Human Interfaces with Machine Learning Recognition Systems

Connor Clarkson

Submitted to Swansea University in fulfilment
of the requirements for the Degree of Doctor of Philosophy
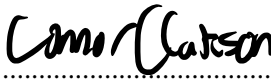
Swansea University
Prifysgol Abertawe

School of Mathematics and Computer Science
Swansea University

June 3, 2025

Declarations

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed .... (candidate)

Date ....03/06/2025........................

This thesis is the result of my own investigations, except where otherwise stated. Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed .... (candidate)

Date ....03/06/2025........................

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed .... (candidate)

Date ....03/06/2025........................

The University's ethical procedures have been followed and, where appropriate, that ethical approval has been granted.

Signed .... (candidate)

Date ....03/06/2025........................

I would like to dedicate this thesis to Lillian Doreen Morgan and Rebecca L. Clarkson.

Abstract

Building large pools of data has become a relatively straightforward task, with many automated ways of obtaining different sources of data. Labelling such data has resulted in becoming an exponential problem, both in terms of time and in the form of an interaction-heavy task. This task only becomes exponential with feature-rich structures of data and labelling systems, as well as requiring more advanced expertise for many different domains of a task to model. A prominent set of techniques utilising this data, and large networks have reformed machine learning into what we call deep learning today. Within this field, we can form levels of supervision that allow for stronger signals of inductive bias for both deep network architectures and in the training scheme. In this work, we explore both types with the target application and domain being the manufacturing of steel.

Firstly, we present an exploratory approach to assist in decision-making for the task of clustering by utilising the feature-rich representations provided by generative models. By forming it as a semi-supervised problem we can provide varying degrees of supervision to enhance performance as a form of inductive bias into the training scheme. Supervision can be formalised into labels from data or in an active learning setting where we request help from an expert. If we are required to make a request, then we must provide information and visualisations so that an accurate decision can be made.

Following this, in our second body of work we extend on an active learning setting by introducing a new acquisition function based on the distance from different representations. We apply it to a data refinement strategy where we fix mistakes in bounding-box labelled datasets to form a dense segmentation. Different forms of user interaction provide different levels of information to the training scheme, we explore the effects of these user interactions on the performance of this refinement task.

Lastly, we apply stronger forms of inductive bias into the network architecture by modelling hierarchical labelling systems, where such relationships between labels form an abstraction and fine-grained level of the data. Inspired by the structure of human cognition and perception where we recognise patterns of various levels of abstraction to define an object. By invoking an explicit form of deep learning with feature-rich structures like graphs we can model these interconnected labels. We define two types of hierarchical relationships: the first is a break-up of the physical or geometric structure of the object, referred to as an encapsulation relationship. The second is sub-classification relationships which are semantic relations of labels provided by domain knowledge of what we are trying to capture in the dataset. We utilise both to solve classification and segmentation tasks.

Acknowledgements

First and foremost I would like to thank both my supervisors Dr. Mike Edwards and Prof. Xianghua Xie. I first met Mike when he was a PhD student and I was an (extremely innocent) first year in the computer science undergraduate course, back in 2015. He would later accept me as his student in both MSc programmes and eventually as a doctoral student. I could always rely on him for advice and moral support, especially during some of the dark times over the years. He would always give his time, be forthcoming with his kindness and reassurance in me to succeed. I'm indebted and very grateful to him for both personal and professional development, even though he abandoned me for a better job. To Xianghua, I can't thank you enough for picking up the pieces of my PhD and getting me over the finish line. Your clear level-headiness and support were incurable over the last year and a half. Even in times when I just wanted to give up your support never sank for me. Both my supervisors were always patient and encouraging, to the point where I struggle to articulate just how thankful I am.

Just before starting my final year of the PhD, I took a full-time job at Technocamps. I'm grateful to Prof. Faron Mollor, Luke Clement and Casey Hopkins for their support in giving me time to finish the PhD. I wouldn't have been able to get here without your help. To Chess Hutin, Dr. Maria Mollor, Dr. Olga Petrovska, Dr. Lee Clift and Jack Roberts. I can't thank you guys enough, especially for the nudges to finish my PhD and stop procrastinating. In particular, I would like to thank Maria for her wisdom, I don't think she realises it but her advice is amazing and much appreciated. I would also like to put some focus on Chess, your kindness yet iron will for honesty is inspiring and incredible to witness.

A huge thank you to the PhD students I started this journey with: Katarzyna Szymaniak, Dr. Elif Firat, Dr. Anna Carter, Dr. Emily Nielsen, Fergus Pick and Luke Thomas. Each of you

contributed to this. Two in particular I would like to thank are Dr. Filippos Pantekis and (soon to be Dr) Ben Lloyd-Roberts. I met Filippos when I was in my second year of the BSc but we did not truly become friends until I joined Technocamps. I enjoy our debates, how we tease each other, rants about Python, tear-shedding sessions, ideas of implementation and even the late nights in the office. It's a shame his car is too small. Ben and I started the BSc together and then later started our PhDs at the same time. We have been through many ups and downs together, I understand how much of a pain I can be and just want to thank you for your support and the time we spent.

I would also like to thank Dr. Liam O'Reilly and Dr. Phil James. The funny remarks and just general sense of humour from both of you were great throughout the years! Even when I felt down I can always rely on both of you making me feel better. I feel very lucky to have mentors with such wisdom, kindness and passion. Liam's eagerness for education is truly inspiring, his knowledge of computer science and engineering for software and hardware is unmatched, and his music taste is, ok. One challenging point of my PhD was seeing Phil leave not only the university but also academia. A mentor, colleague and friend making very difficult choices. He demonstrates great courage, something I aspire to one day have.

My deepest gratitude goes to my long-suffering and cherished mother, Rebecca, and my sister Rhyannon. I tend to spend a lot of time on my work and not enough time with both of you. Your unwavering and loving support made this possible.

Contents

List of Publications

The following publications resulted from the work conducted for this doctoral thesis. An outline of contributions may be read in section 1.3.

1. Connor Clarkson, Michael Edwards, and Xianghua Xie. "Active Anchors". In: *Companion Proceedings of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS '23 Companion. Swansea, United Kingdom: Association for Computing Machinery, 2023, pp. 68–69. ISBN: 9798400702068. DOI: `10.1145/3596454.3597185`

2. Connor Clarkson, Michael Edwards, and Xianghua Xie. "Active Anchors: Similarity Based Refinement Learning". In: *Proceedings of the International Conference on Applied Computing*. 2023, pp. 47–57. ISBN: 978-989-8704-53-5

3. Connor Clarkson, Michael Edwards, and Xianghua Xie. "Dense Semantic Refinement Using Active Similarity Learning". In: *IADIS International Journal on Computer Science and Information Systems*. 2024, pp. 15–30. ISBN: 1646-3692

4. Connor Clarkson, Michael Edwards, and Xianghua Xie. "Modelling on Types of Hierarchical Relationships". (To be published)

List of Acronyms

t-**SNE** t-distributed stochastic neighbour embedding.

CNN Convolutional Neural Network.

DBSCAN Density-Based Spatial Clustering of Applications with Noise.

Faster R-CNN Faster Region-based Convolutional Neural Network.

GAN Generative Adversarial Network.

GNN Graph Neural Network.

GraphSAGE Graph Sample and Aggregation.

IOU Intersection Over Unions.

KL Kullback Leibler Divergence.

ML-kNN Multi-Label K-Nearest Neighbours.

MPNN Message-Passing Neural Network.

NMS Non-Max Suppression.

PCA Principal Component Analysis.

R-CNN Region-based Convolutional Network.

ReLU Rectified Linear Unit.

ResNet Residual Network.

RNN Recurrent Neural Network.

ROI Region of Interest.

RPN Region Proposal Network.

SVM Support Vector Machine.

UMAP Uniform Manifold Approximation and Projection.

VGG Visual Geometry Group.

VLAD Vector of Locally Aggregated Descriptors.

YOLO You Only Look Once.

List of Tables

List of Figures

Chapter 1

Introduction

Contents

1.1 Motivations

A prominent set of techniques utilising data and large networks have reformed machine learning into what we call deep learning today. Stemming from its ability to learn hierarchical representations, which enables an understanding and processing of information that can mimic human cognition [78]. However, the true potential of deep learning lies in its flexibility to incorporate different forms of supervision and architectural choices, which can significantly enhance the inductive bias of models. This refers to a set of assumptions a learning algorithm uses to predict outputs given inputs. We utilise this bias in deep learning to perform well on unseen data. Different types of supervision provide varying degrees of inductive bias, allowing models to be specific to a problem domain [9]. Deep learning was originally restricted to regular domains (such as images) with the most common type of training involving full supervision, where large datasets of labelled examples are used to train models [78]. While effective, this approach can be costly and time-consuming, especially in domains where expert annotation is required [31]. To address the limitations of full supervision, semi-supervised techniques have gained prominence, where we leverage both labelled and unlabelled data, allowing models to learn from the inherent structure of data itself, leading to more robust representations and improved generalisation in situations when labelled data is scarce or incorrect. Other forms of supervision can also come from an expert in an active learning setting. By allowing the model to interactively query for the most informative samples to be labelled, active learning can significantly reduce the amount of labelled data required while maintaining or even improving model performance. The choice of network architecture also plays a role in determining the inductive bias of a model. Graph deep learning, in particular, has emerged as a powerful framework for capturing complex relational structures in data. This motivates the idea that many real-world problems involve data with irregular structures. We can incorporate graph structures into the learning process that forms a strong inductive bias that can lead to more sample-efficient learning and better generalisation due to rich forms of information provided in the training.

1.1.1 Exploratory Analysis on Data Representations For Human-based Decision Making

The ability to make an informed decision based on complex information has become crucial across various domains, from business and healthcare to public policy and scientific research.

The sheer volume and complexity of data often pose a significant challenge to human decision-making. Data representations plays role of bridging the gap between raw data and human comprehension. Effective representations can significantly reduce cognitive load, highlight relevant patterns, and facilitate intuitive understanding of complex relationships within the data [21]. However, determining the most appropriate representation for a given decision-making context remains a challenging task, as it depends on the nature of the data, the specific decision at hand, the complexity of knowledge required by the decision-maker. While attention has been made on data visualisation and decision support systems, there remains a need for systematic exploration of how different data representations impact human decision-making processes, particularly within active learning frameworks. Active learning systems need to present queries in a way that maximises information gain while minimising the amount of effect from the expert. Exploring data representations can help identify optimal query presentation strategies. Certain representations may inadvertently introduce or amplify biases in the decision-making. Exploratory analysis can help identify and mitigate this bias.

1.1.2 Defect Detection in Manufacturing

Due to moving towards industry 4.0, manufacturing processes are increasingly becoming complex and automated. This results in a demand for high-quality products has never been higher, with consumers and industries alike expecting near-perfect reliability and consistency. As context relies on accurate defect detection to ensure this reliability and consistency. Like in many applications and domains, deep learning has become a common direction in deploying defect detection tasks. Many defects are subtle and difficult to detect, this problem becomes even more challenging when using composite materials such as in steel manufacturing. Depending on the composite structure different types of defects can form, even with the same label the visual features of these defects may look different. As the material moves on the conveyor belt, where we apply different operations, defects can also evolve into new types as its to cover large parts of the material. This has resulted in highly specific inspection solutions for a given operation on a particular type of material. The forming of these defects can be viewed as a hierarchical relationship where defects evolve into new ones. However, these types of relationships are not used much due to the complexity of modelling such labels, there is also a limited amount of these samples. Applying deep learning to such problems can help address these challenges due to its adaptability, rich forms of hierarchical representations and continuous learning schemes.

1.1.3 Graph Deep Learning In Hierarchical Labelling Systems

Many domains are facing an increasingly large amount of data with complex relationships that from labels. These relationships exhibit both intricate structures and are multi-faceted, hierarchical categorisation. From biological taxonomies to product categorisation, hierarchical label structures are ubiquitous in real-world applications. Simultaneously, many of these domains naturally lend themselves to graph representations, where entities are interconnected in many ways. Hierarchical labels and graph-structured data present challenges and exciting opportunities within deep learning. Hierarchical labels naturally result in varying levels of granularity requiring the models to make predictions at these different depth levels. Graph deep learning has become a promising area where we can learn the structure if we form these hierarchical labels are a tree structure. Graph deep learning encodes a relational inductive bias, allowing us to capture complex interdependencies between entities. Through message-passing we can then propagate information across the graph, allowing for a rich form of information flow.

1.1.4 The Three Aims of This Work

We began this doctoral work with three main goals which were established around the summer of 2019, with necessary input from both external and internal stakeholders to design the doctoral project. The three original aims are:

A The improved detection, localisation and classification of features observed by imaging systems.

B The improved labelling and analysis of complex classes via a semi-supervised approach.

C The improved integration and use of data visualisation within a user-guided approach to improve understanding of model inference.

Goal A and B have the application of surface-level analysis in steel manufacturing while goal C is aimed at general data pipeline improvements. The current approach utilises a cascade technique to first identify proposed regions of interest followed by a subsequent classification model. For goal A our proposal was to investigate various data-driven approaches for both the current cascade technique and a proposed multi-task model of detection and classification, with a focus on representation of the labels. The current labelling strategy is intensive and requires

expert insight, especially where there are ambiguities in this domain. Many different steel plants use different names for the same surface defect, while others use the same label to mean different defects. There are also various levels of severity within certain defect classes, which can present very different textures and monologues depending on the severity. This domain also has the extra challenge of varying structure spatially across defects, for example the fringes of long lamination appearing as scratches when viewed locally (such as in the cascade technique) rather than globally. These are all reflected in the labelling strategy, which currently relies on subjective hand-labelling of individual samples from a wide array of labels. In goal B, our aim was to explore a semi-supervised approach to dataset labelling and refinement, which can also provide deeper insight into the labels of defects for a quality control process. In goal C our aim was to integrate data visualisation and human-computer interaction methodologies within our pipelines, coupling the expertise of the current and future human experts with the data-driven aspect of deep learning research. By integrating data visualisation within model training and inference, we can provide some insight into the behaviour of the model by showing similar previously observed samples or contributing features in the observation which contributed to the final classification. The feedback will help address over-training of the model as well as identifying key meta-parameters or model selection.

With these 3 goals we explore the complexities of the current range of different defect types, including subjective class boundaries and hierarchical label representations to build proposed models, with the option to feed back this information to users for a deeper understanding. This work provides a blend of deep learning and human-computer interaction for labelling systems that are difficult to assign from purely a human or a machine focused perspective but benefit from a combined approach with domain knowledge being provided in both the hierarchical labels and in an active learning setting.

1.2 Overview

Based on the goals that formed this thesis in section 1.1.4 and the motivations in section 1.1, we aim to explore levels of supervision that allow for stronger signals of inductive bias for both deep network architectures and in the training schemes. We focus on image-based datasets and application towards defect detection in manufacturing. In Chapter 3 we present an exploratory approach to performance of generative models for the task of deep learning within the manufacturing domain. The focus of this chapter is twofold: first, we explore the effectiveness of

these models in detecting and categorising steel defects, and second we present a series of exploratory analysis through a graphical user interface. The interface is designed to allow domain experts to provide reasoning on how these models detect defects, forming a bridge of interoperability between deep learning algorithms and practical applications. This form of supervision information can benefit the embedding space, improving model performance. In chapter 4, we present an acquisition function utilising the current embedding space and a refinement strategy to fix mistakes in bounding-box labelled datasets. By sampling pixels to form patches of the dataset, we can then mine this pool to get a set of the most informative ones that would better improve the classification and representations of the generative models. This refines these bounding-box labels to a dense segmentation of labels. In chapter 5 we present a network architecture to model hierarchical labelled datasets via a graph-based deep learning approach to model the feature extraction of the labels. We build a chain of classifier heads where each target has a different depth level of the hierarchical structure. Predictions from previous classifiers flow into subsequent classifiers, allowing for a richer form of information flow to capture varying degrees of granularity. We also define two types of hierarchical relationships to solve classification and segmentation tasks. The first is a break up of physical or geometric structure of the object, referred to as encapsulation relationship. The second is a sub-classification relationship which are semantic relations of labels provided by domain knowledge of what we are trying to capture in the dataset.

1.3 Contributions

The main contributions of this work are the following:

An acquisition function based on current feature representation positions. We present a new acquisition function for finding a set of samples within the dataset that gets labelled by an expert will result in the most informative update to the model within an active learning setting. This utilises the current embedding space of generative models and the triplet loss. We use mining strategies based on an anchor, a sample with the sample label as the anchor and a negative which is close to the anchor. The mining strategies are based on the distance between samples. We request the help of an expert to relabel or reinforce correct labels of negatives, which focuses training to create dense clusters of related samples. The methodology of this contribution appears in Chapter 4.

6

Refinement strategy for fuzzy-labelled datasets. We present a refinement strategy within an active learning setting to fix mistakes in bounding-box labelled datasets. By uniformly sampling pixels to form patches of images, we then mine this pool to get a set of the most informative ones that would better improve the classification and generative models. We use the classification head of these models to predict a dense segmentation over the refinement process. The methodology of this contribution appears in Chapter 4.

Incorporating explicit domain knowledge into a data-driven approach via a hierarchical labelling system. We present a network architecture to model hierarchical labelled datasets. These hierarchical labels are modelled via a graph-based deep learning approach where the leaf nodes are individual samples and the root is the full dataset. Interconnecting nodes are the aggregation of their children which forms a hierarchical relationship. As a result the nodes further up the hierarchical structure are more generalised labels and as we move down they become more specialised. These node embeddings then feed into one of three types of classifier chains which target different depth levels. Previous classifier heads are used to inform new predictions in subsequent classifiers forming a chain of information flow. We evaluate our approach with a 5-fold of MNIST, CIFAR-100, ADE-20k datasets. We also build a synthetic data generator to test edge cases of our methodology. This work also utilises two types of hierarchical relationships to solve classification and segmentation tasks. The first is a break up of the physical or geometric structure of the object, referred to as encapsulation relationships. The second is sub-classification relationships which are semantic relations of labels provided by domain knowledge of what we are trying to capture in the dataset. The methodology of this contribution appears in Chapter 5.

Detecting label collisions during the training process. We explore the use of density based deep clustering where it forms a graph. Each node represents a sample and the edges from the clusters. As clusters of samples merge this forms a collision to which we reform the labelling system. Clusters builds a hierarchical dataset where if they do form then this becomes a parent node of the two children which are colliding. If a cluster starts to separate then this forms a set of children where the cluster is the parent instead. The methodology of this contribution appears in Chapter 3.

An acquisition function for evolving graphs. This contribution expands on the density based

deep clustering approach by applying an active training scheme. This is where each node of the graph is a sample and the edges form the clusters. If clusters of nodes start to merge during the training process we request an expert to inform the model if the clusters should join together or not. If that merging does happen then this forms a hierarchical set of labels as the joining clusters form a single node while its children will represent the two clusters. The methodology of this contribution appears in Chapter 3.

The outcomes of this thesis have also contributed to several publications as outlined in the List of Publications. The contributions of each paper to the contents of the thesis are summarised below:

Connor Clarkson, Michael Edwards, and Xianghua Xie. "Active Anchors". In: *Companion Proceedings of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. **EICS '23 Companion. Swansea, United Kingdom: Association for Computing Machinery, 2023, pp. 68–69.** ISBN: **9798400702068.** DOI: `10.1145/3596454.3597185`

We demonstrate a refinement strategy within an active learning setting to fix mistakes in bounding-box labelled datasets. This is demonstration paper where we focus on different user interactions to inform the model. We measure the impact of these user interactions in the manufacturing domain. This contributes application, human-centred perspective and the user interface sections of chapter 4.

Connor Clarkson, Michael Edwards, and Xianghua Xie. "Active Anchors: Similarity Based Refinement Learning". In: *Proceedings of the International Conference on Applied Computing*. **2023, pp. 47–57.** ISBN: **978-989-8704-53-5**

We propose a new acquisition function based on the similarity of defects for refining labels over time by showing the user only the most required to be labelled. We explore different initial labels for refinement and ways in which we can feed these refinements back into the model. This contributes the methodology and the some of the results in chapter 4.

Connor Clarkson, Michael Edwards, and Xianghua Xie. "Dense Semantic Refinement

Using Active Similarity Learning". In: *IADIS International Journal on Computer Science and Information Systems*. 2024, pp. 15–30. ISBN: 1646-3692

This work is a continuation from our earlier work on Active Anchors following an invitation to IADIS International Journal on Computer Science and Information Systems. We extend on the work by presenting it in more detail, a larger discussion and more experiments. We introduce a new way to feed refinements back into the model based on the full patch instead of a single pixel. We highlight trade-off between speed to convergence over accuracy gains.

Connor Clarkson, Michael Edwards, and Xianghua Xie. "Modelling on Types of Hierarchical Relationships". (To be published)

We present a network architecture and training scheme to model hierarchical labelled datasets in both classification and segmentation tasks. We define two types of hierarchical relationships. The first is a break up of the physical or geometric structure of the object. The second is subclassification relationship which are semantic relations of labels providing domain knowledge of what we are trying to capture in the dataset. We structure our network based on the labelling system of the dataset where the root represents the full set, the leaf node represents a single sample and the interconnecting nodes define the hierarchical labels. While learning the node embeddings we also feed it into a chain of classifiers where each one targets a depth level. Previous classifier heads are used to inform new predictions in subsequent classifiers forming a chain of information flow.

1.4 Outline

The rest of this work is structured as follows:

Chapter 2 - Background: Deep Learning In this chapter we introduce and review deep learning, different problem domains within the field and the types of training schemes. The discussion then moves to limitations of conventional deep learning, to then motivate the use of graph deep learning for irregular problems. Following on we introduce graph-based tasks and convolution on graphs, discussing the many different architectures. We show the key challenges of learning on graphs and how we can blend human decision

making and more explicit training schemes to address these problems. The topics discussed in this chapter are the background knowledge required for how we address the three aims of this thesis.

Chapter 3 - Active Deep Clustering: Exploratory Approach to Assist in Decision-Making
In this chapter we introduce an exploratory analysis framework and tool for use by domain experts, with a focus on steal manufacturing. Many industrial settings build datasets with fuzzy labels, manufacturing is no extension to this rule with many Region of Interests (ROIs) containing more than one label, some going outside the bound, and others which are not labelled. We form a multi-head reconstruction and dense segmentation model around our framework to build off from our analysis on fuzzy labels. We explore different dimensionality reduction methods and user interactions to assist domain experts in labelling these datasets. Finally we form an active learning experiment to improve performance of the targeted model.

Chapter 4 - Active Anchors: Similarity Based Learning for Dataset Refinement In this chapter we propose a new acquisition function for active learning based on the similarity of defects in steal manufacturing. Labelling datasets that need to be verified by domain experts is time-consuming and a interaction-heavy task due to defect characteristics and composite nature. We build a data refinement task based around the new acquisition function where we start with no labelling and form a dense segmentation over time.

Chapter 5 - Modelling on Types of Hierarchical Relationships The topic of this chapter is to model hierarchical labels in computer vision based tasks. We explore two different types of labelling systems: encapsulation and sub-classification labels. This forms a more explicit form of learning where we can embed expert domain knowledge into the rich data structures such as in graph deep learning. We blend graph feature extraction with classifier chains to predict a path from the root of the label hierarchy to a leaf node, which represents a single instance of a dataset. We also explore how robust our approach is with synthetically generated hierarchical datasets.

Chapter 6 - Conclusions and Future Work We conclude by reviewing the outcomes of this work with the original aims and motivations. We then consider how to build from this work with ideas from other fields.

Chapter 2

Background: Deep Learning

Contents

We start our background on deep learning by introducing the main learning problems of the field and the different problem domains that are associated with each. We focus this chapter on deep learning but we do cover some aspects of machine learning, where it is relevant for supporting work to the thesis of this doctoral work. We start the second section on the development of neural computation with some initial remarks on the general formulation which then leads to feed-forward neural networks and the background information needed for modern approaches. The third section introduces the Convolutional Neural Network (CNN) and how we target different problem domains, with a focus on computer vision as this forms the majority of the thesis. The final potion this thesis focuses using neural networks in irregular domains for the use with hierarchical labelling systems, therefore we form the fourth and fifth sections on graph deep learning and active learning for different domain-specific problems.

The author gives thanks and acknowledgement to several textbooks which aided in the theoretical and practical development of deep learning, graph deep learning and active learning. The statistics, mathematics and background on deep learning owes a great deal to *Machine Learning and Pattern Recognition* by Bishop (2006), *The Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman (2009), *Deep Learning* by Goodfellow, Bengio, and Courville (2016) and *Deep Learning: Foundations and Concepts* by Bishop (2024). Many of the graph deep learning concepts and general background information was aided by *Deep Learning on Graphs* by Ma and Tang (2021), *Graph Representation Learning* by Hamilton (2020) and *Graph Deep Learning: State of the Art and Challenges* by Georgousis, Kenning and Xie (2021). Finally, some of the background information on active learning has also been supplemented by *Human-in-the-Loop Machine learning* by Monarch and *Active Learning Literature Survey* by Settles (2010).

2.1 Learning Problems

Deep learning is a specific kind of machine learning, where the difference is that neural computation approaches are only used in deep learning while machine learning includes any kind of data-driven approach. The development and application of deep learning are bond by the challenges of the type of learning problem, the nature of the data by the problem domain and the complexity of the task. For proposes of this thesis we define the following learning problems:

- **Supervised Learning**, where the aim of such problems involves predicting an outcome based on a set number of input measurements. We present the input measurements and the mapped labels at training time, the learning algorithm then learns to map input to output.

- **Unsupervised Learning**, where such problems do not have a output measurement, instead we aim to find associations or patterns among the set of input measurements that can minimise some measurable outcome. This results in not showing output measurements (also known as targets) during training time.

- **Semi-Supervised Learning**, traditionally this is a mixture of both supervised and unsupervised learning properties, therefore using both unlabelled and labelled inputs to predict an outcome during training time. In deep learning, semi-supervised problems usually refers to learning internal representations so that different sets of inputs with the same label have a similar representation, this is because unsupervised properties allow us to group inputs together in a representation space.

- **Multi-Task Learning**, where we use the same set of inputs to learn multiple tasks simultaneously. This leverages shared learnt knowledge and domain information across many tasks. Within deep learning, this learning problem is often viewed as a type of regularisation (see subsection 2.2.5 on regularisation) because a single task is restricted and using additional input data from a related task can be used to help in learning the mapping from input to output.

These learning problems are not a strict breakdown of categories, but such definitions help us select what form a learning algorithm we should take, how we form a task and what measures are required.

The objective of a learning algorithm is able to learn a task by mapping a set n inputs $X = \{x_0, x_1, ..., x_{n-1}\}$, where $x \in \mathbb{R}^n$, to a set of d outputs Y. A task is defined as how a learning algorithm should process X, where we measure from some object or event. The learning forms a optimisation process where we measure performance from an *objective function* or *loss function*. These functions measure how close the model's output is to the answer we expect. The further away from zero the *loss* is, the more incorrect the learning algorithm is and therefore we use the loss as a measure of error. We call the correct output, the *ground-truth*, if our output

predictions are wrong then adjusting the parameters of the learning algorithm is required. The derivative of the loss function with respect to each parameter allows us to find the direction in which a parameter should be changed, as well as the magnitude of its change. The space of these parameters, known as the *search space*, often there are many local minima in which we optimise to. The common problem with this process is that many of the local minima are sub-optimal and the learning algorithm could become stuck. It is often very difficult to find the global optimal due to the complexity of the search space.

2.2 Neural Computation
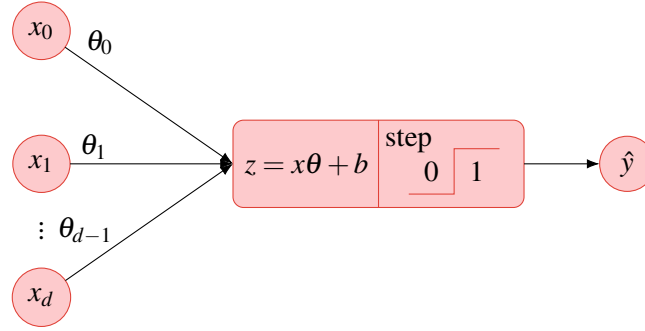
Figure 2.1: The perceptron takes in inputs $x_0, x_1, ..., x_{d-1}$ weighted by parameters $\theta_0, \theta_1, ..., \theta_{d-1}$ along with a added bias term b. The result z is passed into a step function, determining if the perceptron should activate or not.

The function $f : X \rightarrow Y$ representing a mapping from input to output, where we view this as a probability distribution $p(Y \mid X)$. The aim is then to approximate this distribution given a function $f(X)$. In learning algorithms the function is parameterised with a set of variables θ such that it takes the form:

$$f(X, \theta) = p(Y \mid X) + \varepsilon, \tag{2.1}$$

where ε is the error of the estimation. A set of parameters that best approximates $p(Y \mid X)$ is denoted θ^*. The learning algorithm then makes changes to θ to make the error as small as possible, depending on the data, the type of learning algorithm and task used can make this a challenge to find. This is pace the range of all possible values of θ is the search space, the more parameters we have, the larger the search space is, resulting in a more complex function. The changing of parameters happens during the training phase where we have a *training dataset*, while we validate the current parameters on a *test dataset*. If the error on both datasets are

similar then the learning algorithm *generalised* well to the test dataset, however it is often the case that the errors are not similar resulting in a few potential problems. The first of these problems is when the function $f(X, \theta)$ *underfits* to the training data, meaning there is a large error even through we provide the model with targets. The other problem that can happen is that the model *overfits* to the training dataset, resulting in a low error on this set but a high error on the test dataset. These problems are related to the *capacity* of the model as it is connected to the number of parameters. By having more parameters which in turn is more capacity results in a higher chance that the model overfits. However, if we have too few parameters the model might not be able to estimate the distribution, meaning that we get a function that underfits to the training dataset. Balancing between these problems is one of many challenges with learning algorithms.

The original development of neural network models was inspired by studies of information processing in the brain of humans and other mammals. This processing of information are electrically active cells called *neurons*, which are the basic processing units in brains. Neurons can be defined by a mathematical model called *neural network*, which forms the base for computational approaches to learning [93]. The idea behind a neuron is an all-or-nothing response, in which they are activated by enough electrical impulse. Many neurons connect to each other creating a complex network, and if a neuron activates then this can result in other neurons that are connected to activate creating a chain of activations. The extent to which one neuron causes another to activate depends on the strength of the electrical impulse, the changes in this strength is the key mechanism whereby the brain can store information and learn from experience. The perceptron is a set of neurons, where each take a vector of inputs $x \in \mathbb{R}^c$, they are then transformed by a fixed linear transformation $\phi(x) \in \mathbb{R}^d$, and bound to a set of trainable weights $w \in \mathbb{R}^d$. The weight vector w_0 is the bias term which is also added. The formula for a neuron is written as

$$z = y(x, w) = w^T \phi(x). \tag{2.2}$$

The result of neuron is passed through an activation function $f(\cdot)$, which was in the form of the following step function:

$$f(z) = \begin{cases} 0, & z \leqslant 0 \\ 1, & z > 0. \end{cases} \tag{2.3}$$

This model simulates neuron activating (often referred as firing), if the total weighted input exceeds a threshold of 0. The modern implementation and convention of the perceptron uses

$t_i = \{-1, +1\}$ instead of $\{0, 1\}$ as allow for faster convergence on zero-centered data (see section 2.2.3 for detail). The perceptron was introduced by Rosenbatt and is one of the most important models in neural computing [115]. Rosenbatt also developed a training algorithm which has the property that if there exists a set of weight values for which the perceptron can achieve a perfect classification of the training data then the algorithm is guaranteed to find the solution in a set number of steps [13]. The weights are trained on a target by a stochastic gradient descent with respect to a classification error, measured by the loss function. The loss function used by the perceptron is the sum of all z that are misclassified:

$$E(w) = - \sum_{i \in \mathscr{M}} w^T \phi(x) t_i, \qquad (2.4)$$

where $\mathscr{M}$ is the set of indices of misclassified samples. Using a stochastic gradient descent algorithm we update the weights at time τ from the weights and the error at $\tau - 1$ is:

$$w^{(\tau)} = w^{(\tau-1)} - \eta \nabla E(w) = w^{(\tau-1)} - \eta \phi(x) t_i, \qquad (2.5)$$

where η is a suitably chosen learning rate parameter and we initialises the weights to some starting vector. As the perceptron is a linear model it is limited to representing linear functions, resulting in not being able to distinguish non-linear patterns in data. The properties of perceptrons were analysed by Minsky and Papert, in which they gave formal proofs on the limited capabilities of perceptron algorithms (single-layer networks) as well as speculating on extending these networks to have multiple layers [95]. It would be latter shown that multiple layers speculation was incorrect but this lead to a dampen of enthusiam for neural network research during the 1970s and early 1980s. The main challenge researchers faced was the lack of an effective algorithm for training multiple layers as the perceptron algorithm were specific to single-layer models. The solution to training neural networks with more than one layer came from the use of differential calculus and apply it to gradient-based optimisation, known as the backprogagation algorithm. To distinguish between non-linear patterns, the step function was replaced with a differentiable activation function which have a non-zero gradient.

2.2.1 Feed-forward Neural Networks
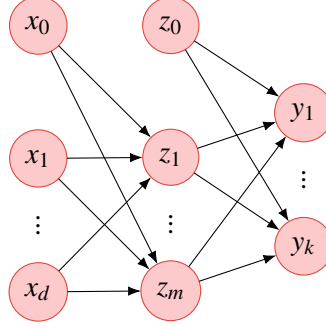
Figure 2.2: A feed-forward neural network with 2 hidden layers. The input, hidden, and output neurons are represented by nodes. The weight parameters are presented by links between nodes. The bias parameter is denoted z_0. Arrows show the direction of information flow through the network during forward propagation.

The *feed-forward neural network* is often described in the form that each layer has a linear transformation followed by an activation function. Each layer is often called a *fully connected* or *dense* layer, which is to say that the neurons in a layer is connected to every neuron in the next layer. We represent the first and last layers as *input* and *output* layers. Input layers contain the input features while the output layer contains the prediction of each label in a supervised task. Layers in between the input and output are termed *hidden layers*. The input features $x \in \mathbb{R}^c$ are fed into the first hidden layer of neurons, where the ith neuron has a matrix of weights $\Omega_{1,i} \in \mathbb{R}^{c \times d}$. The output of the ith neuron in this first hidden layer is:

$$z_i^{(1)} = \sigma(\Omega_{1,i}^T x). \tag{2.6}$$

The superscript of z represents the index of the first layer, σ is a non-linear function and $z^{(l)} \in \mathbb{R}^d$. We can then define each new layer denoted as index l in terms of the previous layer $l-1$:

$$z_i^{(l)} = \sigma(\Omega_{l,i}^T z^{(l-1)}). \tag{2.7}$$

The resulting output layer is denoted $\hat{y} = z^{(L)} \in \mathbb{R}^d$, where L is the final hidden layer before the output, and d is the number of features in the output. d is task dependent and can be bounded or normalised in many ways. For example, in supervised tasks, the error of the output $\hat{y}$ is measured against a target y with a loss function. The loss is used to adjust the parameters of the model, see section 2.2.3 for details.

We have shown that the output of a step function in a perceptron is determined by the input, which is modelled by the weight parameters. The step function is one of many activation functions, but we require one which is non-linear and differentiable. A neural network can have many hidden layers in which we could give a different choice for each part of the network, in practice we often use the same activation function. The simplest option is the identity function, which means that all hidden neurons are linear, and therefore any network with this type of activation function is equivalent to a network without hidden layers [13]. Therefore its representational capability is no greater than that of a single linear layer. If we define a network with a hidden layer that has a smaller amount of neurons than the input or the output, then the data transformation after this hidden layer is not the most general possible linear transformation due to information that is lost in the dimensionality reduction of that hidden layer. We refer to these types of networks as *bottlenecks* which correspond to a technique called *principal component analysis* and if we form non-linearly to this network we get a *Autoencoder*. One common non-linear differentiable activation function is the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + e^{-z}}. \tag{2.8}$$

This was used widely in the early years of research on multi-layer neural networks and is partly inspired by studies of properties of biological neurons [14]. Another common activation function is tanh:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}. \tag{2.9}$$

tanh differs from the sigmoid function by its linear transformation of its input and output values, therefore for any network with sigmoid activations in hidden neurons there is a equivalent network with tanh activation functions [13]. In practice however we often do not see equivalent results due to the gradient-based optimisation of tuning weights. In modern deep learning it is discouraged to use these two types of activation functions (expect in the final layer of a neural network) due to risk of saturation [49]. saturation is when the gradients of these activation functions go to zero when the inputs have either large positives or negative values. This has lead to a sub-group of activation functions which have non-zero gradients. The *softplus* is one such function:

$$h(z) = \log(1 + e^z). \tag{2.10}$$

Softplus is a non-zero gradient activation function because if we have a large z then $h(z) \simeq z$, and so the gradient remains non-zero [14]. Another common non-zero activation function and

one of the best performing is Rectified Linear Unit (ReLU):

$$h(z) = \max(0, z). \tag{2.11}$$

Its worth noting that the derivative of ReLU when $z = 0$ is not defined, however in practice this can be safely ignored because in floating-point arithmetic, the probability of getting an input that is exactly 0.0 is extremely low. Most inputs will be very small positive or negative numbers. If we do get an input of exactly 0.0 it has minimal impact on the overall training process because it affects such a small portion of the data. In situations where we do get this numerical instability, a machine epsilon (or another small constant) is used in practice. The introduction of ReLU gave improvements in training efficiency over other activation functions, allowing for deeper networks to be much less sensitive to random initialisation of the weights [73].

In supervised tasks, the network aims to distinguish between two classes, refereed to as *binary classification*. In practice we often label these as 0 and 1. The output layer would use a single output neuron with a sigmod (2.8), this will bound the output to the interval $(0, 1)$. The decision boundary is typically set at 0.5, where the outputs greater than 0.5 are classified as positive (class 1) or negative (class 0). A more common task for neural networks is predicting one of multiple classes given some input, that is to say that the target is not a single value z but can be many values. We first normalise the output with a function called *softmax*. Just like the binary case, this bounds the output to the interval $[0, 1)$. If the output vector is $z \in \mathbb{R}^c$, then the class output at a_i:

$$\hat{y}_i = \text{softmax}(a_i) = \frac{e^{a_i}}{\sum_{j=0}^{c} e^{a_j}}. \tag{2.12}$$

The sum of the outputs is $\sum_{i=0}^{c} y_i = 1$. Due to this normalisation it reflects a probabilistic interpretation which we call *class probabilities* for a multi-class output. Small differences in the input get amplified which can lead to more decisive outputs, this problem is known as exponential scaling. In practice this can lead to the softmax having numerical stability issues. A common approach to addressing this is to subtract the maximum value from each input before applying the function. This transformation does not alter the output but significantly enahances numerical stability by preventing overflow in the exponential calculations. By using the softmax function in the final layer as an activation function, each neural network layer learn to transform the input into a representation that can be separated by this softmax layer.

2.2.2 Universal Approximation Theorem

The universal approximation theorem states that any Borel measurable function from one finite-dimensional space to another finite-dimensional space can approximate any continuous function to an arbitrary degree of error. As any continuous function on a closed subset of $\mathbb{R}^n$ is Borel measurable, therefore a neural network can approximate any function [49]. There are two main design choices for building a standard neural network architecture, the first is the breadth of the network, which is the number of neurons in each layer. While the second is the depth of the network, which is the number of hidden layers [56]. In theory, we could design a neural network with one very large hidden layer and under the universal approximation theorem, it should be sufficient to learn and represent a function. However, in practice such a model will fail to learn and generalise correctly due to the optimisation technique, generalisation issues and efficient representation of the function [49]. As a result in practice we build deeper neural networks with more hidden layers and reduce the number of neurons in each. The success of building deeper neural networks is what lead to the current deep learning era and calling the neural network a *universal approximator* [13]. This is because a two-layer network (such as figure 2.2) with the first hidden layer using a non-linear activation function, can approximate a continuous function [13, 49]. Its worth noting that even through its possible to approximate any function given enough capacity, it does not mean that will learn the given function. This could be due generalisation issues such as overfitting, or could not give good approximation on the test dataset due to errors in the training dataset, or simply not enough data. The neural network has still become the tool of choice for approximation of functions over other statistical methods due to how the neural network encodes a belief that a function we want to learn is a composition of simple functions [49].

2.2.3 Gradient Descent

Neural networks are a board class of functions that in principle can approximate any desired function given a sufficient number of hidden layers but this also depends on the weights of the model. We choose these weights by optimising the loss function. One such way to adjust the model parameters θ is *maximum likelihood estimation*:

$$\theta^* = \underset{\theta}{\arg\max} J(\hat{Y}, Y), \tag{2.13}$$

where $J(-,-)$ is the loss function measuring the prediction $\hat{Y}$ with the target Y. Selecting a good function J is determined by data scientist and requires knowledge about the data. A

simple example is L_1 norm, which is the sum of absolute differences between the prediction $\hat{Y}$ and the target Y:

$$J(\hat{Y}, Y) = |\hat{Y} - Y|. \tag{2.14}$$

A challenge with L_1 loss is that there is a break in a function, which results in it not being continuous and smooth. This is refereed as discontinuity. In the case of L_1 the discontinuity is at the y-intersect, where the predictions exactly match the ground-truth. This is a problem for gradient-based optimisation methods as they might overshoot the optimal point due to a sudden change in the gradient. An alternative is the L_2 norm, which is the Euclidean distance between the predictions $\hat{y}$ and the ground-truth target y:

$$J(\hat{Y}, Y) = \sqrt{|\hat{Y} - Y|^2} = |\hat{Y} - Y|_2^2. \tag{2.15}$$

Although in principle the error functions(2.14 and 2.15) can be minimised numerically through a series of direct loss function evaluations, this is very inefficient. Instead we look into optimising the loss function by evaluating the derivatives of the loss with respect to the model weights, this is a form of using gradient information to find a optimal solution.

The goal in the training stage is to find values for the weights w in the neural network that will allow for a correct prediction. By adjusting w with respect to a loss function E we can make small steps in weight space from w to $w + \delta w$. This change in the loss function is given by:

$$\delta E \simeq \delta w^T \nabla E(w), \tag{2.16}$$

where $\nabla E(w)$ points in the direction of the greatest rate of increase of the loss function, as a result the smallest value will be at the point in weight space such that the gradient of the loss function is $\nabla E(w) = 0$. If this is not the case we can make small steps in the direction of $-\nabla E(w)$ to further reduce the error. Points in weight space where the gradient vanishes are called stationary points and are further sub-categorised as *minima*, *maxima* and *saddle points*. Even through our goal is to find a w such that $E(w)$ evaluates to the smallest value, many loss functions typically have a highly non-linear dependence on the weights, and so there are many stationary points in weight space where the gradient vanishes. The smallest value of the loss function over the whole weight space is called *global minimum*, while any other minima that is higher is called a *local minima*. In deep neural networks it is not uncommon to have many poor local minima, and while it was thought that gradient-based optimisation approaches might get trapped in one of these poor local minima, in practice this is not the case [14].

Using gradients to update weights gradually is called *gradient descent*. The standard algorithm is to compute the average error across the training dataset, then compute the gradients of the parameters with respect to the error, the magnitudes of which is then used to update the weights. Updates to the weights happen after one run-through of the train dataset, called a *epoch*. These succession of updates forms:

$$w^{(\tau)} = w^{(\tau-1)} + \Delta w^{(\tau-1)}, \tag{2.17}$$

where τ states the epoch step. The different types of gradient descent algorithms involve choices for updating the weight vector $\Delta w^{(\tau)}$. A better approach to using this gradient information is to update the weights based on a small step in the direction of the negative gradient:

$$w^{(\tau)} = w^{(\tau-1)} - \eta \Delta w^{(\tau-1)}, \tag{2.18}$$

where the parameter η is called the *learning rate*, which determines the step size at each iteration. Selecting a learning rate is dataset and task dependent, if we select a η that is too small, gradient descent will converge slowly, while if η is too large then we may overshoot the minimum, potentially leading to divergence or oscillation around the optimal point. Algorithms that use the whole training dataset at one step is refereed to as *batch* methods. While deep learning benefits greatly from large datasets these batch methods become inefficient with many data points, because the gradient update uses the whole data to be processed. The solution to this is a version of gradient descent called *stochastic gradient descent* where we update the weights based on one data point at a time [16]. Advantages of this approach over batch methods is that stochastic gradient descent can handle redundancy in the data much more efficiently and there is a possibility of escaping a poor local minima, since a point with respect to the loss function for the whole training dataset will usually not be a stationary point for each data point individually. In practice however we do not update our weights θ for each data-point, instead we build a small subset of the training set called a *mini-batch* and update on that. This is due to single data points providing noisy estimates of the gradient of loss function. This approach is called *mini-batch stochastic gradient descent*. In practice we also randomly shuffle of the data before creating batches as there might be correlations between successive data points. We also may shuffle the data between iterations as this can help escape a poor local minima.

Two situations which can create a challenge on complex deep learning functions is the sandle point within the weight space or if the gradient has a large increase to then decrease further into a better local minima. A sandle point is when the gradient vanishes $\nabla E(w) = 0$. A simple

technique for dealing with these problems is adding *momentum* term to the gradient descent formula which adds inertia to the motion through weight space:

$$w^{(\tau-1)} = -\eta \nabla E(w^{(\tau-1)}) + \mu \Delta w^{(\tau-2)}, \tag{2.19}$$

where μ is the momentum parameter. This creates a kind of velocity, allowing the optimisation to build up speed in directions with consistent gradients. If gradient is changing frequently, the momentum can help dampen the oscillations and stay on course towards the minimum.

2.2.4 Backpropagation through a Neural Network

In this section we discuss the process of discovering and evaluating the gradients of the loss function. This is done by using a local message-passing scheme which information is sent backwards through the neural network, known as *backpropagation* or *backprop*. In a standard feed-forward neural network we have a set of neurons where we compute the weighted sum of its inputs, these are then passed into an activation function to determine if the neuron should fire. This set of neurons is one layer of the network, we may have many layers in which the neurons would link to each of the neurons in the successive layer. This firing of neurons flows through the network until we get to the output which is defined by the task. This process of feeding in data and it flowing through the network is referred to as *forward propagation*. The evaluation of the derivative of our loss function E_n with represent to a weight w_{ji} where n is a index of a data point, j is the index of a neuron in the current layer while i is the index of a neuron in the previous layer, so w_{ji} is the weight of the connection from i-th neuron to the previous layer to the j-th neuron in the current layer. Note that E_n depends on the weight $w_j i$ by the input x_j to neuron j. We can then apply the chain rule for partial derivatives to give:

$$\frac{\delta E_n}{\delta w_{ji}} = \frac{\delta E_n}{\delta x_j} \frac{\delta x_j}{\delta w_{ji}}. \tag{2.20}$$

We can then generalise this equation to a set of hidden neurons using the chain rule:

$$\frac{\delta E_n}{\delta x_j} = \sum_k \frac{\delta E_n}{\delta x_k} \frac{\delta x_k}{\delta x_j}, \tag{2.21}$$

where the sum runs over all neurons k to which connects to neuron j. If we substitute part of the formula in 2.20 with the formula 2.21 to obtain the backpropagation formula:

$$\delta_j = h'(x_j) \sum_k w_{kj} \frac{\delta E_n}{\delta x_j}, \tag{2.22}$$

The backpropagation formula shows use that the value of δ for a neuron can be obtained by propagating the δ backwards from neurons high up in the network.

2.2.5 Regularization

In section 2.2 Neural Computation we discussed challenges of training a neural network can include over-fitting, slow convergence and gradient-based issues like having very large or very small gradients, which can cause instability of adjusting weights. Regularization is a technique and a set of methods designed to prevent these challenges by introducing additional information during the training process. This idea stems from classical statistical learning called the *bias-variance trade-off*, which helps in understanding the estimate of how well the neural network performs on the test dataset. In regards to neural networks, bias refers to the model's inability to capture true underlying function that describes the data. While the variance is the sensitivity to small fluctuations in the training dataset. Both bias and variance are built into the design of the network, as the depth increase we reduce bias by allowing the network to learn more complex functions, but can be harder to optimise. In statistical learning it is believed that this increase in depth also increases the variance, but as we will see later very deep neural networks often exhibit lower variance in practice. Regularization allows us to not need to limiting the number of weights according to the size of training dataset. It is considered unsatisfying in deep learning to limit the number of weights in this way because the complexity of the problem being solved does not necessarily correlate with the size of the training dataset, i.e. a small dataset might represent a complex underlying function or group of functions, while a large dataset could represent a simple one. It also ignores that modern deep learning architectures and training techniques generalise well and limits the potential for transfer learning.

There are many choices to make when building neural networks in deep learning, this is because most tasks are examples of *inverse problems*, which is to infer an entire distribution given only a finite number of data samples. This means that task is intrinsically ill-posed due to infinitely many distributions which could be responsible for generating the data, therefore having potentially infinitely many samples. Our goal in these tasks is to make a good prediction for an unseen value of x, therefore we need to choose a distribution from the infinitely many possibilities. Selecting a choice over others is called *inductive bias* (also called *prior knowledge*). This inductive bias usually comes from domain knowledge of the task, data or even a solution. In practice we want small changes in the input values to lead to small changes in the output values, therefore we should bias our solution towards smooth varying functions. This can be done by adding constraints or penalties to the learning process so that we can encourage the weights to target this type of function. We should be careful when incorporating these con-

straints that are inconsistent with the underplaying data generation process, as by making these assumptions we could produce inaccurate results. One way we can add these penalties is with the *L1 regularization* (also called *Lasso regularization*), where we add the absolute values of the weights to our loss function:

$$E(w) = E(w) + \lambda \sum_{i=0} \mid w_i \mid, \tag{2.23}$$

where λ is the strength of the regularization. In practice we tend to see that the weights drive exactly zero, leading to a sparse weight matrix, which is a form of feature selection as if the weight is 0 for a connecting input then the influence of that input is removed. A similar constraint is *L2 regularization* which is the sum of squared weights:

$$E(w) = E(w) + \frac{\lambda}{2} \sum_{i=0} w_i^2, \tag{2.24}$$

which encourages the network to use smaller weights by penalising larger weights resulting in smaller functions. Unlike L1, L2 rarely sets the weights to exactly zero but still towards zero, ensuring a more stable optimising process. By allowing L2 into the optimisation algorithm rather than an explicit term in the loss function we get a technique called *weight decay*.

$$w_i = w_i - \eta * (\frac{\delta E(w_i)}{\delta w_i + \lambda w_i}) = (1 - \eta \lambda)w_i - \eta * \frac{\delta E(w_i)}{\delta w_i}, \tag{2.25}$$

When L2 is used explicitly in the loss function or if it is used in the optimisation algorithm both are mathematically equivalent when the learning rate is constant. However, when using adaptive optimisation methods such as *Adam* the result is different in practice. This is because L2 regularization term is part of the loss and the gradients are computed with respect to that loss, while weight decay is applied directly to the weight update after gradient computation. This means that when we having changing learning rates within adaptive optimisation methods the gradient statistics used by Adam will be different. We should be careful not to use L2 in the loss when using adaptive optimisation methods like Adam because this results in optimisation not being scale invariant, meaning that if we do not use L2 in the loss and scale the weights up by a constant factor then the algorithm will behave in the same way just in that new scale. Weight decay does not have this affect on Adam.

Double descent is a phenomenon in deep learning that challenges the classical understanding of the bias-variance trade-off [98, 106]. The trade-off is often shown as a U-shaped curve where as the model capacity increases the bias decreases and the variance increases, therefore

a optimal capacity for a given task is at the point where the bias and the variance is minimised. In deep learning however it has been shown that initially the test error decreases as the model capacity increases, in many double descent papers this is called first descent, following what we know about statistical learning and the bias-variance trade-off. As we increase the model capacity further, we begin to see over-fitting, what the double descent phenomenon shows is even through the model becomes highly over parameterised, the test error will start to decrease again, refereed to as the second decent. Nakkiran *et al.* introduced *effective model complexity* which is the maximum size of training samples on which a model can achieve a loss of 0.0 on the training error, and so double descent is observed when the effective model complexity exceeds the number of data points in the training dataset. This behaviour can be seen on many different types of large deep learning models. As many of these large models will have many possible solutions and stochastic gradient descent has a implicit bias for the simplest solution, it is believed that any model trained on gradient descent will produce this phenomenon [98]. Another consequence of these findings is that when increasing the size of the training dataset can reduce performance, which is contrary to the conventional view of deep learning that more data is better [98]. This is due to showing the model more samples that represent the understanding function of the data generation process.

2.2.6 No Free Lunch Theorem

The no free lunch theorem was presented by Wolpert and it states that every learning algorithm is as good as any other when averaged over all possible problems[149]. For our domain of deep learning it is referring to the idea that if a particular model is better than average on some task, it must be worse than average on other tasks [14]. This theorem is considered to be theoretical in its notion due to the space of possible problems includes all types of tasks as it implies relationship between input and output, i.e a image classification model might perform bad on language generation task. In section 2.2.5 we discussed how it is useful for generalisation purposes to have a degree of smoothness in the training of a model so that a small change in the input has a small change in the output. Neural networks exhibit this in form of inductive bias, which is one reason as to why they have a broad applicability. The importance of the no free lunch theorem in regards to deep learning is that it shows how bias is used in determining the performance of an algorithm, as if we do not use bias then learning purely from data can be challenging, where some advocate it to be impossible [148]. In this school of thought many argue for making all assumptions in deep learning model be explicit so that the appropriate

choices can be made for inductive biases because stronger inductive biases that are specific to an application tend to perform better in practice [14]. In Chapter 5 we explore using this idea of explicit along with the standard implicit learning on graph neural networks, to being in domain more stronger both in the structure of the data as well as the structure of the network.

2.3 Convolutional Neural Network

The standard feed-forward neural network assume that the observed data values are unstructured, that is to say each element of a data $x = \{x_1, ..., x_d\}$ are treated as if we do not know about how the elements might be related to each other a head of training a model. However, many applications of deep learning involve some structured data where there is additional relationships to the input variables. For example the pixels of an image have a well-defined spatial relationship to each other in that the input is arraigned into a grid, and pixel neighbours are highly correlated. In such cases hand-crafted features where fundamental to conventional learning algorithms, where careful consideration of the task and the domain are needed before being feed into a neural network. However this way of modelling data ito build accurate image recognition systems by hand are impossible [75]. In this section we introduce the CNN which directly works on raw input information and avoids the conventional approach to being hand-crafted features [78].

2.3.1 Convolution

We motivate the idea of the CNN and the convolution kernel based on image data. If we design a standard neural network with fully connected layers then we would require a large number of parameters due to the high-dimensional nature. For example, if we have a colour image of $10^3 \times 10^3$ pixels, each with values for red, green and blue intensities. If we have 1000 neurons then the number of weights would be 3×10^9. The network would also need to learn the invariances and equivalences of the data, this would require a very large dataset. If we can design a network that incorporates a inductive bias about the structure then we can reduce the amount of data and also improve generalisation with respect to symmetries in the image space [49, 14]. To create this design we introduce four interrelated concepts *hierarchy*, *locality*, *equivalence*, and *invariance*. There is a natural hierarchical structure to all images, in fact there is usually more than more natural hierarchical structure [127]. Consider the task of detecting faces in images, one sample image may contain several faces, and each face includes

0.4	1.7	0.9
2.3	−2.1	4.0
−1.4	0.7	2.1

A B
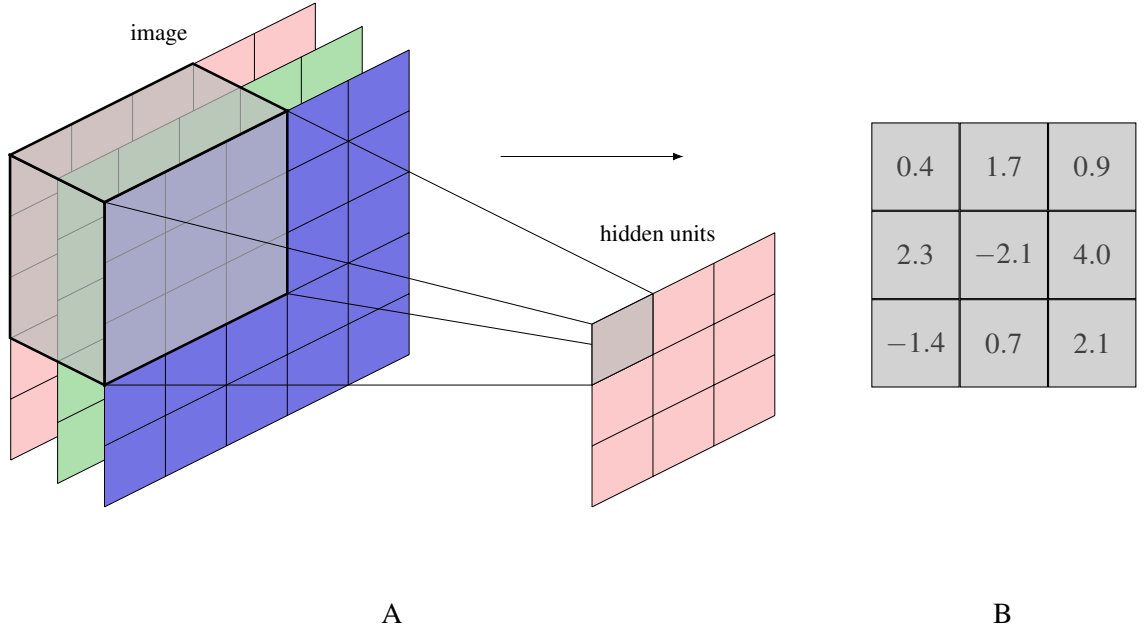
Figure 2.3: A: a multi-dimensional filter that takes input from the red, green, and blue channels of an image. The layer receives input from pixels in a $3 \times 3 \times 3$ patch, the pixels in this patch are known as the receptive field. B: The weight values associated with the hidden unit, known as a kernel.

sub-objects like eyes, each eye has a iris, which itself has a structure of edges. This forms a hierarchy of objects where at the lowest level, a node in a neural network can detect this edge using information that is *local* to the overall image. As a result this neuron would only need to see a small subset of pixels to accurately detect it. As we move up the hierarchy, more complex structures can be detected by composing multiple features from previous levels. This approach of building hierarchical models fit naturally into the deep learning framework as it allows for complex concepts to be extracted from raw data through the succession of layers, in a system which is trained end-to-end.

CNNs are designed work on regular domains, which is to process data on a grid. The core of this design is the *convolution*. The input to a single unit is a set of pixel values from a small rectangular patch, which is referred as the *receptive field* of that unit and captures some local information based on the associated weights. The output is comprising a weighted linear combination of the input, which then gets transformed using a non-linear activation function:

$$z = \text{ReLU}(w^T x + w_0), \tag{2.26}$$

where x is the vector of pixels. As there is one weight associated with each input pixel, the

weights form a grid known as a *filter* (or *kernel*). This convolutional filter is shown in figure 2.3, where our input is a RGB image and we use a multi-dimensional filter of $3 \times 3 \times 3$ over that image to produce an output *z*. The largest output response for this filter will be when it detects a patch of the image that looks like the kernel. Most image datasets have a hierarchical structure at different locations, often with the same pixel values. If we use the face dataset example, a patch represents an eye at a location in the image, then we will have the same set of pixel values in a different part of the image as that represents the second eye. The neural network needs to be able to generalise what it has learnt in one location and be able to apply it to other locations of the image. It should be able to do this without having examples in the training dataset for every possible location. This is done by replicating the same hidden-unit weights at different locations across the image. These different units of the layer form a feature map, in that all the units share the same weight. Locations that have a large output response in one location then the same set of pixel values in a different location will produce the same output, this will show in the feature map, this is referred to as *translation equivariance*. As we will only get a large output response based on the kernel then the connections in the network will mostly be sparse and the neurons in the layer will have a set of *shared* weights. As the kernel slides over the image using these shared weights, this forms a *convolution*. We show the convolution in the form of a 2D image *I* with pixel intensities $I[j,k]$, and the filter *K* with pixel values $K[l,m]$ and the output feature map *O*:

$$O[j,k] = \sum_l \sum_m I[j+l,k+m]K[l,m], \tag{2.27}$$

where we usually express the convolution as $O = I * K$. As shown in figure 2.4 the output is smaller than the original input, if our image has a shape of $J \times K$ pixels and we convolve with a kernel of shape $M \times M$ then the shape of the output will be $(J - M + 1) \times (K - M + 1)$. In practice this is referred to as a *valid convolution* as we only apply the kernel to the complete overlaps with the input. Depending on the task and the network architecture this can be issue as information at the edges of the input is used less in the output. In these cases we want to have an output that has the same shape as the original image, this can be done by *padding* the input with additional pixels around the output. By padding with *P* pixels the shape of the output is $(J + 2P - M + 1) \times (K + 2P - M + 1)$. To get the convolution to produce an output that is the same shape as the input *P* must equal $\frac{(M-1)}{2}$, which is called *same convolution*. Of course, we can select any *P* giving us control over the output and allowing use to handle input sizes that are varying in sizes within the dataset. In computer vision tasks we usually select a odd value for kernel shape *M*, this allows the padding to be symmetric on all size of the

input and that there is a well-defined central pixel at the location of the kernel. There are a few different choices for what the value of the padded pixel should be, such as *zero padding*, this is a typical choice for CNNs as we do not introduce new non-zero values that can be mistaken for features. However zero padding can create artificial edges at the borders of the image and is not suitable for data where zero has a specific meaning. *Reflection padding* can help reduce boarder artefacts by reflecting the input pixel values at the edges, this means that the edges of the image are smooth unlike other padding techniques. This is useful as there are no new values added to the image and the gradients can flow smoothly from the padded region back to the original input, which could improve learning at those boundaries. Padding is also applied to feature maps for processing by subsequent layers. In many situations the images can be very

$$I \qquad\qquad K \qquad\qquad O$$
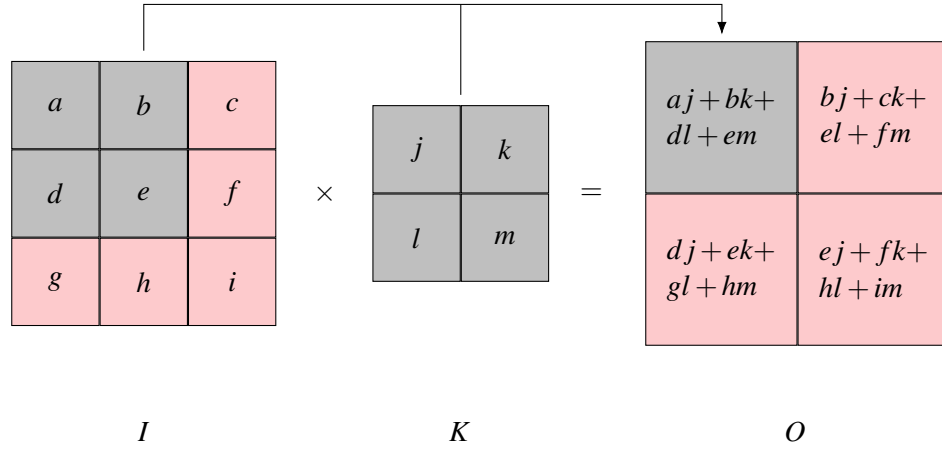
Figure 2.4: A 3×3 image (labelled *I*) convolved with a 2×2 filter (labelled *K*) which results in a 2×2 feature map (labelled *O*). This operation is called valid convolution as we restrict it to the boundaries of the image, i.e. only valid pixels are used. We could also pad the image before performing the convolution so we can keep the image dimensions, this is known as same convolution.

large and we often use small kernels, we may want to build a network where the feature maps are significantly smaller than the input image that the convolutions we have explored so far cannot do. We can achieve this by using *strided convolutions*, therefore, instead of stepping the kernel over each pixxel in the image at a time, we can take larger steps of size *S*. These skips over the image are called *stride* and is applied both horizontally and vertically. This results in the shape of the feature map being:

$$\left\lfloor \frac{J + 2P - M}{S} - 1 \right\rfloor \times \left\lfloor \frac{K + 2P - M}{S} - 1 \right\rfloor, \qquad (2.28)$$

where $\lfloor x \rfloor$ denotes the floor function.

Image   Kernel   New image
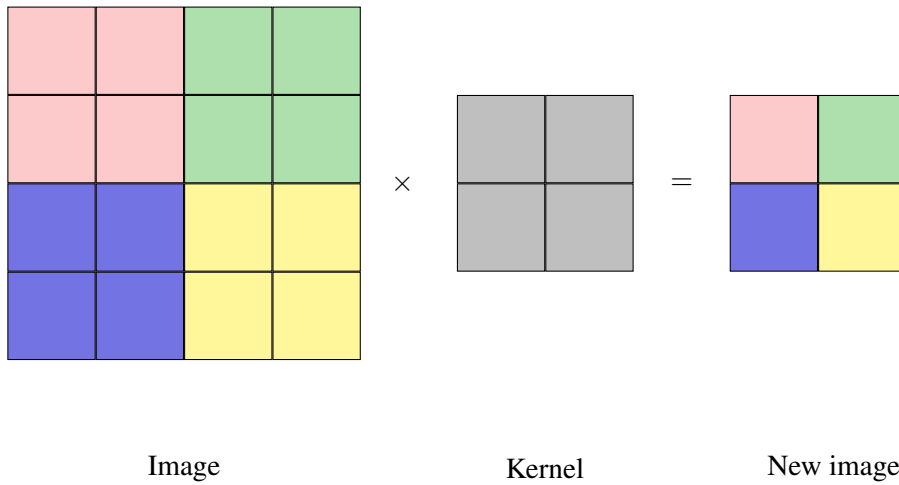
Figure 2.5: A convolution with a stride of 2 will halve the dimensions of the output with regards to the input. We show colour-coded convolved locations, where each colour of four pixels is multiplied with the kernel to get the new image.

For a kernel that has a shape of $M \times M \times C$ where C is the number of channels in the input, we will have M^2C weights in which they are shared to all points in a feature map. There is also a bias parameter associated this operation, this means that the convolution is analogous to a single hidden neuron in a fully connected layer of a neural network. Like a single hidden neuron, this convolution unit can only learn one kind of feature. Therefore we can use multiple kernels where each has its own independent set of weights and results in its own feature map.

We have shown that a convolutional layer encodes equivariance by using the receptive field to move to different locations in the image, and the outputs of the feature map moves to that location, useful in domains like object detection where we want to find an object in an image but for domains like image classification we want the output to be invariant to translations between groups of images that have similar features in different locations. We will now show that the CNN can learn hierarchical structure of images from higher up complex features that are built from simpler features at lower levels of the hierarchy, along with having this invariant to small translations of features with an operation called *pooling*. A pooling layer is very similar to that of a convolutional layer as our input is still arranged in a grid, we have a receptive field and the choice of a kernel size along with a stride parameter. The difference is that in pooling there is no learnable parameters where its purpose is to reduce the spatial shape of the feature map and add some local translation invariance, this is why it is often referred to as *down-*

sampling. Note that if we have a convolutional layer with a stride that is greater than 1, then we will also get a down-sampling affect on the feature map. This first type of pooling we will introduce is *max-pooling* by Zhou and Chellappa [171]. The operation of this type pooling works by simply outputting the max value within the receptive field. Another type of pooling is called *average pooling* which computes the average of the values in the receptive field. Both of these operations introduce some degree of local translation invariance as well as allowing the network to capture features at different scales.

2.3.2 Network Architectures

Since the introduction of LeNet-5 by Yann LeCun *et al.* in 1998 there has been many evolutions to network architecture, both for varying tasks and domains but also in performance due to how information flows through a network [75, 20]. In this section we discuss landscape of CNN-based network architecture that have emerged over the years in the literature. Many have achieved state-of-the-art, others have shown how we can capture information more efficiently. In general these architectures have become fundamental in deep learning research and applications.

AlexNet. The first architecture that we will discuss the AlexNet convolutional network architecture by Krzhevsky, Sutskever, and Hinton in 2012 [73]. This model won the 2012 ImageNet competition with a top-5 error of 15.3%, it was the first deep network to win with only 8 layers. The network introduced many of the concepts that we consider fundamental today, such as use of the ReLU, dropout, data augmentation and GPUs to train the model in parallel. This architecture also introduced blocks of layers that usually repeat throughout the feature extraction section of the network. In this case it was a convolutional layer, followed by a max-padding layer, deeper in the network this changed to having 3 convolutional layers. Despite employing regularization techniques like dropout and data augmentation, AlexNet still faced over-fitting issues, particularly with smaller image sizes. This was partly due to its large capacity of 60 million parameters.

VGGNet. The Visual Geometry Group (VGG) model, a common backbone architecture, comes in two main variants based on the number of layers: VGG-16 and VGG-19 [122]. VGG uses a simple design of convolutional blocks that create a uniform architecture. Like other backbone architectures, it reduces the number of hyper-parameters, simplifying the net-

work design process. The standard version takes in a $224 \times 224 \times 3$ input, followed by a set of convolutional blocks with down-sampling achieved through max-pooling layers. Each convolutional layer is of type 'same', has a kernel of shape 3×3, a stride of 1, and ReLU activation. One of the key innvolutions was the use of 3×3 kernels over 5×5. Three stacked 3×3 convolutions have the same effective receptive field as one 5×5, thereby reducing the number of weights. This led to quicker convergence and less risk of over-fitting, despite the increased number of layers compared to other architectures at the time. The authors demonstrated that depth in the network architecture often leads to higher performance compared to networks that focused more on width. VGG-16, with its 138 million weights, achieved state-of-the-art results on ImageNet, demonstrating that highly hierarchical feature representations can be learnt. However, the amount of weights meant for a computationally intensive task to train. With in the field this network shifted the focus from complex, heterogeneous architectures to ones with repeatable patterns, which influenced subsequent architectures to prioritise simplicity and modularity such as with ResNet.

ResNet. Rich forms of feature representations can be learnt very well on deep networks, this stems from the use of multiple layers of processing. Many architectures demonstrated that increasing the number of layers in a network can increase the generalisation performance. However, it becomes increasingly more difficult to train networks with a large number of layers. Very deep networks that have an extremely small change in the weights in the early layers can produce significant change in the gradient, this is referred to as *shattered gradients* [7]. A modification that is now used in many architectures to help in training deep networks is that of *residual connections*, which is a type of *skip-layer connection* [57]. A residual connection adds the input back onto the output of a layer:

$$
\begin{aligned}
z_1 &= F_1(x) + x \\
z_2 &= F_2(z_1) + z_1 \\
y &= F_3(z_2) + z_2
\end{aligned}
\tag{2.29}
$$

where $F_l(\cdot)$ could be one or more layers of a neural network. The combination of a layer and a residual connection is called a *residual block* and a *residual network* or *ResNet* consists of multiple layers of these blocks in a sequence. This allowed for a smaller weight space as we learn the difference between the identity and the descried output [82].

2.3.3 Computer Vision Applications of CNNs

Computer vision is defined as the automatic analysis and interpretation of image data and has become on of the largest areas of research in deep learning and machine learning [127]. Within this field, CNNs have become the most common approach for many tasks. Before deep learning computer vision was largely based on building hand-credited features to be used as input into different learning algorithms [55]. Computer vision was one of the first areas to be transformed by neural networks and brought into the domain of deep learning, most notably this was due to a CNN called LeNet-5 [75]. Computer vision can roughly be broken up into the following application areas (note that we are only discussing the areas that are relevant to this thesis):

- **Classification**: Given an image can it be classified from a set of labels. This is often referred to as image recognition.

- **Object Detection**: The task is to locate and classify each object. This often results in a ROI around every object as well as what is inside the ROI.

- **Segmentation**: The task involves individually classifying pixels, which divides the image into regions sharing a common label.

Classification was the first task to show how the CNN architecture could be successfully deployed to computer vision. One early example was the LeNet which would classify images of handwritten digits called MNIST [77, 75]. The research accelerated with the introduction of a large-scale benchmark dataset called ImageNet, which consists of 14 million images each of which belong to 22,000 categories with a hierarchical labelling scheme called word-net [34, 40]. This was one of the largest datasets of its time and showed the importance of using large-amount of data in deep learning. A subset of ImageNet with 1000 categories formed the competition called *ImageNet Large Scale Visual Recognition Challenge* which brought many of the architectures shown in section 2.3.2. This many categories made the problem very hard because if the classes were distributed uniformly, a random guess would have an error of 99.9% [14]. The dataset has around 1.28 million training images, 50000 validation images and 100000 test images. The metrics of this challenge was to produce a ranked list of predicted output classes on test images, which is reported in terms of top-1 and top-5 error rates, which means that if true class appears at the top of the list or if if it is in the top 5 then the image is cor-

rectly classified. Since this challenge classification accuracy has continued to improve, driven by deeper networks and better training algorithms [127]. For this task the community has moved from recognising categories to the problem of *fine-grained* category recognition, where differences between sub-categories are more sutible, often including grainlier labels with a hierarchical relationship. As a result of having such detailed labels the number of samples are quite low. Datasets that have these fine-grain labels include flowers [100], cats and dogs [104], and cars [160]. The overall design to building a deep learning solution to fine-grained classification to use meta-data of the images such as sub species, colour of animal or the action the object is performing in the image like the INaturalist system [133]. Extracting these attributes from images can enable a model to predict these attributes given it has previously not seen a category before, referred as *zero-shot learning*. Use of this meta-data in images for tagging with computer vision algorithms has made it much easy find on the web, both from a dataset building and a classification perspective. This has lead to application areas such as *visual similarity*, where our aim is to find a group of images that a model predicts as being similar to its input. Approaches usually use whole image descriptors such as Fisher kernels and Vector of Locally Aggregated Descriptors (VLAD) [3], or pooled CNN activations [129] to represent images used to measure similarity in large image-based databases [67].

In the task of object detection, there is two different tasks becoming completed, the first is locating all objects in an image, and the second is classifying each object. The latter has the aim of putting a ROI of that object which has challenges such as ensuring that the box is as tight as possible and restricting the learning structured features even through most objects in images have organic irregular structure. In the former you are replying on the position and shape of the ROI to help in determining what the object is, this leads to challenges such as other objects being in the ROI as well as the object not being fully in the ROI. In general this task breaks down into two main research groups: *two-stage* and *single-stage* models.

One of the first two-stage object detection methods to use neural networks was the Region-based Convolutional Network (R-CNN) [48], which selects 2000 region proposal using an algorithm called selective search [131]. These proposes would be recalled to a 224 square image so that they could be fed into AlexNet which would learn the features of the dataset. The embeddings would then be fed into a Support Vector Machine (SVM) as the final classifier. The follow up on this approach was Fast R-CNN which replaces the SVM with fully connected lay-

ers that can compute the prediction and box refinement of the ROI [47]. This model was able to do this due to having a shared backbone architecture with two separate *heads*. Each output head would have a different loss function for the given task. The computation of the CNN lead to much faster training and inference time with much better accuracy. The bottleneck of this approach was the selective search algorithm and so this was replaced with a convolutional-based Region Proposal Network (RPN) to make Faster Region-based Convolutional Neural Network (Faster R-CNN) [111]. An input image first goes through a series of convolutional layers, these are pre-trained using one many backbone architectures, the feature maps are then feed into the RPN. The model then slides over the feature maps and considers different predefined boxes of scales and aspect ratios called *anchor boxes*. For each anchor box the RPN predicts if there is an object or not, and adjusts the box coordinates for a tight fit around the object. The family of R-CNN models up to this point in time operate on a single resolution convolutional feature map, making detection challenging on objects of different sizes. A solution to this is to use the different levels of feature maps as the data flows through the model, this represents different resolutions of the image due to pooling and convolutions. Each of these layers have a top-down connection to merge semantic information from the early layers with spatial information from the later layers [86]. This approach is called a *Feature Pyramid Network* and it was inspired by the human visual system as feedback connections throughout the visual cortex influence lower-level visual perception from higher-level cognitive processes [45].

In two-stage object detection methods we first use a region proposal approach to select locations to be considered and then use a second approach to then classify them, an alternative to this are single-stage object detection methods which uses a single neural network to output ROI along with the classification. One such family of detection methods are called You Only Look Once (YOLO) [109, 110, 108]. The idea of these approaches is to divide the input image into a grad, each cell is treated like a regression problem where the model predict the shifting of the centre coordinates as well as the height and width of the cell. The result is many cells that are overlapping an object, we then use Intersection Over Unions (IOU) or Non-Max Suppression (NMS) so that we keep only the most relevant boxes. Another is RetinaNet which is built on a feature pyramid network discussed above, it uses a *focal loss* to focus the training on harder samples in the dataset by down-weighting the loss on well-classified ones such as background sections of the image [87]. This is mainly due to unbalancing of classes in the dataset and therefore stops the more common classes from overwhelming the training. The focal loss also

has many conceptual similarities to active learning, see section 2.6 for details.

The task of segmentation can roughly be broken up into *semantic segmentation*, *instance segmentation* and *panoptic segmentation*, with the main goal of all of these approaches is to label each pixel in an image with a class. Just like the other areas of computer vision the CNN helped in enabling per-pixel semantic labelling with a single approach [90]. The challenge with these early models is that accuracy around the boundaries of each segment is often poor. In section 2.3 we had seen that a CNN can use a number of levels of down-sampling so that the number of channels increase and the size of the feature maps decrease, allowing the network to extract semantically meaningful high-order features from the image [14, 49]. We can use this idea to create semantic augmentations where we create a bottleneck in the middle of the architecture so that we can learn low-dimensional internal representations before we transform it back up to the original image resolution, known as deconvolutional upsamling [101]. With this type of layer we can then build other architectures such as the Autoencder and UNet [114]. Most modern semantic segmentation approaches are built from the feature pyramid network, where the top-down connections can help with percolate semantic information down to higher-resolution feature maps. Models like the pyramid scene parsing network with spatial pyramid pooling as a way to aggregate features at different levels of resolution [168, 58], the unified perceptual parsing network [153], and the HRNet are based on this type of network [141].

2.4 Deep Networks

In previous sections we have motivated the development of neural networks and CNNs, followed by discussing different network architectures both from a state-of-the-art and efficient use of capturing information prospective. As a result deep networks have introduced a few concepts that are of particular use for computer vision. The first is how deep networks encode a particular form of inductive bias such that the outputs are related to the input through a hierarchical representation. Within computer vision the relationship between pixels of an image and a high-level concept object (such as a car, cat or dog) is complex and very non-linear. Deeper networks can detect low-level features like edges in the early layers, to then combine them in later layers to make high-level features (such as car wheel, cat eyes or dog fur). This in turn leads to the output layers which detect the full object. At each layer we are combining different components which gives us gains in the number of possibilities as we increase the depth of the network. Deep networks also take advantage of *distributed representation*, which

is to say that each neuron is thought to represent a single feature at that hierarchical level of the network [14]. However, a deep network learn different representations in which combinations of neurons represent a single feature later in the layers of the network. We view these later layers as performing transformations of the data into new spaces, so that it is easier to solve for a task. This technique to discover non-linear transformations of the data is called *representation learning* [12]. The space at which the input data is transformed into is called *embedding space* and is given by the output of a hidden layer of the network via forward-propagation. This area of deep learning has become quite popular in the literature due to how we can exploit unlabelled data. Learning from such data is called *unsupervised learning*. In this thesis we use a modified version of is type of learning called *semi-supervised learning*, where some of the data is labelled so that we can utilise the embeddings to find matching embeddings for the associated unlabelled data (see chapters 3 and 4 for examples of this type of learning). *transfer learning* is a techqiue that allows us to take advantage of some of these learnt embeddings. Conceptually, it is the process of using the internal representation learnt from one task applied to a different but related task. A typical scenario is to train a backbone model on a very large dataset of common objects, we then take the classification head of the network and retrain it on a smaller dataset for a more specific task. This often allows for higher accuracy to be achieved then if we just used the smaller dataset, as we can exploit common features that are shared by both datasets. We must ensure that the inputs are a similar type and there should be some commonality between tasks, otherwise the low-level features learnt on the task with the large dataset will not be useful the task with the smaller dataset. Transfer learning has become commonplace within computer vision due to how the CNN will see that most image processing tasks require similar low-level features in the early layers of the network, while the later layers are focused on the task.

The rest of the following subsections discuss important developments in deep learning from the prospective of computer vision.

2.4.1 Attention-based Methods

The core concept that underpins the transformer is the *attention* mechanism which was originally designed to enhance Recurrent Neural Network (RNN) for the task of machine translation in 2014 [6]. It was not until 2017 when Vaswani *et al.* showed that removing the recurrence structure and instead focus on the attention mechanism we could get far better performance

[134]. The input to a transformer is a set of vectors with some dimensionality, we refer to these vectors as tokens which comes from domain of neural language processing but for our purpose these could be patches of an image or learnt image embeddings from a CNN. The elements of each of these vectors are the features, which makes for a useful property of the transformer, in that we do not have to design a new architecture to handle a mix of different data types, we can instead combine the data to make a token. At each step the input is split with a *query key* and a *value* matrices. Each have there own independent linear transformation:

$$Q = XW^q$$
$$K = XW^k \tag{2.30}$$
$$V = XW^v$$

where each of the learnable weight matrices W^q, $W^k \in \mathbb{R}^{D \times D_k}$ and $W^v \in \mathbb{R}^{D \times D_v}$. D_v, D_k is the shape of the hidden layer output. If we use the same shape then we can stack many of these transformation layers. *Scaled dot-product attention* is an operation that multiples the queries and the keys with a scaling factor $\frac{1}{\sqrt{D_k}}$, to then multiply this result by the value matrix:

$$Y = \text{attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V. \tag{2.31}$$

The above attention layer allows the outputs to attend to data dependent patterns of the input, called an *attention head*. There are many cases where multiple patterns of attention are useful information at the same time. For these cases we may want to use multiple attention heads in parallel, where each have independent weights:

$$H_h = \text{attention}(Q_h, K_h, V_h) \tag{2.32}$$

where we have H heads indexed by $h = 1, ..., H$. We also have separate queries, keys and value matrices for each head of h. The output is given by first concatenating the heads into a single matrix then feeding it into dense layer:

$$Y(X) = \text{concat}(H_1, ..., H_H)W^o, \tag{2.33}$$

where W^o is the learnable weight. The multi-head attention layer is the core element of the transformer network. We would like to stack many of these self-attention layers on top of each other but to improve training a residual connection is often used to bypass the multi-head structure. *Layer* normalisation is also used [5]. The result is written as:

$$Z = \text{LayerNorm}(Y(X) + X), \tag{2.34}$$

where Y is defined by 2.33. As there is no recursion in what we have discussed so far there is no sense of order in the input. This is because the transformer architecture matrices W_h^q, W_h^k and W_h^v are shared across the tokens like a feed-forward neural network. This has a benefit in that we can massively parallel the processing of the transformer during training, and it allows the network to learn both long-range and short-range dependencies. In domains there we need to consider sequential data like in natural language this can be a major limitation. To fix this limitation Vaswani *et al.* introduce the *positional encoding* [134]. We construct a position encoding vector r_n that is associated with each input at position n. We then add the positron to the input $\tilde{x}_n = x_n + r_n$. This can be seen as corrupting the input, however in practice it is not. This is because two uncorrelated vectors tend to be almost orthogonal in spaces of high dimensionality, which indicates that a model can process input and position information with some separately [49]. The positional encoding that Vaswani *et al.* introduced is given by position n:

$$r_{ni} = \begin{cases} \sin\left(\frac{n}{L^{i/D}}\right), & \text{if } i \text{ is even,} \\ \cos\frac{n}{L^{(i-1)/D}}, & \text{if } i \text{ is odd.} \end{cases} \tag{2.35}$$

where i is a dimension of the embedding space, D is the total number of dimensions and L is a hyper-parameter which acts as a scaling factor to control how quickly the frequencies of the sine and cosine function decrease as the dimension i increases, the authors suggest a value of 10000. This formula alternates between sine and cosine functions for even and odd dimensions creating a unique pattern for each position. The transformer layer has shown to be a very flexible in many applications such as *large language models* with one being ChatGPT [169].

2.4.2 Autoencoder-based Methods
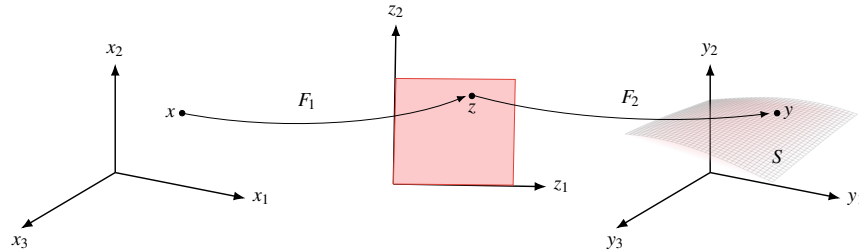
Figure 2.6: An autoencoder is built from 2 parts: encoder F_1 and a decoder F_2. Using F_1 we map our 3 dimension data to an internal representation of shape 2. F_2 takes this representation and maps it back to the original shape of the input, where our goal is to generate the output to be as similar as possible to the input.

A common goal in deep learning is to learn internal representations of data so that it can be used for other applications, one such way to do this is with a *auto-associative* neural network, more commonly refereed to as a *autoencoder*. In such a model the shape of the output is the same as the input and we train to generate a output to be as close to the input as possible. The model as a internal middle layer which is the lowest form of the representation (i.e. layer with fewest weights and smallest output) of the input. This results in a model with two sections, the first is called an *encoder* that maps the input to this internal layer, and the second is the *decoder* which maps the result of the internal layer to the output. Figure 2.6 demonstrates a geometrical interpretation of this mapping, where we have an input with 3 dimensions that is mapped to a layer with 2 neurons shown as F_1. F_2 takes the embedding from the internal layer and maps it to some output space with the same dimensions as the input. If we have a feed-forward neural network with one hidden layer with M neurons, an input and output of D neurons, we introduce a constraint to M so that it must be less than D. Otherwise the network will just copy the input values to the output. The targets y are the inputs x, so a autoencoder learns to map each input onto itself, this kind of mapping is called *auto-associative* mapping. As the internal layer has a constraint, a perfect reconstruction of all data is not possible, so we have to find a set of weights that minimises a loss function capable of capturing the degree of mismatch between input and reconstructions. A common choice is the sum-of-squares loss function:

$$E(w) = \frac{1}{2} \sum_{n=1}^{N} \| y(x_n, w) - x_n \|^2 \tag{2.36}$$

Bourlard and Kamp showed that if the hidden neurons have linear or non-linear activations then the loss function has a unique global minimum and that at this minimum a network will project the data onto the M-dimensioanl space that is spanned by the first M principal components [17]. Therefore no advantage is gained by using a two-layer neural network on such a task over Principal Component Analysis (PCA), as PCA will give the same solution in quicker time. However, if we build a deeper network with more non-linear activations then we can learn complex non-linear relationships in the data by utilising hierarchical representation property of neural networks. We have shown how we can use convolutional layers as a learnable down-sample operation on the input in section 2.3. In the encoder we use these convolutional layers just like a standard CNN. However, to get the internal embedding, a flatten layer is used, which simply collapses the feature map into a vector. This then gets feed into a dense layer to produce the internal embedding. The decoder then needs to generate the input from this embedding, we could use a up-pooling or a resizing method but it is preferred to use a learnable operation to

build the input. This can be performed with a *transposed convolution*. In a standard convolution we slide the receptive field over a image performing the dot product of the current patch with a weight matrix, resulting in a smaller output. In transposed convolutions does this same operation in reverse, other in practice we implement by interleaving the input with zeros and apply the standard convolution, where stride and padding manipulate the output size.

A common problem with autoencoders is that they tend to learn the identity mapping. To avoid this and force the model to learn the internal structure of the data, a *denoising autoencoder* was proposed [137]. The approach takes the input vector and corrupts it by adding noise. We then train the model to reconstruct the noise-free version of the input given the one with noise. There are many different options for the type of noise to use. A common approach is to randomly select elements of the input to be zero. This approach learns aspects of the structure in the data. If we applied this to images, pixel neighbours will have a correlated value to the corrupted pixel, which is corrected by minimising the sum-of squares between the corrupted image and the non-corrupted version.

2.4.3 Limitations on Irregular Domains

CNNs have shown to be very effective with data that forms to some regular structure. In images we index based on a pixel location, this forms a gird structure allowing a kernel to slide over the image, computing convolutions. We use the kernel as there is a regular relationship between centre pixel and its neighbours. Another view to this is that the relationship between a pixel and its neighbours are a constant distance apart. There are many domains that do not form into this regular structure and if we do apply a CNN to such data, this would result in forcing irregularly structured samples into some regular space. We then lose important information that is embedded in the irregular structure, meaning the model would have difficulty learning this irregular structure that is important for the task [121]. Examples of domains where no regular relationship exists between data include the positional reading from points on the human body [37], using the molecule structure to predict different properties of that structure [70], road networks [64], hierarchical labelling systems [40, 170]. Motivation into using irregular data has resulted in a new field of deep learning called *Graph Deep Learning*. The following section gives discussion on this field.

2.5 Graph Deep Learning

In all sections up to this point we have discussed structured data, mainly in the form of images. We now explore the use of graph data as a way to describe complex relations between samples, and later look into modelling these samples by introducing the *graph neural network*, where the grid structure is relaxed, so that samples are not constrained to have a relationship to other samples. This can be of use in many domains, for example a molecule has different types of interconnecting atoms. Each atom can have a label such as carbon, trogen or hydrogen and then the connection to that atom can also have different labels, like single bond or double bond. In world wide web we can connect to different web pages, in this case the connection here has a direction as a page *A* having a hyperlink to page *B* does not necessarily mean that *B* also has a hyperlink back to *A*. All these examples can be represented as a graph, in fact a image is simply a special instance of a graph [38]. In this section we describe graph deep learning, first by introducing the graph and some common properties and notations, this then follows how we can capture feature representations of the graph and the different types of tasks involved. We end the section discussing common network architectures and some challenges in graph deep learning.

2.5.1 Graph Properties

A graph *G* is a ordered pair of sets (V, E), where *V* is a set of *nodes* and *E* is a set of *edges*. We index into the nodes like any vector $n = 1, ...N$, and write the edge from node *n* to *m* as (n, m). If two nodes are connected by an edge then they are called *neighbours* and the set of all neighbours for a node *n* is $\mathcal{N}(n)$. A sub-graph of *G* is a graph $H = \{W, F\}$ where $W \subset V$ and $F \subset E$. A graph can also have *weighted* edges when the edge maps to a set of real values, denoted $w_{n,m}$, a weight from node *n* to *m*. Each node also has some associated data to it which is represented as a *D*-dimensional vector x_n for node *n*.

Both in practice and as a way of visualising a graph we use a *adjacency matrix* denoted as *A* to specify the edges. We must follow a chosen order for *V* such as the index order as stated above. The shape of the *A* is therefore $N \times N$ and if the cell contains a 1 at index location n, m then there is an edge connecting *n* to *m*. All other cells are set to 0. For a graph that has underacted edges *A* will be symmetric since it implies that $A_{n,m} = A_{m,n}$. We often use *A* in some form as input to a neural network, however as the nodes are ordered a neural network will simply learn

this permutation. We could implement a permutation in-variance into the training scheme, but this is infeasible as the number of permutations increase factorially with the number of nodes. Instead we build a network architecture with this in-variance property. First we introduce a *permutation matrix P*, which has the same shape as *A*. *P* defines a particular permutation of node ordering. A row in *P* only has a cell given the value 1 while the rest are 0. This indicates that 1 at position *n, m* is moving node *n* to the new position of *m*. The permutation matrix is more commonly shown as the *standard unit vector u_n^T*, where at element *n* equals to 1. We also define the permutation as a function $\pi(\cdot)$ that maps *n* to $m = \pi(n)$:

$$P = \begin{pmatrix} u_{\pi(1)}^T \\ u_{\pi(2)}^T \\ ... \\ u_{\pi(N)}^T \end{pmatrix} \tag{2.37}$$

Now we if want to reorder the nodes of a *G* and keep that order in our data *X* we can $\tilde{X} = PXP^T$, where *P* is the permutation for the rows while P^T is the permutation of the columns. We now need to build a network architecture so that its predictions are invariant to the node reordering $y(\tilde{X}, \tilde{Y}) = y(X, Y)$. We also want to explicit deep networks for their hierarchical representation via layers where each layer captures information about the graph. The layer must also be non-linear and differentiable with respect to the weights. The architecture should also be flexible enough to support graphs in various sizes, support different tasks like node prediction or predicting some property at the graph level. As a result of these requirements a fixed-length representation used by neural networks is not usable.

2.5.2 Convolutions on Graphs
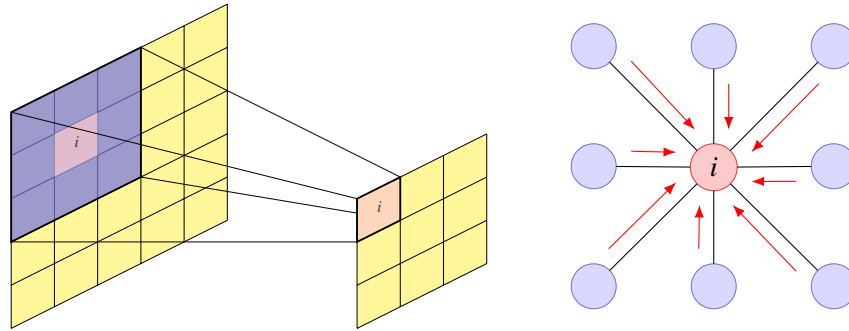
Figure 2.7: *A* is a convolutional filter computed by node *i* in layer *l* + 1, which is a function of activated values in layer *l* over a patch. *B* In a graph we can express the same computation of node *i* by aggregating its neighbours.

CNNs make successive transformations of the input so that a pixel at a layer computes a function of pixels from the previous layer via a convolutional kernel. If we consider a single pixel i at layer $l+1$ as shown by figure 2.7 the result is:

$$z_i^{l+1} = \text{ReLU}(\sum_j w_j z_j^l + b), \tag{2.38}$$

where the sum over j is all pixels within the receptive field of layer l. The same operation is applied as the receptive field slides over the image, so that the weights w_j and bias b are shared across all patches. The issue when applying this to graphs is that the weights are not invariant under different permutations. If we view the receptive field as a sub-graph where we target node i and get its neighbours $\mathcal{N}(i)$ we can then adapt equation 2.38 to the following:

$$z_i^{l+1} = \text{ReLU}(w_{\text{neighbour}} \sum_{j \in \mathcal{N}(i)} z_j^l + w_{\text{self}} z_i^l + b), \tag{2.39}$$

where $w_{\text{neighbour}}$ is our weight shared across all neighbours and w_{self} is the weight for node i. In this case we are *aggregating* the neighbour nodes via a sum and then pass it through an activation function. This can be viewed as passing *messages* from neighbours into node i, where the message is an activation of other nodes. Like how we can select a size of kernel to determine how much information we gather in a patch on a image, we can also select the amount of information to gather by defining $\mathcal{N}(\cdot)$. The number of edges given by a node i is referred to as the *degree* of i and it denoted $d(i)$. We stated that the first set of neighbours is $\mathcal{N}(i)$, this is, of course, equal to the degree of i. We can also define $\mathcal{N}_s(i)$ where we get all neighbours s steps away from node i. This introduces one of the earliest graph frameworks Message-Passing Neural Network (MPNN) [46]. Equation 2.39 is doing two steps, we first combine information from neighbouring nodes, called *aggregation* then *update* the node as a function of the current embedding creating a new embedding vector for the node. An aggregation function for all node neighbours of n:

$$z_n^l = \text{Aggregate}(\{h_m^l : m \in \mathcal{N}_s(n)\}) = \sum_{m \in \mathcal{N}_s(n)} h_m^l, \tag{2.40}$$

where h_n^l is a node embedding. We then update the embedding at node n to get h_n^{l+1}:

$$h_n^{l+1} = \text{Update}(h_n^l, z_n^l) = \text{ReLU}(W_{\text{self}} h_n^l + W_{\text{neighbour}} z_n^l + b). \tag{2.41}$$

We then perform these two operations to every node on the graph. The problem with equation 2.40 is that a sum gives a stronger single over nodes that have more neighbours than those with fewer, which leads to numerical stability issues in practice. We could normalise the sum to be

a average over all neighbours but has been shown to be worse in performance than the sum [54]. However, we if we take into account the number of neighbours for each neighbouring node then nodes with many neighbours do not overly dominate the aggregation:

$$z_n^l = \text{Aggregate}(\{h_m^l : m \in \mathcal{N}_s(n)\}) = \sum_{m \in \mathcal{N}_s(n)} \frac{h_m^l}{\sqrt{|\mathcal{N}_s(n)| |\mathcal{N}_s(m)|}}. \tag{2.42}$$

We can now view a graph neural network as a set of layers each of which transform node embedding of the same shape. The final layer determines the task for a given problem.

The first type of task we will discuss predicting the label for each node in a graph, called *node classification*. Just like standard classification tasks we use the softmax function but on each node over C classes:

$$y_{ni} = \frac{\exp(W_i^T h_n^L)}{\sum_j \exp(w_j^T h_n^L)}, \tag{2.43}$$

where $i = 1, ..., C$. We then use a loss function such as cross-entropy over all nodes and classes. Another common task is being able to predictions about edges of the graph, called *edge classification*. This is often formulated as determines if an edge between two nodes should exist or not. We can do this using logistic sigmoid on pairs of node embeddings n and m to form a probability for an edge, $p(n,m) = \sigma(h_n^T h_m)$. The final type of classification we will discuss is *graph classification* where our goal is to predict a label for a graph once trained a a dataset of labelled graphs $G_1, ..., G_N$. One such approach is to take the sum of all node embeddings:

$$y = f(\sum_{n \in V} h_n^L), \tag{2.44}$$

2.5.3 Spatial Graph Convolution

The methods of graph convolution is split into two approaches, *spectral* and *spatial*. Spectral explores the frequency domain on graphs by applying graph fourier transformations, while in spatial we focus directly on the graph structure. In this work we focus on the spatial domain. We first introduced the MPNN framework as most modern spatial approaches are expressed as instances of this framework. The following is a discussion on some of these approaches where appropriate for our work.

MoNet. A general framework proposed by Monti, Bronstein and Bresson adjusts the convolutional kernel to be a mixture of k normal distributions in a *pseudo-coordinate space* [97]. An

initial set of coordinates for each node and its neighbours:

$$y = u(x,y) = \left(\frac{1}{\sqrt{d(x)}}, \frac{1}{\sqrt{d(y)}} \right)^T, \qquad (2.45)$$

where x and y are nodes and y is the neighbour of x. This is then transformed in a dense layer with the tanh activation function, $\tilde{u}(x,y) = \tanh(Wu(x,y) + b)$. The authors restricted the weights to be $W \in \mathbb{R}^{r \times 2}$ where r is depended on the dataset. This results in pseudo-coordinates to which they are then used in defining k weight functions, $w_i : \mathbb{R}^r \to \mathbb{R}$ that the the distance of the pseudo-coordinate of each distribution Gaussian kernel. These weight functions are defined by:

$$w_i(\tilde{u}) = \exp\left(-\frac{1}{2}(\tilde{u} - \mu_i)^T \Sigma_i^{-1} (\tilde{u} - \mu_i) \right), \qquad (2.46)$$

where $\mu_i \in \mathbb{R}^r$ and $\Sigma_i \in \mathbb{R}^{r \times r}$ are learnable weights representing the mean and covariance matrix of the i-th Gaussian kernel. The convolution in MoNet is then defined as a weighted sum of features:

$$\frac{1}{|\mathcal{N}(x)|} \sum_{y \in \mathcal{N}(x)} \sum_{i=1}^{k} w_i(\tilde{u}(x,y)) \cdot (W_i f(y) + b_i), \qquad (2.47)$$

where $\mathcal{N}_1(x)$ is the neighbours of node x, $W_i \in \mathbb{R}^{F_{\text{out}} \times F_{\text{in}}}$ and $b_i \in \mathbb{R}^{F_{\text{out}}}$ are learnable parameters for each Gaussian component, and F_{in} and F_{out} are the dimensions of the input and output feature spaces.

GraphSAGE. Another type of general framework for generating node embeddings which was proposed by Hamilton, Ying and Leskovec is Graph Sample and Aggregation (GraphSAGE) [53]. Each layer has two operations: sampling and aggregation. In sampling stage we uniformly sample a subset of each nodes neighbours and at different layers a different set of samples is used. The number of samples is a hyper-parameter. As there is a different sampling for each layer then there are going to be overlapping nodes. As a result the authors provide an additional mechanism where the nodes with degrees that are lower than the sample size are over-sampled. One of the main contributions in this work was the aggregation-stage. GraphSAGE learns a set of aggregator functions that collect and summarise information from a node's local neighbourhood. For each node v, GraphSAGE defines a representation at layer k as:

$$h_v^k = \sigma\left(W^k \cdot \text{concat}\left(h_v^{k-1}, \text{aggregate}_k\left(h_u^{k-1}, \forall u \in \mathcal{N}(v) \right) \right) \right), \qquad (2.48)$$

where h_v^k is an embedding vector of node v at layer k, σ is a non-linear activation function, W^k is a matrix of learnt weights of layer k. In the original paper, three different types of aggregator

functions where proposed. The first is the mean aggregator:

$$\text{aggregate}(h_u, \forall u \in \mathcal{N}(v)) = \frac{1}{|\mathcal{N}(v)|} \sum u \in \mathcal{N}(v) h_u \tag{2.49}$$

The second is a LSTM based aggregator which uses a implicit assumption of order therefore we use a random permutation of neighbours π:

$$\text{aggregate}(h_u, \forall u \in \mathcal{N}(v)) = \text{LSTM}(h_u, \forall u \in \pi(\mathcal{N}(v))) \tag{2.50}$$

The third is the pooling aggregation which takes the maximum value for each channel:

$$\text{aggregate}(h_u, \forall u \in \mathcal{N}(v)) = \max(\sigma(W_{\text{pool}} h_u + b), \forall u \in \mathcal{N}(v)) \tag{2.51}$$

where max is an element-wise max operator, and W_{pool} and b are learnable weights. We then minimise a loss function that encourages nearby nodes to have similar representations while pushing apart embeddings for disparate nodes, a common choice is *negative sampling loss*.

Graph Attention Network. We introduced the attention mechanism in section 2.4.1 as a way to perform selective focus on the most relevant parts of the input. It has also been shown that we could use the attention mechanism in graph deep learning to make a aggregation function that has the same selective focus but on neighbouring nodes [135]:

$$z_n^l = \text{aggregate}(\{h_m^l : m \in \mathcal{N}_s(n)\}) = \sum_{m \in \mathcal{N}_s(n)} A_{nm} h_m^l. \tag{2.52}$$

where $A_n m$ is a attention coefficient used as a wight for h_m^l. There are a few different ways to construct these attention coefficients for example:

$$A_{nm} = \frac{\exp(h_n^T W h_m)}{\sum_{m \in \mathcal{N}(n)} \exp(h_n^T W h_m)}, \tag{2.53}$$

This allows the model to assign different importance to different neighbours, allowing it to focus on the most informative nodes for a given task. The graph attention network is often expanded to include multiple attention heads A_{nm}^h, where h is one of those heads. Just like in a transformer we combine the different heads using concatenation and dense layer.

2.5.4   Limitations on Graph Deep Learning

By extending the properties of deep learning to irregular domains, graphs have emerged as a potential way to model geometrical objects in many different tasks. However, they do not come

without challenges. One that is common in practice is scalability of graph deep models due to many tasks require processing of the entire graph at once, resulting most graph deep learning problems needing high computational and memory requirements. We also cannot perform parallelisation as updating one node can have an affect on others. Some solutions exist but are problem dependent. For example we could perform a fixed sampling technique like with GraphSAGE or we could extract graph into smaller sub-graphs like with gapthSAINT [165]. In both of these cases some information is discarded which can result in difficulty to capture the global structure. Another common problem is when embeddings of nodes start to become similar after interactions of training, this is called *over-smoothing*. This has been shown to happen in deep networks which results in poor performance on node classification tasks [81]. The over-smoothing is a result of a small positive eigenvalue of the normalised Laplacian matrix. This matrix is often used to represent a graph by encoding its structural properties. It is defined as $L = I - D^{(-1/2)}AD^{(-1/2)}$, where I is the identity matrix and D is the degree matrix. As the shape and density of a graph grows the problem also gets worse [102]. Three solutions currently exist to help mitigate over-smoothing: residual connections, normalisation and edge sampling [44]. Residual connections allow the model to preserve information from earlier layers by flowing information to later layers, allowing for deeper models [81]. Normalisation methods like layer normalisation [5] or batch normalisation [65] can help regularise the model to help mitigate over-smoothing. Another approach is to add a weight to the residual connection which is a form of weight normalisation [102]. Edge sampling involves randomly dropping out edges during the training process [113]. The propose of this is to reduce the propagation of information between nodes and the adding some randomness has shown to focus the model to lean diverse feature representations. The final challenge we will discuss is when the graph structure evolves. *Dynamic graphs* which stems from sequence forecasting but applied to irregular domains where nodes and edges can be added, removed or modified over time [69] With such a well established being applied to irregular domains it is not surprising to see many applications of graphs being applied to this temporal domain [71, 158, 164]. However most work where the graph structure evolves such as predicting the appearance or disappearance of a node or edge is not so commonly been explored [44]. This is due to the computational complexity of training models on evolving graphs is every high [144]. One direction we could tackle this challenge is the application of active learning strategies, which could potentially allow models to selectively update and refine their knowledge based on the most informative changes in the graph structure.

2.6 Active Learning

Active learning is one of many types of training schemes used within deep learning where the goal is to have users have some inference on smoothing the training of a function, this is especially useful strategy when there is limited amounts of labelled data. This is accomplished by intelligently selecting the most informative samples for learning during the training process. Active learning forms the second significant proportion of this work in which it is used in chapters 3 and 4. The following sections give discussion on this field.

2.6.1 Acquisition Functions

The general framework for an active learning task is given some unlabelled dataset U and maximum number of samples to label M, we aim to select some subset of M samples from U to be labelled so that they can result in the maximised improvement in model performance. In essence we obtain as many performance gains as possible having the least amount of samples labelled. This trade-off between labelled data and performance is defined by the task, the complexity of the function and the type of data. In the context of active learning the *acquisition function* is the scoring function which determines the expected performance gain from a sample being labelled. The *query strategy* is the process of identifying the best samples to be labelled. The field is usually split into the type of query strategy being used for a given task. Uncertainty sampling is one of the classical approaches to this problem where we select samples which the model produces the most uncertain predictions such as entropy:

$$-\sum_{yinY} P_\theta(y \mid x) log P_\theta(y \mid x) \tag{2.54}$$

where Y is the set of classes, θ is the current weights of the neural network and $P_\theta(y \mid x)$ is the softmax probability of some sample x and one of the classes y. Another strategy is to find a group of samples that can best represent the entire data distribution, referred to as *diversity sampling*. The most common approach is quantifying the similarity between samples, in deep learning this is called *Contrastive learning* and we use this in chapter 4 to find samples that labelled the same but are being represented in the embedding very differently. We then use these samples as a acquisition function. Other approaches change the type of neural network such as what Gal *et al.* did with a *Bayesian neural network*, where instead of a weight being a single value it is a probability distribution then using one of many inference techniques to approximate better posterior than what softmax can give [42]. In the case of

this work, they ensembled many samples with different dropout masks applied in the inference to estimate model uncertainty, referred to as *Monte Carlo dropout*. A common challenge with these approaches is computational complexity as the deeper the network gets the more intractable Bayesian inference becomes, as a result approximation methods is required such as using Kullback-Leibler divergence between different weight distributions, however they often over-fit. Deep networks have successfully been applied to the Bayesian active learning approach such as Gal and Ghahramani where they used a Bayesian-based CNN [41]. This approach still requires careful consideration in choice of priors and the type of inference technique, otherwise over-fitting becomes quite common on large dataset. Some advocate for a 2 model solution and focus on the training scheme such as Yoo and Kweon where the first model is a standard deep network and the second is a network that predicts the loss for unlabelled input data, this is a way of estimating how good the first model is at prediction, called a *loss prediction module* [163]. The task for this module is split a batch in pairs of samples x_i and x_j and then predict which one is expected to correctly have the larger loss. We can then use the one with the higher loss to be queried by the user for relabelling if needed. There has also been adversarial examples to the 2 model approach such as variational adversarial active learning [123]. In this case the the proposed approach is to use a Generative Adversarial Network (GAN) where the discriminator tries to distinguish unlabelled data from labelled ones. Large datasets where used on these deep networks without over-fitting and they showed that robust to different initial labelling pools, even when a bias is used for selection.

2.6.2 Measuring Training Effects

An application to active learning is to measure the model performance as new labelled data is added to the training dataset. This can be in both online learning as well as the traditional active learning approach where newly labelled data is added during the training process. Either way measuring the training effects shows how effective the current method is. A study by Mariya Toneva *et al.* designed an experiment to track the predictions of model for each sample and count the transitions from being correctly classified to incorrectly [130]. The authors then categorised samples by saying that if a label changes across epochs many times then the same is forgettable. While if the label assignment is consistent then it is not forgetting the learnt features, these are referred to as unforgettable samples. The experiment was performed on image data where they found that many samples are never forgotten are ones that are common features among the labels. They then removed these unforgettable samples without compromising

model performance. If a samples keep being forgotten then this can be used as a acquisition function, as proposed by Bengar *et al.* [10]. We can also quantity model change by finding samples that have the greatest update to the model if the label is known, called *expected gradient length* [120]. The gradient of the loss function with respect to the weights ∇E_θ, if we give an unlabelled sample x_i, we can calculate the gradient assuming the label y, ∇E_θ^y. Expected gradient length uses the current model belief to compute the expected gradient change as y_i is unknown:

$$\text{EGL}(x_i) = \sum_{y_i \in Y} P(y = y_i \mid x) \, || \, \nabla E_\theta^{y_i} \, || \tag{2.55}$$

As well measuring training effects this approach can also be used as acquisition function as we can use the label with the greatest update as the suggestion what the sample should be.

2.6.3 Challenges of Combining Deep Learning and Active Learning

We have shown that deep learning can learn complex functions for many different tasks and domains by training on a large dataset to find patterns that form hierarchical feature representations on different layers of the network. One of the largest dependencies in achieving good performance is the dataset. Active learning has shown significant potential in reducing the need for high-quality labelled data and smoothing out the complexity of the learning on some functions. Active learning also adds a level interoperability in exploring the feature representations of deep learning models, in which they can provide insight into samples of data that has been forgetting and shows forms of detecting drifting. Therefore, the approach to combine both fields was proposed by considering the advantages of both. Although there is high expectations on the results of this field and active learning has quite a rich literature, there is still difficulty in applying directly to deep learning. As we discussed a common strategy for finding training samples that are challenging the model to learn form is via measuring uncertainty. In classification tasks the softmax layer is used to obtain the probability-like distribution of the labels, which is then used to measure this uncertainty. The challenge is that softmax is often too confident in its distribution, drastically skewing the certainty of the labels. In active learning many advocate that the softmax response is unreliable as a measure of confidence [143], some even saying that it is worse than random sampling [139]. Even through active learning reduces the need for high-quality labelled data, there is still a need for it. As active learning relies on this for initial patterns of the data, however, the bigger challenge is that deep learning is often very greedy for the data [62]. Another alignment issue with the two fields is that most active learning algorithms focus on training of classifiers by using a query strategy on fixed feature

representations. In deep learning the feature representations and the classifier are optimised in a joint training process, resulting in many divergent issues [143]. We address some of these challenges in this thesis. In 4 we look into insufficient labelling of data and alignment issues by formatting the problem as a data refinement task to improve steel defects in images captured at different angles in the manufacturing pipeline. We explore using active learning as fine-tuning to the deep learning model and compare it to one where active learning was part of the training from the beginning. We also explore using psudo-labels of different complexities to expand the labelled training dataset in a semi-supervised training scheme. In chapter 3 we also look into the issue of insufficient data and pipeline alignment but additionally, we also look into model uncertainty in deep learning by exploring clusters of samples with a interoperability tool of different interactive visualisations. This allows to see how the affect of softmax responses has on the query and the feature representations. We also look at how we can form a set of hierarchical labels with clusters of labels that are either maligning into one node or if they split, in either class they become children of the current node.

2.7 Summary

At the start of this chapter we introduced deep learning by the different types of problems that it could help solve that are directly related to this thesis. We explored the core concept of deep learning, the feed-forward neural network and how all other approaches extend from this. However, when applied to the domain of computer vision, the computational complexity is high due to each pixel value having an associated weight. This motivated the idea to share a group of weights over the whole image based getting patches called a convolution and the success it brought to the field. These CNN concepts directly support our contribution in Chapters 4 and 5, where we leverage them for our acquisition function and hierarchical labelling systems.

The success of deep networks based on highly feature-rich representations based on a form of hierarchy led to many different architectures. The drawback of this is that they are restricted to regular domains and when applied to irregularly structured data. This motivated our discussion towards graph deep learning, which forms the foundation for our contribution in chapters 5 and 3, where we develop approaches for hierarchical labelling systems and density-based deep clustering.

Active learning was introduced in this chapter as a means to selectively update and refine

model knowledge, which directly supports our contribution in chapters 4 and 3. This is where we develop acquisition functions for feature representation positions and evolving graphs. The limitation of traditional approaches on evolving graph and hierarchical structures, discussed in this chapter, are specifically addressed by our contributions in the subsequent technical chapters, providing a targeted solution to these identified challenges.

Chapter 3

Active Deep Clustering: Exploratory Approach to Assist in Decision-Making

Contents

3.1 Introduction

Deep learning has made significant strides in many domains including manufacturing, where many processes are performed automatically [142]. In steel manufacturing however, quality control remains an active challenge due to the complex nature of steel, how it gets produced for different targeted purposes and the type of data that often gets produced in these manufacturing plants. The composite nature of steel forms defects labelled into one class but contain many shared features of other classes resulting in difficulty of optimising such functions [147]. Some defects form into different types as the damage spreads throughout the coil of steel. Many have approached these challenges from various directions. Some have framed the problem as a sequence forecasting task by capturing a video of steel as it moves down on the conveyor belt and predict how the steel defect forms into a new class [162]. Others treat it as a fine-grained classification task, acknowledging that steel is often designed and manufactured for specific customers, resulting in rare types of detects due to unique composite combinations [157].

In this chapter, we examine the performance of generative models for the task of deep clustering within the manufacturing domain. Our focus is twofold: first, we explore the effectiveness of these models in detecting and categorising steel defects, and second, we present a series of exploratory analysis approaches through a graphical user interface. This interface is designed to allow domain experts to provide reasoning on how these models detect defects, forming a bridge of interoperability between complex algorithms and practical application. We investigate the user of an auto-encoder architecture with a clustering loss, an approach that has shown promise in unsupervised learning tasks [2]. Additionally, we explore whether incorporating supervisory information can benefit the embedding space, potentially improving the model's ability to distinguish between defects and artefacts that are not defects in the capturing process [1]. Finally, we influence the training scheme with an active learning approach to enhance a patch-based classification task on the learnt embeddings from the auto-encoder, effective in scenarios with limited and error-prone labelled data [120]. By combining deep learning techniques with interactive visualisation tools we contribute to providing tangible insight for the manufacturing sector, potentially leading to more efficient and accurate quality control processes.

3.2 Background

Unsupervised learning is a set of problems where we have no target, instead we extract information from the features of the dataset by minimising a loss function for a given task. Common problems involve density estimation, learning to draw samples from a dataset, learning to denoise data, finding a manifold that the data lies near, or clustering data into groups [49]. This is usually accomplished by a learning a representation of that data that can be best utilised for the task, where the representation of the data is simpler or formed to be in a more accessible way, while keeping as much information as possible. Once in this new representation it can be more useful for different analytical tasks, creating a 2-step pipeline. Step 1 is therefore the process of transforming data into a new, more useful representation called *representation learning* [12]. The effectiveness of the pipeline is largely dependent on the quality and relevance of the new representations. In step 2 is the downstream task such as learning to generate new data samples [50], finding anomalies that deviate significantly from representations [103, 172], find associations that have been uncovered via the new representation [66] or used as a form of acquisition function within active learning by grouping of samples to name a few [31]. The rest of this section discusses the alternative approaches and the necessary background knowledge to motivate this work.

3.2.1 Low-dimensional representations

Many dataset have the property that each sample lie close to a manifold where the dimensionality is much lower than the original feature space, meaning while the dataset has many features in some high dimensional space, the meaningful variation that allows us to gain better performance in a given task lies in a much lower-dimensional space. If we focus this discussion to computer vision where we have a dataset of faces with 100s of pixels, this is of course, a high-dimensional. However, the actual variations between faces might be described by just a few factors like age, gender, expression. All of which from on a lower-dimensional manifold within the same high-dimensional pixel space. In practice, we often do not have data that are confined to a nicely-smooth manifold, especially when samples that are a large distance away from the manifold. We define these hidden or unobserved variations within the images as *latent variables* as they are inferred from other observed information in the data, in this case pixel values. The process of finding these latent variables is a form of dimensionality reduction where the high-dimensional observed space maps to a lower-dimensional latent space, that is

more compact and meaningful representation of the data. This leads to a linear dimensionality reduction technique called PCA, which finds the lower-dimensional linear manifold with the most variance in the data. PCA finds orthogonal directions called *principal components* in the feature space. If X is a some centred data matrix, PCA finds an eigenvalue solution for the covariance matrix $X^T X$. The eigenvectors form the principal components, and the associated eigenvalues is the amount of variance explained by each component. If we select the top k components and project our data into a k-dimensional subspace then we capture the most variance in the original data. The result is k top latent continuous variables, where each is a linear transformation of the features. The space spanned by the top principal components forms a linear manifold in the original feature space. The main limitation with PCA is that it is a linear method while most dataset have a complex, non-linear relationship that PCA would struggle to capture. Autoencoders are non-linear alternatives which utilises the properties of deep networks to find feature-rich representations of a hierarchical nature. Just other generative methods, these form a fundamental component of deep clustering. For a discussion on autoencoders see 2.4.2.

3.2.2 Clustering Methods

Clustering is a widely used unsupervised learning technique which aims to define a groups of similar samples based on similar features or representations. The goal is to have cluster where the data is homogeneous with each other and dissimilar to other data clusters. This is data-driven process where in convention the features or representation do provide any information about the group the data should become to. Through this process we can learn useful information about the structure such as how they are related. Clustering via partitioning is the process of splitting data into K groups. These methods are categorised into if a sample should become to more than one cluster, called *fuzzy clustering*, or if the same should only become to one cluster, called *hard clustering* [68]. In this chapter we focus on a hard clustering method called *k-means clustering*. *K*-means divides the training dataset into k different clusters based on a distance metric of the input [92]. The method first initialises k different *centroids* $\{\mu^1, ..., \mu^k\}$ around the feature or representation space then alternates between two different steps until convergence. In step 1 each sample is assigned to a cluster i, where i is the index of the nearest centroid μ^i based on the selected distance metric. In the step 2 each centroid is re-positioned to the mean of all samples assigned to the cluster i, this done for all centroids k before moving back to step 1. Convergence in the context of k-means is when there is little

to no change in position of each centroid. The distance metric of choice is usually *Euclidean distance*. Pre-training clusters in this way makes it inherently ill-posed in such that domain knowledge is required to use *k*-means effectively. Sensitivity to initial placement of centroids determines performance to a significant degree. Measuring performance can also be challenging as there is no single way to know if the data generalises well to the real world. However, we can measure properties of the current clustering such as *intra-class* cluster which shows average euclidean distance between samples of a cluster. The smaller the intra-class distance is the more similar the samples within the cluster are. Another form of measuring performance is the *inter-class* distance which is the average euclidean distance between different clusters, which determines the dissimilarity between the groups. One of the most common limitation with clustering methods is that they are not holistic to properties of the world and thus do not generalise well. This is due to the representation and how it is not changed in a learnable approach for the task. This has lead to using more meaningful representations in *deep clustering* methods.

3.2.3 Deep Clustering Methods

Like many areas in the machine learning, deep networks have contributed to cluster analysis to form deep clustering. Using rich feature representations that have a hierarchical inductive bias have shown to be beneficial component in supporting traditional algorithms. Generative methods are of particular interest due to the constraint in focusing only the essential features of the dataset instead of relying on the targets. In this section we focus on deep clustering approaches where representations are used for the task of clustering. We do not use features of raw data or traditional dimensionality reduction in these methods as it has been demonstrated that performance is limited as the dimensionality increases, especially in computer vision. Therefore many approaches use non-linear generative methods that propose a set of latent variables which get used for clustering. A common network architecture family is that of the autoencoder. Existing work in deep clustering is then classified into 2 categories: *separated clustering* and *embedded clustering*. Separated clustering involves learning a lower dimensional representation space of the training dataset then perform cluster analysis in a 2 step process [128, 63, 80]. Such schemes do take advantage of deep networks to map data into a representative feature space but the learning of this space and the clustering are two separate processes, due to the objectives not being optimised jointly. Embedding clustering takes the objective of clustering into the training scheme, forming a deep clustering task and a new framework [124, 155, 51].

Most approaches in embedding clustering form a closed-loop 2-step process where a deep network maps the original data into a representative feature space then perform clustering on the current embeddings. The approach alternates the steps in a loop until convergence by optimising a Kullback Leibler Divergence (KL) loss to enforce a self-training target distribution. The limitation with these approaches is that by using KL the distance between a sample A to sample B is not necessarily the same as B to A. This results in directly influencing the quality of the clusters being generated. The second approach within embedding clustering is using deep networks with clustering on the embedding space simultaneously, by optimising the reconstruction and a clustering loss in a single step process [2, 1]. Forming cluster analysis as a semi-supervised task by providing varying degrees of supervision has potential to enhancing performance as a form of domain knowledge inference into the training scheme. This domain knowledge could be from utilising labelled data or a form of human supervision via iterative user feedback in an active learning task [33, 8, 105, 138, 1]. By incorporating supervision of some form into the training scheme involves a great deal of time-consuming effort of labelling such data, because in many situations if clustering is the proposed method of processing then usually no labelling is given. This results in the potential direction of active learning, where we may refine data over time or simply use it to enhance the performance of the clustering. In this case a decision needs to be made by a human, in many domains where a higher form of expertise is required then more information is needed to be given to the human in order to make an accurate decision. In the following section we propose an exploratory approach to assist the human in making that decision.

Our methodology is an iterative process to exploratory analysis where we first experiment with deep clustering at varying levels of supervision and apply active learning by querying users, which is discussed in section 3.3. Secondly, we focus on dataset refinement to form hierarchical labelled datasets where we start with flat homogenous labels, which is discussed in section 3.4. We then evaluate these hierarchical labelled datasets with a few different methods in chapter 5.

3.3 Exploratory Approach

We introduce an exploratory approach to assist in decision making by a human for an active deep clustering task. For ease of use we build a graphical user interface with customisation options for different views as well as different interaction techniques. This allows a human to not only get some insight into what the selected model is deciding based on the current

embedding space, but also to assist in making decisions by the human. In this section we introduce a manufacturing dataset that our experiments are based around, the graphical user interface and the active learning experiment.

Throughout the set of experiments we evaluate this work on two datasets. The first is MNIST, which consists of handwritten digits between 0 and 9 in the form of a 28×28 grayscale image [76]. The numbers from labels of 10 classes. MNIST has $60,000$ samples in the training dataset and $10,000$ samples in the testing dataset. The second is a steel data from quality control within a steel manufacturing plant, which was captured during the cold rolling process. The dataset consists of $5,000$ grayscale images, with each containing between one and three defects. As steel is produced for a different set of purposes and depending on what that is, will mean sets of features are more common than others. This is due to the composite nature of steel and how it is targeted for a given purpose. For this reason we use 4 different types of steel coils, where each coil has 32 cameras capturing different angles and positions along the mill. Labelling is provided as a ROI over the defect. These labels should be used only as a suggestion due to the many errors, some defects do not have an ROI, while others have more than one defect in the ROI. The variance in types of steel and the accuracy of the labels motivate the need for semi-supervised approach to model such data.

The initial view of the graphical user interface can be shown in figure 3.1. Users can select different models based on the amount of supervision is given as part of that experiment, some models also do not include a clustering training scheme as a form of comparison. Users interact with the scatter plot representing an embedding space from the model, note that the embeddings have been reduced to 2-D from a selected dimensionality reduction technique. Each point in this scatter plot is a patch from the dataset, where users can right-click to see extra information such as which coil it it comes from, the segment of steel within that coil, patch location and its current label if it has one. By selecting different points, users can then view location of the patch in the image via section *c* of the figure. Within the scatter plot users can use a lasso section on a group of points, individual selection of points or a multi selection of points. Showing global context of patch location is important in domains like steel defect detection due to the changing nature of defects as they move across the conveyor belt. By showing its location, experts are able to examine if the defect morphs into a different one in a different part of the image. This view gets updated as different labelling sessions are provided. After
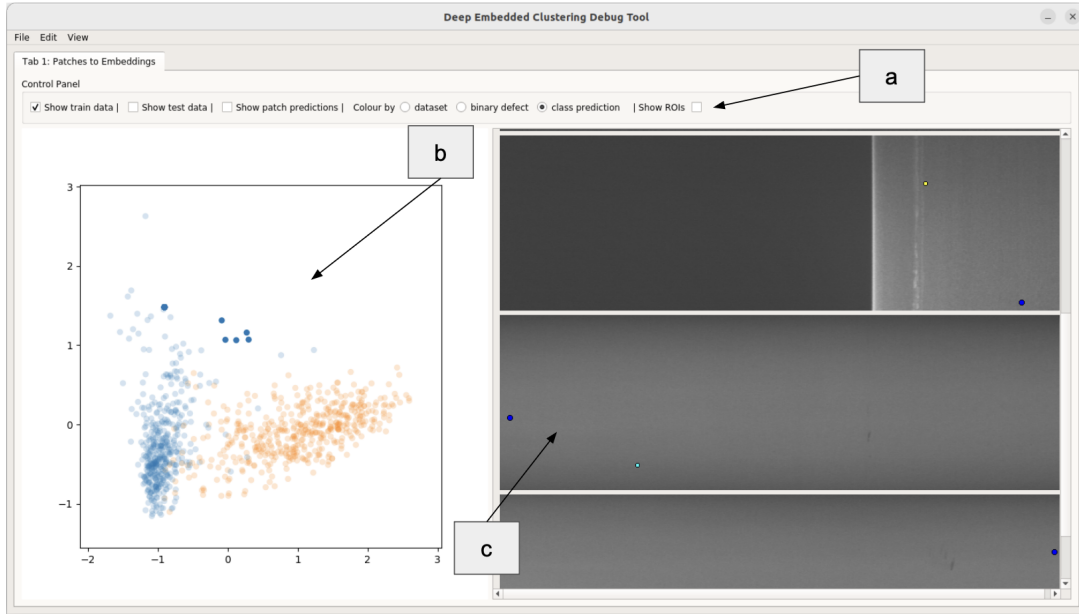
Figure 3.1: Explore current embedding space via a selected model in the user interface. This embedding space changes over each session of labelling in an active learning experiment. *a*: the control panel of this view where we can customise how we want to show the *b* and *c*. *b*: a selected dimensionality reduction technique is performed on the embeddings to view in 2-D. Users can select different points that represent patches of images and there location is then shown in *c*. *c*: segments of steel for different clusters are shown. The dots hovering on the image are the locations of the selected points from *b*.

each labelling session training of the model is performed until convergence. This of course changes the embedding space and therefore the user interface needs to respond accordingly. As successive labelling sessions are performed more densely packed clusters can be seen in this view. Sub-clusters can also be viewed in that different segments of the cluster will form smaller groups, implying a hierarchical labelling structure.

The second view of the graphical interface are grouped into user interaction and exploring different dimensionality reduction techniques, as shown in figure 3.2. In this view experts can upload images to the interface or select segments from the dataset. The experts can then perform inference on patches of the selected image based on the type of user interaction. The first is shown in section *a* of the figure, where bounding box hovers on the image. Experts move the box around the image where at each step a patch is taken from that box and it is given placement within the scatter plot. As a result, we can see patch positioning as the box moves closer to a defect in the image. By performing many of these actions the manifold
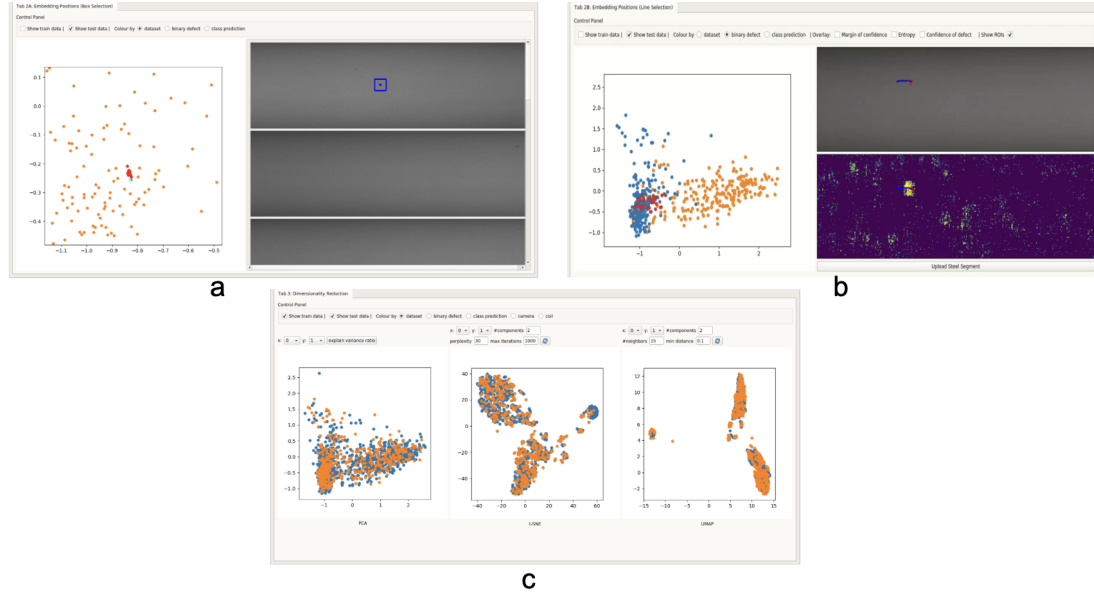
Figure 3.2: *a* and *b* use different user interaction techniques to explore the embedding space. *a* is a bounding box approach where we slide it over a selected image and for each slide step a patch is captured and positioned in the embedding space. This is a mapping from bounding box, to embedding space, to a 2-D plot. *b* used a drawn line by the user where points over that line become centre pixels of patches. The patches are then shown in the embedding space like the bounding box interaction. *c* allows for exploring different dimensionality reductions.

then starts to appear. Different forms of uncertainty are represented by an overlay on top of the selected image. This functionally only works if a model with a supervised component is selected. The second form of user interaction is via a line drawing on the image. Pixel samples are taken over the line where each pixel becomes the centre of a patch. To which they are then shown on the scatter plot in red and the newest point to be added is shown in green. Section *c* of the figure shows different dimensionality reduction techniques that can be selected by the expert. In this work the expert can select PCA, *t*-distributed stochastic neighbour embedding (*t*-SNE) [61] and Uniform Manifold Approximation and Projection (UMAP) [94]. As there are different hyper-parameters associated to each of the dimensionality reduction techniques, this view allows the expert to change them and select the main technique which gets used in the other views of the interface. Like the other views in the interface we can customise the scatter plot to show different information, including if it is a training or testing sample, and current prediction.

The final view of the interface forms the active learning experiments, where experts can relabel

65
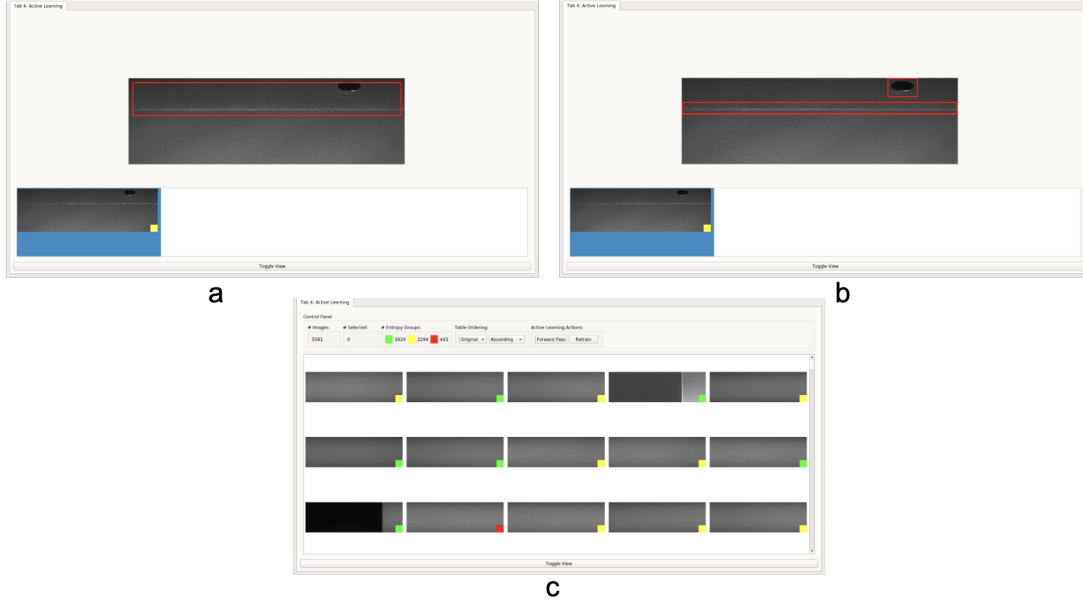
Figure 3.3: We perform an active deep clustering task where some supervision information is provided as part of the training. *a* shows a segment of steel before it was relabelled by a human. The coloured box in the corner represents level of uncertainty. The red bounding box is the current label. In *b* a human has relabelled the segment. *c* is a image viewer of a selected dataset with its own control panel for different ordering options, performing a forward pass, and re-training of the selected model.

different images of the dataset. This is shown in figure 3.3. When internally starting the main interface is section *c* in the figure, which is the image viewer. A forward pass of the dataset is performed and a colour is given to each image. The colour represents the average entropy of the softmax predictions, its an average because we sample each image *n* times, creating *n* patches. Red refers to the image being uncertain to the model, green means that the model is confident in its prediction while yellow falls in the middle of these two options. Therefore, samples that are indicated as red or yellow requires inspection by an expert. The image viewer allows us to sort images, see the total number of each entropy group, as well as perform forward pass or train the model until convergence. Any newly labelled samples gets added to the pool of supervision sample. Experts can then fine-tune with this larger pool of samples. Once complete the interface updates to the newest version of the model. Allowing experts to explore the different views again. Section *a* and *b* show a relabelling of a selected sample from the image viewer. Experts can remove labels, add new labels or change current shape of the label.

3.4 Building Labelling Systems

Algorithm 1: Modified version of SSDBSCAN

    Input: Dataset *D*, Labelled points *DL*, neighbourhood radius *eps*, min points *minPts*

    Output: Cluster assignments for all points in *D*

1 begin

2   **foreach** *labelled object P in dataset* **do**

3   Find all *P*'s neighboring points based on *eps* Get the context class of *P* Label *P* as *C* **if** *length of core points < minPts* **then**

4   Continue to next labelled *P*;

5   **end**

6   **else**

7   Grow_cluster(*D*, *labels*, *P*, *core_points*, *C*, *eps*, *minPts*);

8   **end**

9   **end**

10 end

11 Function *Grow_cluster(D, labels, P, core_points, C, eps, minPts)*:

12   Create a queue and add all *core_points* to it;

13   **while** *the queue is not empty* **do**

14   Pop the queue, known as P_n as it is a neighbour of *P*;

15   **if** P_n *is a labelled point and* P_n *label is not equal to P* **then**

16   Continue to next point in queue;

17   **end**

18   **else**

19   **if** P_n *has not been checked yet* **then**

20   Label P_n as *C*;

21   Get neighbours of P_n based on *eps*;

22   **if** *length of neighbours ≥ minPts* **then**

23   Add all neighbour to queue;

24   **end**

25   **end**

26   **end**

27   **end**
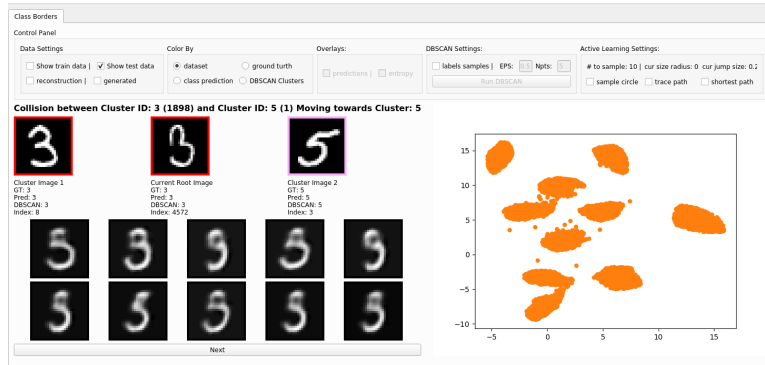
Figure 3.4: The graphical user interface for labelling generated samples based on the path between two centroids. We detect colliding clusters using a semi-supervised extension to DBSCAN. Initial centroids are labelled and we perform algorithm around these centroids. If a bridge can be made between different centroids this forms a path and we sample in the middle of that path. The user can then label those generated samples.

An application of clustering is building labelling systems that can then be utilised either in an end-to-end or in a two stage approach to supervised problems. In this section we explore the aim of identifying similar images based on different levels within the label hierarchy. We cluster embeddings in a semi-supervised fashion and identifying the colliding clusters. A shortest path is then generated between clusters, synthetic data is uniformly generated around the middle point of that path. Users label this data, which is added to the supervision component of clustering. This process is repeated by changing the shortest path to include one of the newly labelled image samples from one of the clusters. Label hierarchies in deep learning can improve accuracy but determining the appropriate level of granularity for the hierarchy is challenging.

We utilise a semi-supervised density-based clustering algorithm which is an extension on Density-Based Spatial Clustering of Applications with Noise (DBSCAN) called SSDBSCAN [79]. Labelled data points are treated as separate classes, and the algorithm tries to find the optimal clustering of the remaining unlabelled data points. The labelled data points serve as reference point to guide the clustering process. We loop over each labelled point performing DBSCAN to form clusters. Figure 3.4 displays the graphical user interface for this active learning experiment. DBSCAN builds a graph based on the density. By clustering the samples based on a labelled sample, we may hit another labelled sample that should be its own cluster. This forms a bridge between the two centroids. We display this path to the user in the interface, we then generate samples in the middle of the path. These generated samples are then shown
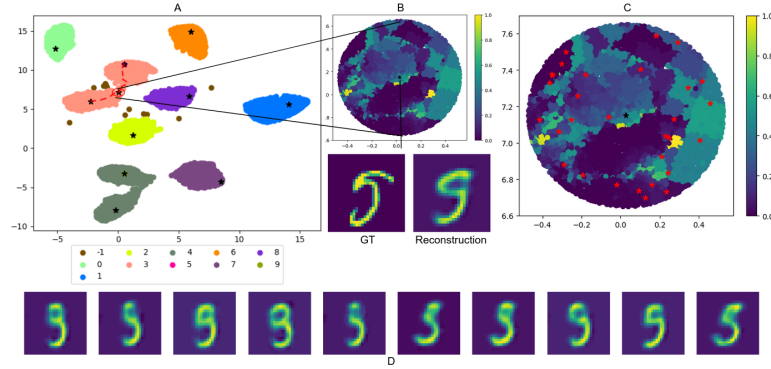
Figure 3.5: Once a path has been made we generate samples. In this figure we perform a uniform sampling approach to measure uncertainty in the predictions. We then select the most uncertain samples within an area to be labelled by the user.
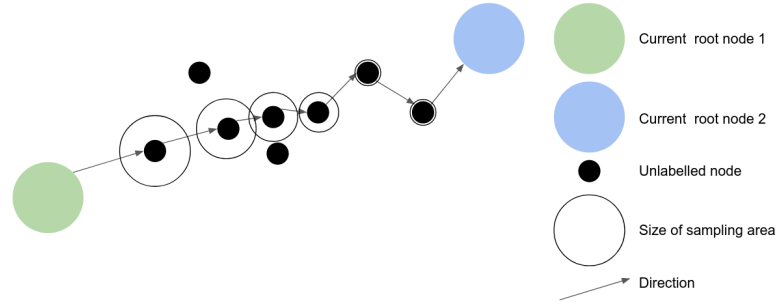
Figure 3.6: Adaptive Sampling: Given a starting ε, a decay value and a minimum ε_{min}, the size of the sample area gets smaller as we move further away from the current root node. This represents a bridge of nodes where samples closer to the root are more related. Within the sample area, we take the embedding vectors, decode them and ask the user to label.

to the user. In section 3.4.1 we demonstrate two different ways in generating these samples to be shown to the user. Once the user has labelled the generated samples to either be part of centroid one or centroid two, we then build a new path starting with the sample that is closes to centroid two. Samples that have not been labelled by the user are ignored. By following this loop over generating new paths to centroid 2 we can explore the boundary of that centroid.

3.4.1 Sampling Strategies

Sampling refers to the location of where we select our generated samples to be labelled by experts. We explore two approaches: adaptive sampling and uncertainly-based sampling. In adaptive sampling we have a ε which is the initial size of the sample area the starting point after centroid 1. It then decays after each jump to a new point. We also have a minimum sample

69
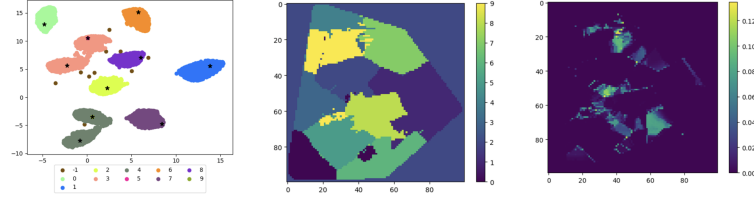
Figure 3.7: We compute the uncertainly of a circle defined by the point of interest and the radius. This is computed by generating synthetic samples via a generator and then compute the class probabilities. We then select *n* samples and choose *m* most uncertain ones. These are then labelled by the user.

size size referred to as ε_{min}. As we get closer to the boarder of a new class the more uncertain the model will generally become. By adjusting ε we can start making more fine-grained labels. Uncertainly-based sampling instead computes the uncertainly of a circle defined by the point of interest. In this case that would be the middle point of the path between both centroids. We accomplish this by generating synthetic samples via a generator model, and then compute the class probabilities. We select *n* synthetic samples and choose *m* most uncertain ones. These uncertain samples are then labelled by the user.

3.5 Implementation

In this section we discuss the implementation and design structure of our experiments. We first explore our data generation pipeline which creates batches of local patches between the different steel coils. This follows a discussion on learning representations with a clustering training scheme. We also explore adding forms of supervision into the training to form better representations. This supervision can be from the labels or a human in an active learning setting.

A common challenge in datasets that have a ROI labelling system is balancing between background and the object inside the ROI, this becomes extra problematic when there are parts of the object's spatial structure leaking outside of the ROI, more than one object inside the ROI and objects which are not labelled. The spatial nature of most objects also mean that there are many areas within the ROI which is actually background. To help in learning better spatial structure of the defect we uniformly sample pixel locations on steel segments between defects and background. The pixels from the centre of a patch and the current label is given based on if it is located within the ROI or not, forming a binary labelled dataset of images. As different steel coils are targeted for a specific customer purpose, the defects and background features

can change and as a result we split into training, testing and validating data over all segments of steel.

3.5.1 Learning Representation

The training scheme involves using the *k*-means algorithm incorporated as part of a loss [2]. This forces the embeddings proposed by the autoencoder to form an embedding space where the latent variables have a high signal if the samples are similar. We minimise the distance between samples and a given assigned centroid in the embedding space

$$E_2 = \lambda \times \frac{1}{2N} \sum_{i=1}^{N} || h^t(x_i) - c_i^* ||^2 \tag{3.1}$$

where N is the number of samples, λ is a clustering weight scaler to determine the amount of contribution the clustering has on the overall loss function, $h^t(\cdot)$ is the internal representation of a sample at t iteration, and c_n^* is the assigned cluster to the nth sample. The overall loss function is then the combination of E_2 and a reconstruction loss E_1, which is mean squared error for our experiments. At each epoch we minimise the reconstruction while keeping the cluster centers fixed. After each epoch and we have obtained a new internal representation for each sample, we assign each one to a new closest centroid:

$$c_n^* = \arg\min_{c_m^{i-1}} || h^t(x_n) - c_m^{t-1} ||^2 \tag{3.2}$$

where c_m^{t-1} is the cluster center from the previous epoch. Like *k*-means we then update the cluster centers using the sample assignment:

$$c_m^t = \frac{\sum_{x_n \in c_m^t - 1} h^t(x_n)}{\sum c_m^{t-1}} \tag{3.3}$$

where c_m^{t-1} are all samples becoming to m^{th} cluster, $\sum c_m^{t-1}$ is the number of samples that belong to m^{th} cluster. We also have the option to use some supervision into the training scheme then we add a third loss E_3 that we need to minimise, in our class we use the cross-entropy loss function:

$$E_3 = -\sum t_{i,j} log(p_{i,j}) \tag{3.4}$$

We use a modified version of the autoencoder architecture proposed by A. Alqahtani *et al.* [1]. We still target the clustering loss at the internal representation but to inject some supervision we use the learnt features from the encoder by attaching classification head to it. The autoencoder architecture consists of three convolutional layers, followed by two dense layers which

have *k* neurons, these are the hidden representations for each cluster during the training process. The representations are fed into clustering loss. For supervision we have a dense layer with a softmax activation attached to the layer before the internal representation layer. For the decoder we use a dense layer so that we can reshape the representations, this then follows three deconvolutional layers. ReLU is the activation function of choice throughout the network. This forms an end-to-end training scheme where supervision knowledge either from the labels or from a human are used across the learning process, this also creates a compact and discriminative clusters using the *k*-means algorithm.

3.6 Results

We evaluate our results using the clustering accuracy, which measures the proportion of samples that are correctly assigned to their true clusters [150]:

$$\text{accuracy} = \frac{\sum_{i=1}^{n} \delta(y_i map(c_i))}{n} \tag{3.5}$$

where *n* is the number of samples, y_i is the ground truth label of sample *i*, c_i is a cluster, $\delta(Oy,c)$ is a mapping function that equals one if $y = c$, else it is zero, $map(c_i)$ is the permutation function that maps the cluster labels to the corresponding ground truth label. We loop over all permutations to find the best match.

Table 3.1 is our list of results from the deep clustering experiments. We use MNIST as a baseline, which is commonly used within deep clustering literature. We also use the patch-based steel dataset provided by a steel manufacturer. For both datasets we use three different amounts of labelled samples to form our supervision experiments, these are 20%, 50% and 100% of the training dataset. We also use the graphical user interface to form our active learning experiments where if a sample of the MNIST or a patch of the steel dataset gets labelled then this is added to the pool of labelled samples seen by the model during training. The supervision experiments also include both convergence on reconstruction of the input as well as clustering the internal representations. In general all experiments perform well when using learnt feature representations to perform clustering compared without. The features of MNIST are quite well defined even if we disregard the spatial aspect of the data and simply flatten into a 1-D vector, we get good accuracy of 59.98%. By using a clustering loss within the optimisation we get more compact clusters and this only improves with more supervision. When running supervision via a human to label we get a slightly worse result, this is because

some of the labels made by a human are incorrect, which also demonstrates a disadvantage with active learning in that the inference from a human is very strong and if it is indeed incorrect then it can have a large affect on the optimisation. This could have been solved with a majority vote approach to labelling by humans. The steel dataset shows to be more of a challenge than MNIST, this is because we sample patches based on selecting pixels in and out of the ROI. There will be many samples which a lot of background within a patch that is labelled as a defect. The spatial geometry of the defect also has an effect on the reconstruction in other ways because many defects are relatively unique in its features. Adding more supervision helps in general with the best being 73.23% but we get a much better score of 96.32% if we incorporate samples labelled by a human. In this case we use all samples that have been labelled which is why the table states 100% labelled. This is not all labelled samples of the data like it is for MNIST.

	MNIST	Steel
k-means	59.98%	27.2%
Reconstruction	82.72%	36.98%
+ clustering	84.84%	58.5%
+ supervision (20%)	89.99%	54.21%
+ supervision (50%)	92.12%	63.08%
+ supervision (100%)	98.78%	73.23%
+ supervision via human	98.43%	96.32%

Table 3.1: Table of results for clustering quality with the clustering accuracy metric. For the supervision results we converge with both incorporated reconstruction and clustering loss terms. Varying levels of supervision is used as part of the training process. The only exception to this is when we perform an active learning experiment, all labelled data from a human is used during the training process.

3.7 Discussion

Traditional active methods struggle in a few ways when used within deep networks, this is because as we scale to higher dimensional spaces selecting the most informative sample becomes an issue due to uncertainly estimation on the predictions as well as the changing internal representations. Inspired by the work on Refinement learning with Human Feedback (RLHF) we could use more complex feedback from the human and then train a reward scheme to improve deep clustering tasks [84]. This feedback can be in the form of continuous scale where the human rates how much the patch looks like a defect or some other artefact like dirt on the steel. Richer forms of feedback have generally shown to improve performance in many tasks, stabilising the acquisition function. Clustering methods that form partitions within the

embedding space require domain knowledge from both the task and the data perspective, such as the number of clusters. This domain knowledge becomes more complex in domains like manufacturing where huge amounts of learning about the discipline is required to make an accurate choice. Model-based clustering approaches would have some benefit where we have fewer hyper-parameters to influence to optimisation and instead explore the embedding space. This could be in the form of a density-based or a bayesian approach. As there is a form of hierarchical relationship between similar defects of varying features hierarchical clustering would also be an option to explore. In chapter 5 we explore using hierarchical labels in the from of a graph and use that to predict depth based classification. The same approach could also be used to clustering where we build hierarchical clusters based on the graph structure.

3.8 Summary

In this chapter is explored the performance of generative models with the use of a clustering loss within the training scheme to form a deep clustering model. We focus on the manufacturing domain by grouping steel defects together. This exploratory approach is twofold: first we examine the effectiveness of these models in detecting and categorising steel defects with varying forms of supervision both from labels of the dataset as well as being labelled by a human using an interactive tool. In the second we use that tool to provide some interpretability to the positioning of the defects within the embedding space as well as some interaction techniques to explore the manifold. The domain expert can also customise many parts of the tool such as using different dimensionality reduction techniques and changing the scatter plot. The goal of this tool being that it bridge of interperability between complex algorithms and practical application in the form of decision-making during active learning experiments. Labelling clearly showed to have a large impact for improvement of performance by creating more compact clusters within that embedding space. The main challenge was with the steel dataset due to the quality of labels. Another direction to influence the embedding space in a more direct way is shifting the positions of the embeddings based on a distance metric, where we still use patches of steel defects but if the distance is far away from a target then we can query a human on it. This forms the next chapter of this work where we build an acquisition function based distance of samples within an embedding space. This results in refining data over time while also improving detection.

Chapter 4

Active Anchors: Similarity Based Learning for Dataset Refinement

Contents

4.1   Introduction

Gathering large pools of data has become a relatively straightforward task, with many automated ways of obtaining various sources of data. The development of pattern mining and feature representation learning approaches which leverage large collections of observations has resulted in data becoming a prized resource in recent years. Labelling such data has become an exponential problem, being a time-consuming and interaction-heavy task that involves a great amount of user effort. This development has led to the rise of active learning as a semi-supervised alternative to data labelling, where a selection of samples is labelled to refine the model's behaviour. Many domains require skilled expert knowledge to label such data; including medical image analysis [19], manufacturing quality control [126], and even genomics research [167]. In the context of the manufacturing industry, steel is a diverse and heavily used product with many different use cases. Variability in how steel is processed can result in various types of defects such as lamination, heavy scales and edge damage. Often these defects are highly variant in shape and characteristics, lighting issues in capturing the defect, and other types of artefacts not considered defects, like soot or water [126]. These often result in a challenging environment within manufacturing, and therefore it is commonplace to have visual inspection systems to help ensure a level of quality in the final product. The setup and running of such systems are complex due to the many different ways we can manufacture such products, and as such the deployment of such inspection systems will often be adapted for their circumstances [99]. These systems will often have a classification component, which attempts to recognise the different kinds of defects present on the surface of the material so that suitable approaches can be taken later in the development pipeline to correct the issue, whether this is cutting out defective regions or repurposing the product. Such systems often rely on large datasets, with engineering domain experts providing ground truth labels for the training of supervised approaches, with labelling often falling into two camps; dense labelling of pixels within the images captured by the system, or sparse labelling using bounding boxes. The dense labelling provides a fine resolution label of the defect but requires significant input by the domain expert and so is often prohibitive. The bounding-box approach allows an expert to label the defect with less overhead but can introduce incorrectly labelled pixels to what should be a ground-truth target for supervised approaches to utilise [156]. Defects that have forms of curvature, such as lamination or scratches, can be a challenge for a region of interest (ROI) based detection system, due to noise and artefacts around the defect. Other challenges for ROI-based systems include the composite nature of some defects that can formalise from

micro defects, resulting in regions that have multiple defects. This leads to predictions with less than desired bounding boxes, and uncertainty in their location and classification. The nature of a bounding box also means that labelling is often not pixel-perfect, resulting in sampled observations which are incorrectly labelled.

In this work, we propose a data refinement strategy based on querying the similarity of embedding vectors with a human-in-the-loop approach to fix mistakes in bounding-box labelled datasets for steel defect detection. We uniformly sample patches of the labelled image and learn an embedding space of patch clusters. Querying the embedding space allows us to create a deep segmentation of the steel surface, which can then assist the inspection team.

4.2 Background

The following sections provide an overview on the concepts and related work that underpin our approach. We begin by examining traditional image processing techniques that have been employed in surface defect detection, including edge detection and texture analysis. These methods from the basis for many automated inspection systems and continue to play a role in many hybrid approaches. Moving on from this section, we explore applications of deep learning in manufacturing. Various neural network architectures such as CNNs, the family of R-CNN and single-stage detectors, have improved the performance of detecting and classifying surface defects. We discuss these architectures and application to many domains. Our approach focuses around similarity learning and how it can be used for a refinement task as well as an active learning set of experiments. Techniques like FaceNet and triplet loss are critical for creating embedding spaces where similar defects cluster together, which allows use to form a defect classification approach and a refinement strategy to form a dense segmentation. We discuss these principles and their relevance to our proposed approach. The efficacy of deep learning models often depend on the quality and quantity of labelled data. As a result we discuss a potential direction in mitigating this by employing human expertise in refining the data for industrial applications.

4.2.1 Image Processing for Surface Defect Detection

Image processing plays an important role in surface defect detection in many manufacturing industries. Often contributing to aspects of quality control - which ensures product reliability

- and safety. The task of defect detection is to automatically locate and classify defects on the surface of a material with a high level of precision and efficient complexity of the algorithm. These two goals of the task are often trade-offs of each other due to the approximation requiring more time and resources to compute [109]. In previous work, utilising image processing methods have been used to dynamically define a threshold which detects the outliers, Wang *et al.* developed a histogram of image patches to find differences between samples with classes of defects [145]. A different threshold was learnt for each of the features via a random forest. A hard challenge of detection is the variability in scales and features, therefore effectively distinguishing the diverse nature can result in less than desired results. Work into the spatial domain by localising defects based on this variability have shown promising results. Choi *et al* had such work that used filter-based methods to explore defects on different scales. Good detection performance comes from this, however, the types of detection are restricted to one type known as a hole-like defects, limiting the robustness [27].

4.2.2 Deep Learning for Surface Defect Detection

CNNs have become widely popular in many domains including defect detection, due to being able to learn robust local image features during training. One of the first CNN-based approaches used for quality inspection was used to detect cracks based on image patches of concrete, followed by a sliding window approach to follow the crack during deployment [22]. This work was added upon by Song *et al*, who proposed a method based on U-Net which showed robustness to background noise while detecting cracks [125]. To deal with multiple types of defects and get better localisation results work utilised from object detection have shown great performance [89, 132]. In this new deep learning era two types of detection approaches have been proposed but focus on different ends of the trade-off between speed and accuracy. Two-stage detection uses a paradigm of high-level abstracts to fine-grained. This process attempts to improve recall with the high-level abstracts, then refines localisation in the fine-grained stage based on the high-level abstract learning. Within manufacturing a new structural visual inspection system uses this two-stage process with a method known as Faster R-CNN [23]. They showed good average precision on different types of surface defects, including those on steel. Test-time detection is another challenge in steel manufacturing due to the speed steel moves through production [126]. As a result, test-time detection needs to be at least as fast as production speeds as to not introduce a bottleneck. [83] proposed a real-time detection approach based on You Only Look Once, which can reach speeds of up to 83 FPS on cold-rolled steel

surfaces. These detection approaches are known as one-stage detectors due to completing in one-step inference, however, performance is a challenge when dealing with objects that are dense and small such as within defect detection. In general these different approaches have achieved very good performance but assume that the dataset has a large amount of high-quality labelled images, this is quite impracticable in an industrial manufacturing setting due to the impact of labour-intensive labelling practices on domain experts [91].

4.2.3 Similarity Learning

Similarity methods are a possible alternative to conventional supervised learning techniques, in which we train a model to learn what samples are similar and dissimilar based on a given metric which describes the similarity between two observations, often this metric is a form of distance. Distance is a useful way to measure similarity as the score is of continuous output form, allowing for a more fine-grained understanding of relationships between observations compared to discrete class labels. Since we focus on the relationships between our observations rather than explicit labels its more robust to errors in the labelling system. For these reasons similarity learning makes for a good use-case for training functional models or in cases where we need to transfer to a new task, due to the learnt similar features being richly placed close [154].

The most prominent approach in recent literature is FaceNet [118], which uses a CNN to learn an embedding of pairs of faces. The work is based on the triplet loss, which optimises the embedding space such that samples with the same label are closer to each other while those with different labels are pushed further away [146]. To encourage faster convergence and better generalisation, FaceNet proposes an online variant of mining observed triplets based on a large batch size [118]. The mining strategy involves finding valid triplets given a batch of embedding vectors based on selected anchors. We mine for useful positives and negatives from some metric, where the goal is to move the positives closer to the anchor while moving the negatives away. This creates clusters of similar features within the embedding space. [59] evaluated different variants of the triplet loss, finding that sampling the hardest triplets within a batch and applying a soft margin was the best for person re-identification. Using a suitable mining strategy is task dependent problem as we may have observations that are all very similar and we require an approach that finds nuance in the relationships of different observations. Using a mining strategy that selects triplets where the negative sample is close to the anchor and the

positive sample is further away would be more beneficial for learning as we want to find the observations that are the currently considered the most dissimilar but are actually similar, thus once corrected leads to bigger learning jumps between triplets.

Beyond person re-identification learning similarities naturally becomes useful in image retrieval domain such as in place recognition where given an image of a location we want to retrieve images that are locally close to where this image was taken [3, 112]. A prominent approach to this is NetVLAD, which proposes a trainable VLAD layer that can be used in a deep visual place recognition pipeline for fine-tuning to a task [3, 4]. Utilising the triplet loss in a pre-training scheme benefits from placing similar features close together in a more explicit way, allowing quick return on similar but also dissimilar samples. In general similarity learning gives the benefit of finding pairwise relations between unlabelled objects from the feature similarities and providing a level of robustness due to the transitive property between the anchor and positive pairs. The main challenge of similarity learning is the mining of triplets on large datasets, due to the growth in complexity when mining triplets as the dataset grows in size.

4.2.4 Active Learning

Active refinement and analysis are widely explored domains that leverage user input to handle low-confidence predictions and then feed the changed annotations made by the user back into the model [18, 136]. The main principle of active learning involves finding the samples that will gain the most new knowledge for the model, known as the acquisition function. Most approaches involve finding k samples to relabel based on an uncertainty metric, followed by a subsequent training session. This cycle of labelling and then training completes the framework of active learning. By leveraging user input, active learning assists in model development by providing insight into hard samples the model struggles with, allowing for better focus on complex regions of the domain. This process can also be a two-way interaction between user and model, with users benefiting from seeing which samples the model is having problems with, which can be a way of interpreting the current version of the model during the refinement cycle. The main driving force in active learning is through the acquisition function, identifying samples that warrant further user insight, and measuring which samples the user should label next is commonly categorised into three buckets; uncertainty [15, 43], sample representation [119], and training effects [120]. Once a session of active refinement is complete, an update of model weights incorporates the new knowledge. This is commonly done by updating labels of
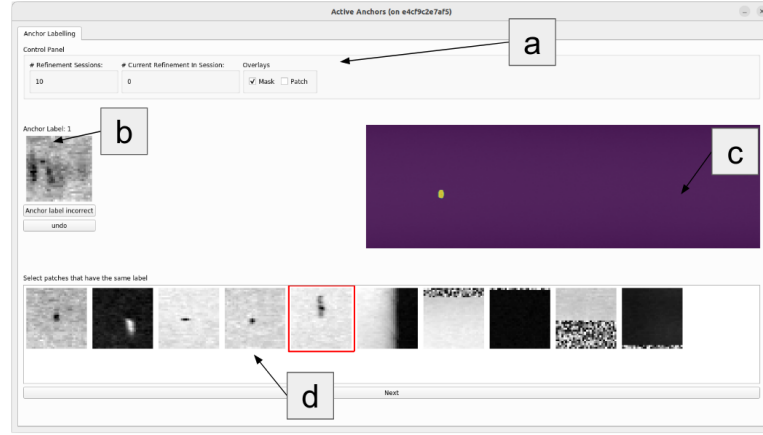
Figure 4.1: a: The control panel consists of the number of refinement sessions done, the number of current refinements done within a session and a mask overlay and an option to show where the selected patch is located within the image. Users can select either the anchor or any of the 10 patches in d and the user will see where that selected patch is located in c. b: The selected anchor of this batch. The user can update the anchor's label and undo the actions performed. c: Displays the image of the currently selected patch. d: Users select patches that they think have the same label as the anchor.

the whole dataset, but can also be achieved by populating a growing database of samples that have been observed (and potentially refined) by the user. Both of these ideas are explored in this work.

4.2.4.1 Human-Centred Perspective

Labelling complex and large datasets is an exponential problem due to time and heavy user-interaction of completing the task. There is also the value of knowledge required to make a good judgement on the uncertain samples. Depending on the dataset domain and the type of problem that needs to be solved (i.e classification, segmentation, object detection) this level of knowledge becomes critical to labelling. An effective interface needs to be carefully considered as to extract the information from the expert but also in how that information should be fed back into the model. A common data refinement problem exists in tasks like segmentation, in which we should consider how much of the sample should be labelled by the user before moving onto the next one, or the type of interaction the user has available to them like pixel selection or brush strokes. In object detection the domain may allow simpler types of classes to be labelled together while in others - such as in manufacturing - require very distant classes but does have an underlining similarity based on the composite nature of steel manufacturing.

As a result of considering these issues above, we propose a refinement process that uses more efficient user interactions via a graphical user interface shown in figure 4.1. Our interaction is a grouping task where experts are given an image and asked to select a group of images that they feel is similar. We do not explicitly ask the expert to label these samples but just inform the model these samples are similar to the one that was provided. The model then learns this similarity before applying an explicit label to it. A goal of this interaction is to extract as much of the knowledge from the expert while limiting the amount of interaction time. By utilising the interaction as a grouping task instead of explicitly labelling pixels of the image, we reduce the amount of interaction time while gaining good accuracy performance. We also add some context to the reason why some of these images are shown and how well the refinement process is going in this iterative process.

In the following section we discuss the methodology of doing this task so that the model can learn effectively with a small amount of data that is not well refined. Leading to a refined dataset and machine learning solution to the domain problem.

4.3 Methodology

Our framework consists of a new acquisition function based on embedded vectors, where we use mined triplets of anchors, positives, and negatives to refine a pre-labelled dataset of images containing surface defects on sheets of steel. During the labelling phase, the user updates the labels of the top five hardest positive and negative samples via a graphical interface. We explore three different types of initial mask labels, one using domain-expert labelled bounding boxes (ROI), a dense segmentation provided by an off-the-shelf pre-trained U-Net architecture, and the other being a uniformly random mask. User refinements are then incorporated into the learning strategy by either updating the original label set, or by developing a new set of samples which have been observed by the user during the active learning loop.

The following sections describe the triplet loss, the mining strategies, and finally how we refine the labels within our dataset.
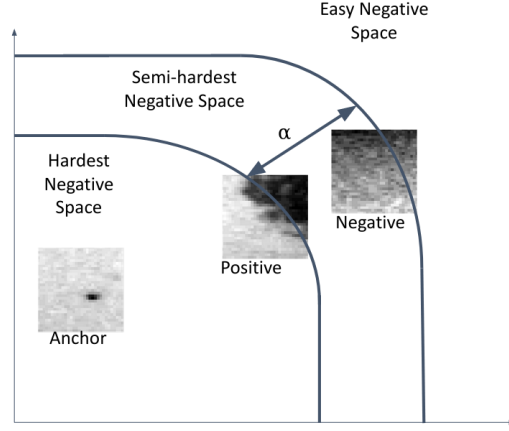
Figure 4.2: Negative samples that are closer to the anchor than a positive are within the hardest negative space. Semi-hard negatives are between the positive and a α margin. Negative samples from this space are easier than hardest negatives as they are not as close to the anchor so the triplet loss will not be as great. We avoid easy negatives as they provide no new knowledge to the model and therefore return a loss of 0.

4.3.1 Triplet Loss

Images are embedded in a d-dimensional Euclidean space, which is represented by $f(x) \in \mathbb{R}^d$ where x is an image. A triplet consists of an anchor x_i^a, a positive which has the same label as an anchor x_i^p and a negative which has a different label to the anchor x_i^n. The goal of this loss is to ensure that the distance between the anchor and the negative is greater than the distance between the anchor and the positive over all possible triplets. Therefore the loss L being minimised is as follows,

$$L = \sum_i^N d(f(x_i^a), f(x_i^p)) - d(f(x_i^a), f(x_i^n)) + \alpha \tag{4.1}$$

Here, d is a metric function, in our case, this is the Euclidean distance between the embedding vectors of the anchor and either the positive or negative. N is the number of samples in a batch and α is a margin used to enforce a distance between positive and negative pairs. Generating all possible triplets would result in some triplets that already satisfy our goal and therefore do not add much to learning, it is becoming exponentially more expensive as the size of the dataset increases. Therefore we deploy a strategy to mine different and useful triplets. The following section discusses a few of these strategies.

4.3.1.1 Active Online Mining

In online mining strategies, we compute meaningful triplets for each batch during the training process. The benefits of this are threefold. Firstly, mining the dataset in a batch typically leads to better generalisation and smoother learning [118], Secondly, if the dataset has some mislabelled data, this would dominate the mining process. This is because if a sample was labelled negative incorrectly then the model is correct in putting it close to its anchor, yet the mining strategy would consider this a hard negative and thus the loss would try to push that sample away from the anchor. We utilise this benefit in our acquisition function. Finally, due to the change in embedding space as the model learns, the triplets would change between being hardest to semi-hardest to easy triplets [59]. If x_i^p is closer to x_i^a plus some α margin than the distance between x_i^a and x_i^n, then this is considered an easy triplet as it already meets the following condition,

$$||f(x_i^a) - f(x_i^p)||_2^2 + \alpha < ||f(x_i^a) - f(x_i^n)||_2^2 \tag{4.2}$$

Easy triplets do not add much new knowledge to the model as they already meet the criteria and therefore should be avoided. Instead, we focus on selecting triplets that break the condition in (4.2). Given that we select positives such that $\text{argmax}_{x_i^p}||f(x_i^a) - f(x_i^p)||_2^2$, we select negatives that meet the following condition,

$$||f(x_i^a) - f(x_i^n)||_2^2 < ||f(x_i^a) - f(x_i^p)||_2^2 \tag{4.3}$$

These are known as the hardest triplets, as we select the closest negative to the anchor. Always selecting the hardest possible images for the model to learn from is a complex task and leads to a difficult learning environment, due to always having the largest possible loss per batch. To ease this task we can select negatives such that,

$$\begin{aligned} ||f(x_i^a) - f(x_i^p)||_2^2 < \\ ||f(x_i^a) - f(x_i^n)||_2^2 < \\ ||f(x_i^a) - f(x_i^p)||_2^2 + \alpha \end{aligned} \tag{4.4}$$

We consider these triplets semi-hard, as the negative is between a positive and the margin. They are not hard negatives as the positive is closer to the anchor, yet they are also not easy negatives as they are not beyond the margin. Throughout this mining process, for a given anchor we will always select the hardest positive such that,

$$\text{argmax}_{x_i^p}||f(x_i^a) - f(x_i^p)||_2^2 \tag{4.5}$$
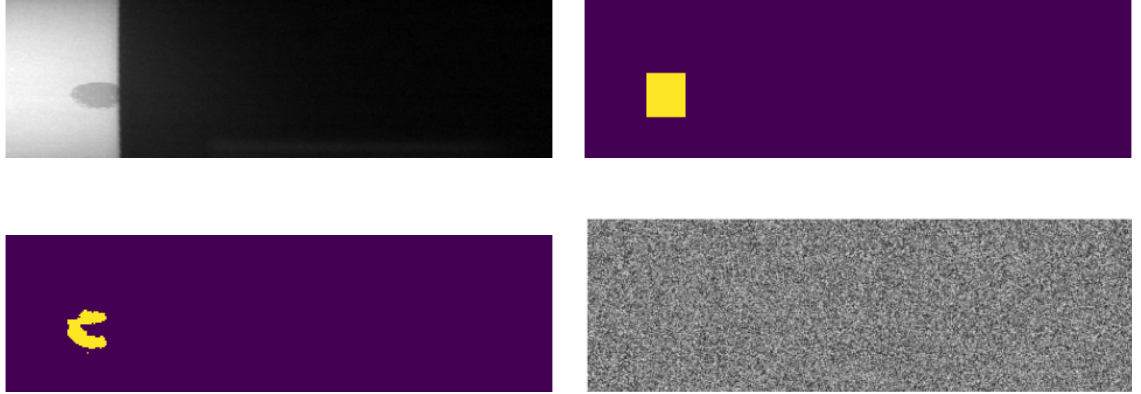
Figure 4.3: From right to left: Example raw steel that contains at least one defect. An ROI label of the defects are represented as a mask. A pre-trained segmentation mask of input. A uniform random mask which acts as our worst-case for refining. Masks allow us to uniformly sample pixel coordinates via the label, which can then be used to extract patches.

Each of the strategies depends on how we select the negative relative to the anchor and the positive. Figure 4.2 demonstrates the different mining strategies.

As we need to evaluate the distance between the anchor and every other patch within the batch, we can then also order them to find the *k*-worst patches. These patches would be considered those the model is struggling with the most, and therefore we query these patches to the user with the graphical interface shown in Figure 4.1.

4.3.2 Dataset Label Refinement

Within our framework, we utilise an online patch-generation procedure that is based on three different types of initial masks; expert-labelled ROI, a pre-trained segmentation and a uniform random mask (Figure 4.3). The expert-labelled ROI masks consist one to three boxes that interact the area where a defect could be. These masks have a large margin of error as more than one type of defect can exist within the bounding box, some defects are not labelled, and finally many bounding boxes do not cover the whole defect. The segmentation mask allows for a pre-training scheme where given the expert-labelled ROIs find common the features - such as similar types of defects - and group them together for labelling. This grouping creates a segmentation, however uncommon defects are often labelled as background class. Uniform random mask acts as our worst-case in which the most refinement is needed, this can also be considered as observation with no labels. Each of these masks act as minor dataset differences

seen in manufacturing. During the labelling phase, the user is shown an anchor patch selected at random or based on model entropy. We then use a mining strategy, to find hard positives and semi-hard negatives creating our triplets. The user interacts with the model via the graphical interface shown in Figure 4.1. Users are shown a selected anchor patch and its corresponding 5 hardest positive and hardest negative patches per the similarity scoring in (4.2) and (4.3). The user selects the candidate patches that they think have the same label as the anchor, effectively either agreeing with the model's prediction or correcting its labelling. After a batch of refinements is carried out, and the underlying labels updated, the model goes into the subsequent training phase. This cycle of labelling and then training defines our framework.

Entropy selection is one of the ways we address which samples require re-labelling over others in a ranked form. Entropy refers to the uncertainty in the predictions made by the model. The entropy calculation on the classification head (i.e. the output layer that produces class probabilities), measures how uncertain the model is about its prediction for a particular sample. The higher the entropy value the more uncertain the model is and therefore requires inspection by the user. By utilising such a method, we can reduce the total number of manual labels needed to achieve good model performance. As we are feeding the model only the samples that will potentially provide the largest learning gains.

4.3.3 Expert Interactions with GUI

In designing an active learning experiment we require to define how experts should interact with a system and how a system should use this new knowledge. The former is described in this section. User interactions are preformed via the graphical interface shown in Figure 4.1. The first type of interaction is a grouping task. An anchor patch is selected randomly within the batch of patches or by the patch that the model is most uncertain about, via entropy. The worst patches compared to the anchor are then shown. The user selects the patches they believe are similar to the anchor, correcting the model. In the second interaction users still require to group patches, however for the ones they believe are similar a shifting operation is performed. Users shift the centre pixel of the patch with the aim of putting the defect in the middle of the patch. If we train the model on user verified patches then this reduces spatial variability in the dataset, allowing for a cleaner learning curve.
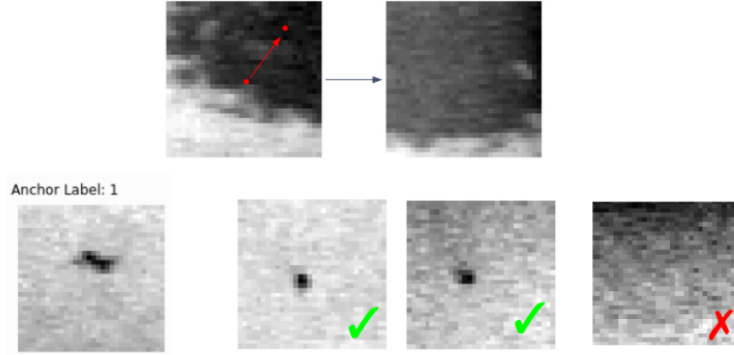
Figure 4.4: Experts have two ways to interact with our framework to refine the data. The bottom approach is a grouping task where the user compares the anchor patch to other patches that the current iteration of the model is having trouble with. Users select patches they believe to be the same as the anchor. In the top approach a user still needs to group similar patches together with the anchor patch but then performs the extra step of shifting. This moves the centre position of the patch by middle clicking a pixel.

4.4 Experimentation

As an evaluation case study, we apply our refinement approach to the domain of surface inspection within steel manufacturing, due to the labelling challenges this domain has; namely non-defect artefacts, bounding-box-based labelling difficulties, and the domain expertise required. The following sections discuss the dataset used to evaluate our framework, followed by how we implemented our approach and deployed our experiments.

4.4.1 Dataset

We evaluate our framework on grayscale images of steel, captured during the cold rolling manufacturing process. The dataset consists of 5000 images, with each containing between one and three defects. There are multiple different types of defects within this dataset, but in this instance we categorise this as a binary classification problem, identifying defect/non-defect on a per-pixel basis. Images are captured at various angles and positions along the mill, creating variance in both scene illumination and placement of the steel sheet within the camera's field of view. For the initial labelling, we utilise both bounding box labelled regions of interest, a U-Net architecture or a uniform random label. The bounding box labels are created by domain experts from a live system. Due to the nature of the defect geometry, ROIs can often provide sub-optimal labelling, with positives labelled as negative, and vice-versa. By contrast, the pre-

trained segmentation model provides dense labelling which can handle the varying shapes of the defect but is reliant on suitable generalisation and performance of the utilised model for the specific application domain.

4.4.2 Implementation

For our implementation, we train a modified version of Residual Network (ResNet) with 15 layers [57] returning a predicted binary label and embedding vector of the input given. We uniformly sample patches based on the selected mask type and then use hard mining to find positives and semi-hard mining for negative patches to form our triplets. In refinement learning is it more important to focus on the training than the validation loss as the training set is what is being refined over time. We use a reasonably large batch of 128 triplets, allowing for more available data to mine, and approach to assist in finding the hardest possible triplets which leads to an increased inter-class embedding distance [35].

Once training has converged, we move to the labelling phase with the graphical interface. The user is asked to select patches they think look similar to the anchor. These patches are considered the worst due to their distance away from the anchor. Patches that the user selected that have a different label to the anchor are updated via the mask. Patches that have the same label but are not selected get updated as well. Users continue this process until 50 refinements have been made, which then triggers the training phase. As part of the experiments, we explore three different ways of showing the refined labels to the model. The first is via our patch generation procedure, where we update the mask and then re-sample that mask during training. The second is that we show only patches that have been refined by the user via the selection of similar patches to the anchor, similar to online incremental learning. Our third approach is based on correcting the patches of defects by selecting a defect pixel and we shift the patch to the new centre, which will then get added as a refinement.

We deploy our experiments to explore how the model should be shown the refined labels with the three different initial masks, these are via mask updates and only refined patches. Each set of experiments also looks into anchor selection as these patches are what define a mined positive and negative sample, therefore we explore anchors that the model finds uncertain as well as randomly selected. The following section discusses what we found during our experiments.

4.5 Results

In this section we present our results by evaluating on the steel dataset (which was discussed in section 4.4.1). We organise the results into two subsections: quantitative and qualitative analysis. We specifically focus on using f1-score, recall and precision as they allow us to view the models ability to correctly classify defects while minimising false positives and negatives in a very imbalanced dataset.

4.5.1 Quantitative Analysis

Quantitatively evaluating refinements in a semi-supervised setting is challenging as we have no way of confirming if the refinement is correct and how good it is without expert domain knowledge. Our only way to measure these results is based on the initial sparse masks within our test set due to the refinement only being performed and saved on the train set. To generate our test set we use 1113 images of steel and select 128 patches uniformly per image, we then compare the mask label from the patch coordinate with the prediction from the final version of the model. This results in allowing us to compute the F1, Recall and Precision for each experiment.

In each experiment the user is shown patches to refine based on the selected anchor. In table 4.1 the user selects patches similar to the anchor and the update is applied to the central pixel, but in table 4.2 the user corrects the labelling by selecting the centre of the defect if one exists in the presented patch. In table 4.3 we perform the same labelling interaction as in table 4.1 but instead of the centre pixel being refined we refine each pixel within the patch. Each contains two ways of feeding the model new information by updating the underlying mask that samples are drawn from, or only using refined samples to build a dataset of user-verified samples. Our results show that updating the mask (table 4.2) provides the highest quantitative metric scores against the pre-refinement labelling; however the pre-refinement labels can be either over- or under-approximations of the actual ground truth, reinforcing the need for qualitative analysis to inspect the impact of refinement on the labelling of the underlying data. Generally, user refinement provides low recall while having a higher precision, we theorise that this is because the model is not shown the global context of the dataset which mask update provides. As we uniformly sample defects from all over the image during training. With this global context from the mask update, we see that recall is normally higher than precision. Generally, our

Table 4.1: Mask update refers to refinements made to the mask over time. During training, we uniformly sample coordinates from the mask that relate to the centre pixel of our patches. User Refined refers to the labels of patches that the user changed in the graphical interface, while observed are labels of patches that the user agrees with.

				Initial Mask					
		ROI			Pre-trained Segmentation			Uniform Noise	
Method	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision
Mask Update	0.91	<u>0.96</u>	0.86	**0.96**	<u>0.95</u>	<u>0.97</u>	0.14	0.18	0.12
+ Entropy Selection	**0.97**	**0.97**	<u>0.98</u>	<u>0.95</u>	0.93	**0.98**	0.18	0.22	0.15
User Refined	<u>0.96</u>	0.93	**0.99**	0.72	**0.96**	0.58	<u>0.64</u>	<u>0.80</u>	**0.54**
+ Entropy Selection	0.90	0.82	**0.99**	0.27	0.21	0.39	**0.67**	**0.98**	<u>0.51</u>
+ Mask Update	0.79	0.88	0.72	0.69	0.74	0.65	0.35	0.63	0.24
All	0.83	0.89	0.77	0.74	0.78	0.70	0.41	0.64	0.30

approaches work better with the initial segmentation from the U-Net, this is presumably due to the start being more faithful to the geometry of the defect in comparison to the bounding box approach. Using uniform random labels as our initial mask is the most challenging of our experiments as we do not provide any knowledge to the framework as the labels are meaningless and thus can be considered as if we are refining the label from the ground up. Refinement in this experiment to a suitable performance is possible but it takes a very long time to complete as many of the initial refinements progress is very small due to many background patches labelled as defects. As a result some iterations of the model can over-fit to background either by only training on user patches or on mask sampling strategy. Updating the mask via a whole patch refinement as shown in table 4.3 generally performs well but refining a single pixel of the patch is better. We also found that labelling whole patches converges to this worse performance compared to single pixel refinement a lot quicker. This is because labelling a single pixel provides less information to the model than a whole patch and secondly, a single pixel refinement allows for more granular knowledge to the model that is often more valuable to the learning process. Providing this nuance knowledge shows the model that there is more to learn and that is impacted in the loss metrics.

4.5.2 Qualitative Analysis

To meaningfully show the quality of the refinement process we need some qualitative analysis as shown in Figures 4.5 and 4.6. We use a single image of steel along with its initial ROI mask, each row then contains a refinement session, until we get to the last row which is the

Table 4.2: Labels in these experiments get updated by the user shifting the centre of the patch to the defect via a graphical interface, where the user is shown the 5 hardest negatives and positives based on the anchor. We then compare patch prediction with an initial mask to compute quantitative results.

| | | | | | Initial Mask | | | | |
| | | ROI | | | Pre-trained Segmentation | | | Uniform Noise | |
Method	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision
Mask Update	**0.97**	<u>0.95</u>	<u>0.98</u>	0.68	<u>0.90</u>	0.55	0.10	0.12	0.09
+ Entropy Selection	0.91	**0.96**	0.87	0.91	**0.92**	0.90	0.29	0.34	0.25
User Refined	0.83	0.72	**0.99**	<u>0.89</u>	0.81	**0.99**	<u>0.68</u>	**0.78**	<u>0.61</u>
+ Entropy Selection	0.86	0.77	**0.99**	<u>0.89</u>	0.82	<u>0.97</u>	**0.72**	0.81	**0.64**
+ Mask Update	<u>0.96</u>	**0.96**	0.97	**0.94**	<u>0.90</u>	**0.99**	0.46	0.69	0.34
All	<u>0.96</u>	<u>0.95</u>	0.97	0.83	0.81	0.85	0.49	<u>0.72</u>	0.37

Table 4.3: Labels in these experiments get updated via every pixel within a patch. During training these newly updated masks are uniformly sampled to get coordinates that are then used to extract patches.

| | | | | | Initial Mask | | | | |
| | | ROI | | | Pre-trained Segmentation | | | Uniform Noise | |
Method	F1	Recall	Precision	F1	Recall	Precision	F1	Recall	Precision
Mask Update	0.70	**0.98**	0.54	**0.96**	**0.96**	**0.97**	<u>0.25</u>	0.32	0.20
+ Entropy Selection	0.74	**0.98**	0.59	<u>0.80</u>	<u>0.89</u>	0.72	0.24	0.31	0.19
+ User Refined	<u>0.79</u>	0.84	**0.75**	0.57	0.62	0.52	**0.32**	**0.47**	**0.24**
Both	**0.80**	<u>0.89</u>	<u>0.72</u>	0.72	0.71	<u>0.73</u>	**0.32**	<u>0.36</u>	<u>0.29</u>

final version of the model for that experiment. We see that the model does refine better for experiments where we just feed refined examples. Often if we uniformly sample from the mask we get a circular segmentation around and filling the ROI, which is why we often see higher precision in these experiments. Selecting the anchor based on the highest entropy often leads to worst results, we believe this is due to the task for the model to learn being harder. This is because we mine the hardest negatives and positives from the hardest anchor in a batch. Our approach is also able to find defects that have not been labelled while also ignoring non-defect artefacts like luminance. We can also see that area within the ROI also gets refined such as in figure 4.5 (right) where we have two defects labelled as one.

In figure 4.6 we show that by adding more user interaction into the training our refinement will get better, this has the caveat that more time from domain experts is needed. Each of these patches are shown with an overlay displaying the pixel confidence of a defect, with ROI based approaches we see that there is a level of uncertainly around the box due to many of the pixels

labelled as positive are actually negative. We find that the best way to deal with this uncertainly is to perform pixel labelling and shifting patches so that the defect is in the centre, this results in better ROIs and dense segmentations.

Figure 4.5: We show a forward pass of a single image of steel producing a dense segmentation over refinement sessions with three different experiments. This is accomplished via the graphical interface where the user is given the 5 worst negatives and positives based on an anchor, then selects new centre pixel for that patch. Row one displays the image and the initial ROI mask. Column one is based on user refinement, column two is mask updates, and column three is user refined with entropy anchor selection. Each row shows a subsequent refinement after a session.
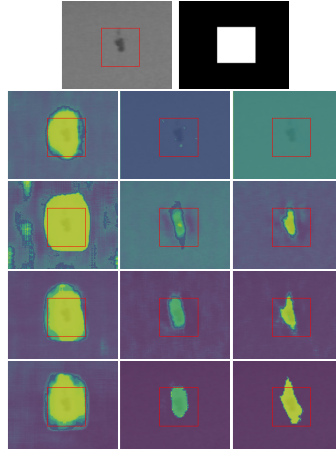
Figure 4.6: Zoomed in crop of the dense segmentation over refinement sessions when users select defect centres from within presented samples. Red box indicates initial ROI labelling, heatmap shows confidence of model output for positive defect detection. From top to bottom: steel image and initial ROI mask, refinement passes 1-4. From left to right: update of label mask only, user verified dataset, and user verified dataset with entropy based anchor selection.

4.6 Summary

In this chapter we proposed a data refinement strategy based on querying the similarity of embedding vectors with a human-in-the-loop approach to fix mistakes in bounding-box labelled datasets. Within this work we focused on manufacturing where quality control is a challenge due to the varying composite nature of steel and the types of defects that get produced. We

uniformly sampled patches of the labelled image, learn an embedding space, then query samples to a domain expert that have been labelled as similar to others but the model finds them to be different. This similarity score is based on the distance between 3 different samples. Our approach takes an bounding-box labelled datasets and refines them to be a deep segmentation, which in the domain of manufacturing can assist the inspection team in finding defects more accurately. We refine our methodology as a data refinement task due to leveraging large collections of labelled observations being a prized resource. This is because labelling such data is a exponential problem, as it is time-consuming and an interaction-heavy task. However, this cannot be a completely automated task and hence the need of a user to help direct the optimisation. In this work we explore binary problems but defects are naturally formed from a hierarchical nature as they can evolve into others as the steel moves on the conveyor belt. This forming from one defect to another is a type of hierarchy in the labelling system. In the next chapter we explore modelling this hierarchy in a classification task where a graph forms the labelling structure and we learn node embeddings based on samples that represent that node. This then feeds into hierarchical classification heads where earlier heads are more abstract and as the data follows through the network, more descriptive labels are used.

Chapter 5

Modelling Types of Hierarchical Relationships

Contents

95

5.1   Introduction

Complex class labelling systems refer to when the target of a supervised learning problem has multiple, intricate, and abstract relationships between categories. In which our aim is to model such relationships in both an abstraction and fine-grained level of data. We usually represent different depth levels of granularity, from low-level, fine-grained features to high-level abstraction in the form of a tree data structure. These systems are highly inspired by structure of human cognition and perception where we recognise patterns of various levels of abstraction to define an object[74]. Most machine learning approaches attempt to mimic this process by having different layers to extract features at different levels of complexity. This hierarchical structure allows for learning progressively more abstract representations of input as data flows deeper into the model [78]. This process is almost always implicit in nature because the model can learn some form of hierarchical representation from training data without explicitly guiding how to form these abstractions and output to some flat space. Due to this implicit nature new research fields attempt to use data to understand reasoning of models, bias and fairness of predictions, adapting to drifting and data efficiency[36]. By utilising a tree data structure to represent labels at different levels of granularity we can invoke explicitly defined relationships and dependencies to form a more structured and interpretable representation of data [52]. This is especially useful in tasks which exhibit complex and interconnected characteristics such as in manufacturing of composite materials or comorbidity analysis of patients in the healthcare domain[85, 151].

In this work we utilise the hierarchical relationship often provided but not used in many datasets as a method to solve classification problems. These relationships are modelled via graph-based machine learning approaches where the leaf nodes are individual samples and intermediate nodes are an aggregation of their children. These nodes from a more generalised label and as we move down the tree, the labels become more specialised. We use these labels to form a classification task and thus our input samples x has many labels associated to it based on the structure of the tree. We define a multi-label classification task where we chain the predictions from one depth level to the next, as to allow the prediction of the depth level to flow into the next. To truly test our methodology we use both syntactic and commonly used vision-based datasets. We build our own syntactic-hierarchical generator to construct many different datasets as to allow us to test class boundary within the input embedding of the model, it also allows us to perform analysis on edges between intermediate nodes, look into the aggregation of the

data in the node and how loss normalisation affects on the different classifiers. We perform this analysis as the prediction of our input higher up in the tree has a huge effect later down the tree as that is used to make predictions in that depth level (or sub tree), this can cause cascading error. We also explore the methodology on MNIST, CIFAR-100 and ADE-20k [76, 72, 170].

In the following section we discuss the literature behind how we define different types of relationships, how these relationships can be modelled into a multi-label problem within machine learning.

5.2 Background

5.2.1 Types of Hierarchical Relationships

Human cognition and perception recognise patterns at various levels of abstraction to define an object, forming a hierarchical structure of labels with different granularities [60]. In our work, we focus on encapsulation and sub-classification relationships within these hierarchies. The concept of encapsulation is inspired by how humans actively sample visual information through rapid eye movements known as saccades [161].

Saccades allow the visual system to sequentially focus on different parts of a scene, building a hierarchical representation of the visual input. This process does not just apply to face perception but to general visual cognition. The brain integrates information from these sequential fixations into a coherent, hierarchical understanding of the scene [161]. Each fixation contributes to a tree-like structure of visual features, where lower levels represent fine-grained details and higher levels capture more abstract and holistic information. This hierarchical processing in human vision motivates our approach to incorporate tree-like structures of labels into a machine learning training scheme, capturing both fine-grained details and high-level abstraction. This approach aligns with recent developments in attention mechanisms within the field, where a model learns to focus on relevant parts of the input sequentially, similar to how saccades direct human visual attention [88]. The encapsulation relationship in our models reflect how individual elements are integrated into higher-level concepts, while sub-classification relationships capture the way humans categorise visual information at different levels of specificity. This rich form of information is due to well defined semantic relations from experts in the targeted domain. The most common form of this sub-classification relationships are dense

in their categorisation, such as canines can be split into wild and domestic, wild branches off into different categories of wolfs while domestic would branch off into different types of retrievers. Categorisation that is spare is less strict in its relationship, this can be objects like paintings belong on a wall in a living room not on the floor. This type of categorisation is often used in robotics where this positional representation of objects within different rooms is used to learn the space being explored [152].

5.2.2 Explicit VS Implicit Learning of Knowledge

Many in the computer vision community have built datasets that have some form of hierarchical relationship. This can form into a physical structure such as in ShapeNet (encapsulation/ part off relationship) or based on semantic relations of labels as in ImageNet or CIFAR100 (sub-classification) [72, 34, 24]. Even with this rich form of information provided as part of the dataset it is rarely used as a form of learning these relationships based on image data, though some attempts have been made to address this gap [159]. With the design of neural networks having different layers of neurons to process information at different levels of abstraction, we can extract features at different levels of complexity. This hierarchical structure allows for learning progressively more abstract representation of input as data flows deeper into the network [78]. This implicit learning of knowledge aims to take single-label and flows into a multi-label without explicitly informing the network. The flow of learnt abstraction is what has led to many fields not using the multi-label structure of datasets, however this does come with many problems from the learning of these structures to the need of having labelled interconnected characteristics in many domains.

When explicit machine learning techniques are used, they are often referred to as a constraint form of learning due to limiting the search space in which the model can learn from [96]. This has lead to literature in fields like semi-supervised, unsupervised and self-supervised learning. These use an implicit learning of knowledge due to limits in providing labels, this could be generated as part of the training process or a mixture of providing labels. With the lack of ground truth examples it can be hard to evaluate performance and adjust the learning process. This can lead to other challenges like extra difficulty in interpretability, risk of overfitting. As a result most machine learning areas use a some form of explicit learning, such as direct input-output pair training phase in supervised learning to graph machine learning which uses a rich amount of information to form a graph data structure [54].

98

The trade-off between explicit and implicit learning approaches should be considered in model design as well as the training scheme given the data and the task to be completed. Explicit learning, while potentially more constrained, often leads to more interpretable models and can leverage domain knowledge effectively [116]. It allows for more richer unstructured forms of data as well as hierarchical structures, however, this approach may limit the model's ability to discover novel patterns or relationships not explicitly encoded in the data. Implicit learning approaches, such as those used in deep learning, offers greater flexibility and the potential to uncover complex, non-linear relationships in the data [11]. This allows for adaption of a wide range of tasks without requiring extensive hand-engineered features or explicit encoding of hierarchical relationships. In general, this often requires larger datasets, are more computationally intensive, and can be less interpretable. The choice between explicit and implicit learning often depends on the specific requirements of the task, domain knowledge, and the need for interpretability or flexibility. By utilising a hybrid approach that combines elements of both explicit and implicit learning, we can offer the strengths of each approach while mitigating a few of the respective weaknesses [117].

Using hierarchical labelling structure naturally leads to the domain of multi-label classification, where each instance can be associated with a set of linked labels rather than a single class. In the following section, we discuss the formulation of multi-label classification and how we can adapt neural networks to handle the complexities of hierarchical, multi-label data in computer vision and graph-based tasks.

5.2.3 Multi-Label Classification

In situations where samples have many associated labels concurrently, then such problems are known as multi-label learning [107]. This approach extends from the standard single label, where we typically have a set of finite labels that can be applied to samples of multi-label data. The usual goal is then to predict all the relevant labels to the single input but this could also involve ranking of those labels. The traditional approaches for solving multi-label problems are algorithm adaptation or problem transformation. Where the former aims to extend method to handle multi-labels such as Multi-Label K-Nearest Neighbours (ML-kNN) [166], multi-label decision trees [28], and adaptations of support vector machines [39]. In the domain of computer vision, CNNs have been adapted for multi-label image classification tasks, such as the CNN-RNN framework which combines CNNs as the feature extraction with RNNs to capture label

dependencies [140]. These adaptations typically involve modifying the network architecture and loss functions to handle multiple labels simultaneously. Problem transformations involves taking the multi-label dataset and convert it to be either one or multiple label classification tasks. The most common methods within problem transformation are label power-set, binary relevance and classifier chains [107]. In our work we use classifier chains in which we have multiple interconnecting classifiers in a sequential chain, where the predictions of the preceding classifiers can serve as features for subsequent classifiers. We can then leverage the learnt hierarchical information to help in the prediction of the sub-labels. The challenges with this approach is there is a high computation cost when the dataset has a large number of labels, models are often sensitive to the order and structure of chains which results in difficulty of capturing complex label dependencies [107]. Traditional classifier chains also have limited ability to capture the higher order correlations between linked labels; therefore, graph deep learning has been employed to address some of these problems [25].

5.3 Methodology

$$x_i = [G(n_{L^0}), G(n_{L^1}),$$
$$G(n_{L^2}), G(n_{L^3}),$$
$$G(n_{L^4})]$$

$$y_i = [\text{‘root’}, \text{‘123457’},$$
$$\text{‘1247’}, \text{‘27’}, \text{‘2’}]$$

$$x_{i+1} = [G(n_{L^0}), G(n_{L^1}),$$
$$G(n_{L^2}), G(n_{L^3}),$$
$$-1)]$$

$$y_{i+1} = [\text{‘root’}, \text{‘123457’},$$
$$\text{‘35’}, \text{‘3’}, \text{‘-1’}]$$

$$x_{i+2} = [G(n_{L^0}), G(n_{L^1}),$$
$$G(n_{L^2}), G(n_{L^3}),$$
$$-1)]$$

$$y_{i+2} = [\text{‘root’}, \text{‘0689’},$$
$$\text{‘08’}, \text{‘0’}, \text{‘-1’}]$$
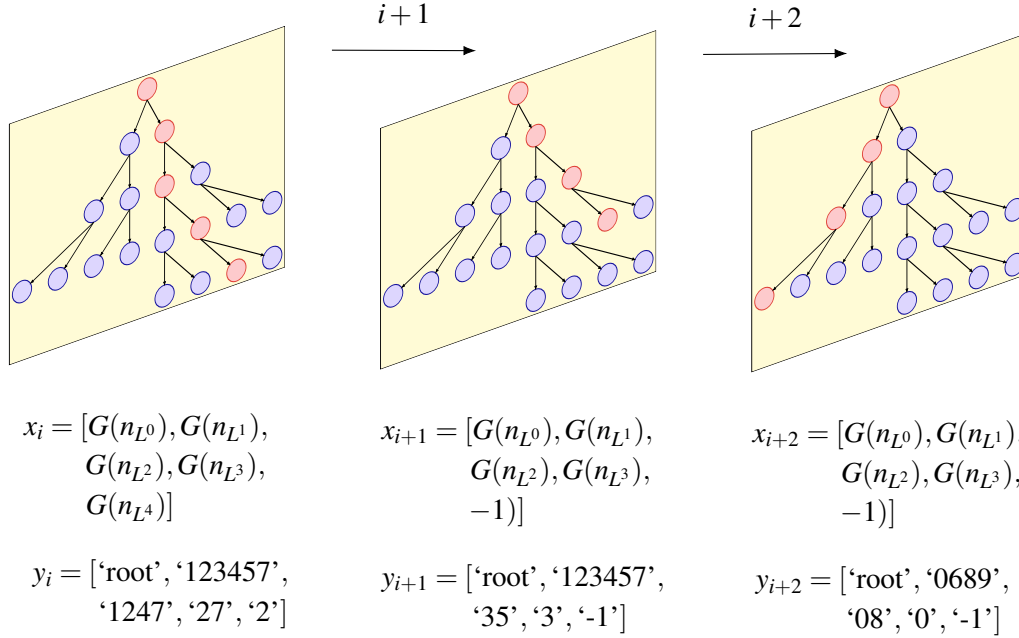
Figure 5.1: Each x_i represents a path to the root, where we sample every node representation within that path based on a function G. This means that the largest path is depth of the tree and therefore we pad the x_i if the current i is an intermediate node. We use a multi-label y based on this path, starting from the root to the current node. In this figure we are using a tree built for MNIST and so the root represents all digits and we split them up as we move down the tree.

Our methodology is based around different variations of classifier chains to model the relationship between nodes of a label tree, where each chain targets a depth level of the tree. Different variations of classifier chains define how predicted information should flow to the next. We use a classifier for each depth level excluding the root and the leaf node, as the root represents the entire dataset in a single label resulting in no change as each experiment uses a single dataset. The leaf node contains an individual instance of the dataset and as such has a unique label for that sample, if utilised this results in a single-instance classification problem and challenges like good generalisation and evaluation comparisons arise. We choose to focus on modelling of relationships between groups but single-instance problems forms part of section 5.6. By using classifiers to target different depth levels of the tree we form a path from a given node to the root, where the node could be either a leaf, intermediate or a root node. Each model is broken up into a graph-based feature extraction section followed by and the classifier chains. Graph Convolutions allow us to form a low-dimensional embedding of nodes while leveraging

hierarchical structure of the data within the representation.

Our input is of shape $n \times n$ where n is the number of nodes in our tree. A row, referred to as a feature vector, is defined as:

$$x_i = [G(n_{L^0}), G(n_{L^1}), ..., G(n_{L^d})], \tag{5.1}$$

where x_i is a instance of the data, n is the node representation of the dataset labels, L^d is the node at depth level d. As a node can represent many data instances, G is an aggregation function which results in a vector of the same size as a single instance of the dataset, this ensures that each $G(n_{L^d})$ have the same length. We build the shortest path from any node in the tree n_{L^d} to the root n_{L^0}. We start with the root because the labels become more specific as we move down the tree, therefore as we pass this data through the model, the deeper classifier chains get more specific and map to the deeper nodes of x_i.

Graph Neural Networks (GNNs) have emerged as a powerful framework for learning on irregular data, in which they extend on the concept of CNNs to non-euclidean domains, allowing for processing of data with complex relational structures [44]. As our data forms a hierarchical tree we use spatial graph convolutions to directly operate on it within the node domain, this allows us to model neighbourhoods of each node with either a filter or an aggregation function. Spatial graph convolutions also easily translate to domain-specific tasks than spectral due to domain-specific knowledge often involving local patterns or rules, which spatial convolutions inherently capture [44]. This is particularly relevant for our hierarchical label structure, where relationships such as sub-classification or encapsulation relationships is the aim of our modelling. The local structure of each node x_i's immediate neighbours is $\Gamma_1(x_i)$, representing one-step, however, we often use a larger number of steps away from the focal node, as to allow for learning of greater spatial relationships, like a convolutional kernel. By changing the steps away from the focal node we capture relationships between different levels of the hierarchy. The framework of choice within the spatial graph domain is GraphSAGE as it considers all nodes in the graph to become a focal node and therefore generates embeddings for each one [53]. Each GraphSAGE layer consists of two procedures: sampling and aggregation. In the sampling stage a subset of each node's neighbours are uniformly sampled and at each layer a different sampling is made, the number of samples is a hyper-parameter that can be tuned based on the depth and breadth of the tree. The aggregation stage then follows by aggregating

the sampled nodes,

$$h_{l,\Gamma(x)} = \text{aggregate}_l(\{h_{l-1,y} \forall_y \in \Gamma(x)\}) \in \mathbb{R}^c, \tag{5.2}$$

where $h_{l,\Gamma(x)}$ is the aggregated feature vector for node x at layer l, and c is the dimension of the output feature vector. We then perform the concatenation of $h_{l,\Gamma(x)}$ with the focal nodes own features from the previous layer, $h_{l-1,x}$ in a dense layer:

$$h_{i,x} = \sigma((h_{l-1,x} \parallel h_{l,\Gamma(x)})W_l), \tag{5.3}$$

where $W_l \in \mathbb{R}^{2c \times d}$ is a matrix of learned weights for the lth layer and σ is a non-linear activation function. In our experiments we use the mean aggregator, one of three that was suggested by the authors as a way to flatten out contributions from over-sampled nodes. As we predict the path from root to a leaf node, GraphSAGE's node embedding process captures the relationship between different levels of the hierarchy. Each node's embedding is influenced by both its parent and child nodes, reflecting the hierarchical nature in the data.
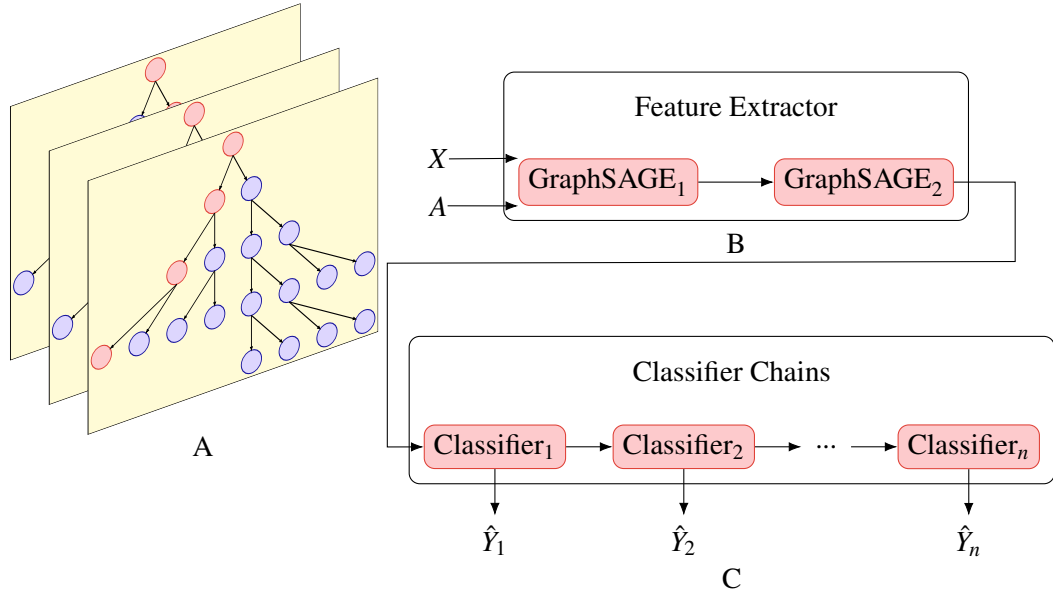
Figure 5.2: This figure expresses the full pipeline of our methodology. A: We first build our tree that forms our input X, Y and A. The root of the tree represents all labels within the data and each leaf node represents an individual data sample, intermediate nodes define the hierarchical structure of labels. Each vector of X is a path starting from the root to a node within the tree. Aggregation of data within each node is used to ensure that there is a fixed size. B: We pass our X and A into the feature extractor part of the model, we use GraphSAGE to learn node embeddings which captures the relationships between different levels of the hierarchy. C: The embeddings, $\hat{X}$, are passed into the classifier chain section of the model, allowing us to predict different depth levels of the tree.

To leverage the hierarchical structure learned by the GraphSAGE layers for a multi-label classi-

fication task, we integrate the use of classifier chains. Figure 5.2 summarises our methodology, where we first structure our data so that each vector of X is a path from the root to a node, this can be a leaf or an intermediate node. The root of our tree represents all data within the dataset while a leaf node is a single sample. The intermediate nodes define the hierarchical labelling structure of the dataset. We use a sampling followed by an aggregation function at each node to allow for a fixed size vector, as different nodes contain any number of samples. Once built we pass the X and the adjacency matrix A into feature extractor section of the model. The embeddings from GraphSAGE for each node in the tree serve a hierarchical features for the classifier chains. The embeddings encapsulate not only the individual characteristics of each label but also its relationship within the labels' children and parents. By feeding the embeddings into the classifier chains, we aim to capture label dependencies from the sequential path of the prediction process. In the following section, we discuss specific configurations of classifier chains in our experiments.

5.3.1 Classifier Chain Configurations
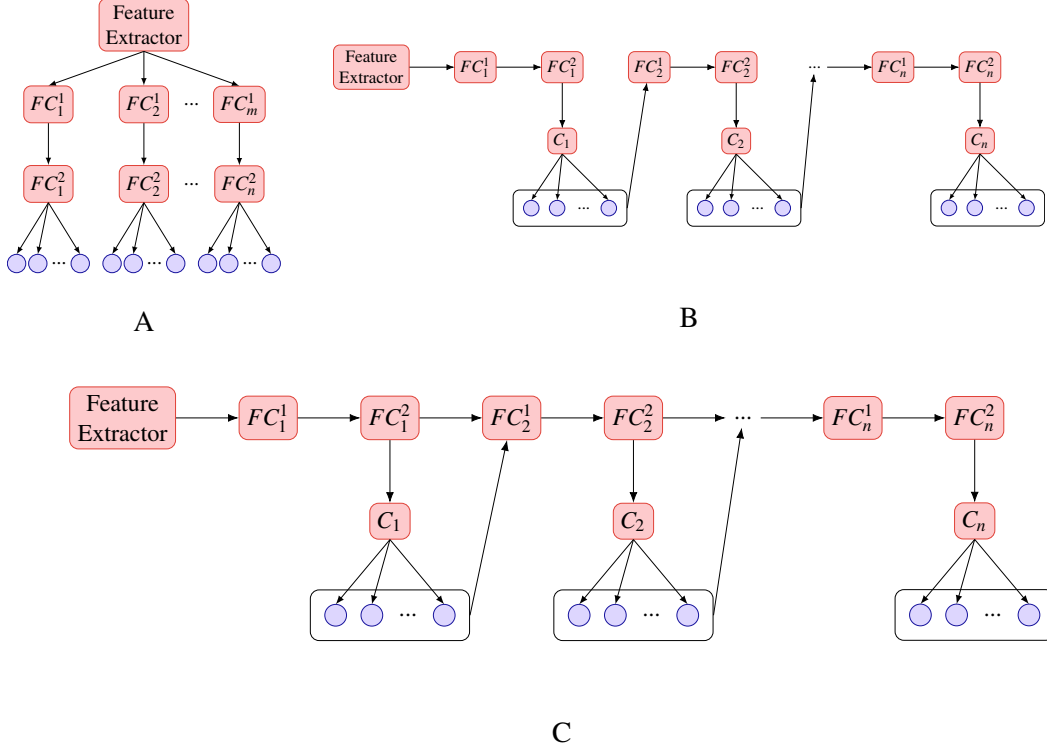
Figure 5.3: A: Classifier Chain Type A is the first of the experiments that build up to different classifier chain experiments. This type is more related to a naïve approach where we do not chain any of our predictions and therefore different dense layers may learn similar properties. Each dense layer connects to a classification head that refers to a depth level of a given tree. B: Classifier Chain Type B chains the predictions from each classification head into a following dense layer. This then feeds into another classification head. We continue this process of building the model until we reach the max depth level of the tree. C: Classifier Chain Type C is similar to type B but we concatenate the predictions from the classification head with the previous dense layer, allowing us to share more of the learnt features. Given the chosen classifier chain configuration, FC_i^j is fully connected layer where i refers to the ith classifier head and j is the jth layer.

The formalisation of classifier chains in a probabilistic setting was first proposed in [26], h_j denotes the j-th classifier,

$$h_j(x) = \underset{y_j \in \{y_1 \times ... \times y_n\}}{\arg\max} P(y_j \mid x, y_1, ..., y_{j-1}), \tag{5.4}$$

where we find the probability of the y_j given x and all the previous labels in the chain and each y_j represents the set of possible values for a given depth of the tree. This formalisation captures the interdependencies between labels, allowing us to model the relationships as each label

prediction on all previous labels in the chain. Depending on the chain structure, we can then inherently models a form of conditional label dependence. As a result of this the general question of how to specifically order and structure the chains around the associated labels within the dataset becomes crucial for optimising model performance. For our methodology we focus on three types of classifier chains referred as A, C and D, shown in figure 5.3. Classifier Chain Type A is our initial approach and serves as a naïve baseline. This configuration is not a true classifier chain but rather a multi-head model where each classification head corresponds to a specific depth of the tree, excluding root and leaf nodes. If a branch of the tree has leaf node but other branches have greater depth then that leaf node is not used in the classification for the particular instance of x. In this setup, each classifier head operates independently, making predictions based solely on the graph based feature extractor. While simple, this approach ignores potential interdependencies between labels but some implicit capturing of hierarchical information is learnt via the feature extractor. Building upon the limitations of configuration A, a true classifier chain is built with Type C. In this case, each classifier in the chain takes into account the predictions of the previous classifier, resulting in modelling conditional label dependencies. However, the flow between classifiers is limited to the raw prediction, resulting in an extremely small shape for the features to be fed into the next classifier. Resulting in not fully exploiting the learned representations. Type C extends on the chain structure of B by introducing a dense layer that concatenates the previous classifier's output with its learned representations. The progression from A to B, and then to C represents an incremental approach to capturing and utilising label dependencies in multi-label classification tasks. Type A serves as a baseline, highlighting the potential shortcomings of treating each label independently. Type B introduces the basic chain structure, allowing for sequential label predictions that consider previous outputs. Finally, type C incorporates the learnt representation of the previous dense layers with the information of the prediction, leading to a richer form of explicit learning of hierarchical relationships.

5.3.2 Building Hierarchical Labelling Systems
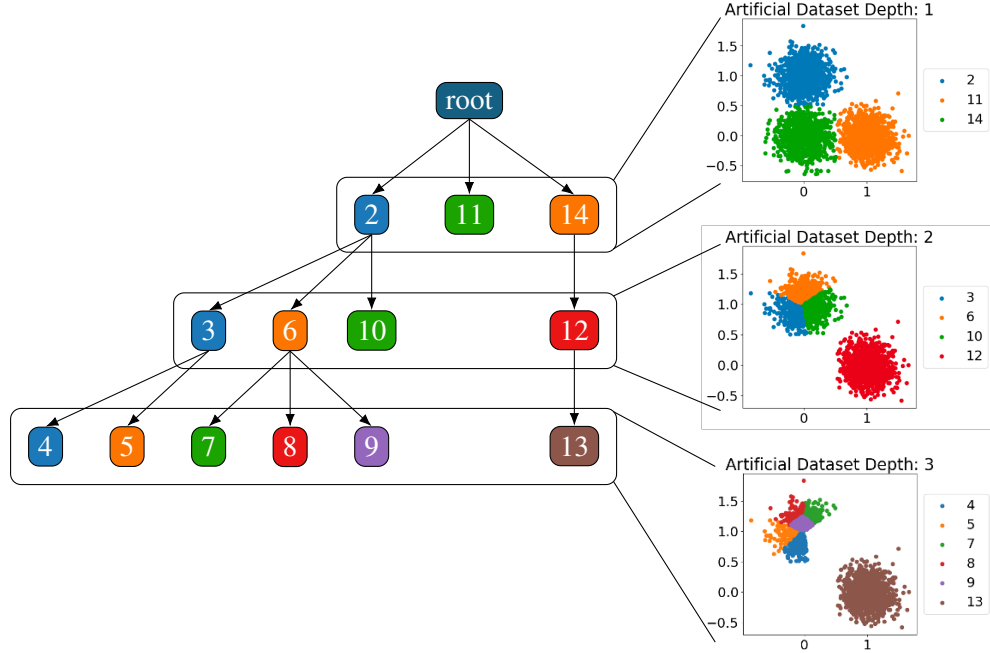
Figure 5.4: The synthetic hierarchical dataset process consists of first generating the super-classes, that is the depth row below the root in this figure. We define a vector size, number of samples and σ which represents a cluster overlap parameter. The samples are then uniformly generated for each class. Building the tree consists of providing the maximum depth and children a parent can have. The parent samples are split based on k-means where k is a random amount of children between 0 and the maximum amount of children. We generate new depths until we have either reached the maximum depth or randomly stop. The tree in this figure consists of one version of a synthetic dataset which 3000 samples, 3 super-classes, a σ of 0.3, maximum depth and children of 3.

While our experiments utilises existing datasets that have some hierarchical labelling, we also developed a synthetic dataset to test the edge cases of the methodology. Having a dataset like ours serves multiple purposes in an experimental framework. Firstly, a synthetic dataset can examine specific aspects of our current experiments' performance. Even through we use GraphSAGE for feature extraction and classifier chains for predictions at each depth level of the tree, uncertainties can arise regarding whether performance issues stem from the model architecture or the structure of the labels. A synthetic dataset enables us to disambiguate these factors via multiple generations of datasets. Secondly, we built a synthetic dataset process that mirrors the hierarchical nature of classification problems, via starting with the generating of super-classes, defining parameters such as shape of input, the amount of samples, and the degree of overlap between class clusters. Changing such parameters allows us to control the

complexity and provide some characteristics to the hierarchical structure.

The samples of our data are then uniformly generated building labelled clusters. We then construct the tree based on the originally generated data with flat labels. We define parameters for the maximum depth and maximum number of children, with branches built randomly within these constraints. For each parent node, we randomly select a number of children and split the data representing that parent node accordingly. This process is repeated for each node at a current depth level before moving down. We continue until we reach the maximum depth level, padding the x_i vector for branches where no children were generated. Our aim with the synthetic dataset experiments is to test varying hierarchical structures with a focus on shallow and dense hierarchies, and create scenarios with different levels of class separability. This is so we can explore areas of the tree that have well-defined labels as well as more ambiguous class boundaries. Figure 5.5 demonstrates a scatter-plot matrix of two parameters in the data generation process. As the rows increase we show different numbers of super-classes while as the columns increase we show different values of degree overlap σ. This value can be between 0 and 1, where 0 has no overlap, showing well-defined classes, while 1 is a complete overlap of the classes, showing extremely ambiguous class boundaries. The colours represent different super-classes in this figure.

5.4 Experimentation

We designed the experiment framework around the different structures of classifier chains, using both image based and synthetic datasets. Most of these would have hierarchical labels with the exception of MNIST. We modified this dataset by building two different sets of trees, referred to as type 1 and type 2 in the results. We use variations of the same basic model, including the graph-based feature extraction so that we can evaluate the proposed approach to classification tasks. We build these models based around the dataset but the structure of the classifier chains are the same across all experiments. In chapter we do not explore changing of the graph during the training process. As they are fixed, they need to be specified before the start of the experiment.

We use categorical entropy as a loss function for each of the classification heads. Different loss normalisation techniques was explored but we ultimately ended designing the experiments with a depth based normalisation. This is because as we move down the tree, our samples become
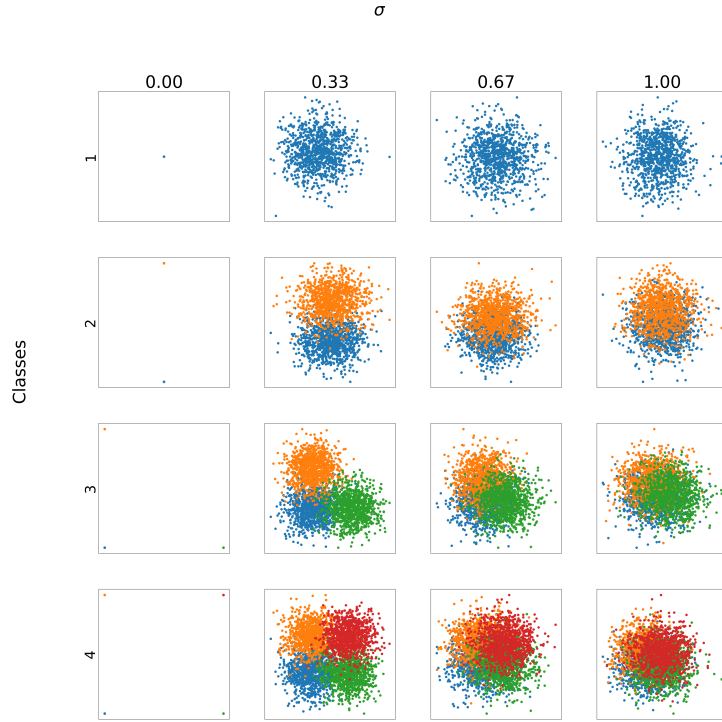
Figure 5.5: We demonstrate two of the parameters of the synthetic data generation process: number of super-classes and degree of overlap σ. The overlap is a value between 0 and 1, where 0 has no overlap and therefore well-defined classes, while 1 is complete overlap of classes, demonstrating extremely ambiguous class boundaries. Each colour is a different class in this figure.

more defined, the information from those representations are more important than that above. This results in a harder training scheme to converge as the model could learn that a branch decision is not as important as below. We trained our models based on an early stopping criteria to ensure that each converged. We use a 5-fold cross validation for each experiment, recording accuracy, recall, precision and F1 scores for each fold, allowing us to see stability in model performance. After each epoch we shuffle our data as well as select a new batch of aggregated nodes with a sample rate of 60% for each node. The Adam optimiser with a learning rate $1 \times e - 2$.

The experiments described where performed in the summer of 2024. Implementation was created in Python version 3.11.9 and the models implemented in TensorFlow version 2.13. Each model was trained on 1 NVIDIA 3070 GPU and a Intel i7 CPU. The following section displays the table of results for each dataset and type of chain used. Note that we are not aiming

for state-of-the-art results in our analysis; rather if we can use explicit labelling structures within the graph deep learning to learn different types relationships among the labels.

In the experiments where we use our synthetic data generation process, we want to explore the difficulty in labels that have data overlapping. This is so we can explore the non-linear separation generalising well under this constrained form of training. To accomplish this, we define two types based on the degree of overlap σ: E for Easy and H for Hard. E has a σ of 0.33 while H is 0.67.

5.5 Results

For each dataset we present a table of results, with the exception for the synthetic datasets. Due to the amount of experiments performed on synthetic data, we split the table up into 3 for each classifier chain. These are tables 5.1, 5.2 and 5.3. Each table describes the performance of the models based on accuracy, recall, precision and f1 scores. Each table also displays some configuration for each experiment. The synthetic dataset experiments display depth of tree, number of super classes and type of tree difficulty used. Difficulty in this case defines how much of an overlap the class clusters should have, easy refers an overlap of 0.2 and hard is a overlap of 0.67, we display these two values in figure 5.5. Table 5.4 displays the results of using MNIST formed with the two types of hierarchical trees. Within each table we separate based on the number of super classes within the trees. Results in bold are the best of that group while the underlined refers to the second best. Table 5.5 displays results on CIFAR-100 and ADE-20K datasets. The ADE experiments form a segmentation task where the leaf nodes are individual pixels and the parent is that group part of an object. As we move up the tree the parts of the object form into a full object where the super classes form all objects within the images.

5.6 Discussion

In our experiments we use relatively small hierarchical trees, approximately between 3 and 10 in depth and width. We based the structure of the classifier chains on the depth level of the tree, even with these small trees computation is still a challenge both in terms of time and space complexity. Scaling to larger trees will only make this problem more challenging. Balancing the depth of GraphSAGE with the width of sampling is crucial. Having many layers

Table 5.1: Results using synthetically generated data with hierarchical labels. The classifier chain of choice in this table is referred as type *A* in this chapter. Each classifier head is independent of each other with no shared features for separating different depth levels of a given tree.

| Depth	Super Class	Type	Accuracy	Recall	Precision	F1
3	3	E	**0.99 ± 0.004**	**0.92 ± 0.05**	0.46 ± 0.03	**0.92 ± 0.05**
3	3	H	0.74 ± 0.002	0.33 ± 0.004	0.36 ± 0.01	0.17 ± 0.003
7	3	E	<u>0.81 ± 0.002</u>	0.50 ± 0.02	**0.73 ± 0.01**	0.3 ± 0.007
7	3	H	0.75 ± 0.01	0.38 ± 0.03	<u>0.56 ± 0.04</u>	0.22 ± 0.016
10	3	E	0.73 ± 0.003	<u>0.51 ± 0.05</u>	0.58 ± 0.04	<u>0.27 ± 0.02</u>
10	3	H	0.69 ± 0.003	0.34 ± 0.0004	0.51 ± 0.07	0.20 ± 0.01
3	5	E	<u>0.78 ± 0.001</u>	<u>0.34 ± 0.002</u>	<u>0.6 ± 0.05</u>	0.21 ± 0.01
3	5	H	**0.79 ± 0.002**	**0.35 ± 0.002**	0.58 ± 0.04	0.22 ± 0.01
7	5	E	0.67 ± 0.0004	0.29 ± 0.01	**0.63 ± 0.07**	**0.39 ± 0.02**
7	5	H	0.63 ± 0.01	0.14 ± 0.01	0.34 ± 0.03	0.20 ± 0.01
10	5	E	0.60 ± 0.02	0.31 ± 0.06	0.39 ± 0.002	<u>0.35 ± 0.017</u>
10	5	H	0.63 ± 0.003	0.16 ± 0.05	0.25 ± 0.01	0.2 ± 0.16
3	7	E	**0.75 ± 0.002**	**0.32 ± 0.001**	**0.63 ± 0.002**	**0.42 ± 0.002**
3	7	H	<u>0.71 ± 0.01</u>	<u>0.26 ± 0.03</u>	<u>0.31 ± 0.01</u>	<u>0.28 ± 0.02</u>
7	7	E	0.64 ± 0.054	0.14 ± 0.02	0.22 ± 0.01	0.17 ± 0.04
7	7	H	0.59 ± 0.01	0.18 ± 0.01	0.23 ± 0.04	0.2 ± 0.03
10	7	E	0.55 ± 0.04	0.19 ± 0.004	0.24 ± 0.003	0.21 ± 0.004
10	7	H	0.59 ± 0.06	0.13 ± 0.01	0.19 ± 0.01	0.15 ± 0.02
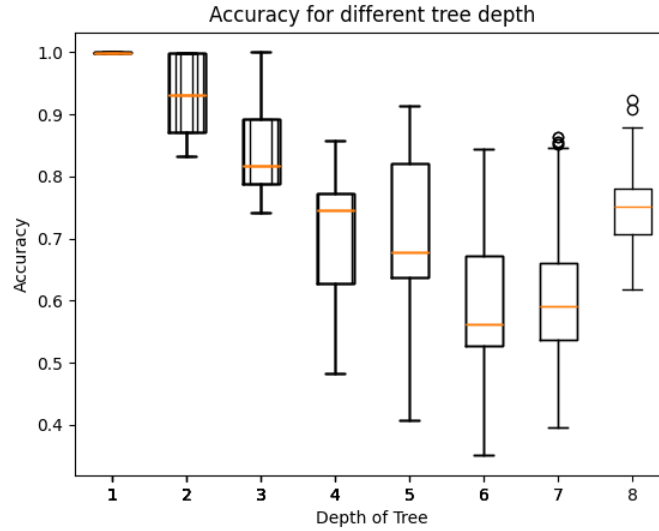
Figure 5.6: Using the synthetic data generator we perform a 10-fold for each increase in depth of the tree. As the depth increases we see an initial decent in accuracy then goes back up, which is similar to the double decent phenomenon seen in deep learning.

111

Table 5.2: Results using synthetically generated data with hierarchical labels. The classifier chain of choice in this table is referred as type *B* in this chapter. We feed in the predictions from a previous chain into a dense layer before going into the next chain. This allows us to utilise learnt features targeted from more general depth layers of the tree

Depth	Super Class	Type	Accuracy	Recall	Precision	F1
3	3	E	0.79 ± 0.025	0.3 ± 0.047	0.32 ± 0.031	0.31 ± 0.039
3	3	H	$\mathbf{0.90 \pm 0.016}$	0.39 ± 0.004	0.43 ± 0.018	0.41 ± 0.01
7	3	E	0.82 ± 0.02	$\mathbf{0.72 \pm 0.01}$	0.7 ± 0.02	$\mathbf{0.71 \pm 0.01}$
7	3	H	$\underline{0.89 \pm 0.014}$	$\underline{0.66 \pm 0.012}$	$\mathbf{0.66 \pm 0.016}$	$\underline{0.66 \pm 0.01}$
10	3	E	0.83 ± 0.021	0.48 ± 0.026	$\underline{0.5 \pm 0.01}$	0.49 ± 0.02
10	3	H	0.71 ± 0.02	0.26 ± 0.018	0.35 ± 0.024	0.3 ± 0.02
3	5	E	$\mathbf{0.89 \pm 0.017}$	$\mathbf{0.61 \pm 0.02}$	$\mathbf{0.64 \pm 0.02}$	$\mathbf{0.63 \pm 0.02}$
3	5	H	$\underline{0.8 \pm 0.04}$	$\underline{0.31 \pm 0.02}$	0.35 ± 0.02	0.33 ± 0.02
7	5	E	0.6 ± 0.03	0.15 ± 0.01	0.25 ± 0.05	0.19 ± 0.02
7	5	H	0.7 ± 0.02	0.27 ± 0.05	0.30 ± 0.04	0.29 ± 0.04
10	5	E	0.60 ± 0.004	0.14 ± 0.005	0.17 ± 0.01	0.15 ± 0.008
10	5	H	0.71 ± 0.02	0.33 ± 0.01	$\underline{0.38 \pm 0.04}$	$\underline{0.35 \pm 0.02}$
3	7	E	$\mathbf{0.93 \pm 0.01}$	$\mathbf{0.53 \pm 0.05}$	$\mathbf{0.56 \pm 0.04}$	$\mathbf{0.55 \pm 0.05}$
3	7	H	$\underline{0.85 \pm 0.02}$	$\underline{0.29 \pm 0.02}$	$\underline{0.31 \pm 0.04}$	$\underline{0.30 \pm 0.03}$
7	7	E	0.59 ± 0.01	0.13 ± 0.01	0.18 ± 0.05	0.15 ± 0.03
7	7	H	0.69 ± 0.01	0.16 ± 0.01	0.29 ± 0.05	0.21 ± 0.02
10	7	E	0.51 ± 0.039	0.16 ± 0.02	0.27 ± 0.39	0.2 ± 0.02
10	7	H	0.56 ± 0.003	0.12 ± 0.01	0.21 ± 0.02	0.16 ± 0.01

results in over-smoothing, while too few does not capture the full depth of the hierarchy. As the depth increase capturing the dependency becomes a problem between distant nodes. Figure 5.6 displays a 10-fold for each instance of depth. We start at a depth of 1 (a flat dataset) and increase the depth on the x axis, displaying the accuracy on the y axis. As depth increase we see a decrease in accuracy but increasing again in a similar way to the double decent phenomenon. This approach is clearly more explicit then that of the double decent appearance seen in deep networks. In these cases as the layers of neural networks increase we often see this phenomenon, we believe that the learnt depths of the tree are matching the implicit training schemes of neural networks.

The synthetic dataset lays the groundwork for future explorations into dataset refinement through active learning, where we can potentially adjust the hierarchical structure on emerging data patterns. While we do not delve into this aspect in this work, its worth noting that such refinements could involve merging classes that show significant overlap or splitting classes that

Table 5.3: Results using synthetically generated data with hierarchical labels. The classifier chain of choice in this table is referred as type *C* in this chapter. We feed in predictions and the embeddings from the previous chain into a dense layer before going into the next chain. This is because the layer from just using the prediction (such as in table 5.2) will result in limiting the representation capacity and even result in a vanishing gradient problems.

Depth	Super Class	Type	Accuracy	Recall	Precision	F1
3	3	E	$\underline{0.81 \pm 0.002}$	0.32 ± 0.03	0.57 ± 0.07	0.41 ± 0.04
3	3	H	$\mathbf{0.86 \pm 0.03}$	0.49 ± 0.03	0.62 ± 0.06	0.55 ± 0.04
7	3	E	0.78 ± 0.004	$\mathbf{0.67 \pm 0.002}$	$\mathbf{0.69 \pm 0.02}$	$\mathbf{0.68 \pm 0.01}$
7	3	H	0.68 ± 0.01	$\underline{0.65 \pm 0.04}$	0.61 ± 0.01	$\underline{0.63 \pm 0.02}$
10	3	E	0.67 ± 0.002	0.25 ± 0.02	0.54 ± 0.05	0.34 ± 0.01
10	3	H	0.67 ± 0.02	0.25 ± 0.01	$\underline{0.63 \pm 0.03}$	0.36 ± 0.01
3	5	E	$\underline{0.78 \pm 0.017}$	$\underline{0.52 \pm 0.05}$	0.6 ± 0.07	$\underline{0.56 \pm 0.06}$
3	5	H	$\mathbf{0.84 \pm 0.03}$	$\mathbf{0.57 \pm 0.02}$	$\mathbf{0.7 \pm 0.02}$	$\mathbf{0.63 \pm 0.02}$
7	5	E	0.72 ± 0.002	0.43 ± 0.01	$\underline{0.67 \pm 0.03}$	0.52 ± 0.02
7	5	H	0.43 ± 0.06	0.18 ± 0.01	0.28 ± 0.05	0.22 ± 0.01
10	5	E	0.69 ± 0.01	0.19 ± 0.02	0.31 ± 0.02	0.24 ± 0.03
10	5	H	0.65 ± 0.02	0.14 ± 0.02	0.29 ± 0.01	0.19 ± 0.01
3	7	E	$\mathbf{0.95 \pm 0.01}$	$\mathbf{0.64 \pm 0.02}$	$\mathbf{0.68 \pm 0.01}$	$\mathbf{0.66 \pm 0.01}$
3	7	H	$\underline{0.86 \pm 0.02}$	$\underline{0.45 \pm 0.01}$	$\underline{0.58 \pm 0.02}$	$\underline{0.51 \pm 0.01}$
7	7	E	0.63 ± 0.04	0.34 ± 0.04	0.49 ± 0.02	0.4 ± 0.01
7	7	H	0.61 ± 0.2	0.36 ± 0.02	0.41 ± 0.01	0.39 ± 0.02
10	7	E	0.57 ± 0.03	0.27 ± 0.01	0.36 ± 0.04	0.31 ± 0.02
10	7	H	0.67 ± 0.01	0.26 ± 0.02	0.37 ± 0.01	0.31 ± 0.02

Table 5.4: This set of results displays experiments for each type of chain and each type of tree used with the MNIST dataset.

Chain Type	Tree Type	Accuracy	Recall	Precision	F1
A	2	0.73 ± 0.02	0.74 ± 0.03	0.35 ± 0.04	0.47 ± 0.04
C	2	0.79 ± 0.034	0.61 ± 0.06	0.37 ± 0.01	0.46 ± 0.02
D	2	0.72 ± 0.1	0.67 ± 0.1	0.32 ± 0.06	0.43 ± 0.07
A	1	$\underline{0.80 \pm 0.01}$	$\mathbf{0.77 \pm 0.06}$	0.67 ± 0.02	0.73 ± 0.01
C	1	$\mathbf{0.84 \pm 0.01}$	0.75 ± 0.01	$\mathbf{0.75 \pm 0.002}$	$\mathbf{0.75 \pm 0.004}$
D	1	$\underline{0.80 \pm 0.06}$	$\underline{0.76 \pm 0.04}$	$\underline{0.73 \pm 0.03}$	$\underline{0.74 \pm 0.03}$

Table 5.5: Two sets of expriemnts where one is a classification task with CIFAR-100 while the other is a segmentation task with ADE-20K. The segmentation tasks froms a part-of relationship while classification is a semantic relationship based on domain knowledge provided by a human.

Dataset	Chain Type	Accuracy	Recall	Precision	F1
CIFAR-100	A	0.46 ± 0.11	0.52 ± 0.09	0.46 ± 0.11	0.24 ± 0.05
CIFAR-100	C	0.37 ± 0.03	0.39 ± 0.03	0.37 ± 0.03	0.19 ± 0.01
CIFAR-100	D	**0.68 ± 0.29**	**0.71 ± 0.28**	**0.68 ± 0.29**	**0.35 ± 0.14**
ADE-20K	A	0.43 ± 0.34	0.49 ± 0.21	0.57 ± 0.29	0.53 ± 0.25
ADE-20K	C	0.31 ± 0.24	0.27 ± 0.39	0.37 ± 0.21	0.31 ± 0.27
ADE-20K	D	**0.69 ± 0.21**	**0.7 ± 0.23**	**0.65 ± 0.19**	**0.67 ± 0.18**

demonstrate clear sub-clusters.

5.7 Summary

Hierarchical labels form a relationship in both an abstraction and fine-grained level of the data. We represented different depth levels of these labels that form a type of granularity in a tree structure. This form of supervision is highly inspired by human cognition and perception where we recognise patterns of various levels of abstraction to define an object. Many machine learning approaches attempt to mimic this process with different layers of a neural network. We explore providing more explicit and richer forms of supervision in two ways. The first is a break up of the physical or geometric structure of the object, referred to as encapsulation relationships (or part off relationships). The second is sub-classification relationships which are semantic relation of labels provided by domain knowledge of what we are trying to capture in the dataset. Capturing this information in a more explicit form of deep learning is especially useful in tasks which have complex and interconnected characteristics such as in manufacturing of composite materials or comorbidity analysis of patients in healthcare domains. This work utilises these two types of relationships to solve both classification and segmentation tasks. We base our approaches on graph deep learning as a form of feature extraction on the tree of nodes where each node is a hierarchical label and the features form samples in the dataset. The root of a tree represents the full dataset while a leaf node is a individual sample or pixel depending on the task, then the interconnecting nodes form the hierarchical relationship. To form our predictions we explore different types of classifier chains. Different structures of these chains

defines the flow of information between classifiers. Each classifier targets a different depth level of the tree. Predictions from previous classifiers are fed into subsequent ones allowing us to capture different depths of the hierarchy and use that information in making a prediction. We evaluated our approach synthetic and real-world datasets and also explored the complexity of increasing the tree in depth and in width. All our experiments involve a 5-fold split of the data, displaying the level of stability and confidence in each performance metric.

Chapter 6

Conclusions and Future Work

Contents

6.1   Conclusions

We started off building this work by setting out 3 goals with input from both external and internal stakeholders. The design of the doctoral project had an application focus of quality control in the domain of steel manufacturing. The first goal was the improved detection, localisation and classification of features observed by imaging systems. In this thesis we present a refinement strategy that converts bounding-box datasets to dense segmentation. There are many common mistakes in these bounding-box datasets such as more than one defect is within the ROI, some defects do not have an ROI, and defects can also leak out of the ROI. Due to the geometric structure of defects many of the pixels within the ROI are actually negative. By providing richer forms of supervision to deep learning models we can gain performance in detection, localisation and classification. The second goal was the improved labelling and analysis of complex classes via a semi-supervised approach. While our refinement strategy is related we also utilise a form of complex classes by using hierarchical labelling system. We model this by presenting a network architecture via a graph-based deep learning approach. The leaf nodes of the structure represent individual samples, the root node represents the full dataset and the interconnecting nodes form the hierarchy. Nodes that contain more than one sample are aggregation of their children. As a result the nodes further up the hierarchical tree are more generalised labels and as we move down they become more specialised. These node embeddings then feed into one of three types of classifier chains which target different depth levels. We structure the different classifier chains based on the amount of information flow between these depth levels. As part of this work we utilise two types of hierarchical relationships to solve classification and segmentation tasks. The first is a break up of the physical or geometric structure of the object, called encapsulation relationships. The second is sub-classification relationships which are semantic relations of labels provided by domain knowledge of what we are trying to capture in the dataset. The third goal is the improved integration and use of data visualisation within a user-guided approach to improve understanding of the model inference. Throughout this thesis we utilise different forms of supervision to allow for stronger signals of inductive bias. Goal three refers to the use of applying this inductive bias towards the training scheme where we go in the direction of active learning. We present different acquisition functions for embedding expert knowledge into the training with the help of a human. In order for someone to make an accurate design we provide varying degrees of exploratory approaches with graphical user interfaces. Experts are able to customise these interfaces, select different algorithms for visualisation and see how their feedback into the model has an influence on the

118

task at hand.

6.2   Contributions

The contributions of this thesis are the following:

An acquisition function based on current feature representation positions. We present a new acquisition function for finding a set of samples within the dataset that gets labelled by an expert will result in the most informative update to the model within an active learning setting. This utilises the current embedding space of generative models and the triplet loss. We use mining strategies based on an anchor, a sample with the sample label as the anchor and a negative which is close to the anchor. The mining strategies are based on the distance between samples. We request the help of an expert to relabel or reinforce correct labels of negatives, which focuses training to create dense clusters of related samples.

Refinement strategy for fuzzy-labelled datasets. We present a refinement strategy within an active learning setting to fix mistakes in bounding-box labelled datasets. By uniformly sampling pixels to form patches of images, we then mine this pool to get a set of the most informative ones that would better improve the classification and generative models. We use the classification head of these models to predict a dense segmentation overtime.

Incorporating explicit domain knowledge into a data-driven approach via a hierarchical labelling system. We present a network architecture to model hierarchical labelled datasets. These hierarchical labels are modelled via a graph-based deep learning approach where the leaf nodes are individual samples and the root is the full dataset. Interconnecting nodes are the aggregation of their children which forms a hierarchical relationship. As a result the nodes further up the hierarchical structure are more generalised labels and as we move down they become more specialised. These node embeddings then feed into a one of three types of classifier chains which target different depth levels. Previous classifier heads are used to inform new predictions in subsequent classifiers forming a chain of information flow. We evaluate our approach with a 5-fold of MNIST, CIFAR-100, ADE-20k datasets. We also build a synthetic data generator for building hierarchical datasets so that we can test edge cases of our methodology. This work also utilises two types of hierarchical relationships to solve classification and segmentation tasks. The first is a break up of the physical or geometric structure of the object,

referred to as encapsulation relationships. The second is sub-classification relationships which are semantic relations of labels provided by domain knowledge of what we are trying to capture in the dataset.

Detecting label collisions during the training process. We explore the use of density-based deep clustering where it forms a graph. Each node represents a sample and the edges form the clusters. As clusters of samples merge this forms a collision to which we reform the labelling system. Clusters builds a hierarchical dataset where if they do form then this becomes a parent node of the two children which are colliding. If a cluster starts to separate then this forms a set of children where the cluster is the parent instead.

An acquisition function for evolving graphs. This contribution expands on the density based deep clustering approach by applying an active training scheme. This is where each node of the graph is a sample and the edges form the clusters. If clusters of nodes start to merge during the training process we request an expert to inform the model if the clusters should join together or not. If that merging does happen then this forms a hierarchical set of labels as the joining clusters form a single node while its children will represent the two clusters.

6.3 Future Work

Traditional active learning methods struggle in a few ways when being used within an deep network, this is because as we scale to higher dimensional spaces selecting the most informative sample becomes an issue. Selecting samples in general is a challenge due to the uncertainly estimation on the predictions and the changing internal representations. Another challenge is that although active learning reduces the need for high-quality labelled data, there is still a need for it. This is because deep learning is often very greedy for the data. Another alignment issue with both areas is that most active learning algorithms focus on training of classifiers by using a query strategy on fixed representations while deep learning the representations and the classifiers are optimised in a joint training process. Inspired by the work on refinement learning with human feedback, we could use more complex feedback from the human and then train a reward scheme to improve the smoothness of the optimisation task. This feedback can be one or more continuous scales. This feedback would feed into a dense layer allowing a more direct information flow of tuning the network. Generally richer forms of feedback have shown to improve performance in many tasks, which stabilises the acquisition function. Active learn-

ing has shown significant potential in reducing the need for high-quality data but can also be used for other tasks such as data refinement and providing insight as a form of interoperability. Measuring the affect that feedback has on the training or even how some samples have greater affect could be explored more. This is especially useful in online machine learning systems or simply deployed models, as we can use acquisition functions to find samples that are being forgotten over others that are not. The application of such work involve around exploring the affects of drifting in machine learning models.

Another significant proportion of this thesis was exploring the use of more explicit training schemes via the structure of the labelling within datatsets, creating a form of constraint optimisation technique within deep learning. Stability becomes more of a problem the greater the constraint we make on the optimisation as shown in chapter 5. Careful discussion needs to be made on network architecture, the task, the loss function and how we present the data to the model. Active learning can stabilise this issue but forming the training scheme seems more desirable as it then becomes fully automated in an end-to-end system. Work on including attention mechanisms into the hierarchy is a promising direction to stabilise the training as we then focus at different depth levels that are more important than others. The design needs to be carefully considered as I suspect the attention would ultimately focus more on the root of the structure as this includes all data in the dataset under a single label. The work we have presented in thesis could also be used as knowledge graph which is queried under a large language model. In applications like robotic planning, having varying forms of symbolic AI can help in decision making, such as informing the model that a painting belongs on a wall or a cooker belongs in a kitchen. Hierarchical information like this forms itself in many applications and domains and trying to capture that remains a challenge. In graph deep learning we structure our data in an irregular way but our embeddings are still trained within euclidean space. Exploring the use of different spaces that more explicitly take advantage of hierarchical nature is preferred in this context. Hyperbolic deep learning is one direction we could explore. Hyperbolic representations form a distance that is almost exponential as we move up the tree structure while sibling nodes are close together. This allows use to capture these relationships with fewer dimensions but it is quite computationally expensive and applying to graphs which in itself is already expensive is a complex challenge.

Bibliography

[1] Ali Alqahtani, X. Xie, J. Deng, and Mark Jones. "Learning Discriminatory Deep Clustering Models". In: *International Conference of Computer Analysis of Images and Patterns (CAIP)* (Aug. 2019), pp. 224–233. DOI: `10.1007/978-3-030-29888-3_18`.

[2] Ali Alqahtani, Xianghua Xie, Jingjing Deng, and Mark W. Jones. "A Deep Convolutional Auto-Encoder with Embedded Clustering". In: *2018 25th IEEE International Conference on Image Processing (ICIP)* (2018), pp. 4058–4062.

[3] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. "NetVLAD: CNN architecture for weakly supervised place recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5297–5307.

[4] Relja Arandjelovic and Andrew Zisserman. "All About VLAD". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2013.

[5] Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer Normalization". In: *ArXiv* abs/1607.06450 (2016).

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2014).

[7] David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. "The Shattered Gradients Problem: If resnets are the answer, then what is the question?" In: *Proceedings of the 34th International Conference on Machine Learning*. Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 342–350.

[8]   Sugato Basu, Arindam Banerjee, and Raymond J. Mooney. "Semi-supervised Cluster-ing by Seeding". In: *Proceedings of the Nineteenth International Conference on Machine Learning*. ICML '02. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2002, pp. 27–34. ISBN: 1558608737.

[9]   Peter Battaglia, Jessica Blake Chandler Hamrick, Victor Bapst, Alvaro Sanchez, Vini-cius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andy Ballard, Justin Gilmer, George E. Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Jayne Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. "Relational inductive biases, deep learning, and graph networks". In: *arXiv* (2018).

[10]   Javad Zolfaghari Bengar, Bogdan Raducanu, and Joost van de Weijer. "When Deep Learners Change Their Mind: Learning Dynamics for Active Learning". In: *Computer Analysis of Images and Patterns*. Ed. by Nicolas Tsapatsoulis, Andreas Panayides, Theo Theocharides, Andreas Lanitis, Constantinos Pattichis, and Mario Vento. Springer International Publishing, 2021, pp. 403–413. ISBN: 978-3-030-89128-2.

[11]   Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 35.8 (Aug. 2013), pp. 1798–1828. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2013.50.

[12]   Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. "Representation Learning: A Review and New Perspectives". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2012), pp. 1798–1828.

[13]   Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, 2006. ISBN: 0387310738.

[14]   Christopher M. Bishop and Hugh Bishop. *Deep Learning: Foundations and Concepts*. Springer, 2024.

[15]   Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. "Weight uncertainty in neural network". In: *International Conference on Machine Learning*. 2015, pp. 1613–1622.

[16] Léon Bottou. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *Proceedings of COMPSTAT'2010*. Physica-Verlag HD, 2010, pp. 177–186. ISBN: 978-3-7908-2604-3.

[17] H. Bourlard and Y. Kamp. "Auto-association by multilayer perceptrons and singular value decomposition". In: *Biological Cybernetics* 59.4 (1988), pp. 291–294. DOI: `10.1007/BF00332918`.

[18] Steve Branson, Grant Van Horn, Catherine Wah, Pietro Perona, and Serge Belongie. "The Ignorant Led by the Blind: A Hybrid Human–Machine Vision System for Fine-Grained Categorization". In: *International Journal of Computer Vision* (Jan. 1, 2014).

[19] Samuel Budd, Emma C Robinson, and Bernhard Kainz. "A survey on active learning and human-in-the-loop deep learning for medical image analysis". In: *Medical Image Analysis* 71 (2021), p. 102062.

[20] A. Canziani, Adam Paszke, and Eugenio Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications". In: *ArXiv* abs/1605.07678 (2016).

[21] Stuart Card, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision To Think*. Academic Press, Jan. 1999. ISBN: 978-1-55860-533-6.

[22] Young-Jin Cha, Wooram Choi, and Oral Büyüköztürk. "Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks". In: *Computer-Aided Civil and Infrastructure Engineering* 32.5 (2017), pp. 361–378. DOI: `https://doi.org/10.1111/mice.12263`.

[23] Young-Jin Cha, Wooram Choi, Gahyun Suh, Sadegh Mahmoudkhani, and Oral Büyüköztürk. "Autonomous Structural Visual Inspection Using Region-Based Deep Learning for Detecting Multiple Damage Types". In: *Computer-Aided Civil and Infrastructure Engineering* 33.9 (2018), pp. 731–747. DOI: `https://doi.org/10.1111/mice.12334`.

[24] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. "ShapeNet: An Information-Rich 3D Model Repository". In: *CoRR* abs/1512.03012 (2015).

[25] Zhao-Min Chen, Xiu-Shen Wei, Peng Wang, and Yanwen Guo. "Multi-Label Image Recognition With Graph Convolutional Networks". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 5172–5181. DOI: 10. 1109/CVPR.2019.00532.

[26] Weiwei Cheng, Eyke Hüllermeier, and Krzysztof J Dembczynski. "Bayes optimal multilabel classification via probabilistic classifier chains". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 279–286.

[27] Doo-chul Choi, Yong-Ju Jeon, Seung Hun Kim, Seokbae Moon, Jong Pil Yun, and Sang Woo Kim. "Detection of pinholes in steel slabs using Gabor filter combination and morphological features". In: *Isij International* 57.6 (2017), pp. 1045–1053.

[28] Amanda Clare and Ross D. King. "Knowledge Discovery in Multi-label Phenotype Data". In: *European Conference on Principles of Data Mining and Knowledge Discovery*. 2001.

[29] Connor Clarkson, Michael Edwards, and Xianghua Xie. "Active Anchors". In: *Companion Proceedings of the 2023 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. EICS '23 Companion. Swansea, United Kingdom: Association for Computing Machinery, 2023, pp. 68–69. ISBN: 9798400702068. DOI: 10.1145/ 3596454.3597185.

[30] Connor Clarkson, Michael Edwards, and Xianghua Xie. "Active Anchors: Similarity Based Refinement Learning". In: *Proceedings of the International Conference on Applied Computing*. 2023, pp. 47–57. ISBN: 978-989-8704-53-5.

[31] Connor Clarkson, Michael Edwards, and Xianghua Xie. "Dense Semantic Refinement Using Active Similarity Learning". In: *IADIS International Journal on Computer Science and Information Systems*. 2024, pp. 15–30. ISBN: 1646-3692.

[32] Connor Clarkson, Michael Edwards, and Xianghua Xie. "Modelling on Types of Hierarchical Relationships". (To be published).

[33] D. Cohn, R. Caruana, and A. McCallum. "Semi-supervised clustering with user feedback". In: *Constrained Clustering: Advances in Algorithms, Theory, and Applications* (Jan. 2008), pp. 17–31.

[34] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: `10.1109/CVPR.2009.5206848`.

[35] Xingping Dong and Jianbing Shen. "Triplet loss in siamese network for object tracking". In: *European Conference on Computer Vision*. 2018, pp. 459–474.

[36] Finale Doshi-Velez and Been Kim. "Towards A Rigorous Science of Interpretable Machine Learning". In: *arXiv: Machine Learning* (2017).

[37] Michael Edwards, Jingjing Deng, and Xianghua Xie. "Labeling subtle conversational interactions within the CONVERSE dataset". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. 2017, pp. 140–145. DOI: `10.1109/PERCOMW.2017.7917547`.

[38] Michael Edwards and Xianghua Xie. "Graph Based Convolutional Neural Network". In: *ArXiv* abs/1609.08965 (2016).

[39] André Elisseeff and Jason Weston. "A kernel method for multi-labelled classification". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press, 2001.

[40] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.

[41] Yarin Gal and Zoubin Ghahramani. "Bayesian Convolutional Neural Networks with Bernoulli Approximate Variational Inference". In: *ArXiv* abs/1506.02158 (2015).

[42] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. "Deep Bayesian active learning with image data". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. JMLR.org, 2017, pp. 1183–1192.

[43] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. "Deep bayesian active learning with image data". In: *International Conference on Machine Learning*. 2017, pp. 1183–1192.

[44] Stavros Georgousis, Michael P. Kenning, and Xianghua Xie. "Graph Deep Learning: State of the Art and Challenges". In: *IEEE Access* 9 (2021), pp. 22106–22140. DOI: `10.1109/ACCESS.2021.3055280`.

[45] Charles D. Gilbert and Wu Li. "Top-down influences on visual processing". In: *Nature Reviews Neuroscience* 14.5 (2013), pp. 350–363. DOI: `10.1038/nrn3476`.

[46] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. "Neural message passing for Quantum chemistry". In: *Proceedings of the 34th International Conference on Machine Learning - Volume 70*. ICML'17. JMLR.org, 2017, pp. 1263–1272.

[47] Ross Girshick. "Fast R-CNN". In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 1440–1448. DOI: `10.1109/ICCV.2015.169`.

[48] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 580–587. DOI: `10.1109/CVPR.2014.81`.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[50] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger. Vol. 27. Curran Associates, Inc., 2014.

[51] Xifeng Guo, Xinwang Liu, En Zhu, and Jianping Yin. "Deep Clustering with Convolutional Autoencoders". In: *International Conference on Neural Information Processing*. 2017.

[52] Will Hamilton, Zhitao Ying, and Jure Leskovec. "Inductive Representation Learning on Large Graphs". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[53] William L. Hamilton, Rex Ying, and Jure Leskovec. "Inductive representation learning on large graphs". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 1025–1035.

[54] William L. Hamilton, Rex Ying, and Jure Leskovec. "Representation Learning on Graphs: Methods and Applications". In: *IEEE Data Eng. Bull.* 40 (2017), pp. 52–74.

[55] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004.

[56] T. Hastie, R. Tibshirani, and J.H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer series in statistics. Springer, 2009. ISBN: 9780387848846.

[57] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. June 2016.

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1904–1916. DOI: `10.1109/TPAMI.2015.2389824`.

[59] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification". In: *CoRR* abs/1703.07737 (2017).

[60] Geoffrey Hinton. "Some demonstrations of the effects of structural descriptions in mental imagery". In: *Cognitive Science* 3.3 (1979), pp. 231–250. ISSN: 0364-0213. DOI: `https://doi.org/10.1016/S0364-0213(79)80008-7`.

[61] Geoffrey Hinton and Sam Roweis. "Stochastic neighbor embedding". In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. NIPS'02. Cambridge, MA, USA: MIT Press, 2002, pp. 857–864.

[62] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *ArXiv* abs/1207.0580 (2012).

[63] Peihao Huang, Yan Huang, Wei Wang, and Liang Wang. "Deep Embedding Network for Clustering". In: *2014 22nd International Conference on Pattern Recognition* (2014), pp. 1532–1537.

[64] Yujun Huang, Yunpeng Weng, Shuai Yu, and Xu Chen. "Diffusion Convolutional Recurrent Neural Network with Rank Influence Learning for Traffic Forecasting". In: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, pp. 678–685. DOI: `10.1109/TrustCom/BigDataSE.2019.00096`.

[65]   Sergey Ioffe and Christian Szegedy. "Batch normalization: accelerating deep network training by reducing internal covariate shift". In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. ICML'15. Lille, France: JMLR.org, 2015, pp. 448–456.

[66]   Fredrik D. Johansson, Uri Shalit, and David Sontag. "Learning representations for counterfactual inference". In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. ICML'16. New York, NY, USA: JMLR.org, 2016, pp. 3020–3029.

[67]   Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-Scale Similarity Search with GPUs". In: *IEEE Transactions on Big Data* 7.3 (2021), pp. 535–547. DOI: `10.1109/TBDATA.2019.2921572`.

[68]   Leonard Kaufman and Peter Rousseeuw. *Finding Groups in Data: An Introduction To Cluster Analysis*. Wiley, Jan. 1990. ISBN: 0-471-87876-6. DOI: `10.2307/2532178`.

[69]   Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. "Representation Learning for Dynamic Graphs: A Survey". In: *Journal of Machine Learning Research* 21.70 (2020), pp. 1–73.

[70]   Filip Ekström Kelvinius, Dimitar Georgiev, Artur Toshev, and Johannes Gasteiger. "Accelerating Molecular Graph Neural Networks via Knowledge Distillation". In: *The Second Learning on Graphs Conference*. 2023.

[71]   Mahdi Khodayar and Jianhui Wang. "Spatio-Temporal Graph Deep Neural Network for Short-Term Wind Speed Forecasting". In: *IEEE Transactions on Sustainable Energy* 10.2 (2019), pp. 670–681. DOI: `10.1109/TSTE.2018.2844102`.

[72]   A. Krizhevsky and G. Hinton. "Learning multiple layers of features from tiny images". In: *Master's thesis, Department of Computer Science, University of Toronto* (2009).

[73]   Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012.

[74]   Norbert Krüger, Peter Janssen, Sinan Kalkan, Markus Lappe, Alevs. Leonardis, Justus H. Piater, Antonio Jose Rodríguez-Sánchez, and Laurenz Wiskott. "Deep Hierarchies in the Primate Visual Cortex: What Can We Learn for Computer Vision?" In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35 (2013), pp. 1847–1871.

[75] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[76] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[77] Yann LeCun. "The MNIST database of handwritten digits". In: *http://yann. lecun. com/exdb/mnist/* (1998).

[78] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[79] Levi Lelis and Jörg Sander. "Semi-supervised Density-Based Clustering". In: *2009 Ninth IEEE International Conference on Data Mining*. 2009, pp. 842–847. DOI: 10.1109/ICDM.2009.143.

[80] Fengfu Li, Hong Qiao, and Bo Zhang. "Discriminatively boosted image clustering with fully convolutional auto-encoders". In: *Pattern Recognition* 83 (2018), pp. 161–173. ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2018.05.019.

[81] Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. "DeepGCNs: Can GCNs Go As Deep As CNNs?" In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 9266–9275. DOI: 10.1109/ICCV.2019.00936.

[82] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. "Visualizing the loss landscape of neural nets". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Curran Associates Inc., 2018, pp. 6391–6401.

[83] Jiangyun Li, Zhenfeng Su, Jiahui Geng, and Yixin Yin. "Real-time Detection of Steel Strip Surface Defects Based on Improved YOLO Detection Network". In: *IFAC Workshop on Mining, Mineral and Metal Processing* 51.21 (2018), pp. 76–81. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2018.09.412.

[84] Youwei Liang, Junfeng He, Gang Li, Peizhao Li, Arseniy Klimovskiy, Nicholas Carolan, Jiao Sun, Jordi Pont-Tuset, Sarah Young, Feng Yang, Junjie Ke, Krishnamurthy Dj Dvijotham, Katherine M. Collins, Yiwen Luo, Yang Li, Kai J Kohlhoff, Deepak

Ramachandran, and Vidhya Navalpakkam. "Rich Human Feedback for Text-to-Image Generation". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2024, pp. 19401–19411.

[85] Maxwell W. Libbrecht and William Stafford Noble. "Machine learning applications in genetics and genomics". In: *Nature Reviews Genetics* 16.6 (2015), pp. 321–332. DOI: `10.1038/nrg3920`.

[86] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2999–3007. DOI: `10.1109/ICCV.2017.324`.

[87] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollár. "Focal Loss for Dense Object Detection". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 2999–3007.

[88] Grace W. Lindsay. "Attention in Psychology, Neuroscience, and Machine Learning". In: *Frontiers in Computational Neuroscience* 14 (2020). ISSN: 1662-5188. DOI: `10.3389/fncom.2020.00029`.

[89] Yajiao Liu, Jiang Wang, Haitao Yu, Fulong Li, Lifeng Yu, and Chunhui Zhang. "Surface Defect Detection of Steel Products Based on Improved YOLOv5". In: *2022 41st Chinese Control Conference (CCC)*. IEEE. 2022, pp. 5794–5799.

[90] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *2015 IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 3431–3440. DOI: `10.1109/CVPR.2015.7298965`.

[91] Qiwu Luo, Xiaoxin Fang, Li Liu, Chunhua Yang, and Yichuang Sun. "Automated visual defect detection for flat steel surface: A survey". In: *IEEE Transactions on Instrumentation and Measurement* 69.3 (2020), pp. 626–644.

[92] J. MacQueen. "Some methods for classification and analysis of multivariate observations". In: *Proceedings of the Berkeley Symposium on Mathematical Statistics and Probability*. 1967.

[93] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133. DOI: `10.1007/BF02478259`.

[94] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Großberger. "UMAP: Uniform Manifold Approximation and Projection". In: *Journal of Open Source Software* 3.29 (2018), p. 861. DOI: 10.21105/joss.00861.

[95] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. Cambridge, MA, USA: MIT Press, 1969.

[96] Tom M. Mitchell. *The Need for Biases in Learning Generalizations*. Tech. rep. New Brunswick, NJ: Rutgers University, 1980.

[97] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael Bronstein. "Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. July 2017, pp. 5425–5434. DOI: 10.1109/CVPR.2017.576.

[98] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. *Deep Double Descent: Where Bigger Models and More Data Hurt*. 2019. arXiv: 1912.02292.

[99] Nirbhar Neogi, Dusmanta K Mohanta, and Pranab K Dutta. "Review of vision-based steel surface inspection systems". In: *EURASIP Journal on Image and Video Processing* 2014.1 (2014), pp. 1–19.

[100] M.-E. Nilsback and A. Zisserman. "A Visual Vocabulary for Flower Classification". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. 2006, pp. 1447–1454. DOI: 10.1109/CVPR.2006.42.

[101] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. "Learning Deconvolution Network for Semantic Segmentation". In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1520–1528.

[102] Kenta Oono and Taiji Suzuki. "Optimization and Generalization Analysis of Transduction through Gradient Boosting and Application to Multi-scale Graph Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., 2020, pp. 18917–18930.

[103] Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. "Deep Learning for Anomaly Detection: A Review". In: *ACM Comput. Surv.* 54.2 (Mar. 2021). ISSN: 0360-0300. DOI: 10.1145/3439950.

[104] Omkar M. Parkhi, Andrea Vedaldi, Andrew Zisserman, and C. V. Jawahar. "Cats and dogs". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), pp. 3498–3505.

[105] W. Pedrycz and J. Waletzky. "Fuzzy clustering with partial supervision". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 27.5 (1997), pp. 787–795. DOI: 10.1109/3477.623232.

[106] Alethea Power, Yuri Burda, Harrison Edwards, Igor Babuschkin, and Vedant Misra. "Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets". In: *1st Mathematical Reasoning in General Artificial Intelligence Workshop, ICLR 2021* abs/2201.02177 (2022).

[107] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. "Classifier chains for multi-label classification". In: *Machine Learning* 85.3 (2011), pp. 333–359. DOI: 10.1007/s10994-011-5256-5.

[108] Joseph Redmon. "Yolov3: An incremental improvement". In: *arXiv* (2018).

[109] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

[110] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[111] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (2017), pp. 1137–1149. DOI: 10.1109/TPAMI.2016.2577031.

[112] Pedro O. C. S. Ribeiro, Matheus M. dos Santos, Paulo L. J. Drews, Silvia S. C. Botelho, Lucas M. Longaray, Giovanni G. Giacomo, and Marcelo R. Pias. "Underwater Place Recognition in Unknown Environments with Triplet Based Acoustic Image Retrieval". In: *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2018, pp. 524–529. DOI: 10.1109/ICMLA.2018.00084.

[113] Yu Rong, Wenbing Huang, Tingyang Xu, and Junzhou Huang. "DropEdge: Towards Deep Graph Convolutional Networks on Node Classification". In: *International Conference on Learning Representations*. 2019.

[114] 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*. Ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi. Springer International Publishing, 2015, pp. 234–241.

[115] Frank Rosenblatt. "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6 (1958), pp. 386–408.

[116] Cynthia Rudin. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead". In: *Nature Machine Intelligence* 1.5 (2019), pp. 206–215. DOI: `10.1038/s42256-019-0048-x`.

[117] Laura von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Michal Walczak, Julius Pfrommer, Annika Pick, Rajkumar Ramamurthy, Jochen Garcke, Christian Bauckhage, and Jannis Schuecker. "Informed Machine Learning - A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems". In: *IEEE Transactions on Knowledge and Data Engineering* PP (May 2021), pp. 1–1. DOI: `10.1109/TKDE.2021.3079836`.

[118] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "Facenet: A unified embedding for face recognition and clustering". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2015, pp. 815–823. DOI: `10.1109/CVPR.2015.7298682`.

[119] Ozan Sener and Silvio Savarese. "Active learning for convolutional neural networks: A core-set approach". In: *arXiv preprint arXiv:1708.00489* (2017).

[120] Burr Settles, Mark Craven, and Soumya Ray. "Multiple-Instance Active Learning". In: *Advances in Neural Information Processing Systems*. Ed. by J. Platt, D. Koller, Y. Singer, and S. Roweis. Vol. 20. Curran Associates, Inc., 2007.

[121] David I Shuman, Sunil K. Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". In: *IEEE Signal Processing Magazine* 30.3 (2013), pp. 83–98. DOI: `10.1109/MSP.2012.2235192`.

[122] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

[123] Samarth Sinha, Sayna Ebrahimi, and Trevor Darrell. "Variational Adversarial Active Learning". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019), pp. 5971–5980.

[124] Chunfeng Song, Feng Liu, Yongzhen Huang, Liang Wang, and Tieniu Tan. "Autoencoder Based Data Clustering". In: *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 117–124. ISBN: 978-3-642-41822-8.

[125] Limei Song, Wenwei Lin, Yan-Gang Yang, Xinjun Zhu, Qinghua Guo, and Jiangtao Xi. "Weak Micro-Scratch Detection Based on Deep Convolutional Neural Network". In: *IEEE Access* 7 (2019), pp. 27547–27554. DOI: `10.1109/ACCESS.2019.2894863`.

[126] Xiaohong Sun, Jinan Gu, Shixi Tang, and Jing Li. "Research Progress of Visual Inspection Technology of Steel Products—A Review". In: *Applied Sciences* 8 (Nov. 2018), p. 2195. DOI: `10.3390/app8112195`.

[127] Richard Szeliski. *Computer Vision : Algorithms and Applications*. eng. 2nd ed. 2022. Texts in Computer Science. Springer International Publishing, 2022. ISBN: 9783030343729.

[128] Fei Tian, Bin Gao, Qing Cui, Enhong Chen, and Tie-Yan Liu. "Learning deep representations for graph clustering". In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*. AAAI'14. Québec City, Québec, Canada: AAAI Press, 2014, pp. 1293–1299.

[129] Giorgos Tolias, Tomas Jenicek, and Ondřej Chum. "Learning and Aggregating Deep Local Descriptors for Instance-Level Recognition". In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm. Springer International Publishing, 2020, pp. 460–477. ISBN: 978-3-030-58452-8.

[130] Mariya Toneva, Alessandro Sordoni, Rémi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J. Gordon. "An Empirical Study of Example Forgetting during Deep Neural Network Learning". In: *ArXiv* abs/1812.05159 (2018).

[131] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. "Selective Search for Object Recognition". In: *International Journal of Computer Vision* 104.2 (2013), pp. 154–171. DOI: `10.1007/s11263-013-0620-5`.

[132] Asad Ullah, Hongmei Xie, Muhammad Omer Farooq, and Zhaoyun Sun. "Pedestrian detection in infrared images using fast RCNN". In: *2018 Eighth International Conference on Image Processing Theory, Tools and Applications (IPTA)*. IEEE. 2018, pp. 1–6.

[133] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. "The iNaturalist Species Classification and Detection Dataset". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 8769–8778. DOI: `10.1109/CVPR.2018.00914`.

[134] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc., 2017.

[135] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. "Graph Attention Networks". In: *6th International Conference on Learning Representations* (2017).

[136] Alexander Vezhnevets, Joachim M. Buhmann, and Vittorio Ferrari. "Active learning for semantic segmentation with expected change". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3162–3169. DOI: `10.1109/CVPR.2012.6248050`.

[137] Pascal Vincent, Hugo Larochelle, Y. Bengio, and Pierre-Antoine Manzagol. "Extracting and composing robust features with denoising autoencoders". In: *Proceedings of the 25th International Conference on Machine Learning*. Jan. 2008, pp. 1096–1103. DOI: `10.1145/1390156.1390294`.

[138] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. "Constrained K-means Clustering with Background Knowledge". In: *Proceedings of the Eighteenth International Conference on Machine Learning*. ICML '01. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001, pp. 577–584. ISBN: 1558607781.

[139] Dan Wang and Yi Shang. "A new active labeling method for deep learning". In: *2014 International Joint Conference on Neural Networks (IJCNN)*. 2014, pp. 112–119. DOI: `10.1109/IJCNN.2014.6889457`.

[140] Jiang Wang, Yi Yang, Junhua Mao, Zhiheng Huang, Chang Huang, and Wei Xu. "CNN-RNN: A Unified Framework for Multi-label Image Classification". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 2285–2294. DOI: `10.1109/CVPR.2016.251`.

[141] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, Wenyu Liu, and Bin Xiao. "Deep High-Resolution Representation Learning for Visual Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.10 (2021), pp. 3349–3364. DOI: `10.1109/TPAMI.2020.2983686`.

[142] Jinjiang Wang, Yulin Ma, Laibin Zhang, Robert X. Gao, and Dazhong Wu. "Deep learning for smart manufacturing: Methods and applications". In: *Journal of Manufacturing Systems* 48 (2018). Special Issue on Smart Manufacturing, pp. 144–156. ISSN: 0278-6125. DOI: `https://doi.org/10.1016/j.jmsy.2018.01.003`.

[143] Keze Wang, Dongyu Zhang, Ya Li, Ruimao Zhang, and Liang Lin. "Cost-Effective Active Learning for Deep Image Classification". In: *IEEE Transactions on Circuits and Systems for Video Technology* 27 (2017), pp. 2591–2600.

[144] Peng Wang, BaoWen Xu, YuRong Wu, and XiaoYu Zhou. "Link prediction in social networks: the state-of-the-art". In: *Science China Information Sciences* 58.1 (2015), pp. 1–38. DOI: `10.1007/s11432-014-5237-y`.

[145] Yalin Wang, Haibing Xia, Xiaofeng Yuan, Ling Li, and Bei Sun. "Distributed defect recognition on steel surfaces using an improved random forest algorithm with optimal multi-feature-set fusion". In: *Multimedia Tools and Applications* 77.13 (2018), pp. 16741–16770. DOI: `10.1007/s11042-017-5238-0`.

[146] Kilian Q Weinberger and Lawrence K Saul. "Distance metric learning for large margin nearest neighbor classification." In: *Journal of Machine Learning Research* 10.2 (2009).

[147] Xin Wen, Jvran Shan, Yu He, and Kechen Song. "Steel Surface Defect Recognition: A Survey". In: *Coatings* 13.1 (2023). ISSN: 2079-6412. DOI: `10.3390/coatings13010017`.

[148] John Winn, Christopher Bishop, and Tom Diethe. *Model-Based Machine Learning*. English. Microsoft Research, 2015.

[149] David H. Wolpert. "The lack of a priori distinctions between learning algorithms". In: *Neural Comput.* 8.7 (Oct. 1996), pp. 1341–1390. ISSN: 0899-7667. DOI: `10.1162/neco.1996.8.7.1341`.

[150] Mingrui Wu and Bernhard Schölkopf. "A local learning approach for clustering". In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. NIPS'06. Canada: MIT Press, 2006, pp. 1529–1536.

[151] Thorsten Wuest, Daniel Weimer, Christopher Irgens, and Klaus-Dieter Thoben. "Machine learning in manufacturing: advantages, challenges, and applications". In: *Production & Manufacturing Research* 4 (2016), pp. 23–45.

[152] Jeremy Wyatt, Alper Aydemir, Michael Brenner, Marc Hanheide, Nick Hawes, Patric Jensfelt, Matej Kristan, Geert-Jan Kruijff, Pierre Lison, Andrzej Pronobis, Kristoffer Sjöö, Alen Vrecko, Hendrik Zender, Michael Zillich, and Danijel Skočaj. "Self-Understanding and Self-Extension: A Systems and Representational Approach." In: *IEEE T. Autonomous Mental Development* 2 (Jan. 2010), pp. 282–303.

[153] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. "Unified Perceptual Parsing for Scene Understanding". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. Sept. 2018.

[154] Tete Xiao, Colorado J Reed, Xiaolong Wang, Kurt Keutzer, and Trevor Darrell. "Region Similarity Representation Learning". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. 2021, pp. 10539–10548.

[155] Junyuan Xie, Ross Girshick, and Ali Farhadi. "Unsupervised Deep Embedding for Clustering Analysis". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 478–487.

[156] Mengmeng Xu, Yancheng Bai, and Bernard Ghanem. "Missing Labels in Object Detection". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. June 2019.

[157] Wenguang Xu, Pengcheng Xiao, Liguang Zhu, Yan Zhang, Jinbao Chang, Rong Zhu, and Yunfeng Xu. "Classification and rating of steel scrap using deep learning". In: *Engineering Applications of Artificial Intelligence* 123 (2023), p. 106241. ISSN: 0952-1976. DOI: `https://doi.org/10.1016/j.engappai.2023.106241`.

[158] Sijie Yan, Yuanjun Xiong, and Dahua Lin. "Spatial temporal graph convolutional networks for skeleton-based action recognition". In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*. AAAI'18/IAAI'18/EAAI'18. New Orleans, Louisiana, USA: AAAI Press, 2018. ISBN: 978-1-57735-800-8.

[159] Zhicheng Yan, Hao Zhang, Robinson Piramuthu, Vignesh Jagadeesh, Dennis DeCoste, Wei Di, and Yizhou Yu. "HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 2740–2748.

[160] Linjie Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. "A large-scale car dataset for fine-grained categorization and verification". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 3973–3981. DOI: `10.1109/CVPR.2015.7299023`.

[161] Scott Cheng-Hsin Yang, Daniel M Wolpert, and Máté Lengyel. "Theoretical perspectives on active sensing". In: *Current Opinion in Behavioral Sciences* 11 (2016). Computational modeling, pp. 100–108. ISSN: 2352-1546. DOI: `https://doi.org/10.1016/j.cobeha.2016.06.009`.

[162] Mohammadreza Yazdchi, Mehran Yazdi, and Arash Golibagh Mahyari. "Steel Surface Defect Detection Using Texture Segmentation Based on Multifractal Dimension". In: *2009 International Conference on Digital Image Processing*. 2009, pp. 346–350. DOI: `10.1109/ICDIP.2009.68`.

[163] Donggeun Yoo and In-So Kweon. "Learning Loss for Active Learning". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019), pp. 93–102.

[164] Bing Yu, Haoteng Yin, and Zhanxing Zhu. "Spatio-temporal graph convolutional networks: a deep learning framework for traffic forecasting". In: *Proceedings of the 27th*

International Joint Conference on Artificial Intelligence. IJCAI'18. Stockholm, Sweden: AAAI Press, 2018, pp. 3634–3640. ISBN: 9780999241127.

[165] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. "GraphSAINT: Graph Sampling Based Inductive Learning Method". In: *ArXiv* abs/1907.04931 (2019).

[166] Min-Ling Zhang and Zhi-Hua Zhou. "ML-KNN: A lazy learning approach to multi-label learning". In: *Pattern Recognition* 40.7 (2007), pp. 2038–2048. ISSN: 0031-3203. DOI: `https://doi.org/10.1016/j.patcog.2006.12.019`.

[167] Min-Ling Zhang and Zhi-Hua Zhou. "Multilabel Neural Networks with Applications to Functional Genomics and Text Categorization". In: *IEEE Transactions on Knowledge and Data Engineering* 18.10 (2006), pp. 1338–1351. DOI: `10.1109/TKDE.2006.162`.

[168] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. "Pyramid Scene Parsing Network". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6230–6239. DOI: `10.1109/CVPR.2017.660`.

[169] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Z. Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jianyun Nie, and Ji-rong Wen. "A Survey of Large Language Models". In: *ArXiv* abs/2303.18223 (2023).

[170] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. "Scene Parsing through ADE20K Dataset". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 5122–5130. DOI: `10.1109/CVPR.2017.544`.

[171] Yi-Tong Zhou and Rama Chellappa. "Computation of optical flow using a neural network". In: *IEEE 1988 International Conference on Neural Networks* (1988), 71–78 vol.2.

[172] Bo Zong, Qi Song, Martin Renqiang Min, Wei Cheng, Cristian Lumezanu, Dae-ki Cho, and Haifeng Chen. "Deep Autoencoding Gaussian Mixture Model for Unsupervised Anomaly Detection". In: *International Conference on Learning Representations*. 2018.