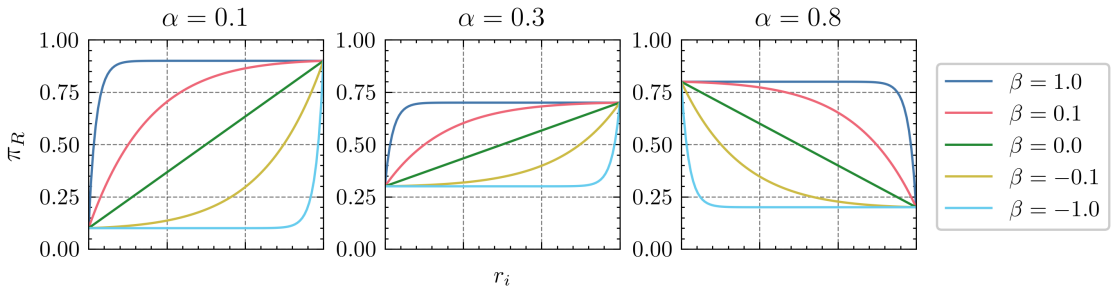

FIGURE 4.1: The effect of α and β on π_R (Equation 4.1).

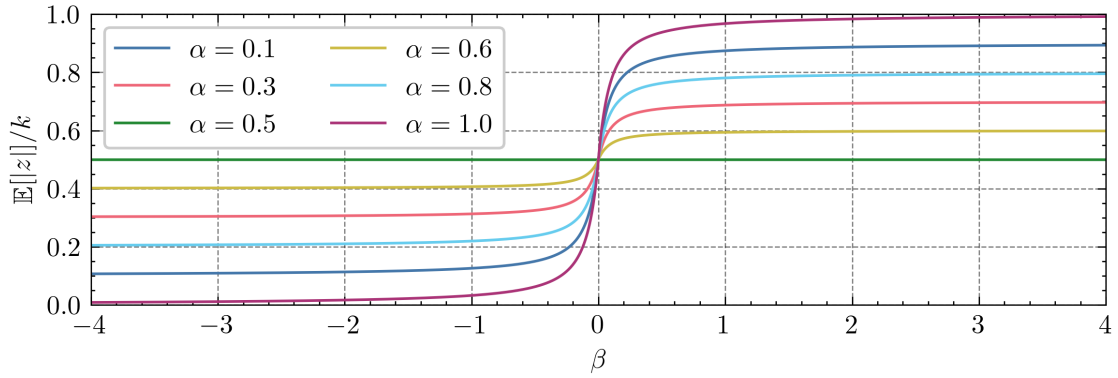

FIGURE 4.2: The effect of α and β on $\mathbb{E}[|\mathbf{v}|]$ (Equation 4.2). We give the expected coalition size as a proportion of the bag size, i.e., $\mathbb{E}[|\mathbf{v}|]/k$.

4.3.3.3 `MILLI`: Surrogate model

The surrogate model G_c is a linear model of the form

$$G_c(\mathbf{v}) = \phi_0 + \sum_{i=1}^{k} \phi_i \mathbf{v}_i, \tag{4.3}$$

where each coefficient $\phi_i \in \{\phi_1, \ldots, \phi_k\}$ is a weight in the model. We use ϕ_i for the weights as these are the resulting importance attributions (interpretability) scores for each instance $\mathbf{x}_i \in \mathbf{X}$ with respect to class c, i.e., the weights of this linear model tell us how much each instance supports/refutes class c. Given the collection of n coalitions $\mathbf{V}$, minimising the loss function

$$L(F, G_c, \pi) = \sum_{\mathbf{v} \in \mathbf{V}} [F_c(\mathbf{S}_\mathbf{X}) - G_c(\mathbf{v})]^2 \pi(\mathbf{v}), \tag{4.4}$$

means G_c is a locally faithful approximation of the original model F for class c.

The loss function is weighted by a kernel π, which determines how important it is for G_c to be faithful for each individual coalition. Here, G_c only approximates F for class c, i.e., to produce interpretations for a bag with respect to all classes, a surrogate model needs to be fit for each class. Similar approaches have been applied in single-instance supervised learning in methods such as LIME and KernelSHAP. Both use different choices for π: LIME employs l_2 or cosine distance, and KernelSHAP uses a weighting scheme that approximates Shapley values (Shapley, 1953). For single instance supervised learning models, when perturbing the inputs, it is not possible to simply 'remove' a feature from an input as the models expect fixed-size inputs — either a new model has to be re-trained without that particular feature, or appropriate sampling from other data has to be undertaken, which can lead to unrealistic synthetic data. In MIL, as models are able to deal with variable-sized bags, instances can be removed from the bags without the need for re-training or sampling from other data, meaning these issues do not occur in our setting.

While it is possible to utilise the weight kernels from LIME and KernelSHAP for MIL, we identify a significant drawback with both methods. Their choice for π weights all coalitions of the same size equally, i.e., they do not consider the content of the coalitions, only their size. Just because two coalitions are of the same size does not mean it is equally important that the surrogate model is faithful to both of them. Furthermore, the sampling strategies of both approaches lead to very large ($|\mathbf{v}|$ close to k) or very small coalitions ($|\mathbf{v}|$ close to 0). Unless the number of samples n is very large, samples of average size ($|\mathbf{v}|$ close to $k/2$) will not be chosen. As we see in Section 4.5, the LIME and KernelSHAP sampling approaches are appropriate for some datasets, but not for others. For a coalition $\mathbf{v}$ and ranking of instance importances $\mathbf{r}$, we define our weight kernel π_M as:

$$\pi_M(\mathbf{v}, \mathbf{r}) = \frac{1}{|\mathbf{v}|} \sum_{i=1}^{k} \mathbf{v}_i \, \pi_R(r_i). \tag{4.5}$$

Note π_R is the same weighting function as used in the coalition sampling approach detailed above. In π_M, the coalition $\mathbf{v}$ is weighted on its content rather than its length, i.e., the distance between a subset and a bag is no longer determined simply by the number of instances removed. This is beneficial, as removing many non-discriminatory instances from the bag likely means the label of the bag remains the same, despite being different in size. Conversely, removing one discriminatory instance could change the bag label, even though the bag is only one instance smaller.

`MILLI` is more expensive to compute that the methods outlined in Section 4.3.2: $\mathcal{O}(Cnk^2)$ compared with $\mathcal{O}(Ck)$. However, when there are interactions between the

instances, this extra complexity is required in order to build a better understanding of the classes that each instance supports and refutes, allowing more accurate answering of *what* questions. It is important to note that `MILLI` is indeed model agnostic — it only requires access to the bag classification function F, and makes no assumptions about the underlying MIL model. As we demonstrate in the next section, this means we can apply it to any MIL model.

4.4 Experimental Setup

We apply our model-agnostic methods to seven MIL datasets. In this section, we detail the evaluation strategy (Section 4.4.1), datasets (Section 4.4.2), models (Section 4.4.3), interpretability hyperparameters (Section 4.4.4), and training process (Section 4.4.5). For implementation details, see Appendix A.1.

4.4.1 Evaluation Strategy

For our evaluation, we do not assume that the interpretability methods have consistent output domains; we only assume a larger value implies larger support. Therefore, the best approach for evaluating the interpretability methods is to use ranking metrics — rather than looking at the absolute values produced by the methods, we compare the relative orderings they produce. As noted by Carbonneau et al. (2018), although a large number of benchmark MIL datasets exist, many do not have instance labels. Without instance labels, evaluating interpretability is challenging, as we do not have a ground truth instance ordering to compare to. We identify two appropriate evaluation metrics: for datasets with instance labels, we propose the use of Normalised Discounted Cumulative Gain at n (NDCG@n), and for datasets without instance labels, we use Area Over the Perturbation Curve to Random (AOPCR), originally proposed by Samek et al. (2017). While AOPCR does not require instance labels, it is very expensive to compute. A significant difference between the two metrics is the way they weight the importance ordering: NDCG@n prioritises performance at the start of the ordering, whereas AOPCR equally prioritises performance across the entire ordering. We are the first to propose both of these metrics for use in evaluating MIL interpretability, and further discuss each metric below.

AOPCR Although originally designed for single instance supervised learning, here we adapt AOPCR for MIL. Given an importance ordering, we successively remove the most important instances (i.e., those with the highest values for ϕ_i) and measure the change in prediction. A better ordering will show a sharper decrease in prediction, as more supportive instances will be removed first. Conversely, a random or incorrect

ordering will lead to a much slower decrease in prediction. Formally, when evaluating the interpretations generated by a classifier F for $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ with respect to class c, we first re-order the bag according to the instance importance scores for class c, with the most important instances first: $\mathbf{O}_{\mathbf{X},c} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_k\}$. Note the contents of $\mathbf{O}_{\mathbf{X},c}$ are the same as that of $\mathbf{X}$ but with a different order of instances. The perturbation metric is calculated by:

$$AOPC(\mathbf{O}_{\mathbf{X},c}) = \frac{1}{k-1} \sum_{j=1}^{k-1} F_c(\mathbf{O}_{\mathbf{X},c}) - F_c(\mathbf{O}_{\mathbf{X},c} \setminus \{\mathbf{o}_1, \ldots, \mathbf{o}_j\}). \tag{4.6}$$

To normalise the results, one approach is to compare against the AOPC scores for random orderings. To compensate for the stochastic nature of using random orderings, we average over several:

$$AOPCR(\mathbf{O}_{\mathbf{X},c}) = \frac{1}{n_{Rand}} \sum_{r=1}^{n_{Rand}} \Big(AOPC(\mathbf{O}_{\mathbf{X},c}) - AOPC(\mathbf{O}_{\mathbf{X},Rand}) \Big), \tag{4.7}$$

where $\mathbf{O}_{\mathbf{X},Rand}$ is a random ordering of $\mathbf{X}$ and n_{Rand} is the number of random orderings. The repeated calls to F in Equation 4.6 make AOPCR very expensive to compute, especially when comparing to random orderings — in our experiments $n_{Rand} = 10$. This can be reduced by limiting the calculation to only consider the first few instances rather than all $k-1$, but that was not something we investigated in this work. Due to the use of random orderings, AOPCR is inherently stochastic, meaning not only do we have variance in the orderings produced in the methods (e.g., from random sampling), but we also have variance due to measurement. We reduced this noise by keeping the random orderings consistent across evaluations through the use of fixed random seeds. Compared to the NDCG@n, which is cheap to compute and deterministic, the distinct advantage of AOPCR is that it does not require instance labels.

NDCG@n To use NDCG@n, we first need a ground truth ordering to compare to. For a particular class, we can place each instance into one of three groups based on their ground truth instance labels: supporting instances, neutral instances, and refuting instances. The ideal instance ordering for that class would then have all of the supporting instances at the beginning, followed by all the neutral instances, and then all of the refuting instances at the end. This ground truth instance importance ordering is then compared to the ordering $\mathbf{O}_{\mathbf{X},c}$ for the first n instances:

$$
\begin{aligned}
NDCG@n(\mathbf{O}_{\mathbf{X},c}) &= \frac{1}{IDCG}\sum_{j=1}^{n}\frac{rel(\mathbf{o}_j)}{log_2(j+1)}, \\
\text{where } IDCG &= \sum_{j=1}^{n}\frac{1}{log_2(j+1)} \\
\text{and } rel(\mathbf{o}_j) &= \begin{cases} 1 & \text{if } \mathbf{o}_j \text{ supports class } c, \\ -1 & \text{if } \mathbf{o}_j \text{ refutes class } c, \\ 0 & \text{otherwise.} \end{cases}
\end{aligned} \tag{4.8}
$$

IDCG is the ideal discounted cumulative gain that normalises the scores across different values of n, and n is the number of instances in the bag that support class c.

4.4.2 Datasets

Below we detail the three main datasets on which we can evaluate our interpretability methods. These datasets were selected as they have instance labels, so we can evaluate them using both NDCG@n and AOPCR. However, we also used the classical MIL datasets Musk, Tiger, Elephant and Fox, which do not have instance labels.

4.4.2.1 Spatially Independent, Variable Area, and Lighting

The Spatially Independent, Variable Area, and Lighting (SIVAL) dataset (Rahmani et al., 2008) consists of 25 classes of complex objects photographed in different environments, where each class contains 60 images. Each image has been segmented into approximately 30 segments (one segment = one MIL instance), and each segment is represented by a 30-dimensional feature vector that encodes information such as the segment's colour and texture. The segments are labelled as containing the object or containing background. We selected 12 of the 25 classes to be positive classes, and randomly selected 30 images from each of the other 13 classes to form the negative class, meaning overall we had 13 classes (12 positive and one negative). In total, we had 60 bags for each of the 12 positive classes, and 390 bags for the single negative class, meaning the class distribution was $\approx 5.4\%$ for each positive class and $\approx 35.1\%$ for the negative class. The arbitrarily chosen positive classes were: *apple*, *banana*, *checkeredscarf*, *cokecan*, *dataminingbook*, *goldmedal*, *largespoon*, *rapbook*, *smileyfacedoll*, *spritecan*, *translucentbowl*, and *wd40can*. We normalised each instance according to the dataset mean and standard deviation. No other data augmentation was used. The dataset was separated into train, validation, and test data using an 80/10/10 split. This was done with stratified sampling in order to maintain the same data distribution across all splits.

4.4.2.2 Four-class MNIST-Bags

The SIVAL dataset has no co-occurrence of instances from different classes, i.e., each bag only contains one class of positive instances. Therefore, the *which* and *what* questions are the same. To explore what happens when there are different classes of positive instances in the same bag, we propose an extension of the MNIST-Bags dataset introduced by Ilse et al. (2018). Our extension, four-class MNIST-Bags (*4-MNIST-Bags*) is set up as follows: class 1 if **8** in the bag, class 2 if **9** in the bag, class 3 in **8** and **9** in the bag, and class 0 otherwise.[1] In this dataset, answering *what* questions goes beyond answering *which* questions, e.g., the existence of an **8** supports classes one and three, but refutes classes zero and two. The bag sizes were drawn from a normal distribution, with a mean of 30 and a variance of 2. We used 2500 training bags, 1000 validation bags, and 1000 test bags. The instances in the training bags were only drawn from the original MNIST training split, and the instances in the validation and test bags were only drawn from the original MNIST test split, i.e., there was no overlap between training, validation, and test instances. The classes were balanced, so, on average, there were 625 bags per class in the training data, and 250 bags per class in the validation and test data. We normalised the MNIST images using the PyTorch normalise transformation, with a mean of 0.1307 and a stand deviation of 0.3081. No other data augmentation was carried out.

4.4.2.3 Colorectal Cancer

To test the applicability of the interpretability methods on larger bag sizes, we apply it to the Colorectal Cancer (CRC) tissue classification dataset (Sirinukunwattana et al., 2016). This is a collection of 100 microscopy images with partially annotated nuclei. Each image is annotated with four classes of nuclei: epithelial, inflammatory, fibroblast, and miscellaneous. We follow the same MIL labelling process as Ilse et al. (2018), in which a bag is positive if it contains one or more nuclei from the epithelial class. This means the problem conforms to the SMIL assumption, and there are no interactions between instances. Of the 100 images, 50 are in the negative class (no epithelial nuclei), and 50 are in the positive class (at least one epithelial nuclei).

Each microscopy image is 500 x 500 pixels and was split into a non-overlapping grid of 27 x 27 pixel patches (324 patches per slide). An alternative patch extraction process, and the process originally used by Ilse et al. (2018), is to extract patches centred on each marked nuclei. However, this method requires the nuclei to be marked for unseen data — something our approach does not require. Our alternative patch extraction approach explains the difference between our results and the results of Ilse et al. (2018) —

[1] We use **n** to refer to an image from the MNIST dataset that represents the number n, even though that is not the assigned class label in the *4-MNIST-Bags* dataset.

extracting the patches using a fixed grid will include patches that do not contain any marked nuclei (but are not just slide background), increasing the amount of non-discriminatory data in each bag and thus making the problem harder to learn.

While we extracted 324 patches per slide, some patches were discarded as they contained only slide background. This was achieved by using a brightness threshold — any patch with an average pixel value above 230 (using pixel values from 0 to 255) was discarded. This left an average of 264 patches per image, and one image was discarded as it had zero foreground patches (this was manually verified). The remaining patches were labelled based on the nuclei they contained. However, as some patches did not contain any marked nuclei, they could not be labelled (it is incorrect to assume they belong to the negative class as not all epithelial nuclei were labelled). As such, we tailored our evaluation using NDCG@n to only consider labelled instances.

The images were normalised using the dataset mean (0.8035, 0.6499, 0.8348) and standard deviation (0.0858, 0.1079, 0.0731). During training, we also applied three transformations to patches at random: horizontal flips, vertical flips, and 90-degree rotations. The dataset was separated into train, validation, and test data using a 60/20/20 split. This was done with stratified sampling in order to maintain the same data distribution across all splits.

4.4.2.4 Classical MIL Datasets

In addition to the three datasets listed above, we also conducted experiments with four classical MIL datasets.[2] The Musk dataset (Dietterich et al., 1997) presents a drug activity problem of identifying musks (aromatic substances) and non-musks, which is an SMIL problem.[3] The dataset contains 47 musks (positive class) and 45 non-musks (negative class), and each molecule (bag) has several potential conformations (instances). These different conformations (molecule shapes) are represented by 166 features. Features were normalised by subtracting the dataset mean and dividing by the dataset standard deviation. A random stratified 70/15/15 dataset split was used.

The Tiger, Elephant, and Fox datasets (TEF; Andrews et al., 2002) operate similarly to the SIVAL dataset detailed above. Each dataset consists of 100 positive examples of a target animal (e.g., pictures of tigers) and 100 negative examples of other animals. The images have been pre-processed using segmentation and feature extraction to give instance feature vectors of length 230 that capture properties such as shape, texture, and colour. Unlike SIVAL, the instances (feature vectors describing image segments) are not labelled — only bag-level labels are provided. Each of the TEF datasets follows the same process but has a different target animal (tiger, elephant, or fox). Like Musk, we

[2]These classical datasets do not have instance labels, therefore can only be evaluated using AOPCR. They are also less complex than the other datasets, for example having only a few instances per bag. As such, we focus our analysis in later sections on the SIVAL, *4-MNIST-Bags*, and CRC datasets.

normalised by the dataset mean and standard deviation, and used a random stratified 70/15/15 dataset split.

4.4.3 Models and Methods

To highlight the model-agnostic abilities of the proposed methods, for each dataset, we trained four different types of multi-class MIL model: `Embedding` (Wang et al., 2018), `Instance` (Wang et al., 2018), `Attention` (Ilse et al., 2018), and `Graph` (Tu et al., 2019). Each model was tuned independently for each dataset, and we also tuned the learning rate, weight decay, and dropout hyperparameters. This tuning was achieved with the Optuna Python library (Akiba et al., 2019). For tuned model architectures and hyperparameters, see Appendix A. We provide further details on each model below.

`Embedding` Each instance is embedded to a fixed size, and then these embeddings are aggregated to give a single bag embedding. This aggregation can either take the mean (mil-mean) or max (mil-max) of the instance embeddings. The bag embedding is then classified to give an overall bag prediction. For this model, we tuned the number and size of fully connected (FC) layers, as well as the choice of aggregation function.

`Instance` A prediction is made for each individual instance, and then these are aggregated to a single outcome for the bag as a whole. Similar to the `Embedding` model, we tuned the number and size of FC layers and the aggregation function (mean or max).

`Attention` Each instance is embedded to a fixed size, and then the mil-attn block produces an attention value for each instance embedding. The instance embeddings are then aggregated to a single bag embedding by performing a weighted sum based on the attention values. The bag embedding is then classified to give an overall bag prediction. The mil-attn block is the same as per the MIL attention mechanism proposed by Ilse et al. (2018), i.e., using a single hidden layer. We tuned the number and size of the FC layers, as well as the size of the hidden attention layer.

`Graph` The GNN model treats the bag as a fully connected graph and uses graph convolutions to propagate information between instances. Initially, the original instances are embedded to a fixed size. These instance embeddings are then passed onto a $\text{GNN}_{\text{embed}}$ block and a $\text{GNN}_{\text{cluster}}$ block, the output of which is used to reduce the graph representation down into a single embedding using differentiable pooling. The final bag representation is then classified to give an overall bag prediction. For

[3]Note that we are using the Musk1 dataset, rather than Musk2, as the latter has much larger bag sizes, making the use of AOPCR infeasible.

more details see Tu et al. (2019). We tuned the number and size of the embedding, GNN, and classifier layers.

Aside from `Embedding`, each of the above models provides its own inherent interpretability method that we can compare our methods against. In addition to comparing our independent-instance methods and `MILLI` with the inherent interpretability of these models, we also compare against LIME and SHAP using both random sampling and guided sampling (choosing coalitions that maximise the weight kernel). This means that in total we are evaluating nine different interpretability methods: inherent interpretability, the three independent-instance methods (Section 4.3.2), two LIME methods, two SHAP methods, and `MILLI` (Section 4.3.3). Note that, aside from the inherent interpretability methods, these methods are either novel (independent-instance methods and `MILLI`) or are applied to MIL for the first time in this study (LIME and SHAP).

4.4.4 Interpretability Hyperparameter Selection

In this section, we discuss our method for hyperparameter selection in the interpretability methods and detail the hyperparameters that we found to be most effective. An advantage of the inherent interpretability and independent-instance methods is that they do not have hyperparameters, i.e., we only had to select hyperparameters for the local surrogate interpretability methods. First, we discuss our choice of hyperparameters for LIME, and then our choice of hyperparameters for `MILLI`.

LIME hyperparameters When using the LIME weight kernel, there are two hyperparameters to tune. Firstly, the distance measures that are commonly used are L2 and cosine distance. In our experiments, we found very little difference between each of these distance measures, therefore we arbitrarily chose to use L2 distance in all of our experiments. The second hyperparameter is the kernel width, which determines the weighting of coalitions, i.e., a large kernel width means all coalitions are weighted more evenly, and a small kernel width prioritises larger coalitions. We chose to use a kernel width for all of our experiments that is determined by the average bag size, such that the half coalition (i.e., $|\mathbf{v}| = 0.5k$) is weighted at 0.5.

`MILLI` hyperparameters For `MILLI`, there were three hyperparameters to tune: the number of samples, α, and β. For the number of samples, we generated sample size plots (see Section 4.6.2), and chose the number of samples to be at the point where all the methods had reasonably converged. For α and β we ran a grid search over the possible values and chose the best-performing pair of parameters. The hyperparameters were tuned for each dataset, except for the TEF datasets, in which we only tuned on

Tiger and then used the same hyperparameters across all three datasets. In Table 4.1, we provide the chosen hyperparameters for each dataset, as well as the expected coalition size $\mathbb{E}[|\mathbf{v}|]$ determined by α and β. For the CRC, Musk and TEF datasets, the chosen parameters produce small values for $\mathbb{E}[|\mathbf{v}|]$, i.e., the sampling is heavily biased towards smaller coalitions, which is expected as these are instance-independent datasets. Conversely, the parameters for the *4-MNIST-Bags* focus on sampling larger coalitions that are able to capture the instance interactions in the dataset. Although the SIVAL dataset is instance-independent, the parameters also focus on sampling larger coalitions. This could be because, although the instances are independent, it may be difficult to classify an object from just one instance, i.e., several instances are needed to make the correct decision. We also note that, in all cases, $\alpha < 0.5$, i.e., the sampling is biased towards instances ranked lower in the initial importance ordering used by `MILLI`. Our explanation for this is that it is easy to understand the contribution of discriminatory instances, as they have a large effect on the model prediction, but it is more difficult to understand the contribution of non-discriminatory instances, as they have much less of an effect. Therefore, more samples containing non-discriminatory instances are required to properly understand their effect, hence biasing the sampling towards them.

TABLE 4.1: `MILLI` hyperparameters selected for the different MIL datasets.

Dataset	Sample Size	α	β	$\mathbb{E}[\|\mathbf{v}\|]$
SIVAL	200	0.05	-0.01	13
4-MNIST-Bags	150	0.05	0.01	16
CRC	1000	0.008	-5.0	2
MUSK	150	0.3	-1.0	2
TEF	150	0.3	0.01	3

4.4.5 Training Process

We used a consistent process for training all the models across all the datasets. A batch size of one was used, i.e., a batch of a single bag. Models were trained to minimise cross-entropy loss using the Adam optimiser. We utilised early stopping based on validation loss — if the validation loss had not decreased for 10 epochs then we terminated the training procedure and reset the model to the point at which it caused the last decrease in validation loss. Otherwise, the maximum number of epochs was 100. We tuned the learning rate, weight decay, and dropout for each model type and on each dataset.

For the *4-MNIST-Bags* and CRC datasets, the models used a convolutional Feature Extractor (FE). These FEs were kept consistent across each model, i.e., the architecture was fixed, but the weights were learnt. For *4-MNIST-Bags*, the FE produced feature

vectors of length 800, and for CRC they were of length 1200. For more details, see Appendix A.

4.5 Results

We split our results into two parts: first the more complex modern MIL datasets (SIVAL, *4-MNIST-Bags*, and CRC), and then the classic MIL datasets (Musk, Tiger, Elephant, and Fox).

4.5.1 Modern Multiple Instance Learning Dataset Results

For each dataset, we measured the performance of each interpretability method on each of the four MIL models. For the SIVAL dataset, we measured the performance on only the negative class and the bag's true class, but for all the other datasets we evaluated the interpretability over every class. This distinction was made as we know that each SIVAL bag can only contain instances from one positive class, therefore it is not necessary to evaluate over all possible classes. We present the interpretability results run against the test set for the SIVAL, *4-MNIST-Bags*, and CRC datasets in Tables 4.2, 4.3, and 4.4 respectively. The results are averaged over ten repeat trainings of each model, and we also give the test accuracy of each of the underlying MIL models.

By analysing the NDCG@n interpretability results across these three datasets, we find that `MILLI` performs best with an average of 0.85, followed by the GuidedSHAP and Combined methods (both with an average of 0.81). For the average AOPCR results

TABLE 4.2: SIVAL interpretability NDCG@n / AOPCR results. For all of the interpretability methods, the standard error of the mean was 0.01 or less.

Methods	`Embedding`	`Instance`	`Attention`	`Graph`	Overall
Model Acc.	0.819	0.808	0.813	0.781	0.805
Inherent	-	0.813 / 0.265	0.717 / 0.005	0.586 / 0.023	0.705 / 0.098
Single	0.825 / 0.280	0.813 / 0.266	0.801 / 0.302	0.734 / 0.194	0.793 / 0.261
One Removed	0.778 / 0.256	**0.837** / 0.308	0.736 / 0.293	0.776 / 0.231	0.782 / 0.272
Combined	**0.828** / 0.291	0.828 / 0.294	**0.803** / 0.316	0.762 / 0.227	0.805 / 0.282
RandomSHAP	0.801 / 0.284	0.807 / 0.300	0.766 / 0.322	0.784 / 0.250	0.789 / 0.289
GuidedSHAP	0.826 / 0.291	0.819 / 0.290	0.790 / 0.313	0.765 / 0.235	0.800 / 0.282
RandomLIME	0.809 / **0.295**	0.815 / **0.310**	0.776 / **0.335**	**0.793** / **0.258**	0.798 / **0.299**
GuidedLIME	0.780 / 0.259	0.830 / **0.310**	0.742 / 0.296	0.776 / 0.233	0.782 / 0.274
`MILLI`	0.823 / 0.283	0.827 / 0.307	0.794 / 0.307	0.790 / 0.239	**0.808** / 0.284

TABLE 4.3: *4-MNIST-Bags* interpretability NDCG@n / AOPCR results. `Graph` takes four times as long for a single model pass than the other models, so calculating its AOPCR results on this dataset was infeasible (see Section 4.4.1). For all of the interpretability methods, the standard error of the mean was 0.01 or less.

Methods	`Embedding`	`Instance`	`Attention`	`Graph`	Overall
Model Acc.	0.971	0.974	0.967	0.966	0.970
Inherent	-	0.723 / 0.136	0.750 / 0.002	0.419 / -	0.630 / 0.069
Single	0.722 / 0.138	0.723 / 0.137	0.778 / 0.164	0.761 / -	0.746 / 0.146
One Removed	0.811 / 0.187	0.810 / 0.186	0.809 / 0.143	0.786 / -	0.804 / 0.172
Combined	0.775 / 0.184	0.775 / 0.185	0.816 / **0.183**	0.804 / -	0.792 / 0.184
RandomSHAP	0.813 / 0.186	0.809 / 0.185	0.825 / 0.178	0.828 / -	0.819 / 0.183
GuidedSHAP	0.773 / 0.187	0.773 / 0.188	0.816 / **0.183**	0.805 / -	0.792 / **0.186**
RandomLIME	0.828 / 0.189	0.825 / **0.189**	0.841 / 0.179	0.838 / -	0.833 / **0.186**
GuidedLIME	0.760 / 0.189	0.756 / 0.187	0.785 / 0.145	0.776 / -	0.769 / 0.174
`MILLI`	**0.947** / **0.190**	**0.943** / **0.189**	**0.917** / 0.181	**0.959** / -	**0.942** / **0.186**

TABLE 4.4: CRC interpretability NDCG@n results. As this dataset has much larger bag sizes (264 instances per bag on average), it is infeasible to compute its AOPCR results (see Section 4.4.1). For all of the interpretability methods, the standard error of the mean was 0.02 or less.

Methods	`Embedding`	`Instance`	`Attention`	`Graph`	Overall
Model Acc.	0.795	0.795	0.830	0.770	0.797
Inherent	-	**0.845**	0.692	0.684	0.740
Single	**0.815**	**0.845**	**0.847**	0.786	0.823
One Removed	0.698	0.682	0.701	0.815	0.724
Combined	**0.815**	**0.845**	0.846	0.803	**0.827**
RandomSHAP	0.695	0.690	0.717	0.703	0.701
GuidedSHAP	**0.815**	**0.845**	0.846	0.804	**0.827**
RandomLIME	0.695	0.702	0.716	0.813	0.731
GuidedLIME	0.687	0.699	0.701	**0.818**	0.726
`MILLI`	0.753	0.800	0.810	0.780	0.786

(including the results on the classical MIL datasets, see Section 4.5.2), we find that Combined, GuidedSHAP, RandomLIME, and `MILLI` are the best-performing methods. However, the overall difference in performance between the methods is much less than what we observe for the NDCG@n metric. For the *4-MNIST-Bags* dataset, the difference in performance of `MILLI` on NDCG@n vs AOPCR is due to the difference in weighting between the metrics: `MILLI` achieves a better ordering for the most important instances (outperforms other methods on NDCG@n), but gives the same ordering as other

methods for less important instances (equal performance on AOPCR). In the majority of cases, all of our proposed model-agnostic methods outperform the inherent interpretability methods and are relatively consistent in performance across all models. For the SIVAL dataset, the independent-instance methods perform well, which is expected as the instances are indeed independent. However, on the *4-MNIST-Bags* dataset, `MILLI` excels as it samples informative coalitions that capture the instance interactions. On the CRC dataset, methods that can isolate individual instances, (i.e., the Single, Combined, and GuidedSHAP methods) perform well due to instance independence in this dataset. Furthermore, if we consider the witness rate (WR; the proportion of key instances in each bag; Carbonneau et al., 2018) of the datasets, we find that the CRC dataset has a higher WR (27.47%) than SIVAL (15.28%) and *4-MNIST-Bags* (8.04%). This means, that with larger coalitions, it becomes more difficult to isolate the contributions of individual instances, which is why the One Removed and Random sampling methods struggle.

4.5.2 Classic Multiple Instance Learning Dataset Results

In this section, we detail our additional results on the Musk, Tiger, Elephant, and Fox classical MIL datasets. Although these datasets do not have instance labels, since they are widely used MIL datasets, it is useful to see how our interpretability methods perform on them. Each of these datasets is instance-independent, and as they have a low number of instances per bag, we adapted our local surrogate methods to allow sampling with repeats (otherwise we cannot sample enough coalitions). In our previous experiments, we restricted the local surrogate methods to sampling without repeats, which was not an issue with the larger bag sizes in the SIVAL, *4-MNIST-Bags*, and CRC datasets. For each of these classic datasets, we used the same model hyperparameters that we used for SIVAL (see Appendix A.2), i.e., we did not retune the training parameters or model architectures. However, we did tune the parameters for the interpretability methods, as discussed in Section 4.4.4. We give the interpretability results as well as the model performance for Musk (Table 4.5), Tiger (Table 4.6), Elephant (Table 4.7), and Fox (Table 4.8) below.

We find that there is very little difference in interpretability performance for each of our proposed methods across these four datasets. This is to be expected, as the instances are independent, therefore the independent-instance methods, as well as the local surrogate methods, are able to identify the important instances, i.e., there is little to be gained by sampling coalitions when each instance can be understood in isolation. However, this reinforces the applicability of all of our proposed methods. We note that the `Attention` and `Graph` perform poorly in these experiments — this is because they are unable to condition their outputs on a specific class (i.e., they can only answer *which* questions, not *what* questions). We also note that the performance is much worse on the Fox

TABLE 4.5: Musk interpretability AOPCR results. For all of the interpretability methods, the standard error of the mean was 0.015 or less.

Methods	Embedding	Instance	Attention	Graph	Overall
Model Acc.	0.871	0.836	0.793	0.793	0.823
Inherent	-	**0.108**	0.002	0.003	0.036
Single	0.123	**0.108**	0.113	**0.090**	0.108
One Removed	0.108	0.107	0.088	0.076	0.095
Combined	**0.124**	0.107	**0.116**	0.088	**0.109**
RandomSHAP	0.115	0.104	0.110	0.077	0.102
GuidedSHAP	0.122	0.106	**0.116**	0.084	0.107
RandomLIME	0.113	0.107	0.110	0.080	0.102
GuidedLIME	0.117	0.106	0.102	0.077	0.101
MILLI	0.117	0.105	0.109	0.084	0.104

TABLE 4.6: Tiger interpretability AOPCR results. For all of the interpretability methods, the standard error of the mean was 0.005 or less.

Methods	Embedding	Instance	Attention	Graph	Overall
Model Acc.	0.827	0.807	0.807	0.800	0.810
Inherent	-	0.124	0.001	0.000	0.042
Single	**0.132**	0.125	0.120	**0.082**	0.115
One Removed	0.123	**0.127**	0.114	0.081	0.111
Combined	0.130	**0.127**	**0.124**	**0.082**	**0.116**
RandomSHAP	0.124	0.122	0.121	0.080	0.112
GuidedSHAP	0.130	0.125	0.121	**0.082**	0.115
RandomLIME	0.128	0.125	0.119	0.081	0.113
GuidedLIME	0.129	0.125	0.122	**0.082**	0.115
MILLI	0.128	0.125	0.120	0.081	0.114

dataset. Here, the underlying MIL models perform poorly, so it is unsurprising that the interpretability methods also perform poorly.

4.6 Discussion

The different sampling approaches used in this work have distinct advantages. When there are interactions between instances, RandomSHAP and RandomLIME are better than GuidedSHAP and GuidedLIME as they form larger coalitions that are more likely to capture the instance interactions. However, for the CRC dataset, where there are a

TABLE 4.7: Elephant interpretability AOPCR results. For all of the interpretability methods, the standard error of the mean was 0.006 or less.

Methods	Embedding	Instance	Attention	Graph	Overall
Model Acc.	0.857	0.863	0.867	0.853	0.860
Inherent	-	0.130	0.000	0.000	0.043
Single	**0.127**	**0.131**	**0.127**	0.105	**0.122**
One Removed	0.117	0.129	0.105	0.103	0.114
Combined	0.126	0.130	**0.127**	**0.106**	**0.122**
RandomSHAP	0.124	0.126	0.119	0.102	0.118
GuidedSHAP	**0.127**	0.130	0.124	0.105	**0.122**
RandomLIME	0.124	0.128	0.121	0.104	0.119
GuidedLIME	0.123	0.130	0.118	0.104	0.119
MILLI	0.124	0.128	0.121	0.103	0.119

TABLE 4.8: Fox interpretability AOPCR results. For all of the interpretability methods, the standard error of the mean was 0.002 or less.

Methods	Embedding	Instance	Attention	Graph	Overall
Model Acc.	0.600	0.610	0.620	0.580	0.603
Inherent	-	**0.050**	0.001	0.000	0.017
Single	0.044	0.049	0.041	0.021	0.039
One Removed	0.043	0.049	0.040	0.021	0.038
Combined	0.044	**0.050**	0.041	0.021	0.039
RandomSHAP	0.044	0.049	0.041	0.021	0.039
GuidedSHAP	**0.045**	**0.050**	**0.042**	0.021	**0.040**
RandomLIME	0.044	**0.050**	0.041	0.021	0.039
GuidedLIME	0.044	**0.050**	0.041	0.021	0.039
MILLI	0.043	0.049	0.041	**0.022**	0.039

large number of independent instances and a high WR, RandomSHAP, RandomLIME and GuidedLIME struggle as they cannot form small coalitions. MILLI performs relatively well across all datasets as the size of its sampled coalitions can be adapted depending on the dataset, demonstrating its generalisability. However, it is still outperformed by GuidedSHAP on the CRC dataset. One possible explanation for this is that MILLI is limited to sampling only smaller or larger coalitions, whereas GuidedSHAP samples both large and small coalitions. One approach for improving MILLI would be to incorporate paired sampling (Covert and Lee, 2021), where for every sampled coalition $\mathbf{v}$, we also sample its complement coalition $\mathbf{1} - \mathbf{v}$. Following the advice of Carbonneau et al. (2018), further MIL studies on more complex datasets, such

as Pascal VOC (Everingham et al., 2010), could also be insightful for evaluating MIL interpretability methods. It would also be beneficial to examine if these techniques are applicable to MIL domains beyond classification, e.g. MIL regression as in Wang et al. (2020a).

4.6.1 Interpretability Outputs

Below we visualise the interpretability outputs for SIVAL (Section 4.6.1.1), *4-MNIST-Bags* (Section 4.6.1.2), and CRC (Section 4.6.1.3).

4.6.1.1 SIVAL Interpretability Outputs

Interpretability outputs for the SIVAL dataset are created by ranking the instances in a bag according to some interpretability function (i.e., by using `MILLI`), and then selecting the top n, where n is the known number of key instances in the bag. Then, the corresponding segment for each instance is weighted by the function $log_2(i+1)^{-1}$, where i is the instance's position in the ranking (this is the same scaling used in NDCG@n). The other instances all received a weighting of zero. The brightness of each segment in the output image then corresponds with its relative importance according to the interpretability method. Figures 4.3 to 4.6 give example SIVAL interpretations.

FIGURE 4.3: The interpretability output for two examples of the *cokecan* class in the SIVAL dataset. The first column shows the original images, the second column the ground truth segments that contain the object, and the final column is the output from `MILLI`. The interpretations show the model is relying heavily on red segments as it also picks up on the red in the background as being important.

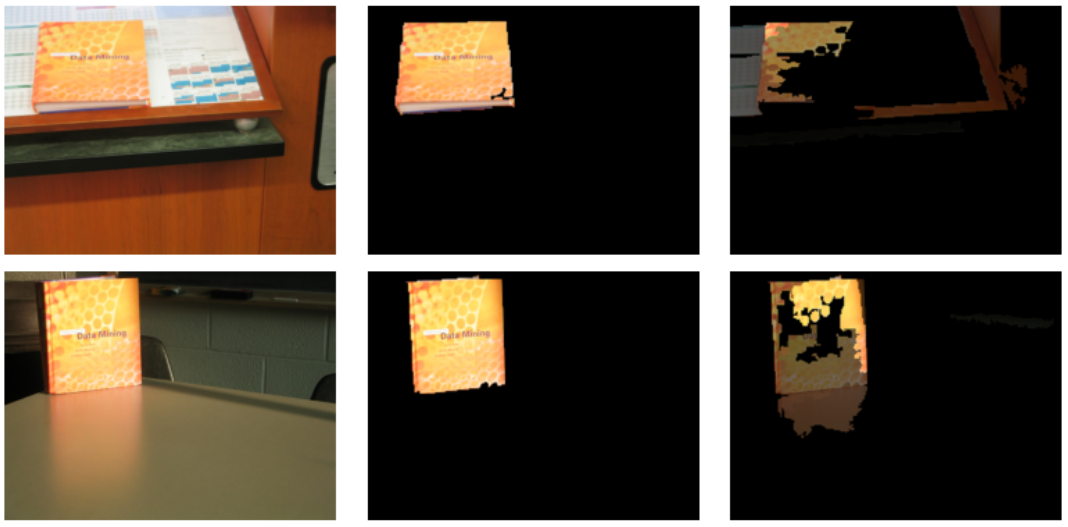

FIGURE 4.4: The interpretability output for two examples of the *dataminingbook* class in the SIVAL dataset. Again, the interpretations show the model is relying heavily on the colour orange as an indicator of the object's location, as it also picks up on the similar colours in the background, including the reflection of the book in the table.

FIGURE 4.5: The interpretability output for two examples of the *goldmedal* class in the SIVAL dataset. Here, the interpretations show the model is ignoring the gold medal itself, and instead focusing on the red ribbon, i.e., if it were shown the medal without the ribbon, it would likely misclassify it.

4.6.1.2 *4-MNIST-Bags* Interpretability Outputs

To demonstrate how `MILLI` captures instance interactions, and to show the limitations of existing methods that cannot condition their output for a particular class, we compare the `MILLI` interpretations with the attention interpretations on the *4-MNIST-Bags* dataset (Figure 4.7). As this bag contains an **8** and a **9**, the correct label for it is class three. Both the `MILLI` and attention methods have identified the **8** and **9**

FIGURE 4.6: The interpretability output for two examples of the *wd40can* class in the SIVAL dataset. Here, the interpretations show the model is predominantly focusing on the strong yellow colour at the top of the can and sometimes picks out the letters and red cap. The rest of the bottle is largely ignored, meaning if the top half were obscured, the model would likely misclassify it.

instances as key instances, i.e., they have both answered the *which* question. However, only `MILLI` correctly identifies that the **8** refutes class two and that the **9** refutes class one, answering the *what* question, something that the attention values do not do. Additional examples for other positive classes are given in Figure 4.8 and Figure 4.9.

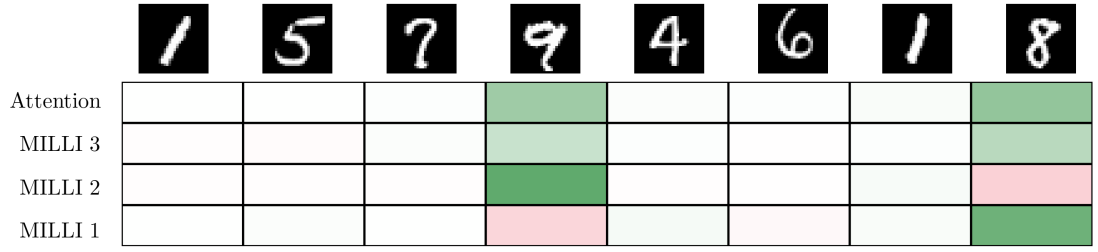

FIGURE 4.7: An example of the identification of key instances for the *4-MNIST-Bags* experiment. The top row shows the attention value for each instance, and the bottom three rows show the output of `MILLI` for classes three, two, and one respectively (green = support, red = refute).

4.6.1.3 CRC Interpretability Outputs

For the CRC interpretability outputs, the important patches are found by using an interpretability method to output the top n patches that support a particular class, where n is the number of known important instances for that class. We then highlight these patches while dimming the other patches to produce an interpretation of the model's decision-making. As the ground-truth labelling was not exhaustive, patches may contain epithelial nuclei without being labelled as such. By using `MILLI`, it is

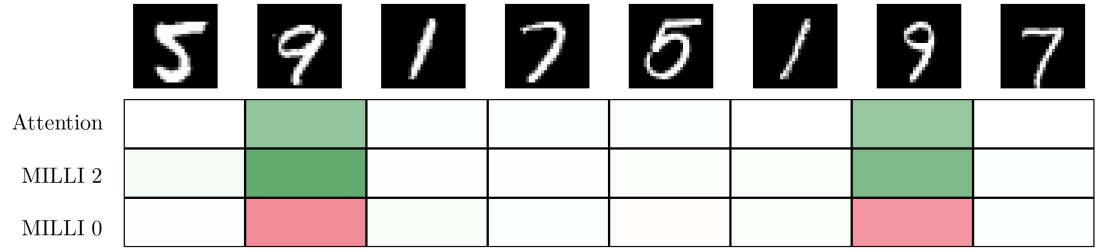

FIGURE 4.8: The interpretability output for a class two bag on the *4-MNIST-Bags* experiment. The top row shows the attention value for each instance, and the bottom two rows show the output of `MILLI` for classes two and zero respectively. We can see that both Attention and `MILLI` have identified the **9**s as important, but only `MILLI` is able to also say that the **9**s support class two and refute class zero.

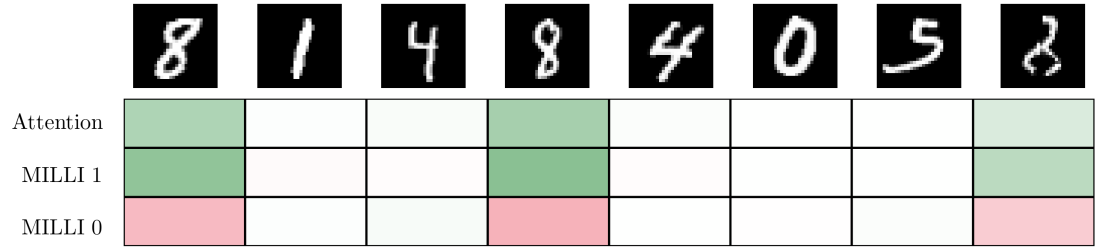

FIGURE 4.9: The interpretability output for a class one bag on the *4-MNIST-Bags* experiment. We can see that both Attention and `MILLI` have identified the **8**s as important, but only `MILLI` is able to also say that the **8**s support class one and refute class zero. Also, note that the colours are less strong for the final **8** — the digit is partially obscured, so the model is less confident.

possible to identify the patches used for decision-making, therefore a further investigation into the types of nuclei in these patches would reveal more information about how the model makes its decisions and potentially identify missed labels in the dataset; something that would not be possible without interpretability. Example outputs are given in Figures 4.10 to 4.13.

4.6.2 Sample Size Experiments

LIME, SHAP, and `MILLI` all require the number of sampled coalitions to be selected. A greater number of coalitions means there is more data to fit the surrogate model on, therefore it is more likely to be faithful to the underlying model. However, the more samples that are taken, the more expensive the computation. We give results for `Embedding` (Figure 4.14), `Instance` (Figure 4.15), `Attention` (Figure 4.16), and `Graph` (Figure 4.17).

`MILLI` and GuidedSHAP are the most consistent in terms of performance and efficiency, and we find the trends are relatively consistent across all the models. `MILLI` is particularly effective on the *4-MNIST-Bags* dataset. One considerable difference is the performance of RandomLIME and GuidedLIME on the CRC dataset for `Graph` — for all

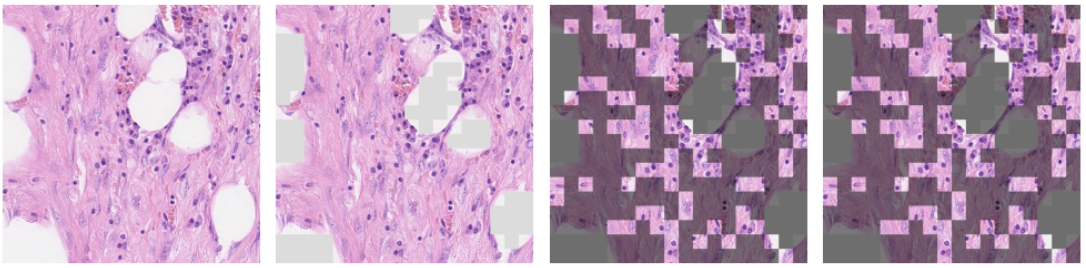

FIGURE 4.10: The interpretability output for an image in the CRC dataset. From left to right, the figure shows the original image, the image with background patches removed, the ground truth patches for the image's label, and the interpretability output from `MILLI`. In this example, the image is class zero (i.e., non-epithelial), meaning the highlighted patches contain mostly fibroblast and inflammatory nuclei. As shown here, `MILLI` has identified that the model is using the same patches as the ground truth to make a decision, indicating that the model has learnt to correctly identify fibroblast and inflammatory nuclei as supporting instances for non-epithelial images.

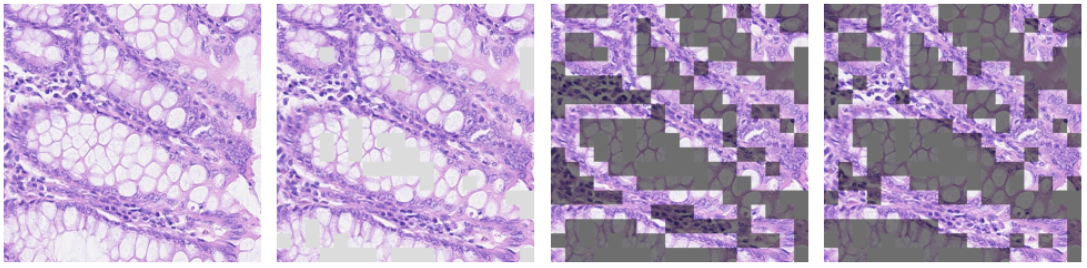

FIGURE 4.11: The interpretability output for a positive image in the CRC dataset. In this example, the ground truth patches all contain at least one epithelial nuclei. `MILLI` has identified that the model is using most of the same patches as the ground truth to make a decision, indicating that the model has learnt to correctly identify epithelial nuclei as supporting class one.

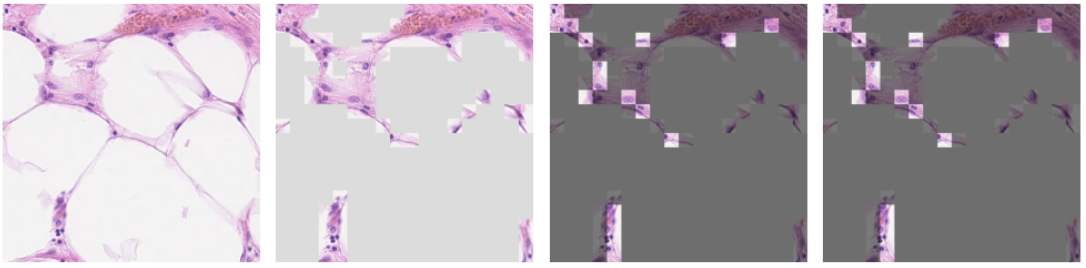

FIGURE 4.12: The interpretability output for a negative image in the CRC dataset. In this example, the patches are sparse — there are a lot of background patches that are removed, and the key patches are spread out (as opposed to being connected as in the previous examples). Again, `MILLI` is able to correctly identify the patches that support the negative class.

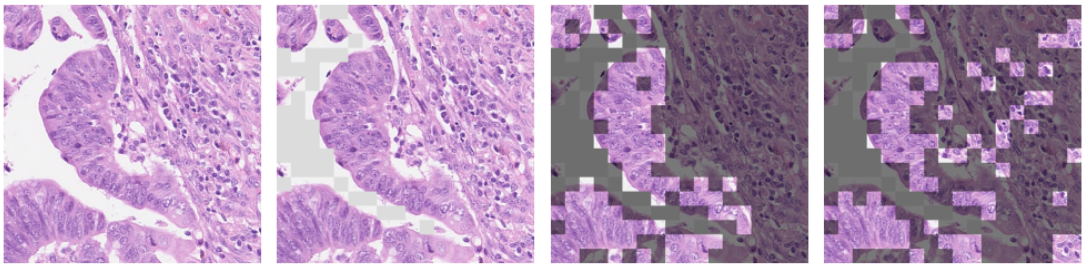

FIGURE 4.13: The interpretability output for a positive image in the CRC dataset. In this example, `MILLI` has identified some of the ground truth patches, but also additional patches that are not labelled as epithelial in the ground truth labelling.

other models, both LIME methods perform poorly on the CRC dataset. Further investigation is required to understand why this happens. However, the LIME methods are still outperformed by `MILLI` and GuidedSHAP on the CRC dataset, even for the `Graph` model. For all methods, it is a case of diminishing returns, where additional samples only lead to a small increase in performance.

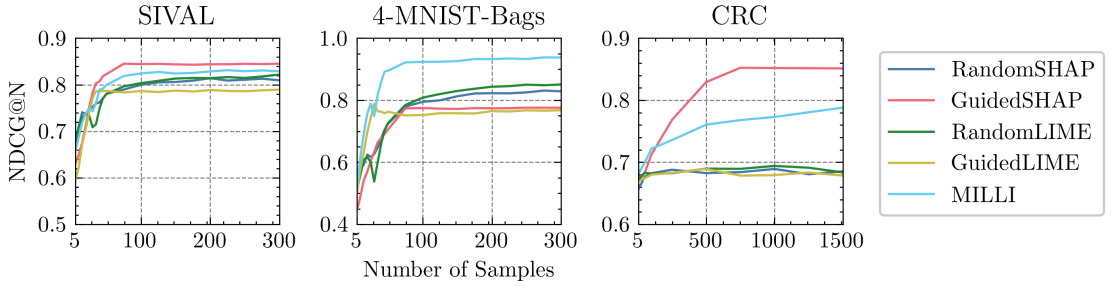

FIGURE 4.14: The effect of sample size on interpretability performance for `Embedding`.

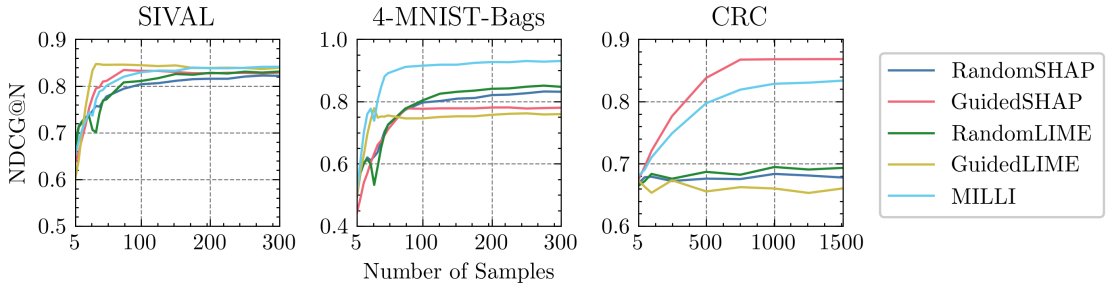

FIGURE 4.15: The effect of sample size on interpretability performance for `Instance`.

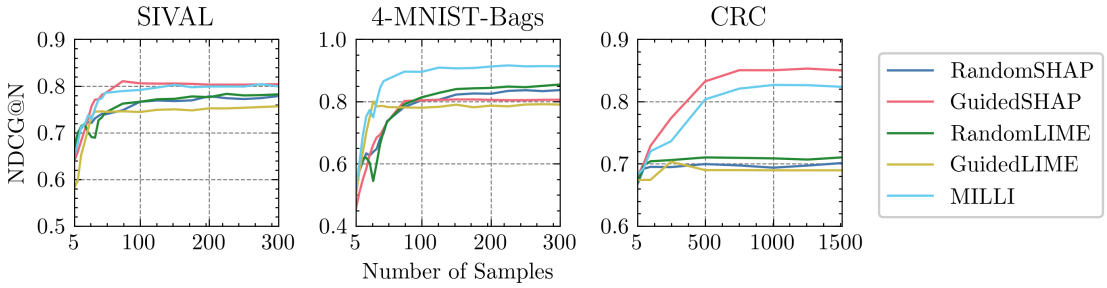

FIGURE 4.16: The effect of sample size on interpretability performance for `Attention`.

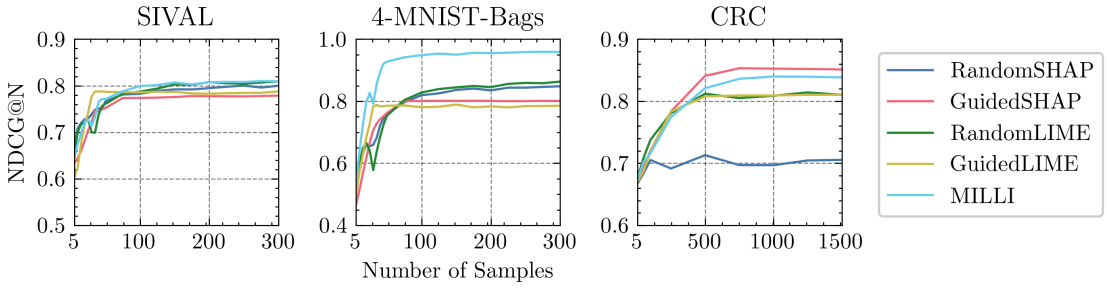

FIGURE 4.17: The effect of sample size on interpretability performance for `Graph`.

4.7 Conclusion

In this chapter, we have discussed the process of model-agnostic interpretability for MIL. Along with defining the requirements for MIL interpretability, we have presented our own approaches and compared them to existing inherently interpretable MIL models. By analysing the methods across several datasets, we have shown that independent-instance methods can be effective, but local surrogate methods are required when there are interactions between the instances. All of our proposed methods are more effective than existing inherently interpretable models and are able to not only identify *which* are the key instances, but also say *what* classes they support and refute.

In this chapter, the effect of changing the patch size for the CRC dataset was not investigated. This has implications for interpretability: a small patch size improves the resolution at which interpretations are made (i.e., fewer cells per patch), but if the patches become too small, the MIL problem becomes impossible to learn as there is not enough information in each patch. Furthermore, this chapter only considered classification problems, so further work is required to investigate MIL interpretability in other styles of problems.

As such, in the next chapter, we continue working with computer vision problems, but transition to the new application domain of Earth Observation (EO) for climate change monitoring. In this domain, we investigate the effect of changing the MIL patch size, extend the approach to support multiple resolutions, and how interpretability can be achieved for MIL regression problems.

Chapter 5

Scene-to-Patch Earth Observation with Multi-Resolution Multiple Instance Learning

This chapter presents a combination of two pieces of work. The first was published in the NeurIPS Workshop: Tackling Climate Change with Machine Learning in 2022, and the second was published in the Journal of Environmental Data Science in 2023. This work was supported by Ying-Jung Deweese (Georgia Institute of Technology) who provided insight on the climate background and manuscript feedback — all technical development and design was done by myself. The content is mostly unchanged from its original format: elements of the original appendix have been brought into the main body, certain sections have been slightly reworded, and some terminology and notation has been modified for consistency with the rest of this thesis.

Workshop Paper

Joseph Early, Ying-Jung Deweese, Christine Evers, and Sarvapali Ramchurn. Scene-to-patch Earth observation: Multiple instance learning for land cover classification. *NeurIPS Workshop: Tackling Climate Change with Machine Learning*, 2022b. URL https://arxiv.org/abs/2211.08247

GitHub: https://github.com/JAEarly/MIL-Land-Cover-Classification

Journal Paper

Joseph Early, Ying-Jung Chen Deweese, Christine Evers, and Sarvapali Ramchurn. Extending scene-to-patch models: Multi-resolution multiple instance learning for Earth observation. *Environmental Data Science*, 2, 2023. URL https://doi.org/10.1017/eds.2023.30

GitHub: https://github.com/JAEarly/MIL-Multires-EO

5.1 Introduction

To tackle critical problems such as climate change and Natural Disaster Response (NDR), it is essential to collect information and monitor ongoing changes in Earth systems (Oddo and Bolten, 2019). Remote Sensing (RS) for Earth Observation (EO) is the process of acquiring data using instruments equipped with sensors, e.g., satellites or unmanned aerial vehicles (UAVs). Due to ever-expanding volumes of EO data, Machine Learning (ML) has been widely used in automated RS for a range of applications. However, curation of the large labelled EO datasets required for ML is expensive — accurately labelling the data takes a lot of time and often requires specific domain knowledge. This results in bottlenecks and concerns about a lack of suitable EO datasets for ML (CCAI, 2022; Hoeser and Kuenzer, 2020). Beyond training, using ML models for real-time monitoring means the model size has to be restricted: models that are too large will take too long to run (as well as being more expensive).

In this work, we focus on two EO RS applications: Land Cover Classification (LCC) and NDR. LCC is an important task in EO and can be applied to agricultural health & yield monitoring, deforestation, sustainable development, urban planning, and water availability (Demir et al., 2018; Hoeser et al., 2020). Natural disasters (floods, earthquakes, etc.) are dynamic processes that require rapid response in order to save assets and lives. ML approaches can provide near real-time monitoring and change detection for NDR, as well as mitigate supply chain issues (Oddo and Bolten, 2019). A common objective in both LCC and NDR is image segmentation (Hoeser et al., 2020; Munawar et al., 2021). This involves separating images into different classes (e.g., urban land, forest land, agricultural land, etc.) or objects (e.g., houses, vehicles, people, etc.). Training ML models to perform automated segmentation typically requires an EO dataset that has already been segmented, i.e., each image has a corresponding set of annotations that mark out the different objects or class regions. The annotation process often has to be done by hand and is very expensive and time-consuming. For example, each image in the FloodNet dataset (used in this work) took approximately one hour to annotate (Rahnemoonfar et al., 2021).

In this chapter, we propose a new approach to segmentation for EO, in which cost- and time-intensive segmentation labels are not required. Instead, only scene-level summaries are needed (which are much quicker to curate). This is achieved by reframing segmentation as a Multiple Instance Learning (MIL) regression problem. To solve this problem, we propose Scene-to-Patch (S2P) MIL models that produce segmented outputs without requiring segmentation labels and are also able to preserve high-resolution data during training and inference. They are effectively able to transform the low-resolution scene labels used in training into high-resolution patch predictions. Furthermore, we explore the use of single-resolution and multi-resolution

S2P approaches for both satellite and aerial imagery. Specifically, our contributions are as follows:

1. We propose S2P models for both single-resolution and multi-resolution inputs and outputs.
2. We apply our novel S2P approach to LCC (satellite data) and disaster response monitoring (aerial imagery), comparing it to three baselines.
3. We investigate how changing the configuration of our approach affects its performance.
4. We show how our approach is inherently interpretable at multiple resolutions, allowing pixel-level segmentation of EO images without requiring pixel-level labels during training.

The rest of this chapter is laid out as follows. Section 5.2 details the background literature and related work. Section 5.3 introduces our S2P approach and its multi-resolution extension. Our experiments are presented in Section 5.4, with a further discussion in Section 5.5. Section 5.6 concludes.

5.2 Background and Related Work

In this section, we provide the background literature to our work, covering LCC, NDR, and their existing ML solutions. We also discuss MIL and its multi-resolution extensions.

LCC In EO, ML can be applied to monitoring and forecasting land surface patterns, involving ecological, hydrological, agricultural, and socioeconomic domains (Liu et al., 2017). For example, satellite images can be used to track carbon sequestration and emission sources, which is useful for monitoring greenhouse gas (GHG) levels (Rolnick et al., 2022). The way land is used is both impacted by and contributes to climate change — it is estimated that land use is responsible for around a quarter of global GHG emissions, and improved land management could lead to a reduction of about a third of emissions (Rolnick et al., 2022). Furthermore, changes in land use can have significant effects on the carbon balance within ecosystem services that contribute to climate change mitigation (Friedlingstein et al., 2020). In order to work towards UN Net Zero emission targets (Sadhukhan, 2022), there is a need for improved understanding and monitoring of how land is used, i.e., real-time monitoring that facilitates better policy design, planning, and enforcement (Kaack et al., 2022). For example, automated LCC with ML can be used to determine the effect of regulation or incentives to drive better

land use practices (Rolnick et al., 2022), and for monitoring the amount of acreage in use for farmland, allowing the assessment of food security and GHG emissions (Ullah et al., 2022).

NDR The increasing magnitude and frequency of extreme weather events driven by climate change are accelerating the change of suitable land areas for cropland and human settlement (Elsen et al., 2022), and raising the need for timely & effective NDR. With recent advances in EO, RS is a valuable approach in NDR efforts, providing near real-time information to help emergency responders execute their response efforts, plan emergency routes, and identify effective lifesaving strategies. For instance, Sentinel-2 data has been used for flood response (Caballero et al., 2019), synthetic aperture radar (SAR) imagery has been used for mapping landslides (Burrows et al., 2019) & wildfire patterns (Ban et al., 2020), and helicopter & UAV imagery has been used to identify damaged buildings and debris following hurricanes (Pi et al., 2020). RS is only one aspect of utilising automated data processing techniques to facilitate improved NDR; other paradigms include digital twins (Fan et al., 2021) and natural language processing (Zhang et al., 2019).

Existing Approaches A common problem type in both LCC and NDR is image segmentation. Here, the aim is to assign each pixel in an input image to a class, such that different objects or regions in the original image are separated and classified. For example, LCC segmentation typically uses classes such as urban, agricultural, forest, etc. In image segmentation settings, most models require the original training images to be annotated with segmentation labels, i.e., all pixels are labelled with a ground-truth class. There are several existing approaches to image segmentation, such as Fully Convolutional Networks (Long et al., 2015), U-Net (Ronneberger et al., 2015), and Pyramid Networks (Lin et al., 2017). Existing works have applied these or similar approaches to LCC (Karra et al., 2021; Kuo et al., 2018; Rakhlin et al., 2018; Seferbekov et al., 2018; Tong et al., 2020; Wang et al., 2020b); we refer readers to Hoeser and Kuenzer (2020) for a more in-depth review of existing work. For NDR, examples of existing work include wildfire segmentation with U-Net (Khryashchev and Larionov, 2020), and flooding segmentation using Multi3Net (Rudner et al., 2019).

MIL In conventional supervised learning, each piece of data is given a label. However, in MIL, data are grouped into bags of instances, and only the bags are labelled, not the instances (Carbonneau et al., 2018). This reduces the burden of labelling, as only the bags need to be labelled, not every instance. In this work, we utilise MIL neural networks (Wang et al., 2018). Extensions such as attention (Ilse et al., 2018), graph neural networks (Tu et al., 2019), and long short-term memory (Early et al., 2022a; Wang et al., 2020a, Chapter 6) exist, but these are not explored in this work. MIL has

previously been used in EO data, for example fusing panchromatic and multi-spectral images (Liu et al., 2017), settlement extraction (Vatsavai et al., 2013), landslide mapping (Zhang et al., 2020), crop yield prediction (Wang et al., 2012), and scene classification (Wang et al., 2022). However, to the best of the authors' knowledge, this work is the first to study the use of MIL for generic multi-class LCC and NDR.

Multi-Resolution MIL When using MIL in imaging applications, patches are extracted from the original images to form bags of instances. As part of this patch extraction process, it is necessary to choose the number and size of patches, which determines the effective resolution at which the MIL model is operating (and affects the overall performance of the model). There is an inherent trade-off when choosing the patch size: patches must be large enough to capture meaningful information, but small enough such that they can be processed with ML (or that downsampling does not lead to a loss of detail). To overcome this trade-off, prior research has investigated the use of multi-resolution approaches, where several different patch sizes are used (if only a single patch size is used, the model is considered to be operating at a single resolution). Existing multi-resolution MIL approaches typically focus on medical imaging (Hashimoto et al., 2020; Li et al., 2021a; Li and Zhang, 2020; Marini et al., 2021) as opposed to EO data (we discuss these methods further in Section 5.3.3). Non-MIL multi-resolution approaches have been applied in EO domains such as ship detection (Wang et al., 2019a) and LCC (Robinson et al., 2019), but to the best of the authors' knowledge, we are the first to propose the use of multi-resolution MIL methods for generic EO. In the next section, we introduce our MIL S2P method for both single and multiple resolutions.

5.3 Methodology

In this section, we present our novel S2P methodology. First, in Sections 5.3.1 and 5.3.2, we introduce our single resolution S2P approach. We then show how this can be extended to multi-resolution settings, explaining our novel model architecture (Section 5.3.3), and how it combines information from multiple resolutions (Section 5.3.4).

5.3.1 Scene-to-Patch Overview

Predictions and labelling in computer vision problems can be split into three tiers of operation:

1. **Scene Level**: The image is considered as a whole. Classic convolutional neural networks (CNNs) are examples of scene-level models, where the entire image is used as input, and each image has a single class label. An EO example of a scene-level approach is brick kiln identification (Lee et al., 2021a).

2. **Patch Level**: Images are split into small patches (typically tens or hundreds of pixels). This is a standard approach in MIL, where a group of patches extracted from the same image form a bag of instances. In most cases, labelling and prediction occur only at the bag (scene) level. However, depending on the dataset and MIL model, labelling and prediction can occur at the patch level.

3. **Pixel Level**: Labelling and predictions are performed on the scale of individual pixels. Segmentation models (e.g., U-Net; Ronneberger et al., 2015) are pixel-level approaches. They typically require pixel-level labels (i.e., segmented images) in order to learn to make pixel-level predictions. A notable exception in EO research is Wang et al. (2020b), where U-Net models are trained from scene-level labels without requiring pixel-level annotations.

With S2P models, both scene- and patch-level predictions can be made whilst only requiring scene-level labels for training. This is achieved by reframing pixel-level segmentation as scene-level regression, where the objective is to predict the coverage proportion of each segmentation class. The motivation for such an approach is that scene-level labels are easy to procure (as they are summaries of the content of an image as opposed to detailed pixel-level annotations), which helps expedite the labelling process. Furthermore, it also reduces the likelihood of label errors, which often occur in EO segmentation datasets (Rakhlin et al., 2018). However, scene-level predictions cannot be used for segmentation. Therefore, it is necessary to have a model that learns from scene-level labels but can produce patch- or pixel-level predictions — we do so by proposing inherently interpretable MIL models. We provide a high-level overview of our S2P approach in Figure 5.1 and discuss our model architecture in more detail in the next section.

5.3.2 Single Resolution Scene-to-Patch Approach

The core S2P approach is based on mi-Net (`Instance`; Wang et al., 2018). This operates on a single input and output resolution, so in the remainder of this work, we call these Single Resolution (SR) models. In Figure 5.2, we give an example S2P SR architecture for the DeepGlobe dataset (Section 5.4.1), which has seven classes. The model takes a bag containing b instance patches as input, where each patch has three channels (red, green, and blue) and is 102 x 102 px.[1] The patches are independently passed through a

[1]Note the number of patches b for S2P is equivalent to the number of instances k in general MIL.

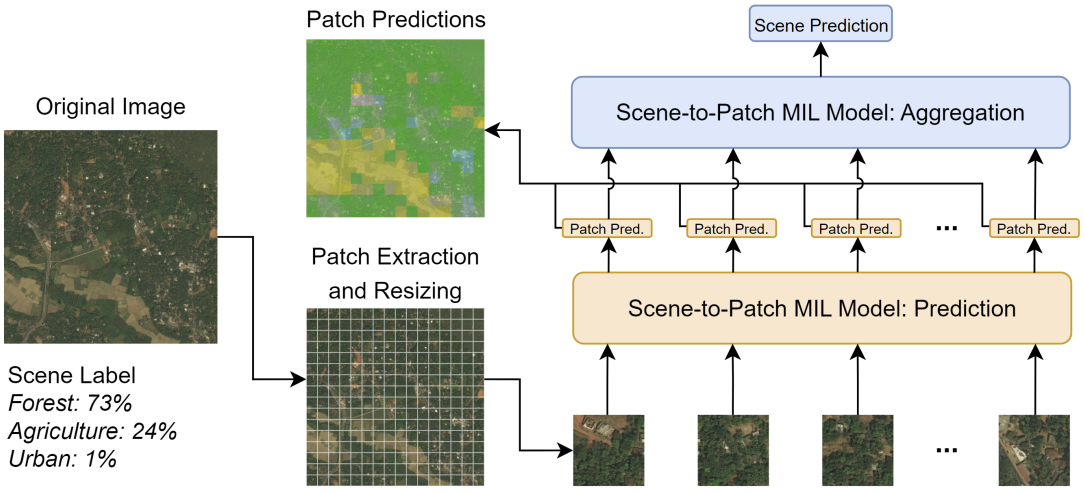

FIGURE 5.1: MIL S2P overview. The model produces both instance (patch) and bag (scene) predictions but only learns from scene-level labels. Example from the DeepGlobe dataset (Section 5.4.1).

Feature Extractor (FE), resulting in b patch embeddings; each of length 128. These patch embeddings are then classified, and the mean of the patch predictions is used as the overall bag prediction.

As predictions are made for each patch as a by-product of making the overall scene-level prediction (i.e., as a form of inherent interpretability), segmentation outputs are produced without the need for a post-hoc process. In contrast, if the comparable `Embedding` approach was used, no patch-level predictions are made, meaning the model does not inherently produce segmentation outputs. As the scene-level predictions are derived directly from the patch-level predictions, the model must learn the differences between patches, and as such produces robust patch-level segmentations. Note these models are trained end-to-end and only learn from scene-level (bag) labels — no patch-level labels are used during training. While patch- or pixel-level labels may be provided for certain datasets in the form of segmentation maps, in this work we explore only learning from scene-level labels but still producing segmentation outputs (primarily because scene-level labels are faster to produce than patch- or pixel-level labels).

Formally, each EO image $\mathbf{X}$ has a corresponding scene-level label $\mathbf{Y}$. Each label is a C-dimensional vector $\mathbf{Y} \in \mathbb{R}^C$, where C is the number of classes in a given dataset. Each element in $\mathbf{Y}$ represents the coverage proportion (a value between 0 and 1) for a different class, i.e., $\mathbf{Y} = \{Y_0, \ldots, Y_{C-1}\}$ such that $\sum_{c=0}^{C-1} Y_c = 1$. A set of b patches $\{\mathbf{x}_1, \ldots, \mathbf{x}_b\}$ is extracted for each image $\mathbf{X}$. The S2P approach makes a prediction for each patch, resulting in patch-level predictions $\{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_b\}$. The mean of these patch predictions is then taken to make an overall scene-level prediction $\hat{\mathbf{Y}} = \frac{1}{b} \sum_{j=1}^{b} \hat{\mathbf{y}}_j$.

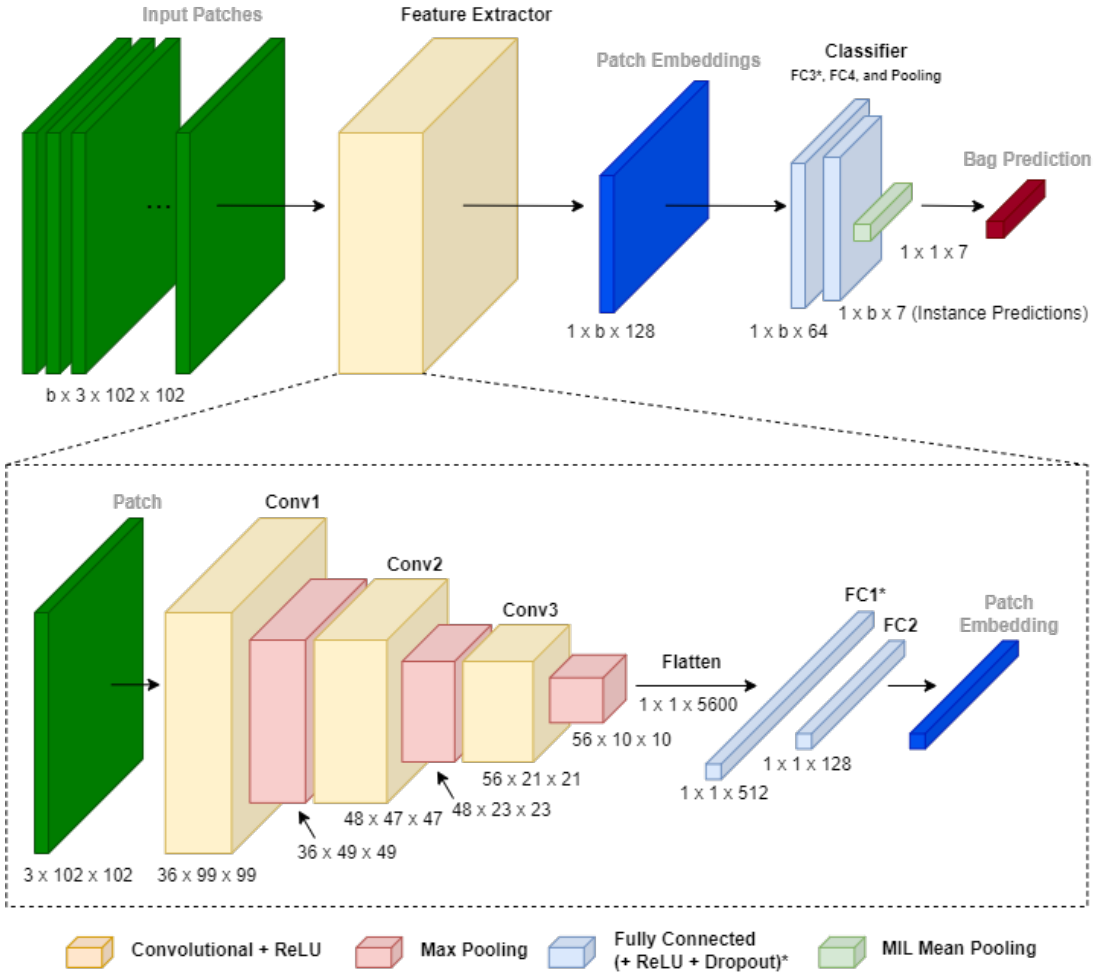

FIGURE 5.2: S2P SR model architecture. Note that some fully connected (FC) layers use ReLU and Dropout (denoted with *) while some do not, and b denotes bag size (number of patches).

The configuration of our S2P model shown in Figure 5.2 uses three convolutional layers in the FE model, but we also experiment with only using two convolution layers (see Appendix B.2 for further details). The S2P models are relatively small (in comparison to baseline architectures such as ResNet18 and U-Net; see Section 5.4.2), which means they are less expensive to train and run. This reduces the GHG emission effect of model development and deployment, which is an increasingly important consideration for the use of ML (Kaack et al., 2022). Furthermore, the S2P approach makes patch predictions inherently — post-hoc interpretability methods such as MIL local interpretations (Early et al., 2022c, Chapter 4) are not required.

Reframing EO segmentation as a regression problem does not necessitate a MIL approach; it can be treated as a traditional supervised regression problem. However, as EO images are often very high resolution, the images would have to be downsampled, and, as such, important data would be lost. With MIL, it is possible to operate at higher resolutions than a purely supervised learning approach. With these SR models, it is

possible to change the resolution at which the model is operating, but the model cannot incorporate multiple resolutions. In the next section, we discuss our extension to S2P models which facilitates the use of multiple resolutions within a single model.

5.3.3 Multi-Resolution Scene-to-Patch Approaches

The objective of our S2P extension is to utilise multiple resolutions within a single model. While separate SR models can be used with different resolutions, this does not allow information to be shared between the resolutions, which may hinder performance. For example, in aerial imagery, larger features such as a house might be easy to classify at lower resolutions (when the entire house fits in a single patch), but higher resolutions might be required to separate similar classes, such as the difference between trees and grass. Furthermore, a multi-resolution approach incorporates some notion of spatial relationships, helping to reduce the noise in high-resolution predictions (e.g., when a patch prediction is different to its neighbours). Combining multiple resolutions into a single model facilitates stronger performance and multi-resolution prediction (which aids interpretability) without the need to train separate models for different resolutions.

Several existing works have investigated multi-resolution MIL models, which we use as inspiration for our multi-resolution S2P models. Most notably, we base our approach on Multi-Scale Domain-Adversarial MIL (MS-DA-MIL; Hashimoto et al., 2020) and Dual Stream MIL (DSMIL; Li et al., 2021a). MS-DA-MIL uses a two-stage approach, where the latter stage operates at multiple resolutions and makes a combined overall prediction. DSMIL also uses a two-stage approach and provides a unique approach to combining feature embeddings extracted at different resolutions. Our novel method differs from these approaches in that it only uses a single stage of training. We extract patches at different resolutions and transform these patches into embeddings at each resolution using separate Feature Extractors (FEs), i.e., distinct convolutional layers for each resolution. In comparison to using a single FE for all resolutions, as done by Marini et al. (2021), this allows better extraction of features specific to each resolution. We propose two different approaches for classification:

1. **Multi-Resolution Single-Out (MRSO)** — This model uses multiple resolutions as input, but only makes predictions at the highest resolution. However, this high-resolution output uses information from all the input resolutions in its decision-making process.

2. **Multi-Resolution Multi-Out (MRMO)** — This model uses multiple-resolution inputs and makes predictions at multiple resolutions. Given $s_n$input resolutions, it makes $s_n + 1$ sets of predictions: one independent set for each input resolution, and one main set, utilising information from all input resolutions (as in the MRSO model).

We give an example of the MRSO/MRMO models in Figure 5.3. Here, we utilise $s_n = 3$ input resolutions: $s = 0$, $s = 1$, and $s = 2$, where each resolution is twice that of the previous. For MRMO, the individual resolution predictions are made independently (i.e., not sharing information between the different resolutions), but the main prediction, $s = m$, utilises information from all input resolutions to make predictions at the $s = 2$ scale. During MRMO training, we optimise the model to minimise the Root Mean Square Error (RMSE) averaged over all four outputs, aiming to achieve strong performance at each independent resolution as well as the combined resolution. MRSO does not make independent predictions at each scale, only the main $s = m$ predictions, meaning it can be trained using standard RMSE. In the next section, we discuss how we combine information across different resolutions for MRSO and MRMO.

5.3.4 Multi-Resolution Multiple Instance Learning Concatenation

In our multi-resolution models MRSO and MRMO, we combine information between different input resolutions to make more accurate predictions. To do so, we use an approach similar to Li et al. (2021a) and partly inspired by Lazebnik et al. (2006). A fundamental part of our approach is how patches are extracted at different resolutions: we alter the grid size, doubling it at each increasing resolution. This means a single patch of size p x p px at resolution $s = 0$ is represented with four patches at resolution $s = 1$, each also of size p x p px. As such, the same area is represented at twice the original resolution, i.e., $2p$ x $2p$ px at $s = 1$ compared to p x p px at $s = 0$. Extending this to $s = 2$ follows the same process, resulting in 16 patches for each patch at $s = 0$, i.e., at four times the original resolution ($4p$ x $4p$ px). This process is visually represented on the left of Figure 5.4. As such, given a bag of size b at $s = 0$, the equivalent bags at $s = 1$ and $s = 2$ are of size $4b$ and $16b$ respectively, as shown in Figure 5.3.

Given this multi-resolution patch extraction process, we then need to combine information across the different resolutions. We do so by using independent FEs (one per resolution, see Figure 5.3), which give a set of patch embeddings, one per patch, for each resolution. Therefore, we will have b embeddings at scale $s = 0$, $4b$ embeddings at $s = 1$, and $16b$ embeddings at $s = 2$. To combine these embeddings, we repeat the $s = 0$ and $s = 1$ embeddings to match the number of embeddings at $s = 2$ (16 repeats per embedding for $s = 0$ and 4 repeats per embedding for $s = 1$). This repetition preserves spatial relationships, such that the embeddings for lower resolutions ($s = 0$ and $s = 1$) cover the same image regions as captured by the higher resolution $s = 2$ embeddings. Given the repeated embeddings, it is now possible to concatenate the embeddings from different resolutions, resulting in an extended embedding for each $s = 2$ embedding that includes information from $s = 0$ and $s = 1$. Note the embeddings extracted at each resolution are the same size ($l = 128$), so the concatenated embeddings are three times as long ($3l = 384$). This process is summarised in Figure 5.4.

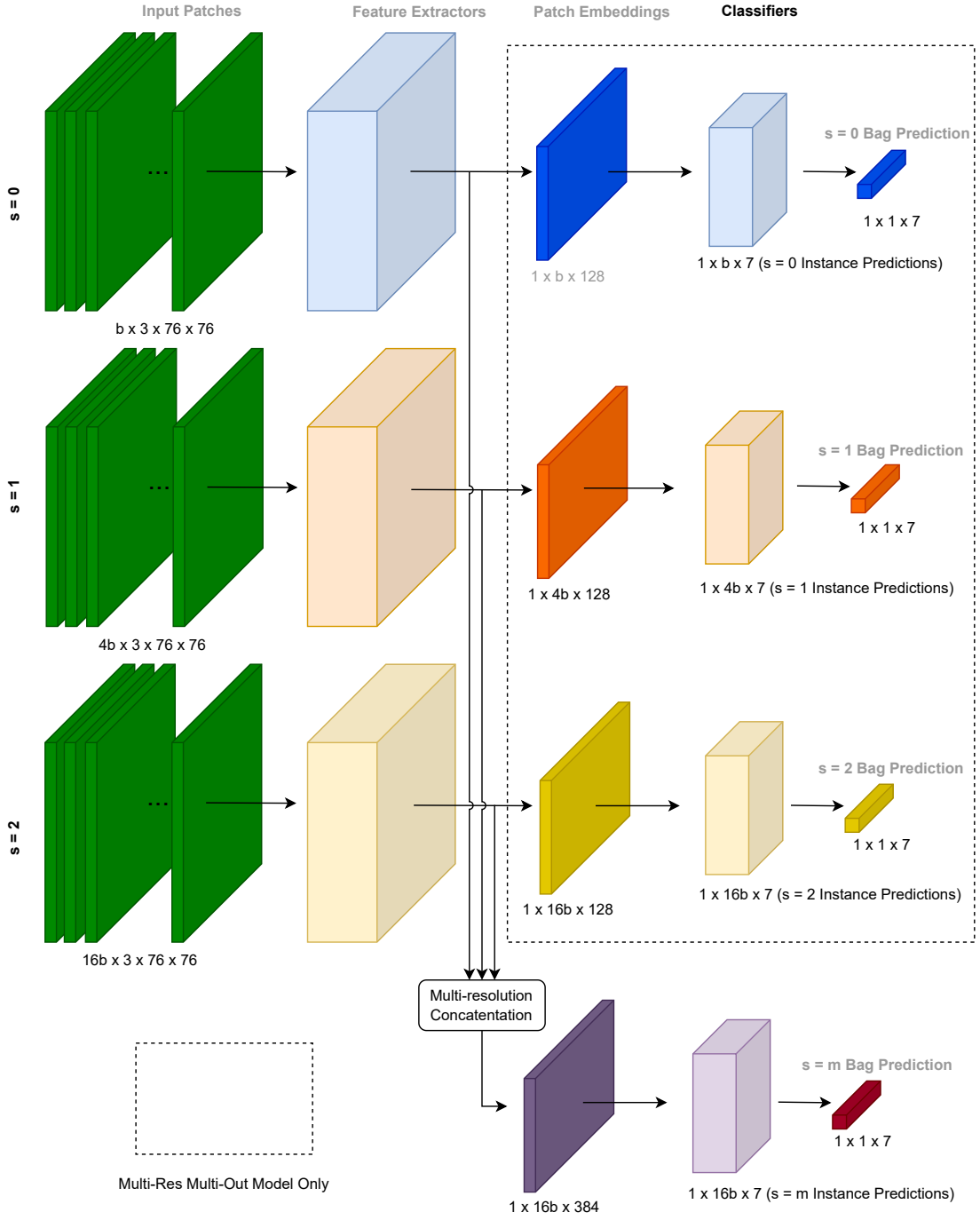

FIGURE 5.3: S2P multi-resolution architecture. The embedding process uses independent FEs (CNN layers, see Figure 5.2), allowing specialised feature extraction for each resolution. The MRMO configuration produces predictions at $s = 0$, $s = 1$, $s = 2$, and $s = m$ resolutions (indicated by the dashed box); MRSO only produces $s = m$ predictions.

5.4 Experiments

In this section, we detail our experiments, first describing the datasets (Section 5.4.1), models (Section 5.4.2), and training procedure (Section 5.4.3). We then provide our

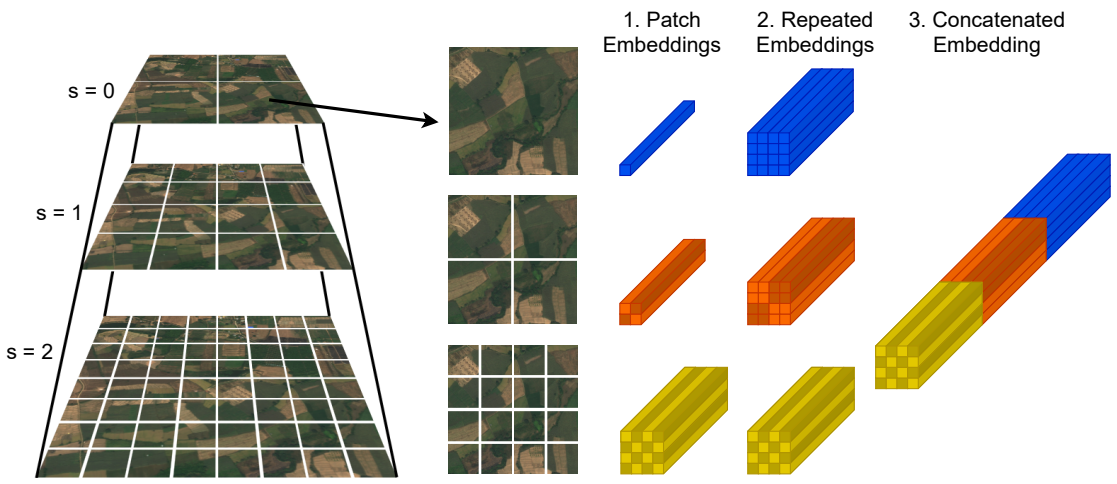

FIGURE 5.4: Multi-resolution patch extraction and concatenation.
Left: Patches are extracted at different resolutions, where each patch at scale $s = 0$ has four corresponding $s = 1$ patches and 16 corresponding $s = 2$ patches.
Right: The $s = 0$ and $s = 1$ embeddings are repeated to match the number of $s = 2$ embeddings, and then concatenated to create multi-resolution embeddings.

results (Section 5.4.4).

5.4.1 Datasets

We conduct experiments on two datasets: DeepGlobe (Demir et al., 2018) and FloodNet (Rahnemoonfar et al., 2021). Both datasets are described in detail below.

5.4.1.1 DeepGlobe

The DeepGlobe-LCC dataset (Demir et al., 2018) consists of 803 satellite images with three channels (red, green, and blue). Each image is 2448 x 2448 pixels with a 50cm pixel resolution. All images were sourced from the WorldView3 satellite covering regions in Thailand, Indonesia, and India. There are also validation and test splits with 171 and 172 images respectively, but as these splits did not include annotation masks, they were not used in this work. The DeepGlobe-LCC dataset is openly available and can be acquired from Kaggle.[2] Below we give further details on the dataset, which are adapted from the Kaggle page.

Each satellite image is paired with a mask image for land cover annotation. Each mask is an image with 7 classes of labels, using colour-coding (RGB) described below. We also give an overview of the class distribution in Figure 5.5.

[2] https://www.kaggle.com/datasets/balraj98/deepglobe-land-cover-classification-dataset

0. **Urban land** (0, 255, 255) — Man-made, built-up areas with human artefacts (ignoring roads which are hard to label).
1. **Agriculture land** (255, 255, 0) — Farms, any planned (i.e., regular) plantation, cropland, orchards, vineyards, nurseries, and ornamental horticultural areas.
2. **Rangeland** (255, 0, 255) — Any non-forest, non-farm, green land, grass.
3. **Forest land** (0, 255, 0) — Any land with x% tree crown density plus clearcuts.
4. **Water** (0, 0, 255) — Rivers, oceans, lakes, wetland, ponds.
5. **Barren land** (255, 255, 255) — Mountain, land, rock, desert, beach, no vegetation.
6. **Unknown** (0, 0, 0) — Clouds and others.

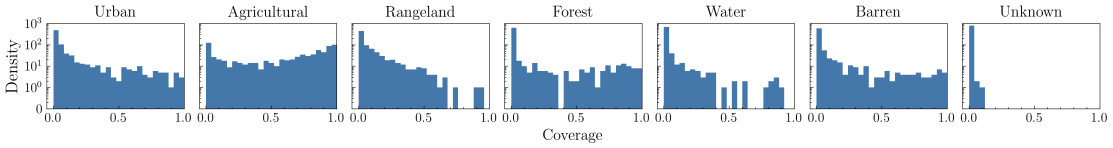

FIGURE 5.5: DeepGlobe Class Distribution. We compute the class coverage for each image in the dataset, then plot a log histogram with 25 bins.

While the DeepGlobe-LCC dataset provides pixel-level annotations, these segmentation labels are *only* used to generate the regression targets for training and for the evaluation of derived patch segmentation, i.e., they are not used during training. However, we would like to stress that these segmentation labels are not strictly required for our approach, i.e., the scene-level regression targets can be created without having to perform segmentation.

We used 5-fold cross-validation rather than the standard 10-fold due to the limited size of the datasets (only 803 images). With this configuration, each fold had an 80/10/10 split for train/validation/test. We normalised the images by the dataset mean (0.4082, 0.3791, 0.2816) and standard deviation (0.06722, 0.04668, 0.04768). No other data augmentation was used.

5.4.1.2 FloodNet

To assess our S2P approach on EO data captured with aerial imagery, we use the FloodNet dataset (Rahnemoonfar et al., 2021). This dataset was originally used as a competition dataset as part of EARTHVISION 2021. It was created by Bina Lab (a computer vision and RS laboratory at the University of Maryland, Baltimore County). This work used the openly available data provided on GitHub.[3]

[3] https://github.com/BinaLab/FloodNet-Supervised_v1.0

FloodNet consists of 2343 high-resolution (4000 x 3000 px) images of Ford Bend County in Texas, captured between August 30th and September 4th 2017 after Hurricane Harvey. Images have three channels (red, green, and blue), and were taken with DJI Mavic Pro quadcopters at 200 feet above ground level (flown by emergency responders as part of the disaster response process). The aim is to capture the post-disaster effects, notably flooding.

Each aerial image is paired with a mask image. Each mask is a single-channel image with 10 classes of labels, where pixel values from 0 to 9 indicate the pixel class (described below). We also give an overview of the class distribution in Figure 5.6.

0. **Background** — Regions that do not fall into any of the other classes.
1. **Building Flooded** — Man-made structures where at least one side is touching flood water.
2. **Building Non-flooded** — Man-made structures where no sides are touching flood water.
3. **Road Flooded** — Roads covered by flood water.
4. **Road Non-flooded** — Roads not covered by flood water.
5. **Water** — Natural water bodies (e.g., rivers and lakes); considered distinct from flood water.
6. **Tree** — Vegetation including trees and bushes.
7. **Vehicle** — Car, lorries, trucks, etc.
8. **Pool** — Man-made swimming pools, typically behind houses.
9. **Grass** — Areas covered with grass.

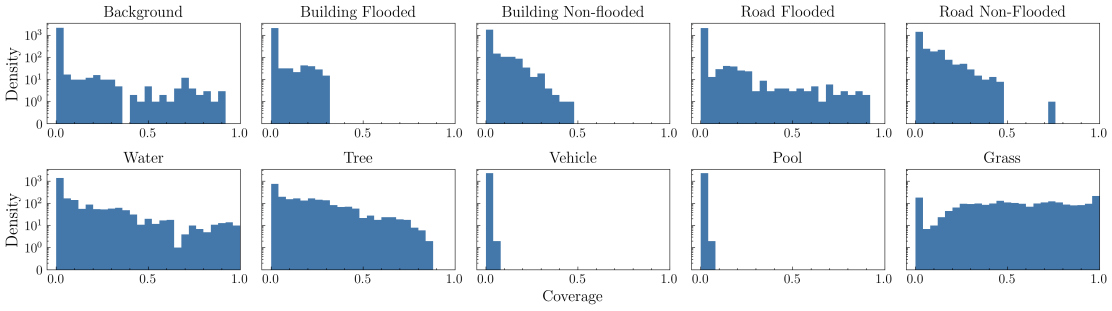

FIGURE 5.6: FloodNet Class Distribution. We compute the class coverage for each image in the dataset, then plot a log histogram with 25 bins.

Similar to the DeepGlobe dataset, FloodNet provides pixel-level annotations, but these segmentation labels are *only* used to generate the regression targets for our training and

for the evaluation of derived patch segmentation, i.e., they are not used during training. The dataset provides fixed train/validation/test splits, so we used these in our work (i.e., no cross-validation; instead five repeats on the same dataset splits). With this configuration, there were 1445/450/448 images (2343 total; $\sim$61.7/19.2/19.1) for train/validation/test. We normalised the images by the dataset mean (0.4111, 0.4483, 0.3415) and standard deviation (0.1260, 0.1185, 0.1177). No other data augmentation was used.

5.4.2 Model Configurations

To apply MIL at different resolutions, we alter the grid size applied over the image (i.e., the number of extracted patches). For the DeepGlobe dataset (square images), we used three grid sizes: 8 x 8, 16 x 16, and 32 x 32. For the FloodNet dataset (non-square images), we used 8 x 6, 16 x 12, and 32 x 24. These grid sizes are chosen such that they can be used directly for three different resolutions in the multi-resolution models, e.g., 32 x 32 ($s = 2$) is twice the resolution of 16 x 16 ($s = 1$), which is twice the resolution of 8 x 8 ($s = 0$).

We also compare our MIL S2P models to a fine-tuned ResNet18 model (He et al., 2016), and two U-Net variations operating on different image sizes (224 x 224 px and 448 x 448 px). These baseline models are trained in the same manner as the S2P models, i.e., using scene-level regression. Although the ResNet model does not produce patch- or pixel-level predictions, we use it as a scene-level baseline as many existing LCC approaches utilise ResNet architectures (Hoeser and Kuenzer, 2020; Hoeser et al., 2020; Rahnemoonfar et al., 2021). For the U-Net models, we follow the same procedure as Wang et al. (2020b) and use class activation maps to recover segmentation outputs. This means U-Net is a stronger baseline than ResNet as it can be used for both scene- and pixel-level prediction. In total, we used 14 different model configurations: one ResNet18 fully supervised approach, two U-Net models, and 11 different configurations of our S2P approach (nine SR and two multi-resolution). Further details are given below, along with summaries in Tables 5.1 and 5.2 for DeepGlobe and FloodNet respectively.

ResNet18 For the ResNet18 model, we treat our regression problem in a fully supervised manner, i.e., without using a MIL approach. Instead, the entire image is resized to 224 x 224 px (the size that ResNet18 expects), and the model makes (only) a scene-level prediction. Conceptually, this is equivalent to using a grid size of one and a patch size of 224 x 224 px (see Tables 5.1 and 5.2). We used a pre-trained ResNet18 model, with weights sourced from TorchVision.[4] We replaced the final classifier layer of the network with a new linear layer of the correct size (7 for DeepGlobe and 10 for

[4] https://pytorch.org/vision/stable/models/generated/torchvision.models.resnet18.html#torchvision.models.resnet18

FloodNet) and then re-trained the entire network, i.e., no weights were frozen during re-training.

U-Net Models We used two different U-Net configurations — one using entire image inputs resized to 224 x 224 px, and the other 448 x 448 px. The model makes scene-level predictions using Global Average Pooling (GAP) over $\mathbf{z}_{Conv}$ (the output of the U-Net's final convolutional layer) followed by a single classification layer L. Using Class Activation Mapping (CAM), it is possible to recover pixel-level segmentation outputs: $\mathbf{M}_c = \mathbf{W}_c\mathbf{z}_{Conv} + \mathbf{B}_c$, where $\mathbf{M}_c$ is the class activation map for class c, $\mathbf{W}_c$ is the weights in L for class c, and $\mathbf{B}_c$ is the bias in L for class c. Note, for the U-Net upsampling process, we experimented with fixed bilinear or learnt convolutional upsampling; the latter increases the number of model parameters. This was included as a hyperparameter during tuning on the DeepGlobe dataset, and it was found that fixed upsampling was best for the U-Net 224 architecture, but learnt upsampling was best for U-Net 448 architecture, leading to an increase in the number of parameters for the U-Net 448 model.

Single Resolution S2P Models We tested nine different configurations of our SR S2P models. Two parameters were changed: the grid size and the patch size. The grid size determines the number of cells extracted from the original image. The patch size is the dimension that each extracted cell is resized to before being input to the model and also determines the model architecture that is used, i.e., we used three different patch sizes and thus designed three different model architectures. This means models with different grid sizes but the same patch size used the same architecture, e.g., the S2P Large 8, S2P Large 16, and S2P Large 32 models all used the same model architecture (hence having the same number of model parameters in Tables 5.1 and 5.2).

Multi-Resolution S2P Models We used two different configurations of the multi-resolution models, see Section 5.3.3. These models use the large SR architecture as their backbone (as this was found to be most effective for S2P SR models, see Section 5.5.3). Both multi-resolution models make predictions at the same resolution as the grid size = 32 SR models ($s = m$), and the MRMO model also makes independent predictions at grid sizes 8, 16, and 32.

Despite using larger patches, the S2P Large architectures have fewer parameters than the S2P Medium architectures as they use an additional convolutional and pooling layer, leading to smaller embedding sizes and thus fewer parameters in the fully connected layers (see Appendix B.2 for further details on the model architectures). Furthermore, while the DeepGlobe multi-resolution MIL models are larger (more parameters) than the DeepGlobe SR MIL models, the DeepGlobe multi-resolution

TABLE 5.1: DeepGlobe model configurations. The grid size determines the number of cells and the size of each cell. Each cell is then resized (patch size), leading to a reduction in the overall image size (effective resolution and scale). # Params is the number of parameters in each model. Note, when using a grid size of 32, a patch size of 102 x 102 px is greater than the maximum possible cell size (i.e., the extracted cells would need to be upsampled and the effective resolution would be greater than 100%). Therefore, for grid size 32, we use a patch size of 76 x 76 px for the large model configurations.

Configuration	Grid Size	Cell Size	Patch Size	Eff. Resolution	Scale	# Params
ResNet18	1 x 1	2448 x 2448 px	224 x 224 px	224 x 224 px	0.8%	11.18M
U-Net 224	1 x 1	2448 x 2448 px	224 x 224 px	224 x 224 px	0.8%	4.31M
U-Net 448	1 x 1	2448 x 2448 px	448 x 448 px	448 x 448 px	3.3%	7.76M
S2P SR Small 8	8 x 8	306 x 306 px	28 x 28 px	224 x 224 px	0.8%	707K
S2P SR Medium 8	8 x 8	306 x 306 px	56 x 56 px	448 x 448 px	3.3%	3.63M
S2P SR Large 8	8 x 8	306 x 306 px	102 x 102 px	816 x 816 px	11.1%	2.98M
S2P SR Small 16	16 x 16	153 x 153 px	28 x 28 px	448 x 448 px	3.3%	707K
S2P SR Medium 16	16 x 16	153 x 153 px	56 x 56 px	896 x 896 px	13.4%	3.63M
S2P SR Large 16	16 x 16	153 x 153 px	102 x 102 px	1632 x 1632 px	44.4%	2.98M
S2P SR Small 32	32 x 32	76 x 76 px	28 x 28 px	896 x 896 px	13.4%	707K
S2P SR Medium 32	32 x 32	76 x 76 px	56 x 56 px	1792 x 1792 px	53.6%	3.63M
S2P SR Large 32	32 x 32	76 x 76 px	76 x 76 px	2432 x 2432 px	98.7%	1.52M
$s = 0$	8 x 8	306 x 306 px	76 x 76 px	608 x 608 px	6.2%	
S2P MRSO $s = 1$	16 x 16	153 x 153 px	76 x 76 px	1216 x 1216 px	24.7%	4.56M
$s = 2$	32 x 32	76 x 76 px	76 x 76 px	2432 x 2432 px	98.7%	
$s = 0$	8 x 8	306 x 306 px	76 x 76 px	608 x 608 px	6.2%	
S2P MRMO $s = 1$	16 x 16	153 x 153 px	76 x 76 px	1216 x 1216 px	24.7%	4.59M
$s = 2$	32 x 32	76 x 76 px	76 x 76 px	2432 x 2432 px	98.7%	

models have fewer parameters than the total number of parameters for the equivalent SR models (a total of 7.48M parameters). This is because the DeepGlobe multi-resolution models use a patch size of 76 x 76 px rather than 102 x 102 px. For the FloodNet multi-resolution models, as the patch size is the same as the SR models, the number of parameters is only slightly higher than the equivalent total for the SR models (a total of 8.94M parameters). The implication of having fewer parameters (in the case of DeepGlobe), or only slightly more (in the case of FloodNet) is that using a single MRMO model is mostly equivalent to training three separate SR models, with the added benefit of improved performance and faster training.

5.4.3 Training Procedure

Models were trained to minimise scene-level RMSE using the Adam optimiser. We utilised early stopping based on validation performance — if the validation RMSE had not decreased for 5 epochs (or the epoch limit was reached, which very rarely happened), we terminated the training procedure and reset the model to the point at which it caused the last decrease in validation loss. Hyperparameter tuning was only

TABLE 5.2: FloodNet model configurations.

Configuration		Grid Size	Cell Size	Patch Size	Eff. Resolution	Scale	# Params
ResNet18		1 x 1	4000 x 3000 px	224 x 224 px	224 x 224 px	0.4%	11.18M
U-Net 224		1 x 1	4000 x 3000 px	224 x 224 px	224 x 224 px	0.4%	4.31M
U-Net 448		1 x 1	4000 x 3000 px	448 x 448 px	448 x 448 px	1.7%	7.76M
S2P SR Small 8		8 x 6	500 x 500 px	28 x 28 px	224 x 168 px	0.3%	707K
S2P SR Medium 8		8 x 6	500 x 500 px	56 x 56 px	448 x 336 px	1.3%	3.63M
S2P SR Large 8		8 x 6	500 x 500 px	102 x 102 px	816 x 612 px	4.2%	2.98M
S2P SR Small 16		16 x 12	250 x 250 px	28 x 28 px	448 x 336 px	1.3%	707K
S2P SR Medium 16		16 x 12	250 x 250 px	56 x 56 px	896 x 672 px	5.0%	3.63M
S2P SR Large 16		16 x 12	250 x 250 px	102 x 102 px	1632 x 1224 px	16.6%	2.98M
S2P SR Small 32		32 x 24	125 x 125 px	28 x 28 px	896 x 672 px	5.0%	707K
S2P SR Medium 32		32 x 24	125 x 125 px	56 x 56 px	1792 x 1344 px	20.1%	3.63M
S2P SR Large 32		32 x 24	125 x 125 px	102 x 102 px	3264 x 2448 px	66.6%	2.98M
S2P MRSO	$s = 0$	8 x 6	500 x 500 px	102 x 102 px	816 x 612 px	4.2%	
	$s = 1$	16 x 12	250 x 250 px	102 x 102 px	1632 x 1224 px	16.6%	8.95M
	$s = 2$	32 x 24	125 x 125 px	102 x 102 px	3264 x 2448 px	66.6%	
S2P MRMO	$s = 0$	8 x 6	500 x 500 px	102 x 102 px	816 x 612 px	4.2%	
	$s = 1$	16 x 12	250 x 250 px	102 x 102 px	1632 x 1224 px	16.6%	8.98M
	$s = 2$	32 x 24	125 x 125 px	102 x 102 px	3264 x 2448 px	66.6%	

carried out on the DeepGlobe dataset, i.e., the same hyperparameters were used for the FloodNet models. This demonstrates that the hyperparameters found through tuning are robust, which is also supported by consistent values across the S2P models. For more details on the implementation, model architectures, and training hyperparameters see Appendices B.1 to B.3.

5.4.4 Results

We evaluate performance on both datasets using four metrics, covering scene-, patch-, and pixel-level prediction. Scene-level performance is scored using RMSE and Mean Absolute Error (MAE), where lower values are better. Both of these metrics compare the scene-level (bag) predictions with the true scene-level coverage labels. For patch-level predictions, we report the patch-level Mean Intersection Over Union (MIOU) (Everingham et al., 2010; Minaee et al., 2021), where larger values are better.[5] Patch labels (only used during evaluation; not during training) are derived from the true segmentation masks, where the labels are the class that has maximum coverage in each patch. For pixel-level prediction, we compute pixel-level MIOU using the original ground-truth segmentation masks.[6] For S2P models, this is achieved by resizing the patch-level segmentation output to the same size as the ground-truth segmentation

[5]Patch-level evaluation is only applicable for our S2P models as the ResNet18 and U-Net approaches do not use patches.

[6]Pixel-level evaluation is not possible for the ResNet18 baseline as it does not produce any segmentation output.

mask. Pixel-level MIOU is the primary metric for evaluation as it best determines the ability of the models to segment the input images — patch-level evaluation is dependent on the grid size so is not a consistent metric across different resolutions (but can be useful to compare patch-level predictions at the same resolution). Strong models should perform well at both scene- and pixel-level prediction, i.e., low scene RMSE, low scene MAE, and high pixel MIOU.

Our results are given in Tables 5.3 and 5.4 for DeepGlobe and FloodNet respectively. We compare the three non-MIL baselines[7] (ResNet18, U-Net 224, and U-Net 448) with S2P SR models operating at three different scales ($s = 0$, $s = 1$, and $s = 2$), the S2P MRSO approach, and the S2P MRMO approach (evaluating the MRMO outputs at all scales). Our first observation is that, for both datasets, all of our S2P models outperform the baseline approaches at both scene- and pixel-level prediction, showing the efficacy of our S2P approach. We also find that, out of all the methods, the MRMO approach achieves the best performance when using its combined $s = m$ output, demonstrating the performance gain of using multiple resolutions. Note the MRMO $s = m$ performance is better than the MRSO $s = m$ performance, suggesting that optimising the model for independent scale predictions ($s = 0$, $s = 1$, and $s = 2$) as well as the combined output ($s = m$) is beneficial for learning — the multi-output nature potentially leads to better representation learning at each individual scale and could help avoid overfitting. However, the MRSO model still outperforms its SR equivalent (S2P SR $s = 2$), again demonstrating the efficacy of using multiple resolutions.

TABLE 5.3: DeepGlobe results. We give the mean performance averaged over five repeat training runs, with bounds using the standard error of the mean. The best performance for each metric is indicated in bold and underlined.

Model	Scene RMSE	Scene MAE	Patch MIOU	Pixel MIOU
ResNet18	0.218 ± 0.008	0.128 ± 0.004	N/A	N/A
U-Net 224	0.136 ± 0.008	0.075 ± 0.004	N/A	0.245 ± 0.008
U-Net 448	0.134 ± 0.008	0.076 ± 0.004	N/A	0.272 ± 0.008
S2P SR $s = 0$	0.090 ± 0.005	0.047 ± 0.002	**0.439 ± 0.014**	0.397 ± 0.014
S2P SR $s = 1$	0.097 ± 0.003	0.051 ± 0.001	0.404 ± 0.016	0.384 ± 0.014
S2P SR $s = 2$	0.104 ± 0.008	0.055 ± 0.004	0.353 ± 0.018	0.345 ± 0.018
S2P MRSO $s = m$	0.093 ± 0.004	0.051 ± 0.002	0.394 ± 0.014	0.388 ± 0.014
S2P MRMO $s = 0$	0.087 ± 0.003	0.048 ± 0.001	0.432 ± 0.012	0.389 ± 0.013
S2P MRMO $s = 1$	0.087 ± 0.004	0.046 ± 0.002	0.412 ± 0.010	0.391 ± 0.010
S2P MRMO $s = 2$	0.095 ± 0.004	0.050 ± 0.002	0.370 ± 0.011	0.362 ± 0.011
S2P MRMO $s = m$	**0.084 ± 0.004**	**0.045 ± 0.002**	0.425 ± 0.013	**0.417 ± 0.013**

[7]The original works reported baseline pixel MIOU scores of 0.43 for DeepGlobe and 0.43 to 0.80 for FloodNet. However, those baselines are trained against segmentation maps rather than coverage labels, so are expected to perform better.

TABLE 5.4: FloodNet results. We give the mean performance averaged over five repeat training runs, with bounds using the standard error of the mean. The best performance for each metric is indicated in bold and underlined.

Model	Scene RMSE	Scene MAE	Patch MIOU	Pixel MIOU
ResNet18	0.145 ± 0.001	0.077 ± 0.002	N/A	N/A
U-Net 224	0.083 ± 0.001	0.039 ± 0.001	N/A	0.193 ± 0.003
U-Net 448	0.080 ± 0.002	0.037 ± 0.002	N/A	0.206 ± 0.004
S2P SR $s = 0$	0.070 ± 0.001	**0.028 ± 0.000**	0.273 ± 0.002	0.233 ± 0.002
S2P SR $s = 1$	0.073 ± 0.000	0.030 ± 0.001	0.269 ± 0.002	0.248 ± 0.002
S2P SR $s = 2$	0.072 ± 0.001	0.030 ± 0.001	0.263 ± 0.003	0.251 ± 0.003
S2P MRSO $s = m$	0.070 ± 0.001	0.029 ± 0.000	0.273 ± 0.004	0.264 ± 0.004
S2P MRMO $s = 0$	0.070 ± 0.001	0.029 ± 0.000	0.273 ± 0.003	0.234 ± 0.002
S2P MRMO $s = 1$	0.070 ± 0.001	0.029 ± 0.000	0.277 ± 0.002	0.256 ± 0.001
S2P MRMO $s = 2$	0.070 ± 0.000	0.030 ± 0.000	0.272 ± 0.001	0.260 ± 0.001
S2P MRMO $s = m$	**0.069 ± 0.001**	**0.028 ± 0.000**	**0.279 ± 0.002**	**0.271 ± 0.002**

To compare the change in performance caused by using different/multiple resolutions, we visualise our S2P results in Figure 5.7. For the DeepGlobe dataset, when using SR models, we observe that increasing the resolution leads to worse performance (greater scene RMSE and MAE, and lower pixel MIOU) — a trade-off between performance and segmentation resolution. However, with the extension to multi-resolution models, it is possible to achieve strong performance whilst operating at higher resolutions, overcoming this trade-off. For the FloodNet dataset, we observe the opposite trend for pixel MIOU — increasing resolution leads to better segmentation performance, even for the SR models. This is likely explained by the different spatial modalities of the two datasets: DeepGlobe images cover much larger spatial areas (e.g., entire towns and fields in a single image) compared to FloodNet (e.g., a handful of buildings or a single road per image). As such, the FloodNet segmentation masks identify individual objects (e.g., a single tree), which require high-resolution inputs and outputs to correctly separate (as opposed to broadly segmenting a forest region as in DeepGlobe). However, we still observe that utilising multiple resolutions gives better performance for FloodNet. We also find that the MRMO independent resolution predictions ($s = 0$, $s = 1$, and $s = 2$) typically outperform the equivalent SR model predictions.

5.5 Discussion

In this section, we further discuss our findings. First, we analyse the model predictions in more detail (Section 5.5.1), then investigate the interpretability and segmentation outputs of the models (Section 5.5.2). Next, we conduct an ablation study into model

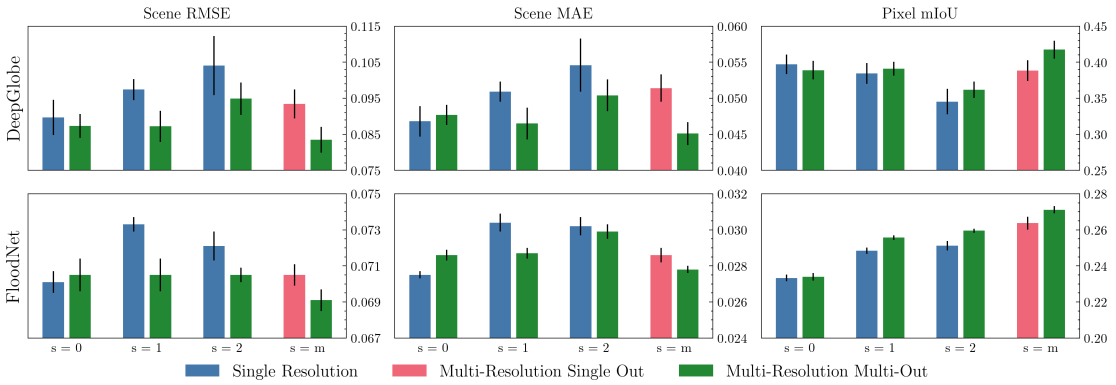

FIGURE 5.7: S2P resolution comparison. We compare model performance for Scene RMSE (left), Scene MAE (middle), and Pixel MIOU (right) for both DeepGlobe (top) and FloodNet (bottom).

architectures (Section 5.5.3). Finally, we discuss the limitations of this study and outline areas for future work (Section 5.5.4).

5.5.1 Model Analysis

We conduct an additional study to further analyse and compare the effectiveness of the different approaches. We use confusion matrices along with precision and recall analysis to investigate the performance of the models on particular classes. The results for the DeepGlobe dataset are presented in Figure 5.8. We observe that all of the analysed models perform best on the agricultural, urban, and forest classes, but struggle on rangeland and unknown classes. For the MRMO $s = m$ model (the best-performing approach) rangeland is most often predicted as agricultural, and unknown is most often predicted as water or agricultural. Rangeland is defined as any green land or grass that is not forest or farmland, so is often difficult to separate from agricultural land. The unknown class is very underrepresented in the dataset compared to the other classes (see Section 5.4.1.1). We note that the MRMO $s = m$ model also achieves the best macro-averaged precision and recall.

In a similar study for FloodNet (Figure 5.9), the models perform best at identifying the road non-flooded, water, tree, and grass classes, but struggle on the background, vehicle, and pool classes. The vehicle and pool classes represent relatively small objects (compared to buildings and roads) and are also very underrepresented (see Section 5.4.1.2), making them hard to identify. However, they are most often classified as building non-flooded, which is an appropriate alternative prediction (i.e., better than classifying them as natural objects such as trees). The background category is a catch-all for any regions that do not belong to the other classes, but it often contains elements of the other classes, making it difficult to segment. Again, we find that the MRMO $s = m$ model has the best precision and recall performance.

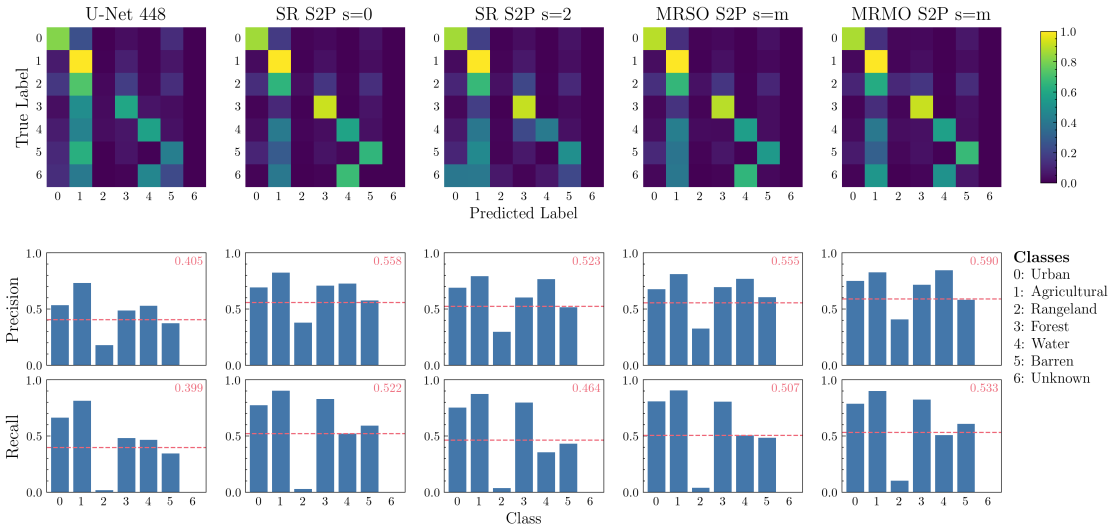

FIGURE 5.8: DeepGlobe model analysis.

Top: Row normalised confusion matrices.

Bottom: Classwise precision and recall, where the dotted line indicates the macro-averaged performance (which is also denoted in the top right corner of each plot).

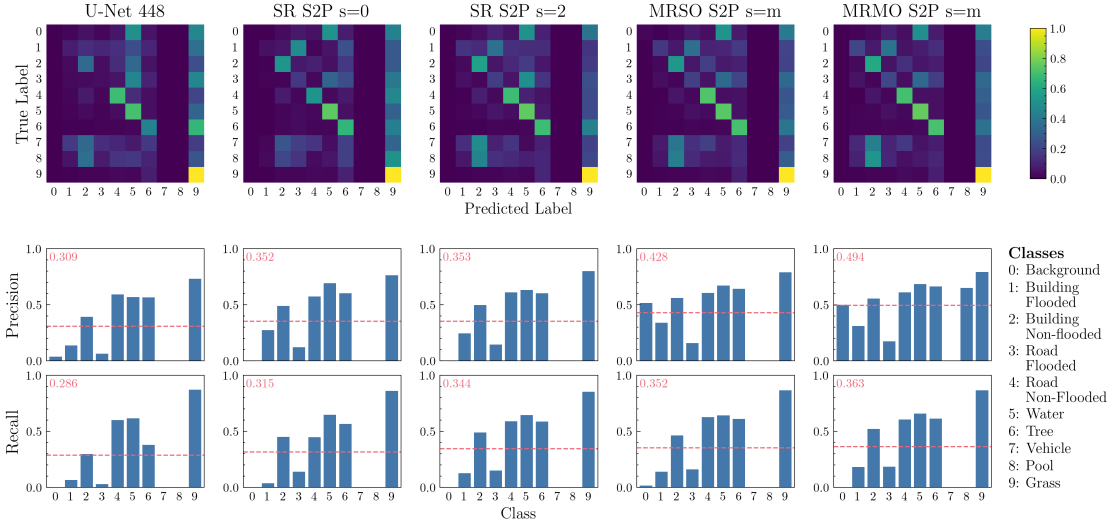

FIGURE 5.9: FloodNet model analysis.

Top: Row normalised confusion matrices.

Bottom: Classwise precision and recall, where the dotted line indicates the macro-averaged performance (which is also denoted in the top left corner of each plot).

5.5.2 Model Interpretability

As our S2P models inherently produce patch-level predictions, we can directly interpret their segmentation outputs. In Figure 5.10, we do so for the MRMO model on the FloodNet dataset. We observe that the model can accurately separate the non-flooded building and road, and also identifies the grass and trees. Furthermore, we find the $s = m$ predictions are an improvement over the independent predictions, notably that

the $s = m$ output is less noisy than the $s = 2$ predictions. We also note that the model is also to support and refute different classes, which further aids interpretability.

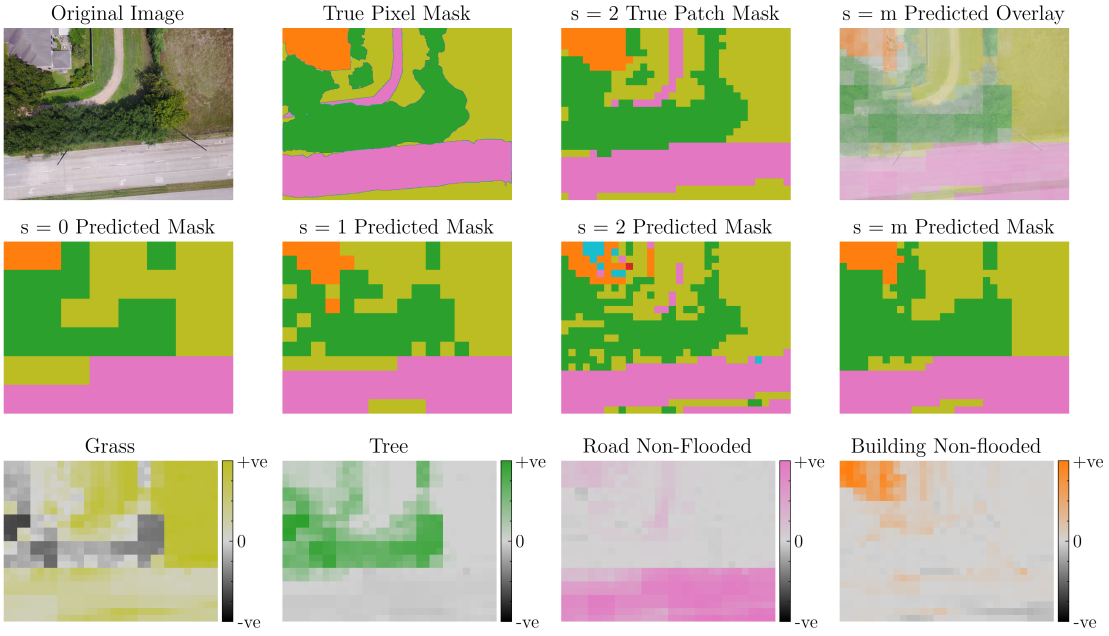

FIGURE 5.10: MRMO interpretability example.

Top (from left to right): the original dataset image; the true pixel-level mask; the true patch-level mask at resolution $s = 2$; the predicted mask from the MRMO model overlaid on the original image.

Middle: Predicted masks for each of the MRMO outputs.

Bottom: MRMO $s = m$ predicted masks for each class, showing supporting (+ve) and refuting regions (-ve).

5.5.3 Ablation Study: Single Resolution Configurations

Changing the patch size (the dimensions to which each cell from the grid extraction process is resized) influences the effective resolution (i.e., the level of downsampling) at which the S2P models are operating and also impacts the MIL model architectures. We conducted an ablation study using three different patch sizes (small, medium, and large). This was done for each of three resolutions $s = 0$, $s = 1$, and $s = 2$, resulting in nine different S2P SR configurations (see Section 5.4.2). As shown in Figure 5.11, we find that the large model configuration has the best average performance on all metrics for both datasets. This matches our intuition that operating at higher resolutions leads to better performance. Given these results, we used the large configuration as the backbone of the multi-resolution S2P models, and as the choice of SR model in our main results (Tables 5.3 and 5.4).

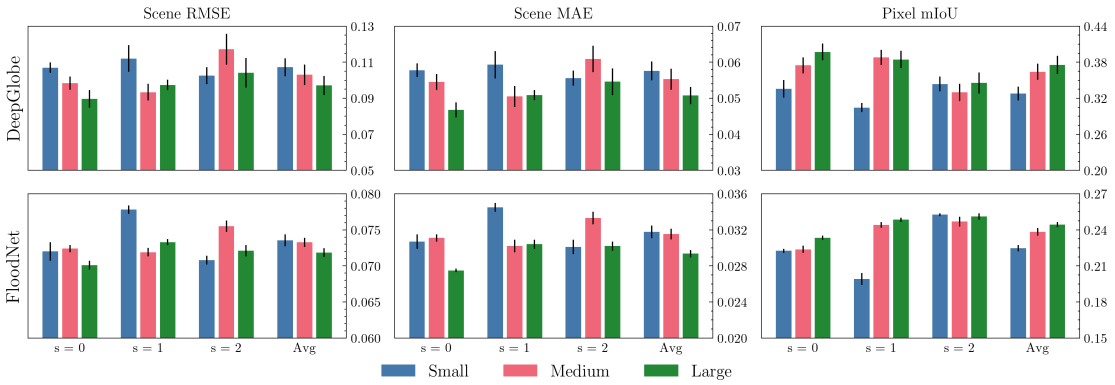

FIGURE 5.11: SR S2P ablation study. We observe that, on average, the large configuration achieves the best performance (lowest RMSE, lowest MAE, and highest MIOU) on both datasets. Error bars are given representing the standard error of the mean from five repeats.

5.5.4 Limitations and Future Work

While our new multi-resolution S2P approach outperforms existing methods, there are still limitations to the method. For the DeepGlobe dataset, the model struggles to separate rangeland and agricultural regions, and for the FloodNet dataset, it struggles to correctly separate flooded regions and identify smaller objects such as pools and vehicles. These problems mostly stem from dataset imbalance; future work could investigate dataset balancing or augmentation to overcome such class imbalance. It could also be beneficial to extend the MIL models to incorporate additional spatial information — this could help separate small objects from the larger objects they often appear next to, e.g., swimming pools from houses. Potential alternatives include utilising positional embeddings (Vaswani et al., 2017), MIL attention (Ilse et al., 2018), or graph neural networks (Tu et al., 2019).

When considering the application of our work to other datasets, there are two points to consider. First, while our solution will scale to datasets containing more images of a similar size to the ones studied in this work, there could be resource issues when using datasets containing larger images (i.e., images larger than 4000 x 3000 px). The computational resources required are determined by the image size, the model's architecture, and the effective resolution at which the model is operating. Reducing the model complexity or lowering the image resolution would overcome potential issues. However, we did not run into any such issues in our work (see Appendix B.1 for details on our compute resources). Second, in this work, we used datasets with existing segmentation labels that were converted into scene-level coverage labels. An area for future work is to investigate how to generate coverage labels for datasets without segmentation labels. This could also incorporate an analysis of model performance in the presence of label noise, i.e., imperfect coverage labels.

Finally, we only tuned training hyperparameters on the DeepGlobe dataset (see Section 5.4.3 for more details). This was an intentional choice, as it allowed us to demonstrate that the hyperparameters found through tuning were robust (i.e., useful default parameters for training on other datasets). However, better performance could potentially be achieved on FloodNet by tuning the training hyperparameters specifically for that dataset and optimising the model architectures as well as the training hyperparameters.

5.6 Conclusion

In this work, we presented a novel method for segmenting EO RS images. Our method extends our previous SR S2P approach to multiple resolutions, which leads to improved performance and interpretability. We demonstrated the efficacy of our approach on two datasets: DeepGlobe (LCC from satellite imagery) and FloodNet (NDR from aerial imagery). As our S2P approaches do not require segmentation labels, faster and easier curation of larger and more diverse datasets can be achieved, facilitating the wider use of ML for climate change mitigation in technology, government, and academia.

This chapter and the previous chapter have broadly focused on interpretable MIL for computer vision problems. In the next two chapters, we develop methods for interpretable MIL applied to sequential data. Specifically, in the next chapter we work on Reward Modelling (RM) for Reinforcement Learning (RL).

Part III

Interpretable Multiple Instance Learning for Sequential Data

Chapter 6

Non-Markovian Reward Modelling from Trajectory Labels via Interpretable Multiple Instance Learning

This chapter contains work completed for a paper published at the conference on Neural Information Processing Systems (NeurIPS) 2022, which was an in-person conference held in New Orleans. The content presented is mostly unchanged from its original format: elements of the original appendix have been brought into the main body, certain sections have been slightly reworded, and some terminology/notation has been modified for consistency with the rest of this thesis. This work was a collaboration with Tom Bewley, a fellow Turing PhD student studying at the University of Bristol. We are both considered lead authors with equal contribution to the original work. Tom's technical contributions focused on the Reinforcement Learning portion of the work, while mine focused on the Multiple Instance Learning and Reward Modelling portions. Both of us contributed equally to writing the manuscript.

Paper

Joseph Early, Tom Bewley, Christine Evers, and Sarvapali Ramchurn. Non-Markovian reward modelling from trajectory labels via interpretable multiple instance learning. *Advances in Neural Information Processing Systems*, 35, 2022a. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b157cfde6794e93b2353b9712bbd45a5-Paper-Conference.pdf

GitHub: https://github.com/JAEarly/MIL-for-Non-Markovian-Reward-Modelling

6.1 Introduction

There is a growing consensus around the view that aligned and beneficial Artificial Intelligence (AI) requires a reframing of objectives as being contingent, uncertain, and learnable via interaction with humans (Russell, 2019). In Reinforcement Learning (RL), this proposal has found one formalisation in Reward Modelling (RM)[1]: the inference of agent objectives from human preference information such as demonstrations, pairwise choices, approval labels, and corrections (Leike et al., 2018). Prior work in RM typically assumes that a human evaluates the *return* (quality) of a sequential trajectory of agent behaviour by summing equal and independent *reward* assessments of instantaneous states and actions, with the aim of RM being to reconstruct the underlying reward function. However, in reality, the human's experience of a trajectory is likely to be temporally extended (e.g., via a video clip (Christiano et al., 2017) or real-time observation), which opens the door to dependencies between earlier events and the assessment of later ones. The independence assumption may be both psychologically unrealistic given human memory limitations (Kahneman, 2000), and technically naïve given the difficulty of building complete instantaneous state representations (Kaelbling et al., 1998). We thus seek to generalise RM to allow for temporal dependencies in human evaluation, by postulating *hidden state* information that accumulates over a trajectory. Reconstruction of the human's preferences now requires the modelling of hidden state dynamics alongside the reward function itself.

In tackling this generalised problem, we identify a structural isomorphism between RM (specifically from trajectory return labels) and the established field of Multiple Instance Learning (MIL) (Carbonneau et al., 2018). Trajectories are recast as *bags* and constituent state-action pairs as *instances*, which collectively contribute to labels provided at the bag level by interacting in potentially complex ways. This mapping inspires a range of novel MIL model architectures that use Long Short-Term Memory (LSTM) modules (Hochreiter and Schmidhuber, 1997) to recover the hidden state dynamics and learn instance-level reward predictions from return-labelled trajectories of arbitrary length. In experiments with synthetic oracle labels, we show that our MIL RM models can accurately reconstruct ground truth hidden states and reward functions for non-Markovian tasks, and can be straightforwardly integrated into RL agent training to achieve performance matching, or even exceeding, that of agents with direct access to true hidden states and rewards. We then apply interpretability analysis to understand what the models have learnt.

Our contributions are as follows:

[1] Also known as reinforcement learning from human feedback (RLHF).

1. We generalise RM to handle *non-Markovian* rewards that depend on hidden features of the environment or the psychology of the human evaluator in addition to visible states/actions.

2. We identify a structural connection between RM and MIL, creating the opportunity to transfer concepts and methods between the two fields.

3. We propose novel LSTM-based MIL models for this generalised RM problem, and develop interpretability techniques for understanding and verifying the learnt reward functions.

4. We compare our proposed models to existing MIL baselines on five non-Markovian tasks, evaluating return prediction, reward prediction, robustness to label noise, and interpretability.

5. We demonstrate that the hidden state and reward predictions of our MIL RM models can be used by RL agents to solve non-Markovian tasks.

The remainder of this work is as follows. Section 6.2 discusses related work in RM and MIL, and Section 6.3 gives a formal problem definition and describes our MIL-inspired methodology. Sections 6.4 and 6.5 present experiments and results on toy and advanced RL problems respectively. We discuss key findings in Section 6.6, and Section 6.7 concludes.

6.2 Background and Related Work

Reward Modelling RM (Leike et al., 2018) aims to infer a reward function from revealed human preference information such as demonstrations (Ng et al., 2000), pairwise choices (Christiano et al., 2017), corrections (Bajcsy et al., 2017), good/bad/neutral labels (Reddy et al., 2020), or combinations thereof (Jeon et al., 2020). Most prior work assumes a human evaluates a trajectory by summing independent rewards for each state-action pair, but in practice their experience is likely to be temporally extended (e.g., via a video clip), creating the opportunity for dependencies to emerge between earlier events and the assessment of later ones. As noted by Bewley and Lecue (2022), such dependencies may arise from cognitive biases such as anchoring, prospect bias, and the peak-end rule (Kahneman, 2000), but they could equally reflect rational drivers of human preferences not captured by the state representation. Some efforts have been made to model temporal dependencies, such as a discrete psychological mode which evolves over consecutive queries about hypothetical trajectories (Basu et al., 2019), or a monotonic bias towards more recently-viewed timesteps due to human memory limitations (Lee et al., 2021b). Elsewhere, Shah et al. (2020) use human demonstrations and binary approval labels to learn temporally

extended task specifications in logical form. In comparison to these restricted examples, our work provides a more general approach to capturing temporal dependencies in RM.

Non-Markovian Rewards In the canonical RL problem setup of a Markov decision process (MDP), rewards depend only on the most recent state-action pair. In a non-Markovian reward decision process (NMRDP) (Bacchus et al., 1996), rewards depend on the full preceding trajectory (Bacchus et al., 1996). NMRDPs can be *expanded* into MDPs (and thus solved by RL) by augmenting the state with a hidden state that captures all reward-relevant historical information, but this is typically not known *a priori*. Data-driven approaches to learning NMRDP expansions (Icarte et al., 2018) often make use of domain-specific propositions and temporal logic operators (Bacchus et al., 1997; Thiebaux et al., 2006; Toro Icarte et al., 2019). Outside of the RM context, recurrent architectures such as LSTMs have been used in NMRDPs to reduce reliance on pre-specified propositions (Jarboui and Perchet, 2021). They also have a long history of use in partially observable MDPs, where dynamics are also non-Markovian (Bakker, 2001; Hausknecht and Stone, 2015; Wierstra et al., 2009).

MIL In MIL (Carbonneau et al., 2018), data is structured into bags of instances, where a bag $\mathbf{X}$ is comprised of instances $\{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$ and has an associated bag-level label $\mathbf{Y}$. The instances also have corresponding instance-level labels $\{\mathbf{y}_1, \ldots, \mathbf{y}_k\}$. The aim is to construct a model that learns solely from bag labels; instance labels are not available during training, but may be used later to evaluate instance-level predictions. The simplest MIL approaches assume that instances are independent and that the bag is unordered, but models exist for capturing various types of instance dependencies (Ilse et al., 2018; Tu et al., 2019; Wang et al., 2018). LSTMs have emerged as a natural architecture for modelling temporal dependencies among ordered bags, where they can be utilised to aggregate instance information into an overall bag representation. They have previously been applied to standard MIL benchmarks (Wang et al., 2020a), as well as specific problems such as Chinese painting image classification (Li and Zhang, 2020). As we discuss in Section 6.3.3, these existing models are somewhat unsuitable for use in RM, leading us to propose our own novel model architectures.

6.3 Methodology

In this section, we present the core methodology of our work. We formally define the new paradigm of non-Markovian RM (Section 6.3.1), and then go on to discuss how we can use MIL RM models for training RL agents on non-Markovian tasks (Section 6.3.2). We then draw on existing MIL literature to propose models that can be used to solve this generalised problem (Section 6.3.3).

6.3.1 Formal Definition of Non-Markovian Reward Modelling

Consider an agent interacting with an environment with Markovian dynamics. At discrete time t, the current environment state $\mathbf{s}_t \in \mathcal{S}$ and agent action $\mathbf{a}_t \in \mathcal{A}$ condition the next environment state $\mathbf{s}_{t+1}$ according to the dynamics function $D : \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$. A trajectory $\xi \in \Xi$ is a sequence of state-action pairs, $\xi = ((\mathbf{s}_0, \mathbf{a}_0), ..., (\mathbf{s}_{T-1}, \mathbf{a}_{T-1}))$, where T is the total length of the trajectory. A human's preferences about agent behaviour respect a real-valued return function $G : \Xi \to \mathbb{R}$. In traditional (Markovian) RM, return is assumed to decompose into a sum of independent rewards over state-action pairs, $G(\xi) = \sum_{t=0}^{T-1} R(s_t, a_t)$, where R is a reward function for individual state-action pairs. The aim is to reconstruct $R' \approx R$ from possibly noisy sources of preference information. In our generalised non-Markovian model, we consider the human to observe a trajectory sequentially and allow for the possibility of hidden state information that accumulates over time and parameterises R:

$$G(\xi) = \sum_{t=0}^{T-1} R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{h}_{t+1});\ \mathbf{h}_{t+1} = \delta(\mathbf{h}_t, \mathbf{s}_t, \mathbf{a}_t), \tag{6.1}$$

where δ is a hidden state dynamics function, and $\mathbf{h}_0$ is a fixed value for the initial hidden state. Reconstruction of the human's preferences now requires the estimation of $\delta' \approx \delta$ and $\mathbf{h}'_0 \approx \mathbf{h}_0$ alongside $R' \approx R$. We visualise the difference between Markovian and non-Markovian RM in Figure 6.1.

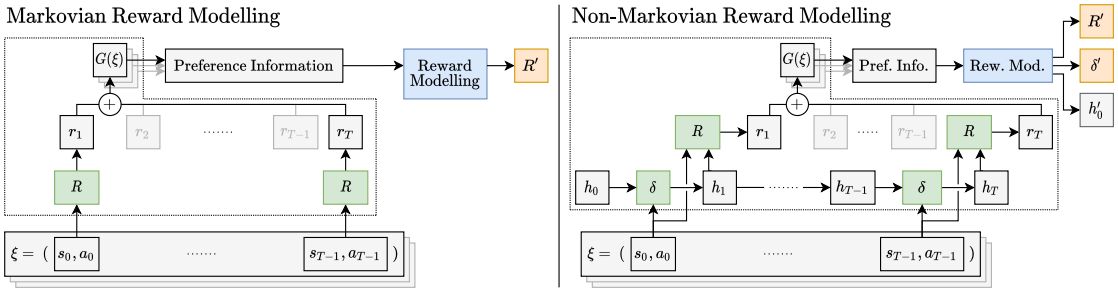

FIGURE 6.1: In Markovian RM, the human is assumed to sum $(+)$ over independent and equal reward assessments for the state-action pairs in a trajectory. In non-Markovian RM, per-timestep rewards additionally depend on hidden state information h that accumulates over time.

The hidden state $\mathbf{h}_t$ may be interpreted as (1) an external feature of the environment that is detectable by the human but excluded from the state, or (2) a psychological feature of the person themselves, through which their response to each new observation is influenced by what they have seen already. The latter framing is more interesting for our purposes and connects to the psychological literature on human judgement, memory, and biases (Kahneman, 2000). In practice, hidden state information may encode the human's preferences about the order in which a sequence of behaviours

should be performed, the effect of historic observations on their subjective mood (and in turn on their reward evaluations), or cognitive biases which corrupt the way they aggregate instantaneous rewards into trajectory-level feedback. All of these complications are liable to arise in practical RM applications, but cannot be handled when the Markovian reward assumption is made.

We note that it is a matter of taste as to whether hidden state information is framed as situated *inside a human evaluator's mind* or *in the environment but only visible to the human*. It is not technically necessary to decide between these two framings, as the mathematical problem of non-Markovian RM is equivalent. From an agent's perspective, a human evaluator is part of an augmented environment, even if they never intervene directly to influence the state. Appendix C.2 elaborates on this discussion, presenting motivating use cases and limitations of non-Markovian RM.

In this work, we focus on one of the simplest and most explicit forms of preference information: direct labelling of returns $G(\xi^i)$ for a dataset of N trajectories $\{\xi^i\}_{i=1}^N$. We aim to solve the reconstruction problem by minimising the squared error in predicted returns:

$$\underset{R',\delta',\mathbf{h}'_0}{\operatorname{argmin}} \sum_{i=1}^{N} \left(G(\xi^i) - \sum_{t=0}^{T^i-1} R'(\mathbf{s}_t^i, \mathbf{a}_t^i, \mathbf{h}_{t+1}'^i) \right)^2 \text{ where } \begin{array}{l} \mathbf{h}_0'^i = \mathbf{h}'_0 \\ \mathbf{h}_{t+1}'^i = \delta'(\mathbf{h}_t'^i, \mathbf{s}_t^i, \mathbf{a}_t^i) \end{array} \forall i \in \{1, \ldots, N\}. \tag{6.2}$$

We observe that Equation 6.2 perfectly matches the definition of a MIL problem. Each trajectory ξ^i can be considered as an ordered bag of instances $((\mathbf{s}_0^i, \mathbf{a}_0^i), \ldots, (\mathbf{s}_{T^i-1}^i, \mathbf{a}_{T^i-1}^i))$ with unobserved instance labels $R(\mathbf{s}_t^i, \mathbf{a}_t^i, \mathbf{h}_{t+1}^i)$, an observed bag label $G(\xi^i)$, and temporal instance interactions via the changing hidden state $\mathbf{h}_t^i$.[2] To explore this relationship through the lens of MIL terminology: only bag-level labels are provided (sparse reward in the form of trajectory-level returns), and we want to create RM models capable of instance (dense reward) prediction while only learning from bag-level (sparse reward) labels. This is a common objective in general MIL interpretability, so this correspondence motivates us to explore how MIL can be applied in RM, and then review the space of existing MIL models (specifically those that model temporal dependencies among instances).

6.3.2 Training Agents with Non-Markovian Reward Modelling

In this work, as in RM more widely, we are not solely interested in learning reward functions to represent human preferences, but also in the downstream application of rewards to train agents' action-selection policies. After optimising our LSTM-based

[2] In this case, the trajectory length T is equivalent to the bag size k in general MIL.

models on offline trajectory datasets (see Section 6.3.3), we deploy them at the interface between conventional RL agents and their environments. Going beyond prior work, where a learnt model is used to either generate a reward signal for an agent to maximise (Christiano et al., 2017) or augment its observed state representation with hidden state information (Icarte et al., 2018), our models serve a dual role, providing *both* rewards and state augmentations. Figure 6.2 describes this setup in detail, and in the next section we detail the MIL model architectures used.

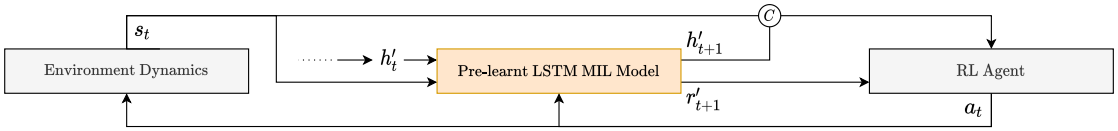

FIGURE 6.2: During RL agent training, our LSTM MIL models sit at the centre of the agent-environment loop by which states $\mathbf{s}_t$ and actions $\mathbf{a}_t$ are exchanged. We focus on episodic tasks, where the environment state periodically resets. The LSTM hidden state is simultaneously reset to $\mathbf{h}'_0$ at the start of an episode, then is iteratively updated over time t given the state-action pairs $\mathbf{s}_t, \mathbf{a}_t$. At time t the environment state $\mathbf{s}_t$ is augmented with the post-update hidden state $\mathbf{h}'_{t+1}$ by concatenation, and this augmented state is observed by the agent. $\mathbf{s}_t$, $\mathbf{a}_t$ and $\mathbf{h}'_{t+1}$ are used to compute a reward r'_{t+1} following the relevant steps from Figure 6.3, and the reward is also sent to the agent. In the language of NMRDPs, the hidden state augmentation *expands* the agent's learning problem into an MDP by providing the additional information required to make the rewards Markovian. Note that unlike during learning of the MIL models, return predictions are never required.

6.3.3 Multiple Instance Learning Reward Modelling Architectures

The MIL literature contains a variety of architectures for handling temporal instance dependencies, including graph neural networks (Tu et al., 2019) and transformers (Shao et al., 2021). While effective for many problems, such architectures are an unnatural fit for non-Markovian RM as they contain no direct analogue of a hidden state $\mathbf{h}'_t$ carried forward in time, instead handling dependencies via some variant of message-passing between instances. LSTM-based MIL architectures (Li and Zhang, 2020; Wang et al., 2020a) provide a more promising starting point since they explicitly represent both $\mathbf{h}'_t$ (implemented as a continuous-valued vector) and its temporal dynamics function δ' (a particular arrangement of gating functions).

Starting from an existing LSTM-based MIL architecture, we propose two successive extensions as well as a naïve baseline that cannot handle temporal dependencies. All four architectures include a Feature Extractor (FE) for mapping state-action pairs into feature vectors and a Head Network (HN) that outputs predictions. These architectures are depicted in Figure 6.3. Note we use the same nomenclature as Carbonneau et al. (2018) and Wang et al. (2018): *embedding space* approaches produce an overall bag representation that is used for prediction, while *instance space* approaches produce predictions for each instance in the bag and then aggregate those predictions to a final bag prediction.

Base Case: Embedding Space LSTM (`EmbeddingLSTM`) This architecture, proposed by Wang et al. (2020a), processes all instances in a bag sequentially and uses the final LSTM hidden state as a bag embedding. This is fed into the HN, which predicts the trajectory return g' (equivalent to a MIL bag-level prediction $\hat{\mathbf{Y}}$). Although this model can account for temporal dependencies, it does not inherently produce reward predictions r'_t (equivalent to MIL instance predictions $\hat{\mathbf{y}}_t$). As such, determining the reward predictions will require post hoc analysis. While methods exist for computing instance importance values as a form of interpretability (Early et al., 2022c, Chapter 4), these are not guaranteed to sum to the bag label as stipulated by the reward-return formulation. We propose a new method: at time t, the predicted reward r'_t is calculated by feeding the LSTM hidden state at times $t-1$ and t into the HN to obtain two *partial* returns g'_{t-1} and g'_t, and computing the difference of the two, i.e., $r'_t = g'_t - g'_{t-1}$, where $g'_0 = 0$. This post hoc computation is shown in purple in Figure 6.3.

Extension 1: Instance Space LSTM (`InstanceLSTM`) The post hoc computation of reward proposed above is rather inelegant and often yields poor predictions (see Sections 6.4 and 6.5), likely because rewards are never computed or back-propagated through during learning. This leads us to propose an improved architecture, which is structurally similar but differs in how network outputs are mapped onto RM concepts. The change places reward predictions on the back-propagation path. Given the LSTM hidden state at time t, the output of the HN is taken to be the instantaneous reward r'_t rather than the partial return. Rewards are computed sequentially for all timesteps in a trajectory and summed to give the return prediction g'. We thereby obtain a model that both handles temporal dependencies and produces explicitly learnt reward predictions.

Extension 2: Concatenated Skip Connection (CSC) Instance Space LSTM (`CSCInstanceLSTM`) In both of the preceding architectures, the LSTM hidden state $\mathbf{h}'_t$ is the sole input to the HN. This requires $\mathbf{h}'_t$ to represent all reward-relevant information from both the true hidden state $\mathbf{h}_t$ and the latest state-action pair $\mathbf{s}_{t-1}, \mathbf{a}_{t-1}$ to achieve good performance. To lighten the load on the LSTM, we further extend `InstanceLSTM` with a skip connection (He et al., 2016; Huang et al., 2017) which concatenates the FE output onto the hidden state before feeding it to the HN. In principle, this should allow the hidden state to solely focus on representing temporal dependencies. As well as improving RM performance compared to an equivalent model without skip connections, we find in Section 6.6.1 that this modification tends to yield more interpretable and disentangled hidden state representations.

Markovian Baseline: Instance Space Neural Network (`Instance`) While the above models are designed for non-Markovian settings, they also work in Markovian settings. To quantify the cost of ignoring temporal dependencies, we also run experiments with a

baseline architecture that is only designed for Markovian settings. This model feeds only the FE output for each state-action pair into the HN, yielding fully-independent reward predictions which are summed to give the return prediction. This independent predict-and-sum architecture has precedence in both MIL, where it is referred to as mi-Net by Wang et al. (2018), and in RM, where it embodies the de facto standard Markovian reward assumption (Christiano et al., 2017). Note this is the same MIL pooling approach used as one of the models in Chapter 4, and the underlying approach for the Scene-to-Patch (S2P) models in Chapter 5.

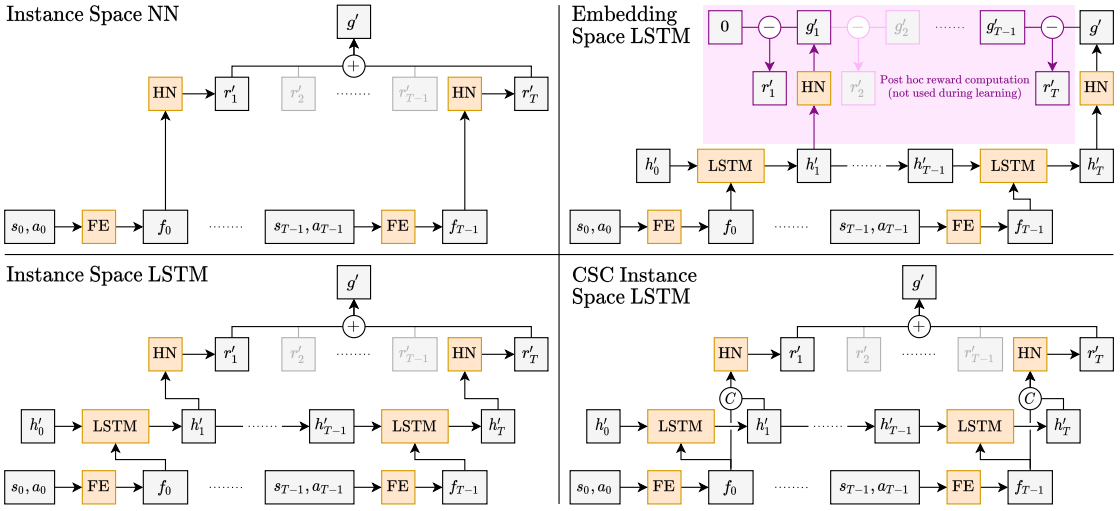

FIGURE 6.3: MIL architectures used in this work. FE = Feature Extractor; HN = Head Network; $(+)$ = scalar summation; $(-)$ = scalar subtraction; (C) = vector concatenation.

6.4 Preliminary Experiments on Toy Datasets

In this section, we detail our preliminary experiments that were run on toy problems to initially develop and validate our approach. Below we outline the datasets (Section 6.4.1), models and training hyperparameters (Section 6.4.2), and results (Section 6.4.3) of these experiments. For further details on our implementation and model architectures, see Appendix C.

6.4.1 Datasets

We introduce three toy datasets, each abstracted from the RL context, to act as benchmark tests for our models. Each of these datasets uses ordered bags comprised of two-dimensional instances, where each instance has an associated label (reward), and the overall bag label (return) is the sum of the instance labels.

Toggle Switch An instance is the position of a toggle switch ts and a value v; if the switch is on ($ts = 1$), then the reward for the instance is v; otherwise the reward for the instance is 0. Here, there is no hidden information, as it is possible to calculate the reward for an instance from its contents ts, v alone. This serves as a null example to elucidate what happens in a Markovian setting.

Push Switch We modify the toggle switch setup so that the instance now represents a push switch ($p = 1$ if this switch is pressed), where pressing the switch flips a binary hidden state ts (the same information as was previously represented by the toggle switch). This hidden state then determines the reward as before (v if $ts = 0$, else 0). As ts must be tracked between successive instances, it is not possible to determine the reward for an instance solely by observing its contents p, v, so this setup is non-Markovian.

Dial We generalise the hidden state from a binary switch to a continuous-valued dial. Given an instance m, v, the dial's current value d is moved up or down by m. The reward is then given as $d \cdot v$. The problem remains non-Markovian, but now the hidden state that needs tracking, d, is continuous rather than discrete.

6.4.2 Multiple Instance Learning Models and Hyperparameters

When training the MIL models on the toy datasets, we used the Adam optimiser with a batch size of one (i.e., one bag per batch) to minimise Mean Square Error (MSE) loss. Training was performed using validation loss early stopping, i.e., if the validation loss did not decrease after a certain number of training epochs (patience value), we terminated the training and selected the model at which the validation loss was lowest. If the patience value was not reached (i.e., the validation loss kept decreasing), we terminated training after a maximum number of epochs had been reached and selected the model at which the validation loss was lowest. The hyperparameters for training the models on each dataset are given in Table 6.1. Dropout was not used. These hyperparameters were found through a small amount of trial and error, i.e., no formal hyperparameter tuning was carried out.

TABLE 6.1: MIL training hyperparameters for toy RM datasets. LR = Learning Rate; WD = Weight Decay.

Dataset	LR	WD	Patience	Epochs
Toggle Switch	1×10^{-4}	1×10^{-5}	20	100
Push Switch	1×10^{-3}	0	30	150
Dial	1×10^{-3}	0	30	150

6.4.3 Results

For each of the toy datasets, we generate 5000 random bags with between 10 and 20 instances per bag (uniformly distributed). We use an 80/10/10 dataset split for training, validation, and testing, and repeat our experiments with ten different variations of this split (so in total we have ten repeats of each model type for each dataset). We show results for both return and reward reconstruction for the toy datasets in Table 6.2. From these results, we can make several observations. Firstly, as expected, `Instance` only works on the Markovian Toggle Switch dataset, i.e., it fails on the non-Markovian Push Switch and Dial datasets as it is unable to deal with temporal dependencies. We also note that our two proposed architectures (`InstanceLSTM` and `CSCInstanceLSTM`) outperform the baseline `EmbeddingLSTM` on both return and reward, with `CSCInstanceLSTM` providing the best results overall. Finally, we observe that a better return performance does not always guarantee better reward performance: for the Dial dataset, `InstanceLSTM` makes better return predictions than `CSCInstanceLSTM`, but worse reward predictions. A similar outcome can be seen for `EmbeddingLSTM` and `Instance` on the Push Switch dataset.

TABLE 6.2: Toy dataset return (top) and reward (bottom) results. Each measurement is the mean MSE averaged over ten repeats, with the standard errors of the mean also given. Bold entries indicate the best-performing model for each (metric, dataset) pair.

Model	Toggle Switch	Push Switch	Dial	Overall
`Instance`	0.030 ± 0.029	3.337 ± 0.054	5.489 ± 0.157	2.952
`EmbeddingLSTM`	0.008 ± 0.002	0.663 ± 0.194	0.434 ± 0.075	0.368
`InstanceLSTM`	0.062 ± 0.058	0.262 ± 0.154	$\mathbf{0.111 \pm 0.014}$	0.145
`CSCInstanceLSTM`	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.140 \pm 0.065}$	0.121 ± 0.043	**0.087**
`Instance`	0.002 ± 0.002	0.086 ± 0.001	0.954 ± 0.011	0.347
`EmbeddingLSTM`	0.003 ± 0.001	0.206 ± 0.100	0.244 ± 0.077	0.151
`InstanceLSTM`	0.004 ± 0.004	0.021 ± 0.008	0.026 ± 0.004	0.017
`CSCInstanceLSTM`	$\mathbf{0.000 \pm 0.000}$	$\mathbf{0.012 \pm 0.004}$	$\mathbf{0.022 \pm 0.007}$	**0.011**

6.5 Advanced RL Experiments and Results

After initially validating our models on several toy datasets, we focus the bulk of our evaluation on five RL tasks. As running experiments with people is costly, we use the standard RM approach of generating synthetic preference data (here trajectory return labels) using ground truth *oracle* reward functions (Christiano et al., 2017). Unlike prior work, these oracle reward functions depend on historical information that cannot be recovered from the instantaneous environmental state, thereby emulating the disparity between the information that a human evaluator possesses while viewing a trajectory sequentially and that contained in the state alone. In this section, we introduce our RL tasks (Section 6.5.1), define the MIL models used for RM (Section 6.5.2), evaluate the

quality of reward reconstruction (Section 6.5.3), investigate the use of MIL RM models for agent training (Section 6.5.4), and evaluate their robustness to label noise (Section 6.5.5). For further details on our implementation and model architectures, see Appendix C.

Oracle-based experiments are often used to evaluate RM methods since they enable scalable quantitative validation (Griffith et al., 2013; Hadfield-Menell et al., 2017; Reddy et al., 2020). However, we identify three concrete differences between our oracle preference labelling method and realistic human labelling: 1) preference form, 2) preference sparsity, and 3) preference noise. Preference form is the different ways of providing labels; in this work, we used return values which are highly informative and easy to learn from, but it has long been understood that humans find it easier to give less direct feedback, such as pairwise rankings (Christiano et al., 2017) or good/bad/neural labels (Kendall, 1957). Preference sparsity occurs when the time- and cost-expensiveness of eliciting human labels reduces the proportion of data that can be labelled, and preference noise arises from uncertainties in the human labelling process (as opposed to perfect oracle labelling). We decided to focus on noise in this work as it is an established way of making oracle experiments more realistic (Lee et al., 2021b) and also fits in with our discussion of human uncertainties and cognitive biases. Our experiments in Section 6.5.5 indicate that our methods degrade gracefully in the presence of noise, which gives us some confidence that they will transfer well to human labels. However, future work should consider preference sparsity and form, the latter of which will involve modifying the data collection pipeline and loss function (e.g., to a contrastive loss in the case of pairwise rankings). Beyond accounting for these three differences whilst still using oracle labels, the next step would be to conduct evaluations using actual human labels.

6.5.1 Reinforcement Learning Task Descriptions

We apply our methods to five non-Markovian RL tasks, the first four of which are within a common 2D navigation environment and are specifically designed to capture different kinds of non-Markovian structures. Each environment has two spawn zones and an episode time limit of $T = 100$; see Figure 6.4. In each case, the environment state contains the x, y position of the agent only. The tasks involve moving into a *treasure* zone, contingent on some hidden information that cannot be derived from the current x, y position, but is instead a function of the full preceding trajectory. In the first two cases, the hidden information varies with time only, but in the other two, it depends on the agent's past positions.

The first four tasks are implemented in Python within a common 2D simulator following the OpenAI Gym standard (Brockman et al., 2016). The agent's position x, y is moved by one of five discrete actions: up, down, left, right and no-op. In the first four

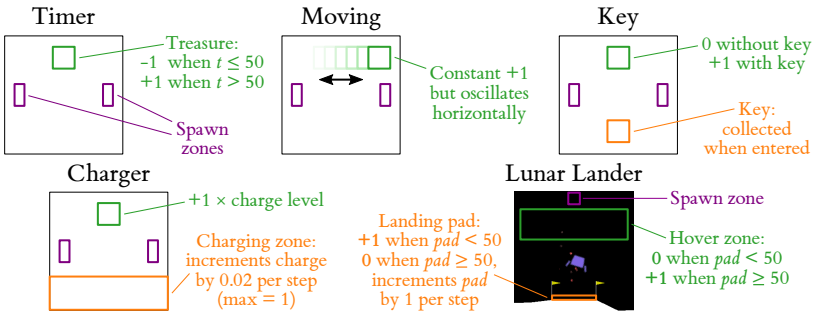

FIGURE 6.4: Visualisations of our proposed non-Markovian RL tasks.

cases, the position is moved by 0.1 in the specified direction. The motion vector is then corrupted by zero-mean Gaussian noise with a standard deviation of 0.02 in both x and y and clipped into the bounds $[0,1]^2$. Zones of interest (spawn, treasure, key, charger zones) are specified as rectangles lying within these bounds. At time t, the environment state $\mathbf{s}_t$ (which is directly observed by the MIL RM models) is the 2D vector of the current position $[x_t, y_t]$; its dynamics are Markovian given the agent's chosen action. The hidden state $\mathbf{h}_t$ is the task-specific information that renders the oracle's reward function R Markovian.

Timer For times $t \leq 50$ the treasure gives a reward of -1 for each timestep that the agent spends inside it, before switching to $+1$ thereafter. Since time is not included in the environment state, recovering the reward function by only observing the agent's current position is impossible. The hidden state simply tracks the current timestep index: $\mathbf{h}_0 = 0$ and $\mathbf{h}_{t+1} = \delta(\mathbf{h}_t, \mathbf{s}_t, \mathbf{a}_t) = \mathbf{h}_t + 1$. Reward is given by

$$R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{h}_{t+1}) = \text{in\_treasure}(\mathbf{s}_t) \times \begin{cases} -1 & \text{if } \mathbf{h}_{t+1} \leq 50, \\ +1 & \text{otherwise,} \end{cases}$$

where $\text{in\_treasure}([x_t, y_t]) = 1$ if $0.4 \leq x_t \leq 0.6$ and $0.7 \leq y_t \leq 0.9$, and 0 otherwise.[3]

Moving The Timer task only captures a binary change, therefore we generalise it to be continuous. In this case, the treasure zone oscillates left and right at a constant speed. Again, this is not captured in the environment state but can be recovered if the length of the preceding trajectory is known. $\mathbf{h}_0 = [0.4, -0.02]$: the initial horizontal position (left edge) and velocity of the moving treasure rectangle. Hidden state dynamics encode the left-right oscillation:

$$\mathbf{h}_{t+1} = \delta(\mathbf{h}_t, \mathbf{s}_t, \mathbf{a}_t) = \left[\mathbf{h}_t[0] + \mathbf{h}_t[1], \begin{cases} \mathbf{h}_t[1] & \text{if } 0 < (\mathbf{h}_t[0] + \mathbf{h}_t[1]) < 0.8, \\ -\mathbf{h}_t[1] & \text{otherwise} \end{cases} \right].$$

[3]Note the timestep indices used here, which result from the order in which environment states, hidden states, and rewards are computed. At time t, the hidden state $\mathbf{h}_t$ is first updated to $\mathbf{h}_{t+1}$ by $\delta(\mathbf{h}_t, \mathbf{s}_t, \mathbf{a}_t)$, then the reward is computed as $R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{h}_{t+1})$, and finally the environment state is updated to $\mathbf{s}_{t+1}$ by $D(\mathbf{s}_t, \mathbf{a}_t)$.

Reward is given by $R([x_t, y_t], \mathbf{a}_t, \mathbf{h}_{t+1}) = 1$ if $\mathbf{h}_{t+1}[0] \leq x_t \leq \mathbf{h}_{t+1}[0] + 0.2$ and $0.7 \leq y_t \leq 0.9$, and 0 otherwise.

Key Before reaching the treasure zone, the agent must first enter a second zone to collect a key; otherwise it receives 0 reward. As the key's status is not captured in the environment state, a temporal dependency exists between the agent's past positions and the reward it obtains from the treasure. $\mathbf{h}_0 = 0$, indicating that the agent initialises without the key. The key collection dynamics are encoded by

$$\mathbf{h}_{t+1} = \delta(\mathbf{h}_t, [x_t, y_t], \mathbf{a}_t) = \begin{cases} 1 & \text{if } 0.4 \leq x_t \leq 0.6 \text{ and } 0.1 \leq y_t \leq 0.3, \\ \mathbf{h}_t & \text{otherwise.} \end{cases}$$

Reward is given by $R(\mathbf{a}_t, \mathbf{a}_t, \mathbf{h}_{t+1}) = \text{in\_treasure}(\mathbf{s}_t) \cdot \mathbf{h}_{t+1}$, where the in_treasure function is the same as in the Timer task.

Charger We generalise the Key task by replacing the key zone with a charging zone that builds up the amount of reward the agent will receive when it reaches the treasure. The reward now depends not only on whether the agent visits a zone (binary) but how long it spends there (continuous). $\mathbf{h}_0 = 0$, indicating an initial charge level of zero. The charging dynamics are encoded by

$$\mathbf{h}_{t+1} = \delta(t, [x_t, y_t], \mathbf{a}_t) = \begin{cases} \min(\mathbf{h}_t + 0.02, 1) & \text{if } y_t \leq 0.3, \\ \mathbf{h}_t & \text{otherwise.} \end{cases}$$

Reward is given identically to the Key task, $R(\mathbf{a}_t, \mathbf{a}_t, \mathbf{h}_{t+1}) = \text{in\_treasure}(\mathbf{s}_t) \cdot \mathbf{h}_{t+1}$.

For the fifth and most complex task, **Lunar Lander**, we adapt `LunarLander-v2` from OpenAI Gym (Brockman et al., 2016), where the task is to land on a landing pad. Our adaptation adds the condition that the lander should take off again and stably hover after 50 timesteps on the landing pad.[4] This is analogous to the Charger task but with a larger state-action space and longer episodes ($T = 500$). We leave the state and action spaces unmodified from the original task. The 8D state vector is $[x, y, v^x, v^y, \theta, \dot{\theta}, c^l, c^r]$, where x, y and v^x, v^y are the landing craft's horizontal and vertical positions and velocities, θ and $\dot{\theta}$ are its angle from vertical and angular velocity, and c^l, c^r are two binary contact detectors indicating whether the left and right landing legs are in contact with the ground. The 2D continuous action $[u^m, u^s]$ is a pair of throttle values for two engines: main u^m and side u^s.

We also retain the default initialisation conditions (the lander spawns in a narrow zone above the landing pad, with slightly randomised orientation and velocities), the

[4] For an example, see https://github.com/JAEarly/MIL-for-Non-Markovian-Reward-Modelling/blob/main/LL_gif.gif.

automatic termination of episodes when $|x|$ exceeds 1 (i.e., when the lander leaves the rendered screen area), and the physics that determine how the lander responds to engine activations. However, we replace the standard reward function with an oracle that rewards the agent for landing on the pad for up to 50 timesteps and then taking off again to hover within a target zone until an episode time limit ($T = 500$) is reached. Rendering this two-stage objective Markovian requires a hidden state $\mathbf{h}_t$ that tracks the number of timesteps spent on the pad so far. Formally, reward is given by

$$R(\mathbf{s}_t, \mathbf{a}_t, \mathbf{h}_{t+1}) = \begin{cases} R_{\text{pad}}(\mathbf{s}_t) + R_{\text{shaping}}(\mathbf{s}_t, 0) & \text{if } \mathbf{h}_{t+1} < 50, \\ R_{\text{no\_contact}}(\mathbf{s}_t) + R_{\text{hover}}(\mathbf{s}_t) + R_{\text{shaping}}(\mathbf{s}_t, 1) & \text{otherwise.} \end{cases}$$

Each sub-reward is as follows:

R_{pad} rewards the agent for being central with both legs on the ground (i.e., on the pad):

$$R_{\text{pad}}(\mathbf{s}_t) = \begin{cases} 1 & \text{if } -0.2 \leq x_t \leq 0.2 \text{ and } c_t^l = 1 \text{ and } c_t^r = 1, \\ 0 & \text{otherwise.} \end{cases}$$

$R_{\text{no\_contact}}$ rewards breaking leg-ground contact:

$$R_{\text{no\_contact}}(\mathbf{s}_t) = \begin{cases} 1 & \text{if } c_t^l = 0 \text{ and } c_t^r = 0, \\ 0 & \text{otherwise,} \end{cases}$$

R_{hover} rewards aerial positions in a target zone above the pad:

$$R_{\text{hover}}(\mathbf{s}_t) = \begin{cases} 1 & \text{if } -0.5 \leq x_t \leq 0.5 \text{ and } 0.75 \leq y_t \leq 1.25, \\ 0 & \text{otherwise,} \end{cases}$$

R_{shaping} promotes slow, stable, central flight towards a target vertical position y_{target}:

$$R_{\text{shaping}}(\mathbf{s}_t, y_{\text{target}}) = \\ 0.1 \times \max\left(2 - \left(\sqrt{(x_t)^2 + (y_t - y_{\text{target}})^2} + \sqrt{(v_t^x)^2 + (v_t^y)^2} + |\theta_t| + |\dot{\theta}_t|\right), 0\right).$$

The hidden state dynamics are

$$\mathbf{h}_{t+1} = \begin{cases} \min(\mathbf{h}_{t+1}, 50) & \text{if } R_{\text{pad}}(\mathbf{s}_t) = 1, \\ \mathbf{h}_t & \text{otherwise,} \end{cases}$$

with $\mathbf{h}_0 = 0$.

An important design decision for the LSTM-based models is the size of the hidden state, as it affects both performance and interpretability. For all the above tasks, we know *a*

priori that it is possible to capture the temporal dependencies in at most two dimensions, so we constrain our models to use 2D hidden states. This allows us to visualise and interpret the hidden representations in Section 6.6.1.

6.5.2 Multiple Instance Learning Models and Hyperparameters

The MIL model training on the Timer, Moving, Key, and Charger tasks used the same process as for the toy model training (see Section 6.4.2). However, we also applied dropout (DO) in these models. There are several differences between the MIL training process for the Lunar Lander task and the four other RL tasks. Firstly, the reward targets (and as such, return targets) were scaled down by a factor of 100 in order to avoid extremely large gradients from high prediction targets. For example, a trajectory with an original return of 700 would have a scaled return of 7. Secondly, the input data was scaled linearly between -0.5 and 0.5 (using the minimum and maximum range of each feature). This was found to give more consistent feature ranges than mean/standard deviation scaling as was used in the other tasks (this was due to large outliers in certain features, e.g., the rotational features were largely clustered around 0, but had extreme values up to $\pm$ 90).

We give the MIL training hyperparameters for these RL tasks in Table 6.3. Again, these hyperparameters were found through a small amount of trial and error, i.e., no formal hyperparameter tuning was carried out. As such, better performance of these models could potentially be achieved with better parameters (including shorter training times with a higher learning rate).

TABLE 6.3: RM MIL training hyperparameters for RL tasks.

Dataset	LR	WD	DO	Patience	Epochs
Timer	5×10^{-4}	0	0.1	50	250
Moving	5×10^{-4}	0	0.1	50	250
Key	5×10^{-4}	0	0.1	30	150
Charger	5×10^{-4}	0	0.1	50	250
Lunar Lander	1×10^{-4}	0	0	30	200

6.5.3 Reward Modelling Results

Below we discuss the performance of the reward reconstruction for the different MIL RM models on our five RL tasks. For each task, we generate initial trajectories to form our MIL RM datasets. We obtain datasets of several thousand trajectories per task, containing a wide distribution of outcomes and return values. To do so, for each task, we define a discrete *trajectory classification* function $C_{Task} : \Xi \rightarrow \mathcal{C}_{Task}$, where $|\mathcal{C}_{Task}|$ is the

number of classes. We also define a limit p_{Task} on the proportion of trajectories in the dataset that are allowed to map to each class. These are given as follows:

- **Timer:** $\mathcal{C}_{Timer} = \texttt{num\_neg} \times \texttt{num\_pos}$, where $\texttt{num\_neg} = \{0, \ldots, 50\}$ counts the number of timesteps the agent spends in the treasure while its reward is negative ($t \leq 50$), and $\texttt{num\_pos} = \{0, \ldots, 50\}$ counts the number while the reward is positive ($t > 50$). $|\mathcal{C}_{Timer}| = 51^2 = 2601$ and the per-class limit is $p_{\text{Timer}} = 0.002$.
- **Moving:** $\mathcal{C}_{Moving} = \texttt{num\_treasure}$, where $\texttt{num\_treasure} = \{0, \ldots, 100\}$ counts the timesteps spent in the treasure. $|\mathcal{C}_{Moving}| = 101$ and $p_{\texttt{Moving}} = 0.05$.
- **Key:** $\mathcal{C}_{Key} = \{\texttt{no\_key}, \texttt{key\_no\_treasure}, \texttt{treasure}\}$, where the class is `no_key` if the key is not collected, `key_no_treasure` if the key is collected but the treasure is not reached, and `treasure` if the treasure is reached after collecting the key. $|\mathcal{C}_{Key}| = 3$ and p_{Key} is defined on a per-class basis: 0.25 for `no_key` and `key`, and 0.5 for `treasure`.
- **Charger:** $\mathcal{C}_{Charger} = \texttt{num\_treasure} \times \texttt{charge\_bin}$, where $\texttt{num\_treasure} = \{0, \ldots, 100\}$ counts the timesteps spent in the treasure and $\texttt{charge\_bin} = \{1, \ldots, 20\}$ is a binned representation of the *mean* charge level when in the treasure (e.g., 0.0 maps to bin 1, 0.48 to bin 10, 0.96 to bin 20). $|\mathcal{C}_{Charger}| = 2020$ and $p_{\text{Charger}} = 0.002$.
- **Lunar Lander:** $\mathcal{C}_{LL} = \texttt{pad\_bin} \times \texttt{take\_off} \times \texttt{hover\_bin}$, where $\texttt{pad\_bin} = \{0, [1:49], 50+\}$ is a binned representation of the number of timesteps spent on the landing pad (i.e., zero, between 1 and 49 inclusive, or at least 50), $\texttt{take\_off} = \{0, 1\}$ is a binary indicator of whether the lander takes off again after being on the pad, and $\texttt{hover\_bin} = \{0, [1:19], 20+\}$ is a binned representation of the number of timesteps spent in the hover zone after being on the pad.[5] $|\mathcal{C}_{LL}| = 18$, of which 9 are actually realisable (e.g., the lander cannot take off from the pad if it never reached it in the first place) and $p_{\text{LL}} = 0.2$.

Using the above functions, we generate a dataset $\mathcal{D}_{Task}$ for each task with a wide distribution of trajectories. Each task-specific monolithic dataset is then broken into training, validation, and testing splits. For the timer, moving, key, and charger tasks, $\mathcal{D}_{Task}$ is assembled iteratively. On each iteration, we generate a length-100 trajectory ξ by sampling agent actions uniform-randomly from the action space (up, down, left, right, no-op) and running them through the simulator. Once the trajectory is complete, we evaluate its class $C_{Task}(\xi)$. If there are already at least $p_{Task} \times 5000$ trajectories in $\mathcal{D}_{Task}$ with this class, ξ is discarded. Otherwise, it is added to $\mathcal{D}_{Task}$. This process repeats until $|\mathcal{D}_{Task}| = 5000$.

[5] If the lander reaches the target of 50 timesteps on the pad, the time in the hover zone is measured from this point onwards. Otherwise, it is measured from the first timestep that the lander leaves the pad.

The state-action space for Lunar Lander is too large for a random generate-and-select algorithm to terminate in any reasonable time. Instead, we recycle the length-500 trajectories generated as a by-product of training the oracle-based RL baselines (black curves in Figure 6.5, plus six more runs not included in the figure). Starting from a bank of 12000 trajectories, filtering based on the threshold $p_{\text{LL}} = 0.2$ yields a final dataset $\mathcal{D}_{LL}$ with 9762 trajectories. Although this approach of relying on oracle-trained agents to generate data may initially appear to "put the cart before the horse", we suggest that it provides a valuable test of the ability of our MIL models to learn from goal-directed (as opposed to random) trajectories, and thus is a step closer to the online bootstrapping approach of simultaneous RM and RL, which we aim to tackle in future work (see Section 6.6.4).

Results from MIL models trained on these trajectories are given in Table 6.4. We observe that `CSCInstanceLSTM` is on average the best-performing model for predicting both trajectory returns and timestep rewards. While `EmbeddingLSTM` performs best at predicting return on the Key and Lunar Lander tasks (as is not constrained to the summation of reward predictions as in the other architectures), it struggles on the reward metric (due to the use of a proxy post hoc method). As it is important for these models to achieve strong performance on both return and reward prediction, the `InstanceLSTM` and `CSCInstanceLSTM` are better candidates than `EmbeddingLSTM`. Also note that `Instance`, which serves as our Markovian RM baseline, performs very poorly on return prediction, indicating that these tasks indeed cannot be learnt without modelling temporal dependencies.

TABLE 6.4: MIL RM return (top) and reward (bottom) results, using ten repeats. The Lunar Lander results are the average test set MSE (with scaling applied; see Section 6.5.2) of the top five models (see Section 6.6.3). For the other tasks, each measurement is the test set MSE averaged over all ten repeats. The standard errors of the mean are given, and the Lunar Lander reward results are scaled by 1×10^{-5}.

Model	Timer	Moving	Key	Charger	Lunar Lander
`Instance`	130.8 ± 1.530	22.24 ± 0.441	7.764 ± 0.232	7.783 ± 0.214	2.297 ± 0.058
`EmbeddingLSTM`	3.151 ± 0.662	13.04 ± 0.899	$\mathbf{0.360 \pm 0.055}$	0.689 ± 0.124	$\mathbf{0.416 \pm 0.048}$
`InstanceLSTM`	7.313 ± 2.627	11.13 ± 1.169	0.488 ± 0.062	0.628 ± 0.126	1.223 ± 0.431
`CSCInstanceLSTM`	$\mathbf{0.605 \pm 0.166}$	$\mathbf{5.307 \pm 0.299}$	0.391 ± 0.083	$\mathbf{0.125 \pm 0.012}$	0.501 ± 0.035
`Instance`	0.217 ± 0.001	0.068 ± 0.000	0.011 ± 0.000	0.025 ± 0.000	7.484 ± 0.861
`EmbeddingLSTM`	101.8 ± 60.35	3.033 ± 0.715	0.010 ± 0.008	0.037 ± 0.016	120.2 ± 24.27
`InstanceLSTM`	0.263 ± 0.038	0.069 ± 0.005	0.002 ± 0.000	0.005 ± 0.001	9.336 ± 3.116
`CSCInstanceLSTM`	$\mathbf{0.073 \pm 0.016}$	$\mathbf{0.026 \pm 0.002}$	$\mathbf{0.001 \pm 0.000}$	$\mathbf{0.001 \pm 0.000}$	$\mathbf{7.365 \pm 1.032}$

6.5.4 Reinforcement Learning Training Results

Following the method in Section 6.3.2, we then train RL agents to optimise the rewards learnt by the LSTM-based models. We use Soft Actor-Critic (Haarnoja et al., 2018) for the Lunar Lander task and Deep Q-Network (Mnih et al., 2015) for the four other tasks.

We evaluate agent performance in a post hoc manner by passing its trajectories to the relevant oracle. This evaluation provides an end-to-end measure of both reward reconstruction and policy learning, and is standard in RM (Christiano et al., 2017). We baseline against agents trained with access to a) the oracle reward function and the oracle hidden states, and b) just the oracle reward function without hidden states (i.e., using only the environment states that are missing information). In Figure 6.5, we observe that `CSCInstanceLSTM` enables the best RL agent performance, coming closest to the oracle. Interestingly, for the Timer and Lunar Lander tasks, `CSCInstanceLSTM` actually outperforms the use of the oracle, suggesting that the learnt hidden states are easier to exploit for policy learning than the raw oracle state (we investigate what these models have learnt in Section 6.6.1). Note the poor performance of agents trained without hidden state information, which aligns with expectations. For further details on agent training, see Appendix C.4.

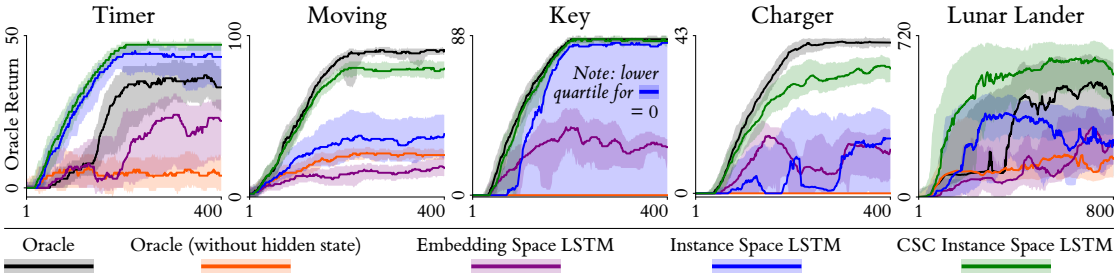

FIGURE 6.5: RL performance for different training configurations on our five RL tasks. The results given are the medians and interquartile ranges. For the oracle results, we trained ten repeats, and for the MIL-LSTM results, we performed one RL training run for each MIL-LSTM model repeat.

For Lunar Lander, we perform a deeper analysis of RL training performance, by decomposing the oracle return curves from Figure 6.5 into the four reward components R_{pad}, $R_{\text{no\_contact}}$, R_{hover} and R_{shaping} (see Section 6.5.1 for definitions). The decomposed curves, shown in Figure 6.6, allow us to diagnose the origins of the performance disparity between runs using different LSTM model architectures. There is relatively little separation in performance on the shaping reward R_{shaping} and pad contact reward R_{pad} (for the latter, all runs end up reliably achieving the maximum possible reward of 49, although those using `EmbeddingLSTM` models require significantly more training time). This suggests that all models have been able to recover these components with reasonable fidelity. However, there are marked differences in performance on $R_{\text{no\_contact}}$ and R_{hover} (the components relating to the second task stage of taking off and moving to the hover zone). For R_{hover}, runs using `CSCInstanceLSTM` peak at a return of around 200 from this component, while those using the other two models rarely achieve a non-zero return, i.e., only the RL agents trained using the `CSCInstanceLSTM` RM models reliably learn to hover. This indicates that the models have learnt very different representations of reward and hidden state dynamics, which are effective for policy learning in the case of the CSC model, and highly ineffective for the others.

Observe that runs using `CSCInstanceLSTM` models outperform those with direct access to the ground truth oracle on all components, and most markedly on R_{hover}. This counterintuitive finding suggests that this model reliably learns hidden state representations that are easier for RL agents to leverage for policy learning than the ground truth ones, and potentially that certain errors in the reward prediction may actually be beneficial for helping agents to complete the underlying task (especially the hovering stage). In typical RL parlance: the model's reward function appears to be better *shaped* than the ground truth. The potential origins of this better-than-oracle phenomenon are investigated in Figure 6.13.

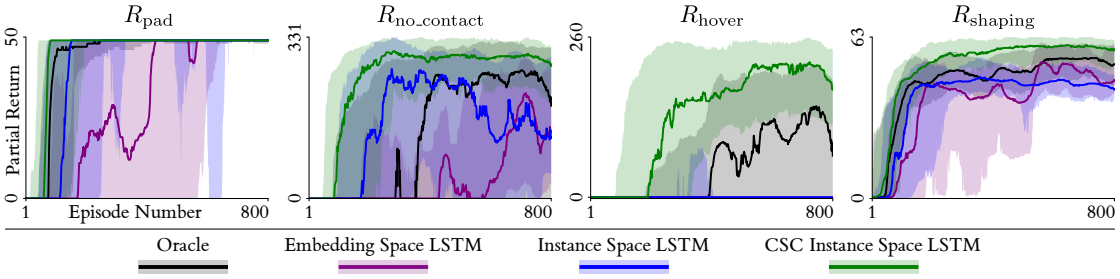

FIGURE 6.6: Decomposed oracle return curves for Lunar Lander.

6.5.5 Robustness to Mislabelling

In this work, the return labels are provided by oracles rather than real people. When using human evaluators, there is likely to be uncertainty in the labels, and it is important to evaluate model robustness against such noise (Lee et al., 2021b). We implement noise through label swapping (Rolnick et al., 2017); this ensures the marginal label distribution remains the same and does not include out-of-distribution returns. In Figure 6.7, we show how both return and reward prediction decay with noise levels increasing from 0 (no labels swapped) to 0.5 (half swapped). The rate, smoothness, and consistency (across three repeats) of this decay vary between tasks, with decay in return prediction generally being smoother. We observe that `CSCInstanceLSTM` remains the strongest predictor of both return and reward in the majority of cases, indicating general robustness and providing evidence that the model should still be effective with imperfect human labels. On all metrics aside from Timer reward loss (where the mix of negative and positive rewards makes the effect of noise especially unpredictable), a noise level of at least 0.3 is required for `CSCInstanceLSTM` to perform as badly as `Instance` does with no noise at all.

6.6 Discussion

In this section, we seek to interpret our MIL RM models, analysing the distribution of learnt hidden states (Section 6.6.1) as well as their temporal dynamics throughout a

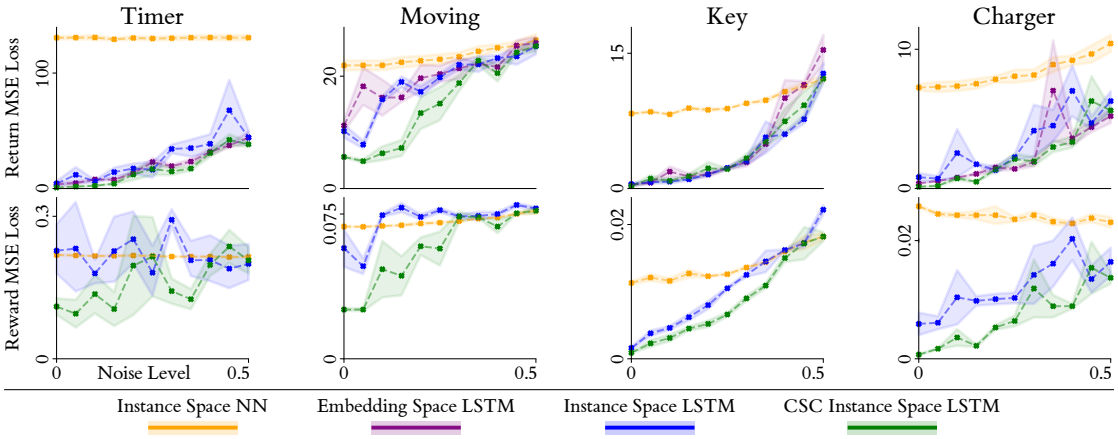

FIGURE 6.7: Performance of MIL RM models subject to label noise. We omit `EmbeddingLSTM` reward losses as they are very high, and the Lunar Lander task due to long training times.

trajectory (Section 6.6.2). We also conduct an in-depth analysis of the Lunar Lander task (Section 6.6.3). Finally, in Section 6.6.4 we discuss the limitations of this work and potential areas for future work.

6.6.1 Hidden State Analysis

The primary purpose of RM is to perform accurate reward reconstruction to facilitate agent training, but there is a secondary opportunity to improve understanding of human preferences through interpretability analysis of the learnt models. We can directly visualise the 2D LSTM hidden states of our oracle experiments, which enables a qualitative comparison of the various model architectures (see Figure 6.8). Visualising the hidden states with respect to the temporal dependencies indicates that `CSCInstanceLSTM` has learnt insightful hidden state representations. Breaking down the `CSCInstanceLSTM` hidden embeddings: for the Timer task, time is represented along a curve, with a sparser representation around $t = 50$ (the crossover point when the treasure becomes positive). For the Moving task, time is similarly captured along with an additional notion of the change in treasure direction from right to left. For the Key task, the binary state of no key vs key is separated, with additional partitioning based on x position, denoting the two different start points of the agent. For the Charger task, the charge level can be recovered and the inflexion point between under-charging and over-charging is captured, i.e., this is where the optimal charge level lies, subject to some noise based on where the agent starts in the spawn zones. In the Lunar Lander task, the model has learnt a strong separation between states on either side of the crossover point when the time on the pad is equal to 50, with high sparsity around the crossover point. In comparison, the Embedding and `InstanceLSTM` have not learnt as sparse a representation.

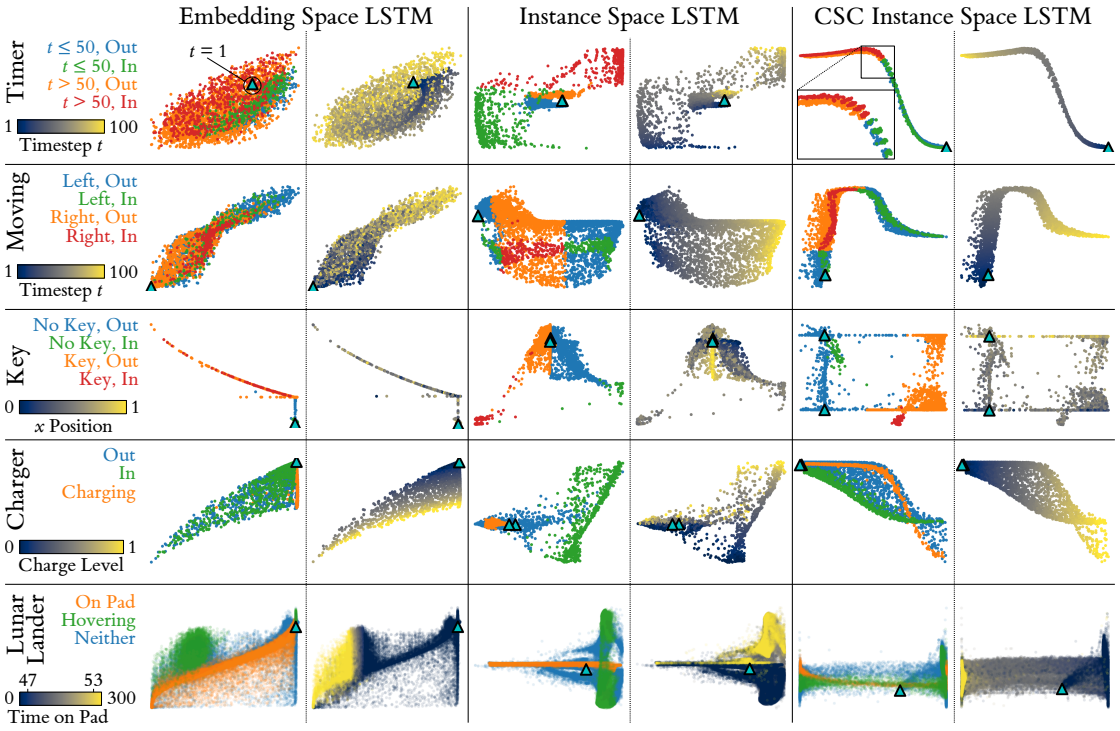

FIGURE 6.8: Learnt hidden state embeddings for our MIL RM models. For each model and task, we categorise the hidden state embeddings depending on the true environment state (first column for each model). In and Out environments states indicate whether the agent is in the treasure zone or not, and for the Moving task, Left and Right indicate the direction in which the goal is currently moving. We also provide labelling based on temporal information (second column for each model). Furthermore, we include markers to indicate the hidden states for the centres of the agent spawn zones. In each case, we elected to use the best-performing repeat for each model as assessed by the reward reconstruction (see Table 6.4). Note, for the Key task, as the temporal information is captured in the state categorisation (No Key vs Key), we use the second column to show the relationship between the hidden embeddings and the agent's x position.

6.6.2 Trajectory Probing

We further interpret our models by visualising the learnt reward with respect to the environment state, and by using hand-specified *probe* trajectories to verify that the learnt hidden state transitions mimic the true transitions. We analyse the best-performing `CSCInstanceLSTM` (according to the reward reconstruction metric) for the Timer (Figure 6.9), Moving (Figure 6.10), Key (Figure 6.11), Charger (Figure 6.12), and Lunar Lander (Figure 6.13) tasks.

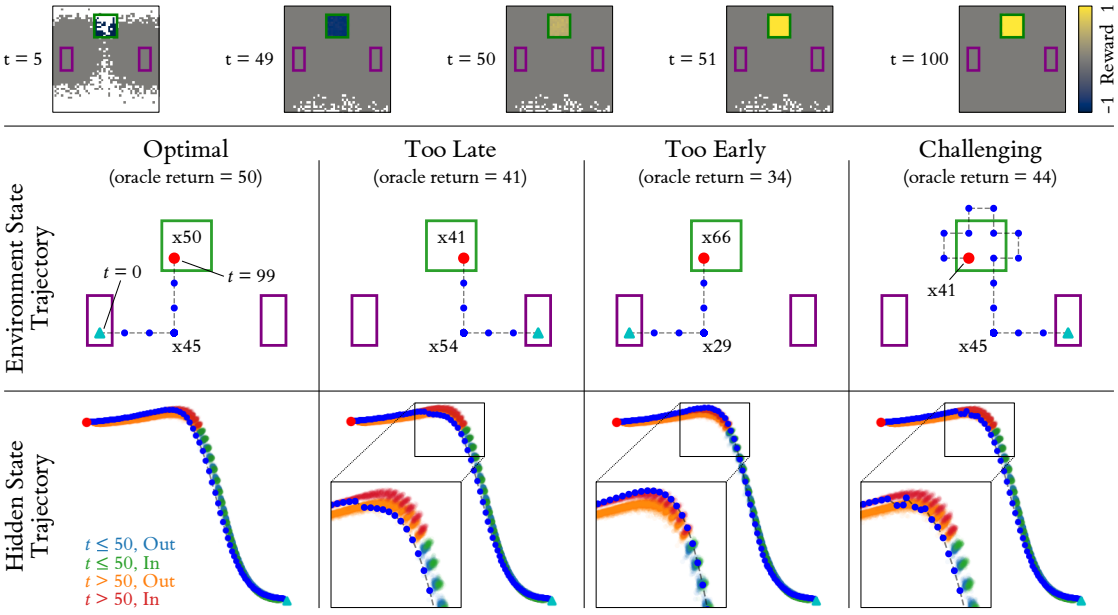

FIGURE 6.9: Timer task trajectory probing.

Top: Predicted reward with respect to time and position. We observe that the model has correctly captured the transition from negative to positive reward in the treasure region at $t = 50$, with no reward outside of this region. Although the reward is positive at $t = 50$, the model is uncertain at this point, i.e., the transition from negative to positive reward happens over two timesteps rather than one.

Middle/bottom: Four trajectory probes demonstrating the model's hidden state transitions. "xn" labels indicate the agent remaining in a position for n timesteps.

Optimal: The agent moves into the treasure region at $t = 50$ and remains there, receiving the maximum possible reward.

Too Late: The agent moves into the treasure region a while after the treasure has already become positive, i.e., it is missing out on reward by not being in the region for as long as possible. This is reflected in the hidden state plot, where the state transitions from the orange to the red region after the $t = 50$ boundary.

Too Early: The agent moves into the treasure region before the treasure becomes positive, therefore, while it earns the maximum amount of positive reward, it also earns negative reward, leading to a sub-optimal result.

Challenging: The agent moves into the treasure region at the correct time, but proceeds to jump in and out of the treasure region before settling, leading to lost reward. The hidden state trajectories somewhat mimic this movement by transitioning between the orange and red regions, although the jumps are less clear near the $t = 50$ transition point, suggesting the model is uncertain at this point.

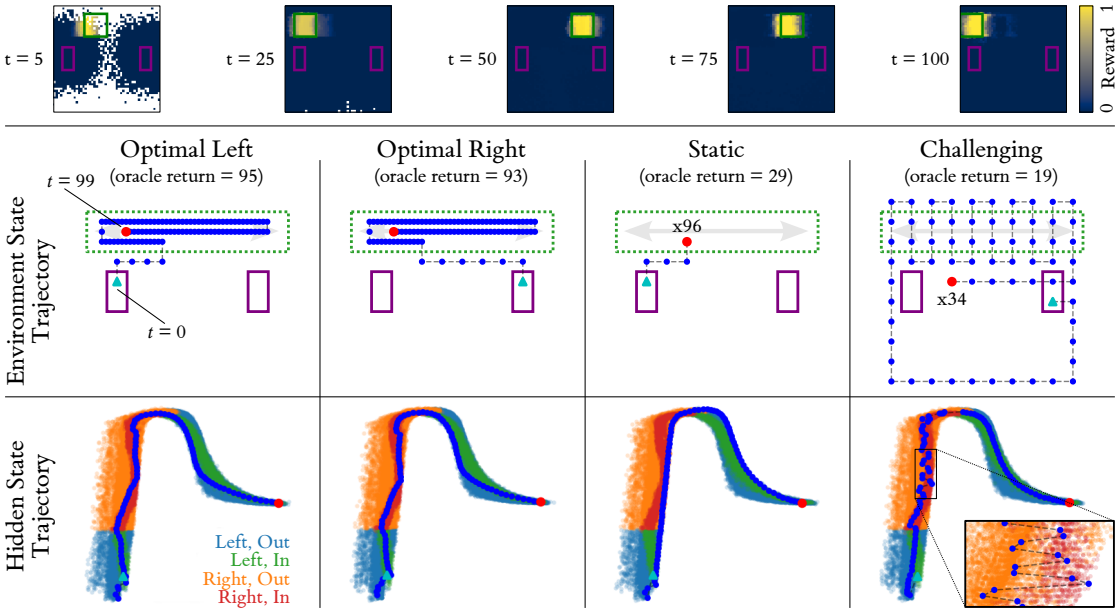

FIGURE 6.10: Moving task trajectory probing.

Top: The predicted reward with respect to time (and thus implicitly, the position of the treasure region). The model has learnt to track the treasure region as it moves, although there is noise around the left and right edges of the region, highlighting the difficulty of recovering the treasure region's horizontal position exactly.

Middle/bottom: Four trajectory probes showing the model's hidden state transitions. Note the green dotted region indicates the overall boundary of the treasure region, i.e., the treasure lies somewhere within that boundary, with its true horizontal position dependent on time.

Optimal Left: The agent moves within the treasure region as quickly as possible and then moves with the treasure (keeping within it) for the remainder of the episode. The hidden state trajectories follow the "In" regions (green and red).

Optimal Right: A similar optimal probe where the agent starts from the right-hand spawn zone rather than the left. In this case, the agent has further to move before moving into the treasure region, leading to slightly less overall reward than when it starts from the left.

Static: Rather than moving with the treasure region, the agent stays still and allows the treasure to pass over it, gaining reward for some timesteps but not others. We observe that the hidden state trajectory also reflects this — the state transitions between the "In" and "Out" regions.

Challenging: The agent takes a while to move towards the treasure region, and then passes in and out of the boundary in which the treasure resides, picking up some reward. We can see from the hidden state trajectory that this motion is captured in the hidden states — the trajectory transitions between the orange and red regions as the agent passes back and forth through the treasure region.

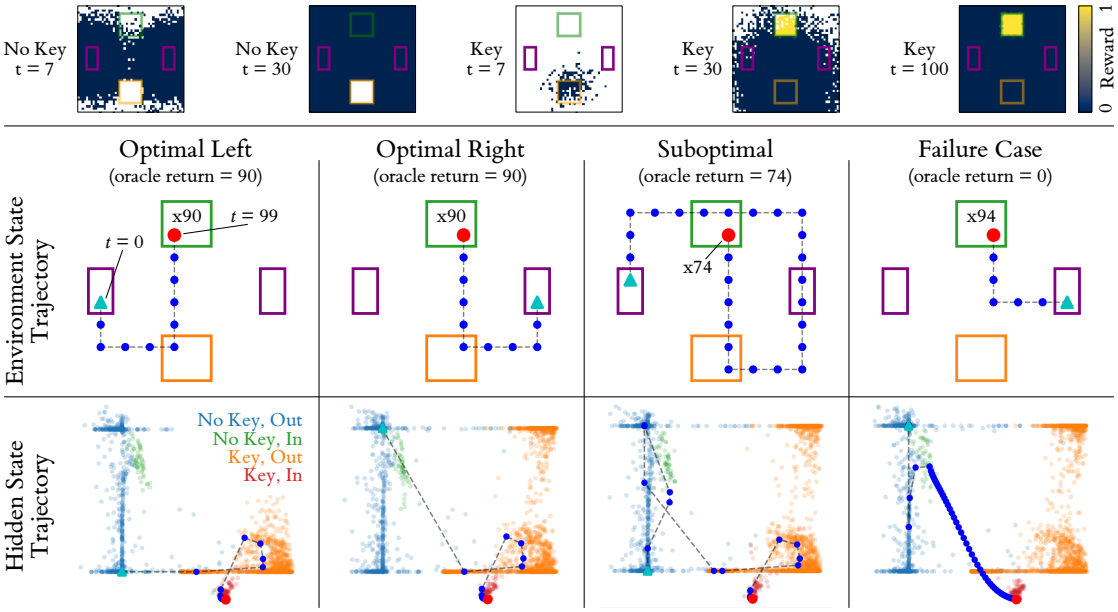

FIGURE 6.11: Key task trajectory probing.

Top: The predicted reward with respect to time and collection of the key. The model has learnt to only give reward when the agent is in the treasure region after collecting the key.

Middle/bottom: Four trajectory probes showing the model's hidden state transitions.

Optimal Left: The agent collects the key as quickly as possible, and then proceeds to move into the treasure region and wait there until the end of the episode. The hidden state trajectory shows two notable transitions, first when the key is collected (blue to orange) and then when the agent enters the treasure region (orange to red).

Optimal Right: A similar optimal policy, but from the right-hand spawn zone rather than the left. Note the hidden state trajectory is very similar, apart from the initial hidden state, which is at the opposite side of the blue region, representing the different spawn position.

Suboptimal: The agent passes through the treasure region before collecting the key, and then follows an optimal policy. We observe a transition in the hidden state trajectory from the blue to the green region that corresponds to the agent's premature entrance into the treasure region.

Failure case: The agent enters the treasure region without collecting the key and remains there for the rest of the episode. This highlights a failure case of the model, where the hidden state "drifts" from the green region to the red region, i.e., the model convinces itself that the agent must have picked up the key at some point. Such *causal confusion* errors are well-documented in the RM literature (Tien et al., 2022). In our case, we suspect that the error is due to a bias in our data generation method inducing a correlation between key possession and time spent in the treasure region. As described in Section 6.5.3, we specifically screen for trajectories where the treasure is visited with the key (the "treasure" class) but not for those where it is visited without it. There are thus likely to be very few training trajectories that spend a lot of time in the treasure region without collecting the key, making this probe an extreme outlier on which performance is poor. Adding and selecting for a fourth "key_no_treasure" class during data generation may have mitigated this issue.

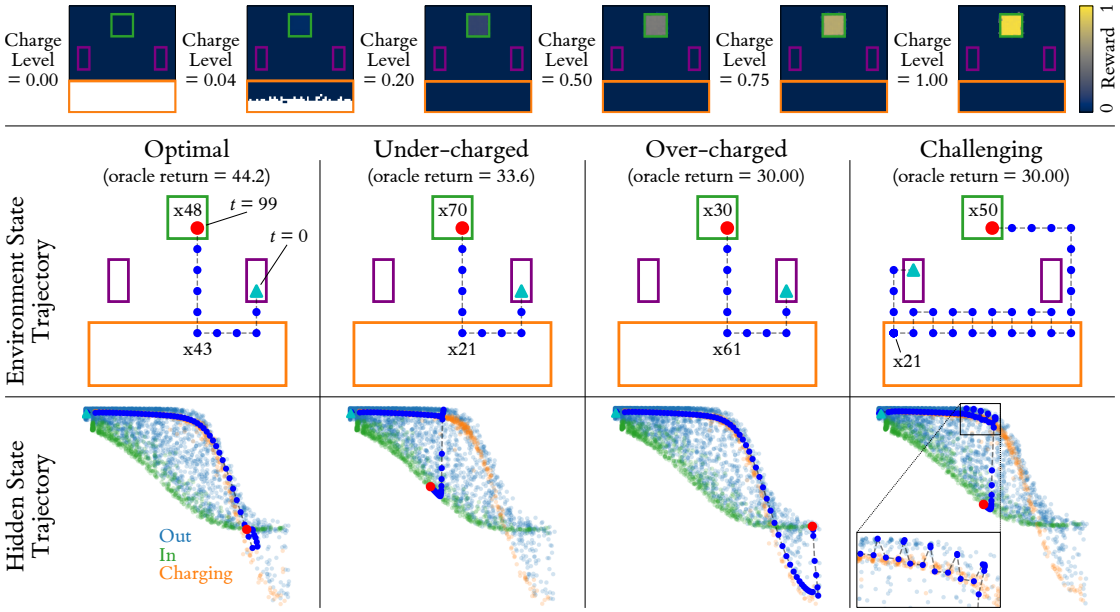

FIGURE 6.12: Charger task trajectory probing.

Top: the learnt relationship between agent position, charge level, and reward. The model has correctly learnt the relationships between position, charge, and reward (reward increases in the treasure zone as charge increases).

Middle/bottom: Four probe trajectories demonstrating hidden state transitions as the trajectory progresses.

Optimal: best possible return (charge to sufficient level then maximise time in treasure).

Under-charged: spending more time in the treasure zone but less time charging, which leads to sub-optimal return, i.e., it is better to spend less time in the treasure zone but with more charge.

Over-charged: continuing to charge after maximum charge of 1 is reached.

Challenging: The agent moves in and out of the charging zone before proceeding to the treasure zone. The learnt hidden states align with the agent moving in and out of the charging zone.

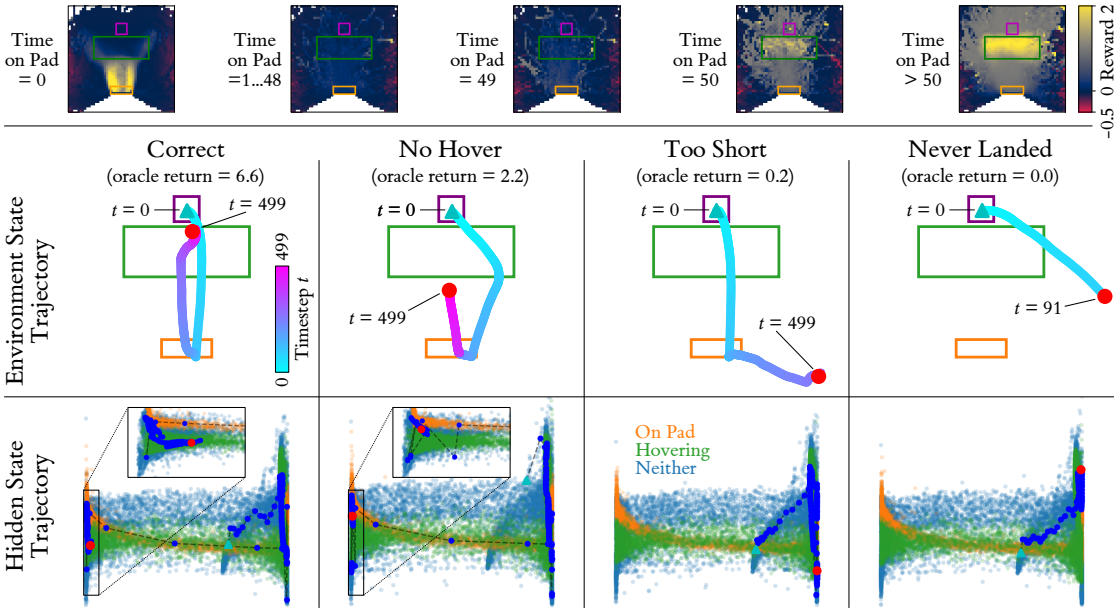

FIGURE 6.13: Lunar Lander task trajectory probing.

Top: The predicted reward with respect to the agent's position and the length of time that it has been on the pad. The model has learnt to reward the agent for landing on the pad when it has not previously landed, and then reward it for taking off and remaining in the hover zone after 50 timesteps on the pad, as per the oracle. However, observe how the top half of the hover zone is attributed higher reward than the lower half. This feature is not present in the oracle (which rewards the entire hover zone equally) but makes great practical sense, as a stochastic policy is less likely to drop out of the zone under the effect of gravity if it remains some distance from the bottom edge. Also note the small negative rewards near the environment boundaries, which disincentivise positions with a high risk of leading to early termination, and the "funnel" of intermediate positive reward, which may help to guide the agent up from the pad to the hover zone. Collectively, these deviations from the oracle reward function could actually be seen as *improvements*, and explain why we observe significantly higher performance on the R_{hover} reward component when the `CSCInstanceLSTM` reward model is used, compared with using the oracle itself.

Middle/bottom: Four trajectory probes showing the model's hidden state transitions.

Correct: The agent lands on the pad as quickly as possible, waits for 50 timesteps, and then takes off again and hovers in the hover zone. The hidden state trajectory shows a transition from right to left once the agent has been on the pad for long enough. It also clearly shows the agent is in the Hovering hidden state region.

No Hover: In this trajectory, the agent remains on the pad for too long and does not enter the hover zone. It has a similar hidden state transition to the *Correct* trajectory but does not enter the Hovering hidden state region.

Too Short: The agent lands on the pad but does not stay there for the required 50 timesteps. In this particular case, the agent lands correctly but then slips out of the landing pad, coming to rest on a different area of terrain until the end of the trajectory. In the hidden state trajectory, there is no transition from the right side to the left side, matching the idea that the agent has not remained on the pad for long enough.

Never Landed: The agent does not land on the pad at all. In this particular example, the agent reaches the boundary of the environment, which causes the episode to terminate early. Similar to the *Too Short* example, the hidden state trajectory does not transition from right to left.

6.6.3 Lunar Lander Multiple Instance Learning Analysis

In Table 6.4, we presented results for Lunar Lander using only the top five performing models by reward MSE (50% of all models). In this section, we discuss why this is the case.

When training the Lunar Lander architectures with linear activation functions in the HNs, we found that the models were struggling to learn the correct return predictions as they were making negative reward predictions. We know *a priori* that the Lunar Lander task only has positive rewards, therefore we added a Leaky ReLU activation (with a negative slope of 1×10^{-6}) to each architecture's HN to encourage positive predictions. This led to an immediate improvement in performance for all model architectures (excluding `Instance`, but this was expected to fail on this task as it cannot model temporal dependencies).

However, we found that a proportion of model initialisations remained unable to overcome a certain local minimum during training (corresponding to a return prediction MSE of around 2.1). In other cases, after this threshold was passed, the model performance would rapidly improve. Note this problem occurred in each of the LSTM-based models (`Instance` was never observed to pass this threshold). Therefore, we focus our evaluation on the top 50% of models, which is equivalent to discarding the worse-performing models, the majority of which had not passed the problematic threshold during training.

We investigate this training issue further in Figure 6.14, focusing specifically on `CSCInstanceLSTM` models. Although the Leaky ReLU activation encourages models to make positive predictions, we can see that it does not prevent them entirely. Furthermore, the models that are not able to cross the return threshold of 2.1 tend to output a greater proportion of negative reward predictions. We thus hypothesise that a better mechanism for preventing negative reward prediction would increase the chance that a given model training run achieves a return loss of less than 2.1. Below we list several such mechanisms as alternatives to our current Leaky ReLU approach.

Replace Leaky ReLU with ReLU One option to remove negative reward predictions entirely is to use ReLU rather than Leaky ReLU in the final activation function of the HNs. However, in the case that all the reward predictions are negative, the gradient of the network will be zero, so no learning can take place. The ability of the network to learn is entirely dependent on achieving at least one positive prediction in order to generate non-zero gradients, which is determined by the initial network weights. This could potentially be overcome with different network initialisation approaches, or by simply discarding training runs that fail to begin to learn.

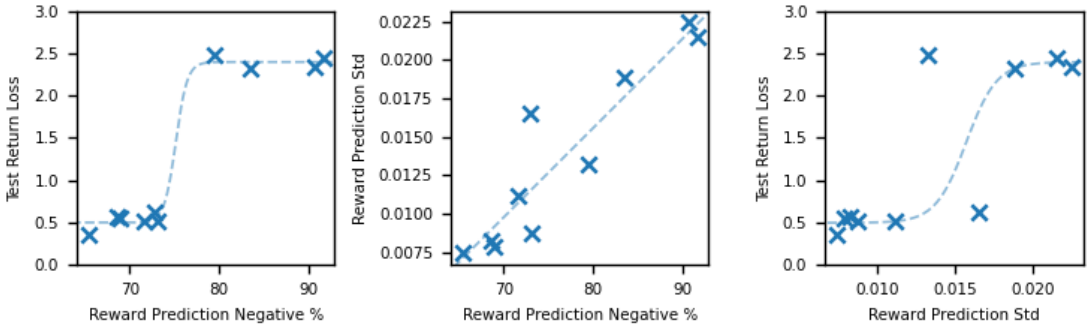

FIGURE 6.14: An analysis of the negative reward prediction of the ten Lunar Lander `CSCInstanceLSTM` models trained in this work. Note we observed similar trends for `EmbeddingLSTM` and `InstanceLSTM`.

Left: Models that are able to cross the return loss threshold of 2.1 make fewer negative reward predictions (given here as a percentage of all reward predictions) than those that cannot.

Middle: As the number of negative reward predictions increases, so too does the standard deviation of the reward predictions. In order to sum to the correct return predictions, the positive reward predictions must increase to compensate for the negative reward predictions, leading to a larger variance, and ultimately worse reward prediction.

Right: The standard deviation of the reward predictions also highlights the loss threshold of 2.1, although not as clearly as the percentage of negative reward predictions.

Replace Leaky ReLU with Sigmoid Instead of using a Leaky ReLU or ReLU activation in the HN, the Sigmoid activation function could be used. This would overcome the gradient issue presented with the use of ReLU and would ensure that no negative rewards are predicted. However, this would require *a priori* knowledge of the maximum reward target in order to scale the network outputs correctly, and the non-linear activation could make accurate prediction of rewards more difficult in some cases.

Remove target normalisation A further approach to decrease the number of negative reward predictions would be to remove or reduce the reward target scaling. As discussed above, this was initially included to avoid very large gradients. However, reducing this scaling would move the average reward prediction away from zero, potentially reducing the number of negative predictions. Care would have to be taken to not reintroduce the large gradient problem.

Regularise reward prediction variance A final alternative approach could be to introduce an additional loss term that encourages the reward predictions to have low variance. This would deter a large range of reward predictions, leading indirectly to fewer negative reward predictions (see Figure 6.14). However, this approach is only applicable to the LSTM models that produce instance predictions, i.e., it cannot be used with the `EmbeddingLSTM` as that architecture does not produce reward predictions

during training. Furthermore, this would result in an additional hyperparameter that requires tuning (a coefficient for the new loss term).

6.6.4 Limitations and Future Work

Although we analyse the performance of our methods in the presence of noisy labels in Section 6.5.5, a major area of future work is to apply our methods to human labelling (as discussed at the start of Section 6.5). Another area of future work involves more complex environments, for example, the use of tasks with image observations, similar to the Atari environments in OpenAI Gym (Brockman et al., 2016). Furthermore, we perform RM from either an offline dataset or from only one RL training iteration; an iterative bootstrapping approach with multiple RL + RM training iterations could lead to improved RL results. There are also limitations with our MIL RM approach for the Lunar Lander task; see Section 6.6.3 for details and suggestions for future work. More generally, we hope that our identification of the link between RM and MIL may inspire a bidirectional transfer of tools and techniques.

6.7 Conclusion

We posed the problem of non-Markovian RM, which removes an unrealistic assumption about how humans evaluate temporally extended agent behaviours. After identifying an isomorphism between RM and MIL, we proposed and evaluated novel MIL-inspired models that allow us to reconstruct non-Markovian reward functions, augment agent training, and interpret their learnt representations.

In the next research chapter, we remain working on sequential data, but change the application domain to Time Series Classification (TSC). Note in this chapter we focused on MIL regression, but the next chapter uses multi-class classification problems. As well as changing the domain, the next chapter has a greater focus on "*plug-and-play*" methods: integrating MIL into existing State-Of-The-Art (SOTA) methods to enhance interpretability and predictive performance.

Chapter 7

Inherently Interpretable Time Series Classification via Multiple Instance Learning

This chapter presents a piece of work published as a spotlight paper (top 5%) at ICLR 2024. The work was completed during my internship with Amazon Prime Video (March to September 2023). It was done in collaboration with a team at Amazon, in which I led the research and was the lead author of the paper. The co-authors were Gavin Cheung (mentor), Kurt Cutajar, Hanting Xie, Jas Kandola, and Niall Twomey (supervisor), who all provided supervision and manuscript feedback — all technical development and design was done by myself. Appropriate permission has been given by Amazon to include it in this thesis. The content is mostly unchanged from its original format: elements of the original appendix have been brought into the main body, certain sections have been slightly reworded, and some terminology/notation has been modified for consistency with the rest of this thesis.

Paper

Joseph Early, Gavin KC Cheung, Kurt Cutajar, Hanting Xie, Jas Kandola, and Niall Twomey. Inherently interpretable time series classification via multiple instance learning. *Internation Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xriGRsoAza

GitHub: https://github.com/JAEarly/MILTimeSeriesClassification

7.1 Introduction

Time Series Classification (TSC) is the process of assigning labels to sequences of data and occurs in a wide range of settings — examples from the popular UCR collection of datasets include predicting heart failure from electrocardiogram data and identifying household electric appliance usage from electricity data (Dau et al., 2019). Each of these domains has its own set of class-conditional discriminatory signatures that determine the class of a time series. Deep Learning (DL) methods have emerged as a popular family of approaches for solving TSC problems. However, we identify two drawbacks with these conventional supervised learning approaches: 1) representations are learnt for each time point in a time series, but these representations are then lost through an aggregation process that weights all time points equally, and 2) these methods are *black boxes* that provide no inherent explanations for their decision making, i.e., they cannot localise the class-conditional discriminatory signatures. These drawbacks not only limit predictive performance but also introduce barriers to their adoption in practice as the models are not transparent.

To mitigate these shortcomings, we take an alternative view of DL for TSC, approaching it as a Multiple Instance Learning (MIL) problem. MIL is a weakly supervised learning paradigm in which a collection (*bag*) of elements (*MIL instances*) all share the same label. In the context of TSC, a bag is a time series of data over a contiguous interval. In the MIL setting, the learning objective is to assign class labels to unlabelled bags of time series data whilst also discovering the salient signatures within the time series that explain the reasons for the predicted class. As we explore in this work, MIL is well-suited to overcome the drawbacks identified above, leading to inherent interpretability without compromising predictive performance (even improving it in some cases). We propose a new general framework applying MIL to TSC called `MILLET`: **M**ultiple **I**nstance **L**earning for **L**ocally **E**xplainable **T**ime series classification. Demonstrative `MILLET` model outputs are depicted in Figure 7.1.

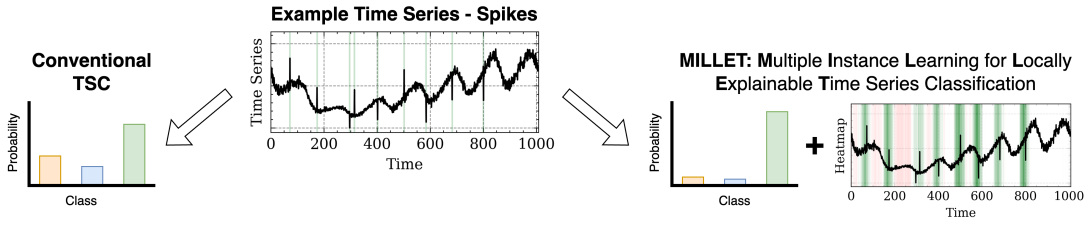

FIGURE 7.1: Conventional TSC techniques (left) usually only provide class-level predictive probabilities. In addition, our proposed method (`MILLET`, right) also highlights class-conditional discriminatory signatures that influence the predicted class. In the heatmap, green regions indicate support for the predicted class, red regions refute the predicted class, and darker regions are more influential.

MIL is well-suited to this problem setting since it was developed for weakly supervised contexts, can be learnt in an end-to-end framework, and boasts many successes across

several domains. Furthermore, MIL has the same label specificity as TSC: labels are given at the bag level, but not at the MIL instance level. To explore the intersection of these two areas, we propose *plug-and-play* concepts that are adapted from MIL and applied to existing TSC approaches (in this work, DL models[1]). Furthermore, to aid in our evaluation of the interpretability of these new methods, we introduce a new synthetic TSC dataset, *WebTraffic*, where the locations of the class-conditional discriminatory signatures within time series are known. The time series shown in Figure 7.1 is sampled from this dataset.

Our key contributions are as follows:

1. We propose `MILLET`, a TSC framework that utilises MIL to provide inherent interpretability without compromising predictive performance (even improving it in some cases).
2. We design *plug-and-play* MIL methods for TSC within `MILLET`.
3. We propose a new method of MIL aggregation, `Conjunctive pooling`, that outperforms existing pooling methods in our TSC experiments.
4. We propose and evaluate 12 novel `MILLET` models on 85 univariate datasets from the UCR TSC Archive (Dau et al., 2019), as well as a novel synthetic dataset that facilitates better evaluation of TSC interpretability.

The remainder of this chapter is organised as follows. We first cover relevant background and related work (Section 7.2), and then propose our novel methodology for TSC using MIL (Section 7.3). This is followed by two sets of experiments: the first on a synthetic dataset (Section 7.4), and the second on 85 benchmark datasets (Section 7.5). Finally, the chapter ends with a discussion (Section 7.6) and conclusion (Section 7.7).

7.2 Background and Related Work

In this section, we outline the relevant background for our work. We first briefly introduce TSC (Section 7.2.1) and MIL (Section 7.2.2), and then give an overview of existing interpretability approaches for TSC (Section 7.2.3).

7.2.1 Time Series Classification

While a range of TSC methods exist, in this work we apply MIL to DL TSC approaches. Methods in this family are effective and widely used (Foumani et al., 2023; Ismail Fawaz

[1]While we focus on DL TSC in this work, we envision that our `MILLET` framework can be applied to other TSC approaches in the future, such as the `ROCKET` family of methods (Dempster et al., 2020, 2023).

et al., 2019); popular methods include Fully Convolutional Networks (`FCN`), Residual Networks (`ResNet`), and `InceptionTime` (Ismail Fawaz et al., 2020; Wang et al., 2017). Indeed, a recent TSC survey, *Bake Off Redux* (Middlehurst et al., 2024), found `InceptionTime` to be competitive with State-Of-The-Art (SOTA) approaches such as the ensemble method HIVE-COTE 2 (`HC2`; Middlehurst et al., 2021) and the hybrid dictionary-convolutional method Hydra-MultiRocket (`Hydra-MR`; Dempster et al., 2023). Although the application of Matrix Profile for TSC also yields inherent interpretability (Guidotti and D'Onofrio, 2021; Yeh et al., 2017), we choose to focus on DL approaches due to their popularity, strong performance, and scope for improvement (Middlehurst et al., 2024).

7.2.2 Multiple Instance Learning

In its standard assumption, MIL is a binary classification problem: a bag is positive if and only if at least one of its instances is positive (Dietterich et al., 1997). As we are designing `MILLET` to be a general and widely applicable TSC approach, we do not constrain it to any specific MIL assumption except that there are temporal relationships, i.e., the order of instances within bags matters (Early et al., 2022a; Wang et al., 2020a, Chapter 6). As we explore in Section 7.3.5, this allows us to use positional encodings in our `MILLET` methods. Although the application of MIL to TSC has been explored prior to this study, earlier work focused on domain-specific problems such as intensive care in medicine and human activity recognition (Dennis et al., 2018; Janakiraman, 2018; Poyiadzi et al., 2018; Poyiadzis et al., 2019; Shanmugam et al., 2019). Furthermore, existing work considers MIL as its own unique approach separate from existing TSC methods. The work most closely related to ours is Zhu et al. (2021), which proposes an uncertainty-aware MIL TSC framework specifically designed for long time series (marine vessel tracking), but without the generality and *plug-and-play* nature of `MILLET`. Therefore, to the best of our knowledge, our work with `MILLET` is the first to apply MIL to TSC in a more general sense and to do so across an extensive variety of domains.

7.2.3 Interpretability

TSC interpretability methods can be grouped into several categories. Following the taxonomy of Theissler et al. (2022), we focus on class-wise time point attribution (saliency maps), i.e., identifying the discriminatory time points in a time series that support and refute different classes. Our proposed method, `MILLET`, is one such method. Time point-based interpretations are a form of local interpretation, where model decision-making is explained for individual time series (Molnar, 2020). It also aligns with MIL interpretability as proposed by Early et al. (2022c, Chapter 4): *which* are the key MIL instances in a bag, and *what* outcomes do they support/refute? `MILLET`

facilitates interpretability by inherently enhancing existing TSC approaches such that they provide interpretations alongside their predictions with a single forward pass of the model.

There are a range of existing interpretability methods that are not specific to TSC. Comparable methods to MILLET (i.e., those that provide time point-based interpretations), include Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016a), Shapley Additive Explanations (SHAP) (Lundberg and Lee, 2017), Occlusion Sensitivity (Zeiler and Fergus, 2014), and MILLI (Early et al., 2022c, Chapter 4). However, these are often very expensive to run (requiring 100+ forward passes per interpretation). To overcome the computational cost of these methods, an alternative set of approaches is subsequence-based techniques, which provide interpretations at a lower granularity. However, we do not consider these comparable approaches because they do not produce time point-based interpretations, i.e., we compare against interpretability methods with the same level of granularity as MILLET.

TSC-specific interpretability approaches have also been proposed in prior work. Many of these have one or both of the issues mentioned above: they are expensive to compute and/or reduce computational complexity by grouping time points together (so are subsequence, not time point interpretability methods). This includes time series-specific SHAP approaches such as TimeSHAP (Bento et al., 2021) and WindowSHAP (Nayebi et al., 2023), and time series-specific LIME approaches such as LIMESegements (Sivill and Flach, 2022). Further methods include Dynamask (Crabbé and Van Der Schaar, 2021), WinIT (Leung et al., 2022), and TimeX (Queen et al., 2023).

Based on the requirements outlined above, we choose two interpretability methods to compare MILLET against. The first is Class Activation Mapping (CAM) (Wang et al., 2017; Zhou et al., 2016), which uses the model's weights to identify discriminatory time points. It is comparable to MILLET as it produces interpretations with only a single forward pass of the model. The second is SHAP, which is chosen as it does not need careful tuning of hyperparameters for individual datasets (which other methods often do). This means it can be easily applied to the large number of datasets used in this work. A further piece of work that is worth mentioning is Temporal Saliency Rescaling (Ismail et al., 2020), which has been shown to improve time point-based interpretability methods in multivariate settings. While not applicable in this work as we only study univariate time series, it could be applied on top of MILLET if future work extends MILLET to multivariate settings. For reviews of existing TSC interpretability methods, see Theissler et al. (2022) and Šimić et al. (2021).

7.3 Methodology

To apply MIL to TSC, we propose the broad framework `MILLET`: **M**ultiple **I**nstance **L**earning for **L**ocally **E**xplainable **T**ime series classification. We advocate for the use of MIL in TSC as it is a natural fit that provides inherent interpretability (explanations for free) without requiring any additional labelling beyond that already provided by existing TSC datasets. We first further discuss why MIL is appropriate for TSC (Section 7.3.1), and then introduce our `MILLET` framework (Section 7.3.2). This is followed by details on our approach for applying `MILLET` to DL methods (Section 7.3.3), `MILLET` interpretability (Section 7.3.4), model design (Section 7.3.5), and evaluation metrics (Section 7.3.6).

7.3.1 Why Multiple Instance Learning?

To answer the question of "why MIL", we first consider the requirements for interpreting TSC models. Our underlying assumption is that, for a classifier to predict a certain class for a time series, there must be one or more underlying signatures (a single time point or a collection of time points) in the time series that the model has identified as being indicative of the predicted class. In other words, only certain time points within a time series are considered discriminatory by the model (those that form the signatures), and the other time points are ignored (background time points or noise). Note the values and position of the discriminatory time points within a time series can differ within in a collection of time series. This could also be extended to class refutation, where the model has learnt that certain signatures indicate that a time series does not belong to a particular class. To this end, the objective of TSC interpretability is then to uncover these supporting and refuting signatures, and present them as an explanation as to why a particular class prediction has been made.

We next consider how we could train a model to find these signatures if we already knew where and what they were, i.e., in a conventional supervised learning sense where the signatures are labelled. In this case, we would be able to train a model to predict a signature label from an input signature — in the simplest case this could be predicting whether a signature is discriminatory or non-discriminatory. This model could then be applied to an unlabelled time series and used to identify its signatures. Effectively, this hypothetical signature model is making time point level predictions.

Unfortunately, it is very difficult to train models in the above manner. The vast majority of time series datasets are only labelled at the time series level — no labels are provided at the time point level, therefore the locations of signatures are unknown. While there are several successful DL models that can learn to make time-series level predictions from time-series level labels (e.g `FCN`, `ResNet`, `InceptionTime`), they are *black boxes* —

they provide no inherent interpretation in the form of supporting/refuting signatures via time point level predictions. Without time point level labels, conventional supervised learning is unable to develop models that inherently make time point level predictions. While post-hoc interpretability methods could be used to uncover the signatures, they can be expensive (in the case of LIME and SHAP), or as we show in this work, inferior to inherent interpretability. As such, we can draw a set of requirements for a new and improved interpretable TSC approach:

1. **Inherent interpretability:** The model should provide time point predictions (i.e., signature identification) as part of its time series prediction process. This means interpretations are gained effectively for free as a natural byproduct of time series prediction. It also means expensive or ineffective post-hoc interpretability approaches are not required.
2. **Learn from time series labels:** As stated above, TSC datasets rarely provide time point-level labels. Therefore, the model must be able to learn something insightful at the time point level given only time series labels.
3. **Provide a unified framework:** As there is a diverse range of existing TSC methods of different families, an effective TSC interpretability approach should be as widely applicable as possible to facilitate continued research in these different areas. This would also mean existing bodies of research can be applied in conjunction with the framework.

Given these requirements, we advocate for MIL as an appropriate method for an inherently interpretable TSC framework. MIL is designed for settings with bags of instances, where only the bags are labelled, not the instances. In a TSC setting, this equates to having time series labels without time point labels, which is exactly what we outlined above (and describes the vast majority of TSC datasets). Furthermore, certain types of existing MIL approaches, for example, `Instance` and `Additive`, work by first making a prediction for every time point, and then aggregating over these predictions to make a time series prediction. This is inherently interpretable and facilitates signature identification. Finally, the over-arching concept of MIL for TSC, i.e., learning from time series labels but making both time point and time series predictions, is not tied to any one family of Machine Learning (ML) approach.

We also consider answers to potential questions about alternative solutions:

1. **Q: Why not label the signatures/time points to allow for supervised learning?**
   A: As discussed above, it is rare for a time series dataset to have time point labels. It could be possible to label the signatures, for example by having medical practitioners identify discriminatory irregular patterns in ECG data. However,

this labelling process is very time-consuming, and in some cases would require domain expertise, which is challenging and expensive to acquire. Furthermore, it would then require anyone interested in applying such supervised techniques to their own data to fully label everything at the time point level, increasing the cost and time of developing new datasets.

2. **Q: Why not label *some* of the signatures/time points and use a semi-supervised approach?**
   A: While it might be possible to train a semi-supervised model that only requires some of the time points to be labelled, the model is no longer end-to-end. As the model only learns to predict at the time point level, it does not provide time series level predictions itself. Rather, some additional process is required to take the time point predictions and transform them into a time series prediction. Furthermore, a semi-supervised model has no method for leveraging both the time series labels and the partial time point labels.

3. **Q: What does MIL achieve beyond methods such as attention?**
   A: While attention has been utilised previously in TSC, it does not provide class-specific interpretations, only general measures of importance across all classes. So while attention might identify signatures, it does not state which class these signatures belong to, nor whether they are supporting or refuting. Furthermore, attention is far less explicit than time point predictions — there is no guarantee that attention actually reflects the underlying signatures of decision-making, whereas in MIL the time point predictions directly determine the time series prediction.

Given the above motivation for using MIL in TSC, in the next section, we propose our novel `MILLET` framework.

7.3.2 The `MILLET` Framework

A TSC model within our `MILLET` framework has to satisfy three requirements:

Requirement 1: Time Series as MIL Bags A TSC dataset consists of time series, where each time series $\mathbf{X}$ is formed of $T > 1$ time points: $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$.[2] Each time step is an h-dimensional vector, where h is the number of channels in the time series. In this work, we focus on univariate time series ($h = 1$) and assume all time series in a dataset have the same length. We consider each time series as a MIL bag, meaning each

[2]Following convention from MIL, we use uppercase variables to denote MIL bag / time series data and lowercase variables to denote MIL instance / time point data. The time series length T is equivalent to the number of instances k in general MIL.

time point is a MIL instance.[3] A time series bag can be considered a matrix $\mathbf{X} \in \mathbb{R}^{T \times h}$, and each bag has an associated bag-level label $\mathbf{Y}$ which is the original time series label. There is also the concept of MIL instance labels $\{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$, but these are not provided for most MIL datasets (like the absence of time point labels in TSC). Framing TSC as a MIL problem allows us to obtain interpretability by imposing the next requirement.

Requirement 2: Time Point Predictions To facilitate interpretability in our framework, we specify that models must provide time point predictions along with their time-series predictions. Furthermore, the time point predictions should be inherent to the model — this makes it possible to identify which time points support and refute different classes without having to use post-hoc methods.

Requirement 3: Temporal Ordering TSC is a sequential learning problem so we impose a further requirement that the framework must respect the ordering of time points. This is in contrast to classical MIL methods that assume MIL instances are iid.

7.3.3 `MILLET` for Deep Learning TSC: Rethinking Pooling

To demonstrate the use of `MILLET`, we apply it to DL TSC methods. Existing DL TSC architectures (e.g., `FCN`, `ResNet`, and `InceptionTime`) mainly consist of two modules: a Feature Extractor (FE) ψ_{FE} (we refer to these as backbones) and a classifier ψ_{CLF}. For an input univariate time series $\mathbf{X}$, ψ_{FE} produces a set of d-dimensional feature embeddings $\mathbf{Z} \in \mathbb{R}^{T \times d} = [\mathbf{z}_1, \ldots, \mathbf{z}_T]$ (Equation 7.1). These embeddings are consolidated via aggregation with Global Average Pooling (GAP) to give a single feature vector of length d. This is then passed to ψ_{CLF} to produce a prediction $\hat{\mathbf{Y}} \in \mathbb{R}^C$ for the time series, where C is the number of classes:

$$\text{Feature Extraction: } \mathbf{Z} = \psi_{FE}(\mathbf{X}); \tag{7.1}$$

$$\text{GAP + Classification: } \hat{\mathbf{Y}} = \psi_{CLF}\left(\frac{1}{T}\sum_{t=1}^{T}\mathbf{z}_t\right). \tag{7.2}$$

The specification of ψ_{FE} naturally satisfies Req. 1 from our `MILLET` framework as discriminatory information is extracted on a time point level. Req. 3 is satisfied as long as the DL architecture makes use of layers that respect the ordering of the time series such as convolutional or recurrent layers. In the MIL domain, the GAP + Classification

[3] There is an overlap in TSC and MIL terminology: both use the term 'instance' but in different ways. In MIL it denotes an element in a bag, and in TSC it refers to an entire time series (e.g., "instance-based explanations" from Theissler et al., 2022). To avoid confusion, we use 'time series' to refer to entire time series (a TSC instance) and 'time point' to refer to a value for a particular step in a time series (a MIL instance).

process (Equation 7.2) is known as mean `Embedding` pooling, as used in methods such as MI-Net (Wang et al., 2018). However, this aggregation step does not inherently produce time point class predictions, and consequently does not fit Req. 2.

To upgrade existing DL TSC methods into the `MILLET` framework and satisfy Req. 2, we explore four MIL pooling methods for replacing GAP. `Attention`, `Instance`, and `Additive` are inspired by existing MIL approaches, while `Conjunctive` is proposed in this work. Replacing GAP in this way is *plug-and-play*, i.e., any TSC method using GAP or similar pooling can easily be upgraded to one of these methods and meet the requirements for `MILLET`.

`Attention pooling` (Ilse et al., 2018) does weighted averaging via an attention head ψ_{ATTN}:

$$a_t \in [0,1] = \psi_{ATTN}(\mathbf{z}_t); \qquad \hat{\mathbf{Y}} = \psi_{ATTN}\left(\frac{1}{T}\sum_{t=1}^{T} a_t \mathbf{z}_t\right). \tag{7.3}$$

`Instance pooling` (Wang et al., 2018) makes a prediction for each time point:

$$\hat{\mathbf{y}}_t \in \mathbb{R}^C = \psi_{CLF}(\mathbf{z}_t); \qquad \hat{\mathbf{Y}} = \frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{y}}_t. \tag{7.4}$$

`Additive pooling` (Javed et al., 2022) is a combination of `Attention` and `Instance`:

$$a_t = \psi_{ATTN}(\mathbf{z}_t); \qquad \hat{\mathbf{y}}_t = \psi_{CLF}(a_t \mathbf{z}_t); \qquad \hat{\mathbf{Y}} = \frac{1}{T}\sum_{t=1}^{T} \hat{\mathbf{y}}_t. \tag{7.5}$$

`Conjunctive pooling` is our proposed novel pooling approach, where attention and classification are independently applied to the time point embeddings, after which the attention values are used to scale the time point predictions. This is expected to benefit performance as the attention and classifier heads are trained in parallel rather than sequentially, i.e., the classifier cannot rely on the attention head to alter the time point embeddings prior to classification, making it more robust. We use the term `Conjunctive` to emphasise that, from an interpretability perspective, a discriminatory time point must be considered important by both the attention head *and* the classification head. Formally, `Conjunctive` is described as:

$$a_t = \psi_{ATTN}(\mathbf{z}_t); \qquad \hat{\mathbf{y}}_t = \psi_{CLF}(\mathbf{z}_t); \qquad \hat{\mathbf{Y}} = \frac{1}{T}\sum_{t=1}^{T} a_t \hat{\mathbf{y}}_t. \tag{7.6}$$

Figure 7.2 shows a schematic representation comparing these pooling approaches with `Embedding` (GAP). Note there are other variations of these MIL pooling methods, such

as replacing mean with max in `Embedding` and `Instance`, but these alternative approaches are not explored in this work.

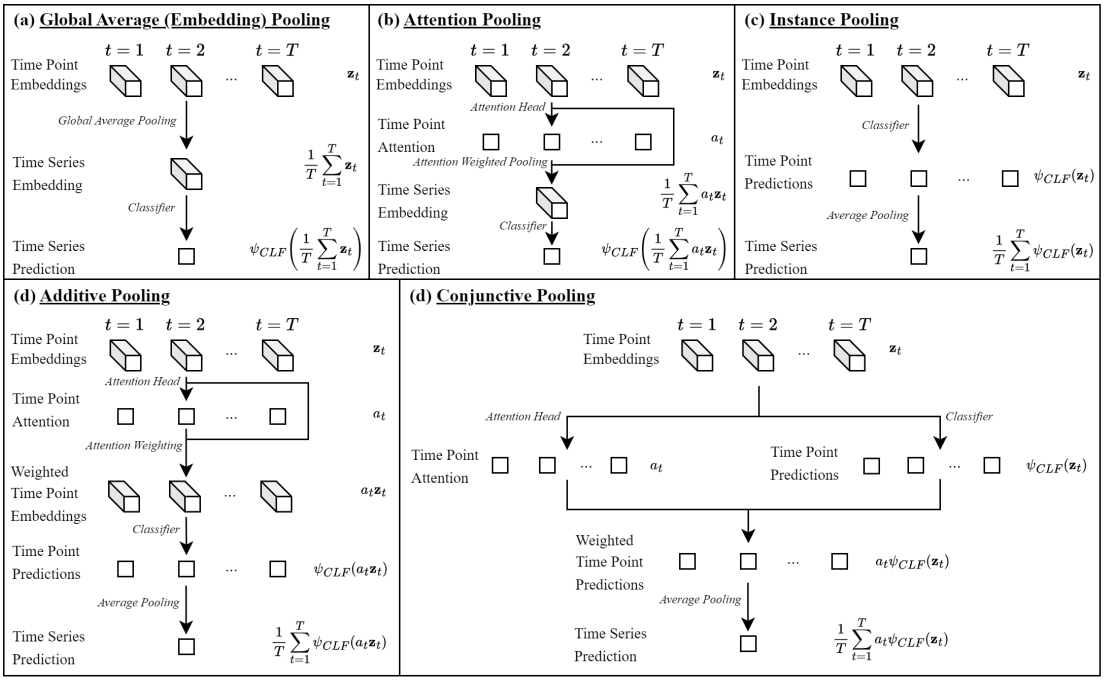

FIGURE 7.2: The five different MIL pooling methods used in this work. Each takes the same input: a bag of time point embeddings $\mathbf{Z} \in \mathbb{R}^{T \times d} = [\mathbf{z}_1, \ldots, \mathbf{z}_T]$. While they all produce the same overall output (a time series prediction), they produce different interpretability outputs.

7.3.4 `MILLET` Deep Learning Interpretability

As a result of Requirement 2 in Section 7.3.2, we expect the models to be inherently interpretable. For DL methods, this is achieved through the MIL pooling methods given in Section 7.3.3 — different MIL pooling approaches provide alternative forms of inherent interpretability. `Instance` performs classification before pooling (see Equation 7.4), so it produces a set of time point predictions $[\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_T] \in \mathbb{R}^{T \times C}$. `Additive` and `Conjunctive` also make time point predictions, but include attention. To combine these two outputs, we weight the time point predictions by the attention scores: $[a_1\hat{\mathbf{y}}_1, \ldots, a_T\hat{\mathbf{y}}_T] \in \mathbb{R}^{T \times C}$.

On the other hand, `Attention` is inherently interpretable through its attention weights $[a_1, \ldots, a_T] \in [0, 1]^T$. These attention values can be interpreted as a measure of importance for each time point, but unlike `Instance`, `Additive`, and `Conjunctive`, the interpretability output for `Attention` is not class-specific (but only a general measure of importance across all classes).

7.3.5 MILLET Deep Learning Model Design

We design three MILLET DL models by adapting existing backbone models that use GAP: FCN, ResNet, and InceptionTime. While extensions of these methods and other DL approaches exist (see Foumani et al., 2023), we do not explore these as none have been shown to outperform InceptionTime (Middlehurst et al., 2024). Nevertheless, the MILLET framework can be applied to any generic DL TSC approach that uses GAP or follows the high-level structure in Equation 7.2.

Replacing GAP with one of the four pooling methods in Section 7.3.3 yields a total of 12 new models. In each case, the backbone models produce feature embeddings of length $d = 128$ $\left(\mathbf{Z} \in \mathbb{R}^{T \times 128}\right)$. The models are trained end-to-end in the same manner as the original backbone methods — we discuss additional options for training in Section 7.7. We introduce three further enhancements:

1. **Positional Encoding:** As time point classification and attention are applied to each time point independently, the position of a time point within the times series can be utilised (with GAP, positional encoding would be lost through averaging) — this allows for further expressivity of the ordering of time points and enforces Req. 3 of our MILLET framework. We inject fixed sinusoidal positional encodings (Vaswani et al., 2017) after feature extraction (see Appendix D.2.2 for more details). Future work could investigate using learnt rather than fixed encodings. While the convolutional layers in the deep learning backbones used in this work facilitate relative temporal dependencies (helping to identify phase-independent features), injecting positional encodings allows for absolute positional encoding (helping to identify phase-dependent features). For the MIL pooling methods, our intuition is that the use of positional encodings facilitates more accurate time point prediction and attention scores, which benefits both interpretability and predictive performance.

2. **Replicate padding:** Zero padding is used in the convolutional layers of the backbone architectures. However, in our interpretability experiments, we found this biased the models towards the start and end of the time series — padding with zeros was creating a false signal in the time series. As such, we replaced zero padding with replicate padding (padding with the boundary value) which alleviated the start/end bias. However, we note that particular problems may benefit from other padding strategies.

3. **Dropout:** To mitigate overfitting in the new pooling methods, we apply dropout after injecting the positional encodings ($p = 0.1$). This is applied in all the MILLET pooling methods; after positional encoding but before pooling and classification (see Appendix D.2 for more details). No dropout was used in the original backbones.

The original `InceptionTime` approach is an ensemble of five identical network architectures trained from different initialisations, where the overall output is the mean output of the five networks. To facilitate a fair comparison, we use the same approach for `FCN`, `ResNet`, and `MILLET`. See Appendix D.1 for implementation details, and Appendix D.2 for model, training, and hyperparameter details.

7.3.6 Interpretability Evaluation Metrics

To quantitatively evaluate interpretability, we use the same process as Early et al. (2022c, Chapter 4; redefined below for clarity). This approach uses ranking metrics, i.e., looking at the predicted importance order rather than actual interpretation values. The two metrics used are Area Over the Perturbation Curve to Random (AOPCR) and Normalised Discounted Cumulative Gain at n (NDCG@n). The former is used to evaluate without time point labels, and the latter is used to evaluate with time point labels. Different metrics are used in existing TSC interpretability works, but these come with their own set of disadvantages. Metrics such as Area Under the Precision Curve (AUPC) and Area Under the Recall Curve (AURC), as used by Crabbé and Van Der Schaar (2021); Ismail et al. (2020); Leung et al. (2022); Queen et al. (2023), are useful in that they separate whether the identified time points are indeed discriminatory (precision) from whether all discriminatory time points are identified (recall/coverage). While this would be beneficial in evaluating the effect of sparsity, it does not account for the ordering of time points in the interpretation, i.e., we want to reward the interpretation method for placing discriminatory time points earlier in the ordering (and punish it for placing non-discriminatory time points earlier on); this is something that NDCG@n achieves. The mean rank metric (Leung et al., 2022) also suffers from this issue — it is effectively an unweighted version of NDCG@n. Furthermore, AUPC and AURC metrics require time point labels and sometimes need access to the underlying data generation process for resampling; AOPCR does not need either. NDCG@n can be used for our synthetic dataset, *WebTraffic* (Section 7.4), as we know the locations of discriminatory time points. However, for the UCR datasets used in Section 7.5, the discriminatory time points are unknown, therefore only AOPCR can be used. Below we provide more details on the metrics used to evaluate interpretability.

AOPCR: Evaluation without time point labels When time point labels are not present, the model can be evaluated via perturbation analysis. The underlying intuition is that, given a correct ordering of time point importance in a time series, iteratively removing the most important (discriminatory) time points should cause the model prediction to rapidly decrease. Conversely, a random or incorrect ordering will lead to a much slower decrease in prediction. Formally, when evaluating the interpretations generated by a classifier F for a time series $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ with respect to class c, we

first re-order the time series according to the importance scores (with the most important time points first): $\mathbf{O}_{\mathbf{X},c} = \{\mathbf{o}_1, \mathbf{o}_2, \ldots, \mathbf{o}_T\}$. Note the contents of $\mathbf{O}_{\mathbf{X},c}$ is the same as that of $\mathbf{X}$ but with a different order of instances. The perturbation metric is then calculated by:

$$AOPC(\mathbf{O}_{\mathbf{X},c}) = \frac{1}{T-1} \sum_{t=1}^{T-1} F_c(\mathbf{O}_{\mathbf{X},c}) - F_c(\mathbf{O}_{\mathbf{X},c} \setminus \{\mathbf{o}_1, \ldots, \mathbf{o}_t\}). \tag{7.7}$$

Note when computing $F_c(\mathbf{O}_{\mathbf{X},c})$ and $F_c(\mathbf{O}_{\mathbf{X},c} \setminus \{\mathbf{o}_1, \ldots, \mathbf{o}_t\})$, the time points remain in the same order as in $\mathbf{X}$, and the positional embeddings respect the original position of the time points. In Equation 7.7, the perturbation curve is calculated by removing individual time points and continues until all but one time point (the least important as assessed by the model) is left. This is expensive to compute, as a call to the model must be made for each perturbation. To improve the efficiency of this calculation, we group time points together into blocks equal to 5% of the total time series length, and only perturb the time series until 50% of the time points have been removed. As such, we only need to make 10 calls to the model per time series evaluation.

To facilitate better comparison between models, we normalise by comparing to a random ordering. To compensate for the stochastic nature of using random orderings, we average over several different random orderings:

$$AOPCR(\mathbf{O}_{\mathbf{X},c}) = \frac{1}{n_{Rand}} \sum_{r=1}^{n_{Rand}} \Big(AOPC(\mathbf{O}_{\mathbf{X},c}) - AOPC(\mathbf{O}_{\mathbf{X},Rand}) \Big), \tag{7.8}$$

where $\mathbf{O}_{\mathbf{X},Rand}$ is a random ordering of $\mathbf{X}$ and n_{Rand} is the number of random orderings. In our experiments, $n_{Rand} = 3$.

NDCG@n: Evaluation with time point labels If the time point labels are known, a perfect ordering of time point importance would have every discriminatory time point for class c occurring at the start. If there are n discriminatory time points, we would expect to see these in the first n places in the ordered interpretability output. The fewer true discriminatory time points there are in the first n places, the worse the interpretability output. Furthermore, we want to reward the model for placing discriminatory time points earlier in the ordering (and punish it for placing non-discriminatory time points earlier on). This can be achieved by placing a higher weight on the start of the ordering. Formally,

$$
\begin{aligned}
NDCG@n(\mathbf{O}_{\mathbf{X},c}) &= \frac{1}{IDCG}\sum_{t=1}^{n}\frac{rel(\mathbf{o}_t)}{log_2(t+1)}, \\
\text{where } IDCG &= \sum_{t=1}^{n}\frac{1}{log_2(t+1)}, \\
\text{and } rel(\mathbf{o}_t) &= \begin{cases} 1 \text{ if } \mathbf{o}_t \text{ is a discriminatory time point for class } c, \\ 0 \text{ otherwise.} \end{cases}
\end{aligned} \tag{7.9}
$$

7.4 Initial Case Study

7.4.1 Synthetic Dataset

To explore our MILLET concept, and to evaluate the inherent interpretability of our models, we propose a new synthetic dataset called *WebTraffic*. By demonstrating daily and weekly seasonality, it is designed to mimic trends observed in streaming and e-commerce platforms. We inject different signatures into a collection of synthetic time series to create ten different classes: a zeroth normal class and nine signature classes. The signatures are partially inspired by the synthetic anomaly types proposed in Goswami et al. (2023). The discriminatory time points are known as we can control the location of the injected signatures. Therefore, we are able to evaluate if models can identify both the signature (which of the ten classes) and the location of the discriminatory time points — both can be achieved inherently by our MILLET models. Figure 7.3 provides an example for each class in this dataset, and further details are given below.

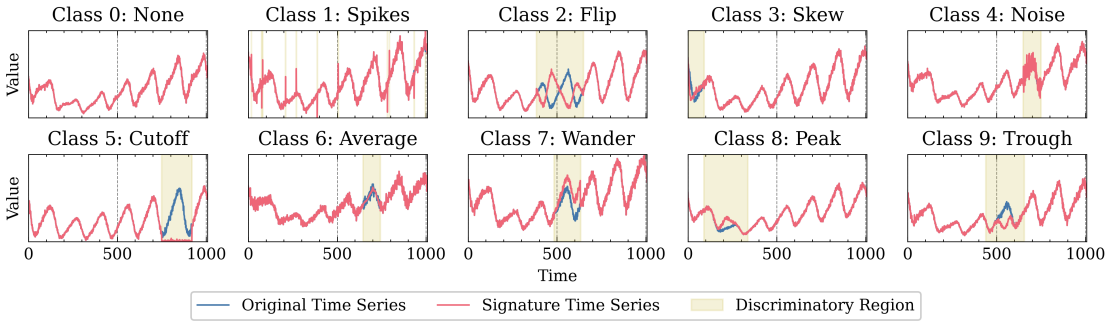

FIGURE 7.3: An example for each class of our *WebTraffic* dataset. Signatures are injected in a single random window, with the exception of Class 1 (Spikes), which uses random individual time points.

In *WebTraffic*, each time series is a week long with a sample rate of 10 minutes ($T = 60 * 24 * 7/10 = 1008$). The training and test set are independently generated using fixed seeds to facilitate reproducibility and are both balanced with 50 time series per class (500 total time series for each dataset).

To explain the synthetic time series generation process, we first introduce a new function:

$$WarpedSin(a, b, p, s, x) = \frac{a}{2} \sin\left(x' - \frac{\sin(x')}{s}\right) + b, \text{where } x' = 2\pi(x - p). \quad (7.10)$$

Parameters a, b, p, and s control amplitude, bias (intercept), phase, and skew respectively. $WarpedSin$ is used to generate daily seasonality in the following way:

$$SampleDay(a_D, b, p, s, \sigma, t) = \mathcal{N}(RateDay(a_D, b, p, s, t), \sigma), \quad (7.11)$$

$$RateDay(a_D, b, p, s, t) = WarpedSin(a_D, b, p + 0.55, s, t/144) \quad (7.12)$$

where $t \in [1, \ldots, 1008]$ is the time index and σ is a parameter that controls the amount of noise created when sampling (via a normal distribution). The daily rates provide daily seasonality (i.e., peaks in the evening and troughs in the morning). However, to take this further we also add weekly seasonality (i.e., more traffic at the weekends than early in the week). To do so, we further utilise $WarpedSin$:

$$RateWeek(a_W, t) = WarpedSin(a_W, 1, 0.6, 2, t/1008). \quad (7.13)$$

To add this weekly seasonality, we multiply the daily sampled values by the weekly rate. Therefore, to produce a base time series, we arrive at a formula with six parameters:

$$SampleWeek(a_D, a_W, b, p, s, \sigma) = SampleDay(a_D, b, p, s, \sigma, t) * RateWeek(a_W, t) \quad \text{for } t \in [1, \ldots, 1008]. \quad (7.14)$$

To generate a collection of n time series, we sample the following parameter distributions n times (i.e., once for every time series we want to generate):

- Amplitude daily: $a_D \sim \mathcal{U}_{\mathbb{R}}(2, 4)$
- Amplitude weekly $a_W \sim \mathcal{U}_{\mathbb{R}}(0.8, 1.2)$
- Bias $b \sim \mathcal{U}_{\mathbb{R}}(2.5, 5)$
- Phase $p \sim \mathcal{U}_{\mathbb{R}}(-0.05, 0.05)$

- Skew $s \sim \mathcal{U}_{\mathbb{R}}(1, 3)$
- Noise $\sigma \sim \mathcal{U}_{\mathbb{R}}(2, 4)$

We use $a \sim \mathcal{U}_{\mathbb{R}}(b, c)$ to denote uniform random sampling between b and c, where $a \in \mathbb{R}$. Below, we also use uniform random *integer* sampling $a' \sim \mathcal{U}_{\mathbb{Z}}(b, c)$, where $a' \in \mathbb{Z}$.

The above generation process results in a collection of n time series, but currently, they are all class zero (Class 0: None) as they have had no signatures injected. We describe how we inject each of the nine signature types below. Aside from the Spikes signature (Class 1), all signatures are injected in random windows of length $l \sim \mathcal{U}_{\mathbb{Z}}(36, 288)$ starting in position $p \sim \mathcal{U}_{\mathbb{Z}}(0, T - l)$. The minimum window size of 36 corresponds to 0.25 days, and the maximum size of 288 corresponds to 2 days. In all cases, values are clipped to be non-negative, i.e., all time points following signature injection are ≥ 0. These methods are inspired by, but not identical to, the work of Goswami et al. (2023). We provide an overview of the entire synthetic dataset generation process in Figure 7.4. Exact details on the injected signatures are given below, along with focused examples in Figure 7.5.

Class 1: Spikes Spikes are injected at random time points throughout the time series, with probability $p = 0.01$ for each time point. The magnitude of a spike is drawn from $\mathcal{N}(3.0, 2.0)$, and then added to or subtracted from the original time point value with equal probability.

Class 2: Flip The randomly selected window is flipped in the time dimension.

Class 3: Skew A skew is applied to the time points in the randomly selected window. A skew amount is first sampled from $\mathcal{U}_{\mathbb{R}}(0.25, 0.45)$, which is then added to or subtracted from 0.5 with equal probability. This gives a new skew value $w \in [0.05, 0.25] \cup [0.75, 0.95]$. The random window is then interpolated such that the value at the midpoint is now located at time point $\lfloor w * l \rfloor$ within the window, i.e., stretching the time series on one side and compressing it on the other.

Class 4: Noise Noise is added to the random window. The amount of noise is first sampled from $\sigma_{Noise} \sim \mathcal{U}_{\mathbb{R}}(0.5, 1.0)$. Then, for each time point in the selected window, noise is added according to $\mathcal{N}(0, \sigma_{Noise})$.

Class 5: Cutoff A cutoff value is sampled from $c \sim \mathcal{U}_{\mathbb{R}}(0.0, 0.2)$ The values in the random window are then set to $\mathcal{N}(c, 0.1)$.

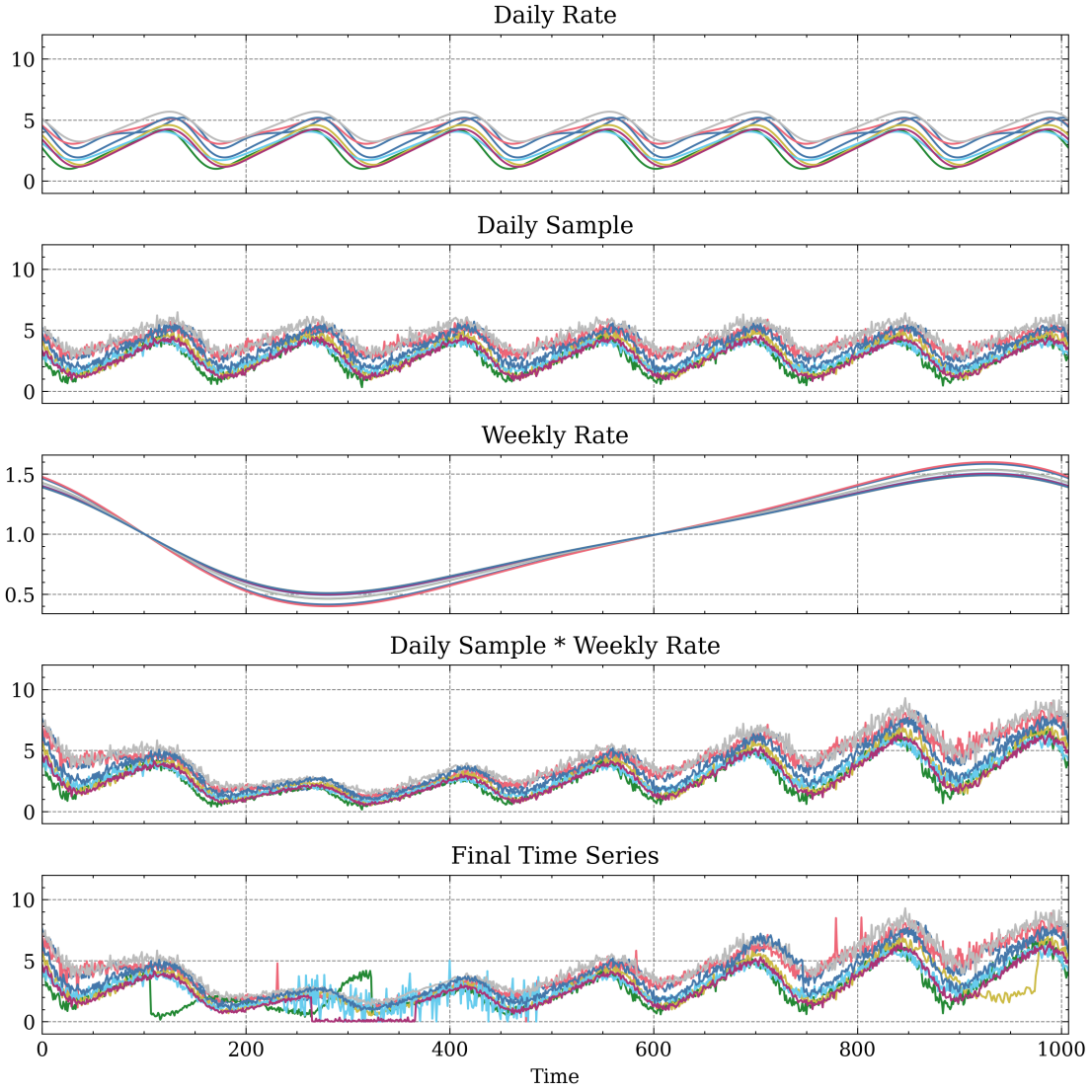

FIGURE 7.4: An overview of our *WebTraffic* synthetic dataset generation. From top to bottom: daily seasonality rate, daily seasonality with sampled noise, weekly seasonality rate, base time series with daily and weekly seasonality, and final time series with signatures injected into the base time series.

Class 6: Average This signature is the opposite of noise injection, i.e., applying smoothing to the values in the random window. This is achieved through applying a moving average with a window size sampled from $\mathcal{U}_{\mathbb{Z}}(5, 10)$.

Class 7: Wander A linear trend is applied to values in the random window. The trend linearly transitions from 0 to $\mathcal{U}_{\mathbb{R}}(2.0, 3.0)$, and is then added to or subtracted from the values in the window with equal probability.

Class 8: Peak A smooth peak is created from the probability density function (PDF) of $\mathcal{N}(0, 1)$ from -5 to 5, and then the values are multiplied by a scalar sampled from

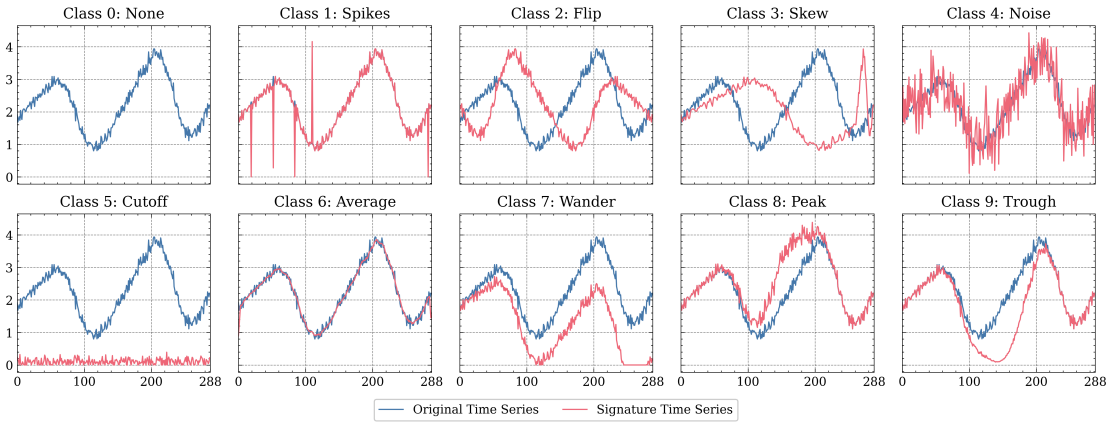

FIGURE 7.5: Examples of injected signatures. Each example focuses on the window in which the signatures are injected (i.e., omitting the rest of the time series), and windows are set to a fixed length of 288 (the maximum length when selecting random windows) to aid visualisation.

$\mathcal{U}_{\mathbb{R}}(1.5, 2.5)$. Values in the random window are then multiplied by the values of the peak, creating a smooth transition from the existing time series.

Class 9: Trough The same method to generate the Peak signatures is used to generate a trough, but the PDF values are instead multiplied by $\mathcal{U}_{\mathbb{R}}(-2.5, -1.5)$ (same scalar sample range but negative).

Given this *WebTraffic* dataset, in the next section, we use it to analyse both the predictive performance and interpretability of `MILLET`.

7.4.2 *WebTraffic* Results

We compare the four proposed MIL pooling approaches for `MILLET` with GAP on our *WebTraffic* dataset. Each pooling method is applied to the `FCN`, `ResNet`, and `InceptionTime` backbones. We conduct five training repeats of each model, starting from different network initialisations, and then ensemble them into a single model. Tables 7.1, 7.2, and 7.3 give results on accuracy, Area Under the Receiver Operating Characteristic (AUROC), and loss respectively. We find that `MILLET` improves accuracy averaged across all backbones from 0.850 to up to 0.874, with a maximum accuracy of 0.940 for `Conjunctive InceptionTime`.

Table 7.4 shows the interpretability results for *WebTraffic* across all backbones and pooling methods. In this evaluation, we compare to baselines CAM (applied to the original GAP models) and SHAP (applied to all models, see Appendix D.3). CAM is a lightweight post-hoc interpretability method (but not intrinsically part of the model output unlike our `MILLET` interpretations). SHAP is much more expensive to run than CAM or `MILLET` as it has to make repeated forward passes of the model. In this case, we

TABLE 7.1: *WebTraffic* accuracy.

	FCN	RNet	ITime	Mean
GAP	0.756	0.860	0.934	0.850
`Attention`	**0.820**	**0.866**	0.936	**0.874**
`Instance`	0.782	0.862	0.938	0.861
`Additive`	0.814	0.858	**0.940**	0.871
`Conjunctive`	0.818	0.850	**0.940**	0.869

TABLE 7.2: *WebTraffic* AUROC.

	FCN	RNet	ITime	Mean
GAP	0.961	0.982	**0.997**	0.980
`Attention`	**0.973**	**0.984**	**0.997**	0.984
`Instance`	0.962	0.982	**0.997**	0.980
`Additive`	**0.973**	**0.984**	**0.997**	0.984
`Conjunctive`	**0.973**	**0.984**	0.996	**0.985**

TABLE 7.3: *WebTraffic* loss.

	FCN	ResNet	ITime	Mean
GAP	0.939	**0.633**	0.268	0.614
`Attention`	**0.863**	0.701	0.279	0.614
`Instance`	0.871	0.709	0.257	0.612
`Additive`	0.882	0.678	**0.252**	0.604
`Conjunctive`	0.866	0.638	0.277	**0.594**

use SHAP with 500 samples, meaning it is 500 times more expensive than `MILLET`. In actuality, we find `MILLET` is over 800 times faster than SHAP (see Section 7.6.3).

SHAP performs particularly poorly, especially considering it is so much more expensive to run. Due to the exponential number of possible coalitions, SHAP struggles with the large number of time points. In some cases, it even has a negative AOPCR score, meaning its explanations are worse than random. For each backbone, `MILLET` has the best AOPCR and NDCG@n performance. The exception is NDCG@n for CAM on `InceptionTime`, where CAM is better (despite `MILLET` having a better AOPCR score). This is likely due to the sparsity of `MILLET` explanations: as shown for the Cutoff and Peak examples in Figure 7.6, `MILLET` produces interpretations that may not achieve full coverage of the discriminatory regions. While sparsity is beneficial for AOPCR (fewer time points need to be removed to decay the prediction), it can reduce NDCG@n as some discriminatory time points may not be identified (for example those in the middle of the Cutoff region). Finally, the best performance for each backbone comes from one of `MILLET` `Instance`, `Additive`, or `Conjunctive`. `Attention` is consistently worse than the other methods as it does not make class-specific explanations.

In Figure 7.6 we give example interpretations for `Conjunctive` `InceptionTime`; the best-performing model by accuracy. The model is able to identify the correct discriminatory regions for the different classes but focuses on certain parts of the different signatures. For example, for the Spikes class, the model identifies regions surrounding the individual spike time points, and for the Cutoff class, the model mainly identifies the start and end of the discriminatory region. This demonstrates how our interpretability outputs are not only able to convey where the discriminatory

TABLE 7.4: Interpretability performance (AOPCR / NDCG@n) on our *WebTraffic* dataset. Results are generated using the ensembled versions of the models.

	FCN	ResNet	InceptionTime	Mean
CAM	12.780 / 0.532	20.995 / 0.582	12.470 / **0.707**	15.415 / 0.607
SHAP - `Attention`	1.507 / 0.271	-3.293 / 0.250	-5.376 / 0.249	-2.387 / 0.257
SHAP - `Instance`	1.977 / 0.283	-0.035 / 0.257	-4.020 / 0.259	-0.692 / 0.266
SHAP - `Additive`	0.900 / 0.270	-1.077 / 0.250	-5.952 / 0.249	-2.043 / 0.256
SHAP - `Conjunctive`	0.987 / 0.267	-0.552 / 0.257	-5.359 / 0.246	-1.641 / 0.257
SHAP Best	1.977 / 0.283	-0.035 / 0.257	-4.020 / 0.259	-0.692 / 0.266
`MILLET - Attention`	4.780 / 0.425	6.382 / 0.380	-1.597 / 0.420	3.188 / 0.408
`MILLET - Instance`	12.841 / **0.540**	23.090 / 0.584	**13.192** / 0.704	16.374 / **0.609**
`MILLET - Additive`	**14.522** / 0.532	**24.880** / 0.589	10.274 / 0.684	**16.559** / 0.602
`MILLET - Conjunctive`	13.221 / 0.539	24.597 / **0.591**	11.100 / 0.694	16.306 / 0.608
`MILLET Best`	**14.522** / **0.540**	**24.880** / **0.591**	**13.192** / 0.704	**17.531** / **0.612**

regions are located but also provide insight into the model's decision-making process, providing transparency.

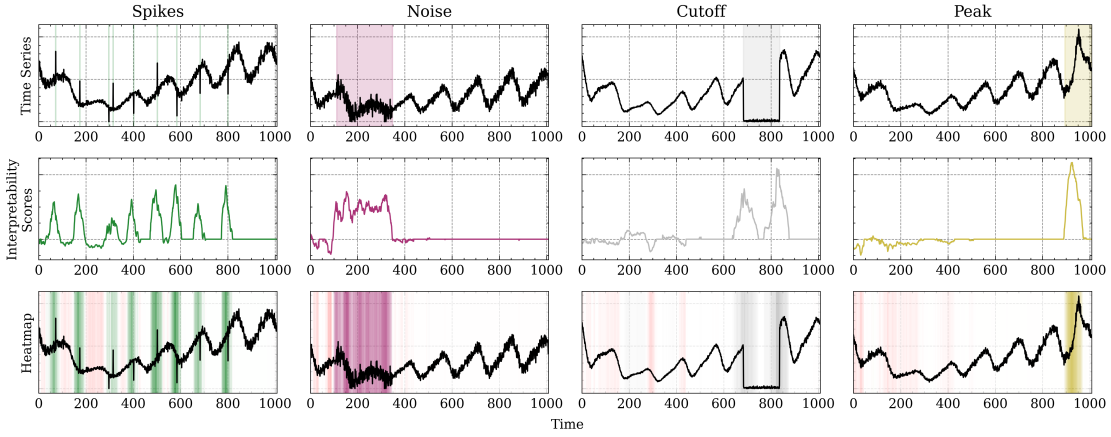

FIGURE 7.6: Interpretations for `Conjunctive InceptionTime` on our *WebTraffic* dataset.
Top: Time series with the known discriminatory time points highlighted.
Middle: Interpretability scores for each time point with respect to the target class.
Bottom: Interpretability scores heatmap as in Figure 7.1.

To further investigate the interpretability performance of `Conjunctive InceptionTime`, we conducted additional analysis on its *WebTraffic* performance. We found its accuracy was between 0.86 and 1.0 for nine of the ten classes, showing it had a consistently strong performance in identifying most signatures. However, for class 8 (Peak), its accuracy dropped to 0.76 — a rather large drop relative to the other classes. Class 0 (None) was the prediction that was made for the majority of the incorrect predictions for class 8, i.e., the model failed to identify the peak and did not find any other class signatures, so predicted the None class. We used the interpretability facilitated by `MILLET` to

investigate what was happening with these incorrect predictions. We found that, in some cases, the model was able to identify the correct region (positive predictions for class 8 at the location of the peak) despite its final prediction being incorrect. Examples are shown in Figure 7.7 — observe how the middle example shows support for class 8 (Peak) in the correct region despite the model getting the overall prediction incorrect. This identification of incorrect predictions and the ability to analyse the model's decision-making in more detail further highlights how `MILLET` can be useful in production.

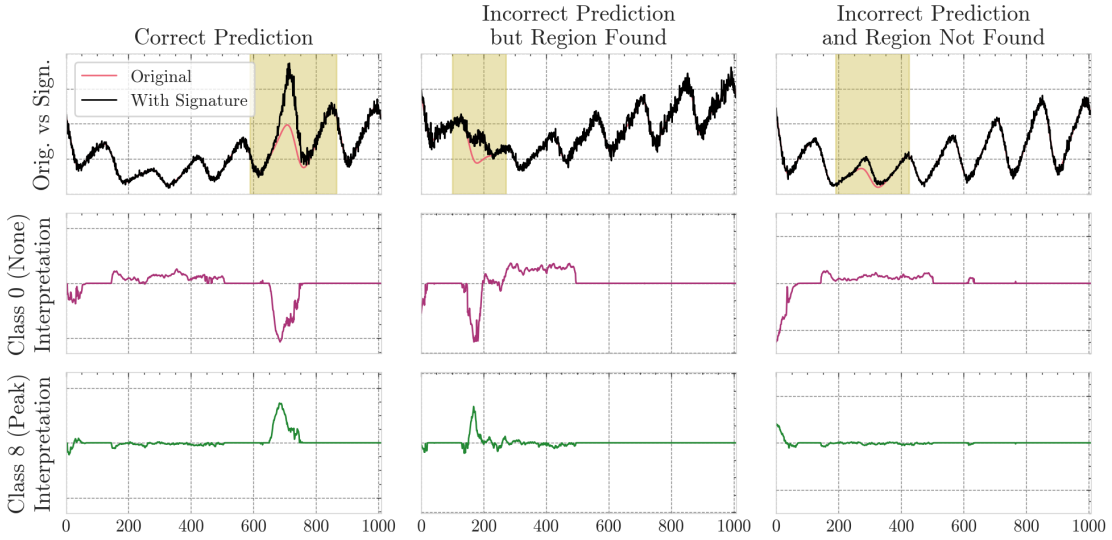

FIGURE 7.7: False negative investigation for `Conjunctive InceptionTime` on the *Web-Traffic* dataset. For the final example, the deviation from the original time series is very small and centred near an existing peak, which makes correct identification more difficult.

Top: The signature time series for an example from class 8 (Peak), where the original time series and signature region are shown.

Middle: The model's interpretation for class 0 (None).

Bottom: The model's interpretations for class 8 (Peak).

7.5 UCR Results

We evaluate `MILLET` on the UCR TSC Archive (Dau et al., 2019) — widely acknowledged as a definitive TSC benchmark spanning diverse domains across 85 univariate datasets (see Appendix D.4). Below are results for predictive performance, followed by results for interpretability.

7.5.1 Predictive Performance

We first compare the performance of the four MIL pooling methods with that of GAP across the three different backbones used in this work. This is used to evaluate the

change in predictive performance when using MILLET in a wide variety of different domains and determine which of the pooling methods proposed in Section 7.3.3 is best. It also facilitates insight into the applicability of each pooling method with different backbones. Table 7.5 shows that Conjunctive gives the best performance averaged across all backbones, and that the trend for MILLET is the same as the trend for GAP with respect to the choice of backbone, i.e., InceptionTime > ResNet > FCN. As such, it is unsurprising that Conjunctive InceptionTime has the best performance for both accuracy and balanced accuracy.

TABLE 7.5: MILLET predictive performance (accuracy / balanced accuracy) on 85 UCR datasets.

	FCN	ResNet	InceptionTime	Mean
GAP	0.828 / 0.804	0.843 / 0.819	0.853 / 0.832	0.841 / 0.818
Attention	0.781 / 0.754	**0.846** / **0.823**	0.855 / 0.832	0.827 / 0.803
Instance	0.829 / 0.804	0.842 / 0.818	0.855 / 0.833	0.842 / 0.819
Additive	0.835 / 0.810	0.845 / 0.822	0.855 / 0.832	0.845 / 0.822
Conjunctive	**0.838** / **0.814**	0.845 / 0.822	**0.856** / **0.834**	**0.846** / **0.823**

Given this result, we then compare the performance of MILLET with current SOTA methods, which is intended to provide better context for the performance of our newly proposed models. We select the top performing method from seven families of TSC approaches as outlined by Middlehurst et al. (2024), the top two of which are HC2 and Hydra-MR.[4]

In Figure 7.8 we give a critical difference (CD) diagram (Demšar, 2006) for balanced accuracy.[5] We find that 1) using Conjunctive improves performance in all cases, and 2) Conjunctive InceptionTime is comparable to the SOTA of HC2 and Hydra-MR. Although it has marginally better performance, strong claims cannot be made as the difference is not statistically significant.

We also perform a further direct comparison against the best two SOTA results, HC2 and Hydra-MR, allowing us to evaluate how many and on which datasets MILLET performs better. As shown in Figure 7.9, we find that our best-performing MILLET approach (Conjunctive InceptionTime) wins or draws on 48/85 (56.5%), 49/85 (57.7%), and 52/85 (61.2%) UCR datasets against InceptionTime, HC2, and Hydra-MR respectively.

Complete results are given in Table 7.6. While Conjunctive InceptionTime is marginally the best approach on balanced accuracy (outperforming the HC2 and Hydra-MR SOTA methods, but not with statistical significance), it is not quite as strong

[4]SOTA results obtained from *Bake Off Redux* using column zero of the results files (original train/test split). We did not use the FCN, ResNet, or InceptionTime results as we trained our own versions of these models.

[5]Based on the implementation from https://github.com/hfawaz/cd-diagram using Wilcoxon-Holm post-hoc analysis.

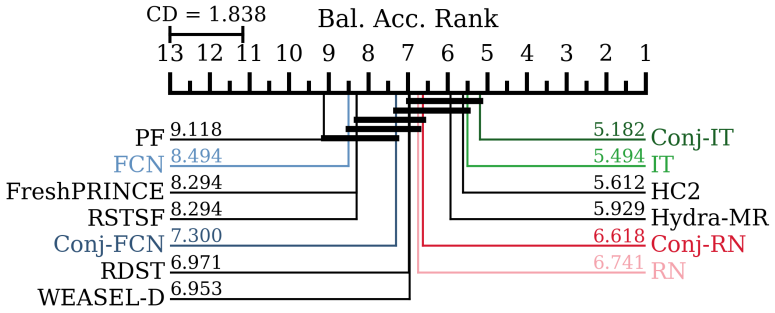

FIGURE 7.8: Critical difference diagram comparing `Conjunctive` `MILLET` methods with SOTA.

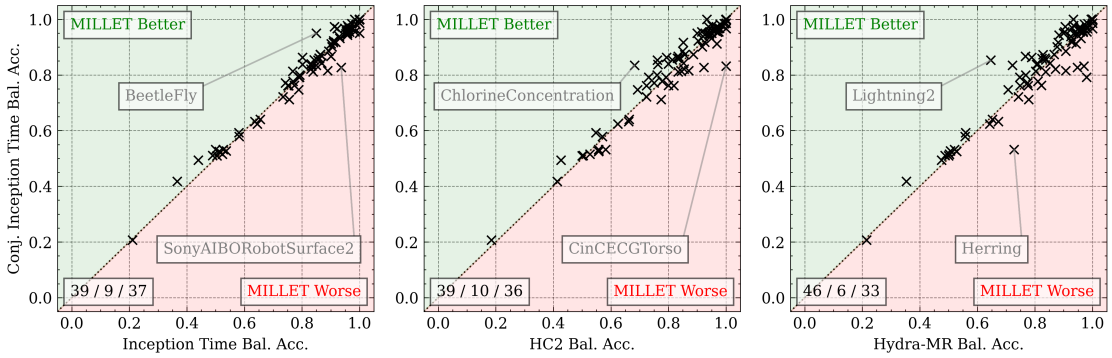

FIGURE 7.9: Head-to-head comparison of our best `MILLET` method against SOTA. Numbers in the bottom left indicate the number of wins / draws / losses for `Conjunctive InceptionTime`. Datasets with the greatest positive and negative differences are indicated.

on the other metrics. However, it remains competitive, and for each backbone, the use of `MILLET` improves performance across all metrics.

7.5.2 Interpretability Performance

To understand the interpretability of our `MILLET` methods on a wide variety of datasets, we evaluate their interpretability on the same set of 85 UCR datasets used in Section 7.5.1. As the UCR datasets do not have time point labels, we can only evaluate model interpretability using AOPCR. Averaged across all backbones, we find that `MILLET` has a best AOPCR of 6.00, compared to 5.71 achieved by GAP. Of the individual pooling methods within `MILLET`, we find that `Conjunctive` has the best interpretability performance. `Attention` performs poorly — this is expected as it does not create class-specific interpretations, but only general measures of importance; something that was also identified in general MIL interpretability by Early et al. (2022c, Chapter 4). In Table 7.7 we give `MILLET` interpretability results on the UCR datasets for both individual and ensemble models. Interestingly, we find that `MILLET` performs well in all cases except the ensemble `ResNet` models. In this case, its performance drops significantly compared to the individual ResNet performance — something that does

TABLE 7.6: Results for MILLET against baselines on 85 UCR datasets. Results are given in the form mean / rank / number of wins.

Method	Accuracy ↑	Bal. Accuracy ↑	AUROC ↑	NLL ↓
Hydra-MR	0.857/8.688/19	0.831/9.994/**17**	0.875/18.647/7	0.953/11.194/9
FreshPRINCE	0.833/12.841/14	0.801/13.824/13	0.944/10.206/19	0.714/10.800/11
PF	0.821/14.729/8	0.795/15.000/6	0.924/12.141/14	0.709/10.906/4
RDST	0.850/10.729/13	0.822/11.529/11	0.870/19.165/6	0.997/12.612/9
RSTSF	0.842/12.382/12	0.810/13.835/10	0.945/9.835/22	0.723/11.212/8
WEASEL-D	0.850/10.376/17	0.823/11.535/13	0.871/19.559/6	0.999/12.288/7
HC2	**0.860**/**7.988**/**20**	0.830/9.353/**17**	**0.950**/**6.006**/**42**	**0.607**/**8.388**/**14**
FCN	0.828/15.053/6	0.804/14.512/6	0.929/13.818/13	1.038/11.529/3
Attn. FCN	0.781/14.929/6	0.754/14.547/5	0.927/13.029/12	1.194/13.082/2
Ins. FCN	0.829/14.653/7	0.804/14.376/8	0.930/13.353/18	1.023/10.412/3
Add. FCN	0.835/13.547/5	0.810/13.100/5	0.933/11.594/14	0.994/10.247/3
Conj. FCN	0.838/12.624/6	0.814/12.247/6	0.934/11.012/16	0.973/9.635/3
ResNet	0.843/11.741/7	0.819/11.271/6	0.937/10.559/18	1.091/13.024/0
Attn. ResNet	0.846/11.400/8	0.823/10.671/9	0.939/9.888/13	1.051/12.188/1
Ins. ResNet	0.842/11.918/10	0.818/11.653/9	0.936/10.394/17	1.071/12.553/2
Add. ResNet	0.845/11.147/10	0.822/10.500/9	0.936/10.282/14	1.073/13.082/0
Conj. ResNet	0.845/11.335/10	0.822/10.806/10	0.939/9.329/15	1.035/12.176/1
ITime	0.853/9.724/16	0.832/8.965/15	0.939/9.500/24	1.078/11.571/7
Attn. ITime	0.855/9.512/10	0.832/9.018/11	0.940/8.876/20	1.069/11.618/3
Ins. ITime	0.855/9.471/16	0.833/9.053/16	0.940/8.406/25	1.050/10.929/5
Add. ITime	0.855/9.235/11	0.832/8.729/12	0.940/8.394/21	1.067/11.488/3
Conj. ITime	0.856/8.976/15	**0.834**/**8.482**/**17**	0.939/9.006/22	1.085/12.065/4

not happen for the other backbones. We observe something similar in our ablation study, see Section 7.6.4.

TABLE 7.7: MILLET AOPCR interpretability performance (individual / ensemble) on 85 UCR datasets. The best results over all MILLET models are given for reference.

	FCN	ResNet	InceptionTime	Mean
GAP	6.518 / 6.534	6.341 / **6.445**	3.392 / 4.144	5.417 / 5.707
Attention	-0.023 / 0.474	0.620 / 1.513	-0.909 / -0.936	-0.104 / 0.351
Instance	**6.868** / **6.925**	6.338 / 5.903	4.260 / 3.973	5.822 / 5.600
Additive	6.443 / 6.298	**6.526** / 5.871	**4.963** / **5.083**	**5.977** / 5.751
Conjunctive	6.361 / 6.498	6.438 / 6.006	4.553 / 4.764	5.784 / **5.756**
MILLET Best	*6.868 / 6.925*	*6.526 / 6.006*	*4.963 / 5.083*	*6.119 / 6.004*

In Figure 7.10 we observe a trade-off between interpretability and prediction, something we believe is insightful for model selection in practice. As backbone complexity increases, predictive performance increases while interpretability decreases.[6] The Pareto front shows MILLET dominates GAP for FCN and InceptionTime, but not ResNet. As stated above, MILLET gives better interpretability than GAP for *individual* ResNet

models, but struggles with the ensemble `ResNet` models. Out of the `MILLET` pooling approaches, we observe that only `Conjunctive` is never dominated (i.e., it lies on the Pareto front for all three backbones), and `Attention` is dominated in all cases.

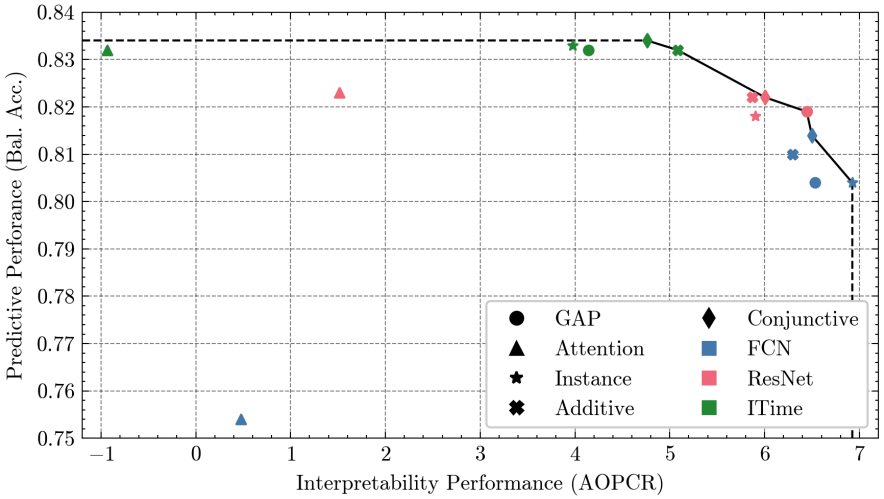

FIGURE 7.10: The interpretability-predictive performance trade-off on the 85 UCR datasets used in this work.

Figure 7.11 compares interpretations for `LargeKitchenAppliances`; a UCR dataset for identifying household electric appliances (Washing Machine, Tumble Dryer, or Dishwasher) from electricity usage. From the `MILLET` interpretability outputs, we identify that the model has learnt different signatures for each class: long periods of usage just above zero indicate Washing Machine, spikes above five indicate Tumble Dryer, and prolonged usage at just below five indicates Dishwasher. The Washing Machine example contains short spikes above five but `MILLET` identifies these as refuting the prediction, suggesting the model does not relate these spikes with the Washing Machine class. SHAP provides very noisy interpretations and does not show strong performance on the perturbation curves. Similar to our findings for *WebTraffic* (Section 7.4.2), `MILLET` provides sparser explanations than CAM, i.e., focusing on smaller regions and returning fewer discriminatory time points, which is helpful when explaining longer time series.

7.6 Discussion

In this section, we provide additional analysis of `MILLET`. We first examine performance by dataset properties such as dataset imbalance and time series length in Section 7.6.1. We then provide results for a model variance study in Section 7.6.2, followed by run time analysis in Section 7.6.3. Finally, in Section 7.6.4 we conducted an ablation study to examine the different components of `MILLET` in greater detail.

[6] `InceptionTime` is the most complex backbone, followed by `ResNet`, and then `FCN` is the simplest.

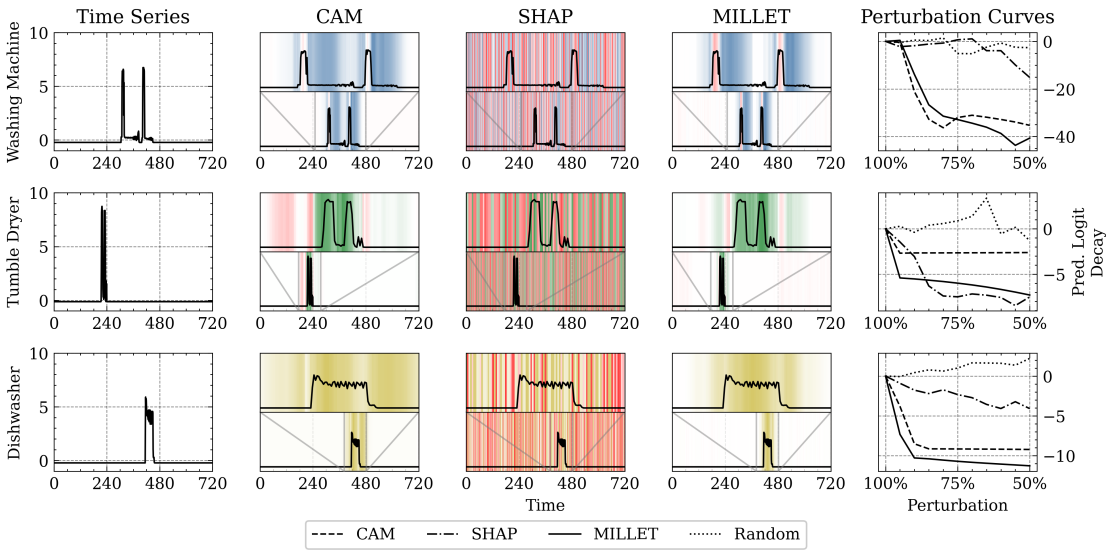

FIGURE 7.11: Interpretations on the `LargeKitchenAppliances` UCR dataset.
Left: Original time series.
Middle: Interpretability scores heatmap for CAM, SHAP, and `MILLET` as Figure 7.1.
Right: Perturbation curves showing the rate at which the model prediction decays when time points are removed following the orderings proposed by the different interpretability methods.

7.6.1 Performance by Dataset Properties

As discussed in Section 7.5.1, `Conjunctive InceptionTime` had the strongest performance on balanced accuracy when evaluated over 85 UCR datasets (outperforming SOTA methods such as `HC2` and `Hydra-MR`). To investigate whether this improvement is due to better performance on imbalanced datasets, we assessed balanced accuracy with respect to test dataset imbalance. To measure the imbalance of a dataset, we use normalised Shannon entropy:

$$\text{Dataset Balance} = -\frac{1}{\log C}\sum_{c=0}^{C-1}\frac{n_c}{n}\log\frac{n_c}{n}, \tag{7.15}$$

where C is the number of classes, n_c is the number of time series for class c, and n is the total number of time series in the dataset $\left(\sum_{c=0}^{C-1} n_c = n\right)$. This gives a score of dataset balance between 0 and 1, where 1 is perfectly balanced (equal number of time series per class) and values close to 0 indicate high levels of imbalance. We used a threshold of 0.9 to identify imbalanced datasets, and provide comparisons of `Conjunctive InceptionTime` with GAP `InceptionTime`, `HC2`, and `Hydra-MR` on these datasets in Figure 7.12.

From these results, we identify that `MILLET` has better performance than the SOTA methods when there is a high (test) dataset imbalance. It wins on 8/13, 9/13, and 9/13 datasets against GAP `InceptionTime`, `HC2`, and `Hydra-MR` respectively, and improves

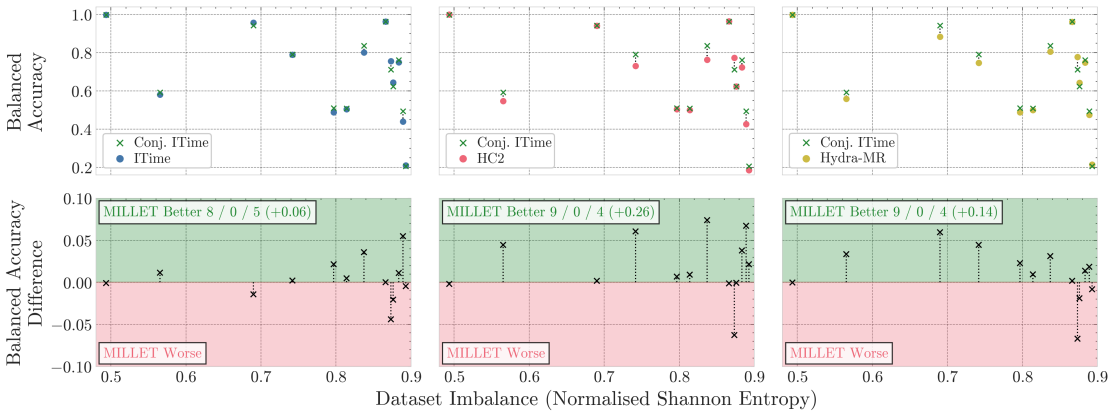

FIGURE 7.12: The effect of dataset imbalance on `MILLET` (`Conjunctive InceptionTime`) compared with GAP `InceptionTime` (left), `HC2` (middle), and `Hydra-MR` (right).

Top: Balanced accuracy on the imbalanced datasets for `MILLET` and each comparative method.

Bottom: The gain in balanced accuracy on the imbalanced datasets when using `MILLET`. Numbers indicate `MILLET`'s win/draw/loss and total improvement in balanced accuracy on these 13 datasets.

balanced accuracy by up to 7.4%. This demonstrates `MILLET` is more robust to dataset imbalance than the other methods, potentially because it has to make timestep predictions. Note this is without any specific focus on optimising for class imbalance, e.g., weighted cross entropy loss could be used during training to further improve performance on imbalanced datasets.

Using the UCR results, we now compare performance across time series length and the number of training time series. Figure Figure 7.13 shows the average balanced accuracy rank on different partitions of the UCR datasets for `HC2`, `Hydra-MR`, `InceptionTime`, and `Conjunctive InceptionTime`. The results are relatively consistent across different time series lengths. However, for the number of training time series, we see that `Conjunctive InceptionTime` is worse than GAP `InceptionTime` for smaller datasets, but excels on larger datasets.

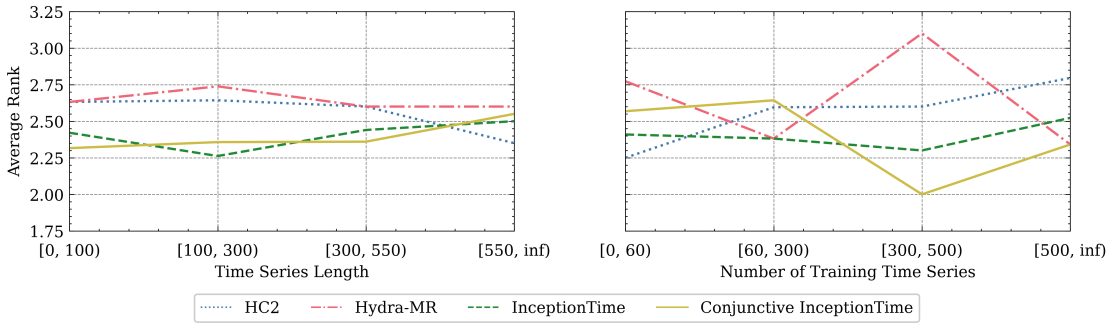

FIGURE 7.13: Comparison of `Conjunctive InceptionTime` against GAP, `HC2`, and `Hydra-MR` for two dataset properties. Bin ranges are chosen to give approximately equal bin sizes.

7.6.2 Model Variance Study

As noted by Middlehurst et al. (2024), while `InceptionTime` performs well overall, it often performs terribly on certain datasets, i.e., it has high variance in its predictive performance across different datasets. In Figure 7.14, we show that `MILLET` does aid in reducing this variance while improving overall performance. Notably, `Conjunctive InceptionTime` has lower variance than `HC2` and `Hydra-MR`.

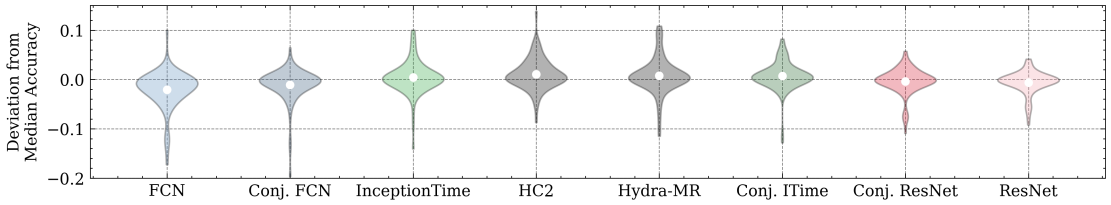

FIGURE 7.14: Evaluation of model variance with respect to median accuracy. Models are ordered from left to right by total variance.

7.6.3 Run Time Analysis

Below we analyse how `MILLET` increases the complexity of the original backbone models. We first compare the number of model parameters (Section 7.6.3.1) for different backbone and pooling combinations applied to the UCR `Fish` dataset, which is chosen as it is relatively central in the distribution of dataset statistics (175 training time series, 463 time points per time series, and 7 classes). We then compare the training and inference times on this same dataset (Section 7.6.3.2). Finally, we calculate the time complexity of the different pooling approaches and assess how they scale with respect to the number of timesteps and the number of classes (Section 7.6.3.3).

7.6.3.1 Number of Parameters

In Table 7.8, we detail the number of model parameters for the different backbones and aggregation approaches on `Fish`. `Instance` has the same number of parameters as GAP as the only change in the aggregation process is to swap the order in which pooling and classification are applied. Similarly, `Attention`, `Additive`, and `Conjunctive` all have the same number of parameters as each other, as they all include the same attention head (just applied in different ways; see Tables D.2, D.4, and D.5). Including this attention head only leads to an increase of approximately 0.4% in the number of parameters. Note these exact values will change for datasets with different numbers of classes, but the number of additional parameters for the attention head will remain the same.

TABLE 7.8: Number of model parameters for `Fish`. Percentages indicate the relative increase in parameters from the GAP model.

Pooling	FCN	ResNet	InceptionTime
GAP	**265.6K**	**504.9K**	**422.3K**
`Attention`	266.6K (+0.4%)	505.9K (+0.2%)	423.3K (+0.2%)
`Instance`	**265.6K (+0.0%)**	**504.9K (+0.0%)**	**422.3K (+0.0%)**
`Additive`	266.6K (+0.4%)	505.9K (+0.2%)	423.3K (+0.2%)
`Conjunctive`	266.6K (+0.4%)	505.9K (+0.2%)	423.3K (+0.2%)

7.6.3.2 Training/inference Time

In Table 7.9, we compare model training and inference times for `Fish`, and also include how long SHAP takes to run for these models. Due to the additional complexity of these methods (e.g., making time point predictions, applying attention, and using positional embeddings), the training times increase by up to 6%. Similarly, inference time increases by up to 7.5%. For SHAP, generating a single explanation takes 6+ seconds compared to the `MILLET` explanations which are generated as part of the inference step. Using `Conjunctive` as an example, SHAP is over 800 times slower than `MILLET`.

TABLE 7.9: Run time (wall clock) analysis results using `InceptionTime` on UCR `Fish`. Percentages following the `MILLET` methods give the increase in time relative to the backbone GAP model. Note inference time is given in milliseconds but SHAP is given in seconds.

Model	Train (**seconds**)	Inference (**milliseconds**)	SHAP (**seconds**)
GAP	**565 ± 2**	**7.50 ± 0.02**	**6.21 ± 0.03**
`Attention`	597 ± 1 (+5.7%)	8.02 ± 0.02 (+6.9%)	6.53 ± 0.01 (+5.2%)
`Instance`	582 ± 1 (+3.0%)	7.75 ± 0.01 (+3.3%)	6.38 ± 0.02 (+2.6%)
`Additive`	599 ± 1 (+6.0%)	8.06 ± 0.01 (+7.5%)	6.51 ± 0.02 (+4.8%)
`Conjunctive`	599 ± 1 (+6.0%)	8.05 ± 0.01 (+7.3%)	6.51 ± 0.02 (+4.8%)

7.6.3.3 Time Complexity Analysis

To provide a more thorough theoretical analysis, we give results for the number of real multiplications in the pooling methods (Freire et al., 2022). We focus solely on the pooling methods, omitting the computation of the backbone (as it is independent of the choice of pooling method). We also omit the computation of activation functions and other non-linear operations. Results are given for single inputs (no batching) and ignore possible underlying optimisation/parallelisation of functions. We first define the number of real multiplications for common functions of the pooling methods:

- Feed-forward layer: $T * d * o$, where T is the number of timesteps, d is the input size (128 in this work), and o is the output size (based on Shah and Bhavsar, 2022).
- Attention head consisting of two layers: $T * d * a + T * a$, where a is the size of the hidden layer in the attention head (8 in this work). This is consistent for the architectures that use attention (`Attention`, `Additive`, and `Conjunctive`).
- Weight or average a list of tensors: $n * l$, where n is the number of tensors and l is the length of each tensor.

Given these definitions, we calculate the overall number of multiplications for each pooling method. We provide results in Table 7.10 and visualisation in Figure 7.15. We make several observations:

- `Attention` has the best scaling with respect to time series length T and the number of classes C. However, it remains a poor choice overall due to its poor interpretability performance compared to other methods.
- GAP + CAM and `Instance` are the next fastest pooling methods. `Instance` is more efficient than GAP + CAM when $T * C < d * (T + C)$. In this work, as $d = 128$, `Instance` is marginally quicker than GAP + CAM.
- `Additive` and `Conjunctive` are the two slowest methods due to the additional overhead of applying attention and also making instance predictions, but the margin between them and GAP + CAM / `Instance` is not as large as one might expect. This is due to the small hidden layer size in the attention head ($a = 8$). `Conjunctive` is more efficient than `Additive` when $C < d$, which is true in all cases in this work as $d = 128$.
- In general, `Conjunctive` is the best choice when computational cost is less important than predictive performance. If faster computation is required, or if the number of timesteps or the number of classes is very large, `Instance` becomes a better choice.

7.6.4 Ablation Study

Our `MILLET` models make several improvements over the GAP backbones: MIL pooling, positional encodings, replicate padding, and dropout (Section 7.3.5). To understand where the gains in performance over GAP come from, we conduct an ablation study. To do so, we run additional model training runs, starting with the original backbone models and incrementally adding `MILLET` components in the order: MIL pooling, positional encoding, replicate padding, dropout, and ensembling. The final stage

TABLE 7.10: Time complexity analysis (number of real multiplications) of different MIL pooling approaches. For GAP, we include the calculation of CAM as this is required to produce interpretations (the other pooling methods produce interpretations inherently). We also state how the number of real multiplications scales with respect to the number of timesteps T and the number of classes C.

Pooling	Number of Real Multiplications	Scale w.r.t T	Scale w.r.t C
GAP + CAM	$d*T*C+d*T+d*C$	$d*C+d$	$d*T+d$
`Attention`	$d*T*a+(2d+a)*T+d*C$	$d*a+2d+a$	d
`Instance`	$d*T*C+T*C$	$d*C+C$	$d*T+T$
`Additive`	$d*T*C+d*T*a+(a+d+C)*T$	$d*C+d*a+a+d+C$	$d*T+T$
`Conjunctive`	$d*T*C+d*T*a+(a+2C)*T$	$d*C+d*a+a+2C$	$d*T+2*T$

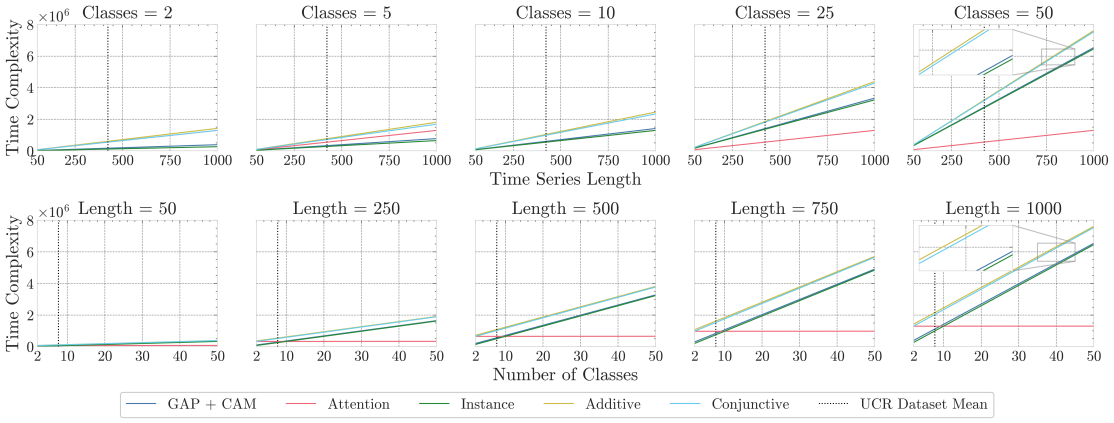

FIGURE 7.15: Visualisation of time complexity (number of real multiplications) for the different pooling methods used in this work. The mean time series length and number of classes across the 85 UCR datasets used in this work are shown (vertical dashed lines).

Top: Time complexity as time series length increases for varying numbers of classes.

Bottom: Time complexity as the number of classes increases for various time series lengths.

represents the full `MILLET` implementation. To reduce overheads in model training time and compute resources, we focus on `Conjunctive InceptionTime` and conduct the study on the three UCR datasets where `MILLET` shows the biggest increase in balanced accuracy over the backbone: `BeetleFly`, `Lightning7`, and `FaceAll`.

In Table 7.11 we provide results of the ablation study for balanced accuracy (predictive performance). We note that, on average, each component of `MILLET` improves performance, and the complete implementation (Step 6) has the best performance. For these datasets, the use of MIL pooling always improves performance over GAP. Additional components then change the performance differently for the different datasets. For example, positional encoding is very important for `BeetleFly`, but replicate padding is most important for `FaceAll`. Interestingly, replicate padding gives the biggest average performance increase across these datasets. However, these results

are confounded by the order of implementation, i.e., replicate padding is only applied after MIL pooling and positional encoding have been applied. As such, further studies are required to untangle the contribution of each component, but that is beyond the scope of this work.

TABLE 7.11: Ablation study on balanced accuracy. We begin with the original backbone model (without ensembling). We then incrementally add `MILLET` components until we reach the complete `MILLET` model. Results are given as balanced accuracy / improvement, where improvement is the difference in balanced accuracy relative to the prior row.

Model Config	`BeetleFly`	`Lightning7`	`FaceAll`	Mean
1. GAP (Single)	0.870	0.819	0.910	0.867
2. + MIL	0.870 / +0.000	0.844 / **+0.024**	0.912 / +0.002	0.875 / +0.009
3. + Pos Enc	0.900 / **+0.030**	0.832 / -0.012	0.911 / -0.001	0.881 / +0.006
4. + Replicate	0.900 / +0.000	0.840 / +0.008	0.967 / **+0.057**	0.903 / **+0.022**
5. + Dropout	0.920 / +0.020	0.848 / +0.008	0.970 / +0.003	0.913 / +0.010
6. + Ensemble	**0.950** / **+0.030**	**0.863** / +0.015	**0.974** / +0.003	**0.929** / +0.016

In Table 7.12 we provide results of the ablation study for AOPCR (interpretability performance). Interestingly, the addition of MIL pooling and positional encoding is detrimental to interpretability in these examples, despite improving predictive performance. However, interpretability improves once replicate padding and dropout are included. Further work is required to understand if these interpretability increases would also occur if replicate padding and dropout were applied to the GAP backbones, or if they improve performance when applied in conjunction with MIL pooling. Finally, we observe that interpretability decreases when ensembling the models. This is somewhat intuitive, as the interpretations are now explaining the decision-making of five models working in conjunction — it would be interesting to explore the difference in interpretations when analysing each model in the ensemble separately rather than together.

TABLE 7.12: Ablation study on AOPCR. Results are given as AOPCR / improvement.

Model Config	`BeetleFly`	`Lightning7`	`FaceAll`	Mean
1. GAP (Single)	0.122	4.085	0.552	1.586
2. + MIL	-0.736 / -0.858	3.985 / -0.100	0.859 / +0.306	1.369 / -0.217
3. + Pos Enc	-1.978 / -1.241	3.863 / -0.121	0.633 / -0.225	0.840 / -0.529
4. + Replicate	-1.926 / +0.052	4.093 / **+0.230**	**4.023** / **+3.390**	2.064 / **+1.224**
5. + Dropout	**1.242** / **+3.168**	**4.253** / +0.159	3.997 / -0.026	**3.164** / +1.101
6. + Ensemble	0.787 / -0.456	4.190 / -0.063	3.585 / -0.412	2.854 / -0.310

7.7 Conclusion

The MILLET framework presented in this work is the first comprehensive analysis of MIL for TSC. Its positive value is demonstrated across the 85 UCR datasets and the *WebTraffic* dataset proposed in this work. MILLET provides inherent mechanisms to localise, interpret, and explain influences on model behaviour, and improves predictive accuracy in most cases. Through the transparent decision-making gained from MILLET, practitioners can improve their understanding of model dynamics without the need for expensive (and often ineffective) post-hoc explainability methods. In addition, MILLET explanations are sparse — they distil the salient signatures of classes to a small number of relevant sub-sequences, which is especially important for long time series. We believe this work lays firm foundations for increased development of MIL methods in TSC and facilitates future work:

Extension to more datasets: A more comprehensive study would be to use the full set of 142 datasets from *Bake Off Redux* (Middlehurst et al., 2024) — we focused on the original set of 85 datasets due to limited compute resources and time constraints. Additional extensions could also include multivariate datasets and those with variable length time series. Variable lengths would not require any methodological changes (contrary to several other methods), but multivariate settings would require interpretability to consider the input channel (Hsieh et al., 2021).

Application to other models: We have demonstrated the use of MILLET for DL TSC models. Future work could extend its use to other types of TSC models, e.g., the ROCKET (convolutional) family of methods (Dempster et al., 2020), which includes Hydra-MR. Our proposed pooling method, Conjunctive, is also applicable in general MIL problems beyond TSC.

Pre-training/fine-tuning: While we trained MILLET models in an end-to-end manner, an alternative approach is to take a pre-trained GAP model, replace the GAP layers with one of the proposed MIL pooling methods, and then fine-tune the network (facilitating faster training). This would enable fast experimentation as the FE layers only have to be trained once.

This chapter presented the final piece of research produced for this thesis. The next chapter contains closing remarks, taking a retrospective look at the research as a whole and discussing some of the remaining open challenges in interpretable MIL.

Chapter 8

Conclusion

This thesis has explored the intersection of interpretability and Multiple Instance Learning (MIL). It has demonstrated the broad applicability of MIL: computer vision (Chapter 4), Earth Observation (EO) (Chapter 5), Reinforcement Learning (RL) (Chapter 6), and Time Series Classification (TSC) (Chapter 7). Common themes occur throughout these different areas, notably the concept of *which* and *what* interpretability questions: *which* are the key instances, and *what* outcomes do they support/refute? Focusing on different application domains naturally led to the development of more methods, some of which are more widely applicable beyond the scope they are studied in as part of this work. Below are retrospective comments on the research presented in this thesis (Section 8.1) along with some remaining challenges in this space (Section 8.2).

8.1 Research Retrospective

Insights can be observed by considering the research presented in this thesis as a whole and considering its impact post-publication.

The General Advantages of MIL Through this work, the advantages of MIL across different Machine Learning (ML) paradigms has been demonstrated. We have shown it can enhance performance, interpretability, and labelling efficiency in vision and sequential tasks, and do so across classification, regression, and segmentation problems. There are common elements of Deep Learning (DL) solutions in these various domains that we have been able to target. For example, a Global Average Pooling (GAP) + Class Activation Mapping (CAM) approach was used as a baseline in both the EO and TSC domains. Replacing this process with a MIL-based approach was shown to be effective, both for increasing predictive performance and for providing inherent (as opposed to post-hoc) interpretability with little additional computational overhead. This was of

particular note in the TSC work (Chapter 7), where the introduction of MIL was done in a "*plug-and-play*" manner. There are likely further domains and ML paradigms that could benefit from MIL-inspired approaches beyond those studied in this work.

Inherent vs Post-hoc Interpretability One outcome of the findings from the research into model-agnostic interpretability presented in Chapter 4 was that `MILLI` (one of the post-hoc methods proposed) was a strong MIL interpretability method. However, in further research (notably in Chapter 7), `MILLI` was not used. This was due to `MILLI`'s two main limitations: 1) despite being more sample efficient than Local Interpretable Model-agnostic Explanations (LIME) or Shapley Additive Explanations (SHAP), it still requires many forward passes of the model, and 2) it requires expensive tuning of hyperparameters to produce strong results. As such, it was prohibitively expensive in the TSC research due to the possibility of very long input time series. This highlights a general trend: while post-hoc methods should still be preferred in some cases, notably where there are complex interactions between instances and the data is not prohibitively large, inherently interpretability methods are also strong candidate methods due to their speed and potential for improving model predictive performance as well as interpretability. Furthermore, for initial development and rapid iteration, inherently interpretable methods are preferable.

Time Complexity Analysis The time complexity analysis conducted in Chapter 7, particularly the findings presented in Section 7.6.3.3, are applicable beyond our TSC applications. We focused solely on the time complexity of the pooling methods for the real multiplication calculations, so these results hold when applying the pooling methods to other MIL architectures and problems. As such, this research also has implications for the use of MIL beyond TSC and can help inform model selection when training/inference time is critical (e.g., in very large datasets).

Research Impact The research presented in the thesis has already been used in derivative works. In particular, the work on MIL for non-Markovian Reward Modelling (RM) detailed in Chapter 6 has been used in further research, notably Hejna and Sadigh (2023) and Kim et al. (2023), demonstrating its usefulness in the active research areas of RM and reinforcement learning from human feedback (RLHF).

8.2 Remaining Challenges

While individual chapters present their own set of tasks for future work, outlined below are four open questions in interpretable MIL.

Experiments with Human Evaluators None of the experiments in this thesis made use of human evaluators. As explained in Section 3.1.1, we focus on interpretability rather than explainability. This avoids the need for a social process that best communicates the interpretations to a specific target audience. However, for using interpretable MIL in production systems, consideration of this social process is required, i.e., communication of underlying interpretations is an open question and one that can only be answered through experimentation with human evaluators.

The Need for MIL Assumptions Historical MIL research explicitly defined the problem assumptions (the relationship between instances and bag labels). This was previously necessary to design specific methods to solve MIL problems. With the modern DL MIL approaches, which are general learners, these assumptions become less important with regard to model design. However, Raff and Holt (2023) recently highlighted that these DL approaches sometimes invalidate MIL assumptions, which has implications regarding trust: one might incorrectly assume that the models are obeying a MIL assumption as they have a high level of performance. The need for MIL assumptions, and their relationship to trust and interpretability, remains an open question.

Ongoing Development of Novel Approaches Since the revitalisation of MIL neural networks by Wang et al. (2018), the use of DL MIL methods has been developing each year. Improvements include pooling approaches such as `Attention` (Ilse et al., 2018), `Additive` (Javed et al., 2022), and `Graph` (Tu et al., 2019). `Conjunctive` pooling, proposed in Chapter 7, can also be seen as extending the use of MIL neural networks in the same way, especially given that it outperformed existing approaches. However, it was only explored within `MILLET` in application to TSC — a worthwhile area of future work would be to explore whether it is equally effective for other MIL problems and applications domains. Similar to other areas of ML, the architectures and techniques used in MIL are expected to continually develop with new innovations. With these expected continued developments, the model-agnostic approaches presented in Chapter 4, such as `MILLI`, will remain applicable.

The Future of MIL Datasets In this thesis, a range of datasets have been used. From an interpretability perspective, datasets with instance labels are particularly useful, as they facilitate additional interpretability evaluation metrics. However, the majority of datasets with instance labels are synthetic, so there is a need for a greater number of real (non-synthetic) datasets with instance labels. Furthermore, the more traditional MIL datasets such as MUSK, TEF, etc. (as used in Chapter 4) are not sufficiently meaningful for benchmarking, as many approaches can achieve very high performance on these datasets with only marginal differences. Instead, more complex datasets should be used to ensure robust evaluation, i.e., those that have multiple classes, interactions between instances, and greater numbers of instances and bags.

In summary, like many other machine learning disciplines, interpretable MIL is a rapidly developing research area with many possible applications. As demonstrated in this thesis, it can be used in different domains such as high-resolution imaging and time series analysis, and these domains have direct real-world applications where interpretability is paramount, such as Earth observation and healthcare. As MIL methods continue to develop and be applied more widely, understanding how these automated systems make decisions will only become more important.

References

Sarah Ahmed, Tayyaba Azim, Joseph Early, and Sarvapali Ramchurn. Revisiting deep Fisher vectors: Using Fisher information to improve object classification. In *33rd British Machine Vision Conference BMVC*. BMVA Press, 2022. URL https://bmvc2022.mpi-inf.mpg.de/0900.pdf.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2019. URL https://doi.org/10.1145/3292500.3330701.

Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. *Artificial Intelligence*, 201, 2013. ISSN 0004-3702. URL http://dx.doi.org/10.1016/j.artint.2013.06.003.

Stuart Andrews, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. *Advances in Neural Information Processing Systems*, 15, 2002. URL https://proceedings.neurips.cc/paper_files/paper/2002/file/3e6260b81898beacda3d16db379ed329-Paper.pdf.

Stefanos Angelidis and Mirella Lapata. Multiple instance learning networks for fine-grained sentiment analysis. *Transactions of the Association for Computational Linguistics*, 6, 2018. ISSN 2307-387X. URL http://dx.doi.org/10.1162/tacl_a_00002.

Sule Anjomshoae, Amro Najjar, Davide Calvaresi, and Kary Främling. Explainable agents and robots: Results from a systematic literature review. In *18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019), Montreal, Canada, May 13–17, 2019*. International Foundation for Autonomous Agents and Multiagent Systems, 2019. URL https://www.diva-portal.org/smash/get/diva2:1303810/FULLTEXT01.pdf.

Dan Ariely and Ziv Carmon. Gestalt characteristics of experiences: The defining features of summarized events. *Journal of Behavioral Decision Making*, 13(2), 2000. ISSN 1099-0771. URL http://dx.doi.org/10.1002/(SICI)1099-0771(200004/06)13:2<191::AID-BDM330>3.0.CO;2-A.

B. Babenko, Ming-Hsuan Yang, and S. Belongie. Robust object tracking with online multiple instance learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(8), 2011. ISSN 0162-8828. URL http://dx.doi.org/10.1109/TPAMI.2010.226.

Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *Proceedings of the National Conference on Artificial Intelligence*, 1996. URL http://www.cs.toronto.edu/~cebly/Papers/behaviors.pdf.

Fahiem Bacchus, Craig Boutilier, and Adam Grove. Structured solution methods for non-Markovian decision processes. In *AAAI/IAAI*. Citeseer, 1997. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=c5758bec4560070a78b665eb4fa6e2c12dbb5157.

Andrea Bajcsy, Dylan P Losey, Marcia K O'Malley, and Anca D Dragan. Learning robot objectives from physical human interaction. In *Conference on Robot Learning*. PMLR, 2017. URL http://proceedings.mlr.press/v78/bajcsy17a/bajcsy17a.pdf.

Bram Bakker. Reinforcement learning with long short-term memory. *Advances in Neural Information Processing Systems*, 14, 2001. URL https://proceedings.neurips.cc/paper/2001/file/a38b16173474ba8b1a95bcbc30d3b8a5-Paper.pdf.

Yifang Ban, Puzhao Zhang, Andrea Nascetti, Alexandre R Bevington, and Michael A Wulder. Near real-time wildfire progression monitoring with Sentinel-1 SAR time series and deep learning. *Scientific Reports*, 10(1), 2020. URL https://doi.org/10.1038/s41598-019-56967-x.

Sam Baron. Explainable AI and causal understanding: Counterfactual approaches considered. *Minds and Machines*, 33(2), 2023. ISSN 1572-8641. URL http://dx.doi.org/10.1007/s11023-023-09637-x.

Chandrayee Basu, Erdem Biyik, Zhixun He, Mukesh Singhal, and Dorsa Sadigh. Active learning of reward dynamics from hierarchical queries. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019. URL http://dx.doi.org/10.1109/IROS40897.2019.8968522.

Sander Beckers. Causal explanations and XAI. In *Conference on Causal Learning and Reasoning*. PMLR, 2022. URL https://proceedings.mlr.press/v177/beckers22a/beckers22a.pdf.

João Bento, Pedro Saleiro, André F. Cruz, Mário A.T. Figueiredo, and Pedro Bizarro. TimeSHAP: Explaining recurrent models through sequence perturbations. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21. ACM, 2021. URL http://dx.doi.org/10.1145/3447548.3467166.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. *Advances in Neural Information Processing Systems*, 24, 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Kent C. Berridge and John P. O'Doherty. From experienced utility to decision utility. In *Neuroeconomics*. Elsevier, 2014. URL http://dx.doi.org/10.1016/B978-0-12-416008-8.00018-8.

Tom Bewley and Freddy Lecue. Interpretable preference-based reinforcement learning with tree-structured reward functions. In *21st International Conference on Autonomous Agents and Multiagent Systems*, 2022. URL https://ifaamas.org/Proceedings/aamas2022/pdfs/p118.pdf.

Lukas Biewald. Experiment tracking with Weights and Biases, 2020. URL https://www.wandb.com/.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016. URL https://arxiv.org/abs/1606.01540.

Razvan Bunescu and Raymond Mooney. Learning to extract relations from the web using minimal supervision. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, 2007. URL https://aclanthology.org/P07-1073.pdf.

Katy Burrows, Richard J Walters, David Milledge, Karsten Spaans, and Alexander L Densmore. A new method for large-scale landslide classification from satellite radar. *Remote Sensing*, 11(3), 2019. URL https://doi.org/10.3390/rs11030237.

Ruth M. J. Byrne. Counterfactuals in explainable artificial intelligence (XAI): Evidence from human reasoning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, IJCAI-2019. International Joint Conferences on Artificial Intelligence Organization, 2019. URL http://dx.doi.org/10.24963/ijcai.2019/876.

Isabel Caballero, Javier Ruiz, and Gabriel Navarro. Sentinel-2 satellites provide near-real time evaluation of catastrophic floods in the West Mediterranean. *Water*, 11 (12), 2019. URL https://doi.org/10.3390/w11122499.

Marc-André Carbonneau, Veronika Cheplygina, Eric Granger, and Ghyslain Gagnon. Multiple instance learning: A survey of problem characteristics and applications. *Pattern Recognition*, 77, 2018. URL https://doi.org/10.1016/j.patcog.2017.10.009.

CCAI. Climate Change AI dataset wishlist, 2022. URL https://www.climatechange.ai/dataset-wishlist.pdf.

Arjun Chandrasekaran, Deshraj Yadav, Prithvijit Chattopadhyay, Viraj Prabhu, and Devi Parikh. It takes two to tango: Towards theory of AI's mind. *arXiv preprint arXiv:1704.00717*, 2017. URL https://arxiv.org/abs/1704.00717.

Lingji Chen and Kumpati S. Narendra. Nonlinear adaptive control using neural networks and multiple models. *Automatica*, 37(8), 2001. ISSN 0005-1098. URL http://dx.doi.org/10.1016/S0005-1098(01)00072-3.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems*, 30, 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/d5e2c0adad503c91f91df240d0cd4e49-Paper.pdf.

Ian Covert and Su-In Lee. Improving KernelSHAP: Practical Shapley value estimation using linear regression. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2021. URL http://proceedings.mlr.press/v130/covert21a/covert21a.pdf.

Jonathan Crabbé and Mihaela Van Der Schaar. Explaining time series predictions with dynamic masks. In *International Conference on Machine Learning*. PMLR, 2021. URL http://proceedings.mlr.press/v139/crabbe21a/crabbe21a.pdf.

Abhishek Das, Harsh Agrawal, Larry Zitnick, Devi Parikh, and Dhruv Batra. Human attention in visual question answering: Do humans and deep networks look at the same regions? *Computer Vision and Image Understanding*, 163, 2017. ISSN 1077-3142. URL http://dx.doi.org/10.1016/j.cviu.2017.10.001.

Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica*, 6(6), 2019. ISSN 2329-9274. URL http://dx.doi.org/10.1109/JAS.2019.1911747.

Ilke Demir, Krzysztof Koperski, David Lindenbaum, Guan Pang, Jing Huang, Saikat Basu, Forest Hughes, Devis Tuia, and Ramesh Raskar. DeepGlobe 2018: A challenge to parse the earth through satellite images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2018. URL https://doi.org/10.1109/cvprw.2018.00031.

Angus Dempster, François Petitjean, and Geoffrey I. Webb. ROCKET: Exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5), 2020. ISSN 1573-756X. URL http://dx.doi.org/10.1007/s10618-020-00701-z.

Angus Dempster, Daniel F. Schmidt, and Geoffrey I. Webb. HYDRA: Competing convolutional kernels for fast and accurate time series classification. *Data Mining and*

Knowledge Discovery, 37(5), 2023. ISSN 1573-756X. URL http://dx.doi.org/10.1007/s10618-023-00939-3.

Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7, 2006. URL https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf.

Don Dennis, Chirag Pabbaraju, Harsha Vardhan Simhadri, and Prateek Jain. Multiple instance learning for efficient sequential data classification on resource-constrained devices. *Advances in Neural Information Processing Systems*, 31, 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/d9fbed9da256e344c1fa46bb46c34c5f-Paper.pdf.

Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1–2), 1997. ISSN 0004-3702. URL http://dx.doi.org/10.1016/S0004-3702(96)00034-3.

Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning. *Communications of the ACM*, 63(1), 2019. ISSN 1557-7317. URL http://dx.doi.org/10.1145/3359786.

Joseph Early, Tom Bewley, Christine Evers, and Sarvapali Ramchurn. Non-Markovian reward modelling from trajectory labels via interpretable multiple instance learning. *Advances in Neural Information Processing Systems*, 35, 2022a. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/b157cfde6794e93b2353b9712bbd45a5-Paper-Conference.pdf.

Joseph Early, Ying-Jung Deweese, Christine Evers, and Sarvapali Ramchurn. Scene-to-patch Earth observation: Multiple instance learning for land cover classification. *NeurIPS Workshop: Tackling Climate Change with Machine Learning*, 2022b. URL https://arxiv.org/abs/2211.08247.

Joseph Early, Christine Evers, and Sarvapali Ramchurn. Model agnostic interpretability for multiple instance learning. In *International Conference on Learning Representations*, 2022c. URL https://openreview.net/forum?id=KSSfF5lMIAg.

Joseph Early, Ying-Jung Chen Deweese, Christine Evers, and Sarvapali Ramchurn. Extending scene-to-patch models: Multi-resolution multiple instance learning for Earth observation. *Environmental Data Science*, 2, 2023. URL https://doi.org/10.1017/eds.2023.30.

Joseph Early, Gavin KC Cheung, Kurt Cutajar, Hanting Xie, Jas Kandola, and Niall Twomey. Inherently interpretable time series classification via multiple instance learning. *Internation Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=xriGRsoAza.

Paul R Elsen, Earl C Saxon, B Alexander Simmons, Michelle Ward, Brooke A Williams, Hedley S Grantham, Salit Kark, Noam Levin, Katharina-Victoria Perez-Hammerle, April E Reside, et al. Accelerated shifts in terrestrial life zones under rapid climate change. *Global Change Biology*, 28(3), 2022. URL https://doi.org/10.1111/gcb.15962.

David Engel, Anita Williams Woolley, Lisa X. Jing, Christopher F. Chabris, and Thomas W. Malone. Reading the mind in the eyes or reading between the lines? Theory of mind predicts collective intelligence equally well online and face-to-face. *PLoS ONE*, 9(12), 2014. ISSN 1932-6203. URL http://dx.doi.org/10.1371/journal.pone.0115212.

Gregory Everett, Ryan J. Beal, Tim Matthews, Joseph Early, Timothy J. Norman, and Sarvapali D. Ramchurn. Inferring player location in sports matches: Multi-agent spatial imputation from limited observations. In *International Foundation for Autonomous Agents and Multiagent Systems*, AAMAS '23, 2023. ISBN 9781450394321. URL https://dl.acm.org/doi/10.5555/3545946.3598821.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision*, 88(2), 2010. URL https://doi.org/10.1007/s11263-009-0275-4.

Chao Fan, Cheng Zhang, Alex Yahja, and Ali Mostafavi. Disaster city digital twin: A vision for integrating artificial and human intelligence for disaster management. *International Journal of Information Management*, 56, 2021. URL https://doi.org/10.1016/j.ijinfomgt.2019.102049.

James Foulds and Eibe Frank. A review of multi-instance learning assumptions. *The Knowledge Engineering Review*, 25(1), 2010. ISSN 1469-8005. URL http://dx.doi.org/10.1017/S026988890999035X.

Navid Mohammadi Foumani, Lynn Miller, Chang Wei Tan, Geoffrey I. Webb, Germain Forestier, and Mahsa Salehi. Deep learning for time series classification and extrinsic regression: A current survey. *arXiv preprint arXiv:2302.02515*, 2023. URL https://arxiv.org/pdf/2302.02515.pdf.

Maria Fox, Derek Long, and Daniele Magazzeni. Explainable planning. *IJCAI Workshop on Explainable AI*, 2017. URL https://arxiv.org/pdf/1709.10256.pdf.

Pedro J Freire, Sasipim Srivallapanondh, Antonio Napoli, Jaroslaw E Prilepsky, and Sergei K Turitsyn. Computational complexity evaluation of neural network applications in signal processing. *arXiv preprint arXiv:2206.12191*, 2022. URL https://arxiv.org/pdf/2206.12191.pdf.

Pierre Friedlingstein, Michael O'sullivan, Matthew W Jones, Robbie M Andrew, Judith Hauck, Are Olsen, Glen P Peters, Wouter Peters, Julia Pongratz, Stephen Sitch, et al.

Global carbon budget 2020. *Earth System Science Data*, 12(4), 2020. URL https://doi.org/10.5194/essd-12-3269-2020.

Yosuke Fukuchi, Masahiko Osawa, Hiroshi Yamakawa, and Michita Imai. Autonomous self-explanation of behavior for interactive reinforcement learning agents. In *Proceedings of the 5th International Conference on Human Agent Interaction*, HAI '17. ACM, 2017. URL http://dx.doi.org/10.1145/3125739.3125746.

Saul Fuster, Trygve Eftestøl, and Kjersti Engan. Nested multiple instance learning with attention mechanisms. In *2022 21st IEEE International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2022. URL http://dx.doi.org/10.1109/ICMLA55696.2022.00038.

Leilani H. Gilpin, David Bau, Ben Z. Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE, 2018. URL http://dx.doi.org/10.1109/DSAA.2018.00018.

Alvin I. Goldman. Theory of mind. In *The Oxford Handbook of Philosophy of Cognitive Science*. Oxford University Press, 2012. ISBN 9780195309799. URL https://doi.org/10.1093/oxfordhb/9780195309799.013.0017.

Mononito Goswami, Cristian Challu, Laurent Callot, Lenon Minorics, and Andrey Kan. Unsupervised model selection for time-series anomaly detection. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/pdf?id=gOZ_pKANaPW.

Keri Grieman and Joseph Early. A risk-based approach to AI regulation: System categorisation and explainable AI practices. *SCRIPTed*, 20, 2023. URL https://script-ed.org/?p=4109.

Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. *Advances in Neural Information Processing Systems*, 26, 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/e034fb6b66aacc1d48f445ddfb08da98-Paper.pdf.

Riccardo Guidotti and Matteo D'Onofrio. Matrix profile-based interpretable time series classifier. *Frontiers in Artificial Intelligence*, 4, 2021. ISSN 2624-8212. URL http://dx.doi.org/10.3389/frai.2021.699448.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. A survey of methods for explaining black box models. *ACM Computing Surveys*, 51(5), 2018. ISSN 1557-7341. URL http://dx.doi.org/10.1145/3236009.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b/haarnoja18b.pdf.

Dylan Hadfield-Menell, Smitha Milli, Pieter Abbeel, Stuart J Russell, and Anca Dragan. Inverse reward design. *Advances in Neural Information Processing Systems*, 30, 2017. URL https://proceedings.neurips.cc/paper/2017/file/32fdab6559cdfa4f167f8c31b9199643-Paper.pdf.

Joseph Y. Halpern and Judea Pearl. Causes and explanations: A structural-model approach. Part I: Causes. *The British Journal for the Philosophy of Science*, 56(4), 2005. ISSN 1464-3537. URL http://dx.doi.org/10.1093/bjps/axi147.

Noriaki Hashimoto, Daisuke Fukushima, Ryoichi Koga, Yusuke Takagi, Kaho Ko, Kei Kohno, Masato Nakaguro, Shigeo Nakamura, Hidekata Hontani, and Ichiro Takeuchi. Multi-scale domain-adversarial multiple-instance CNN for cancer subtype classification with unannotated histopathological images. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2020. URL http://dx.doi.org/10.1109/CVPR42600.2020.00391.

Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In *2015 AAAI Fall Symposium Series*, 2015. URL https://cdn.aaai.org/ocs/11673/11673-51288-1-PB.pdf.

Bradley Hayes and Brian Scassellati. Challenges in shared-environment human-robot collaboration. *Learning*, 8(9), 2013. URL https://bradhayes.info/papers/hayes_challenges_HRI_collab_2013.pdf.

Steven R. Haynes, Mark A. Cohen, and Frank E. Ritter. Designs for explaining intelligent agents. *International Journal of Human-Computer Studies*, 67(1), 2009. ISSN 1071-5819. URL http://dx.doi.org/10.1016/j.ijhcs.2008.09.008.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. URL http://dx.doi.org/10.1109/CVPR.2016.90.

Joey Hejna and Dorsa Sadigh. Inverse preference learning: Preference-based RL without a reward function. *Interactive Learning with Implicit Human Feedback Workshop at ICML*, 2023. URL https://arxiv.org/abs/2305.15363.

Miguel A. Hernán, John Hsu, and Brian Healy. A second chance to get causal inference right: A classification of data science tasks. *CHANCE*, 32(1), 2019. ISSN 1867-2280. URL http://dx.doi.org/10.1080/09332480.2019.1579578.

Koen V. Hindriks. Debugging is explaining. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2012. ISBN 9783642327292. URL http://dx.doi.org/10.1007/978-3-642-32729-2_3.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8), 1997. ISSN 1530-888X. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Thorsten Hoeser and Claudia Kuenzer. Object detection and image segmentation with deep learning on Earth observation data: A review-part I: Evolution and recent trends. *Remote Sensing*, 12(10), 2020. ISSN 2072-4292. URL http://dx.doi.org/10.3390/rs12101667.

Thorsten Hoeser, Felix Bachofer, and Claudia Kuenzer. Object detection and image segmentation with deep learning on Earth observation data: A review-part II: Applications. *Remote Sensing*, 12(18), 2020. ISSN 2072-4292. URL http://dx.doi.org/10.3390/rs12183053.

Daniel Holliday, Stephanie Wilson, and Simone Stumpf. The effect of explanations on perceived control and behaviors in intelligent systems. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI '13. ACM, 2013. URL http://dx.doi.org/10.1145/2468356.2468389.

Andreas Holzinger, Georg Langs, Helmut Denk, Kurt Zatloukal, and Heimo Müller. Causability and explainability of artificial intelligence in medicine. *WIREs Data Mining and Knowledge Discovery*, 9(4), 2019. ISSN 1942-4795. URL http://dx.doi.org/10.1002/widm.1312.

Tsung-Yu Hsieh, Suhang Wang, Yiwei Sun, and Vasant Honavar. Explainable multivariate time series classification: A deep neural network which learns to attend to important variables as well as time intervals. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21. ACM, 2021. URL http://dx.doi.org/10.1145/3437963.3441815.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2017. URL http://dx.doi.org/10.1109/CVPR.2017.243.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2018. URL http://proceedings.mlr.press/v80/icarte18a/icarte18a.pdf.

Maximilian Ilse, Jakub Tomczak, and Max Welling. Attention-based deep multiple instance learning. In *International Conference on Machine Learning*. PMLR, 2018. URL http://proceedings.mlr.press/v80/ilse18a/ilse18a.pdf.

Aya Abdelsalam Ismail, Mohamed Gunady, Hector Corrada Bravo, and Soheil Feizi. Benchmarking deep learning interpretability in time series predictions. *Advances in Neural Information Processing Systems*, 33, 2020. URL https://proceedings.neurips.cc/paper/2020/file/47a3893cc405396a5c30d91320572d6d-Paper.pdf.

Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: A review. *Data Mining and Knowledge Discovery*, 33(4), 2019. ISSN 1573-756X. URL http://dx.doi.org/10.1007/s10618-019-00619-1.

Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre-Alain Muller, and François Petitjean. InceptionTime: Finding AlexNet for time series classification. *Data Mining and Knowledge Discovery*, 34(6), 2020. ISSN 1573-756X. URL http://dx.doi.org/10.1007/s10618-020-00710-y.

Vijay Manikandan Janakiraman. Explaining aviation safety incidents using deep temporal multiple instance learning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18. ACM, 2018. URL http://dx.doi.org/10.1145/3219819.3219871.

Firas Jarboui and Vianney Perchet. Trajectory representation learning for multi-task NMRDP planning. In *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021. URL http://dx.doi.org/10.1109/ICPR48806.2021.9412601.

Syed Ashar Javed, Dinkar Juyal, Harshith Padigela, Amaro Taylor-Weiner, Limin Yu, and Aaditya Prakash. Additive MIL: Intrinsically interpretable multiple instance learning for pathology. *Advances in Neural Information Processing Systems*, 35, 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/82764461a05e933cc2fd9d312e107d12-Paper-Conference.pdf.

Hong Jun Jeon, Smitha Milli, and Anca Dragan. Reward-rational (implicit) choice: A unifying formalism for reward learning. *Advances in Neural Information Processing Systems*, 33, 2020. URL https://proceedings.neurips.cc/paper/2020/file/2f10c1578a0706e06b6d7db6f0b4a6af-Paper.pdf.

Lynn H. Kaack, Priya L. Donti, Emma Strubell, George Kamiya, Felix Creutzig, and David Rolnick. Aligning artificial intelligence with climate change mitigation. *Nature Climate Change*, 12(6), 2022. ISSN 1758-6798. URL http://dx.doi.org/10.1038/s41558-022-01377-7.

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1–2), 1998. ISSN 0004-3702. URL http://dx.doi.org/10.1016/S0004-3702(98)00023-X.

Daniel Kahneman. Evaluation by moments: Past and future. In *Choices, Values, and Frames*. Cambridge University Press, 2000. URL http://dx.doi.org/10.1017/CBO9780511803475.039.

Krishna Karra, Caitlin Kontgis, Zoe Statman-Weil, Joseph C Mazzariello, Mark Mathis, and Steven P Brumby. Global land use/land cover with Sentinel 2 and deep learning. In *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*. IEEE, IEEE, 2021. URL https://doi.org/10.1109/igarss47720.2021.9553499.

M. G. Kendall. Rank correlation methods. *Biometrika*, 44(1/2), 1957. ISSN 0006-3444. URL http://dx.doi.org/10.2307/2333282.

Vladimir Khryashchev and Roman Larionov. Wildfire segmentation on satellite images using deep learning. In *2020 Moscow Workshop on Electronic and Networking Technologies (MWENT)*. IEEE, IEEE, 2020. URL https://doi.org/10.1109/mwent47943.2020.9067475.

Changyeon Kim, Jongjin Park, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. Preference transformer: Modeling human preferences using transformers for RL. *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Peot1SFDX0.

Qiuqiang Kong, Changsong Yu, Yong Xu, Turab Iqbal, Wenwu Wang, and Mark D. Plumbley. Weakly labelled AudioSet tagging with attention neural networks. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 27(11), 2019. ISSN 2329-9304. URL http://dx.doi.org/10.1109/TASLP.2019.2930913.

Michal Kosinski. Theory of mind might have spontaneously emerged in large language models. *arXiv preprint arXiv:2302.02083*, 2023. URL https://arxiv.org/abs/2302.02083.

Tzu-Sheng Kuo, Keng-Sen Tseng, Jia-Wei Yan, Yen-Cheng Liu, and Yu-Chiang Frank Wang. Deep aggregation net for land cover classification. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2018. URL http://dx.doi.org/10.1109/CVPRW.2018.00046.

Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, volume 2, pages 2169–2178. IEEE, 2006. URL https://inc.ucsd.edu/mplab/users/marni/Igert/Lazebnik_06.pdf.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998. ISSN 0018-9219. URL http://dx.doi.org/10.1109/5.726791.

Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial Intelligence and Statistics*. PMLR, 2015. URL http://proceedings.mlr.press/v38/lee15a.pdf.

Jihyeon Lee, Nina R Brooks, Fahim Tajwar, Marshall Burke, Stefano Ermon, David B Lobell, Debashish Biswas, and Stephen P Luby. Scalable deep learning to identify brick kilns and aid regulatory capacity. *Proceedings of the National Academy of Sciences*, 118(17), 2021a. URL https://doi.org/10.1073/pnas.2018863118.

Kimin Lee, Laura Smith, Anca Dragan, and Pieter Abbeel. B-Pref: Benchmarking preference-based reinforcement learning. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021b. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/file/d82c8d1619ad8176d665453cfb2e55f0-Paper-round1.pdf.

Sau-lai Lee, Ivy Yee-man Lau, Sara Kiesler, and Chi-Yue Chiu. Human mental models of humanoid robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005. URL https://ieeexplore.ieee.org/document/1570532/.

Jan Leike, David Krueger, Tom Everitt, Miljan Martic, Vishal Maini, and Shane Legg. Scalable agent alignment via reward modeling: A research direction. *arXiv preprint arXiv:1811.07871*, 2018. URL https://arxiv.org/abs/1811.07871.

Kin Kwan Leung, Clayton Rooke, Jonathan Smith, Saba Zuberi, and Maksims Volkovs. Temporal dependencies in feature importance for time series prediction. In *The Eleventh International Conference on Learning Representations*, 2022. URL https://www.cs.toronto.edu/~mvolkovs/ICLR23_WinIT.pdf.

Bin Li, Yin Li, and Kevin W Eliceiri. Dual-stream multiple instance learning network for whole slide image classification with self-supervised contrastive learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. IEEE, 2021a. URL https://doi.org/10.1109/cvpr46437.2021.01409.

Daxiang Li and Yue Zhang. Multi-instance learning algorithm based on LSTM for chinese painting image classification. *IEEE Access*, 8, 2020. URL https://doi.org/10.1016/j.artmed.2019.101744.

Hang Li, Fan Yang, Xiaohan Xing, Yu Zhao, Jun Zhang, Yueping Liu, Mengxue Han, Junzhou Huang, Liansheng Wang, and Jianhua Yao. Multi-modal multi-instance learning using weakly correlated histopathological images and tabular clinical information. In *Medical Image Computing and Computer Assisted Intervention–MICCAI 2021: 24th International Conference, Strasbourg, France, September 27–October 1, 2021, Proceedings, Part VIII 24*. Springer, 2021b. URL https://link.springer.com/chapter/10.1007/978-3-030-87237-3_51.

Sirui Li, Weixing Sun, and Tim Miller. Communication in human-agent teams for tasks with joint action. In *Coordination, Organizations, Institutions, and Norms in Agent Systems XI*. Springer International Publishing, 2016. ISBN 9783319426914. URL http://dx.doi.org/10.1007/978-3-319-42691-4_13.

Xirong Li, Yang Zhou, Jie Wang, Hailan Lin, Jianchun Zhao, Dayong Ding, Weihong Yu, and Youxin Chen. Multi-modal multi-instance learning for retinal disease recognition. In *Proceedings of the 29th ACM International Conference on Multimedia*, MM '21. ACM, 2021c. URL http://dx.doi.org/10.1145/3474085.3475418.

Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2017. URL https://doi.org/10.1109/cvpr.2017.106.

Zachary C. Lipton. The mythos of model interpretability. *Communications of the ACM*, 61 (10), 2018. ISSN 1557-7317. URL http://dx.doi.org/10.1145/3233231.

Guoqing Liu, Jianxin Wu, and Zhi-Hua Zhou. Key instance detection in multi-instance learning. In *Asian Conference on Machine Learning*. PMLR, 2012. URL http://proceedings.mlr.press/v25/liu12b/liu12b.pdf.

Xu Liu, Licheng Jiao, Jiaqi Zhao, Jin Zhao, Dan Zhang, Fang Liu, Shuyuan Yang, and Xu Tang. Deep multiple instance learning-based spatial–spectral classification for PAN and MS imagery. *IEEE Transactions on Geoscience and Remote Sensing*, 56(1), 2017. URL https://doi.org/10.1109/tgrs.2017.2750220.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2015. URL https://doi.org/10.1109/cvpr.2015.7298965.

Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in Neural Information Processing Systems*, 30, 2017. URL https://proceedings.neurips.cc/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

Niccolo Marini, Sebastian Otálora, Francesco Ciompi, Gianmaria Silvello, Stefano Marchesin, Simona Vatrano, Genziana Buttafuoco, Manfredo Atzori, and Henning Müller. Multi-scale task multiple instance learning for the classification of digital pathology images with global annotations. In *MICCAI Workshop on Computational Pathology*. PMLR, 2021. URL https://proceedings.mlr.press/v156/marini21a/marini21a.pdf.

Oded Maron and Tomás Lozano-Pérez. A framework for multiple-instance learning. *Advances in Neural Information Processing Systems*, 10, 1997. URL

https://proceedings.neurips.cc/paper_files/paper/1997/file/82965d4ed8150294d4330ace00821d77-Paper.pdf.

Matthew Middlehurst, James Large, Michael Flynn, Jason Lines, Aaron Bostrom, and Anthony Bagnall. HIVE-COTE 2.0: A new meta ensemble for time series classification. *Machine Learning*, 110(11–12), 2021. ISSN 1573-0565. URL http://dx.doi.org/10.1007/s10994-021-06057-9.

Matthew Middlehurst, Patrick Schäfer, and Anthony Bagnall. Bake off redux: a review and experimental evaluation of recent time series classification algorithms. *Data Mining and Knowledge Discovery*, pages 1–74, 2024. URL https://link.springer.com/article/10.1007/s10618-024-01022-1.

Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial Intelligence*, 267, 2019. ISSN 0004-3702. URL http://dx.doi.org/10.1016/j.artint.2018.07.007.

Shervin Minaee, Yuri Y. Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. ISSN 1939-3539. URL http://dx.doi.org/10.1109/TPAMI.2021.3059968.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 2015. ISSN 1476-4687. URL http://dx.doi.org/10.1038/nature14236.

Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020. URL https://christophm.github.io/interpretable-ml-book/.

Hafiz Suliman Munawar, Fahim Ullah, Siddra Qayyum, Sara Imran Khan, and Mohammad Mojtahedi. UAVs in disaster management: Application of integrated aerial imagery and convolutional neural network for flood detection. *Sustainability*, 13(14), 2021. URL https://doi.org/10.3390/su13147547.

Amin Nayebi, Sindhu Tipirneni, Chandan K. Reddy, Brandon Foreman, and Vignesh Subbian. WindowSHAP: An efficient framework for explaining time-series classifiers based on Shapley values. *Journal of Biomedical Informatics*, 144, 2023. ISSN 1532-0464. URL http://dx.doi.org/10.1016/j.jbi.2023.104438.

Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, volume 1. PMLR, 2000. URL http://www.datascienceassn.org/sites/default/files/Algorithms%20for%20Inverse%20Reinforcement%20Learning.pdf.

Perry C Oddo and John D Bolten. The value of near real-time Earth observations for improved flood disaster response. *Frontiers in Environmental Science*, 7, 2019. URL https://doi.org/10.3389/fenvs.2019.00127.

Cecilia Panigutti, Ronan Hamon, Isabelle Hupont, David Fernandez Llorca, Delia Fano Yela, Henrik Junklewitz, Salvatore Scalzo, Gabriele Mazzini, Ignacio Sanchez, Josep Soler Garrido, and Emilia Gomez. The role of explainable AI in the context of the AI Act. In *2023 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '23. ACM, 2023. URL http://dx.doi.org/10.1145/3593013.3594069.

S Parise, S Kiesler, L Sproull, and K Waters. Cooperating with life-like interface agents. *Computers in Human Behavior*, 15(2), 1999. ISSN 0747-5632. URL http://dx.doi.org/10.1016/S0747-5632(98)00035-1.

Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4), 1995. ISSN 1464-3510. URL http://dx.doi.org/10.1093/biomet/82.4.669.

Judea Pearl. The seven tools of causal inference, with reflections on machine learning. *Communications of the ACM*, 62(3), 2019. ISSN 1557-7317. URL http://dx.doi.org/10.1145/3241036.

Judea Pearl and Dana Mackenzie. *The book of why: The new science of cause and effect*. Basic Books, 2018. URL https://www.academia.edu/download/84468205/35a8d542d40b569f13c7403cd63728505ca3.pdf.

Sang Phan, Duy-Dinh Le, and Shin'ichi Satoh. Multimedia event detection using event-driven multiple instance learning. In *Proceedings of the 23rd ACM international conference on Multimedia*, MM '15. ACM, 2015. URL http://dx.doi.org/10.1145/2733373.2806330.

Yalong Pi, Nipun D Nath, and Amir H Behzadan. Convolutional neural networks for object detection in aerial imagery for disaster response and recovery. *Advanced Engineering Informatics*, 43, 2020. URL https://doi.org/10.1016/j.aei.2019.101009.

Rafael Poyiadzi, Raul Santos-Rodriguez, and Niall Twomey. Label propagation for learning with label proportions. In *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2018. URL http://dx.doi.org/10.1109/MLSP.2018.8517083.

Rafael Poyiadzis, Raul Santos-Rodriguez, and Niall Twomey. Active learning with label proportions. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019. URL http://dx.doi.org/10.1109/ICASSP.2019.8682748.

Owen Queen, Thomas Hartvigsen, Teddy Koker, Huan He, Theodoros Tsiligkaridis, and Marinka Zitnik. Encoding time-series explanations through self-supervised model behavior consistency. *arXiv preprint arXiv:2306.02109*, 2023. URL https://arxiv.org/pdf/2306.02109.pdf.

Gwenole Quellec, Guy Cazuguel, Beatrice Cochener, and Mathieu Lamard. Multiple-instance learning for medical image and video analysis. *IEEE Reviews in Biomedical Engineering*, 10, 2017. ISSN 1941-1189. URL http://dx.doi.org/10.1109/RBME.2017.2651164.

Edward Raff and James Holt. Reproducibility in multiple instance learning: A case for algorithmic unit tests. *arXiv preprint arXiv:2310.17867*, 2023. URL https://arxiv.org/pdf/2310.17867.pdf.

Saqib Rahman, Joe Early, Matt De Vries, Megan Lloyd, Ben Grace, Gopal Ramchurn, and Timothy Underwood. Predicting response to neoadjuvant therapy using image capture from diagnostic biopsies of oesophageal adenocarcinoma. *European Journal of Surgical Oncology*, 47(1), 2021a. URL https://www.sciencedirect.com/science/article/pii/S0748798320309197.

Saqib Rahman, Joseph Early, Ben Sharpe, Megan Lloyd, Matt De Vries, Ben Grace, Sarvapali Ramchurn, and Timothy Underwood. Predicting survival and response to therapy using diagnostic biopsies: A machine learning approach to facilitate treatment decisions for oesophageal adenocarcinoma. *British Journal of Surgery*, 108, 2021b. URL https://doi.org/10.1093/bjs/znab430.185.

R. Rahmani, S.A. Goldman, Hui Zhang, S.R. Cholleti, and J.E. Fritts. Localized content-based image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11), 2008. ISSN 0162-8828. URL http://dx.doi.org/10.1109/TPAMI.2008.112.

Maryam Rahnemoonfar, Tashnim Chowdhury, Argho Sarkar, Debvrat Varshney, Masoud Yari, and Robin Roberson Murphy. FloodNet: A high resolution aerial imagery dataset for post flood scene understanding. *IEEE Access*, 9, 2021. URL https://doi.org/10.1109/access.2021.3090981.

Alexander Rakhlin, Alex Davydow, and Sergey Nikolenko. Land cover classification from satellite imagery with U-Net and Lovász-softmax loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2018. URL https://doi.org/10.1109/cvprw.2018.00048.

Siddharth Reddy, Anca Dragan, Sergey Levine, Shane Legg, and Jan Leike. Learning human objectives by evaluating hypothetical behavior. In *International Conference on Machine Learning*. PMLR, 2020. URL http://proceedings.mlr.press/v119/reddy20a/reddy20a.pdf.

Chris Reed, Keri Grieman, and Joseph Early. Non-Asimov explanations regulating AI through transparency. *Nordic Yearbook of Law and Informatics*, 2021. URL https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3970518.

Marco Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should I trust you?": Explaining the predictions of any classifier. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*. Association for Computational Linguistics, 2016a. URL http://dx.doi.org/10.18653/v1/N16-3020.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Model-agnostic interpretability of machine learning. *ICML Workshop on Human Interpretability in Machine Learning*, 2016b. URL https://arxiv.org/abs/1606.05386.

Caleb Robinson, Le Hou, Kolya Malkin, Rachel Soobitsky, Jacob Czawlytko, Bistra Dilkina, and Nebojsa Jojic. Large scale high-resolution land cover mapping with multi-resolution data. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019. URL http://dx.doi.org/10.1109/CVPR.2019.01301.

David Rolnick, Andreas Veit, Serge Belongie, and Nir Shavit. Deep learning is robust to massive label noise. *arXiv preprint arXiv:1705.10694*, 2017. URL https://arxiv.org/pdf/1705.10694.pdf.

David Rolnick, Priya L Donti, Lynn H Kaack, Kelly Kochanski, Alexandre Lacoste, Kris Sankaran, Andrew Slavin Ross, Nikola Milojevic-Dupont, Natasha Jaques, Anna Waldman-Brown, et al. Tackling climate change with machine learning. *ACM Computing Surveys (CSUR)*, 55(2), 2022. URL https://doi.org/10.1145/3485128.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015. ISBN 9783319245744. URL http://dx.doi.org/10.1007/978-3-319-24574-4_28.

Avi Rosenfeld and Ariella Richardson. Explainability in human–agent systems. *Autonomous Agents and Multi-Agent Systems*, 33(6), 2019. ISSN 1573-7454. URL http://dx.doi.org/10.1007/s10458-019-09408-y.

Tim G. J. Rudner, Marc Rußwurm, Jakub Fil, Ramona Pelich, Benjamin Bischke, Veronika Kopačková, and Piotr Biliński. Multi3Net: Segmenting flooded buildings via fusion of multiresolution, multisensor, and multitemporal satellite imagery. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01), 2019. ISSN 2159-5399. URL http://dx.doi.org/10.1609/aaai.v33i01.3301702.

Stuart Russell. *Human Compatible: Artificial Intelligence and the Problem of Control*. Penguin, 2019. URL https://cdn.penguin.co.uk/dam-assets/books/9780141987507/9780141987507-sample.pdf.

Jhuma Sadhukhan. Net-zero action recommendations for scope 3 emission mitigation using life cycle assessment. *Energies*, 15(15), 2022. URL https://doi.org/10.3390/en15155522.

Wojciech Samek, Alexander Binder, Gregoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Muller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11), 2017. ISSN 2162-2388. URL http://dx.doi.org/10.1109/TNNLS.2016.2599820.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61, 2015. ISSN 0893-6080. URL http://dx.doi.org/10.1016/j.neunet.2014.09.003.

Stephen Scott, Jun Zhang, and Joshua Brown. On generalized multiple-instance learning. *International Journal of Computational Intelligence and Applications*, 05(01), 2005. ISSN 1757-5885. URL http://dx.doi.org/10.1142/S1469026805001453.

Selim Seferbekov, Vladimir Iglovikov, Alexander Buslaev, and Alexey Shvets. Feature pyramid network for multi-class land segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2018. URL https://doi.org/10.1109/cvprw.2018.00051.

Ankit Shah, Samir Wadhwania, and Julie Shah. Interactive robot training for non-Markov tasks. *arXiv preprint arXiv:2003.02232*, 2020. URL https://arxiv.org/abs/2003.02232.

Bhoomi Shah and Hetal Bhavsar. Time complexity in deep learning models. *Procedia Computer Science*, 215, 2022. ISSN 1877-0509. URL http://dx.doi.org/10.1016/j.procs.2022.12.023.

Divya Shanmugam, Davis Blalock, and John Guttag. Multiple instance learning for ECG risk stratification. In *Machine Learning for Healthcare Conference*. PMLR, 2019. URL http://proceedings.mlr.press/v106/shanmugam19a/shanmugam19a.pdf.

Zhuchen Shao, Hao Bian, Yang Chen, Yifeng Wang, Jian Zhang, Xiangyang Ji, et al. TransMIL: Transformer based correlated multiple instance learning for whole slide image classification. *Advances in Neural Information Processing Systems*, 34, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/10c272d06794d3e5785d5e7c5356e9ff-Paper.pdf.

L. S. Shapley. *A Value for n-Person Games*. Princeton University Press, 1953. URL http://dx.doi.org/10.1515/9781400881970-018.

Ilija Šimić, Vedran Sabol, and Eduardo Veas. XAI methods for neural time series classification: A brief review. *ICML Workshop on Human Interpretability in Machine Learning*, 2021. URL https://arxiv.org/pdf/2108.08009.pdf.

Korsuk Sirinukunwattana, Shan E Ahmed Raza, Yee-Wah Tsang, David R. J. Snead, Ian A. Cree, and Nasir M. Rajpoot. Locality sensitive deep learning for detection and classification of nuclei in routine colon cancer histology images. *IEEE Transactions on Medical Imaging*, 35(5), 2016. ISSN 1558-254X. URL http://dx.doi.org/10.1109/TMI.2016.2525803.

Torty Sivill and Peter Flach. LIMESegment: Meaningful, realistic time series explanations. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2022. URL https://proceedings.mlr.press/v151/sivill22a/sivill22a.pdf.

Keith E. Stanovich. Higher-order preferences and the master rationality motive. *Thinking & Reasoning*, 14(1), 2008. ISSN 1464-0708. URL http://dx.doi.org/10.1080/13546780701384621.

P.J. Sudharshan, Caroline Petitjean, Fabio Spanhol, Luiz Eduardo Oliveira, Laurent Heutte, and Paul Honeine. Multiple instance learning for histopathological breast cancer image classification. *Expert Systems with Applications*, 117, 2019. ISSN 0957-4174. URL http://dx.doi.org/10.1016/j.eswa.2018.09.049.

Andreas Theissler, Francesco Spinnato, Udo Schlegel, and Riccardo Guidotti. Explainable AI for time series classification: A review, taxonomy and research directions. *IEEE Access*, 10, 2022. ISSN 2169-3536. URL http://dx.doi.org/10.1109/ACCESS.2022.3207765.

S. Thiebaux, C. Gretton, J. Slaney, D. Price, and F. Kabanza. Decision-theoretic planning with non-Markovian rewards. *Journal of Artificial Intelligence Research*, 25, 2006. ISSN 1076-9757. URL http://dx.doi.org/10.1613/jair.1676.

Alessandro Tibo, Paolo Frasconi, and Manfred Jaeger. A network architecture for multi-multi-instance learning. In *Lecture Notes in Computer Science*. Springer International Publishing, 2017. ISBN 9783319712499. URL http://dx.doi.org/10.1007/978-3-319-71249-9_44.

Alessandro Tibo, Manfred Jaeger, and Paolo Frasconi. Learning and interpreting multi-multi-instance learning networks. *The Journal of Machine Learning Research*, 21 (1), 2020. URL https://www.jmlr.org/papers/volume21/18-811/18-811.pdf.

Jeremy Tien, Jerry Zhi-Yang He, Zackory Erickson, Anca Dragan, and Daniel S Brown. A study of causal confusion in preference-based reward learning. In *ICML 2022: Workshop on Spurious Correlations, Invariance and Stability*, 2022. URL https://openreview.net/forum?id=WaZZ0Sw9fWf.

Xin-Yi Tong, Gui-Song Xia, Qikai Lu, Huanfeng Shen, Shengyang Li, Shucheng You, and Liangpei Zhang. Land-cover classification with high-resolution remote sensing images using transferable deep models. *Remote Sensing of Environment*, 237, 2020. URL https://doi.org/10.1016/j.rse.2019.111322.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019. URL https://proceedings.neurips.cc/paper/2019/file/532435c44bec236b471a47a88d63513d-Paper.pdf.

Ming Tu, Jing Huang, Xiaodong He, and Bowen Zhou. Multiple instance learning with graph neural networks. *ICML Workshop on Learning and Reasoning with Graph-Structured Representations*, 2019. URL https://arxiv.org/abs/1906.04881.

Fida Ullah, Jincheng Liu, Muhammad Shafique, Sami Ullah, Muhammad Nawaz Rajpar, Adnan Ahmad, and Muhammad Shahzad. Quantifying the influence of Chashma Right Bank Canal on land-use/land-cover and cropping pattern using remote sensing. *Ecological Indicators*, 143, 2022. URL https://doi.org/10.1016/j.ecolind.2022.109341.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 30(1), 2016. ISSN 2159-5399. URL http://dx.doi.org/10.1609/aaai.v30i1.10295.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Ranga Raju Vatsavai, Budhendra Bhaduri, and Jordan Graesser. Complex settlement pattern extraction with multi-instance learning. In *Joint Urban Remote Sensing Event 2013*. IEEE, IEEE, 2013. URL https://doi.org/10.1109/jurse.2013.6550711.

Paul Voigt and Axel von dem Bussche. *The EU General Data Protection Regulation (GDPR)*. Springer International Publishing, 2017. ISBN 9783319579597. URL http://dx.doi.org/10.1007/978-3-319-57959-7.

Sandra Wachter, Brent Mittelstadt, and Luciano Floridi. Transparent, explainable, and accountable AI for robotics. *Science Robotics*, 2(6), 2017. ISSN 2470-9476. URL http://dx.doi.org/10.1126/scirobotics.aan6080.

Kaili Wang, Jose Oramas, and Tinne Tuytelaars. In defense of LSTMs for addressing multiple instance learning problems. In *Proceedings of the Asian Conference on Computer Vision*. Springer International Publishing, 2020a. URL https://doi.org/10.1007/978-3-030-69544-6_27.

Ning Wang, David V Pynadath, and Susan G Hill. The impact of POMDP-generated explanations on trust and performance in human-robot teams. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, 2016. URL https://www.ifaamas.org/Proceedings/aamas2016/pdfs/p997.pdf.

Peiqi Wang, William M. Wells, Seth Berkowitz, Steven Horng, and Polina Golland. Using multiple instance learning to build multimodal representations. In *Information Processing in Medical Imaging*. Springer Nature Switzerland, 2023. ISBN 9783031340482. URL http://dx.doi.org/10.1007/978-3-031-34048-2_35.

Sherrie Wang, William Chen, Sang Michael Xie, George Azzari, and David B Lobell. Weakly supervised deep learning for segmentation of remote sensing imagery. *Remote Sensing*, 12(2), 2020b. URL https://doi.org/10.3390/rs12020207.

Xinggang Wang, Yongluan Yan, Peng Tang, Xiang Bai, and Wenyu Liu. Revisiting multiple instance neural networks. *Pattern Recognition*, 74, 2018. URL https://doi.org/10.1016/j.patcog.2017.08.026.

Xinyu Wang, Haixia Xu, Liming Yuan, Wei Dai, and Xianbin Wen. A remote-sensing scene-image classification method based on deep multiple-instance learning with a residual dense attention convnet. *Remote Sensing*, 14(20), 2022. URL https://doi.org/10.3390/rs14205095.

Yuanyuan Wang, Chao Wang, Hong Zhang, Yingbo Dong, and Sisi Wei. Automatic ship detection based on retinanet using multi-resolution Gaofen-3 imagery. *Remote Sensing*, 11(5), 2019a. URL https://doi.org/10.3390/rs11050531.

Yun Wang, Juncheng Li, and Florian Metze. A comparison of five multiple instance learning pooling functions for sound event detection with weak labeling. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019b. URL http://dx.doi.org/10.1109/ICASSP.2019.8682847.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017. URL http://dx.doi.org/10.1109/IJCNN.2017.7966039.

Zhuang Wang, Liang Lan, and Slobodan Vucetic. Mixture model for multiple instance regression and applications in remote sensing. *IEEE Transactions on Geoscience and Remote Sensing*, 50(6), 2012. ISSN 1558-0644. URL http://dx.doi.org/10.1109/TGRS.2011.2171691.

Nils Weidmann, Eibe Frank, and Bernhard Pfahringer. A two-level learning method for generalized multi-instance problems. In *European Conference on Machine Learning*. Springer, 2003. URL https://www.academia.edu/download/40113836/A_Two-Level_Learning_Method_for_Generali20151117-13615-1w266qx.pdf.

D. Wierstra, A. Forster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 18(5), 2009. ISSN 1368-9894. URL http://dx.doi.org/10.1093/jigpal/jzp049.

Anita Williams Woolley, Christopher F. Chabris, Alex Pentland, Nada Hashmi, and Thomas W. Malone. Evidence for a collective intelligence factor in the performance of human groups. *Science*, 330(6004), 2010. ISSN 1095-9203. URL http://dx.doi.org/10.1126/science.1193147.

Jiajun Wu, Yinan Yu, Chang Huang, and Kai Yu. Deep multiple instance learning for image classification and auto-annotation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2015. URL http://dx.doi.org/10.1109/CVPR.2015.7298968.

Yan Xu, Tao Mo, Qiwei Feng, Peilin Zhong, Maode Lai, and Eric I-Chao Chang. Deep learning of feature representation with multiple instance learning for medical image analysis. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2014. URL http://dx.doi.org/10.1109/ICASSP.2014.6853873.

Chin-Chia Michael Yeh, Nickolas Kavantzas, and Eamonn Keogh. Matrix profile IV: Using weakly labeled time series to predict outcomes. *Proceedings of the VLDB Endowment*, 10(12), 2017. ISSN 2150-8097. URL http://dx.doi.org/10.14778/3137765.3137784.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer, 2014. URL http://dx.doi.org/10.1007/978-3-319-10590-1_53.

Cheng Zhang, Chao Fan, Wenlin Yao, Xia Hu, and Ali Mostafavi. Social media for intelligent public information and warning in disasters: An interdisciplinary review. *International Journal of Information Management*, 49, 2019. ISSN 0268-4012. URL http://dx.doi.org/10.1016/j.ijinfomgt.2019.04.004.

Min Zhang, Wenzhong Shi, Shanxiong Chen, Zhao Zhan, and Zhicheng Shi. Deep multiple instance learning for landslide mapping. *IEEE Geoscience and Remote Sensing Letters*, 18(10), 2020. URL https://doi.org/10.1109/lgrs.2020.3007183.

Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2016. URL http://dx.doi.org/10.1109/CVPR.2016.319.

Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(1), 2017. ISSN 2053-714X. URL http://dx.doi.org/10.1093/nsr/nwx106.

Zhi-Hua Zhou and Min-Ling Zhang. Neural networks for multi-instance learning. In *Proceedings of the International Conference on Intelligent Information Technology, Beijing, China*, 2002. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f15126efc9cf353e9eb5ad5ec58a817e67de3fde.

Zhi-Hua Zhou, Yu-Yin Sun, and Yu-Feng Li. Multi-instance learning by treating instances as non-I.I.D samples. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009. URL https://icml.cc/Conferences/2009/papers/422.pdf.

Yuansheng Zhu, Weishi Shi, Deep Shankar Pandey, Yang Liu, Xiaofan Que, Daniel E. Krutz, and Qi Yu. Uncertainty-aware multiple instance learning from large-scale long time series data. In *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021. URL http://dx.doi.org/10.1109/BigData52589.2021.9671469.

Appendix A

Appendix for Model Agnostic Interpretability for Multiple Instance Learning

A.1 Implementation Details

The code for this work can be found on GitHub: https://github.com/JAEarly/MILLI. All code for this work was implemented in Python 3.8, using the PyTorch library for the Machine Learning (ML) functionality. Hyperparameter tuning was carried out using the Optuna library. Some local experiments were carried out on a Dell XPS Windows laptop, utilising a GeForce GTX 1650 graphics card with 4GB of VRAM. Graphics Processing Unit (GPU) support for ML was enabled through CUDA v11.0. Other longer-running experiments, such as hyperparameter tuning, were carried out on a remote GPU node utilising a Volta V100 Enterprise Compute GPU with 16GB of VRAM. The longest model to train was the Graph Neural Network (GNN) for the Colorectal Cancer (CRC) dataset, which took just under half an hour. The longest-running experiment was the hyperparameter analysis for the CRC dataset at just under 40 hours. This was due to the high number of samples for the local surrogate methods and the fact that the results were averaged over 10 different models. The randomisation of data splits was fixed using seeding; the seed values are provided in the code for each experiment. We use the following sources for our data:

- The annotated Spatially Independent, Variable Area, and Lighting (SIVAL) dataset was downloaded from the publicly accessible page: http://pages.cs.wisc.edu/~bsettles/data/.
- The MNIST dataset was accessed directly from the PyTorch Python library: https://pytorch.org/vision/stable/datasets.html#mnist.

- The CRC dataset was downloaded from the publicly accessible page: https://warwick.ac.uk/fac/cross_fac/tia/data/crchistolabelednucleihe/.
- The Musk dataset was downloaded from the publicly accessible page: https://archive.ics.uci.edu/ml/datasets/Musk+%28Version+2%29
- The Tiger, Elephant and Fox datasets were downloaded from the publicly accessible page: http://www.cs.columbia.edu/~andrews/mil/datasets.html

A.2 SIVAL Experiment Details

SIVAL hyperparameters are given in Table A.1, detailing the values for the learning rate (LR), weight decay (WD), and dropout (DO). Additional hyperparameters in the training process are: batch size of one, early stopping patience of ten, and 100 epochs maximum — these are kept consistent across the models and are also the same for *4-MNIST-Bags* and CRC. The tuned architectures are given in Tables A.2 to A.5, where FC denotes a fully connect layer and ReLU denotes the rectified linear unit activation function. Finally, the results for each model are compared in Table A.6.

TABLE A.1: SIVAL training and model hyperparameters.

Model	LR	WD	DO
`Embedding`	5×10^{-3}	1×10^{-3}	0.45
`Instance`	5×10^{-4}	1×10^{-5}	0.25
`Attention`	1×10^{-3}	1×10^{-5}	0.15
`Graph`	5×10^{-4}	1×10^{-5}	0.2

TABLE A.2: SIVAL `Embedding` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	30	128
2	FC + ReLU + DO	128	256
3	mil-max	256	256
4	FC	256	13

TABLE A.3: SIVAL `Instance` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	30	512
2	FC + ReLU + DO	512	256
3	FC + ReLU + DO	256	64
4	FC	64	13
5	mil-mean	13	13

TABLE A.4: SIVAL `Attention` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	30	128
2	FC + ReLU + DO	128	256
3	FC + ReLU + DO	256	128
4	mil-attn(256) + DO	128	128
5	FC	128	13

A.3 *4-MNIST-Bags* Experiment Details

The training hyperparameter details are given in Table A.7. For each model, we first passed the MNIST instances through a convolutional architecture to produce initial

TABLE A.5: SIVAL `Graph` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	30	128
2a $\text{GNN}_{\text{embed}}$	SAGEConv + ReLU + DO	128	128
	SAGEConv + ReLU + DO	128	256
	SAGEConv + ReLU + DO	256	64
2b $\text{GNN}_{\text{cluster}}$	SAGEConv + Softmax	128	1
3	gnn-pool	64	64
4	FC + ReLU + DO	64	128
5	FC	128	13

TABLE A.6: SIVAL predictive performance results. Mean performance was calculated over ten repeat trainings of each model, and the standard error of the mean is given.

Model	Train Accuracy	Val Accuracy	Test Accuracy
`Embedding`	**0.984 ± 0.004**	**0.850 ± 0.009**	**0.819 ± 0.012**
`Instance`	0.967 ± 0.005	0.835 ± 0.010	0.808 ± 0.011
`Attention`	0.972 ± 0.007	0.830 ± 0.011	0.813 ± 0.012
`Graph`	0.932 ± 0.014	0.803 ± 0.014	0.781 ± 0.019

instance embeddings. This encoder was not tuned for each model (i.e., the architecture was fixed, but the weights were learnt). The architecture for this encoder is given in Table A.8: for the convolutional (Conv2d) and pooling (MaxPool2d) layers, the numbers in the brackets are the kernel size, stride, and padding. The model architectures are given in Tables A.9 to A.12. The encoder produces features vectors with 800 features, therefore the input size to each of the models is 800. The model results are given in Table A.13.

TABLE A.7: *4-MNIST-Bags* hyperparameters.

Model	LR	WD	DO
`Embedding`	1×10^{-4}	1×10^{-3}	0.3
`Instance`	1×10^{-4}	1×10^{-4}	0.3
`Attention`	1×10^{-4}	1×10^{-4}	0.15
`Graph`	5×10^{-5}	1×10^{-5}	0.3

TABLE A.8: *4-MNIST-Bags* convolutional encoding architecture.

Layer	Type	In	Out
1	Conv2d(5, 1, 0) + ReLU	1	20
2	MaxPool2d(2, 2, 0) + DO	20	20
3	Conv2d(5, 1, 0) + ReLU	20	50
4	MaxPool2d(2, 2, 0) + DO	50	50

A.4 CRC Experiment Details

The training hyperparameter details are given in Table A.14. Following the same procedure as the *4-MNIST-Bags* experiments, for each model, we first passed the patches through a convolutional architecture to produce initial instance embeddings. The

TABLE A.9: *4-MNIST-Bags* `Embedding` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	800	128
2	FC + ReLU + DO	128	512
3	mil-mean	512	512
4	FC	512	4

TABLE A.10: *4-MNIST-Bags* `Instance` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	800	512
2	FC + ReLU + DO	512	128
3	FC + ReLU + DO	128	64
4	FC	128	4
5	mil-mean	4	4

TABLE A.11: *4-MNIST-Bags* `Attention` architecture.

Layer	Type	In size	Out size
1	FC + ReLU + Dropout	800	64
2	FC + ReLU + Dropout	64	256
3	mil-attn(64) + Dropout	256	256
4	FC + ReLU + Dropout	256	64
5	FC	64	4

TABLE A.12: *4-MNIST-Bags* `Graph` architecture.

Layer	Type	In size	Out size
1	FC + ReLU + Dropout	800	64
2	FC + ReLU + Dropout	64	64
3a $\text{GNN}_{\text{embed}}$	SAGEConv + ReLU + Dropout	64	128
	SAGEConv + ReLU + Dropout	128	128
	SAGEConv + ReLU + Dropout	128	128
3b $\text{GNN}_{\text{cluster}}$	SAGEConv + Softmax	64	1
4	gnn-pool	128	128
5	FC + ReLU + Dropout	128	64
6	FC	64	4

TABLE A.13: *4-MNIST-Bags* model results. The mean performance was calculated over ten repeat trainings of each model, and the standard error of the mean is given.

Model	Train Accuracy	Val Accuracy	Test Accuracy
`Embedding`	**0.995 ± 0.001**	0.972 ± 0.002	0.971 ± 0.002
`Instance`	0.993 ± 0.001	**0.973 ± 0.002**	**0.974 ± 0.002**
`Attention`	0.991 ± 0.002	0.967 ± 0.003	0.967 ± 0.003
`Graph`	0.984 ± 0.002	0.968 ± 0.002	0.966 ± 0.002

architecture for this encoder is given in Table A.15, and then the model architectures are given in Tables A.16 to A.19. The encoder produces features vectors with 1200 features, therefore the input size to each of the models is 1200. The model results are given in Table A.20.

TABLE A.14: CRC training hyperparameters.

Model	LR	WD	DO
`Embedding`	5×10^{-4}	1×10^{-3}	0.3
`Instance`	5×10^{-4}	1×10^{-2}	0.25
`Attention`	1×10^{-3}	1×10^{-6}	0.2
`Graph`	1×10^{-3}	1×10^{-2}	0.35

TABLE A.15: CRC convolutional encoding architecture.

Layer	Type	In	Out
1	Conv2d(4, 1, 0) + ReLU	3	36
2	MaxPool2d(2, 2, 0) + DO	36	36
3	Conv2d(3, 1, 0) + ReLU	36	48
4	MaxPool2d(2, 2, 0) + DO	48	48

TABLE A.16: CRC `Embedding` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	1200	64
2	FC + ReLU + DO	64	512
3	mil-max	512	512
4	FC + ReLU + DO	512	128
4	FC + ReLU + DO	128	64
4	FC + ReLU + DO	64	2

TABLE A.17: CRC `Instance` architecture.

Layer	Type	In	Out
1	FC + ReLU + DO	1200	64
2	FC + ReLU + DO	64	64
3	FC + ReLU + DO	64	64
4	FC	64	2
5	mil-max	2	2

TABLE A.18: CRC `Attention` architecture.

Layer	Type	In size	Out size
1	FC + ReLU + Dropout	1200	64
2	FC + ReLU + Dropout	64	64
3	FC + ReLU + Dropout	64	256
4	mil-attn(128) + Dropout	256	256
5	FC	256	2

TABLE A.19: CRC `Graph` architecture.

Layer	Type	In size	Out size
1	FC + ReLU + Dropout	1200	64
2	FC + ReLU + Dropout	64	128
3a $\text{GNN}_{\text{embed}}$	SAGEConv + ReLU + Dropout	128	128
3b $\text{GNN}_{\text{cluster}}$	SAGEConv + Softmax	128	1
4	gnn-pool	128	128
5	FC + ReLU + Dropout	128	128
6	FC	128	2

TABLE A.20: CRC model results. The mean performance was calculated over ten repeat trainings of each model, and the standard error of the mean is given.

Model	Train Accuracy	Val Accuracy	Test Accuracy
`Embedding`	**0.880 ± 0.041**	0.805 ± 0.034	0.795 ± 0.042
`Instance`	0.856 ± 0.041	0.810 ± 0.043	0.795 ± 0.049
`Attention`	0.870 ± 0.014	**0.860 ± 0.017**	**0.830 ± 0.028**
`Graph`	0.791 ± 0.039	0.765 ± 0.031	0.770 ± 0.032

Appendix B

Appendix for Scene-to-Patch Earth Observation with Multi-Resolution Multiple Instance Learning

B.1 Implementation Details

This work was implemented in Python 3.10 and the Machine Learning (ML) functionality used PyTorch. The code and a list of required libraries are available at https://github.com/JAEarly/MIL-Multires-EO. The majority of model training was carried out on a remote Graphics Processing Unit (GPU) service using a Volta V100 Enterprise Compute GPU with 16GB of VRAM, which utilised CUDA v11.0 to enable GPU support (IRIDIS 5, University of Southampton). Training each model took a maximum of four hours. Trained models can be found in the code repository. Fixed seeds were used to ensure consistency of dataset splits between training and testing; these are included in the scripts that are used to run the experiments. We used Weights and Biases (Biewald, 2020) to track our experiments, along with Optuna for hyperparameter optimisation (Akiba et al., 2019). During hyperparameter optimisation, we ran 40 trials with pruning using the Tree-structured Parzen Estimator sampler (Bergstra et al., 2011).

B.2 Model Architectures

The Single Resolution (SR) Scene-to-Patch (S2P) models all use a consistent architecture: a Feature Extractor (FE) of convolutional and pooling layers, followed by a patch classifier (fully connected layers), and finally a Multiple Instance Learning (MIL) mean aggregator. The output of the classifier is a C-dimensional vector, which represents the

prediction for the C classes ($C = 7$ for DeepGlobe and $C = 10$ for FloodNet). Each patch is passed independently through the FE + patch classifier to produce a prediction for each patch, and then MIL mean aggregation is used to produce a scene-level prediction. For the multi-resolution models Multi-Resolution Single-Out (MRSO) and Multi-Resolution Multi-Out (MRMO), there are three FEs, one for each input resolution, and a combined patch classifier that utilises embeddings from each input resolution (see Section 5.3.3). The MRMO model also has independent patch classifiers for each input resolution. Both datasets use the same model architectures for each configuration. Below, in Tables B.1 to B.8, we give the exact architectures used.

TABLE B.1: S2P SR small architecture; patch size 28. For the Conv2d and MaxPool2d layers, the numbers in the brackets are the kernel size, stride, and padding. b is the bag size (number of patches), and C is the number of classes. The final three rows represent the aggregation and classification; the other rows are for the FE.

Layer	Type	Input	Output
Conv1	Conv2d(4, 1, 0) + ReLu	b x 3 x 28 x 28	b x 36 x 25 x 25
	MaxPool2d(2, 2, 0)	b x 36 x 25 x 25	b x 36 x 12 x 12
Conv2	Conv2d(3, 1, 0) + ReLu	b x 36 x 12 x 12	b x 48 x 10 x 10
	MaxPool2d(2, 2, 0)	b x 48 x 10 x 10	b x 48 x 5 x 5
	Flatten	b x 48 x 5 x 5	1 x b x 1200
FC1	FC + ReLU + Dropout	1 x b x 1200	1 x b x 512
FC2	FC	1 x b x 512	1 x b x 128
FC3	FC + ReLU + Dropout	1 x b x 128	1 x b x 64
FC4	FC	1 x b x 64	1 x b x C
-	MIL Mean Pooling	1 x b x C	1 x 1 x C

TABLE B.2: S2P SR medium architecture; patch size 56.

Layer	Type	Input	Output
Conv1	Conv2d(4, 1, 0) + ReLu	b x 3 x 56 x 56	b x 36 x 53 x 53
	MaxPool2d(2, 2, 0)	b x 36 x 53 x 53	b x 36 x 26 x 26
Conv2	Conv2d(3, 1, 0) + ReLu	b x 36 x 26 x 26	b x 48 x 24 x 24
	MaxPool2d(2, 2, 0)	b x 48 x 24 x 24	b x 48 x 12 x 12
	Flatten	b x 48 x 12 x 12	1 x b x 6912
FC1	FC + ReLU + Dropout	1 x b x 6912	1 x b x 512
FC2	FC	1 x b x 512	1 x b x 128
FC3	FC + ReLU + Dropout	1 x b x 128	1 x b x 64
FC4	FC	1 x b x 64	1 x b x C
-	MIL Mean Pooling	1 x b x C	1 x 1 x C

TABLE B.3: S2P SR large architecture; patch size 76. Visualised in Figure 5.2.

Layer	Type	Input	Output
Conv1	Conv2d(4, 1, 0) + ReLu	b x 3 x 76 x 76	b x 36 x 73 x 73
	MaxPool2d(2, 2, 0)	b x 36 x 73 x 73	b x 36 x 36 x 36
Conv2	Conv2d(3, 1, 0) + ReLu	b x 36 x 36 x 36	b x 48 x 34 x 34
	MaxPool2d(2, 2, 0)	b x 48 x 34 x 34	b x 48 x 17 x 17
Conv3	Conv2d(3, 1, 0) + ReLu	b x 48 x 17 x 17	b x 56 x 14 x 14
	MaxPool2d(2, 2, 0)	b x 56 x 14 x 14	b x 56 x 7 x 7
	Flatten	b x 56 x 7 x 7	1 x b x 2744
FC1	FC + ReLU + Dropout	1 x b x 2744	1 x b x 512
FC2	FC	1 x b x 512	1 x b x 128
FC3	FC + ReLU + Dropout	1 x b x 128	1 x b x 64
FC4	FC	1 x b x 64	1 x b x C
-	MIL Mean Pooling	1 x b x C	1 x 1 x C

TABLE B.4: S2P SR large architecture; patch size 102.

Layer	Type	Input	Output
Conv1	Conv2d(4, 1, 0) + ReLu	b x 3 x 102 x 102	b x 36 x 99 x 99
	MaxPool2d(2, 2, 0)	b x 36 x 99 x 99	b x 36 x 49 x 49
Conv2	Conv2d(3, 1, 0) + ReLu	b x 36 x 49 x 49	b x 48 x 47 x 47
	MaxPool2d(2, 2, 0)	b x 48 x 47 x 47	b x 48 x 23 x 23
Conv3	Conv2d(3, 1, 0) + ReLu	b x 48 x 23 x 23	b x 56 x 21 x 21
	MaxPool2d(2, 2, 0)	b x 56 x 21 x 21	b x 56 x 10 x 10
	Flatten	b x 56 x 10 x 10	1 x b x 5600
FC1	FC + ReLU + Dropout	1 x b x 5600	1 x b x 512
FC2	FC	1 x b x 512	1 x b x 128
FC3	FC + ReLU + Dropout	1 x b x 128	1 x b x 64
FC4	FC	1 x b x 64	1 x b x C
-	MIL Mean Pooling	1 x b x C	1 x 1 x C

TABLE B.5: S2P MRSO architecture; patch size 76. This architecture utilises the FE from the SR large architecture (Table B.3; layers Conv1 to FC2) to create embeddings at each input resolution. The embeddings are then concatenated (see Section 5.3.4), and finally classified using the same classifier and MIL pooling approach as in the other models. Note, the patch predictions of size 1 x b x C are still produced by this model but are omitted from the table for simplicity. Visualised in Figure 5.3.

Layer	Input	Output
$s = 0$ FE	b x 3 x 76 x 76	1 x b x 128
$s = 1$ FE	$4b$ x 3 x 76 x 76	1 x $4b$ x 128
$s = 2$ FE	$16b$ x 3 x 76 x 76	1 x $16b$ x 128
Multi-resolution Concatenation	1 x $\{b, 4b, 16b\}$ x 128	1 x $16b$ x 384
$s = m$ Classifier	1 x $16b$ x 384	1 x 1 x C

TABLE B.6: S2P MRSO architecture; patch size 102. This architecture utilises the FE from the SR large architecture (Table B.4; layers Conv1 to FC2).

Layer	Input	Output
$s = 0$ FE	b x 3 x 102 x 102	1 x b x 128
$s = 1$ FE	$4b$ x 3 x 102 x 102	1 x $4b$ x 128
$s = 2$ FE	$16b$ x 3 x 102 x 102	1 x $16b$ x 128
Multi-resolution Concatenation	1 x $\{b, 4b, 16b\}$ x 128	1 x $16b$ x 384
$s = m$ Classifier	1 x $16b$ x 384	1 x 1 x C

TABLE B.7: S2P MRMO architecture; patch size 76. This architecture utilises the FE from the SR large architecture (Table B.3; layers Conv1 to FC2) to create embeddings at each input resolution. The embeddings are then concatenated (see Section 5.3.4), and finally classified using the same classifier and MIL pooling approach as in the other models. In addition, each set of embeddings is also independently classified. Patch predictions are produced for each independent resolution as well as the combined resolution, but are omitted from the table for simplicity. Visualised in Figure 5.3.

Layer	Input	Output
$s = 0$ FE	b x 3 x 76 x 76	1 x b x 128
$s = 1$ FE	$4b$ x 3 x 76 x 76	1 x $4b$ x 128
$s = 2$ FE	$16b$ x 3 x 76 x 76	1 x $16b$ x 128
$s = 0$ Classifier	1 x b x 128	1 x 1 x C
$s = 1$ Classifier	1 x $4b$ x 128	1 x 1 x C
$s = 2$ Classifier	1 x $16b$ x 128	1 x 1 x C
Multi-resolution Concatenation	1 x $\{b, 4b, 16b\}$ x 128	1 x $16b$ x 384
$s = m$ Classifier	1 x $16b$ x 384	1 x 1 x C

TABLE B.8: S2P MRMO architecture; patch size 102. This architecture utilises the FE from the SR large architecture (Table B.4; layers Conv1 to FC2).

Layer	Input	Output
$s = 0$ FE	b x 3 x 102 x 102	1 x b x 128
$s = 1$ FE	$4b$ x 3 x 102 x 102	1 x $4b$ x 128
$s = 2$ FE	$16b$ x 3 x 102 x 102	1 x $16b$ x 128
$s = 0$ Classifier	1 x b x 128	1 x 1 x C
$s = 1$ Classifier	1 x $4b$ x 128	1 x 1 x C
$s = 2$ Classifier	1 x $16b$ x 128	1 x 1 x C
Multi-resolution Concatenation	1 x $\{b, 4b, 16b\}$ x 128	1 x $16b$ x 384
$s = m$ Classifier	1 x $16b$ x 384	1 x 1 x C

B.3 Hyperparameters

Optuna was used for hyperparameter optimisation (Akiba et al., 2019) — 40 trials with pruning using the Tree-structured Parzen Estimator sampler (Bergstra et al., 2011). Hyperparameter tuning was only carried out on the DeepGlobe dataset, i.e., the same hyperparameters were used for the FloodNet models. This demonstrates that the hyperparameters found this tuning process are robust, which is also supported by consistent values across the S2P models.The resulting hyperparameters are given in Table B.9.

TABLE B.9: Model training hyperparameters.

Configuration	Max Epochs	Learning Rate	Weight Decay	Dropout
ResNet18	30	0.05	0.10	N/A
U-Net 224	30	5×10^{-4}	1×10^{-5}	0.25
U-Net 448	30	5×10^{-4}	1×10^{-6}	0.2
S2P SR Small 8	30	1×10^{-4}	1×10^{-6}	0.05
S2P SR Medium 8	30	1×10^{-4}	1×10^{-5}	0.35
S2P SR Large 8	30	1×10^{-4}	1×10^{-5}	0.25
S2P SR Small 16	30	5×10^{-4}	1×10^{-6}	0.10
S2P SR Medium 16	30	1×10^{-4}	1×10^{-6}	0.05
S2P SR Large 16	30	1×10^{-4}	1×10^{-5}	0.35
S2P SR Small 32	30	1×10^{-4}	1×10^{-6}	0.25
S2P SR Medium 32	30	1×10^{-4}	1×10^{-4}	0.00
S2P SR Large 32	30	1×10^{-4}	1×10^{-6}	0.40
S2P MRSO	40	1×10^{-4}	1×10^{-6}	0.20
S2P MRMO	40	1×10^{-4}	1×10^{-6}	0.10

Appendix C

Appendix for Non-Markovian Reward Modelling from Trajectory Labels via Interpretable Multiple Instance Learning

C.1 Implementation and Resource Details

Code for this work is available in a publicly accessible GitHub repository: https://github.com/JAEarly/MIL-for-Non-Markovian-Reward-Modelling. This work was implemented in Python 3.8 / 3.10 and the Machine Learning (ML) functionality used PyTorch. All required libraries for our work are given in a `requirements.txt` file in the code repository. The majority of Multiple Instance Learning (MIL) model training was carried out on a remote Graphics Processing Unit (GPU) service using a Volta V100 Enterprise Compute GPU with 16GB of VRAM, which utilised CUDA v11.0 to enable GPU support (IRIDIS 5, University of Southampton). For the Lunar Lander task, training each MIL model took a maximum of eight hours. For the other tasks, this was a maximum of two hours. Trained models are included alongside the code. Fixed seeds were used to ensure consistency of dataset splits between training and testing; these are included in the scripts that are used to run the experiments. All our datasets were generated from code; both the scripts to generate the data and the derived datasets themselves are included alongside our model training code. Dataset generation, as well as all Reinforcement Learning (RL) agent training, was conducted on a second remote GPU service using a compute node with two Nvidia Pascal P100 cards. Data generation took a maximum of three hours per dataset (BlueCrystal Phase 4, University of Bristol). Agent training for the 2D navigation tasks was computationally light, requiring 8-12 minutes per 400-episode run, although we

completed ten repeat runs for each permutation of task and MIL model architecture (one per MIL training repeat). 800-episode RL runs for Lunar Lander took approximately two hours each. Further details on executing the scripts to reproduce our results can be found in the `README.md` file in our code repository.

C.2 Use Cases for Non-Markovian Reward Modelling

The non-Markovian reward formulation applies to cases where rewards depend on hidden state information $\mathbf{h}_t$ in addition to environment states $\mathbf{s}_t$ and actions $\mathbf{a}_t$, and this information is a function of previous state-action pairs but *not* vice versa (i.e., there is no causal path from $\mathbf{h}_t$ to $\mathbf{s}_{t+i}$ for any $i > 1$). This crucial caveat distinguishes the formulation from the more general class of partially observable MDPs and demarcates the set of domains to which it can be applied: those involving a secondary Markovian system that "spectates" on events in the environment without intervening. In the Reward Modelling (RM) context, this secondary system is a black box (making its internal state $\mathbf{h}_t$ hidden) and explicit rewards are unavailable, being replaced by a sparser and potentially noisy form of reward-dependent feedback (trajectory return labels in our work). Below we identify three classes of use cases which fit this technical specification and provide one concrete example for each:

Ambiguous Subtasks Cases involving an extended task with a sequential structure, where it is hard to formally define the conditions for subtask completion, but RM is feasible because a human "knows it when they see it". Here, the hidden state to be learnt represents the current subtask and any auxiliary information needed to determine its completion status.

- **Concrete Example:** Using judges' scores to learn a performative display (e.g., gymnastics, aerobatics) chaining several manoeuvres whose start and end conditions are difficult to formalise *a priori*. This could be considered as an extension of the single backflip task studied in the foundational RM work by Christiano et al. (2017).

Dependencies on Subjective Affect Cases where a human's reward function is dependent on their affective (emotional) status, which in turn depends on their prior experiences. Assuming this information is not directly available in the observed environment state, it must be inferred from data.

- **Concrete Example:** Using periodic satisfaction ratings to train a personal assistant robot whose owner's mood, needs and preferences vary from day to day. These

variations may influence the preferred driving style of a chauffeur service or choice of evening meal.

Irrationalities/Cognitive Biases Cases where one or more forms of bias colour a human's post hoc rating of an observed trajectory, even if their instantaneously-experienced reward is Markovian. Psychological studies of how humans aggregate immediate rewards into retrospective evaluations of the quality of an experience find that a straight summation assumption is unrealistic, with subjects exhibiting high sensitivity to contrast effects and recency bias (collectively termed the peak-end rule) (Kahneman, 2000), and factoring in anticipated future states in addition to those actually observed (Ariely and Carmon, 2000).

Here, the hidden state captures an aggregate representation of the biases at play in a given human's evaluation.[1]

- **Concrete Example:** Using customer "star ratings" to improve a holiday planning agent whose recommendations aim to account for an unknown mix of biases such as the peak-end rule. The agent may use the learnt bias model to prioritise key moments in a holiday when managing the travel schedule and budget, in order to maximise future customers' star ratings.

C.3 Model Architectures

C.3.1 Toy Dataset Multiple Instance Learning Model Architectures

In Tables C.1 to C.4 we give the architectures for the MIL models we used in the toy dataset experiments. The models are a combination of Fully Connected (FC) layers along with different MIL pooling mechanisms. Rectified Linear Unit (ReLU) activation is applied to the FC hidden layers. We label the layers based on the part of the network they belong to: Feature Extractor (FE), Head Network (HN), or Pooling (P); see Section 6.3.3. Where used, MIL Long Short-Term Memory (LSTM) components are indicated in the pooling layers. We also give the input and output sizes: T x n

[1] A fascinating philosophical question arises here. When (following the method of Section 6.3.2) an RL agent is trained using a non-Markovian RM model that captures a cognitive bias, should the agent learn to maximise rewards *including* the bias (which, for example, might lead it to prioritise its peak and final reward rather than seek uniformly good performance), or *excluding* it (which would revert to uniform prioritisation). This issue of whether intelligent agents should seek to exploit human irrationalities when optimising for their revealed preferences, or appeal to their unbiased "better angels", relates to distinctions between first- and second-order preferences (Stanovich, 2008) or between experienced and remembered (or decision) utility (Berridge and O'Doherty, 2014). We defer this question to those with more relevant expertise but note that regardless of the answer, it is essential to include the biases in the reward model using a method such as the one proposed in this work.

represents an input or output where there is a representation of size n for each step in a trajectory of length T (the number of instances T).

TABLE C.1: Toy `Instance`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU	T x 2	T x 2
2 (HN)	FC	T x 2	T x 1
3 (P)	mil-sum	T x 1	1

TABLE C.2: Toy `EmbeddingLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU	T x 2	T x 2
2 (P)	mil-emb-lstm	T x 2	2
3 (HN)	FC	2	1

TABLE C.3: Toy `InstanceLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU	T x 2	T x 2
2 (P-1)	mil-ins-lstm	T x 2	T x 2
3 (HN)	FC	T x 2	T x 1
4 (P-2)	mil-sum	T x 1	1

TABLE C.4: Toy `CSCInstanceLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU	T x 2	T x 2
2 (P-1)	mil-csc-ins-lstm	T x 2	T x 2
3 (HN)	FC	T x 2	T x 1
4 (P-2)	mil-sum	T x 1	1

C.3.2 Advanced RL MIL Model Architectures

In Tables C.5 to C.8 we give the architectures for the MIL models we used in the Timer, Moving, Key, and Charger tasks. Input features were normalised using mean/standard deviation scaling.

TABLE C.5: RL `Instance`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 64
2 (FE)	FC + ReLU + DO	T x 64	T x 32
3 (FE)	FC + ReLU + DO	T x 32	T x 32
4 (HN)	FC + ReLU + DO	T x 32	T x 32
5 (HN)	FC + ReLU + DO	T x 32	T x 16
6 (HN)	FC	T x 16	T x 1
7 (P)	mil-sum	T x 1	1

TABLE C.6: RL `EmbeddingLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 64
2 (FE)	FC + ReLU + DO	T x 64	T x 32
3 (FE)	FC + ReLU + DO	T x 32	T x 32
4 (P)	mil-emb-lstm	T x 32	2
5 (HN)	FC + ReLU + DO	2	32
6 (HN)	FC + ReLU + DO	32	16
7 (HN)	FC	16	1

For the Lunar Lander task, we used similar architectures to the other four RL tasks but with larger layers; see Tables C.9 to C.12. The depth of the models remained the same, as did the size of the hidden state embedding. Also note the addition of the Leaky ReLU activation function in the HNs, which is discussed in Section 6.6.3.

TABLE C.7: RL `InstanceLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 64
2 (FE)	FC + ReLU + DO	T x 64	T x 32
3 (FE)	FC + ReLU + DO	T x 32	T x 32
4 (P)	mil-ins-lstm	T x 32	T x 2
5 (HN)	FC + ReLU + DO	T x 2	T x 32
6 (HN)	FC + ReLU + DO	T x 32	T x 16
7 (HN)	FC	T x 16	T x 1
8 (P)	mil-sum	T x 1	1

TABLE C.8: RL `CSCInstanceLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 64
2 (FE)	FC + ReLU + DO	T x 64	T x 32
3 (FE)	FC + ReLU + DO	T x 32	T x 32
4 (P)	mil-csc-ins-lstm	T x 32	T x 2
5 (HN)	FC + ReLU + DO	T x 34	T x 32
6 (HN)	FC + ReLU + DO	T x 32	T x 16
7 (HN)	FC	T x 16	T x 1
8 (P)	mil-sum	T x 1	1

TABLE C.9: Lunar Lander `Instance`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 128
2 (FE)	FC + ReLU + DO	T x 128	T x 64
3 (FE)	FC + ReLU + DO	T x 64	T x 64
4 (HN)	FC + ReLU + DO	T x 64	T x 64
5 (HN)	FC + ReLU + DO	T x 64	T x 32
6 (HN)	FC + Leaky ReLU	T x 32	T x 1
7 (P)	mil-sum	T x 1	1

TABLE C.10: LL `EmbeddingLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 128
2 (FE)	FC + ReLU + DO	T x 128	T x 64
3 (FE)	FC + ReLU + DO	T x 64	T x 64
4 (P)	mil-emb-lstm	T x 64	2
5 (HN)	FC + ReLU + DO	2	64
6 (HN)	FC + ReLU + DO	64	32
7 (HN)	FC + Leaky ReLU	32	1

TABLE C.11: LL `InstanceLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 128
2 (FE)	FC + ReLU + DO	T x 128	T x 64
3 (FE)	FC + ReLU + DO	T x 64	T x 64
4 (P)	mil-ins-lstm	T x 64	T x 2
5 (HN)	FC + ReLU + DO	T x 2	T x 64
6 (HN)	FC + ReLU + DO	T x 64	T x 32
7 (HN)	FC + Leaky ReLU	T x 32	T x 1
8 (P)	mil-sum	T x 1	1

TABLE C.12: LL `CSCInstanceLSTM`.

Layer	Type	Input	Output
1 (FE)	FC + ReLU + DO	T x 2	T x 128
2 (FE)	FC + ReLU + DO	T x 128	T x 64
3 (FE)	FC + ReLU + DO	T x 64	T x 64
4 (P)	mil-csc-ins-lstm	T x 64	T x 2
5 (HN)	FC + ReLU + DO	T x 66	T x 64
6 (HN)	FC + ReLU + DO	T x 64	T x 32
7 (HN)	FC + Leaky ReLU	T x 32	T x 1
8 (P)	mil-sum	T x 1	1

C.4 Further Details on Reinforcement Learning Training

Adopting OpenAI Gym terminology (Brockman et al., 2016), non-Markovian reward functions (both ground truth oracles and learnt LSTM-based models) are implemented as *wrappers* on rewardless base environments. The role of a wrapper is to track the hidden state of either the oracle or the LSTM throughout an episode and use this to compute rewards to return to the agent. In the "Oracle (without hidden state)" baseline, we return the raw environment state (e.g., the 2D position $[x_t, y_t]$) to the agent unmodified. Otherwise, we concatenate the post-update hidden state h_{t+1} onto the end of the environment state, thereby expanding the state space from the agent's perspective and making rewards Markovian.

This wrapper-based approach allows us to use a completely vanilla RL algorithm. For the 2D navigation tasks, we use a Deep Q-Network (DQN) agent (Mnih et al., 2015) with the double Q-learning trick (Van Hasselt et al., 2016) enabled. For Lunar Lander, which has a continuous action space, we use Soft Actor-Critic (SAC) (Haarnoja et al., 2018). In both cases, we use a value network with ReLU activation functions, which is updated on every timestep by sampling batches of size 128 from a replay buffer. Bellman updates use a discount factor of $\gamma = 0.99$ and are implemented by the Adam optimiser with a learning rate of $1e^{-3}$. A target network tracks the primary one by Polyak averaging of parameters with a coefficient of $5e^{-3}$ per timestep. Additional hyperparameters are given below:

- **2D tasks (DQN):** number of training episodes $= 400$; replay buffer capacity $= 5e^{4}$; value network hidden layer sizes $= [256, 128, 64]$; policy greediness ϵ linearly decayed from 1 to 0.05 over the first 200 episodes and held constant thereafter.
- **Lunar Lander (SAC):** number of training episodes $= 800$; replay buffer capacity $= 1e^{5}$; value/policy network hidden layer sizes $= [256, 256]$; policy entropy regularisation coefficient $\alpha = 0.2$; policy updates with Adam optimiser (learning rate $1e^{-4}$).

Appendix D

Appendix for Inherently Interpretable Time Series Classification via Multiple Instance Learning

D.1 Implementation Details

The code for this project was implemented in Python 3.8, with PyTorch as the main library for Machine Learning (ML). A standalone code release is available at: https://github.com/JAEarly/MILTimeSeriesClassification. This includes our synthetic dataset and the ability to use our *plug-and-play* `MILLET` models. Model training was performed using an NVIDIA Tesla V100 Graphics Processing Unit (GPU) with 16GB of VRAM and CUDA v12.0 to enable GPU support. For reproducibility, all experiments with a stochastic nature (e.g., model training, synthetic data generation, and sample selection) used pseudo-random fixed seeds. A list of all required libraries is given in the code release mentioned above.

D.2 Model Details

D.2.1 `MILLET` Model Architectures

In the following section, we provide details on our `MILLET` models. For conciseness, we omit details on the backbone architectures. See Wang et al. (2017) for details on `FCN` and `ResNet`, and Ismail Fawaz et al. (2020) for details on `InceptionTime`. Each of the three Feature Extractor (FE) backbones used in this work produces fixed-size time point

embeddings of length $d = 128$: $\mathbf{Z} \in \mathbb{R}^{T \times 128} = [\mathbf{x}_1, \ldots, \mathbf{x}_T]$, where T is the input time series length. This is the initial *Feature Extraction* phase and uses unchanged versions of the original backbones.

A breakdown of the general model structure is given in Equation D.1. Note how *Positional Encoding* is only applied after *Feature Extraction*. Furthermore, *Positional Encoding* and *Dropout* are only applied in `MILLET` models, i.e., when using `Instance`, `Attention`, `Additive`, or `Conjunctive` pooling.

$$\textit{Feature Extraction} \rightarrow \textit{Positional Encoding} \rightarrow \textit{Dropout} \rightarrow \textit{MIL Pooling} \tag{D.1}$$

Below we detail the *Positional Encoding* processes and then give architectures for each of the *Multiple Instance Learning (MIL) Pooling* approaches. These are kept unchanged across the backbones.

D.2.2 Positional Encoding

Our approach for *Positional Encoding* uses fixed positional encodings (Vaswani et al., 2017):

$$\begin{aligned} PE_{(t,2i)} &= \sin(t/10000^{2i/d}), \\ PE_{(t,2i+1)} &= \cos(t/10000^{2i/d}), \end{aligned} \tag{D.2}$$

where $t = [1, \ldots, T]$ is the position in the time series, $d = 128$ is the size of the time point embeddings, and $i = [1, \ldots, d/2]$. As such, the *Positional Encoding* output is the same shape as the input time point embeddings ($\mathbb{R}^{T \times 128}$). The positional encodings are then simply added to the time point embeddings.

In cases where time points are removed from the bag, e.g., when calculating Area Over the Perturbation Curve to Random (AOPCR) (see Section 7.3.6), we ensure the positional encodings remain the same for the time points that are still in the bag. For example, if the first 20 time points are removed from a time series, the 21$^{\text{st}}$ time point will still have positional encoding $PE_{(21)}$, not $PE_{(1)}$.

D.2.3 Multiple Instance Learning Pooling Architectures

In Tables D.1 to D.5 we provide architectures for the different `MILLET` pooling methods used in this work (see Figure 7.2 for illustrations). Each row describes a layer in the pooling architecture. In each case, the input is a bag of time point embeddings

(potentially with positional encodings and dropout already applied, which does not change the input shape; see Appendix D.2.2). The input is batched with a batch size of b, and each time series is assumed to have the same length T. Therefore, the input is four-dimensional: batch size $\times$ number of channels $\times$ time series length $\times$ embedding size. However, in this work, as we are using univariate time series, the number of channels is always one. The problem has C classes, and the pooling methods produce logit outputs – softmax is later applied as necessary.

TABLE D.1: MIL pooling: `Embedding` (GAP).

Process	Layer	Input	Output
Pooling	Mean	$b \times 1 \times T \times 128$ (Time Point Embs.)	$b \times 1 \times 1 \times 128$ (TS Emb.)
Classifier	Linear	$b \times 1 \times 1 \times 128$ (Time Series Emb.)	$b \times 1 \times 1 \times C$ (TS Pred.)

TABLE D.2: `MILLET` pooling: `Attention`. We use an internal dimension of 8 in the attention head and apply sigmoid rather than softmax due to the possibility of long time series. Attention weighting scales the time point embeddings by their respective attention scores.

Process	Layer	Input	Output
Attention	Linear + tanh	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times 8$
	Linear + sigmoid	$b \times 1 \times T \times 8$	$b \times 1 \times T \times 1$ (Attn. Scores)
Pooling	Attn. Weighting	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times 128$
	Mean	$b \times 1 \times T \times 128$	$b \times 1 \times 1 \times 128$ (TS Emb.)
Classifier	Linear	$b \times 1 \times 1 \times 128$ (TS Emb.)	$b \times 1 \times 1 \times C$ (TS Pred.)

TABLE D.3: `MILLET` pooling: `Instance`.

Process	Layer	Input	Output
Classifier	Linear	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times C$ (TP Preds.)
Pooling	Mean	$b \times 1 \times T \times C$ (TP Preds.)	$b \times 1 \times 1 \times C$ (TS Pred.)

TABLE D.4: `MILLET` pooling: `Additive`.

Process	Layer	Input	Output
Attention	Linear + tanh	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times 8$
	Linear + sigmoid	$b \times 1 \times T \times 8$	$b \times 1 \times T \times 1$ (Attn. Scores)
Classifier	Attn. Weighting	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times 128$
	Linear	$b \times 1 \times T \times 128$	$b \times 1 \times T \times C$ (TP Preds.)
Pooling	Mean	$b \times 1 \times T \times C$ (TP Preds.)	$b \times 1 \times 1 \times C$ (TS Pred.)

TABLE D.5: MIL pooling: `Conjunctive`. In this case, attention weighting scales the time point predictions by their respective attention scores, rather than scaling the embeddings.

Process	Layer	Input	Output
Attention	Linear + tanh	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times 8$
	Linear + sigmoid	$b \times 1 \times T \times 8$	$b \times 1 \times T \times 1$ (Attn. Scores)
Classifier	Linear	$b \times 1 \times T \times 128$ (TP Embs.)	$b \times 1 \times T \times C$ (TP Preds.)
Pooling	Attn. Weighting	$b \times 1 \times T \times C$ (TP Preds.)	$b \times 1 \times T \times C$
	Mean	$b \times 1 \times T \times C$	$b \times 1 \times 1 \times C$ (TS Pred.)

D.2.4 Training and Hyperparameters

In this work, all models were trained in the same manner. We used the Adam optimiser with a fixed learning rate of 0.001 for 1500 epochs and trained to minimise cross-entropy loss. Training was performed in an end-to-end manner, i.e., all parts of the networks (including the backbone FE layers) were trained together, and no pre-training or fine-tuning was used. Dropout (if used) was set to 0.1, and batch size was set to $\min(16, \lfloor \textit{num training time series}/10 \rfloor)$ to account for datasets with small training set sizes. For example, if a dataset contains only 100 training time series, the batch size is set to 10.

No tuning of hyperparameters was used – values were set based on those used for training the original backbone models. As the existing Deep Learning (DL) Time Series Classification (TSC) methods used fixed hyperparameters, our decision not to tune the hyperparameters facilitates a fairer comparison. It also has the benefit of providing a robust set of default values for use in derivative works. However, better performance could be achieved by tuning hyperparameters for individual datasets. We would expect hyperparameter tuning for `MILLET` to have a greater impact than doing so for the Global Average Pooling (GAP) versions of the models as `MILLET` provides more scope for tuning (e.g., dropout, attention-head size, and whether to include positional encodings). As greater flexibility is achieved by having the ability to add/remove/tune the additional elements of `MILLET`, this would lead to more specialised models (and larger performance improvements) for each dataset. For example, in our ablation study (Section 7.6.4), we found that positional encoding was beneficial for some datasets but not others.

No validation datasets were used during training or evaluation. Instead, the final model weights were selected based on the epoch that provides the lowest training loss. As such, training was terminated early if a loss of zero was reached (which was a very rare occurrence, but did happen). Models started with random weight initialisations, but pseudo-random fixed seeds were used to enable reproducibility. For repeat training, the

seeds were different for each repeat (i.e., starting from different random initialisations), but these were consistent across models and datasets.

D.3 SHAP Details

In our Shapley Additive Explanations (SHAP) implementation we used random sampling of coalitions. Guided sampling (selecting coalitions to maximise the SHAP kernel) would have proved too expensive: the first coalitions sampled would be all the single time point coalitions and all the $T-1$ length coalitions (for a time series of length T), which results in $2T$ coalitions and thus $2T$ calls to the model. In the case of *WebTraffic*, this would be 2016 samples, rather than the 500 we used with random sampling (which still took far longer to run than `MILLET`, see Section 7.6.3.2). Furthermore, Early et al. (2022c, Chapter 4) showed random sampling to be equal to or better than guided sampling in some cases.

D.4 UCR Dataset Details

For the UCR datasets, we used the original train/test splits as provided by the archive source.[1] z-normalisation was applied to datasets that were not already normalised. The exhaustive list of univariate UCR datasets used in this work is:

`Adiac`, `ArrowHead`, `Beef`, `BeetleFly`, `BirdChicken`, `Car`, `CBF`, `ChlorineConcentration`, `CinCECGTorso`, `Coffee`, `Computers`, `CricketX`, `CricketY`, `CricketZ`, `DiatomSizeReduction`, `DistalPhalanxOutlineAgeGroup`, `DistalPhalanxOutlineCorrect`, `DistalPhalanxTW`, `Earthquakes`, `ECG200`, `ECG5000`, `ECGFiveDays`, `ElectricDevices`, `FaceAll`, `FaceFour`, `FacesUCR`, `FiftyWords`, `Fish`, `FordA`, `FordB`, `GunPoint`, `Ham`, `HandOutlines`, `Haptics`, `Herring`, `InlineSkate`, `InsectWingbeatSound`, `ItalyPowerDemand`, `LargeKitchenAppliances`, `Lightning2`, `Lightning7`, `Mallat`, `Meat`, `MedicalImages`, `MiddlePhalanxOutlineAgeGroup`, `MiddlePhalanxOutlineCorrect`, `MiddlePhalanxTW`, `MoteStrain`, `NonInvasiveFetalECGThorax1`, `NonInvasiveFetalECGThorax2`, `OliveOil`, `OSULeaf`, `PhalangesOutlinesCorrect`, `Phoneme`, `Plane`, `ProximalPhalanxOutlineAgeGroup`, `ProximalPhalanxOutlineCorrect`, `ProximalPhalanxTW`, `RefrigerationDevices`, `ScreenType`, `ShapeletSim`, `ShapesAll`, `SmallKitchenAppliances`, `SonyAIBORobotSurface1`, `SonyAIBORobotSurface2`, `StarLightCurves`, `Strawberry`, `SwedishLeaf`, `Symbols`, `SyntheticControl`, `ToeSegmentation1`, `ToeSegmentation2`, `Trace`, `TwoLeadECG`, `TwoPatterns`, `UWaveGestureLibraryAll`, `UWaveGestureLibraryX`, `UWaveGestureLibraryY`, `UWaveGestureLibraryZ`, `Wafer`, `Wine`, `WordSynonyms`, `Worms`, `WormsTwoClass`, `Yoga`.

[1] https://www.cs.ucr.edu/~eamonn/time_series_data_2018/