UNIVERSITY OF CAPE TOWN

FULL DISSERTATION PRESENTED FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF MATHEMATICS

Deep Adaptive Anomaly Detection using an Active Learning Framework

Author:
Emmanuel Sekyi (SKYEMM001)

Supervisor:
Prof. Bruce Bassett

October 6, 2022

Declaration

I, **Emmanuel Kwame Sekyi**, hereby declare that:

1. I am presenting this dissertation in full fulfilment of the requirements for my degree.
2. I know the meaning of plagiarism and declare that all of the work in the dissertation, except where acknowledgements indicate otherwise, is my own.
3. I grant the University of Cape Town free licence to reproduce for the purpose of research either the whole or any portion of the contents in any manner whatsoever of the above dissertation.

Signed: Signed by candidate Date: ____________

Abstract

University of Cape Town

Faculty of Science, Department of Mathematics

by Emmanuel Kwame Sekyi

October 6, 2022

Anomaly detection is the process of finding unusual events in a given dataset. Anomaly detection is often performed on datasets with a fixed set of predefined features. As a result of this, if the normal features bear a close resemblance to the anomalous features, most anomaly detection algorithms exhibit poor performance. This work seeks to answer the question, can we deform these features so as to make the anomalies standout and hence improve the anomaly detection outcome? We employ a Deep Learning and an Active Learning framework to learn features for anomaly detection. In Active Learning, an Oracle (usually a domain expert) labels a small amount of data over a series of training rounds. The deep neural network is trained after each round to incorporate the feedback from the Oracle into the model. Results on the MNIST, CIFAR-10 and Galaxy Zoo datasets show that our algorithm, **Ahunt**, significantly outperforms other anomaly detection algorithms used on a fixed, static, set of features. Ahunt can therefore overcome a poor choice of features that happen to be suboptimal for detecting anomalies in the data, learning more appropriate features. We also explore the role of the loss function and Active Learning query strategy, showing these are important, especially when there is a significant variation in the anomalies.

Key words: Anomaly detection – Deep learning - Active Learning

Acknowledgements

This thesis is the result of the support and encouragement of many people some of whom are listed below.

Firstly, I would like to extend my deepest gratitude to my Supervisor, Prof. Bruce Bassett, whose valuable feedback and patience has helped shape the quality of this work.

I would like to thank my former and current research group members especially, Dr. Nadeem Oozeer, Dr. Alireza Vafaei Sadr and Dr. Ethan Roberts for their encouragement, insightful suggestions and support without which this thesis wouldn't have seen the light of day.

Last but most importantly, I would like to thank my family for their love and support.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Motivation

Anomaly detection aims at mining datasets for patterns which do not conform to some defined notion of normality (*Chandola et al.*, 2009). Techniques in anomaly detection span several research areas including, but not limited to, medical research (*Churová et al.*, 2021), fault detection (*Li et al.*, 2010), Astronomy (*Lochner and Bassett*, 2021) and cyber-security (*Evangelou and Adams*, 2020).

The goal of anomaly detection varies widely with the application domain. For example, anomaly detection when applied to the medical domain may lead to the detection of an early onset of a disease (*Fernando et al.*, 2020) which is in contrast to financial data where the goal might be to detect fraudulent activities.

In most anomaly detection setups, it is often important to choose features which make the anomalies stand out. For example, if one's aim is to discover new types of animals, using the number of legs or eyes might not help if the new class of animal has two eyes and four legs but also has wings and breathes fire.

Feature extractors are often used to deform the features in the raw data into suitable representations for anomaly detection. However, most feature extractors are often generic or fitted to describe the "normal" data (*Tailanián et al.*, 2021). This is evident in *Lochner and Bassett* (2021) and *Chenguang et al.* (2020) where the authors use ellipse fitting as a feature extractor for galaxies and body parts (head and torso) respectively.

Anomalies which do not stand out in the chosen set of features might escape detection even with the most sensitive anomaly detection algorithms. Addressing this difficulty is the main motivation of this thesis.

1.2 Aims and Objectives of Study

The aim of this thesis is to explore dynamically learned features that allow for improved anomaly detection in images.

We do this by using Active Learning. In Active Learning, an Oracle (typically an expert human) labels a small number of examples for training a model. In the context of this work, the Oracle labels a small subset of images for training a neural network.

This aim is reduced to 2 main objectives:

1. To use deep Convolutional Neural Networks (CNN) to learn features that make anomalies stand out.
2. To employ Active Learning to provide direct human feedback to the model about interesting features that might help detect anomalies.

1.3 Related Work

Feature extraction for anomaly detection is a longstanding research problem (*Nguyen and Gopalkrishnan*, 2010). Techniques such as Principal Component Analysis (PCA) and Subspace learning have been explored for feature extraction (*Thudumu et al.*, 2020). *Sadr et al.* (2019) proposed a 3 stage framework for anomaly detection in high dimensions. In the first stage, approaches such as PCA (*Jolliffe and Springer-Verlag*, 2002), autoencoders (*Hinton and Zemel*, 1993) and variational autoencoders (*Kingma and Welling*, 2014) are employed for dimensionality reduction. These features are then clustered in the second stage. In the final stage, the distance between the cluster centres and the test set is computed to find the relevant anomalies. This framework exhibited competitive performance on high dimensional datasets.

In an attempt to introduce human feedback into the anomaly detection setup, *Lochner and Bassett* (2021) introduced an active learning approach to rank interesting anomalies in astronomical data. Their algorithm doubled the number of relevant anomalies found in the first 100 data points shown to the user. However, in their approach, the features are always static and unchanging. This makes it susceptible to poor performance if the chosen features were inappropriate.

With the advent of deep learning, there has been a movement to automatically learn representational features for anomaly detection. *Andrews et al.* (2016) employed both a pretrained network and a CNN trained from scratch to extract features from different image datasets for anomaly detection. They noticed that, although using a pretrained network is a good baseline for extracting features, it is important for dataset to be semantically similar to the one on which the pretrained network was originally trained.

However, most deep learning approaches require huge amount of data to learn meaningful representations of the features. Various Approaches in Zero and One-Shot learning have been adopted to help tackle the data scarcity in anomaly detection (*O′ Mahony et al.*, 2019; *Xian et al.*, 2017).

1.4 Thesis Organisation

This thesis is organized into 5 main sections. **Chapter 1** introduces the topic of anomaly detection. We discuss the aims, objectives and previously published work that is closely related to this thesis. We also discuss some applications and challenges faced in anomaly detection.

Chapter 2 gives a walk through of some classical anomaly detection algorithms. We explore approaches such as Isolation Forests, Local Outlier Factors and One-Class Support Vector Machines. This chapter also discusses some commonly used evaluation metrics in classification and anomaly detection. To conclude the chapter, we introduce Active Learning, discuss various query strategies and how Active Learning is used in anomaly detection.

Deep Learning is at the core of this thesis hence we dedicate **Chapter 3** to discussing them. Even before discussing the history of neural networks, we introduce different machine learning paradigms. We proceed to discuss the commonly used loss and activation functions used in training deep learning. This chapter also examines different Neural Network architectures especially those used in anomaly detection. We conclude the chapter by unifying how the different networks and learning paradigms are used in anomaly detection.

Chapter 4 discusses We start by giving a brief overview of Ahunt along with our methodology and comparison algorithms. We discuss the datasets used in our experiments as well as the nitty gritties of the experiments. **Chapter 4** also discusses our results. In **Chapter 5**, we conclude our work and suggest future research directions.

1.5 Introduction to Anomaly Detection

The term "anomaly" has its origin in the Greek word **anomolia** which means uneven or irregular (*Madhuri and Usha Rani*, 2020). Anomalies can be broadly defined as unusual or inconsistent observations in a given dataset. These observations are also referred to as outliers, abnormalities or rare events.

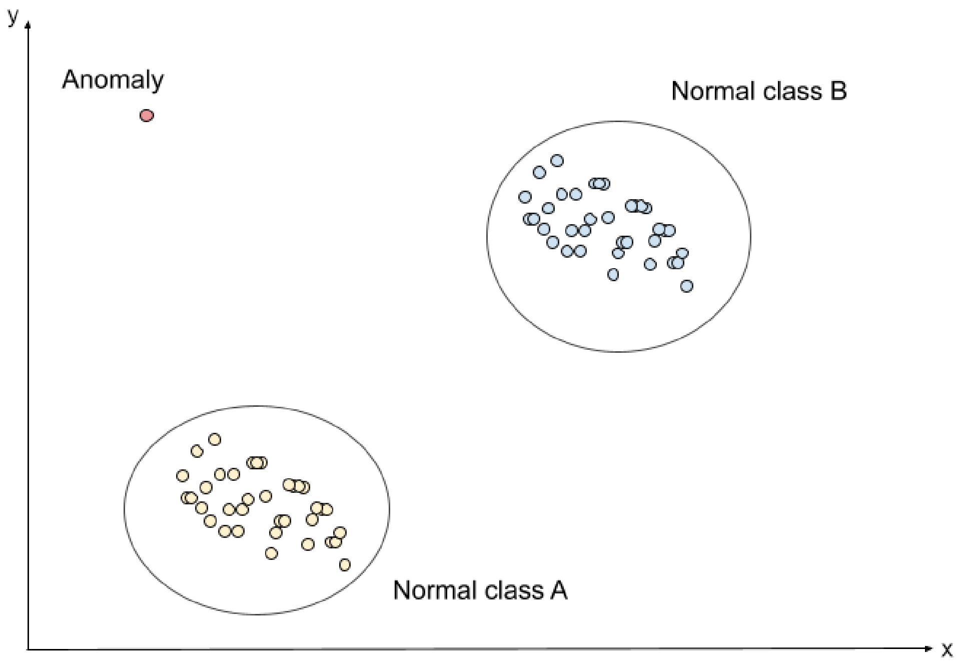


Figure 1.1: Diagram illustrating two classes of normal objects (class A and B). Similar objects have close proximity, forming two different clusters. The anomaly, indicated in the red dot, is completely separated from the two clusters. Contextually, this might indicate that the observation shown in red does not share similar characteristics as those in class A or class B.

In Figure 1.1 it is assumed that normal instances are those that form clearly defined clusters as seen in class A and B. The anomalies therefore are the points which do no share any similarity with any of the normal clusters.

The clear separation of the anomalies from the normal classes is dependent on the choice of features. A different choice of features in Figure 1.1 might bring the anomaly closer to the big clusters and hence make it difficult to detect. This thesis seeks to address this challenge.

1.5.1 Types of Anomalies

Anomalies can be classified based on their nature. Most anomalies fall in 3 broad categories:

1. **Point Anomalies**: A data point is considered a point anomaly if it deviates from other data points in a given dataset. It is the simplest type of anomaly. An example of a point anomaly is seen in fraudulent bank transactions. If a transaction is significantly larger than the normal range from historical data, this amount can be categorised as a point anomaly. The anomaly in Figure 1.1 is an illustration of a point anomaly.

2. **Contextual Anomalies**: These are data instances that are considered anomalies with respect to a specific context. They are also called conditional anomalies (*Song et al.*, 2007). The anomalies are specified with using the attributes of the data with respect to a certain context. This means that, the same data instance in another context might be considered normal. An example is seen in temperature data. Temperatures of $-20°C$ in the summer might be considered abnormal but the same temperature in the winter will be considered normal. The context as specified in this example is the season in which the temperature was recorded.

3. **Collective Anomalies**: Collective anomalies refers to data instances considered anomalous when they occur together. The data points in the group might not be considered anomalous individually but their occurrence together forms an anomaly. Collective anomalies usually occur in data instances which are related. A single neighbour moving out of their house on a particular day may not be considered abnormal behavior but if all the neighbours move out on the same day, that might be considered abnormal.

1.6 Challenges in Anomaly Detection

The nature of anomalies introduce unique complexities in their detection. Some of these challenges include:

1. Anomalies are often unknown prior to their occurrence. This makes it difficult to anticipate their nature and subsequently detect them. This is the case in cyber-security where hackers use zero-day exploits to identify vulnerabilities before developers become aware of the existence of such vulnerabilities.

2. Most anomaly detection problems lack labelled data. This makes it difficult to apply supervised machine learning algorithms to learn the anomalous patterns in the data.

3. Anomalies are often rare events. This means that, when compared to normal events, anomalies will likely be under-represented in a given dataset. Their under-representation might lead to a class imbalance problem which subsequently makes it difficult to detect anomalies.

4. Anomalies are often heterogeneous. For example, in video surveillance, actors might be involved in fights, burglary or robbery. Although all these things might be labelled as anomalies, they differ in how they manifest.

5. In some datasets, noise and artefacts appear to be highly anomalous but not interesting. Distinguishing them is often challenging. This might affect the outcome of detecting anomalies.

6. Anomalies often evolve. In cases such as fraud detection, where malicious actors are involved, they constantly adapt their attacks to escape detection.

7. Anomaly detection in high dimensions is challenging. Most anomaly detection techniques try to reduce the dimensionality of the dataset. This introduces further problems since there is no guarantee that the anomalous features are preserved.

Due to these and many more challenges, most of the existing anomaly detection techniques try to solve a specific formulation of the problem. The formulation is often influenced by the type of anomaly, the nature of the data, the availability of labelled instances and the application domain.

1.7 Applications of Anomaly Detection

Anomaly detection can be applied to almost any domain. In some cases, it is used as a preprocessing step to remove outliers. In this section, we examine some of the widely known applications of anomaly detection.

1. **Fraud Detection**: Fraud detection refers to the detection of illegal activities in commercial settings such as banks, insurance companies, the stock market, telecommunication companies and credit card companies. The attackers might be existing users of the system or outsiders posing as existing users. Their activities are labelled fraudulent when they access or use resources without authorization. A common attack is seen in credit card systems. Malicious actors steal the identity of an existing user and go on to make unauthorized transactions. In such scenarios, an anomaly detection system is expected to flag the transaction as soon as it occurs (*Lucas*, 2020).

   Bypass fraud (SIMboxing) is also a major concern in the telecommunications industry (*Sallehuddin et al.*, 2015). SIMboxing is the illegal setting up of low-cost local Subscriber Identity/Identification Module cards such that callers pay local rates instead of the expected international rates. This leads to a huge loss of revenue on the part of telecommunication companies. *Elmi et al.* (2013) used a Multi Layer Perceptron on carefully curated features to build a classifier to mitigate SIMboxing.

2. **Intrusion Detection**: Intrusion detection is the ability to monitor malicious activities (break-ins, penetrations, and other forms of computer abuse) in computer systems (*Phoha*, 2002). An intrusion is different from the normal

functioning of the computer system hence the problem can be formulated as an anomaly detection task. Data from intrusion detection systems (IDS) comes in large volumes. The requires the anomaly detection problem in IDS to be computationally efficient in order to handle such large streams of data. As a result of the large volume of input data, anomaly detection tasks in IDS are likely to have a high false alarm rate. Also, malicious actors in this domain adapt their attacks to evade detection by most anomaly detection techniques. This adds an extra layer of complexity to the anomaly detection process in this domain.

3. **Healthcare**: In the healthcare domain, a variety of patient data points are collected. The data could come from X-rays, Magnetic Resonance Imaging or Computed Tomography scans. Anomalies in these datasets may be indicative of a health condition. Identifying anomalies in medical data enables practitioners to diagnose and provide early treatments for a variety of medical conditions.

   The fields of genetics, metabolomics and proteomics use anomaly detection algorithms to find unusual mutations that may signal specific diseases. Furthermore, anomaly detection algorithms are applied to identify points in time at which an effective treatment for a disease ceases to be potent. This signifies the emergence of a drug-resistant mutation of the responsible pathogen (*Caroprese et al.*, 2009).

   Detecting anomalies in the medical domain can be challenging. This is because the cost of false negatives is often detrimental to patient outcomes. This makes it difficult to use black-box models such as deep learning in this domain even though they exhibit superior performance (*Fernando et al.*, 2020). In recent times, interpretable models have been developed to help mitigate this issue (*Margeloiu et al.*, 2020; *Schutte et al.*, 2021).

4. **Manufacturing and Industry**: Mechanical systems in industries are prone to

wear and tear. This could lead to sub-optimal performance in their usage. Anomaly detection is leveraged to detect and mitigate these problems. One main challenge with this is the lack of labelled data. In most cases, the only data available is the normal data of the machine parts. It is often difficult to find data on damaged machine parts. *Michau et al.* (2018) explored learning features from healthy or normal machine conditions to detect faults.

Anomaly detection can also be applied to understand customer behaviour. The behaviour of customers is learned from previous transactions. Anomaly detection algorithms help to identify deviations for further investigation.

Chemical processing industries also use anomaly detection techniques to detect discrepancies in chemical processes. These discrepancies might range from sudden unexpected chemical reactions to a decrease in expected yield. The main challenge with this task is the varying nature of the anomalies. Data collected on one type of failure might not necessarily generalize to other types of failures. *Chen* (2018) explored the use of neural networks to detect process abnormalities in the Petrochemical industry.

5. **Defense and Internal Security**: Anomaly detection is applied in the detection of unusual behaviour of people in public spaces. This is done via video surveillance. Based on the regular behaviour of people in such spaces, we can identify patterns that deviate from this behaviour (*Bhakat and Ramakrishnan*, 2019).

6. **Astronomy**: Astronomical observations produce massive amounts of data. For example, the Sloan Digital Sky Survey has produced images and spectroscopic measurements containing about 230 million celestial objects. The Square Kilometer Array is also projected to produce exabyte-scale datasets. This amount of data is too large for researchers to manually explore and find interesting astronomical phenomena. Anomaly detection techniques are explored to find interesting astronomical phenomena. *Lochner and Bassett*

(2021) introduced a general framework using active learning for detecting interesting anomalies.

1.8 Summary

In this chapter, we introduced the anomaly detection landscape. We also discussed our problem of interest along with the aims and objectives of this thesis. We proceed to Chapter 2 where we discuss some classical anomaly detection algorithms, evaluation metrics and how active learning is used in anomaly detection.

2 | Algorithms for Anomaly Detection

2.1 Chapter Introduction

In Chapter 1, we introduced the broad subject of anomaly detection, its applications and some of the challenges encountered in anomaly detection. At the core of the anomaly detection problem are anomaly detection algorithms. This chapter presents some of the commonly used algorithms in anomaly detection. We divide the algorithms into 3 categories namely:

1. Proximity-based approaches: This category of algorithms explore the distances and densities between features to detect likely anomalies.
2. Machine Learning-based approaches: This category uses machine learning techniques for anomaly detection. We will focus on the case where the anomaly detection problem is framed as a classification problem. This requires some labelled data of either the normal class, the anomaly class or both. Typically, the labels for the normal classes will be readily available and hence the problem is framed as a one-class classification problem.
3. Statistical approaches: This approach explores the statistical properties of the data to detect anomalies. A probability distribution model is often built for the given dataset. The assumption here is that, anomalies have a low probability with respect to this model.

Another central theme of the anomaly detection setup is the evaluation metric.

Given that our proposed approach frames the problem as a special case of classification, we discuss some evaluation metrics used in classification.

Finally, we examine Active Learning and how it is applied in anomaly detection.

2.2 Proximity-based Methods

Proximity-based approaches define a point as anomalous if its locality is sparsely populated (*Aggarwal*, 2017). Proximity-based approaches can be classified into distance and density based approaches.

In distance-based approaches, a data point is considered anomalous if its distance to the other data points is beyond a defined threshold whereas in density-based approaches the relative density of a point is compared with that of its neighbours. In density-based approaches, it is assumed that the density around a normal data point is similar to the density around its neighbours and the density around an anomaly is significantly different to that of its neighbours.

Examples of proximity-based algorithms include Isolation Forests and Local Outlier Factor. In the sub-sections below, we discuss some of the commonly used proximity-based anomaly detection algorithms.

2.2.1 Isolation Forests

Isolation forest (iForests) consist of decision trees known as Isolation trees (*Liu et al.*, 2008). Each tree is represented as a proper binary tree. In a proper binary tree, every node consists of either 0 or 2 children nodes. Isolation forest builds an ensemble of these isolation trees to isolate instances of anomalies. iForests do this by taking advantage of two properties of anomalies:

1. they are in the minority of the data sample
2. they have attribute values that are very different from that of the normal

instances.

The trees recursively partition the data by random sub-sampling until all instances are isolated as shown in Figure 2.1. The anomalies are the instances with shorter average path length (the path from the root node to the terminating node). The number of partitions required to isolate a point is equivalent to the path from the root node to the terminating node. To estimate whether an instance is an anomaly, the path lengths of individual trees is averaged.

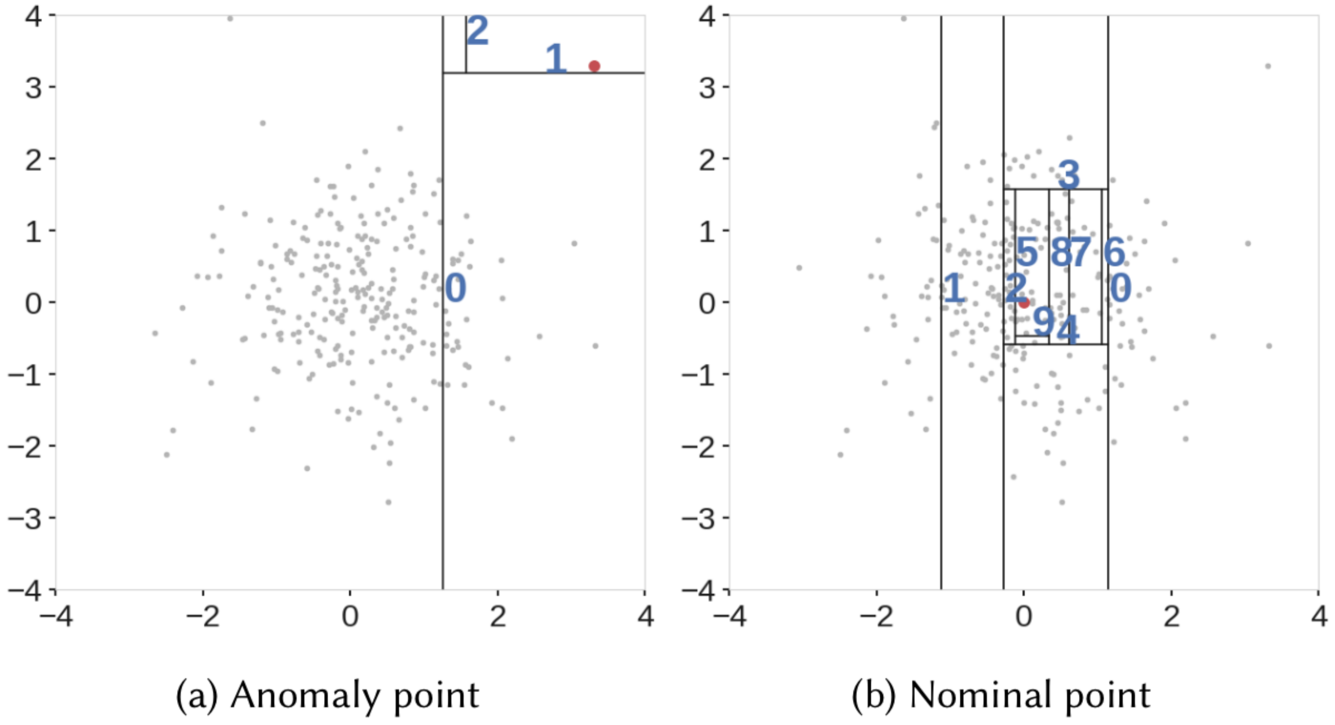


Figure 2.1: A diagram indicating the number of partitions required to isolate an anomaly vs a normal point using Isolation Forest. The figure on the left indicates the number of partitions required to separate an anomaly (which is indicated in the red dot). It took 3 partitions to isolate the anomaly. The figure on the right required 9 partitions to isolate a randomly chosen normal point. This goes to assert the claim that, in Isolation Forests, the anomalies are the instances with the shortest average path length *Hariri et al.* (2018).

2.2.1.1 Estimating the Anomaly Score

As discussed in section 2.2.1, anomalies in the context of iForests are data instances with shorter average path length. For a data point x, its path length $(h(x))$ is the number of edges it traverses from the root node to the terminating node.

The average path length, $c(n)$, of an iForests is the same as an unsuccessful search in a Binary Search Trees (BST). For a dataset with n instances, the average path length of an unsuccessful search in a BST is given by the equation:

$$c(n) = 2H(n-1) - \left(\frac{2(n-1)}{n}\right) \tag{2.1}$$

where $H(n)$ is the harmonic number and it can be estimated as $\ln(n) + 0.5772156649$ (Euler's constant).

The path length ($h(x)$) of a data instance x is normalized using the average path length of all the instances in the dataset, $c(n)$. The anomaly score of x is defined as:

$$s(x, n) = 2^{-\frac{E(h(x))}{c(n)}} \tag{2.2}$$

where $E(h(x))$ is the average of the path lengths ($h(x)$) from a collection of isolation trees. If the anomaly score (**s**) of a data instance x is very close to 1, then x is an anomaly. If **s** is significantly smaller than 0.5, then x is a normal instance.

2.2.1.2 Advantages of Isolation Forests

1. It works well with small sample sizes. This helps to reduce the effects of masking[1] and swamping[2] .
2. It is computationally efficient. iForests utilize no distance or density measures to calculate anomalies removing the need for high computational cost. The algorithm runs in linear time.
3. It scales to handle large data sizes and high-dimensional problems.

[1]Masking is the existence of too many anomalies therefore concealing their presence

[2]Swamping refers to wrongly identifying normal instances as anomalies. This can occur when the normal instances and anomalies are too close hence the number of partitions required to isolate the anomalies becomes approximately equal to that of the normal instances

2.2.2 Local Outlier Factor

The Local Outlier Factor (LOF) algorithm compares the local density of a point to the local density of k of its neighbours (*Breunig et al.*, 2000). The points which have a lower density than their neighbours are considered outliers. LOF is said to be local because only a restricted neighbourhood of each data point is taken into account when estimating the outlier score. The local outlier factor is estimated in the 4 steps discussed below.

2.2.2.1 K-Distance and K-Neighbors

K-distance is the distance between a point and its k^{th} nearest neighbour. The K-neighbours of point A, $N_k(A)$, consists of a set of points that lie within a circle of radius K-distance as shown in Figure 2.2.

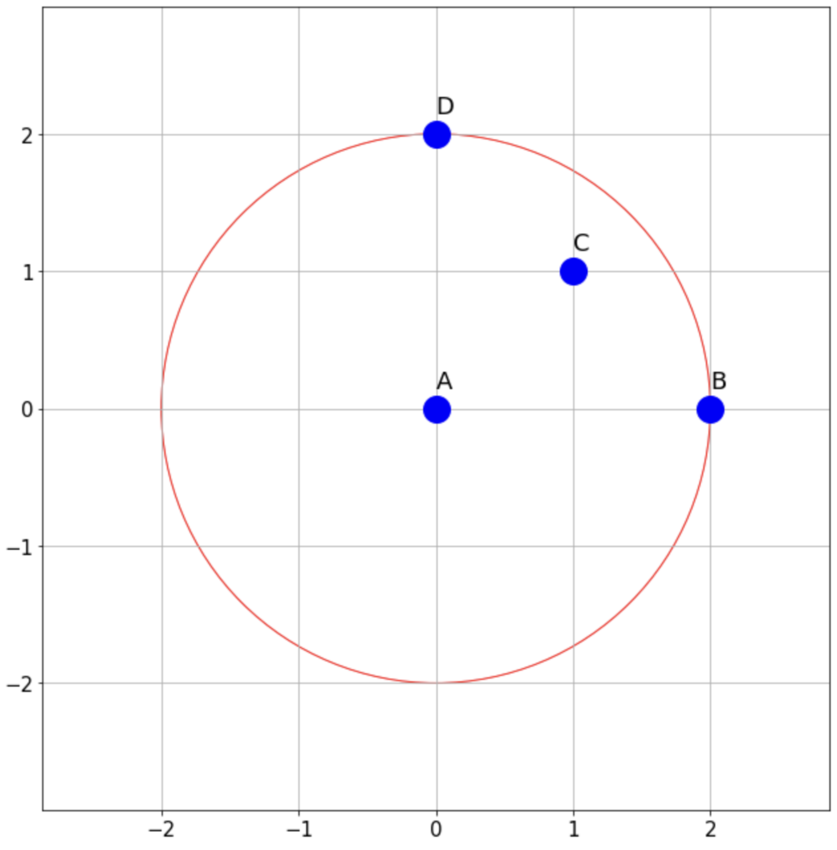


Figure 2.2: The diagram shows the K-distance of point A. In this case, K = 2 making C, B and D the K-neighbours of A (*Jayaswal*, 2020).

Specifying a small value of K makes the algorithm sensitive to noise and a large value may skip anomalies. Hence the choice of an optimal K is done carefully based on the nature of the data.

2.2.2.2 Reachability Distance

The reachability distance (RD) of an object X_j with respect to X_i is defined as

$$RD_k(X_j, X_i) = max(K{-}distance(X_i),\ d(X_j, X_i)) \tag{2.3}$$

where d is a normal distance measure betweeen X_j and X_i as indicated in the orange line in Figure 2.3. It can be either Euclidean, Manhattan, Minkowski or other distance measures depending on the problem.

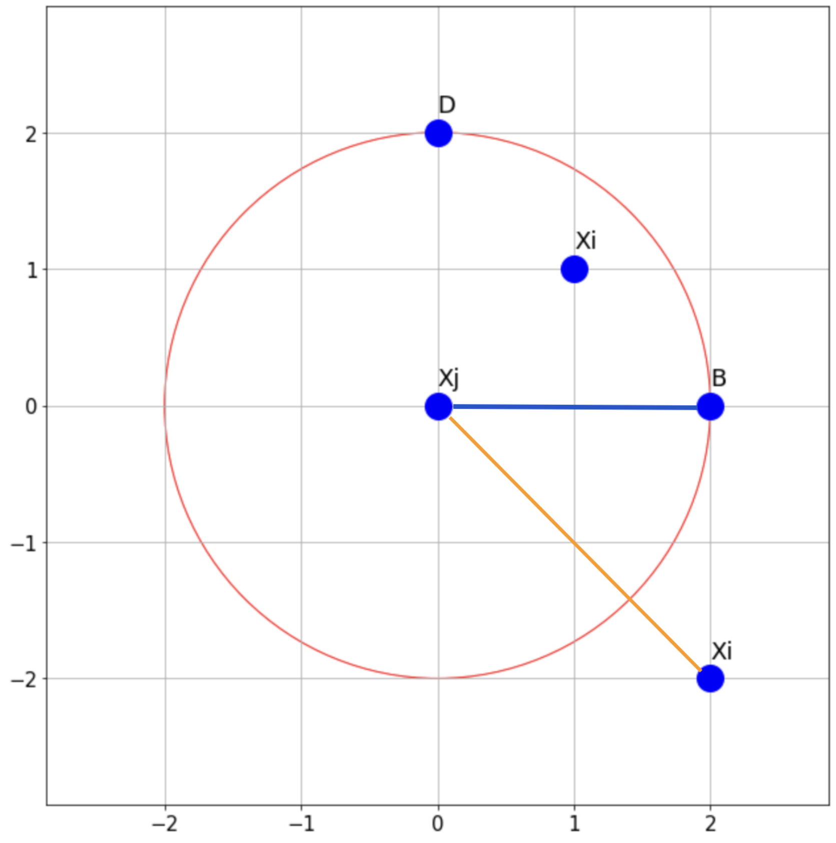


Figure 2.3: This diagram shows points lying in a red circle with a K-distance of 2. The points are the K-neighbours of X_j, that is, they are within the circle, will have a reachability distance equal to the K-distance which is indicated with the blue line. For points outside the K-neighbours, such as X_i, the reachability distance will be the distance between X_i and X_j as indicated in the orange line.

2.2.2.3 Local Reachability Density

The Local Reachability Density (LRD) estimates how far a point is from a cluster of points. It is calculated as the inverse of the average reachability distance of a point from its neighbours. A low LRD implies that a cluster is far from the point under consideration. The LRD of a point A is estimated as:

$$\mathrm{LRD}_k(A) = \frac{1}{\sum_{X_j \in N_k(A)} \frac{\mathrm{RD}_k(A, X_j)}{||N_k(A)||}} \tag{2.4}$$

where $N_k(A)$ represents the K-neighbours of point A and $||N_k(A)||$ represents the number of points within that neighbourhood.

2.2.2.4 The Local Outlier Factor

The Local Outlier Factor (LOF) of A is the ratio of the average LRD of the K-neighbors of A to the LRD of A.

$$\mathrm{LOF}_k(A) = \frac{\sum_{X_j \in N_k(A)} \mathrm{LRD}_k(X_j)}{||N_k(A)||} \times \frac{1}{\mathrm{LRD}_k(A)} \tag{2.5}$$

The LOF is calculated for each data point. The data points are ranked by their LOF to find the most likely outliers. If a point is an inlier, the ratio of the average LRD of its neighbours is approximately equal to its LRD (they are part of the same cluster) making its LOF approximately 1. However, for an outlier, its LRD will be less than the average LRD of its neighbours hence it'll have a high LOF.

2.3 Machine Learning-based Methods

2.3.1 Learning Paradigms

The goal of machine learning is to learn underlying patterns from a set of observations. The learning of these patterns can be captured in 3 main learning paradigms. The different paradigms are examined in subsequent sections.

2.3.1.1 Supervised Learning

In supervised learning, the model is presented with labelled instances of the observation. The process is described as supervised in the sense that, a 'supervisor' has labelled each input. Supervised learning is the most commonly used paradigm. It is also the most studied and mature paradigm in machine learning. Supervised learning can be broadly classified into two categories: classification and regression. These categories vary depending on the nature of their labels. In classification, the labels are discrete values. An example is the classification of handwritten digits, MNIST (*LeCun et al.*, 2010). Regression problems have continuous value labels. An example is the prediction of sales in a retail store.

2.3.1.2 Unsupervised Learning

In unsupervised learning, the observations do not have labels. It can be viewed as a way of creating a higher level of representation of the data.

An example of unsupervised learning is clustering. In clustering, the algorithm learns some underlying patterns from unlabelled data and groups them by some similarity metric.

2.3.1.3 Reinforcement Learning

In reinforcement learning, data is not explicitly fed to the algorithm. The algorithm learns by interacting with its environment. This paradigm is concerned with how an agent can take actions in an environment so as to maximize some results. It is applied in settings such as robot control and games (*Silver et al.*, 2016).

2.3.1.4 Other Learning Paradigms

There are other paradigms some of which are amalgamations of the 3 main paradigms described above. Training data often comes in different formulations. An example is when there are huge number of unlabelled data but few labelled instances. Semi-supervised learning is applied in such scenarios (*Sambasivam and Opiyo*, 2021).

Data with noisy labels can also be used for training. This approach is called weakly supervised learning (*Zhou*, 2017). It is used in scenarios where labelled data can be expensive to obtain.

Another type of learning which has received a lot of attention in recent years is Self-Supervised Learning (SSL). In Self-Supervised learning, the unlabelled training data derives supervisory signal from itself. It does this by solving an automatically generated task called a pretext task. The key assumption here is that, the model learns useful semantic representations in the process of solving the pretext task. Examples of pretext tasks include Inpainting (*Pathak et al.*, 2016), Patch prediction (*Doersch et al.*, 2015) and solving jigsaw puzzles (*Noroozi and Favaro*, 2016).

Classification-based approaches frame the anomaly detection problem as a classification task. The One-Class Support Vector Machine is one of the most commonly used algorithms in this category.

2.3.2 Support Vector Machines

A Support Vector Machine (SVMs) is a commonly used algorithm for classification and regression tasks. Consider a binary classification problem where the classes are well separated as shown in Figure 2.4, an SVM finds a hyperplane with the largest margin between the classes. SVMs were first proposed by *Vapnik and Lerner* (1963).

SVMs find a separating hyperplane that depend on the points that lie on the margin, called support vectors.

While traditional SVMs are useful for binary and multi-class classification, they are not well-suited for domains where there is only one class as typically seen in some

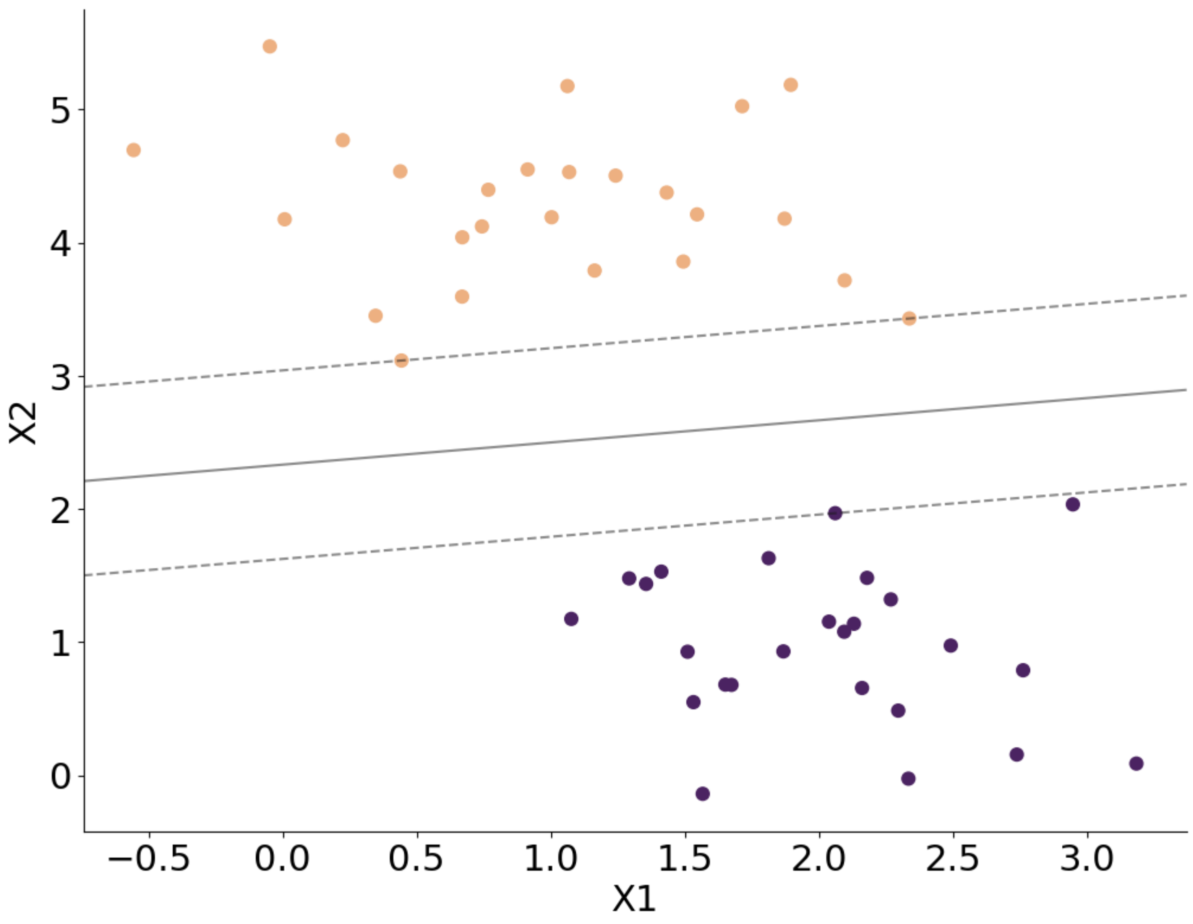


Figure 2.4: Diagram illustrating a separation margin obtained from fitting an SVM to a binary classification problem. The two classes indicated in the purple and orange dots are separated by the largest margin indicated by the two dashed lines. Adapted from code in *Vanderplas* (2017).

anomaly detection problems. In such domains, we're presented with a problem where there is abundance of labels for the normal class but little to no label for the anomalous class. The One-class Support Vector Machine (OCSVMs) was created to address this problem (*Muñoz and Moguerza*, 2004). OCSVMs build a model of the normal instances and identifies points which deviate from this model. It does this by minimizing a hypersphere enclosing the normal class. The hypersphere is characterized by a center $\mathbf{c}$ and a radius $R > 0$ as distance from the center to the support vectors on the boundary. A data point is considered normal when its distance to the center is smaller than or equal to R while anomalous points exist outside the hypersphere hence their distance to the center exceeds R.

2.4 Statistical Models for Anomaly Detection

2.4.1 Kernel Density Estimator

A density estimator is an algorithm that models the underlying probability distribution of a given dataset. Given a one dimensional dataset, a histogram is a simple way of doing this estimation. The histogram divides the data into discrete bins and counts the number points in each bin. However, depending on the choice and location of the bin, our interpretation of the distribution might vary.

Kernel density estimators (KDE) improve on histograms by avoiding the loss of information due to binning. It also gives a smooth curve defined at all points, which is often more useful for practical applications. KDE places a smooth bump of each data point and sums them to obtain the final density estimate.

For some identically distributed independent samples, $(x_1, x_2, \ldots, x_n)$, drawn from an unknown univariate distribution, f, at any given point x, its kernel density estimator is given by

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right), \tag{2.6}$$

where K is called the kernel function that is generally a smooth, symmetric function such as a Gaussian kernel. There are several kernels to choose from as illustrated in Figure 2.5.

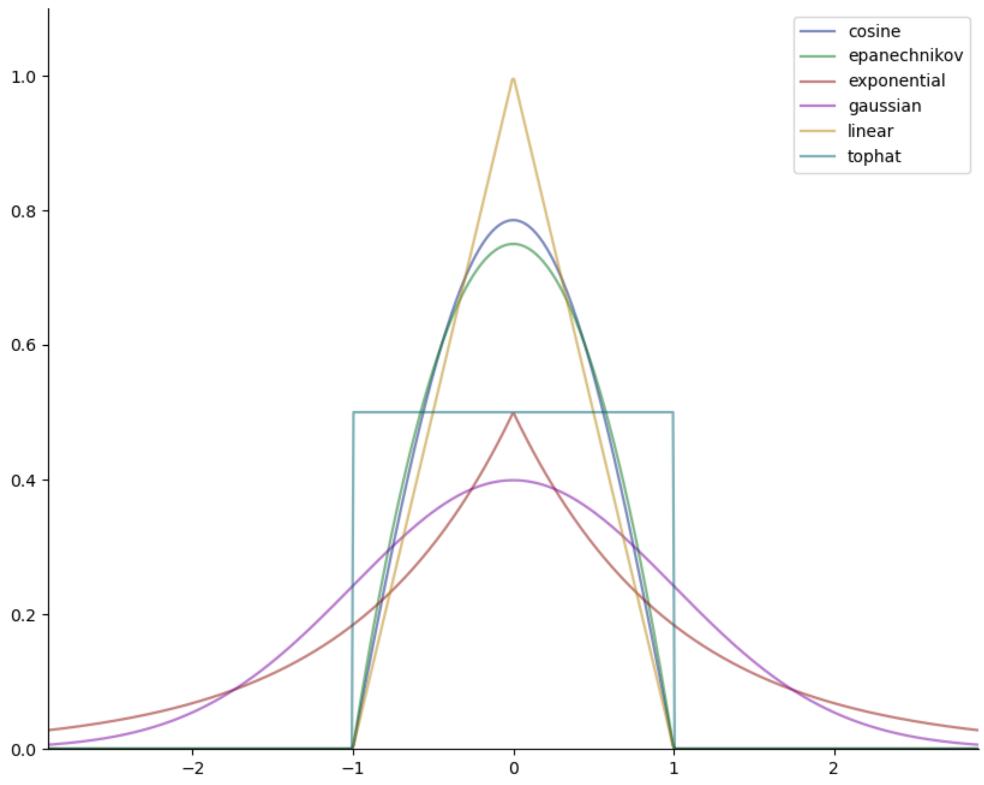


Figure 2.5: Diagram of available Kernel functions for KDE in the sklearn library. Different kernel shapes might affect smoothness of the resulting distribution and hence the interpretation of the underlying data.

$h > 0$ is called the smoothing bandwidth that controls how peaked or spread out the KDE is around each data point as shown in Figure 2.6. When h is too small, it results in under-smoothing but when h is too big, it results in over-smoothing.

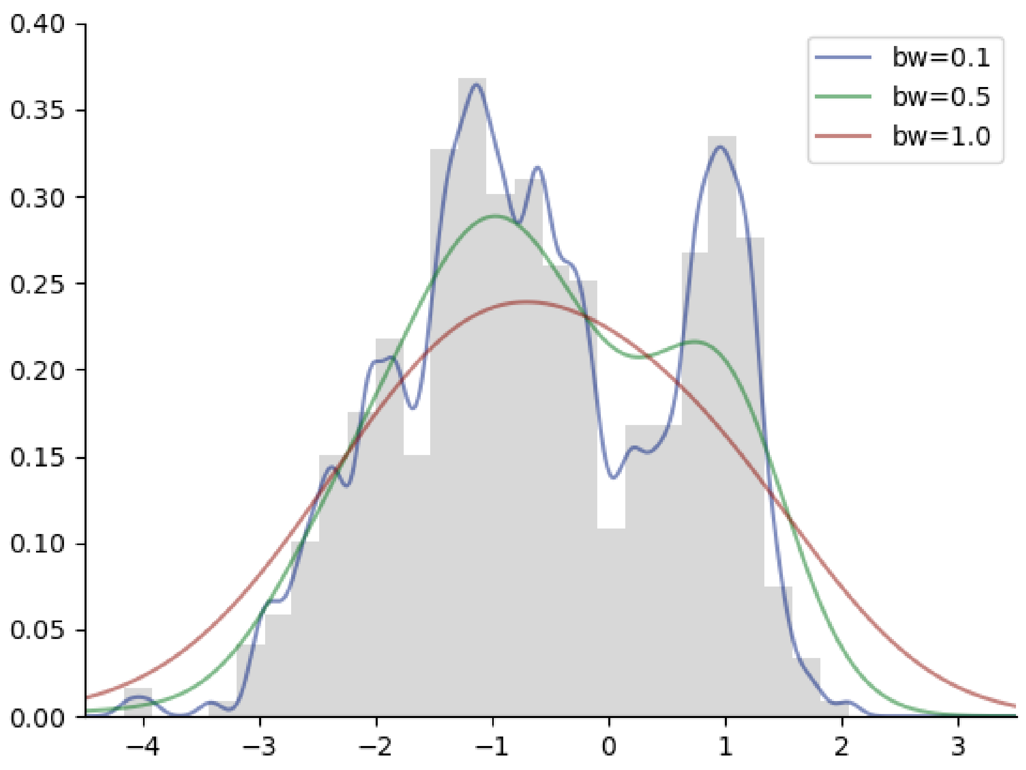


Figure 2.6: This figure illustrates how the bandwidth affects how smooth the resulting curve is. Using the same kernel (Gaussian), we plot the resulting curve from the KDE using bandwidth values of 0.1, 0.5 and 1. If we have a small bandwidth, we end up with a highly multimodal distribution.

As illustrated in Figure 2.6, the choice of a bandwidth is crucial to the performance of KDE. A narrow bandwidth can lead to over-fitting while a wide bandwidth might lead to under-fitting. Bandwidth selection is therefore an active area of research in statistics as reviewed in *Heidenreich et al.* (2013). The machine learning community however adopts a more empirical approach such as **cross validation** for finding the bandwidth. In cross validation, the data is divided into n mutually exclusive partitions of approximately equal size. The KDE is fitted to the points in $n-1$ partitions and tested on the remaining partition. The process is repeated n times and each time choosing a different partition for testing. The bandwidth is varied over a specified range and the bandwidth that produced the best fit is consequently chosen as the most suitable for the entire data.

2.4.1.1 Kernel Density Estimator for Anomaly Detection

To illustrate the use of KDE for anomaly detection, let's take some randomly generated data containing 3 clusters as shown in Figure 2.7.

The first step is to fit the KDE algorithm to the data points and return the log-likelihood of each sample under the model *Pedregosa et al.* (2011). All the points are ranked from the lowest log-likelihood and then a threshold is chosen for identifying outliers. The choice of threshold score is an active area of research (*Gao et al.*, 2011; *Latecki et al.*, 2007b,a; *Schubert et al.*, 2014) and more recently *Li et al.* (2021).

However, for this illustration, we assume that samples with log-likelihood score above the 95 percentile are outliers. Figure 2.7 shows a plot of sample outliers.

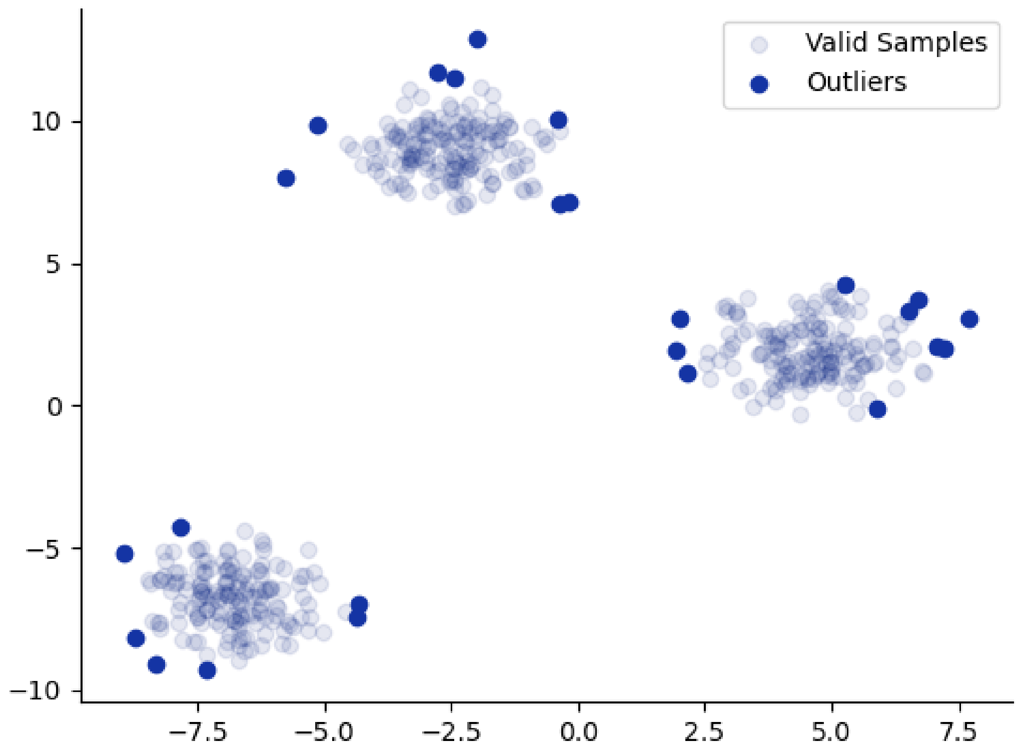


Figure 2.7: The outliers from our example are the data samples which are farthest from the center of the closest cluster. They are illustrated in the deep blue dots. The estimator assigns a higher likelihood to the clustered data points.

An even more interesting approach will be to vary the threshold to select the most interesting boundary for detecting anomalies. In many application domains, the choice of threshold is based on some domain knowledge about the task at hand.

2.4.2 Gaussian Mixture Model

In modelling the underlying distribution of observations, we often make simple assumptions such as all observations come from one distribution. From this assumption, we proceed to estimate the parameters (such as mean and variance) of this distribution. However, data in the real world is often more complex and might not hold for this assumption. For example, the data might be multi-modal. In such a scenario we might model the data as a mixture of several components (such as a Gaussian or a Poisson distribution). Here, we assume each observation belongs to several components and then proceed to infer which component it belongs to. A Gaussian Mixture Model (GMM) is the case where each component is modelled as a Gaussian distribution.

Formally, a distribution f is a mixture of $K - Component$ distributions $f_1, f_2 ... f_k$ if

$$f(x) = \sum_{k=1}^{K} \pi_k f_k(x) \tag{2.7}$$

where π_k is the mixture weight representing the probability that an observation x belongs to a $K - mixture$ component.

Unlike the KDE which puts a kernel on each point, a GMM fits a mixture of global kernels on the whole dataset. The parameters of the model are estimated using the Expectation Maximization (EM) technique. The EM technique consist of the expectation and maximization steps. The expectation step estimates the weights which describe the probability of each data point belonging to each of the clusters while the maximization step updates the parameters of each cluster based on all the data points. The expectation and maximization steps are repeated until the model converges, giving a maximum likelihood estimate of the data.

A data instance that exists in a "low-density" region is considered an anomaly. A region is considered low-density if it falls below a user-defined density threshold.

2.4.2.1 Applications of Gaussian Mixture Models for Anomaly Detection

Reddy et al. (2017) proposed a two-stage process for detecting anomalies in traffic networks using GMM. In the first step, a GMM is fitted to the training data. Anomalies are removed by examining the probability associated with each data point. In the second stage, the GMM is re-computed on the remaining historical data without the outliers. This model is used for detecting outliers on the test data.

Roberts and Tarassenko (1994) improved on choosing more robust threshold for novelty detection using a Gaussian mixture model. They achieved this by learning a representation of normality from the dataset using a GMM. When new data is presented, the previous threshold from training is used to define a novelty decision boundary. In an attempt to make the predictions of neural networks more robust, *Bishop* (1994) also used a GMM to estimate a novelty threshold for the trained model. At test time, any data that falls below this threshold is classified as a novelty.

2.5 Evaluation Metrics

Supervised anomaly detection can be viewed as a special case of classification. Due to this, it adopts classification based metrics for its evaluation. In this sub-section, we examine some of the commonly used metrics in classification. We also explain why certain metrics might not be suitable for anomaly detection.

In a supervised learning, the labelled data is divided into training and testing sets. A model is trained on the training set and then evaluated on the test set. The model predicts the labels of the test data. The predicted labels is compared to the known labels (ground-truth labels). This comparison is summarized in a **confusion matrix**.

2.5.1 The Confusion Matrix

The confusion matrix is a table that provides insights into the predictions of a model.

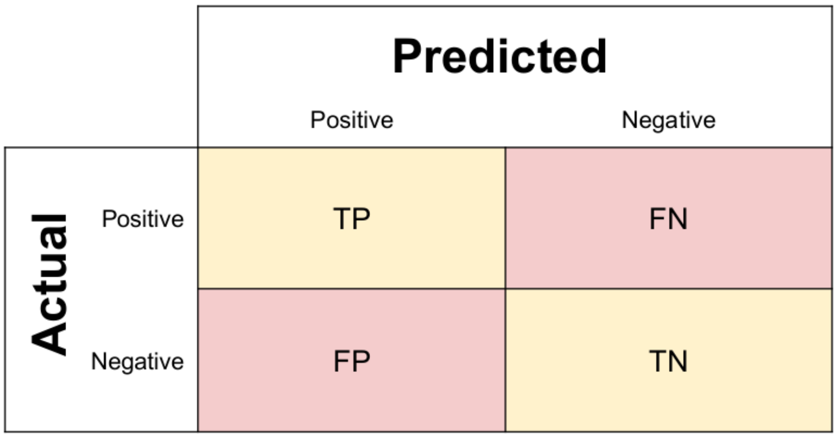


Figure 2.8: Illustrations of how various predictions are represented in a confusion matrix. The confusion matrix consists of 2 rows and 2 columns. The rows represent the actual class instances and the columns represents the model predicted classes.

To explain the various parts of Figure 2.8, it can be assumed we have a binary classification problem involving the detection of anomalies. We can deduce the following:

1. **Negative (N)**: A data instance is normal.
2. **Positive (P)**: A data instance is an anomaly.
3. **True positive (TP)**: With respect to the preamble above, it means that the model correctly predicted a data instance to be an anomaly.
4. **True negative (TN)**: The model correctly predicted that a data instance is

normal.

5. **False negative (FN)**: The model incorrectly predicted an anomaly data instance to be normal.

6. **False positive (FP)**: The model incorrectly predicted a normal data instance to be an anomaly.

Depending on the use-case, different metrics can be deduced from the confusion matrix.

2.5.1.1 Accuracy

The most straightforward metric to deduce from the confusion matrix is the accuracy. The accuracy is defined as the number correctly predicted data instances divided by the total number of predicted instances. It is expressed as:

$$\mathrm{Accuracy} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{2.8}$$

Accuracy does not account for False Positives and the False Negatives. It also assigns equal cost to the True Negatives and the True Positives. In scenarios where the data is imbalanced, a classifier predicting every data instance to be in the majority class will have a very good accuracy score. This is however not optimal since it misses all the data instances in the minority class.

2.5.1.2 Recall

Recall is the fraction of True Positives divided by the total number of data instances predicted to be positive.

$$\mathrm{Recall} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{2.9}$$

The recall score reveals the ability of the classifier to find all the positive samples.

2.5.1.3 Precision

It is the number of true positives and the false positives.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.10}$$

The precision score indicates the ability of the classifier not to label as positive a sample that is negative.

2.5.1.4 Matthews Correlation Coefficient

The Matthews Correlation Coefficient (MCC) offers a balanced measure by taking into account all the quadrants of the confusion matrix. It ranges from -1 and 1.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{2.11}$$

An MCC of 1 indicates a perfect prediction, 0 an average random prediction and -1 indicates total disagreement between the predicted labels and the true labels.

2.6 Active Learning

2.6.1 Introduction

This thesis explores the use of active anomaly detection in detail hence we will give a brief overview of the topic. We will discuss various Active Leaning Scenarios and query strategies..

In a supervised learning setup, labelled data is required to train a model. Sometimes, these labels might be expensive to obtain. Most real world data however comes unlabelled. Active learning seeks to address this challenge. It works on the assumption that learning algorithms will perform better if they are allowed to choose the data from which they learn.

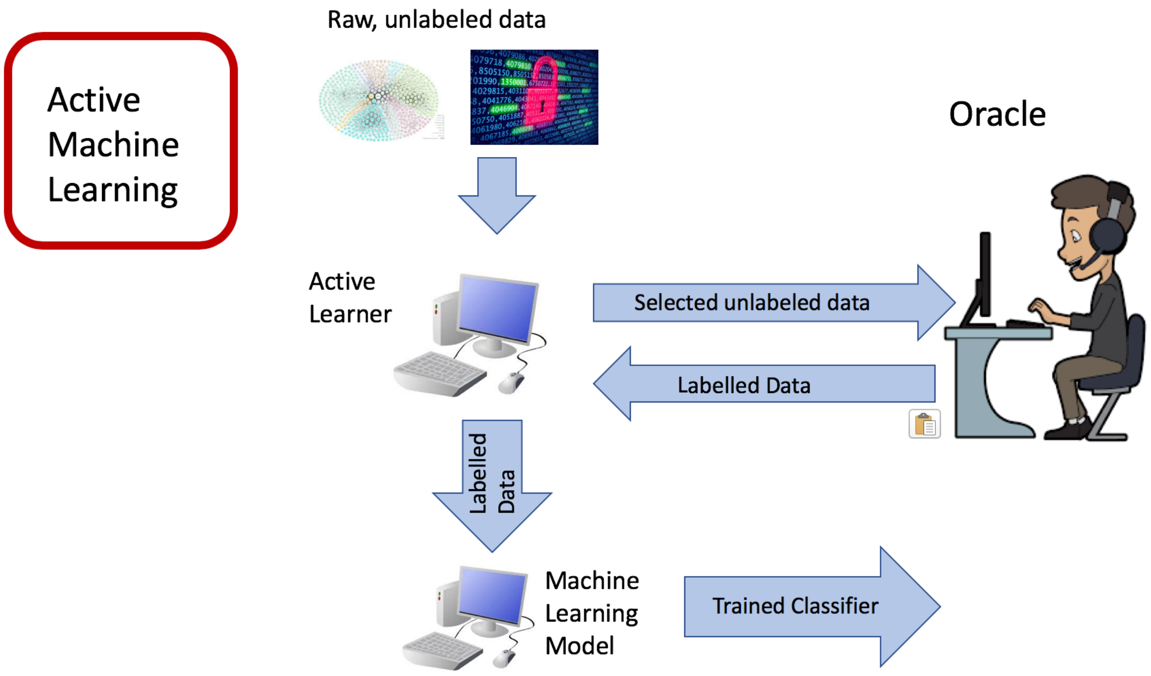


Figure 2.9: Flow process describing how Active Learning integrates with machine learning. A machine learning model is trained with a small number of labelled instances. The model is used to predict the labels of some unlabelled data. An Active Learning query strategy is used to select the most informative predicted instances to be shown to the Oracle for classification.The instances labelled by the Oracle are then added back to the original training set. The setup proceeds to train the machine learning model in the standard supervised way. The process is repeated over several rounds (*Zhao*, 2020).

In Active Learning systems, Oracles are presented with queries for annotation over several rounds of training. These queries often come in the form of unlabelled data instances as shown in Figure 2.9. Active Learning algorithms are evaluated by plotting some evaluation metric of interest (e.g., accuracy) as a function of the number of new queries that are labelled and added to the training data. This curve is compared to a random baseline such as a traditional passive supervised learning model.

2.6.2 Active Learning Scenarios

In Active Learning the algorithm tries to select a small amount of data for the Oracle to label that are likely to give the biggest improvement in the machine learning model. Active Learning Scenarios describe the way in which the data is selected to be shown to the Oracle. In all these Scenarios, it is assumed that the queries take the form of unlabelled instances to be labelled by the Oracle. There are 3 main Active Learning Scenarios namely (*Settles et al.*, 2008):

1. **Membership Query Synthesis**: This describes a scenario in which the learner generates its own query instances from an underlying distribution. It was first introduced in *Angluin* (1988). This is often used when there is a scarcity of unlabelled instances (*Wang et al.*, 2015).

   One drawback with this approach is that, the generated instances might not contain any semantic information. This might increase the time it takes for the Oracle to identify informative instances (*Lang and Baum*, 1992; *Zhu and Bento*, 2017; *Huijser and van Gemert*, 2017);.

2. **Stream-Based Selective Sampling**: In this scenario, the data instances are examined one at a time. The model selects which instances to display based on some "informativeness measure" or "query strategy". The Oracle assigns a label to the chosen instance.

3. **Pool-Based Sampling**: In this scenario, queries are drawn from a pool of unlabelled data based on some "informativeness measure". Unlike the stream-based sampling which scans the data sequentially and makes individual query decisions, pool-based sampling ranks the available unlabelled data before making a query decision.

2.6.3 Query Strategies

Query strategies in Active Learning describe ways in which the informativeness of unlabelled instances are evaluated. Several query strategies have been extensively studied. We examine a non-exhaustive list of some of these strategies below:

1. **Uncertainty Sampling**: Uncertainty sampling was first introduced in (*Lewis and Gale*, 1994). In this approach, the model queries instances which it is least

certain about.

2. **Expected Model Change**: The model selects the instance that will impact the greatest change to the model if its label was known. An example query strategy that uses this approach is the "expected gradient length" (EGL) (*Settles et al.*, 2008). EGL queries a data instance which when added to the training set, will result in the largest magnitude of the training gradient of the parameters of the model. EGL can be applied to any model trained using gradient based techniques. Intuitively, this approach queries a data instance that is likely to have the greatest impact on the parameters of the model.

3. **Query By Committee**: Query by committee strategy trains different models on the currently available data (*Seung et al.*, 1992). The different models represent competing hypotheses. The instances on which they most disagree the most are selected for the Oracle to classify.

2.6.4 Active Anomaly Detection

Active Learning has seen many different applications in anomaly detection. It is commonly used in domains where the percentage of anomalies in the dataset is extremely rare (rare-category detection) or in situations where anomalies are difficult to distinguish from the normal class.

One of the earliest application was by (*Pelleg and Moore*, 2004). They applied Active Learning to rare category detection in Astronomy and reported significant improvements over other methods. *Pimentel et al.* (2020) also introduced an Active Learning layer in unsupervised anomaly detection.

More recently, *Lochner and Bassett* (2021) also applied Active Learning for detecting rare or unknown astrophysical phenomena. They tested their framework on the Galaxy zoo dataset (*Willett et al.*, 2013) and reported that it doubles the number of interesting anomalies shown to the Oracle.

Generally, most Active Learning approaches follow algorithm 1 as shown in *Lochner and Bassett* (2021) and (*Pelleg and Moore*, 2004):

Algorithm 1 Active anomaly detection algorithm

```
Input:
  $\mathtt{epochs} \leftarrow$ number of epochs for training model
  $D_{\mathtt{training}} \leftarrow$ labelled training data
  $D_{\mathtt{unlabelled}} \leftarrow$ unlabelled pool of data
  $\mathtt{labels} \leftarrow$ Labels of training data
  $\mathtt{Oracle} \leftarrow$ A domain expert
  $n \leftarrow$ number of examples chosen to be shown to the Oracle per round
  $\mathtt{rounds} \leftarrow$ number of rounds to go through the unlabelled dataset
  procedure ActiveAnomalyDetection(epochs, $D_{\mathtt{training}}, D_{\mathtt{unlabelled}}, \mathtt{labels}, \mathtt{Oracle}, n, \mathtt{rounds}$)
    while $i < \mathtt{rounds}$ do
        $\mathtt{model.train(epochs}, D_{\mathtt{training}}, \mathtt{labels})$
        $\mathtt{topAnomalies} \leftarrow \mathtt{model.getTopAnomalies}(D_{\mathtt{unlabelled}}, n)$
        $\mathtt{oraclesSelection} \leftarrow \mathtt{Oracle.select(topAnomalies)}$
        $D_{\mathtt{training}} \leftarrow D_{\mathtt{training}} \bigcup \mathtt{oraclesSelection}$
        $i \leftarrow i + 1$
    end while
```

2.7 Summary

In this chapter, we examined various traditional anomaly detection algorithms and some evaluation metrics applied in classification. We also explored Active Learning and how it is applied to anomaly detection.

Deep Learning has shown state-of-the-art performance in many Computer Vision and Natural Language tasks. Their ability to learn relevant information from data without handcrafted features makes it a technique worth exploring for our use-case.

The next chapter will focus on deep learning and their application in anomaly detection.

3 Deep Learning for Anomaly Detection

3.1 Chapter Introduction

In the last chapter, we discussed various classical anomaly detection algorithms. This chapter discusses deep learning and its application to anomaly detection.

Deep learning has shown impressive results in several application domains. Architectures such as Convolution Neural Networks (CNNs) and Transformers (*Vaswani et al.*, 2017) have achieved state-of-the-art results in computer vision (*He et al.*, 2015a) and Natural Language Processing tasks respectively.

Our proposed algorithm, **Ahunt**, uses CNNs to deform the raw image features for anomaly detection hence we dedicate a subsection of this chapter to discuss CNNs.

This chapter also introduces other commonly used architectures in anomaly detection such as Recurrent Neural Networks, Autoencoders and Generative Adversarial Networks.

Finally, we explore the specific application of these architectures in anomaly detection.

3.2 Introduction to Neural Networks

3.2.1 A brief history of Neural Networks: from McCulloch-Pitts to ImageNet

Warren McCulloch and Walter Pitts developed the first mathematical model of an artificial neuron. In their paper, 'A Logical Calculus of the Ideas Immanent in Nervous Activity' (*Mcculloch and Pitts*, 1943), they stipulated that a biological neuron can be represented computationally by the addition and thresholding of input signals as shown in Figure 3.1.

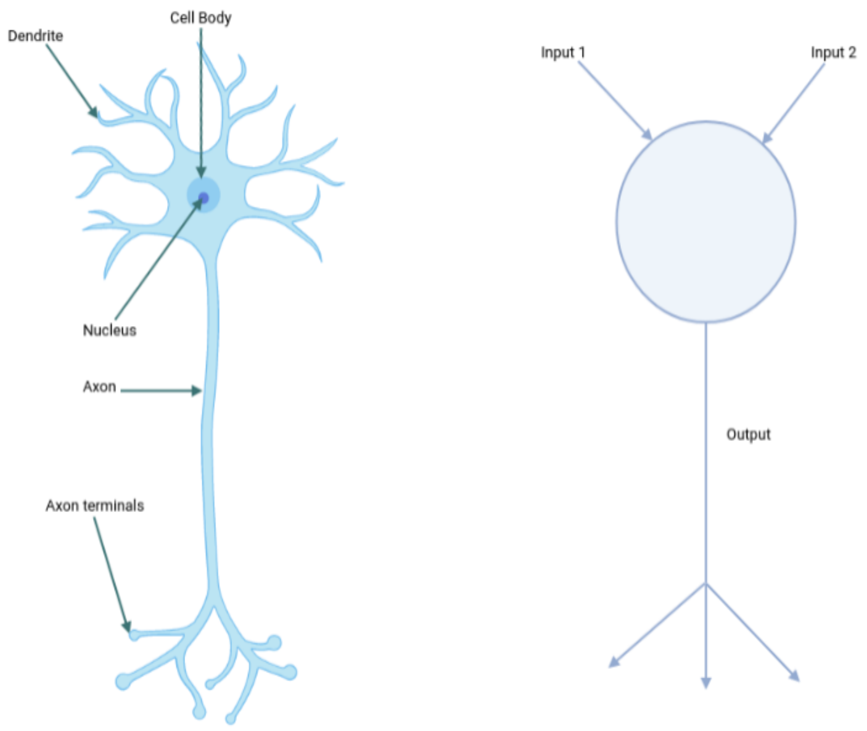


Figure 3.1: Diagram of a biological neuron (left) and a its mathematical representation (right). The dendrites receive signals which is comparable to how the input neurons in the mathematical cells also input data. Adapted from *Howard and Gugger* (2020).

Frank Rosenblatt built the Mark I perceptron in 1958. The device was based on principles developed by Warren and Pitts in *Mcculloch and Pitts* (1943). The Mark I perceptron, in Rosenblatt's words was supposed to be "a machine capable of perceiving, recognizing and identifying its surroundings without any human training or control" (*Frank Rosenblatt et al.*, 1956). Even though the device did not live up to this goal, it had the ability to recognize simple shapes. During this period, Marvin Minsky and Frank Rosenblatt publicly debated the merits of Rosenblatts goals with the Mark I perceptron. Marvin Minsky argued that the functions based on which the device was built was too simplistic for the rather grandiose goal set out by Rosenblatt.

In 1969, Marvin Minsky and Seymour Papert wrote their seminal book "Perceptrons: An Introduction to Computational Geometry" (*Minsky and Papert*, 1969). They proved that a single layer of a Perceptron does not learn simple mathematical functions such as the XOR function. This led them to conclude that, although Perceptrons and their extensions were interesting, research in that direction was generally an effort in futility.

The conclusions from *Minsky and Papert* (1969) slowed down research into Perceptrons until 1986 when the Parallel Distributed Processing research group introduced a computational framework for modelling cognitive processes (*Rumelhart et al.*, 1986b). Instead of modelling cognition as logic gates as seen earlier from *Mcculloch and Pitts* (1943) to *Minsky and Papert* (1969), they suggested a more nuanced approach which can be summarized as follows *Rumelhart et al.* (1986b):

- A set of processing units with an output function for each unit and a pattern of connectivity among units.
- A set of learning and propagation rules for learning patterns by experience and propagating patterns of activities through the network, respectively.
- A defined system with which the learning occurs. We can refer to this as the environment.
- A state of activation.

Rumelhart et al. (1986b) also introduced the use of the backpropagation algorithm for training neural networks. The backpropagation algorithm calculates the gradient of the network's error with respect to the parameters of the network. Conceptually, it finds how the parameters should be tweaked so as to reduce the error. We discuss the algorithm in further details in subsection 3.2.5. Even though backpropagation was useful in training neural networks (*LeCun et al.*, 1989), it was computationally expensive to compute the gradients of the layers in the network. This meant that it didn't scale with respect to the available computational resources at the time. Most researchers therefore abandoned the neural network approach for algorithms such as Support Vector Machines (*Boser et al.*, 1992) which had lower computational requirements.

The early 2000s gave rise to the adoption of Graphical Processing Units (GPUs) for general purpose computing with researchers in Stanford university discussing the use of GPUs in machine learning applications (*Raina et al.*, 2009). This coalesced with large volumes of data in domains such as computer vision (*Deng et al.*, 2009) rejuvenated interest in deep learning research.

The recent widespread adoption of deep learning happened after Alex Krizhevsky (*Krizhevsky et al.*, 2012) won the ImageNet (*Deng et al.*, 2009) in 2012. Their model achieved an error rate of 16% as compared to the previous year's 28%. Since then, several variations of deep learning architectures have shown impressive performance in domains such as Computer Vision and Natural Language Processing.

3.2.2 Feedforward Neural Networks

A feedforward neural network approximates a function, f that maps input data to some defined outputs. Using classification as an example, assuming we want to find a function $y = f(x)$ that maps x to a corresponding label y, a feedforward neural networks defines the mapping from x to the label $y = f(x : w)$ by learning the value of the parameter w that best approximates the function.

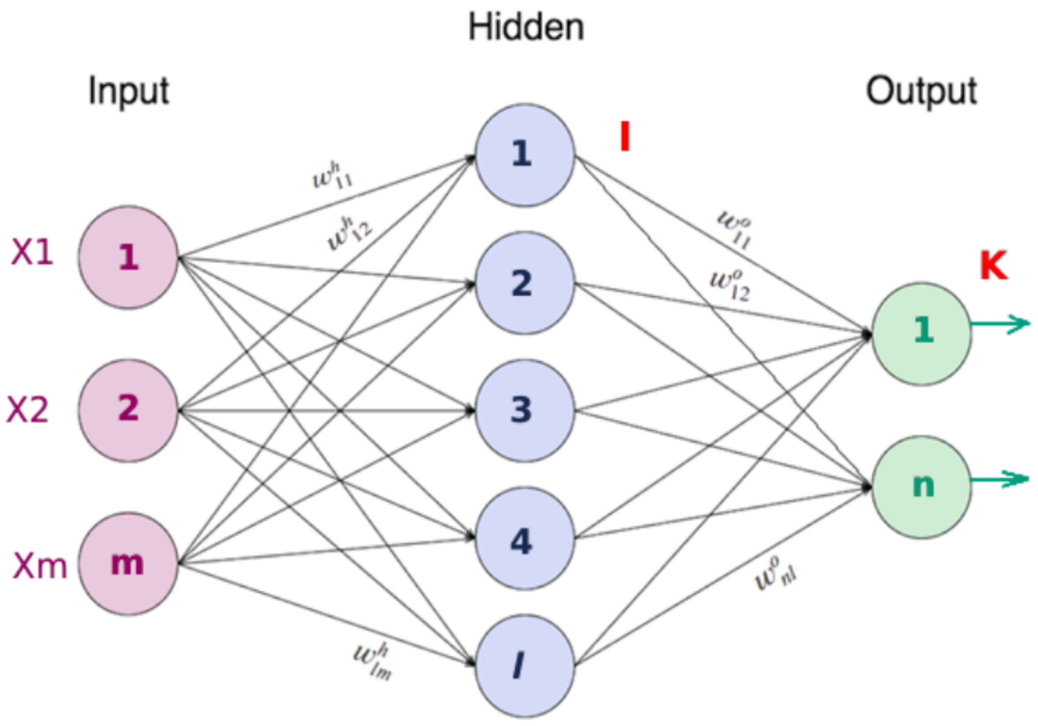


Figure 3.2: Diagram of a feedforward neural network showing the neurons (the colored circles). The neurons are organized into layers with the red circles representing the input layer, the blue representing the hidden layer and the green representing the output layer. This particular illustration is consists of a fully connected layer. It is referred to as fully connected because each neuron in the input layer is connected to every neuron in the hidden layer and each neuron in the hidden layer is also connected to all the neurons in the output layer (*Chodey and Shariff*, 2021).

A feedforward neural network is the first type of neural network (*Schmidhuber*, 2014). It consists of basic neuron-like processing units which are often called nodes. It receives data via the input nodes as shown in Figure 3.2, passes it through the hidden layers and finally to the output layer. The dimensionality of the hidden layer determines the width of the network while the number of hidden layers determines its depth.

Each neuron, as shown in Fig. 3.3, performs a weighted summation of its input and passes the value to an activation function. Activation functions are usually non-linear continuously differentiable functions. In theory (*Hornik et al.*, 1989), a two-layer neural network with a non-linear activation function can approximate any function given a sufficient number of units in the hidden layer.

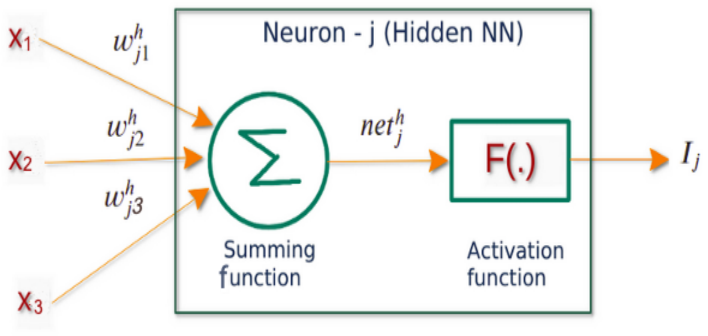


Figure 3.3: A Single Neuron showing some input features (x_1, x_2 and x_3) with their respective weights. The weighted sum of these inputs is passed to an activation function as indicated in the diagram.

Feedforward neural networks, just like other types of neural networks, learn by iteratively updating the weights by minimizing a loss function $L(y, \hat{y})$. Using backpropagation (*Rumelhart et al.*, 1986a), the gradient of each parameter, θ, with respect to the loss function is computed using the chain rule.

Feedforward neural networks are limited in their ability to capture interactions in data points which are related in time, eg. video data. This is addressed using recurrent neural networks. We will examine this in a later section.

3.2.3 Activation Functions

A neural network is composed of functions from different layers where the output of a previous layer is used as input in the next layer. As discussed in 3.2.2, each neuron performs a weighted sum of its inputs before passing it to an activation function. Activation functions introduce non-linearity into the neural network. Without the activation function, the network will be a linear function with a linear decision boundary. This is undesirable since most problems have complex decision boundaries.

While there isn't a general consensus on what makes an activation function desirable for deep learning, most activation functions used have the following properties:

1. They need to be non-linear. The universal approximation theorem states that, in theory, given a sufficient number of units in the hidden layer, a neural network can approximate any function (*Hornik et al.*, 1989).
2. They should be continuously differentiable. This property allows backpropagation and other gradient-based methods to find the optimal weights that minimizes the loss function. Gradient-based methods achieve a stable performance making them desirable in deep learning. It is important to note that, not all activation functions used in deep learning are continuously differentiable. A typical example is ReLU which is differentiable at all points except at 0.

The following section examines some commonly used activation functions.

3.2.3.1 Softmax

The softmax activation function, is typically used in the output layer of the neural network in a multi-class classification setting. It normalizes a vector with K elements into a probability distribution over the elements. Its output therefore can be interpreted as the probability of each class.

The softmax activation function is given by:

$$\mathrm{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{3.1}$$

3.2.3.2 Sigmoid

The sigmoid activation function has an "S" shape as shown in Figure 3.4. It returns values between 0 and 1 for each class label. This makes it suitable for multi-label classification problems where the labels are not mutually exclusive.

The function is expressed as:

$$f(x) = \frac{1}{1 + \exp(x)} \tag{3.2}$$

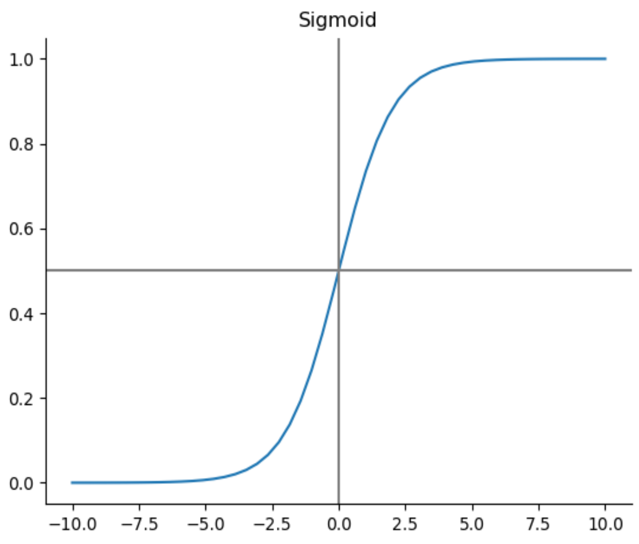


Figure 3.4: The Sigmoid Activation Function is **S** shaped with output values ranging from 0 to 1.

3.2.3.3 Rectified Linear Unit

Rectified linear unit (ReLU) is one of the most widely used activation functions. It is more computationally efficient than the Sigmoid and the Softmax activation functions. Its computational efficiency stems from the ease of computing its derivatives. This property helps the network train faster. The ReLU is mathematically expressed as:

$$f(x) = max(0, x) \tag{3.3}$$

As seen in Figure 3.5, for activations in the negative region of the ReLU, the gradient will be zero. This means that gradient descent will not alter the weights in this region hence there will be no learning. This phenomena is called the dying ReLU problem. The dying ReLU problem broadly relates to the vanishing gradient problem which occurs when gradients get smaller as the algorithm approaches the earlier layers (layers closer to the input layer). As a result, the weight update in the early layers stall, preventing the model from converging to an optimal solution.

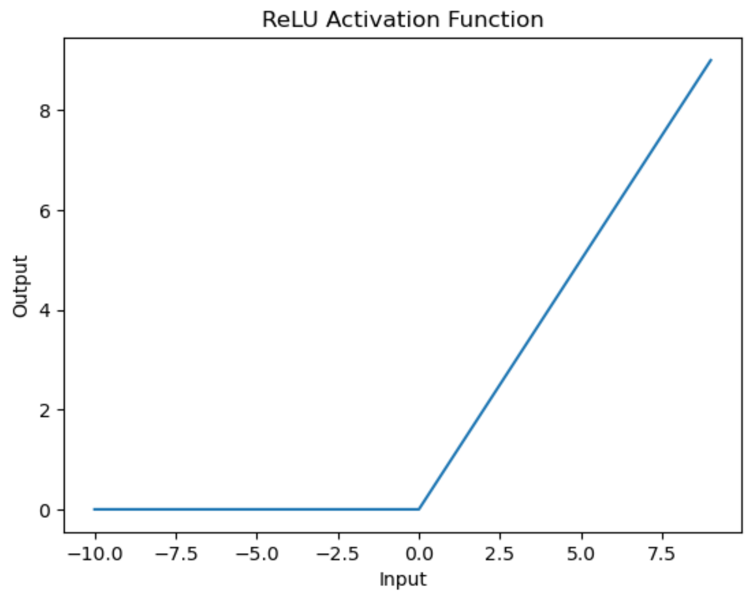


Figure 3.5: Illustration of the shape of the ReLU Activation Function. For activations in the negative region, the gradient will be zero. This leads to the dying ReLU problem.

3.2.3.4 Leaky ReLU

The leaky ReLU (*Maas et al.*, 2013) addresses the dying ReLU problem by adding a slight slope in the negative region as indicated in Fig. 3.6.

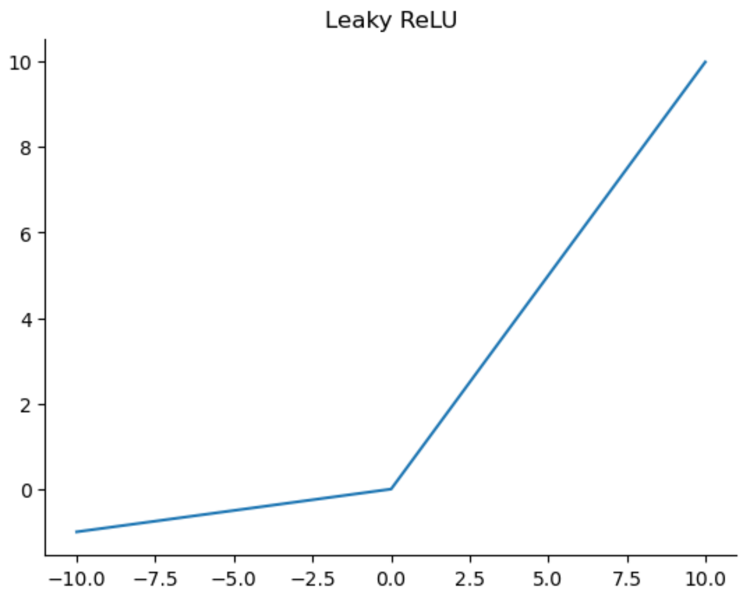


Figure 3.6: The small negative slope in the Leaky ReLU helps prevent the dying ReLU problem.

It is defined as :

$$f(x) = \begin{cases} x & \text{if } x > 0, \\ 0.01x & \text{otherwise.} \end{cases} \tag{3.4}$$

Although leaky ReLU is shown to theoritically solve the dying ReLU problem, results from *Pedamonti* (2018) shows that, its performance is similar to that of ReLU on the MNIST dataset. Other ReLU activation functions such as the Parametric ReLU and the Exponential ReLU can also be used to mitigate the dying ReLU problem.

3.2.3.5 Hyperbolic Tangent

The hyperbolic function is also known as tanh. It takes any real-valued input and returns values between (-1,1) range. As shown in Fig.3.7, it is very similar to the Sigmoid function.

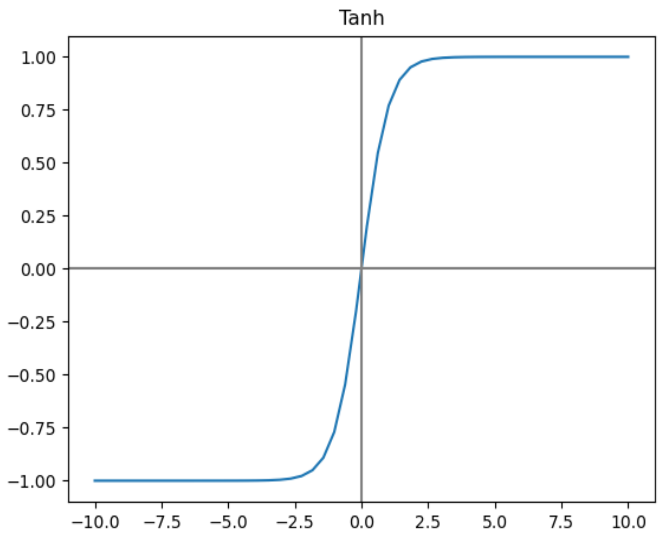


Figure 3.7: The hyperbolic tangent is very similar to the Sigmoid except that its middle is more sharply defined.

The hyperbolic tangent is mathematically expressed as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3.5}$$

3.2.4 Loss Functions

Loss functions help to estimate errors after a forward pass. It does this by comparing the predictions from the forward pass with the actual ground truth targets. In this section, we examine various loss functions applied in machine learning.

3.2.4.1 Mean Absolute Error

The Mean Absolute Error (MAE) calculates the average of absolute distance between the predicted and the true value. It is also referred to as the L1 loss. It is expressed as:

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y}_i|}{N} \tag{3.6}$$

where N is the number of samples in the dataset, $\hat{y}_i$ is the predicted value and y_i is the true value.

The MAE is often used in regression problems. MAE is less sensitive to outliers when compared to the Mean square error (MSE). This is because it does not square the errors as done by the MSE.

3.2.4.2 Mean Square Error

The Mean square error (MSE) calculates the square of the difference between the predicted values and the ground truth outputs. It is also called the L2 loss. It is sensitive to outliers given that it squares the errors. The MSE is also commonly used in regression problems. It is expressed as:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \tag{3.7}$$

where N is the number of samples in the dataset, $\hat{y}_i$ is the predicted value and y_i is the true value.

3.2.4.3 Root Mean Square Error

The Root Mean Squared Error (RMSE) is the square root of the Mean Squared Error. It is expressed as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{3.8}$$

Just like the MSE and the MAE, it is also used in regression problems. The root mean squared error offers better interpretation of the errors. It is the average distance of a point from a fitted line along the vertical axis.

3.2.4.4 Cross Entropy Loss

Cross-entropy loss is mostly used in binary classification problems. Its output is between 0 and 1. The cross-entropy is given as:

$$L(\hat{y}_i, y_i) = -\sum_{i=1}^{N} y_i \ln \hat{y}_i + (1 - y_i) \ln (1 - \hat{y}_i) \tag{3.9}$$

where N is the number of samples in the dataset, $\hat{y}_i$ is the predicted value and y_i is the true value.

A variant of the cross entropy can also be used for multi-class classification. It is called the multi-class cross entropy loss. It is represented as:

$$-\sum_{i} y_i \ln \hat{y}_i \tag{3.10}$$

3.2.5 Backpropagation and Gradient Descent

Backpropagation was invented in the 1970s but was popularized by Rumelhart, Hinton and Williams in their paper, "Learning representations by back-propagating errors" (*Rumelhart et al.*, 1986a).

It uses the chain rule to calculate the gradient of the loss function with respect to the parameters of the network. This tells us how a small change in the parameters will affect the total loss of the network. From this, the parameters are then adjusted using gradient descent.

The loss surface of most deep neural networks have multiple local minima. The network might therefore get trapped in one of its many local minima and hence negatively affect its ability to generalize to unseen training examples. This is counter-intuitive to the surprisingly superior performance of networks trained with gradient descent. *Dauphin et al.* (2014) argued that the local minima is less of a problem than the saddle points and that gradient descent finds it difficult to efficiently escape the saddle points. *Jin et al.* (2017) proved that gradient descent augmented with suitable perturbations escapes saddle points efficiently.

Gradient descent is a common approach used in optimizing neural networks. There are three main variants of gradient descent used in training neural networks. The variation comes from how much data is use to compute the gradient of the loss function. In most cases, there is a trade-off between the accuracy of the parameter update and the time it takes to perform the update. The three variants of gradient descent include:

1. **Batch Gradient Descent:** This is the original gradient descent algorithm otherwise known as the vanilla gradient descent. For an objective function $J(\theta)$, with model parameters $\theta \in \mathbb{R}^d$, the batch gradient descent computes the gradients with respect to the entire dataset:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{3.11}$$

   The need to compute the gradient of the entire dataset in each update makes batch gradient descent very slow. Given the large size of most datasets used in deep learning, this method requires large amounts of memory to be feasible. These two problems makes the batch gradient descent a non-viable option in many practical settings

2. **Stochastic Gradient Descent:** Stochastic Gradient Descent (SGD) computes the gradient and updates the parameters with respect to each training example x_i and label y_i:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x_i, y_i) \tag{3.12}$$

   This makes it more computationally efficient. However, the updates become more frequent with a high variance causing the objective function to fluctuate heavily. Given the frequency of the updates, SDG is more likely to overshoot an update hence might not converge at an exact minima. This problem can be mitigated by slowly reducing the learning rate over time.

3. **Mini-batch Gradient Descent:** Mini-batch gradient descent (MBGD) combines the best of vanilla gradient descent and SDG. It performs an update for every mini-batch in the training set. For n training examples, MBGD is given by:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x_{i:i+n}, y_{i:i+n}) \tag{3.13}$$

   where x_i and y_i represent the training data and its labels respectively.

   Given that MBGD leverages the mini batches, its updates are less frequent than SDG but more frequent than vanilla gradient descent. This reduces the variance and provides a better trade-off on speed.

3.3 Architectures for Deep Learning

3.3.1 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are networks that utilize the convolution operation in its layers. CNNs have its roots in the work of David Hubel and Torsten Wiesel. In experiments extending over 25 years, they studied the impacts of vision impairment in humans using kittens (*Hubel*, 1964). They discovered that humans use simple and complex cells for visual recognition. The simple cell recognizes things such as vertical edges in an image and the complex cells recognizes more complex shapes while showing spatial invariance. The complex cells achieve this by summing the information from the simple cells. Inspired by this work, *Fukushima* (1988) built the first artificial neural network that mimics the simple and complex cell structure. He used this network for digit recognition. *Lecun et al.* (1998) was the first to build a multilayered CNN. This network was used in the now famous image recognition task, MNIST (*LeCun et al.*, 2010).

CNNs shot to popularity when Alex Krizhevsky used a variant of it, (*Krizhevsky et al.*, 2012), to win the ImageNet competition (*Deng et al.*, 2009) in 2012. Since then, different variations of CNNs have produced exceptional performance on the ImageNet competition as shown in Figure 3.8. CNNs achieve exceptional performance by exploiting the spatial information in the data. This makes them particularly useful in visual tasks such as object and video recognition.

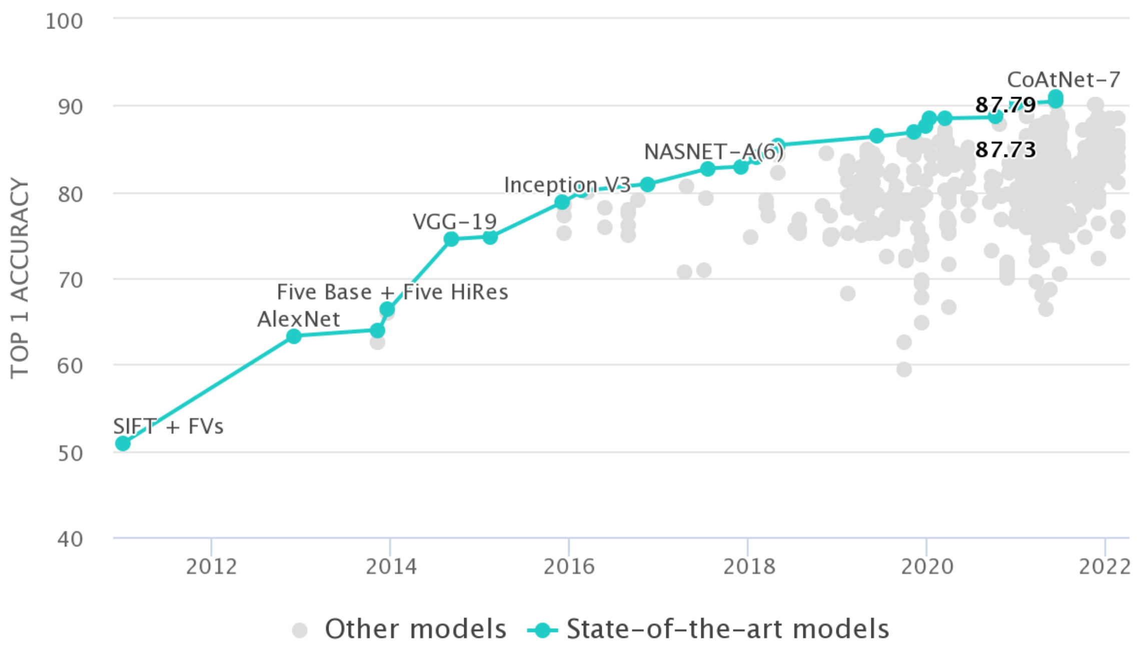


Figure 3.8: Performance improvement of different variations of CNNs on the ImageNet dataset over the past decade.The top 1 accuracy considers the model's prediction with the highest probability as the expected answer. This is the conventional notion of accuracy. Generated from *Papers with code*.

3.3.1.1 Convolution Operation

The convolution operation is the basic building block of the of CNNs. The convolution (continuous) operation are expressed as:

$$s(t) = \int I(a)K(t-a)dt \tag{3.14}$$

where I is the input and K is the kernel. However, in most CNNs, discrete convolutions are used. A one-dimensional discrete operation is expressed as:

$$s(t) = \sum_{a} I(a)g(t-a) \tag{3.15}$$

where K is a kernel and I is the input. Convolving the input and the kernel results in an output referred to as the feature map.

The convolution computes how similar a portion of an input is with the kernel. It does this by systematically sliding the kernel over various parts of the input. The parts of the input which is most similar to the kernel returns the highest value. Conceptually, we can think of the kernel as searching for similar patterns at different parts of the input.

3.3.1.2 Filters

Convolutional filters are use to extract features from the inputs. The name filters is used interchangeably with kernels. They are a matrix of integers applied across the entire image during the convolution operation. The width of the network is determined by the size of the convolution kernel. From *Krizhevsky et al.* (2012) to *He et al.* (2015a) in the ImageNet competition, there has been a general trend of stacking smaller kernels to achieve deeper networks rather than using large kernel sizes (bigger width of network).

As explained in *Eldan and Shamir* (2015), although wider networks learn the underlying features in the input data, they tend to overfit the data. This negatively impacts their ability to generalize. However, deeper networks capture intermediate features in different layers of the network. This gives rise to richer features which improves performance on downstream tasks. It is important to note that, the description above is specific to CNNs applied to vision problems.

3.3.1.3 Pooling Operation

The pooling operation involves down-sampling of the feature map. It reduces the parameters of the model thereby reducing the risk of over-fitting. Conceptually, pooling helps to extract the most useful information from localized sections of an input. The pooling operation helps with the following:

1. It helps to reduce the dimension of the feature maps. This reduces the number of parameters in the network, lessens the computational cost and reduces over-fitting.
2. It summarizes the features in the local regions of the input.

The two most commonly used pooling operations are Max Pooling (*Weng et al.*, 1992) and Average Pooling (*Lecun et al.*, 1998). As seen in Fig.3.9, the Max Pooling operation selects the highest value from a localized region whereas the Average Pooling operation finds the average of the values in each localized region. Max Pooling therefore selects the prominent features whereas Average Pooling provides a smoothing effect.

3.3.1.4 Stride

The stride is the number of unit pixel shifts by the kernel over the input. The default stride in many applications is 1. Choosing larger strides may save computational cost and also increase the generalization ability of the network. This however comes at the cost of missing out on important features in the input data.

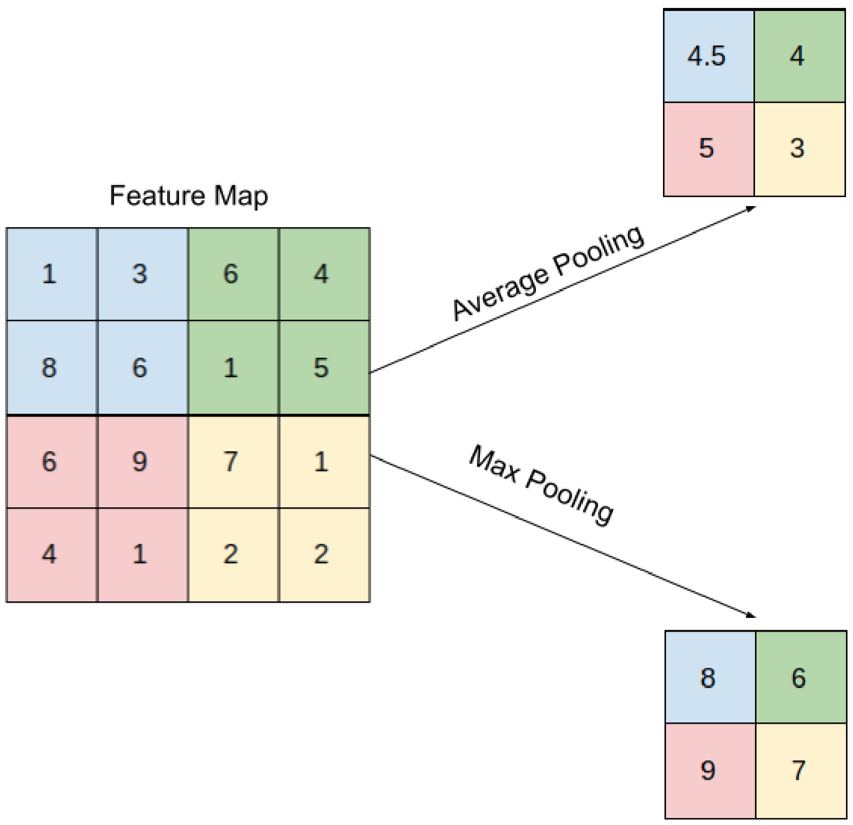


Figure 3.9: This image shows a pooling operation with a stride of 2. The Max Pooling operation selects the highest value in a region whereas the Average Pooling averages all the values in the region.

3.3.2 Visualizing Convolutional Neural Networks

The core of this thesis uses CNNs to dynamically learn features for anomaly detection. It is therefore prudent to understand how CNNs learn features for different types of downstream tasks (classification, object detection, segmentation, anomaly detection etc).

One commonly used method for visualizing CNNs is to cast the feature maps back into their original input images (*Zeiler and Fergus*, 2013). This is done using a deconvolutional network (*Zeiler et al.*, 2011). A deconvolutional network performs the same operations like pooling and filtering as CNNs but in reverse so instead of mapping the input to a feature map, it maps the feature maps back to the input space.

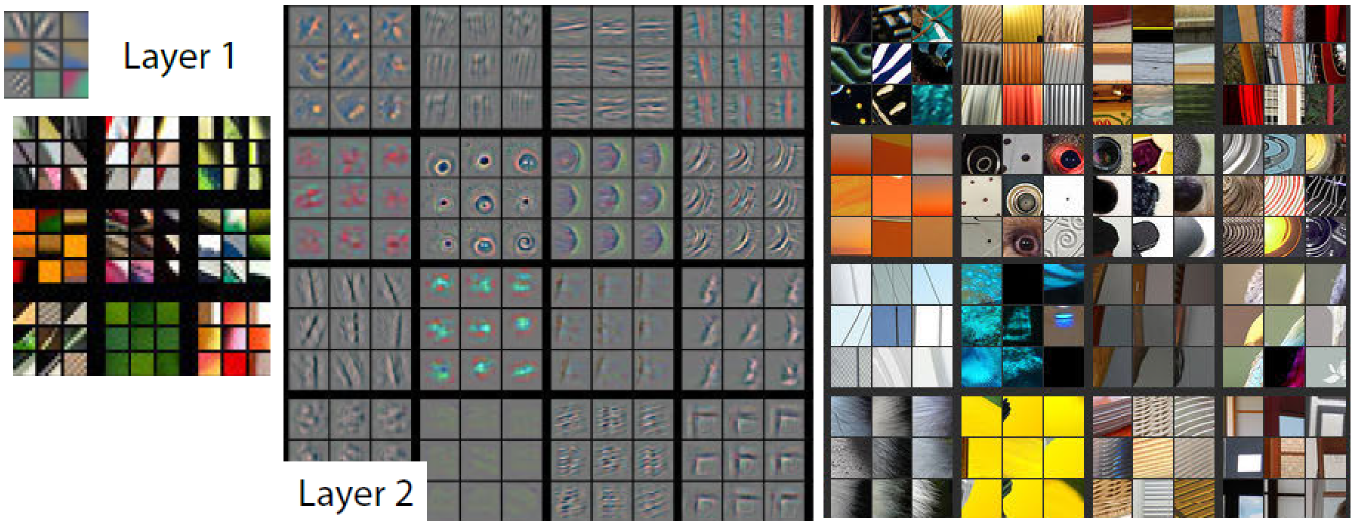


Figure 3.10: Visualization of the first two layers of trained CNN model. In layer 1, the models learns to identify simple lines and color gradients ranging from blue to yellow in some cases an red to green in others. Using the feature map from layer 1, layer 2 learns slightly more complicated shapes such as curves and circles (*Zeiler and Fergus*, 2013).

To visualize the features learnt in each layer of a CNN, a deconvnet is attached to each of the layers in the network. An input image is passed to the trained network and the feature maps are computed for each layer. To examine a given activation in a layer, all other activations in that layer is set to 0 and the feature map is passed to the decovnet to reconstruct the input image (*Zeiler and Fergus*, 2013).

Figures 3.10 and 3.11 shows the increasingly complex representations learnt by various layers in the CNN. This idea forms the foundations for reusing trained models for other tasks different from what the model was originally trained on. This process is called transfer learning. The assumption is that, since the simple features learnt in the earlier layers are the similar for similar datasets, we can freeze these layers and only train on the deeper layers which captures specific details about the dataset under consideration. Here, it is important to note that, the original dataset on which the model was trained must be semantically similar to the subsequent dataset used in transfer learning. For example, a model trained on dogs and cats might perform poorly when used on x-ray images.

This process of gradually learning semantically useful features for downstream

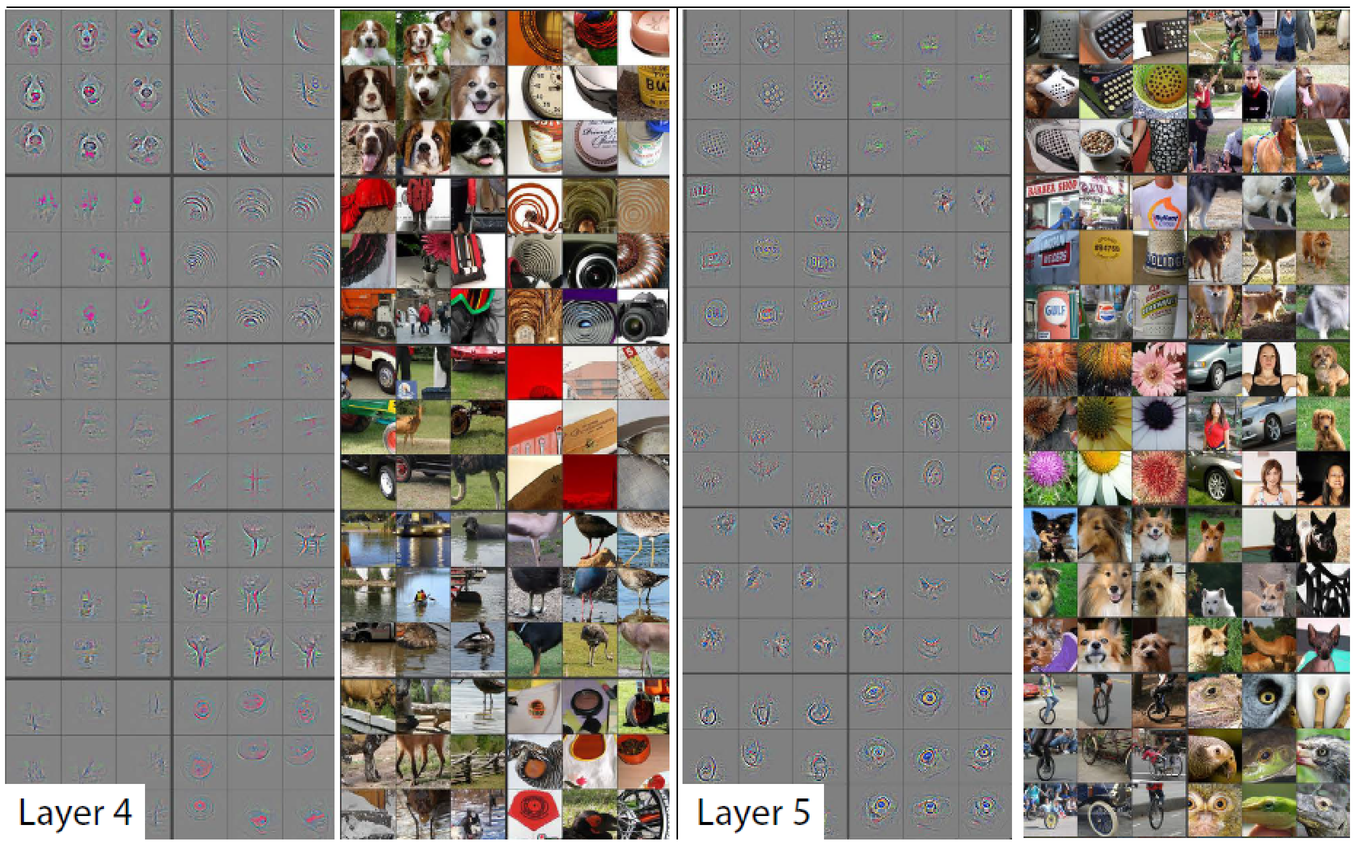


Figure 3.11: Figure visualizing layer 4 and 5 of a CNN. It can be observed that the network learns more complex features such as the face of dogs and round objects in layer 4. By layer 5, it learns to identify human faces and sign posts. This is in contrast to the simple line and contour representations learnt in layer 1 and 2 above (*Zeiler and Fergus*, 2013).

tasks is of particular interest to us.

3.3.3 Recurrent Neural Networks

Feedforward networks like CNN fall short when applied to sequence data. This is because, by design, they do not explicitly capture time dependencies in the data. Recurrent neural networks (RNNs) mitigate this problem by capturing time dependencies. They were first introduced by Hopfield in 1982 in his work to understand associative memory in the human brain (*Hopfield*, 1982).

RNNs work with the assumption that, the preceding elements in a sequence are relevant for making a decision about the current element, that is, for a sequence of entities $x_1, x_2, x_3 ... x_n$, the value of x_n depends on all entities from x_1 to x_{n-1}. This assumption is expressed by introducing hidden states which serve as memory of previous elements in the sequence. The hidden states are updated via a feedback loop as shown in Figure 3.12. These feedback loops allow information to persist over time.

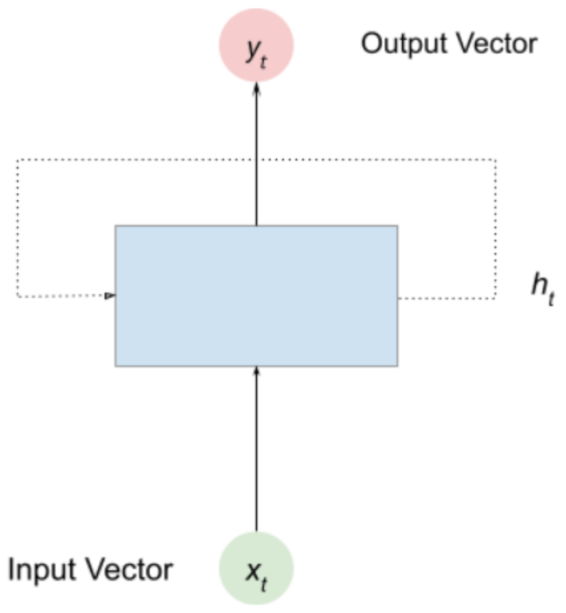


Figure 3.12: Diagram of a recurrent unit. Given some input vector x_t, it computes the hidden state h_t which is passed on to the next state.

The hidden state is computed as:

$$h_t = \sigma(W_{xh}x_t + h_{t-1}W_{hh} + b_h) \tag{3.16}$$

where h_{t-1} is the previous state based on the previous timestamp, x_t is the input at time t, σ is the activation function, W_{xh} is the weight vector of the input, W_{hh} is the weight vector of the hidden state and b_h is the bias of the hidden state.

The output of the unit is mathematically expressed as:

$$y_t = h_t W_{hq} + b_q \tag{3.17}$$

where h_t is the hidden state, W_{hq} is the weight of the output and b_q is the bias of the output. It is important to note that, the same model parameters used in equation 3.16 and 3.17 are shared across the different timestamps. This is necessary to allow variable length sequences and also reduce computational cost.

RNNs are prone to the vanishing and exploding gradient problem (*Hochreiter*, 1998). This is due to how deep the network becomes during backpropagation. In the backpropagation stage, the network is usually unrolled making it have more hidden layers than a typical feedforward network. This increases the risk of the gradients becoming very large (exploding) or very small (vanishing) during training. Vanishing and exploding gradients are solved by rescaling the norm of the gradients when they go over a predefined threshold. This process is called gradient clipping.

Vanilla recurrent neural networks are prone to noise and have a tendency to overfit. A variation of RNNs called the Long Short-Term Memory (LSTMs) (*Hochreiter and Schmidhuber*, 1997) help address this issue. LSTMs consists of computational blocks that control information flow.

3.3.4 Autoencoders

Autoencoders are neural networks used for compressing input data and then constructing that data back into its original form while preserving the relevant features in the data.

The autoencoder network consists of 2 parts as shown in Figure 3.13. The two parts are connected via the bottleneck layer. This layer is also referred to as the compression or the latent space representation.

The encoder network performs dimensionality reduction on the input data. The decoder takes the reduced data as input and reconstructs the original input as closely as possible. Although the decoder and encoder have different parameters, they work together to output the reconstructed input.

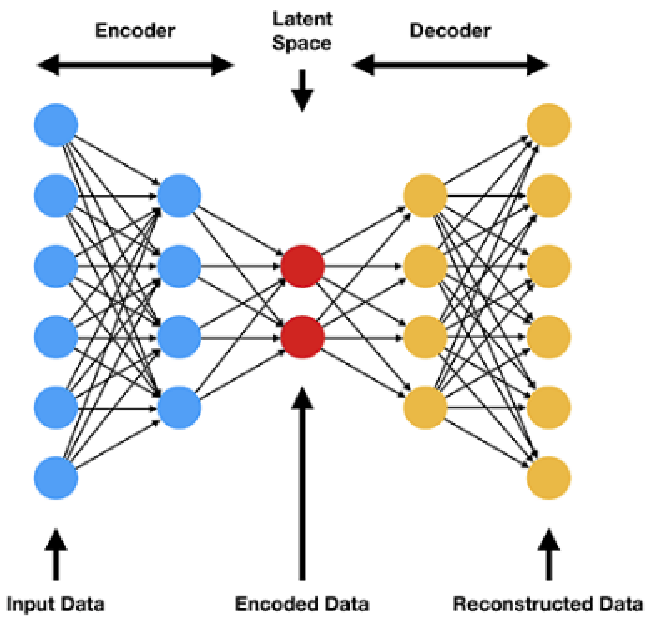


Figure 3.13: Diagram of an autoencoder. The encoder portion (on the left) is indicated in the blue, the bottleneck (in the middle) in red and the decoder(to the right) in yellow (*Chauhan*, 2021).

Given some input, x, and parameter, θ, the encoder of the network can be expressed as:

$$z = f_\theta(x) \tag{3.18}$$

Since the decoder takes the latent representation to reconstruct the output, we can express it as:

$$x' = g_\phi(z) \tag{3.19}$$

Equation 3.18 and 3.19 can be combined to obtain

$$x' = g_\phi(f_\theta(x)) \tag{3.20}$$

Autoencoders have several application including but not limited dimensionality reduction ,removing noise images and anomaly detection as discussed later in this chapter.

3.3.5 Generative Adversarial Networks

Most of the neural network architectures we have discussed so far are applied in the context of either making a prediction about some data or reducing the dimensions of the data. However, we sometimes want the ability to generate new data. Generative Adversarial Networks (GANs) were created to address this issue.

GANs were first proposed by Ian Goodfellow in *Goodfellow et al.* (2014). It consists of a Generator (G) and a Discriminator (D). The Generator attempts to generate new images to fool the Discriminator. The Discriminator on the other hand tries to distinguish between fake and real images as shown in Figure 3.14. The Generator and Discriminator are trained simultaneously to minimize $\log(1 - D(G(z)))$. Over time, the network learns to generate more realistic looking images.

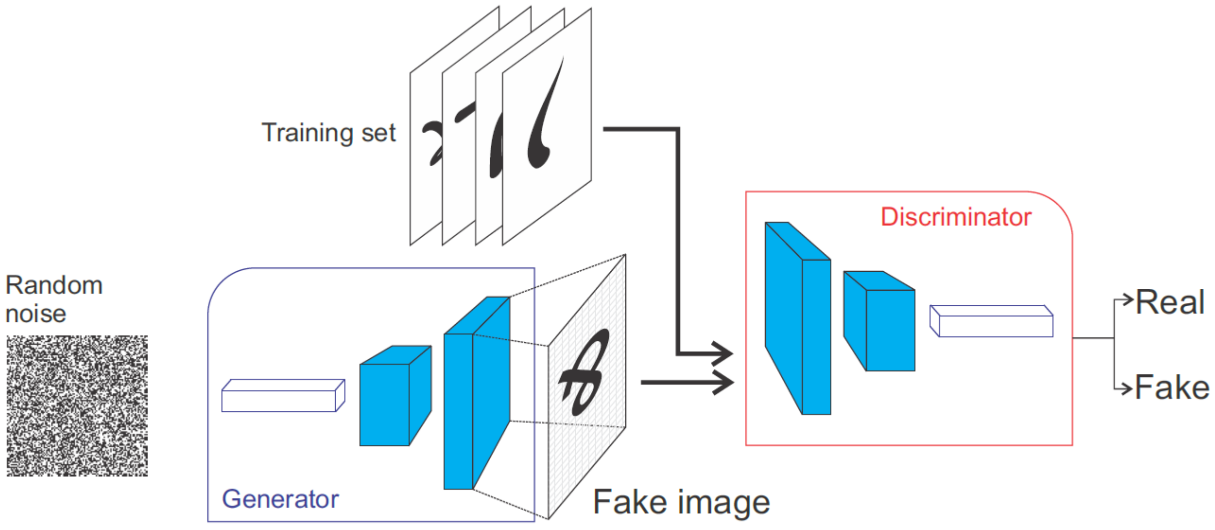
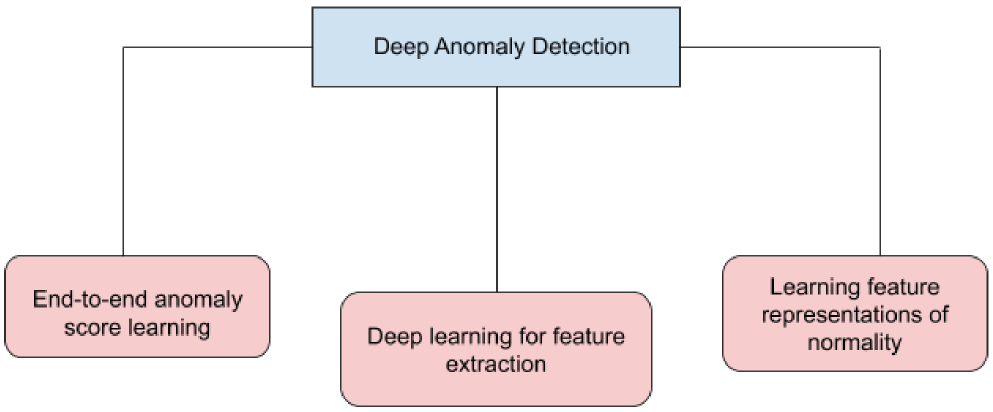


Figure 3.14: Illustration of the Discriminator and Generator of a GAN. The Generator tries to produce realistic looking images and the Discriminator is a classifier that distinguishes between the images from the Generator (fake images) and some real images from a training set (*Karim*, 2018).

3.4 Deep Anomaly Detection

In the previous section, we explored the details of popular deep learning architectures used in anomaly detection. In this section, we will see how these architectures are applied to the anomaly detection task.

Figure 3.15 is a three level hierarchical taxonomy introduced by *Pang et al.* (2020) to classify how deep learning is used for anomaly detection.



Deep Anomaly Detection
End-to-end anomaly score learning
Deep learning for feature extraction
Learning feature representations of normality

Figure 3.15: Summary of proposed taxonomy for deep anomaly detection methods.

3.4.1 Deep Learning for Feature Extraction

This group of methods use deep learning to extract expressive lower dimensional features from a higher dimensional input. These features are then applied to a downstream anomaly detection task. This approach works with the assumption that, deep neural nets preserve the discriminative features that distinguish anomalies from normal instances.

The deep neural network in this approach can be conceptualized as a dimensionality reduction technique. The resulting features obtained is passed to an independent anomaly scoring function. In some cases, some of these features might be passed to traditional algorithms such as Isolation forests, (*Liu et al.*, 2008) and

Local Outlier Factors, (*Breunig et al.*, 2000). Even though methods such as the principal component analysis can be used to achieve a similar purpose, deep neural networks have been shown to extract more expressive semantic features from the data (*Bengio et al.*, 2013).

Pre-trained networks such as VGG (*Simonyan and Zisserman*, 2015), Resnet (*He et al.*, 2015a) are applied to extract features from image and video data for anomaly detection. These pre-trained networks are originally trained on the Imagenet (*Deng et al.*, 2009) dataset.

3.4.1.1 Application of Deep Learning for Feature Extraction

This approach is mainly used in video and image anomaly detection (*Ravanbakhsh et al.*, 2016; *Xu et al.*, 2015). *Marsden et al.* (2017) applied CNNs to extract anomalous events in crowded places. This is contrary to previous approaches such as *Mehran et al.* (2009) and *Amraee et al.* (2008) where the features used for anomaly detection were handcrafted. Results from *Nazaré et al.* (2018) show that pre-trained networks are able to extract better semantic features in video anomaly detection. In crowd abnormality analysis, two approaches are employed:

1. Using a pre-trained network such as VGG (*Simonyan and Zisserman*, 2015) or ResNet (*He et al.*, 2015a) for extracting features
2. A convolutional neural network is trained from scratch on the dataset and then at test time, each image is passed through the network to extract some intermediate features

Another approach is to explicitly train a deep feature extractor for downstream anomaly detection (*Erfani et al.*, 2016) and (*Ionescu et al.*, 2018). In (*Ionescu et al.*, 2018), the authors proposed a framework for abnormal event detection in surveillance data. The framework is summarized as follows:

1. An unsupervised feature learning framework based on a Convolutional Autoencoder. This framework is used in both motion and appearance data.

2. Clustering the features extracted and using the cluster as data labels.

3. Training a classifier to distinguish between normal and abnormal events.

A similar approach is applied in graph anomaly detection. In (*Yu et al.*, 2018), features are extracted from the latent space of a deep Autoencoder to calculate the abnormality of graph vertices and edges.

Extracting features using deep learning has some pros and cons. In the case where a pre-trained model is used, there are several options available. Also, the dimensionality reduction performed by deep learning captures better discriminative features than its linear counterparts like principal component analysis. However, using pre-trained might not be efficient if the features of the downstream task varies widely from that of the data the model was trained on. The separation of feature learning and anomaly scoring might also lead to sub-optimal anomaly detection results.

3.4.2 Learning Feature Representations of Normality

These class of methods combine feature learning and anomaly scoring. This is different from the previous approach where the learning of the features and the anomaly scoring are decoupled. Popular deep learning architectures such as Autoencoders and Generative Adverserial Networks are adapted to learn normal features from the data.

3.4.2.1 Autoencoders

The Autoencoder based approach works on the assumption that normal instances can be better reconstructed from the latent space than anomalies. Autoencoders learn the low-dimensional representation on which the data instances can be reconstructed. The lower dimensional features contain regularities of the data to minimize reconstruction errors. The data instances with large reconstruction errors are assumed to be anomalies.

As discussed in section 3.3.4, Autoencoders consist of two parts: the encoder and the decoder. The Encoder maps the input to a low-dimensional space. The Decoder attempts to reconstruct the data back to a high dimensional space. The parameters of the network are learnt via a reconstruction loss function. In order to minimize the reconstruction error, the network must retain as much relevant information about the normal class as possible. In this approach, the data reconstruction error can be used as an anomaly score. Different types of Autoencoders are used to capture different features of normality.

There are different types of Autoencoders used in anomaly detection. The sparse Autoencoder (*Makhzani and Frey*, 2014) is used for anomaly detection (*Zhao and Karray*, 2020). It introduces sparsity in its hidden layers by keeping the top-K activation units. This makes it less prone to noise. The Denoising Autoencoder (*Vincent et al.*, 2010) also learns robust representations by learning to reconstruct data from corrupted data instances. The Variational Autoencoder (*Kingma and Welling*, 2014) encodes the input data using a prior distribution over the latent space. This prevents overfitting and also aids the generation of new data instances from the latent space.

Autoencoders have been applied to detect anomalies in sequence data (*Lu et al.*, 2017), graph data (*Ding et al.*, 2019) and image/video data (*Xu et al.*, 2015). Autoencoders can be biased when the training data contains a significant number of irregularities. In that case, the irregularities are treated as normal instances.

3.4.2.2 Generative Adverserial Networks

In section 3.4.2.2, we explored the GAN architecture.To use GANs for anomaly detection, the network is trained on the normal data points. This helps the Generator to learn a latent feature space that captures features of normality in the dataset. When the network is presented with an input which is an anomaly, the reconstruction error of input and generated data will be high. GANs have been adapted in various ways for anomaly detection. Some of this adaptation is discussed below.

AnoGAN (*Schlegl et al.*, 2017) was one of the earliest application of GANs to anomaly detection. Given a data instance **x**, AnoGAN searches for an instance **z** in the latent space such that the generated instance **G(z)** and **x** are as similar as possible using the residual loss. If the instance **x** is an anomaly, it will less likely not have a similar counterpart in the latent space. This is because, the generator learns the underlying distribution of the normal instances. A common problem with AnoGAN is its computational requirement, especially at test time. Given every new input, AnoGAN performs a new search. This makes it computationally inefficient.

Efficient GAN-Based Anomaly Detection (*Zenati et al.*, 2018) solves the computational issues with AnoGAN. It uses a Bidirectional GAN (BiGAN) (*Donahue et al.*, 2016). In addition to the generator and the discriminator, a BiGAN trains an encoder(E) to map **x** to **z**. This removes the need to iteratively search for **z** at test time and hence improving the computational efficiency.

GANomaly, introduced in (*Akcay et al.*, 2018), trains a generator on normal data samples. It simultaneously trains an Autoencoder to encode the latent representations more efficiently. Using the Autoencoder makes the network learn faster since there's no need for a noise prior.

Although GANs have proven to be successful in anomaly detection, they still present some challenges during training. Some of these challenges include their failure to converge and how they can easily be fooled to learn instances that are outside of the manifold of normal data points. (*Metz et al.*, 2016).

3.4.3 End-to-end Anomaly Score Learning

This approach learns anomaly scores in an end-to-end fashion. Unlike the approach in section 3.4.2 which uses generic loss functions, this approach employs custom loss functions to aid the learning of the anomaly scoring network.

One approach is to do an end-to-end one-class classification. The approach works on the assumption that normal instances can be summarized by a discriminative model. It employs a GAN to learn a one-class classifier such that the normal instances are separated from adverserially generated pseudo anomalies. This approach varies from the GANs approach discussed in section 3.4.2.2 in the following ways:

1. The approach in section 3.4.2.2 aims to learn the data distribution of the training set. Since the normal class is in the majority, we expect the distribution learnt to capture normality. On the other hand, the method in this approach learns a classifier that separates normal instances from adversarial generated instances.
2. While the GAN methods in section 3.4.2.2 use the residual between the real instances and the fake instances (from the generator) as the anomaly scores, the approaches in this method directly use the discriminator to classify anomalies.

It is important to mention that there's also no guarantee that the generated anomaly samples will be similar to unknown anomalies.

3.5 Summary

In this Chapter, we discussed deep learning and various applications of deep learning to anomaly detection. This discussion is a necessary precursor to our problem of interest, that is, to use convolutional neural networks to evolve features for anomaly detection.

We dedicated a part of this discussion to topics such as backpropagation, activation and loss functions as they relate to training a deep neural network. We also discussed other neural network architectures such as RNNs, GANs and autoencoders as an antecedent to how different architectures are applied in anomaly detection.

A good part of this chapter was dedicated to CNNs as they are the architecture of choice in our proposed algorithm. An important part of this discussion is visualizing how CNNs learn features. We explained how CNNs learn increasingly complex features as we go deeper in the network and how this gradual learning of the features is an important foundation in our algorithm.

These discussions provide a good segue into the next Chapter. Our core algorithm, **Ahunt**, discussed in the next chapter uses concepts discussed thus far to tackle our research problem. Concepts such as neural network architecture and loss functions and their effects in anomaly detection will be discussed in the next chapter.

4 Ahunt

4.1 Chapter Introduction

This chapter discusses our framework, **Ahunt**. The name Ahunt is derived from the phrase **"Hunting Anomalies"**. We start the chapter with a broad overview of the framework and gradually narrow it down to how the individual components work.

Section 4.3 discusses our methodology. We highlight the algorithms with which **Ahunt** was compared. We also discuss the datasets as well as the data augmentation techniques applied to our setup. Finally, we present a summary of the parameters and hyper-parameters of the Deep Neural Network.

Section 4.4 presents our results and discussions. We discuss results from **Ahunt** being applied to the MNIST, CIFAR-10 and Galaxy Zoo datasets. This is followed by studying how both the loss functions and various active learning strategies affect the performance of **Ahunt**.

4.2 Overview of the Ahunt Algorithm

In **Ahunt**, we begin with some labelled instances of normal data which are fed to a Neural Network for training. The network is made up of **n** normal classes and **1** other class reserved for anomalies.

Learning is done in rounds. In each round, the available pool of unlabelled data is presented to the trained network to identify anomalies. For round 0, referred to as the **zeroth round** henceforth, a Teacher (Oracle) labels a small number of examples from a large pool of unlabelled data. The data is fed to a neural network which learns the underlying features for a classification task.

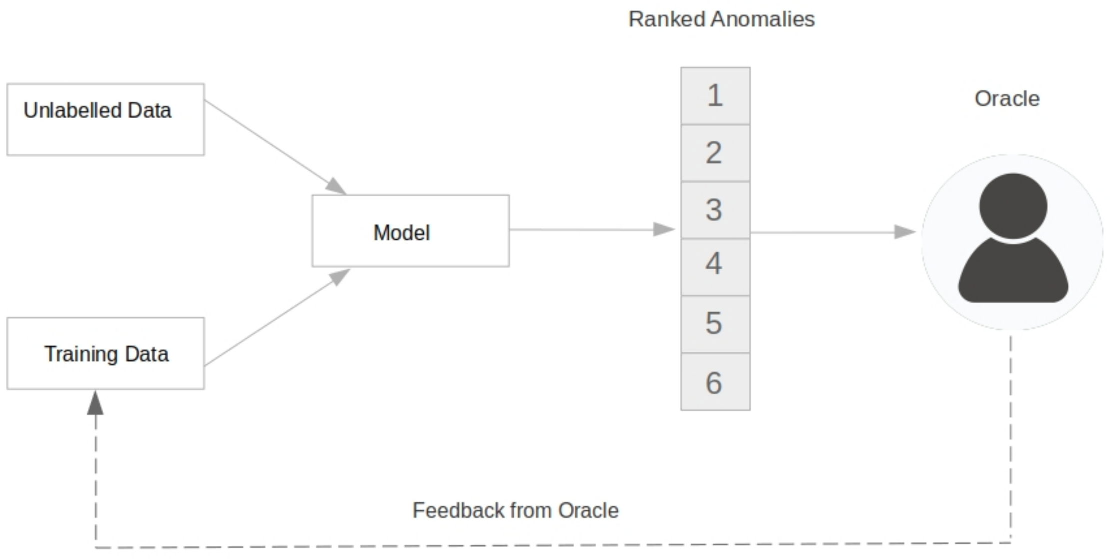


Figure 4.1: In the **Ahunt** framework, we start by labelling instances of the normal classes for training a deep neural network model. The pool of unlabelled data, containing anomalies, is tested on the model to give a ranked list of the most likely anomalies, according to the model. A subset of these anomalies are displayed to the oracle for labelling on the most interesting anomalies. The Oracle's feedback is added back to the training pool for the next round of training. This follows the the algorithm defined in Algorithm 1.

The resulting model is tested on the remaining unlabelled data. The goal is to identify the most useful examples which could help improve the model in the next round. Using some query strategy, the identified examples are shown to the Oracle for labelling. Any anomaly detected is assigned to the reserved class for the next round of training.

Once an anomaly is detected and added to the reserved class, the reserved class is up-sampled using data augmentation techniques in the next round of training. Augmenting the reserved class (now the anomaly class) also helps the network to learn good features to improve the anomaly detection outcome. Here, features refer to output before the final layer in the network. The process is repeated over several rounds of active learning.

4.3 Methodology

4.3.1 Overview of Experiments

We conduct experiments using the datasets described in section 4.3.2. We track the performance of **Ahunt** over 30 rounds of Active Learning. The choice of the number of rounds is dependent on the performance gain in each round. If the improvement from round to round is minimal, then it makes sense to not perform any more rounds of Active Learning. In our setup, 30 rounds is the optimal number of rounds beyond which the performance gain is minimal. Each experiment involves training a neural network on some labelled subset of the data. The resulting model is then used to identify potential anomalies in a large pool of unlabelled data.

In the experiment involving MNIST, we use a basic CNN with two Convolutional layers and some dropout layers. The choice of this architecture was influenced by the relatively simple nature of the MNIST dataset. A similar architecture was used for the CIFAR-10 dataset.

In the Galaxy Zoo dataset, we used a ResNet (*He et al.*, 2015a) which was pretrained on the ImageNet (*Deng et al.*, 2009) dataset. We freeze all the layers except the fully connected layer and fine-tuned the fully connected layer. We used a pretrained ResNet because of the added complexity of images in the Galaxy Zoo dataset. The choice also helped to explore the performance of **Ahunt** in the context of pretrained models, as well as models trained from scratch (as seen in the case of MNIST and CIFAR-10).

Table 4.1 summarizes the datasets and parameters used in our experimental setup. We benchmark **Ahunt** against some algorithms. We discuss these algorithms in section 4.3.3.

4.3.2 Datasets

We use 3 datasets in our experiments. The datasets are chosen according to increasing order of complexity. The term complexity here is loosely used to describe the size as well as the number of channels in the dataset. In the subsequent sections, we examine the 3 datasets with 4 considerations - their structure, shape, why they were chosen and their preprocessing steps, if any.

4.3.2.1 MNIST

The MNIST (Modified National Institute of Standards and Technology) is a database of handwritten digits ranging from $0 - 9$ as shown in Figure 4.2. It was created by (*LeCun et al.*, 2010) by remixing the original NIST dataset (*Grother*, 1995). The MNIST database consists of 60000 training images and 10000 test images.

In our experiments, we arbitrarily choose 2 classes as normal and 1 class to represent the anomaly class. We then add Gaussian noise to the images.

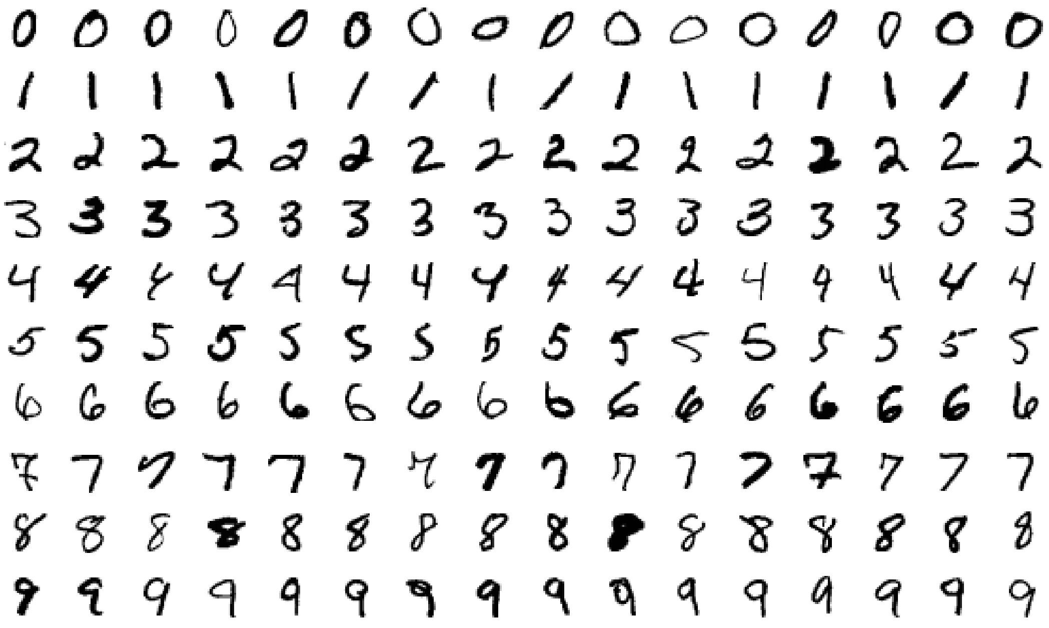


Figure 4.2: Sample images from the MNIST dataset (*Steppan*, 2017). The dataset consists of handwritten digits labelled from 0 to 9.

4.3.2.2 CIFAR-10

CIFAR-10 (*Krizhevsky*, 2009a) is a subset of the 80 million tiny images dataset (*Torralba et al.*, 2008) which consists of tiny images of size 32 x 32 pixels. CIFAR-10 consists of 10 classes with each class containing 6000 images.

There are two experiments in which the CIFAR-10 dataset is applied. In the first experiment, we assume that there is only one sub-class of anomalies. We select 3 arbitrary classes: 2 classes representing the normal classes and 1 class representing the anomaly class. In the second experiment, we form a heterogeneous anomaly class by combining **8** classes. We choose 2 sub-classes as normal and combine the remaining 8 classes into an anomaly class.

Images from the CIFAR-10 dataset represent various real world objects and hence offer more diversity than MNIST which is a 2D representation of digits.

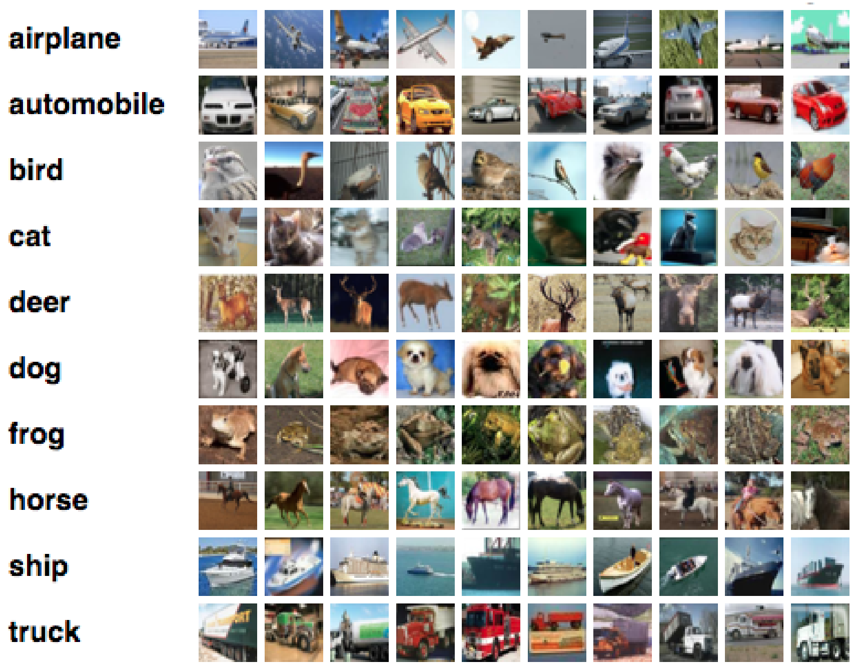
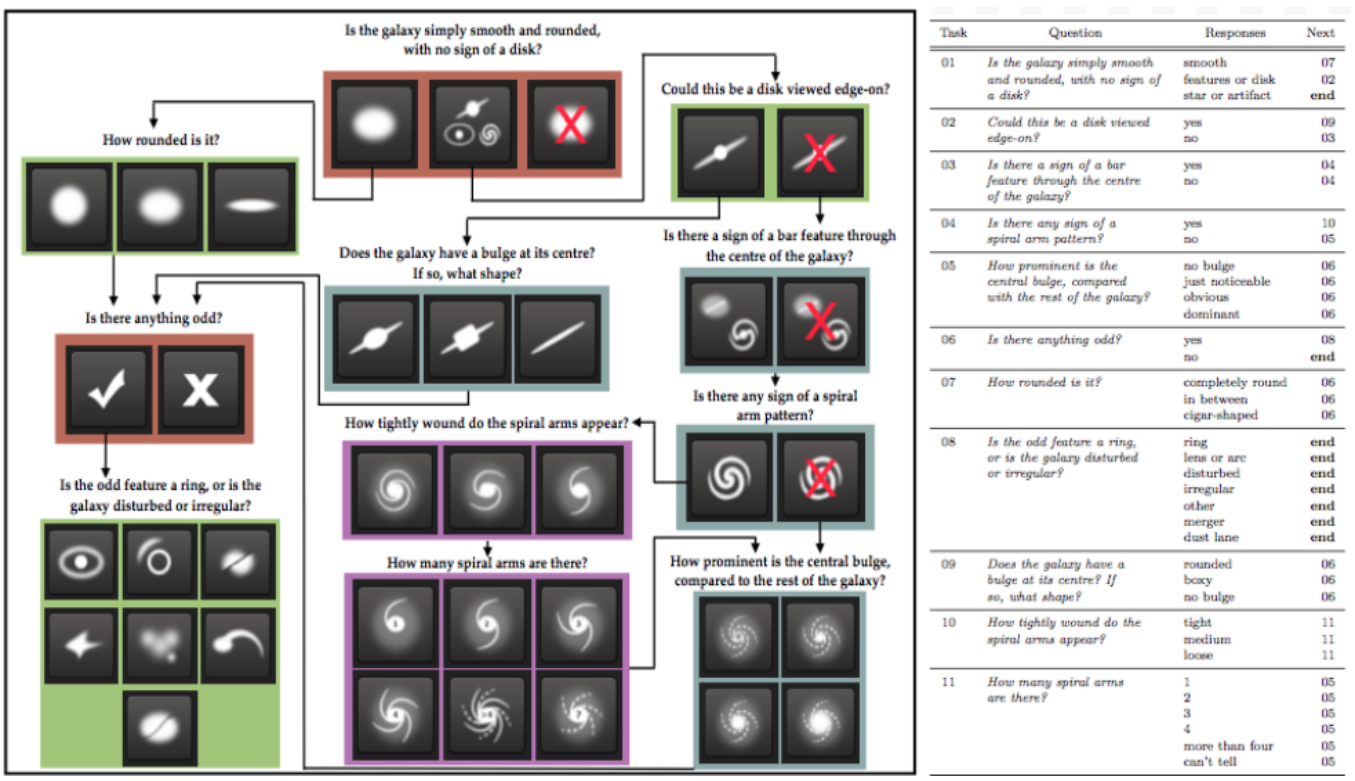


Figure 4.3: Sample images along with their respective classes in the CIFAR-10 dataset (*Krizhevsky*, 2009b).



Is the galaxy simply smooth and rounded, with no sign of a disk?
Could this be a disk viewed edge-on?
How rounded is it?
Does the galaxy have a bulge at its centre? If so, what shape?
Is there a sign of a bar feature through the centre of the galaxy?
Is there anything odd?
Is there any sign of a spiral arm pattern?
How tightly wound do the spiral arms appear?
Is the odd feature a ring, or is the galaxy disturbed or irregular?
How many spiral arms are there?
How prominent is the central bulge, compared to the rest of the galaxy?

Task	Question	Responses	Next
01	*Is the galaxy simply smooth and rounded, with no sign of a disk?*	smooth	07
		features or disk	02
		star or artifact	end
02	*Could this be a disk viewed edge-on?*	yes	09
		no	03
03	*Is there a sign of a bar feature through the centre of the galaxy?*	yes	04
		no	04
04	*Is there any sign of a spiral arm pattern?*	yes	10
		no	05
05	*How prominent is the central bulge, compared with the rest of the galaxy?*	no bulge	06
		just noticeable	06
		obvious	06
		dominant	06
06	*Is there anything odd?*	yes	08
		no	end
07	*How rounded is it?*	completely round	06
		in between	06
		cigar-shaped	06
08	*Is the odd feature a ring, or is the galaxy disturbed or irregular?*	ring	end
		lens or arc	end
		disturbed	end
		irregular	end
		other	end
		merger	end
		dust lane	end
09	*Does the galaxy have a bulge at its centre? If so, what shape?*	rounded	06
		boxy	06
		no bulge	06
10	*How tightly wound do the spiral arms appear?*	tight	11
		medium	11
		loose	11
11	*How many spiral arms are there?*	1	05
		2	05
		3	05
		4	05
		more than four	05
		can't tell	05

Figure 4.4: A combined diagram of the questions asked (right) and the flowchart (left) in the Galaxy Zoo survey. Adapted from *Willett et al.* (2013).

4.3.2.3 Galaxy Zoo

The Galaxy Zoo project is a citizen science project aimed at classifying different patterns in galaxies drawn from the Sloan Digital Sky Survey (*Willett et al.*, 2013).

The dataset consists of 304122 galaxies grouped according to the decision tree shown in Figure 4.4.

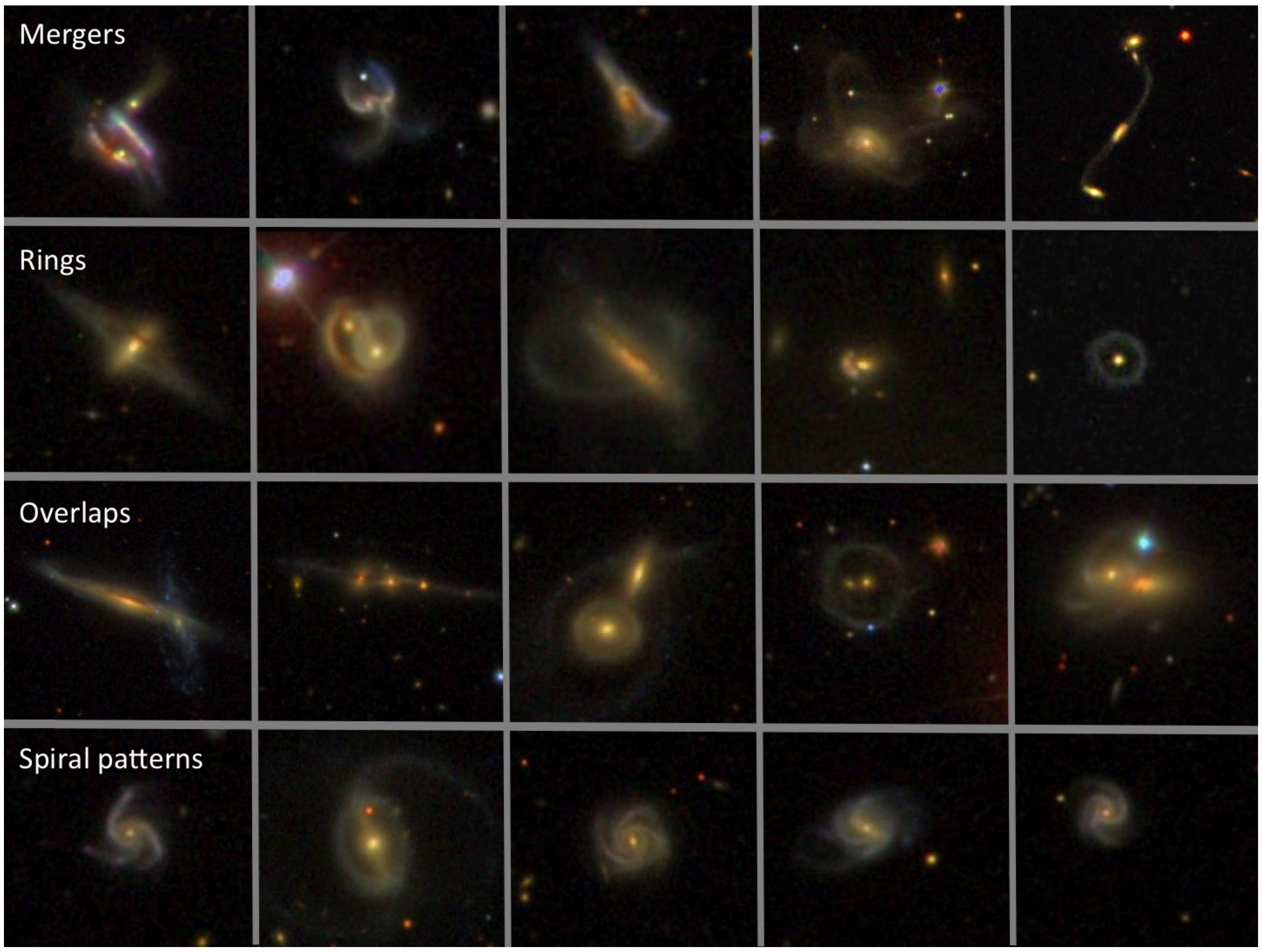


Figure 4.5: Random image samples from the Galaxy Zoo dataset *Willett et al.* (2013).

In this study, we select the odd galaxies as captured in the question "Is there anything odd?" from the decision tree shown in Figure 4.4. This question provides a good segue into identifying galaxies classified as odd or normal. For the galaxies classified as odd, there are six sub-classes namely: ring, lens or arc, distributed, irregular, other, merger and dust lane. These sub-classes result from the question: "Is the odd feature a ring or is the galaxy distributed or irregular?". Our anomaly of interest is the sub-class with a ring as the odd feature. We use this with the normal class for the purposes of our study.

This dataset is interesting because of the following:

1. It contains galaxies classified by humans as odd. This is in contrast to MNIST

where we arbitrarily choose a sub-class to represent the anomaly class.

2. The images offer more diversity than MNIST. It also captures the nature of real-world anomalies.

4.3.2.4 Summary of Data Configuration Used in the Experiments

As discussed in section 4.2, we train our model over 30 rounds of active learning. In each round, the Oracle labels 6 examples as captured in the "Questions / Round" row in table 4.1.

	MNIST	CIFAR-10	Galaxy Zoo
Normal Class	0,1	airplane, bird	class6.2
Anomaly Class	2	dog	class6.1
# 0-th Round	500, 700	800, 500	800
# Normal / Round	150, 140	140, 150	170
# Anomalies / Round	6	6	6
# Rounds	30	30	30
# Questions / Round	6	6	6
# Training time /hours	12	30	48

Table 4.1: The table provides details about the 3 datasets used in our experiments - MNIST, CIFAR-10 and Galaxy Zoo. We state our choice of normal and anomaly classes as well as the number of instances in each class for the 0th Round and any subsequent round thereafter. The table also shows the number of anomalies and the number of questions answered by an Oracle in each round. The number of questions per round is selected based on what makes practical sense for an Oracle to examine. For all 3 datasets, we run our experiments for 30 rounds.

4.3.3 Comparison Algorithms

4.3.3.1 iforest-raw

We refer to the case where the Isolation Forest (iForest) algorithm is applied to the raw pixel data as iforest-raw. Isolation forest is one of the traditional anomaly detection algorithms discussed in section 2.2.1. Isolation forest has been shown in (*Sadr et al.*, 2019) and (*Carrasquilla*, 2010) to achieve impressive results on most anomaly detection datasets. While we could have chosen other algorithms such as Local Outlier Factor (*Breunig et al.*, 2000), we know from the "No Free Lunch" theorem (*Wolpert and Macready*, 1997) that no algorithm is better than random when averaged over all possible data.

In our comparison, we use the sklearn implementation of iForest. We obtain the anomaly scores from the `decision function` of sklearn implementation. The `decision function` returns the average anomaly score (from all the Isolation trees in iForest) of each observation fed to the algorithm.

4.3.3.2 iforest on the Latent Space

A neural network can be conceptualized as consisting of two parts: a feature learner and a projection/classification head, as shown in Figure 4.6.

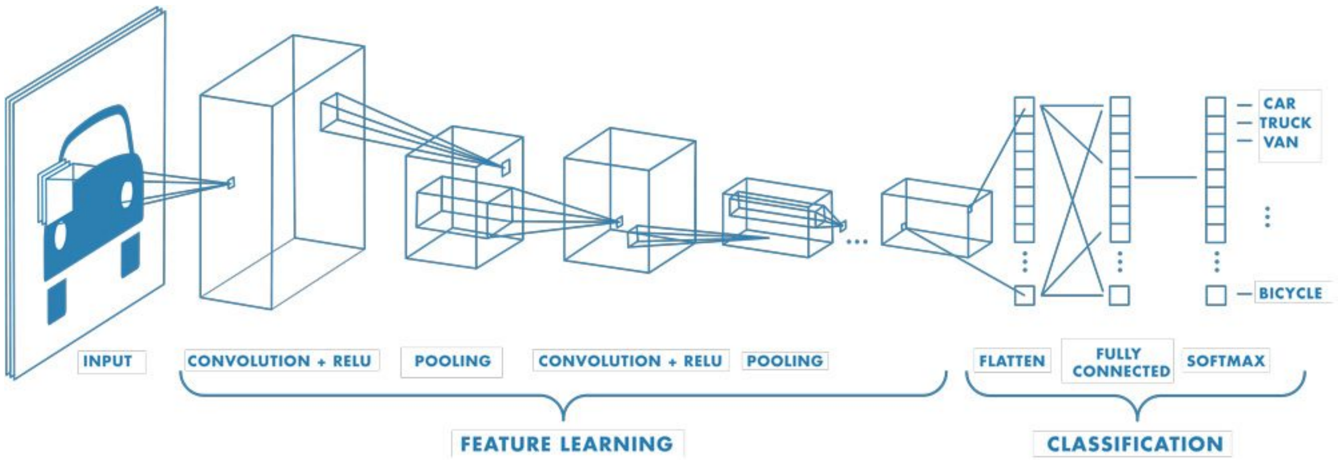


Figure 4.6: Diagram of a neural network (a CNN) indicating the feature learning part and the projection head. The feature learning part learns useful representations which is fed to the classification head for making predictions (*Patel and Pingel*, 2017).

iForest on the latent space uses features from the output of the feature learner as shown in Figure 4.6. The two types of features used in our setup are:

1. **iforest_latent-static**: This describes the case where the features are extracted from the feature learner of a model trained on the zeroth round.

2. **iforest_latent-learning**: This approach also involves feeding the features extracted from the feature learner to the iForest algorithm. However, instead of using a model trained on the zeroth round, we retrain the model for each of the 30 rounds.

4.3.3.3 Random

Random represents a setup with a random Active Learning query strategy. This is compared to other query strategies used in our Active Learning setup.

4.3.4 Data Augmentation

4.3.4.1 Introduction

In most anomaly detection setups, there are significantly more labels for the normal class than the anomaly class. This leads to a class imbalance problem. Machine learning algorithms struggle to achieve optimal performance with imbalanced classes. To mitigate this challenge, we apply data augmentation in 2 ways:

1. To upsample the minority class (the anomaly class).
2. To act as a regularizer and help reduce the risk of overfitting .

Data augmentation also slightly modifies the images leading to increased diversity in the samples. An increased diversity in the data can be intuitively interpreted as more data which helps improve the generalizability of the model. For our setup, we employ two types of augmentation techniques. In choosing these techniques, we ensure that the semantic information in the anomalies is not completely deformed. For example, changing the color of a sunflower as a data augmentation technique might alter the semantic meaning of what a sunflower is known to look like and hence might confuse the network. In the subsequent sections we explore these augmentation techniques.

4.3.4.2 Basic Data Augmentation

We apply affine transformation techniques such as rotation, translation, scaling and shearing using the implementation in the Pytorch framework (*Paszke et al.*, 2019).

To upsample the minority class, we choose samples from the minority class and randomly apply any combination of the above transformations to it. The goal here is to generate as many images from the minority class that balances the total number of images in the major class for each round of training. The random selection of the transformation (augmentation) technique ensures that the final images are not uniformly sampled from the same augmentation.

4.3.4.3 Mixup

Mixup (*Hongyi Zhang*, 2018) is a data augmentation technique that combines random images in a batch according to some predefined weights as shown in Figure 4.7.

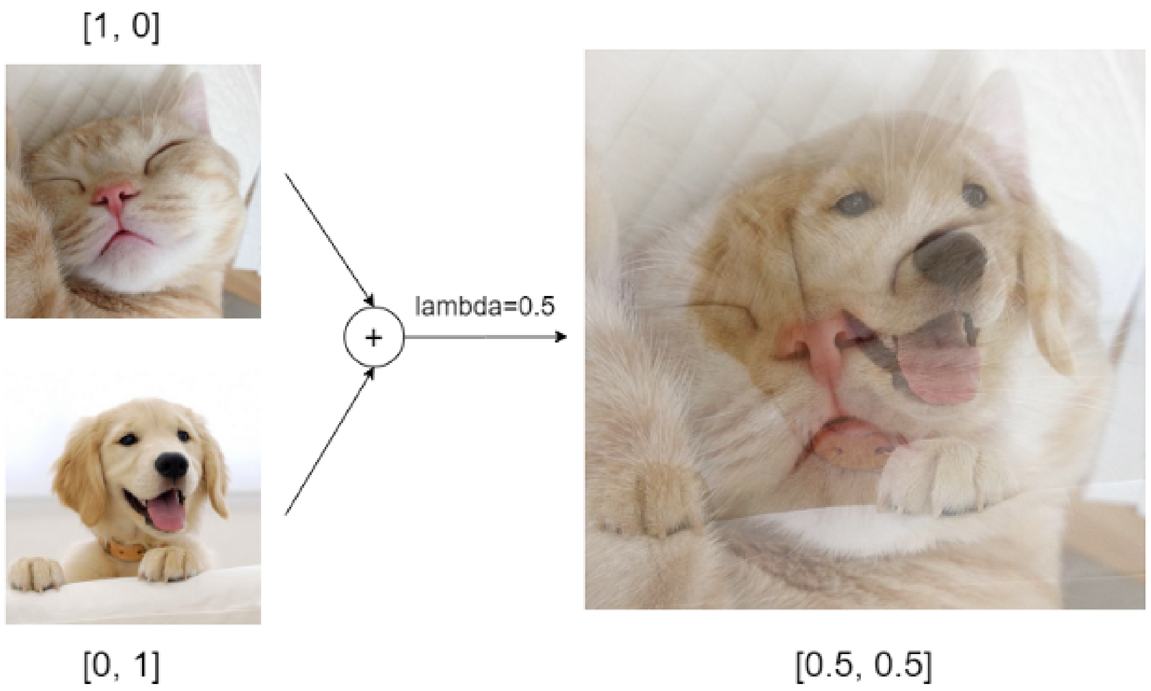


Figure 4.7: Mixup with $\lambda = 0.5$ applied on an image of a dog and an image of a cat (*Khatua*, 2020).

Given two images, (x_i, x_j) with labels (y_i, y_j), an augmented example is generated as follows:

$$\hat{x} = \lambda x_i + (1 - \lambda) x_j \tag{4.1}$$

$$\hat{y} = \lambda y_i + (1 - \lambda) y_j \tag{4.2}$$

We applied mixup in our experiment to upsample the minority class.

4.3.5 Summary of Parameters

4.3.5.1 Summary of Experiment Parameters

Experimental Parameter	**Value**
Number of Epochs	20
Optimizer	Stochastic gradient descent
Batch size	64
Learning rate	0.01
Momentum	0.5
Loss Functions	Categorical crossentropy, IsoMax*, Focalloss*

Table 4.2: Information in this table represents some training parameters and hyper-parameters. All experiments unless explicity stated use the Categorical crossentropy as loss functions. The loss functions in asterisks are studied to evaluate their effects on our setup.

4.3.5.2 Summary of Model Architecture for Each Dataset

We choose our architecture based on the complexity of the dataset.

Dataset	**Architecture**
MNIST	Custom CNN (see Appendix A1.1.1)
CIFAR-10	Custom CNN (see Appendix A1.1.2)
Galaxy Zoo	ResNet-18 (*He et al.*, 2015a)

Table 4.3: In both MNIST and CIFAR-10, we use a custom architecture consisting of convolutional layers for feature learning and Linear layers for prediction. This choice is resonable given the relative size and complexity of both datasets. However, in Galaxy Zoo, we use the default ResNet-18 architecture from Pytorch (*Paszke et al.*, 2019). We chose ResNet-18 instead other ResNet variants like ResNet-152 due to a lack of computational resources.

4.4 Results and Discussion

The results from our experiments show that **Ahunt** deforms the features space to significantly improve anomaly detection. In Figure 4.8, we compare **Ahunt** to Isolation Forest on the raw image (**iforest-raw**) and Isolation Forest on the latent features from a neural network (**iforest_latent-learning**). We compare the improvement of the MCC score over 30 Active Learning rounds. The MCC score was chosen due to the highly unbalanced nature of the datasets given that the anomalies were few in each case. The solid curves represent the mean of the MCC over all the runs in each active learning round. In MNIST and CIFAR-10, the MCC is computed for 20 randomised runs in each round while Galaxy Zoo is computed for 10 randomised runs due to a lack computational resources. The error bars represent 68% regions over these runs.

Even though all 3 algorithms start poorly, **iforest_latent-learning** and **Ahunt** gradually improve with the feedback from the Oracle in each round. **iforest-raw** does not improve throughout the 30 rounds.

It is interesting to note the performance improvement of **iforest_latent-learning** as against **iforest-raw** - same algorithm but different features. The features used in **iforest_latent-learning** are extracted from the learnt representations of a neural network making them standout for anomaly detection.

In MNIST and CIFAR-10, we extract the features from the 64-dimensional layer of the network (see Appendix A1.1.2 and A1.1.1). We observe that, in general, layers with lower dimensions perform better than those with higher dimension.

Ahunt outperforms both algorithms on MNIST, CIFAR-10 and Galaxy Zoo as shown in Figure 4.8. It learns the features in an end-to-end fashion along with the anomaly score as described in Section 3.4.3. **Ahunt** rapidly learns to identify anomalies in the case of MNIST and CIFAR-10 achieving an MCC score of 0.8 by round 15 in both cases. In Galaxy Zoo, the learning is gradual. This can be

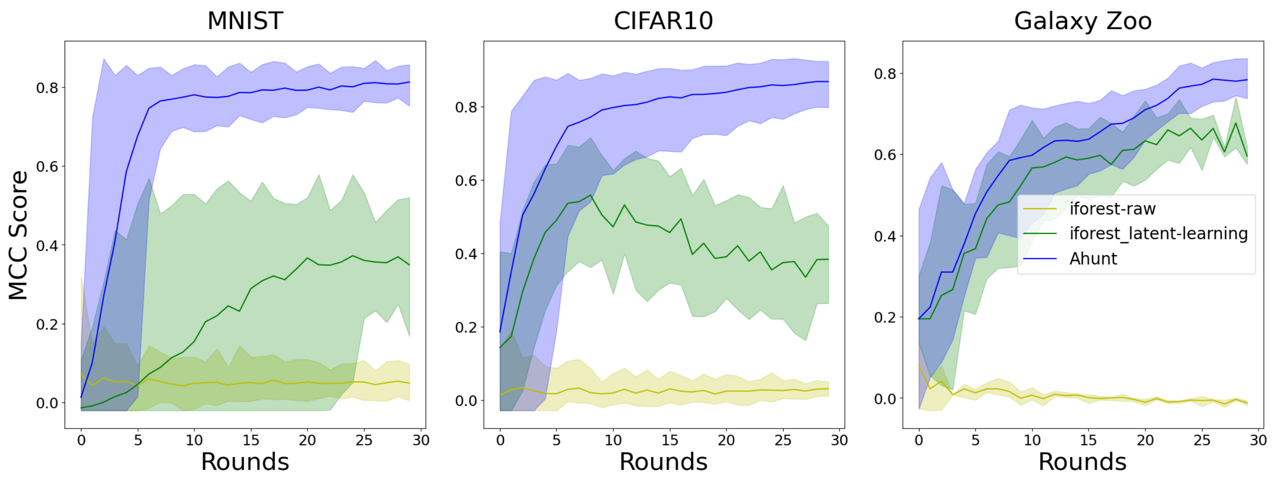


Figure 4.8: Comparison of Isolation Forest and Ahunt across 3 datasets - MNIST, CIFAR-10 and Galaxy Zoo. **iforest-raw** represents Isolation Forest trained on the raw pixel data while **iforest_latent-learning** is Isolation Forest applied to features extracted from a trained model at each round. The error bars in MNIST and CIFAR-10 corresponds to 68% of regions computed for 20 randomised runs at each active learning round while the solid curves represent the mean of the MCC for the 20 randomised runs at each active learning round. Due to the computational requirements needed to run the experiments on Galaxy Zoo, the regions (68%) were computed on 10 randomised runs for each round. In all cases, **Ahunt** outperforms both **iforest-raw** and **iforest_latent-learning**. The performance boost is explained by the evolution of the feature space from one round to the next in **Ahunt**. **iforest_latent-learning** is similar to **iforest-raw** except that its features are evolved from one round to the next. This explains its superior performance to **iforest-raw** and also shows the effectiveness of evolving the features even while still using iForests.

attributed to the complex nature of the anomalies in the Galaxy Zoo dataset.

The performance of **Ahunt** and **iforest_latent-learning** shows that, deformation of features in the latent space significantly improves the outcome of anomaly detection. This is in line with our first research question where we seek to learn a model of the dataset which makes anomalies standout and improve the anomaly detection outcome.

Figure 4.8 shows that Ahunt massively outperforms iforest-raw. In a forthcoming work (A. Vafaei-Sadr, B.Bassett, E.Sekyi, 2022, in preparation), we show that Ahunt also significantly outperform a static active learning algorithm such as the one used in Astronomaly *Lochner and Bassett* (2021). This observation emphasizes the value of learning the features dynamically.

Given the superior performance of **Ahunt** and **iforest_latent-learning**, it is important to study the changes in the feature space that make this possible. In section 4.4.0.1, we'll examine how the latent space evolves with each round and how this aids in anomaly detection.

4.4.0.1 Latent Space Evolution Over 30 Rounds

Using reductions from UMAP (*McInnes et al.*, 2018), we visualize how the feature space is deformed over the 30 rounds to improve anomaly detection.

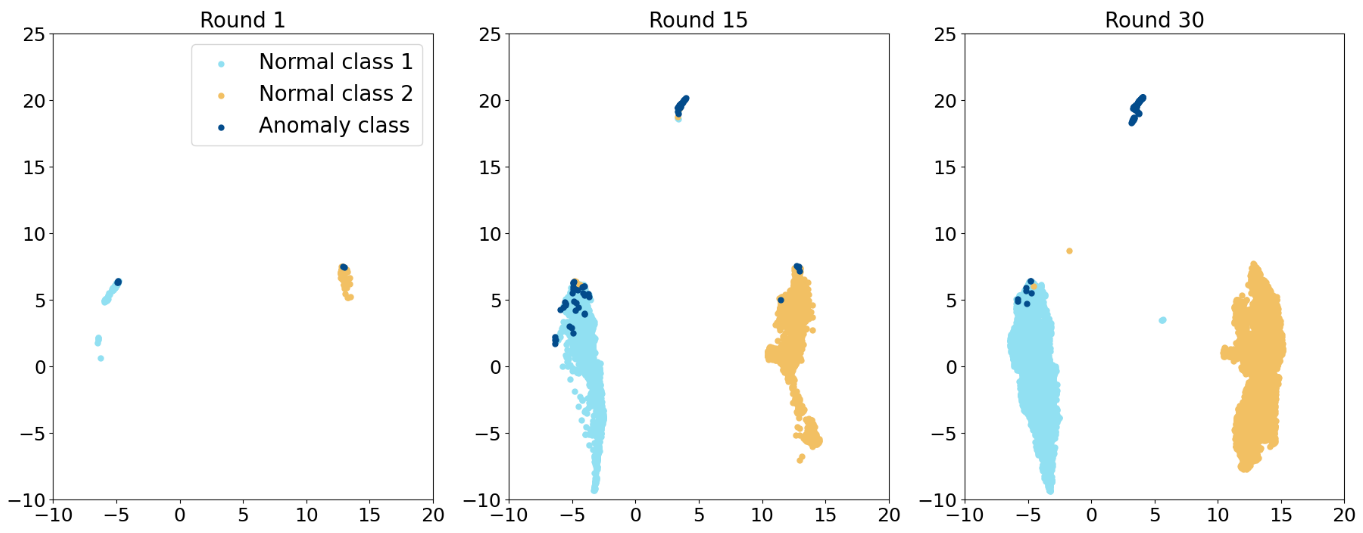


Figure 4.9: Visualization of **Ahunt's** dynamic latent space over 30 rounds of Active Learning using UMAP reductions on the MNIST dataset. We visualize features from round 1, 15 and 30. This diagram explains how deformation of the latent space gradually lead to separation of anomalies from the normal class and thereby improving the anomaly detection outcome.

In round 1, the anomalies, indicated in deep blue dots, are tightly coupled with the two normal classes. By round 15, a cluster of anomalies begin to form with few anomalies still coupled with the normal classes. Here, the feature space begins to adapt to the nature of the anomalies.

As the network receives more feedback from the Oracle, the anomalies are completely separated from the normal class 2 (indicated in beige) as seen in Round 30 in Figure 4.9. The anomalies form an even larger cluster separated from the two normal classes as shown in Appendix A1.2.

Now that we have explored how **Ahunt** deforms the feature space to separate the anomalies from the normal class, it is important to understand how different choice of parameters and hyper-parameters affect the performance of **Ahunt**. Section 4.4.1 explores how different loss functions affects the performance of **Ahunt**.

4.4.1 Effects of Loss Function on Anomaly Detection

In this section, we explore how different loss functions affect the performance of **Ahunt** . Different loss functions can lead to different amounts of tightness of clustering of the known classes, which in turn can make anomaly detection easier or more difficult. We compare 3 loss functions, namely:

1. the standard SoftMax (*Liu et al.*, 2016). Please note that, this SoftMax is different from the Softmax Activation function discussed earlier. The SoftMax refers to a training setup where both the Softmax Activation function and the cross-entropy loss are used.
2. the Focal loss (*Lin et al.*, 2017).
3. the IsoMax loss (*Macêdo et al.*, 2019).

The Focal loss function was designed to help deal with situations with large class imbalance, as typically seen in anomaly detection. The focal loss adds a modulating factor, $(1-p_t)^\gamma$, to the cross entropy loss. The resulting equation becomes:

$$F(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t) \tag{4.3}$$

where p_t is the predicted class probability of a data instance while γ is a tunable hyper-parameter which adjusts the rate at which easy examples are down-weighted. The α parameter assigns weights to the various classes with the minority class receiving higher weights. Results from *Lin et al.* (2017) shows that a $\gamma = 2$ produced the best results hence we use this in our experiments. We also assign α according to the ratio of the normal vs the anomaly class per each round of active learning (see

Table 4.1 for the exact values).

The IsoMax loss function was specifically designed for anomaly detection. Intuitively it works by increasing the intra-class compactness and inter-class separability of features in the latent space. The IsoMax loss is defined as:

$$\mathcal{L}_I(\hat{y}^{(k)}|x) = -\log\left(\frac{\exp(-d(f_\theta(x), p_\phi^k))}{\sum_j \exp(-d(f_\theta(x), p_\phi^j))}\right) \tag{4.4}$$

where $f_\theta(x)$ represents the feature vector associated with some input data x, p_ϕ^j represents a learnable prototype associated with a class j (which is the anomaly class) and the function d represents a non-squared euclidean distance. In addition, $\hat{y}^{(k)}$ represents the true class label of a normal class.

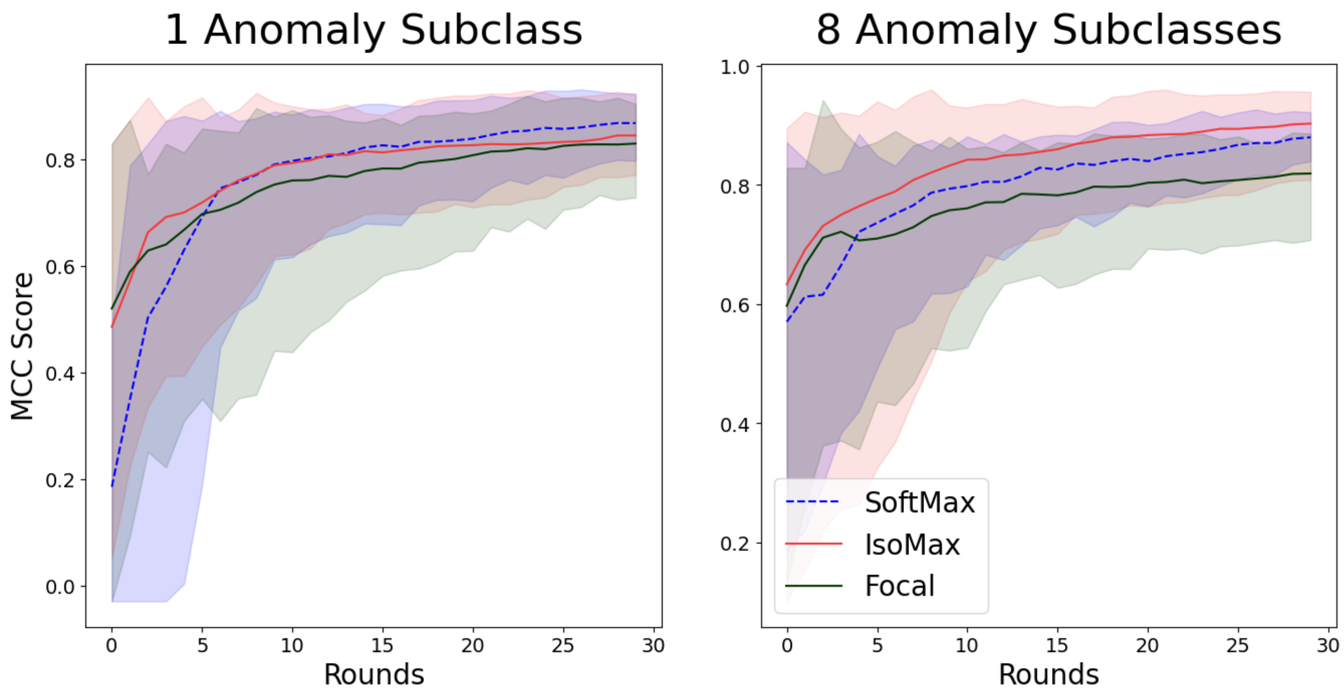


Figure 4.10: This figure compares 3 different types of loss functions; SoftMax, IsoMax and Focal loss functions. The comparison is done for a case where there is only one type of anomaly present (left) and a case where there are multiple groups of anomalies in the dataset (right). The SoftMax slightly outperforms in the single anomaly case but the IsoMax loss shows superior performance when there are multiple groups of anomalies present. The error contours correspond to 68% of regions computed for 20 randomized runs at each active learning round. The solid and dashed curves (in the case of **Ahunt**) represents the mean MCC score for all the runs in each round.

We explore the impact of the following loss functions in 2 scenarios on the

CIFAR-10:

1. the case where only one class of CIFAR-10 is chosen as the anomaly class, e.g. the dog class.
2. the case where 8 different CIFAR-10 classes are lumped together to form a heterogeneous anomaly class.

As illustrated in the Figure 4.10, there was no considerable difference between the performance of the SoftMax and the IsoMax loss in the case where there is a single class of anomalies.

However, in the heterogeneous case (where there are multiple anomaly sub-classes), the IsoMax outperforms both the SoftMax and Focal loss functions. This is expected since in this case, there will be several prototypes produced in the feature space and the model's ability to keep the learned features compact will influence the ease of detecting anomalies. In other words, there will be less overlap between the different sub-classes in the feature space due to the compactness introduced by the IsoMax loss.

It is important to note the performance of the focal loss. In our setup, we balance the anomaly class by rigorously up-sampling it using data augmentation techniques. This helps get around the class imbalance problem and hence the full effect of the focal loss isn't shown. We expect the focal loss to exhibit better performance in the absence any form of upsampling technique.

CIFAR-10 Dataset	Normal Classes	Anomaly Class
1 Anomaly sub-class	airplane, bird	dog
8 Anomaly sub-classes	airplane, bird	(automobile, cat, deer, dog, frog, horse, ship, truck)

Table 4.4: This table contains information on the homogeneous (1 anomaly sub-class) and heterogeneous (8 anomaly sub-classes) scenarios used in studying the loss functions. In the homogeneous class, we choose the dog class to represent the anomalies. The heterogeneous class consists of lumping 8 different classes into a single class to represent the anomalies. This intuitively increases the difficulty of the anomaly detection problem which is more representative of the problems encountered in the real world.

4.4.2 Active Learning Strategies

Having shown that Active Learning significantly improves anomaly detection a natural extension is to explore the effect of different active learning strategies on performance. As discussed in Section 2.6.3, there are different Active Learning Query strategies. The Query Strategies describe the ways in which the informativeness of the unlabelled instances are evaluated. This is used to assess which images are shown to the Oracle.

In our experiments, we explore 3 main query strategies namely:

1. **Most Anomalous:** This describes a case where we simply select the SoftMax probability scores of the anomaly class, rank them in decreasing order and select the **top-n** most anomalous scores. The **top-n** are shown to the Oracle for feedback.

2. **Most Uncertain:** An uncertainty index is defined for the anomalies and the **top-n** anomalies with the highest uncertainty index, p_i, are selected to display to the Oracle. The uncertainty index is given by:

$$p_i = 1 - 2|p_c - 0.5| \tag{4.5}$$

   where p_c is the Softmax probability of the anomaly (reserved) class. In an ideal case, if a model is uncertain about an input data point, we expect it to spread the Softmax probabilities across all the known classes that is 0.5 to the anomaly class and 0.5 to the normal class. If that happens, then we have a perfectly uncertain prediction in which case p_i is 1.

3. **Random:** The random query strategy chooses a random subset of the unlabelled instances to show to the Oracle.

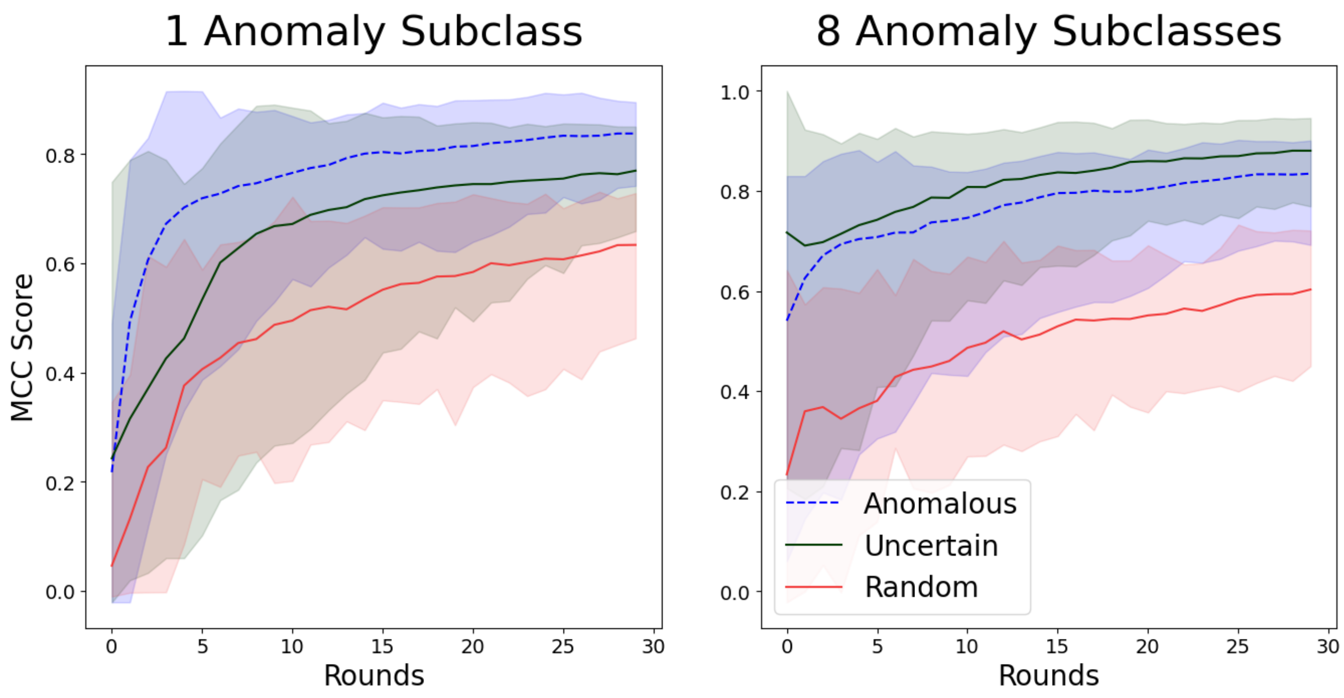


Figure 4.11: Plots showing the effects of Active Learning query strategies in a scenario where there is one anomaly sub-class (indicated on the left) and the case where there are multiple sub-classes of anomalies (indicated on the right). The random query strategy was relatively poor in both scenarios, as expected. The **Most Anomalous** exhibited better performance in the one anomaly sub-class setting but was outperformed by the **Most Uncertain** query strategy in the more complex scenario where there were multiple sub-classes of anomalies.

In general, we expect the Random query strategy to be the least efficient. This is consistent with results from the two scenarios shown in Figure 4.11. A more interesting observation is the performance on the Anomalous and Uncertain query strategies in the single sub-class anomaly case (left) and the multiple sub-class anomaly case (right).

In the 1 Anomaly Sub-class, the **Most Anomalous** query strategy outperformed the

Most Uncertain query strategy. This was in contrast to the case of the 8 Anomaly sub-classes where the **Most Uncertain** exhibited better performance. Conceptually, the **Most Anomalous** strategy captures the best prototypes of a particular class of anomaly. This makes it suitable for identifying that specific group of anomaly as seen in the case of the 1 Anomaly Sub-class.

The **Most Uncertain** query strategy helps the model by teaching it about the examples it is most uncertain about. This related to how learning occurs in humans where we give attention to the most challenging topics. Since the 8 Anomaly sub-class scenario introduces a more heterogeneous dataset, the **Most Uncertain** query strategy helps the Oracle to provide feedback about the examples it is most uncertain about and hence showing a better performance over the **Most Anomalous** query strategy.

Therefore in scenarios with a heterogeneous set of anomalies, the **Most Uncertain** query strategy is likely to exhibit better performance than the **Most Anomalous** query strategy. When the anomaly class is more homogeneous, the **Most Anomalous** query strategy will likely offer the best performance.

4.5 Summary

In this chapter, we discuss the details of **Ahunt** - an algorithm to deform the features of images to enhance anomaly detection.

Our experimental setup starts with a CNN which has **n** normal classes and **1** extra class reserved for the detected anomalies. The goal is learn in rounds of active learning on some labelled training set. The rounds of active learning can be broken down into the following steps:

1. We train a model on some labelled data.
2. We test the resulting model on a pool of unlabelled dataset containing anomalies.

3. Using an active learning query strategy (see discussion in section 2.6.3), we select a small subset from the the unlabelled pool to be shown to the Oracle (a human Expert).

4. The Oracle labels few examples. The labelled examples are added to the training set. Any anomalies found is assigned to the reserved class.

The experiment is repeated over 30 rounds of active learning.

We test **Ahunt** on 3 datasets - MNIST, CIFAR-10 and Galaxy Zoo. While MNIST is a simpler dataset (a 2D representation of handwritten digits), CIFAR-10 and Galaxy Zoo offer a wide range of diverse real world objects with the later containing galaxies labelled as odd (likely anomalies). **Ahunt** is compared to two cases of Isolation forests (iForest):

1. **iforest-raw**: Isolation forest applied to the raw pixel data
2. **iforest_latent-learning**: Isolation forest applied to features extracted from each round of active learning.

Results from all the datasets show that, **Ahunt** consistently outperforms the two cases of iForest. Another interesting observation is that **iforest_latent-learning** also consistently outperforms **iforest-raw**. This can be attributed to the evolution of the features across the active learning rounds in both **Ahunt** and **iforest_latent-learning** unlike **iforest-raw** where the features are fixed across the rounds. This is explored further by using UMAP reductions to visualize how the features in **Ahunt** evolve over the 30 rounds. It was shown that, the classes were gradually separated across the rounds and by round 30, there were 3 clear clusters: 2 representing the normal classes and 1 for the anomaly class. This goes to assert the claim that, the evolution of the features makes the anomalies stand out thereby improving anomaly detection.

We proceed to explore how various loss functions affect the performance of **Ahunt**. We consider two cases. In one case, we assume homogeneity in the anomaly class.

This is achieved by selecting one class from the CIFAR-10 dataset as the anomaly class. In the second case, we combine 8 CIFAR-10 classes into a single class to assume a heterogenous anomaly class. In both cases, we still maintain 2 normal classes chosen from the CIFAR-10 dataset. We test both cases across 3 loss functions namely:

1. SoftMax loss
2. Focal loss
3. IsoMax loss

Our results show that, all three loss functions exhibited similar performance in the homogeneous case but the SoftMax loss performed slightly better. However, in the heterogenous case, the IsoMax exhibited superior performance.

Finally, having demonstrated the effectiveness of active learning, we proceed to examine the impact of query strategies on our setup. We examine 3 query strategies namely:

1. Most Anomalous: The samples the model classifies as the most anomalous is shown to the Oracle to label.
2. Most Uncertain: We select the samples which the model is most uncertain of for the Oracle to label.
3. Random: We randomly select samples for the Oracle to label.

We compare these query strategies for experiments run across the homogeneous and heterogeneous cases described in the preceding paragraph. We observe that the **Random** query strategy was outperformed by the other two query strategies. However, the **Most Anomalous** query strategy exhibited superior performance in the homogeneous case while the **Most Uncertain** performed better in the heterogenous case.

5 | Conclusion and Future Work

This Chapter summarizes the key findings in relation to our research aims and objectives. We discuss both the contributions and limitations of this work. Finally, we recommend research directions for future work.

The aim of this thesis was to test whether dynamically learning features that make anomalies stand will improve anomaly detection. We learnt these features using a convolutional neural network. In particular, we use an Active Learning framework to provide useful feedback to help the DNN learn interesting features for anomaly detection.

We chose the MNIST, CIFAR-10 and Galaxy Zoo datasets to test out the performance of our algorithm, Ahunt. **Ahunt** outperformed all the comparison algorithms on all three datasets.

Furthermore, we examined how the feature space deforms with each round of training. Our visualizations using UMAP show that, the learned features of the anomalies are gradually separated from the normal class as the number of rounds of training increases. Intuitively, this makes it easy for anomaly detection algorithms to identify the anomalies in the dataset.

To better understand why **Ahunt** works and its limitations we studied how components like the loss functions affect its performance. We simulate two scenarios to test this out. In the first scenario, we assume the presence of only one type of anomaly. This can be viewed as a homogeneous case where all the anomalies are semantically similar. In the second scenario, we assume the existence of different types of anomalies. This can be viewed as a heterogenous case which makes the detection of the anomalies more challenging. We tested out the SoftMax, IsoMax and Focal loss functions in both scenarios. Results from our work show that, if the anomaly detection setup is heterogenous, the IsoMax loss functions offers better performance.

In line with our goal of using an Active Learning framework, we tested out 3 different query strategies - Anomalous, Uncertain and Random. These tests we carried out using the two scenarios highlighted in the previous paragraph. Our results show that, in complex anomaly detection setups with different types of anomalies, the Uncertain query strategy offers superior performance.

Unlike in most anomaly detection setups, where a static feature extractor is used, results from **Ahunt** show that, dynamically learnt features improve the outcome of anomaly detection. Active Learning helps to incorporate useful expert feedback into the model and the choice of Active Learning query strategy affects the performance of the model and hence the result of anomaly detection.

5.1 Future Work

Despite the performance boost shown by **Ahunt**, we believe there are some directions worthy of exploration. Some potential directions are as follows:

1. **Better Data Augmentation Strategies:** In **Ahunt** we use basic data augmentation techniques such as rotation and zooming to upsample the minority (anomaly) class. The limitation of this approach is that, the images generated are semantically very similar. This could hinder the resulting model's ability to learn and detect new types of anomalies. A more nuanced upsampling technique like using diffusion models and transformers to generate new types of data samples might lead to a more diverse anomaly class which in turn improves the quality of features learnt for anomaly detection.

2. **Creative transfer learning techniques:** Our current setup retrains the model after each round. This allows the model from the previous round to incorporate the Oracle's feedback in the current round. Unfortunately, training is computationally expensive especially in situations where both the dimensions and the volume of data are huge. An interesting question to explore is, can we apply transfer learning to such scenarios? Typically, transfer learning approaches freeze the feature learning part (some of the hidden layers) of the network and trains on the fully connected part to make predictions. However, freezing the feature learning part is sub-optimal since the goal here is to learn features that makes it easy to distinguish the anomalies from the normal class. Approaches such as (*Liu et al.*, 2021) offer an interesting alternative to the traditional transfer learning approach. AutoFreeze uses an adaptive approach to determine which part of the network can be frozen. This makes room for training parts of the feature while simultaneously freezing other parts which do not help improve the model. This feature is desirable in the anomaly detection setup because it makes room for the model to adjust the feature space to capture new types of anomalies without necessarily re-train the whole network.

Bibliography

Aggarwal, C. C., *Proximity-Based Outlier Detection*, 111–147 pp., Springer International Publishing, Cham, doi:10.1007/978-3-319-47578-3_4, 2017.

Akcay, S., A. A. Abarghouei, and T. P. Breckon, Ganomaly: Semi-supervised anomaly detection via adversarial training, *CoRR*, *abs/1805.06725*, 2018.

Amraee, S., A. Vafaei, K. Jamshidi, and P. Adibi, Anomaly detection and localization in crowded scenes using connected component analysis, *Multimedia Tools and Applications*, *77*, doi:10.1007/s11042-017-5061-7, 2008.

Andrews, J. T. A., T. Tanay, E. J. Morton, and L. D. Griffin, Transfer representation-learning for anomaly detection, in *ICML 2016*, 2016.

Angluin, D., Queries and concept learning, *Journal of Machine Learning Research*, *2*(4), 319–342, doi:10.1023/A:1022821128753, 1988.

Bengio, Y., A. Courville, and P. Vincent, Representation learning: A review and new perspectives, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *35*(8), 1798–1828, doi:10.1109/TPAMI.2013.50, 2013.

Bhakat, S., and G. Ramakrishnan, Anomaly detection in surveillance videos, in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, CoDS-COMAD '19, p. 252–255, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3297001.3297034, 2019.

Bishop, C., Novelty detection and neural network validation, in *IEE Proceedings: Vision, Image and Signal Processing. Special issue on applications of neural networks.*, vol. 141, pp. 217–222, 1994.

Boser, B. E., I. M. Guyon, and V. N. Vapnik, A training algorithm for optimal margin classifiers, in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, p. 144–152, Association for Computing Machinery, New York, NY, USA, doi:10.1145/130385.130401, 1992.

Breunig, M. M., H.-P. Kriegel, R. T. Ng, and J. Sander, Lof: Identifying density-based local outliers, *SIGMOD Record*, *29*(2), 93–104, doi:10.1145/335191.335388, 2000.

Caroprese, L., P. M. A. Sloot, B. O. Nualláin, and E. Zumpano, A logical framework for detecting anomalies in drug resistance algorithms, in *Proceedings of the 2009 International Database Engineering amp; Applications Symposium*, IDEAS '09, p. 23–30, Association for Computing Machinery, New York, NY, USA, doi:10.1145/1620432.1620436, 2009.

Carrasquilla, U., Benchmarking algorithms for detecting anomalies in large datasets, *MeasureIT, Nov*, pp. 1–16, 2010.

Chandola, V., A. Banerjee, and V. Kumar, Anomaly detection: A survey, *ACM Computing Surveys*, *41*(3), doi:10.1145/1541880.1541882, 2009.

Chauhan, S., Autoencoders for audience modeling, [Online; accessed September 24, 2021], 2021.

Chen, X., Anomaly detection of petrochemical process engineering based on neural network, *Chemical Engineering Transactions*, *71*, 259–264, doi:10.3303/CET1871044, 2018.

Chenguang, Y., J. Hu, W. Min, Z. Deng, S. Zou, and W. Min, A novel real-time fall detection method based on head segmentation and convolutional neural network, *Journal of Real-Time Image Processing*, *17*, doi:10.1007/s11554-020-00982-z, 2020.

Chodey, M., and C. Shariff, Neural network-based pest detection with k-means segmentation: Impact of improved dragonfly algorithm, *Journal of Information Knowledge Management*, *20*, 2150,040, doi:10.1142/S0219649221500404, 2021.

Churová, V., R. Vyškovský, K. Maršálová, D. Kudláček, and D. Schwarz, Anomaly detection algorithm for real-world data and evidence in clinical research: Implementation, evaluation, and validation study, *JMIR Med Inform*, *9*(5), e27,172, doi:10.2196/27172, 2021.

Dauphin, Y. N., R. Pascanu, Ç. Gülçehre, K. Cho, S. Ganguli, and Y. Bengio, Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, *CoRR*, *abs/1406.2572*, 2014.

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, ImageNet: A Large-Scale Hierarchical Image Database, in *CVPR09*, 2009.

Ding, K., J. Li, R. Bhanushali, and H. Liu, Deep anomaly detection on attributed networks, in *SDM*, 2019.

Doersch, C., A. Gupta, and A. A. Efros, Unsupervised visual representation learning by context prediction, *CoRR*, *abs/1505.05192*, 2015.

Donahue, J., P. Krähenbühl, and T. Darrell, Adversarial feature learning, *CoRR*, *abs/1605.09782*, 2016.

Dong, W., C. Moses, and K. Li, Efficient k-nearest neighbor graph construction for generic similarity measures, in *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, p. 577–586, Association for Computing Machinery, New York, NY, USA, doi:10.1145/1963405.1963487, 2011.

Eldan, R., and O. Shamir, The power of depth for feedforward neural networks, *CoRR*, *abs/1512.03965*, 2015.

Elmi, A. H., S. Ibrahim, and R. Sallehuddin, Detecting sim box fraud using neural network, in *IT Convergence and Security 2012*, edited by K. J. Kim and K.-Y. Chung, pp. 575–582, Springer Netherlands, Dordrecht, 2013.

Erfani, S. M., S. Rajasegarar, S. Karunasekera, and C. Leckie, High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning, *Pattern Recogn.*, *58*(C), 121–134, doi:10.1016/j.patcog.2016.03.028, 2016.

Evangelou, M., and N. M. Adams, An anomaly detection framework for cyber-security data, *Computers & Security*, *97*, 101,941, doi:https://doi.org/10.1016/j.cose.2020.101941, 2020.

Fernando, T., H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, Deep learning for medical anomaly detection - A survey, *CoRR*, *abs/2012.02364*, 2020.

Frank Rosenblatt, A. L., A. I. A. I. Kitov, and C. A. Laboratory., The design of an intelligent automaton, 1956.

Fukushima, K., Neocognitron: A hierarchical neural network capable of visual pattern recognition, *Neural Networks*, *1*(2), 119–130, doi:https://doi.org/10.1016/0893-6080(88)90014-7, 1988.

Gao, J., W. Hu, Z. M. Zhang, X. Zhang, and O. Wu, Rkof: robust kernel-based local outlier detection, in *Pacific-Asia conference on knowledge discovery and data mining*, pp. 270–283, Springer, 2011.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial networks, 2014.

Grother, P., Nist special database 19 handprinted forms and characters database, 1995.

Hariri, S., M. C. Kind, and R. J. Brunner, Extended isolation forest, *CoRR*, *abs/1811.02141*, 2018.

He, K., X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, *CoRR*, *abs/1512.03385*, 2015a.

He, K., X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *CoRR*, *abs/1502.01852*, 2015b.

Heidenreich, N.-B., A. Schindler, and S. Sperlich, Bandwidth selection for kernel density estimation: A review of fully automatic selectors, *AStA Advances in Statistical Analysis*, *97*, doi:10.1007/s10182-013-0216-y, 2013.

Hinton, G. E., and R. S. Zemel, Autoencoders, minimum description length and helmholtz free energy, in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, p. 3–10, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

Hochreiter, S., The vanishing gradient problem during learning recurrent neural nets and problem solutions, *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *6*, 107–116, doi:10.1142/S0218488598000094, 1998.

Hochreiter, S., and J. Schmidhuber, Long short-term memory, *Neural Computation*, *9*(8), 1735–1780, 1997.

Hongyi Zhang, Y. N. D. D. L.-P., Moustapha Cisse, mixup: Beyond empirical risk minimization, *International Conference on Learning Representations*, 2018.

Hopfield, J., Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences of the United States of America*, *79*(8), 1982.

Hornik, K., M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, *2*(5), 359–366, doi:https://doi.org/10.1016/0893-6080(89)90020-8, 1989.

Howard, J., and S. Gugger, *Deep Learning for Coders with Fastai and Pytorch: AI Applications Without a PhD*, O'Reilly Media, Incorporated, 2020.

Hubel, D., David h. hubel and torsten n. wiesel's research on optical development in kittens, 1964.

Huijser, M. W., and J. C. van Gemert, Active decision boundary annotation with deep generative models, 2017.

Ionescu, R. T., F. S. Khan, M. Georgescu, and L. Shao, Object-centric auto-encoders and dummy anomalies for abnormal event detection in video, *CoRR*, *abs/1812.04960*, 2018.

Jayaswal, V., *Local Outlier Factor (LOF) — Algorithm for outlier identification*, 2020.

Jin, C., R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan, How to escape saddle points efficiently, *CoRR*, *abs/1703.00887*, 2017.

Jolliffe, I., and Springer-Verlag, *Principal Component Analysis*, Springer Series in Statistics, Springer, 2002.

Karim, M. R., *Getting Started With Deep Learning, Generative adversarial networks*, p. 78–79, Packt Publishing, 2018.

Khatua, D. P., Boost image classifier performance(part 1) — mixup augmentation with codes, [Online; accessed October 25, 2021], 2020.

Kingma, D. P., and M. Welling, Auto-encoding variational bayes, 2014.

Krizhevsky, A., Learning multiple layers of features from tiny images, 2009a.

Krizhevsky, A., The cifar-10 dataset, [Online; accessed October 19, 2021], 2009b.

Krizhevsky, A., I. Sutskever, and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in Neural Information Processing Systems 25*, edited by F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, pp. 1097–1105, Curran Associates, Inc., 2012.

Lang, K., and E. Baum, Query learning can work poorly when a human oracle is used, in *IEEE Intl. JointConference on Neural Networks*, p. 335–340, IEEE Press, 1992.

Latecki, L. J., A. Lazarevic, and D. Pokrajac, Outlier detection with kernel density functions, in *Machine Learning and Data Mining in Pattern Recognition*, edited by P. Perner, pp. 61–75, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007a.

Latecki, L. J., A. Lazarevic, and D. Pokrajac, Outlier detection with kernel density functions, in *International Workshop on Machine Learning and Data Mining in Pattern Recognition*, pp. 61–75, Springer, 2007b.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, Backpropagation applied to handwritten zip code recognition, *Neural Comput.*, *1*(4), 541–551, doi:10.1162/neco.1989.1.4.541, 1989.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, *86*(11), 2278–2324, doi:10.1109/5.726791, 1998.

LeCun, Y., C. Cortes, and C. Burges, Mnist handwritten digit database, *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, *2*, 2010.

Lewis, D. D., and W. A. Gale, A sequential algorithm for training text classifiers, in *SIGIR '94*, edited by B. W. Croft and C. J. van Rijsbergen, pp. 3–12, Springer London, London, 1994.

Li, C., K. Sohn, J. Yoon, and T. Pfister, Cutpaste: Self-supervised learning for anomaly detection and localization, *CoRR*, *abs/2104.04015*, 2021.

Li, Q., X. Zhou, P. Lin, and S. Li, Anomaly detection and fault diagnosis technology of spacecraft based on telemetry-mining, in *2010 3rd International Symposium on Systems and Control in Aeronautics and Astronautics*, pp. 233–236, doi:10.1109/ISSCAA.2010.5633180, 2010.

Lin, T., P. Goyal, R. B. Girshick, K. He, and P. Dollár, Focal loss for dense object detection, *CoRR*, *abs/1708.02002*, 2017.

Liu, F. T., K. M. Ting, and Z. Zhou, Isolation forest, in *2008 Eighth IEEE International Conference on Data Mining*, pp. 413–422, doi:10.1109/ICDM.2008.17, 2008.

Liu, W., Y. Wen, Z. Yu, and M. Yang, Large-margin softmax loss for convolutional neural networks, doi:10.48550/ARXIV.1612.02295, 2016.

Liu, Y., S. Agarwal, and S. Venkataraman, Autofreeze: Automatically freezing model blocks to accelerate fine-tuning, *CoRR*, *abs/2102.01386*, 2021.

Lochner, M., and B. Bassett, Astronomaly: Personalised active anomaly detection in astronomical data, *Astronomy and Computing*, *36*, 100,481, doi:10.1016/j.ascom.2021.100481, 2021.

Lu, W., Y. Cheng, C. Xiao, S. Chang, S. Huang, B. Liang, and T. Huang, Unsupervised sequential outlier detection with deep architectures, *IEEE Transactions on Image Processing*, *26*(9), 4321–4330, doi:10.1109/TIP.2017.2713048, 2017.

Lucas, Y., Credit card fraud detection using machine learning with integration of contextual knowledge, Ph.D. thesis, 2020.

Maas, A. L., A. Y. Hannun, and A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

Macêdo, D., T. I. Ren, C. Zanchettin, A. L. I. Oliveira, A. Tapp, and T. B. Ludermir, Distinction maximization loss: Fast, scalable, turnkey, and native neural networks out-of-distribution detection simply by replacing the softmax loss, *CoRR*, *abs/1908.05569*, 2019.

Madhuri, G. S., and M. Usha Rani, Statistical approaches to detect anomalies, in *Emerging Research in Data Engineering Systems and Computer Communications*, edited by P. Venkata Krishna and M. S. Obaidat, pp. 499–509, Springer Singapore, Singapore, 2020.

Makhzani, A., and B. J. Frey, k-sparse autoencoders, *CoRR*, *abs/1312.5663*, 2014.

Margeloiu, A., N. Simidjievski, M. Jamnik, and A. Weller, Improving interpretability in medical imaging diagnosis using adversarial training, *CoRR*, *abs/2012.01166*, 2020.

Marsden, M., K. McGuinness, S. Little, and N. E. O'Connor, Resnetcrowd: A residual deep learning architecture for crowd counting, violent behaviour detection and crowd density level classification, *CoRR*, *abs/1705.10698*, 2017.

Mcculloch, W., and W. Pitts, A logical calculus of ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, *5*, 127–147, 1943.

McInnes, L., J. Healy, and J. Melville, Umap: Uniform manifold approximation and projection for dimension reduction, doi:10.48550/ARXIV.1802.03426, 2018.

Mehran, R., A. Oyama, and M. Shah, Abnormal crowd behavior detection using social force model, in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 935–942, doi:10.1109/CVPR.2009.5206641, 2009.

Metz, L., B. Poole, D. Pfau, and J. Sohl-Dickstein, Unrolled generative adversarial networks, *CoRR*, *abs/1611.02163*, 2016.

Michau, G., Y. Hu, T. Palmé, and O. Fink, Feature learning for fault detection in high-dimensional condition-monitoring signals, *CoRR*, *abs/1810.05550*, 2018.

Minsky, M., and S. A. Papert, *Perceptrons: An Introduction to Computational Geometry*, The MIT Press, 1969.

Muñoz, A., and J. M. Moguerza, One-class support vector machines and density estimation: The precise relation, in *Progress in Pattern Recognition, Image Analysis and Applications*, edited by A. Sanfeliu, J. F. Martínez Trinidad, and J. A. Carrasco Ochoa, pp. 216–223, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

Nazaré, T. S., R. F. de Mello, and M. A. Ponti, Are pre-trained cnns good feature extractors for anomaly detection in surveillance videos?, *CoRR*, *abs/1811.08495*, 2018.

Nguyen, H. V., and V. Gopalkrishnan, Feature extraction for outlier detection in high-dimensional spaces, in *Proceedings of the Fourth International Workshop on Feature Selection in Data Mining*, *Proceedings of Machine Learning Research*, vol. 10, edited by H. Liu, H. Motoda, R. Setiono, and Z. Zhao, pp. 66–75, PMLR, Hyderabad, India, 2010.

Noroozi, M., and P. Favaro, Unsupervised learning of visual representations by solving jigsaw puzzles, *CoRR*, *abs/1603.09246*, 2016.

O' Mahony, N., S. Campbell, A. Carvalho, L. Krpalkova, G. V. Hernandez, S. Harapanahalli, D. Riordan, and J. Walsh, One-shot learning for custom identification tasks; a review, *Procedia Manufacturing*, *38*, 186–193, doi:https://doi.org/10.1016/j.promfg.2020.01.025, 29th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM 2019), June 24-28, 2019, Limerick, Ireland, Beyond Industry 4.0: Industrial Advances, Engineering Education and Intelligent Manufacturing, 2019.

Pang, G., C. Shen, L. Cao, and A. van den Hengel, Deep learning for anomaly detection: A review, *CoRR*, *abs/2007.02500*, 2020.

Papers with code, Papers with code - imagenet benchmark (image classification), available online at https://paperswithcode.com/sota/image-classification-on-imagenet.

Paszke, A., et al., Pytorch: An imperative style, high-performance deep learning library, in *Advances in Neural Information Processing Systems 32*, edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, pp. 8024–8035, Curran Associates, Inc., 2019.

Patel, S., and J. Pingel, Introduction to deep learning: What are convolutional neural networks?, video] Available at:, 2017.

Pathak, D., P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, Context encoders: Feature learning by inpainting, *CoRR*, *abs/1604.07379*, 2016.

Pedamonti, D., Comparison of non-linear activation functions for deep neural networks on MNIST classification task, *CoRR*, *abs/1804.02763*, 2018.

Pedregosa, F., et al., Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research*, *12*, 2825–2830, 2011.

Pelleg, D., and A. Moore, Active learning for anomaly and rare-category detection, in *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, p. 1073–1080, MIT Press, Cambridge, MA, USA, 2004.

Phoha, V. V., *Internet Security Dictionary*, Springer-Verlag, Berlin, Heidelberg, 2002.

Pimentel, T., M. Monteiro, A. Veloso, and N. Ziviani, Deep active learning for anomaly detection, 2020.

Raina, R., A. Madhavan, and A. Y. Ng, Large-scale deep unsupervised learning using graphics processors, in *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, p. 873–880, Association for Computing Machinery, New York, NY, USA, doi:10.1145/1553374.1553486, 2009.

Ravanbakhsh, M., M. Nabi, H. Mousavi, E. Sangineto, and N. Sebe, Plug-and-play CNN for crowd motion analysis: An application in abnormal event detection, *CoRR*, *abs/1610.00307*, 2016.

Reddy, A., et al., Using gaussian mixture models to detect outliers in seasonal univariate network traffic, in *2017 IEEE Security and Privacy Workshops (SPW)*, pp. 229–234, doi:10.1109/SPW.2017.9, 2017.

Roberts, S., and L. Tarassenko, A probabilistic resource allocating network for novelty detection, *Neural Computation*, *6*(2), 270–284, doi:10.1162/neco.1994.6.2.270, 1994.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, *Learning Internal Representations by Error Propagation*, p. 318–362, MIT Press, Cambridge, MA, USA, 1986a.

Rumelhart, D. E., J. L. McClelland, and C. PDP Research Group, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, MIT Press, Cambridge, MA, USA, 1986b.

Sadr, A. V., B. A. Bassett, and M. Kunz, A flexible framework for anomaly detection via dimensionality reduction, *CoRR*, *abs/1909.04060*, 2019.

Sallehuddin, R., S. Ibrahim, A. Mohd Zain, and A. Hussein Elmi, Detecting sim box fraud by using support vector machine and artificial neural network, *Jurnal Teknologi*, *74*(1), doi:10.11113/jt.v74.2649, 2015.

Sambasivam, G., and G. D. Opiyo, A predictive machine learning application in agriculture: Cassava disease detection and classification with imbalanced dataset using convolutional neural networks, *Egyptian Informatics Journal*, *22*(1), 27–34, doi:https://doi.org/10.1016/j.eij.2020.02.007, 2021.

Schlegl, T., P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, Unsupervised anomaly detection with generative adversarial networks to guide marker discovery, *CoRR*, *abs/1703.05921*, 2017.

Schmidhuber, J., Deep learning in neural networks: An overview, *CoRR*, *abs/1404.7828*, 2014.

Schubert, E., A. Zimek, and H.-P. Kriegel, Generalized outlier detection with flexible kernel density estimates, in *Proceedings of the 2014 SIAM International Conference on Data Mining*, pp. 542–550, SIAM, 2014.

Schutte, K., O. Moindrot, P. Hérent, J.-B. Schiratti, and S. Jégou, Using stylegan for visual interpretability of deep learning models on medical images, 2021.

Settles, B., M. Craven, and S. Ray, Multiple-instance active learning, in *Advances in Neural Information Processing Systems*, vol. 20, edited by J. Platt, D. Koller, Y. Singer, and S. Roweis, Curran Associates, Inc., 2008.

Seung, H. S., M. Opper, and H. Sompolinsky, Query by committee, in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, p. 287–294, Association for Computing Machinery, New York, NY, USA, doi:10.1145/130385.130417, 1992.

Silver, D., et al., Mastering the game of go with deep neural networks and tree search, *Nature*, *529*, 484–489, doi:10.1038/nature16961, 2016.

Simonyan, K., and A. Zisserman, Very deep convolutional networks for large-scale image recognition, in *International Conference on Learning Representations*, 2015.

Song, X., M. Wu, C. Jermaine, and S. Ranka, Conditional anomaly detection, *Knowledge and Data Engineering, IEEE Transactions on*, *19*, 631–645, doi:10.1109/TKDE.2007.1009, 2007.

Steppan, J., Mnist database, [Online; accessed October 25, 2021], 2017.

Tailanián, M., P. Musé, and Á. Pardo, A multi-scale A contrario method for unsupervised image anomaly detection, *CoRR*, *abs/2110.02407*, 2021.

Thudumu, S., P. Branch, J. Jin, and J. Singh, A comprehensive survey of anomaly detection techniques for high dimensional big data, *Journal of Big Data*, *7*, 1–30, 2020.

Torralba, A., R. Fergus, and W. T. Freeman, 80 million tiny images: A large data set for nonparametric object and scene recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *30*(11), 1958–1970, doi:10.1109/TPAMI.2008.128, 2008.

van der Maaten, L., and G. E. Hinton, Visualizing data using t-sne, *Journal of Machine Learning Research*, *9*, 2579–2605, 2008.

Vanderplas, J. T., *In-Depth: Support Vector Machines*, O'Reilly, 2017.

Vapnik, V., and A. Y. Lerner, Recognition of patterns with help of generalized portraits, *Avtomat. i Telemekh*, *24*(6), 774–780, 1963.

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need, *CoRR*, *abs/1706.03762*, 2017.

Vincent, P., H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *Journal of Machine Learning Research*, *11*, 3371–3408, 2010.

Wang, L., X. Hu, B. Yuan, and J. Lu, Active learning via query synthesis and nearest neighbour search, *Neurocomputing*, *147*, 426–434, doi:https://doi.org/10.1016/j.neucom.2014.06.042, advances in Self-Organizing Maps Subtitle of the special issue: Selected Papers from the Workshop on Self-Organizing Maps 2012 (WSOM 2012), 2015.

Weng, J., N. Ahuja, and T. S. Huang, Cresceptron: a self-organizing neural network which grows adaptively, *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, *1*, 576–581 vol.1, 1992.

Willett, K. W., et al., Galaxy zoo 2: detailed morphological classifications for 304 122 galaxies from the sloan digital sky survey, *Monthly Notices of the Royal Astronomical Society*, *435*(4), 2835–2860, doi:10.1093/mnras/stt1458, 2013.

Wolpert, D., and W. Macready, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82, doi:10.1109/4235.585893, 1997.

Xian, Y., C. H. Lampert, B. Schiele, and Z. Akata, Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly, *CoRR, abs/1707.00600*, 2017.

Xu, D., E. Ricci, Y. Yan, J. Song, and N. Sebe, Learning deep representations of appearance and motion for anomalous event detection, *CoRR*, *abs/1510.01553*, 2015.

Yu, W., W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks, in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '18, p. 2672–2681, Association for Computing Machinery, New York, NY, USA, doi:10.1145/3219819.3220024, 2018.

Zeiler, M., G. Taylor, and R. Fergus, Adaptive deconvolutional networks for mid and high level feature learning, in *2011 International Conference on Computer Vision, ICCV 2011*, Proceedings of the IEEE International Conference on Computer Vision, pp. 2018–2025, doi:10.1109/ICCV.2011.6126474, 2011 IEEE International Conference on Computer Vision, ICCV 2011 ; Conference date: 06-11-2011 Through 13-11-2011, 2011.

Zeiler, M. D., and R. Fergus, Visualizing and Understanding Convolutional Networks, *arXiv e-prints*, arXiv:1311.2901, 2013.

Zenati, H., C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, Efficient gan-based anomaly detection, *CoRR*, *abs/1802.06222*, 2018.

Zhao, M., Introduction to active learning, [Online; accessed October 25, 2021], 2020.

Zhao, Q., and F. Karray, Anomaly detection for images using auto-encoder based sparse representation, in *Image Analysis and Recognition*, edited by A. Campilho, F. Karray, and Z. Wang, pp. 144–153, Springer International Publishing, Cham, 2020.

Zhou, Z.-H., A brief introduction to weakly supervised learning, *National Science Review*, *5*(1), 44–53, doi:10.1093/nsr/nwx106, 2017.

Zhu, J.-J., and J. Bento, Generative adversarial active learning, *arXiv preprint arXiv:1702.07956*, doi:10.48550/arXiv.1702.07956, 2017.

A1 | Appendix

This Appendix covers extra details about our choice of architecture for training and the full results of visualizing the feature space over 30 rounds of training.

A1.1 Custom Model Architectures for MNIST and CIFAR-10

We use the Pytorch framework (*Paszke et al.*, 2019) for defining our model architecture in all experiments. The layers of the model are defined by inheriting from the `torch.nn` module.

It is important to note that Pytorch initializes both the `nn.Conv2d` and `nn.Linear` layer with the Kaiming initialization (*He et al.*, 2015b).

A1.1.1 Model Architecture for MNIST

```

import torch.nn as nn

class NetMNIST(nn.Module):
    def __init__(self):
        super(NetMNIST, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 64)
        self.fc2 = nn.Linear(64, 3)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)

        x = self.conv2(x)
        x = F.relu(x)

        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = x.view(x.size(0), -1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)

        output = F.log_softmax(x, dim=1)
        return output
```

Listing A1.1: The model architecture for MNIST consists of 2 convolutional layers as well as some dropout and Linear layers

A1.1.2 Model Architecture for CIFAR-10

```

import torch.nn as nn

class NetCifar(nn.Module):
    def __init__(self):
        super(NetCifar, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 64)
        self.fc3 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 5 * 5)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)

        output = F.log_softmax(x, dim=1)
        return output
```

Listing A1.2: This architecture is very similar to the one defined for MNIST except for that fact it takes in a 3 channel input since the CIFAR-10 dataset consists of 3 channels.

A1.2 Visualization of Feature Space over 30 rounds

In Section 4.4.0.1, we discussed how the feature space evolves over 30 rounds. Figure 4.9 is a visualization of a UMAP reduction for round 1, 15 and 30 based on the CIFAR-10 dataset. This Appendix presents the full visualization from round 1 to 30.

Uniform Manifold Approximation and Projection (UMAP) (*McInnes et al.*, 2018) is a dimension reduction technique. It is similar to t-SNE (*van der Maaten and Hinton*, 2008) in that it can also be used for visualization.

The UMAP algorithm consists of two parts:

1. Learning the structure of the manifold in high dimensional space: In this step, the algorithm finds the nearest neighbours using the

Nearest-Neighbor-Descent algorithm (*Dong et al.*, 2011). It then construct a graph connecting the points to their nearest neighbours.

2. Finding a lower dimensional representation of this manifold: Here, the algorithm allows the user to specify a minimum distance between the points identified in step 1. This controls the spread of points in the lower dimensional space.

In our experiment, we extract a 64 dimensional vector from the feature space in each round of training. The vectors represent the learnt features from the input images.

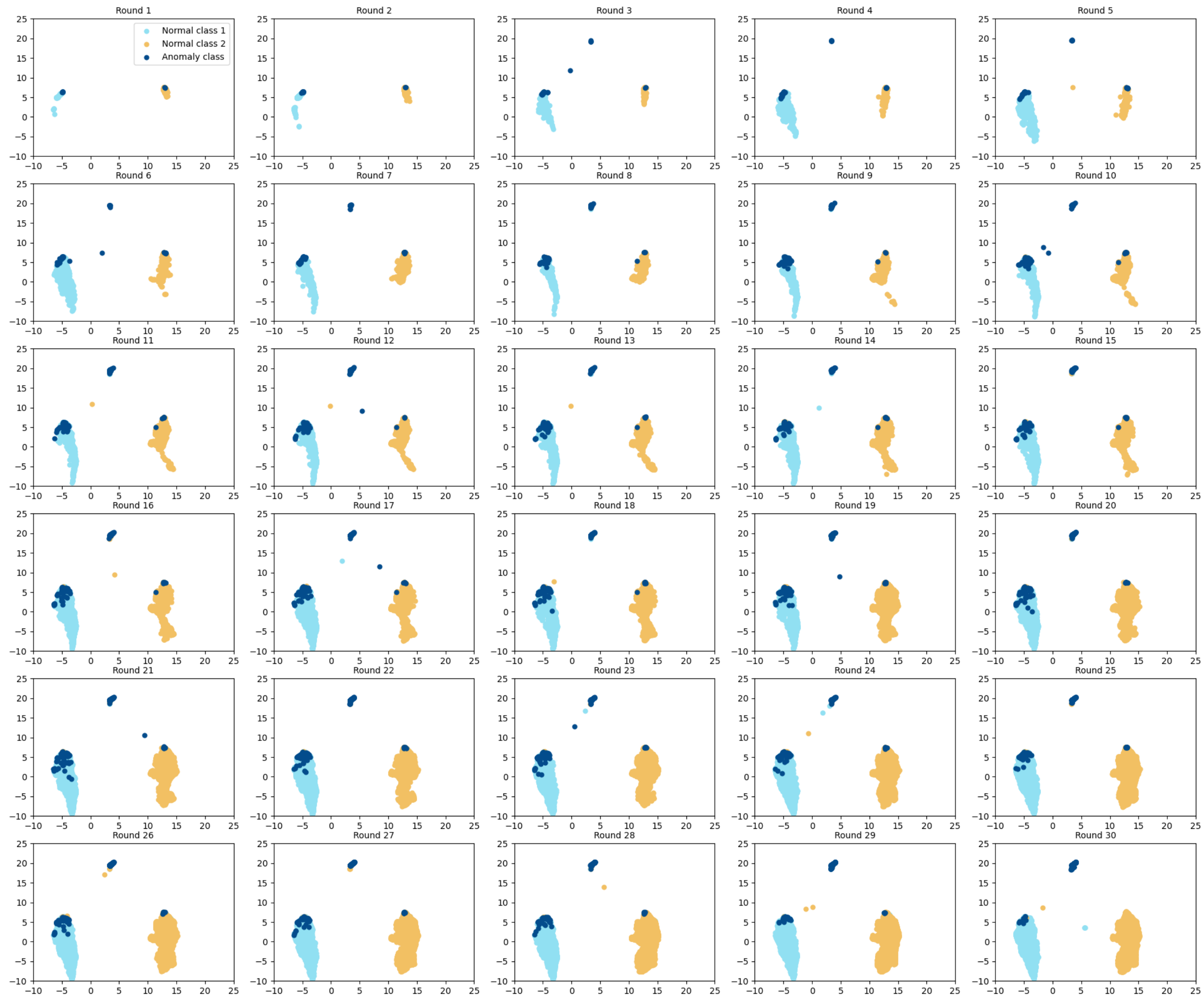


Figure A1.1: UMAP reductions for 30 rounds of training. In each round, there is a gradual separation of the anomalies (indicated in the deep blue dots) from the normal classes. This goes on until round 30 where the anomalies form a cluster and are completely separated from one of the normal classes (indicated in beige). It is also worth noting that, by round 30, there were fewer anomalies tightly coupled with the light blue dots (the other normal). A greater separation of the classes in the latent space indicates a separation of features which in turn helps with identifying anomalies.