BIRMINGHAM CITY UNIVERSITY

DOCTORAL THESIS

A Streaming Approach to Data Discrepancy Detection and Adaptation in Deep Neural Networks

---

Author:
Lorraine CHAMBERS

Supervisors:
Professor Mohamed GABER
Dr. Hossein GHOMESHI

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

in the

Data Analytics and Artificial Intelligence Research Group
College of Computing

Faculty of Computing, Engineering and the Built Environment

May 30, 2024

Declaration of Authorship

I, Lorraine CHAMBERS, declare that this thesis titled, "A Streaming Approach to Data Discrepancy Detection and Adaptation in Deep Neural Networks" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

BIRMINGHAM CITY UNIVERSITY

Abstract

Faculty of Computing, Engineering and the Built Environment
College of Computing

Doctor of Philosophy

A Streaming Approach to Data Discrepancy Detection and Adaptation in Deep Neural Networks

by Lorraine CHAMBERS

Deep learning has achieved remarkable success over the last ten years. However, keeping Deep Neural Networks (DNNs) up to date with changing data remains at the forefront of creating genuinely practical systems. To address some aspects of this challenge, this thesis studies the detection of streaming changing data and the subsequent adaptation of DNNs. The main challenges are to efficiently detect the changes and update the DNN promptly without forgetting the pertinent older information. This presents more of a challenge for DNNs than for other traditional machine learning models as DNNs take longer to train, require more data and suffer from catastrophic forgetting where previously learnt classes are forgotten when the DNN is adapted to the new data. DNNs are typically used with high dimensional unstructured data as opposed to lower dimensional structured data, which the streaming literature has focused upon thus far. Hence, DNN adaptation has not been widely studied in the streaming machine learning literature.

So far, clustering is the preferred method for detecting changes in data however, this can be slow as a number of instances must be received before a change is detected. A DNN is usually considered as a 'black box' where, given an input, it provides outputs, but the specific process by which it arrived at the output is not easily discernible. Inside this 'black box' are many artificial neurons which output values called activations. This thesis investigates if these activations can be used as a different representation of the input data and used as the input to streaming machine learning models to assist in detecting changing data and in DNN adaptation.

To address this, initially, this thesis proposes a method that handles outlier detection, adding an extra classification of 'unknown' to DNNs, so known classes are classified as their class and the detected changed data is classified as 'unknown'. Activations are extracted, providing a unique dynamic trajectory of activations for use with a streaming machine learning clustering model to detect outliers and label them as unknown. It is shown that the DNN activations can be used as a different representation of the input data and used with streaming machine learning techniques in order to detect outliers. Experiments show that our method outperforms the other leading open-set classification methods by a minimum of 2% and a maximum of 30% on the F1-Score and is between 5 and 100 times faster.

The outlier detection method leads onto offering the second contribution, which proposes a solution that handles the scenario of novel classes appearing in a stream of data (known as concept evolution) and DNN adaptation. A novel method of extracting DNN activations is used with an accuracy volatility concept evolution detection method and DNN adaptation process. Experiments show that our method outperforms other leading methods with regards to accuracy when placed in the concept evolution scenario with limited true-labelled data. The results of the experiments are analysed based on accuracy, speed of inference and speed of adaptation. On accuracy, our method outperforms the next best adaptation method by 27% and the next best combined novel class detection and CNN adaptation method by 24%. On speed, our method is within 1.5ms of the fastest inference speed and within 1.6s of the fastest DNN adaptation speed.

Thirdly, this thesis proposes a method that handles the more advanced problem of concept drift detection and DNN adaptation in drift pattern scenarios. DNN activations from multiple hidden layers are used with our novel ensemble drift detection and DNN adaptation method. Experiments show that our method overall outperforms other leading methods of detecting concept drift and DNN adaptation in all drift scenarios. We compare with eleven other leading methods of drift detection, adaptation and combined detection and adaptation methods. Our method outperforms other leading drift detection methods by between 8% and 46% on F1-Score, and other leading drift detection and adaptation methods by between 5% and 20% on accuracy. Our method is within 1.1ms of the fastest inference speed and 7 times faster than other adaptation methods.

These three methods culminate to provide the overall contributions of this thesis, which are: (1) The application of methods to extract activations from DNNs, providing more features than the input data, termed by us as activation classification footprints; and (2) applying these footprints to our own drift detection and DNN adaptation methods in order to detect and adapt to outliers, concept evolution and concept drift. The use of DNN activations in streaming machine learning models to detect data changes and in DNN adaptation offers a unique perspective in this research area and is a step forward towards realising fully adaptive continuous deep learning systems.

Acknowledgements

First and foremost, I would like to extend my sincerest gratitude to my supervisors, Prof. Mohamed Gaber and Dr. Hossein Ghomeshi, and my original second supervisor, Dr. Zahraa Abdallah, for their continuous support, guidance, and infinite patience throughout my PhD journey. Their expertise and insights have been invaluable. Thank you to Mohamed who was instrumental in my endeavour of a PhD instead of a Master's degree. Although at times I wished I had just pursued a Master's, now I am forever grateful and appreciative of this opportunity. Thank you to Hossein, whose perspective and encouragement was very much appreciated, especially having been through the same process only a few years previously. Also, thank you to Zahraa for the first year of insight, guidance and support.

Thank you to the BCU scholarship for the financial support that made this research possible and to the Doctoral Research College (DRC) for their administrative support. I am also thankful for the collaboration and support from colleagues and fellow PhD students at the Data Analytics and Artificial Intelligence (DAAI) research group, who made my PhD experience both enjoyable and memorable.

I would like to thank all of my friends endless patience, waiting for me to socialise, and sometimes insisting on socialising with me anyway, ensuring that I remained rounded. Thank you to Christina and Luzia for being you and making me laugh; to Jill, Lesha and Mandy for the holidays; and to Lucy, Ce, Ruby, Linda and others who have assisted me in a different perspective on everyday life that has helped tremendously in belief of myself and my PhD. Also, thank you to my wider group of friends in my village who have been asking for years if I have finished yet. Especially to Roger - no I haven't taken over the world yet! I could not face any of you if I didn't complete this, so thank you for the motivation.

A special thank you to my family, although shying away from the technical details, their support and words of encouragement have provided me with much comfort. A very special thank you to my partner Andy, who has been instrumental in this endeavour and who's unwavering patience, encouragement and support cannot be put into words, without whom, this would have been a far harder journey. Thank you to my stepson, Josh (who is just at the beginning of his PhD journey - you can do it!) and his wife, Ariel for their support and encouragement. And to my other stepson George, gone far too early, who understood far more about my PhD than he ever thought he did - thank you for being you and entertaining your father while I was too busy. There are too many people to mention all by name, but a sincerest thank you to everyone who has touched my life during this time.

Contents

List of Figures

List of Tables

List of Abbreviations

ADWIN	**A**daptive **WIN**dowing
AI	**A**rtificial **I**ntelligence
CBIR	**C**ontent **B**ased **I**mage **R**etrieval
CF	**C**ontinuous **F**orgetting
CNN	**C**onvolutional **N**eural **N**etwork
CPE	**C**NN based **P**rototype **E**nsemble
DD	**D**rift **D**etection
DDM	**D**rift **D**etection **M**ethod
DNN	**D**eep **N**eural **N**etwork
DS	**D**eep **S**tream
ER	**E**xperience **R**eplay
HDDMW	**H**oeffding **D**rift **D**etection **M**ethod using the statistical **W**-test
iCARL	**i**ncremental **C**lassifier **A**nd **R**epresentation **L**earning
JS	**J**ensen **S**hannon
JSDL	**J**ensen **S**hannon **D**ivergence **L**ast
KSWIN	**K**olmogorov-**S**mirnov **WIN**dowing
LwF	**L**earning without **F**orgetting
MCOD	**M**icro-cluster-based **C**ontinuous **O**utlier **D**etection
MINAS	**M**ult**I**class lear**N**ing **A**lgorithm for data **S**treams
MIR	**M**aximally **I**nterfered **R**etrieval
OCDD	**O**ne **C**lass **D**rift **D**etector
OCI	**O**nline **C**lass **I**ncremental
OCL	**O**nline **C**lass **L**earning
ODI	**O**nline **D**omain **I**ncremental
OOD	**O**ut **O**f **D**etection
PAC	**P**robably **A**utomatically **C**orrect
ReLU	**R**ectified **L**inear **U**nit
RSB	**R**eactive **S**ubspace **B**uffer
RV	**R**e**V**iew trick
SAM-kNN	**S**elf **A**djusting **M**emory k-**N**earest **N**eighbours
TENT	**T**est **ENT**ropy
UDA	**U**nsupervised **D**omain **A**daptation

Chapter 1

Introduction

1.1 Preamble

Humans have imagined or used intelligent machines dating back to philosophers and mathematicians from as long as three thousand years ago [155]. More recently, in the 1950's, Alan Turing explored the mathematical possibility of artificial intelligence and discussed how to build intelligent machines and how to test their intelligence [174]. However, at that time computers could be told what to do but couldn't remember what they did and were also extremely expensive [174]. A few years later, the term *Artificial Intelligence* was conceived by McCarthy et al. [124] and encompasses 'intelligent behaviour' in computers. Different techniques were investigated and as computers became faster, cheaper and more accessible, *Machine Learning* arose as a sub-field of Artificial Intelligence, using statistical modelling to automatically improve algorithms from data observations. Subsequently, in the big-data age, with the ability to store and process more data, *Deep Learning* emerged as a sub-field of machine learning.

Deep Learning is concerned with algorithms inspired by the structure and function of the brain and use models called Deep Neural Networks (DNNs). They require large amounts of data and computing power. DNNs are widely used in the data streaming field due to the volume of data available from sources such as Internet of Things (IoT) sensors [130]. The McKinsey's report on the global economic impact of IoT [126] estimates that the annual economic impact of IoT in 2030 could be up to $12.5 trillion globally in areas such as healthcare, home, retail environments,

offices, production environments, vehicles and cities. In many other real-world scenario's, the data that DNNs use is also often in a streaming environment with emerging data that the DNN has not seen before. In this thesis, this new unseen data is referred to as data discrepancies.

Data discrepancies affect performance in DNNs, causing incorrect classification. This can have catastrophic implications as DNNs are employed in important areas such as healthcare, finance and transportation. They are being used to develop systems that can help doctors diagnose diseases more accurately, such as cancer [112]. In finance, they assist in detecting serious crimes such as credit card and insurance fraud [2]. In transportation, DNNs are instrumental in developing self-driving cars and trucks, and exist in systems such as the Tesla Autopilot and Full Self-Driving features [182]. Incorrect classification is extremely critical in systems such as autonomous vehicles as this may result in catastrophic circumstances. In 2022 in America, self-driving systems were involved in 392 car crashes; six people were killed and five were seriously injured [134]. In 2021, The National Law Review (for America) reported that, "on average, there are 9.1 self-driving car accidents per million miles driven, while the same rate is 4.1 crashes per million miles for regular vehicles" [185]. Various safety incidents have been reported in the media such as Google's self driving car hitting a bus [106] and a Tesla driver's fatal crash [199]. In these circumstances, data discrepancy detection and recognition of previously unseen objects is essential.

The remainder of this introduction provides an overview of the main topics of this thesis in order to set the context. Section 1.2 briefly sets the scene and introduces streaming machine learning, DNNs and our definition of data discrepancies. Section 1.3 introduces the main research for data discrepancy detection and DNN adaptation alongside the key challenges. Sections 1.4 and 1.5 provide the problem definition and the aims and objectives. Section 1.6 lists the main contributions. Section 1.7 lists the publications that comprise the main chapters, with Section 1.8 providing an overview of the content of the thesis.

1.2 Background

1.2.1 Machine Learning

Machine learning encompasses many types of models. These models can be categorised into supervised, unsupervised and reinforcement learning. Supervised methods use labelled datasets to train models to those labelled outcomes. The two main applications for supervised methods are classification and regression problems. Classification converts an input to one or more discrete values. For instance, the dataset may be a set of images that also have the labels describing what those images are. Regression converts an input into a projected number (a continuous value). For instance, estimating the price of a property or a trend in stock prices. Unsupervised methods operate with unlabelled data, where the results are unknown. It is used to explore the structure of the information, discovering hidden patterns or groupings without the need for human labelling. For instance, grouping similar photographs together, or Google grouping news items with similar content [40]. Unsupervised learning also encompasses unsupervised transformations where a different representation of the data is created that might be easier for machine learning models or humans to understand. For instance, reducing data to two dimensions so it can be visualised [133]. Reinforcement learning involves acquiring knowledge about how to respond to situations (mapping situations to actions) in order to maximise a numerical reward. The learner is not explicitly instructed on which actions to take but instead must determine through trial and error which actions result in the highest reward. Actions have an impact on the immediate reward and may also influence the subsequent situation and other subsequent rewards. This trial-and-error search and delayed reward are important characteristics of reinforcement learning [179].

Typically, in classification, a static set of labelled data is used to train and test a model (classifier). Techniques that re-use or split data during training are used to assess the prediction ability of the model such as hold-out or cross-validation. Hold-out is a common technique which splits the dataset into two subsets: One for training the model and one for evaluating its performance, to estimate how well the

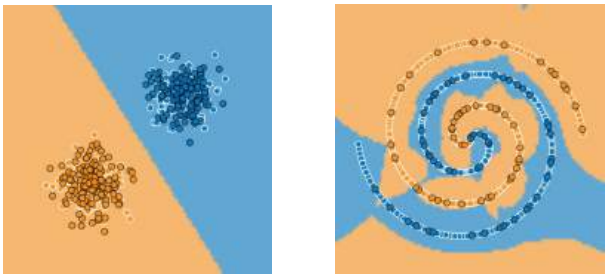
model is likely to perform on new, unseen data. Cross-validation is a re-sampling technique. It involves splitting the dataset into multiple subsets, or "folds" of equal size to repeatedly train and evaluate the model. The classifier will be fitted with the labelled training data and afterwards, test data is used to assess the classifiers performance. One field where classification is common is in Computer Vision, which focuses on analysing and understanding digital images and transforming them into descriptions [53], with deep learning playing an essential role [181].

In many real-world applications, data is in the form of data streams. Handling data streams is more challenging than static datasets for the following reasons: (1) Data streams contain large amounts of continuous data points with time and memory constraints. Each data point is only processed once. (2) Static data techniques such as holdout or cross-validation are not suitable for data streams due to infinite size and high volume of data, where training cannot be independent from testing the updated model. (3) The underlying distribution of the data often changes. These reasons have resulted in the development of new streaming models or modification of existing static models to facilitate learning from data streams [65]. Often streaming machine learning models will be trained with static data (offline) and retrained using new data from the changing data stream (online). In this thesis, changing data is referred to as Data Discrepancies and the retraining of models is called Adaptation. This thesis focuses on detecting data discrepancies and adapting DNNs to this new data. These DNN models contain multiple layers with each layer consisting of many artificial neurons.

1.2.2 Deep Neural Networks

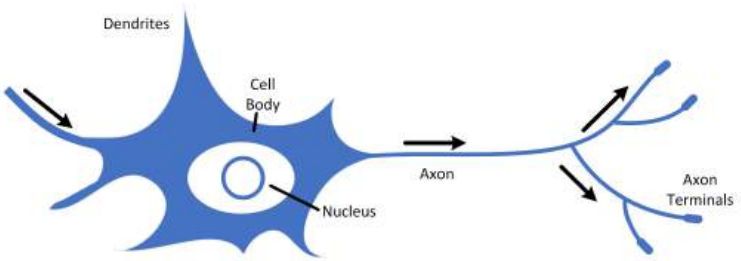
Many machine learning classification methods can only divide classes via a linear boundary as shown in Figure 1.1 (a) whereas DNNs are able to provide highly non-linear classification boundaries as shown in Figure 1.1 (b).

The artificial neuron is inspired by the biological neuron. Figure 1.2 (a) shows a biological neuron and Figure 1.2 (b) shows an artificial neuron as used as the simplest element of a DNN. In a biological neuron, signals are received from the dendrites, and sent down the axon when a high enough signal is received. This outgoing signal can then be used as another input for other neurons, repeating the process.

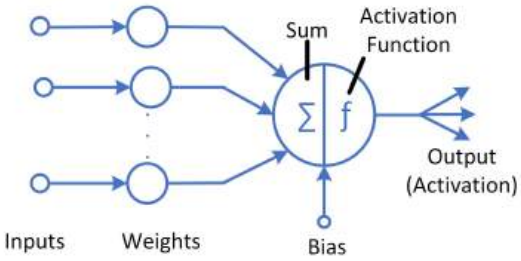


(a) Linear Prediction (b) Non-Linear Prediction

FIGURE 1.1: Linear and Non-Linear Classification Boundary Examples.



(a) Biological Neuron

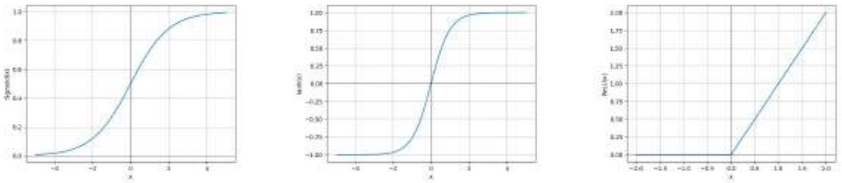


(b) Artificial Neuron

FIGURE 1.2: Representation of a biological and an artificial neuron.

Some signals are more important than others and can trigger some neurons to fire more easily. Connections can become stronger or weaker and new connections can appear while others disappear [51]. The artificial neuron mimics most of this process by having weighted input signals into an activation function and outputs a signal if the sum of these weighted inputs reach a certain bias. The output of the activation function is a numerical value called an activation. Common activation functions are Sigmoid, Tanh (hyperbolic tangent) and ReLU (Rectified Linear Unit) [67]. Early neural networks used Sigmoid and Tanh, with ReLU being recommended for most networks [67]. Figure 1.3 shows the aforementioned types of activation functions. The performance of a DNN can be enhanced by the selection of the activation

function, with non-linear functions able to produce a highly non-linear classification boundary. A DNN consists of many of these activation functions. They combine to produce an overall function that culminates into a complex classification boundary depending on the number of the artificial neurons. DNNs require large amounts of data and computing power but have dramatically improved speech recognition and visual object recognition amongst many other domains [105].



(a) Sigmoid (b) Tanh (c) ReLU

FIGURE 1.3: Types of neural network activation functions.

Deep learning models are engineered systems inspired by the biological brain but they are generally not designed to be realistic models of biological function [67]. Not enough is known about biological learning in neuroscience to be able to offer much guidance for the learning algorithms. The general media often emphasise the similarity of deep learning to the brain. However, deep learning should not be viewed as an attempt to simulate the brain [67].

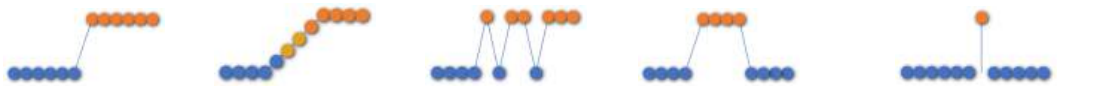
DNNs were designed for a static data environment, where all data is available in advance. They can only predict classes they are trained on. Hence, if a DNN was trained to predict labels for images of cats and dogs; if an image of a frog was presented, the DNN can only predict a cat or a dog. In most real-world situations, data is streaming and changes over time. This means that the DNN could incorrectly classify instances. Confidence is required in deep neural networks operating in a streaming environment so that unseen instances that vary from the training data will be automatically captured.

1.2.3 Data Discrepancies

In real-world scenarios involving machine learning, data often evolves over time. This changing data plays a pivotal role in both challenging and refining the capabilities of algorithms. Handling this changing data is a critical aspect of machine

learning because it can lead to skewed model performance and inaccurate predictions. The ability to adapt and learn is what ultimately enhances the robustness and reliability of machine learning systems in real-world applications. The data discrepancies that this thesis addresses are Outlier Detection, Concept Evolution and Concept Drift. Outlier detection is concerned with identifying instances that stand out from the training data [29]. Concept Evolution is where a completely new class is seen that the deep neural network was not trained on. Concept Drift is where changes in data cause incorrect classification but no new class arises (this could be in patterns of incremental, recurrent, gradual or abrupt) [59]; Figure 1.4 shows a graphical representation of the drift patterns.

Abrupt patterns, as shown in Figure 1.4 (a) is where the concept changes from one concept to another suddenly. A real-world example of this may be the replacement of a sensor in an industrial setting, where the new sensor has a different calibration. The incremental pattern, as shown in Figure 1.4 (b) is where the change consists of many intermediate concepts (i.e. a sensor slowly fails and becomes less accurate). The gradual pattern, as shown in Figure 1.4 (c) is where one concept changes into another slowly (i.e. a user of a house advertising outlet may have an interest in looking to buy a home but may still have an interest in rental homes and keeps going back to the previous interest for some time). The reoccurring pattern as shown in Figure 1.4 (d) is where previously seen concepts may reoccur after some time (i.e. in cyclically reoccurring fashions) [59]. An outlier is shown in (e) for comparison. This is not a drift pattern but a one-off instance that deviates from the expected data.



(a) Abrupt (b) Incremental (c) Gradual (d) Reoccurring (e) Outlier (not a drift pattern)

FIGURE 1.4: Patterns of data drift applied over time. Colours represent different distributions (adapted from [57]).

Detection of discrepancies can be performed at different levels such as at instance-level or at the window-level. At instance level, each individual instance is examined to ascertain if it is outside the expected norm i.e. outlier detection. At window-level, a window of data is analysed collectively via i.e. a statistical summary method and

a decision as to whether a discrepancy occurred in that window is made, but no particular instances are identified.

1.3 Motivation

Whilst DNNs can be applied to low-dimensional data, most real-world scenarios are high dimensional, such as images, audio, text, and other complex data types. DNNs excel at automatically learning hierarchical representations and features from this raw data. This makes them natural feature extractors where the input data is transformed into sets of features that can be captured as the output of the intermediate layers. The streaming machine learning field has tended to focus on lower dimensional structured data [56] where the data is in an organised format, typically tabulated and composed of attributes. Little research has been dedicated to high dimensional unstructured data where the data is not organised in a pre-defined manner such as images, audio and text. This is especially true for DNN adaptation within a streaming environment. To handle high dimensional data, the data is commonly transformed into a different representation to improve the intra-class cohesion and the inter-class separation [85, 191, 60, 195]. A DNN is usually considered as a 'black box' where, given an input, it provides outputs but the specific process by which it arrived at the output is not easily discernible. Inside this 'black box' are many artificial neurons as described in Section 1.2.2, which output numerical values called activations. These activations are employed in the DNN visualisation and interpretability fields where the aim is to understand what the network has learned, how it processes data and how DNNs come to their classification [88, 87]. This thesis investigates if a different representation of the input data can be extracted from the activations and applied to streaming machine learning models to detect data discrepancies and adapt the classifying DNN.

Clustering is commonly used in concept drift detection [214]. It is an implicit method of drift detection, meaning that the change in data is monitored over a number of instances before drift is declared. This is in contrast to explicit drift detection where the change is detected and immediately reported. Implicit methods of drift detection result in a delay of concept drift detection. In data stream classification,

there are very few systems where a classifying DNN is adapted to high dimensional data [101]. This means that DNN adaptation has not been studied in depth in the concept evolution or concept drift field. On the contrary, the Online Continuous Learning (OCL) field has a large number of DNN adaptation techniques [122]. However, they have not been applied to the more dynamic concept drift setting. To the best of our knowledge, only one paper started to address this by unifying OCL with concept drift adaptation [101]. However, drift patterns were not applied.

1.4 Problem Definition

The following issues exist in streaming data and DNN adaptation systems: In the streaming machine learning literature, DNN adaptation of the classifying DNN has not been widely studied as compared to traditional machine learning models [101]. This is because DNNs take longer to adapt [205], require more data, suffer from class imbalance and retraining DNNs causes catastrophic forgetting [122]. Hence, there are few streaming systems that adapt a classifying DNN. Those that do, use implicit drift detection which can be slow as compared to explicit drift detection.

1.5 Aims and Objectives

This thesis proposes data discrepancy detection and DNN adaptation to streaming data with the following aims:

1. Detecting data discrepancies in DNNs via neural activations for data streams.
2. Adapting DNNs in the presence of data discrepancies for data streams.

To achieve these aims, the following objectives have been identified:

1. Critically review data discrepancy detection and DNN adaptation methods.
2. Design, develop and evaluate an Open-Set (outlier) discrepancy detection method.
3. Design, develop and evaluate a concept evolution discrepancy detection and DNN adaptation method.
4. Design, develop and evaluate a concept drift discrepancy detection and DNN adaptation method.

1.6 Contributions

Achieving the objectives of this thesis has led to the advancement of the state-of-the-art in streaming techniques with deep neural networks with the following contributions:

DeepStreamOS: Open-Set Classification in Deep Neural Networks

To satisfy objectives one and two, Chapter 4 proposes *DeepStreamOS* which brings together the use of deep neural network activations with a stream-based outlier detection method for fast identification of instances that belong to unknown classes. DeepStreamOS uses all layers of a Convolutional Neural Network (CNN) to get a trajectory of the activations and applies a stream-based analysis method to determine if an instance belongs to an unknown class. Our contributions are: (1) The application of a statistical method to quantify the difference in activation distribution between any two consecutive hidden layers of deep neural networks to get a dynamic trajectory of activations; and (2) fast interpretation of the reduced activations via a stream-based outlier detection method to detect unknown images.

AdaDeepStream: Streaming DNN Adaptation to Concept Evolution

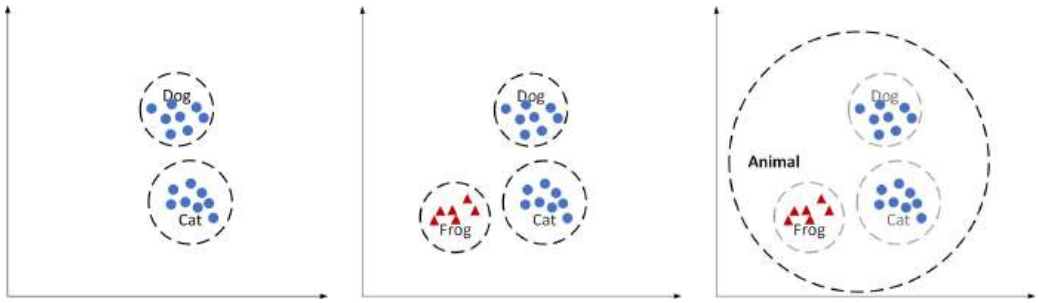
To satisfy objectives one and three, Chapter 5 proposes *AdaDeepStream* which offers a dynamic concept evolution detection and CNN adaptation system using minimal true-labelled samples. Our contributions are: (1) Heuristics for activation reduction of deep neural networks via two methods to apply to concept evolution detection. One of these methods expands upon the activation reduction used in *DeepStreamOS*, the other is an image descriptor-based method; (2) concept evolution detection using neural network activations and streaming machine learning models. In contrast to the outlier detection used in *DeepStreamOS*, we introduce our unique accuracy volatility-based detection method; (3) *DeepStreamOS* did not involve adaptation. In this contribution, we expand the work into CNN adaptation involving neural network activations and streaming machine learning models; and (4) analysis of CNN adaptation techniques in a concept evolution setting.

DeepStreamEnsemble: Streaming DNN Adaptation to Concept Drift

This satisfies objectives one and four. In Chapter 6 *DeepStreamEnsemble* proposes detection and adaptation to concept drift with an experimental study on images where novel sub-classes are arising. In contrast to the concept evolution aspect of *AdaDeepStream*, *DeepStreamEnsemble* focuses upon the more complex scenario of concept drift. Our contributions are: (1) DNN activation reduction method based on image descriptors, expanding upon the method used in *AdaDeepStream*, as input to concept drift detection; (2) explicit concept drift detection method using multi-layer DNN activations and streaming machine learning ensemble; and (3) DNN adaptation method using multi-layer DNN activations and streaming machine learning ensemble.

The experimental setting of this thesis focuses on the CNN as the type of DNN and images for the input data. For images, concept drift is commonly applied as i.e. new background, blur, noise, illumination and occlusion [122, 195]. These are synthetic changes to the images which do not replicate real-world scenarios. We aim to introduce sub-classes of data using real images. For instance, if a DNN classifies images of cats and dogs as Animals, and another animal such as frog is introduced, the classification of Animal already exists, but a frog has not previously been seen by the DNN. Therefore, the input data for the Animal class has drifted from the training data with the addition of the frog image. Figure 1.5 shows (a) the original classes of Dog and Cat. In (b) a new class of Frog is added, resulting in a new class boundary for Frog. In (c) a new sub-class of Frog is added to the Animal class, resulting in no class boundary changes for Animal. Thus, we contribute the use of novel sub-classes as concept drift to provide a scenario of changing real images instead of synthesised ones.

As another contribution, this thesis also brings together the CNN adaptation solutions offered in the Online Continuous Learning field (OCL) with the fields of concept evolution and concept drift via the application of DNN adaptation solutions and data drift patterns in *AdaDeepStream* (Chapter 5) and *DeepStreamEnsemble* (Chapter 6).



(a) Original classes (no discrepancy) (b) New class (concept evolution) (c) New sub-class (concept drift)

FIGURE 1.5: Class and Sub-Class Classification Boundary Examples.
Legend: Blue (circles) are known instances. Red (triangles) are unknown instances.
Black dashed line is a class boundary. Grey dotted line is a sub-class boundary.

1.7 Publications

The following journal papers have been published in contribution to this thesis:

1. L. Chambers, and M.M. Gaber, DeepStreamOS: Fast open-set classification for convolutional neural networks, in Pattern Recognition Letters, vol. 154, pp. 75-82, Elsevier, 2022, ISSN: 0167-8655.

2. L. Chambers, M.M. Gaber and H. Ghomeshi, AdaDeepStream: streaming adaptation to concept evolution in deep neural networks. Applied Intelligence, vol. 53, pp. 27323–27343, Springer, 2023, ISSN: 1573-7497.

3. L. Chambers, M.M. Gaber, and H. Ghomeshi, DeepStreamEnsemble: Streaming Adaptation to Concept Drift in Deep Neural Networks. In communication.

1.8 Thesis Outline

The remainder of this thesis is organised as follows: Chapter 2 provides an overview of the related work in data discrepancy detection and DNN adaptation. Detection techniques for identifying unknown instances are reviewed followed by the techniques for discrepancy detection and CNN adaptation for streaming images. To gain further knowledge on CNN adaptation techniques, the OCL and streaming DNN learning fields are reviewed with respect to CNNs. These detection and adaptation techniques are summarised in a novel taxonomy, categorising them via their

detection technique, catastrophic forgetting mitigation method and which parts of the CNN is adapted.

Chapter 3 provides a background and theoretical explanation of the models and algorithms employed or modified in this thesis. An overview of DNN architecture is given in order to explain DNN activations. Pre-existing statistical methods, streaming machine learning algorithms and other algorithms used within our novel solutions are explained and discussed.

Chapter 4 presents the *DeepStreamOS* method as our contribution to objective two. This method provides an algorithm that is focused upon instance-based detection and detects data discrepancies for outliers applied as novel classes and subclasses.

Chapter 5 presents the *AdaDeepStream* method as our contribution to objective three. This method is built upon *DeepStreamOS*, focusing upon concept evolution, adding DNN adaptation and moving from instance-based detection to window-based detection.

Chapter 6 presents the *DeepStreamEnsemble* method as our contribution to objective four. This method focuses upon the more challenging problem of concept drift and DNN adaptation.

Chapter 7 summarises the work presented in this thesis and draws conclusions from the presented algorithms. It contains a brief summary of the novel methods and specifies important directions for future work.

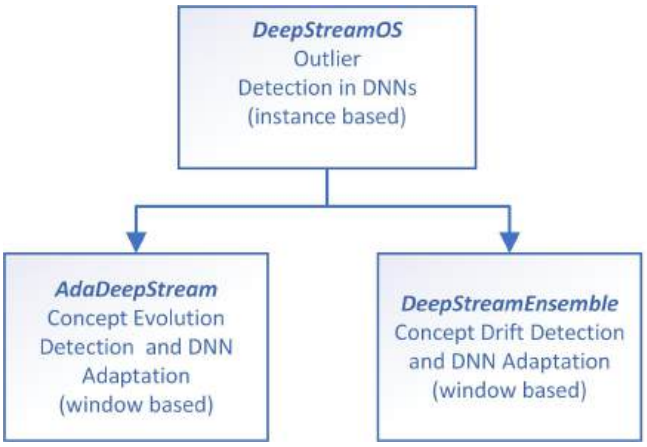


FIGURE 1.6: Hierarchical representation of the contributions to this thesis

Figure 1.6 shows a hierarchical representation of the contributions to this thesis.

DeepStreamOS lays the groundwork by investigating DNN activations in outlier detection in DNNs, operating on a per-instance detection basis. *AdaDeepStream* and *DeepStreamEnsemble* progresses this by moving into a window-based detection context. *AdaDeepStream* detects concept evolution and additionally adapts the DNN. *DeepStreamEnsemble* progresses onto the more advanced problem of concept drift detection and also adapts the DNN.

Chapter 2

Data Discrepancy Detection and DNN Adaptation: A Review

This chapter presents an extensive review of state-of-the-art algorithms designed for detecting discrepancies and adapting DNNs accordingly. In our experimental methodologies in Sections 4.3, 5.3 and 6.3, the type of DNN employed is the CNN with images as the high dimensional data. Therefore, we focus our review in this direction. Firstly, in Section 2.2, the open-set recognition field is reviewed as this is the setting for our instance-based outlier discrepancy detection contribution of *DeepStreamOS* in Chapter 4. Section 2.3 then progresses on to review the concept evolution and concept drift discrepancy detection and adaptation fields. Section 2.4 reviews online CNN adaptation. Both of these sections provide the literature review for our concept evolution and concept drift discrepancy detection and CNN adaptation contributions of *AdaDeepStream* and *DeepStreamEnsemble* in Chapters 5 and 6 respectively. Additionally, a unique taxonomy of discrepancy detection and CNN adaptation methods for streaming images is proposed. This categorises the current methods based on their detection and adaptation approaches. This chapter aligns with the first objective outlined in Chapter 1.

2.1 Introduction

DNNs are widely used and have achieved state-of-the-art performance in static data classification tasks [105, 180]. However, data evolves in real-world scenarios and standard DNNs are not responsive to changing data. DNNs only recognise classes they are trained on. Therefore, novel classes are attributed to known labels from the

training data. This will result in incorrectly classified instances. Non-responsiveness to changing data could be dangerous in safety-critical applications. Detection and adaptation to changing data in these circumstances is essential.

In this thesis, we consider three discrepancy types of outlier detection, concept evolution and concept drift. Where outliers are instances that deviate from the expected data. Section 3.6 gives a formalisation and explanation of concept evolution and concept drift. To understand the terminology, a historic context of the field is required. The term 'concept drift' was first introduced in 1986 by Schlimmer and Granger Jr [169] and is an overall term concerned with changes in data stream distribution over time. There are two types of concept drift: Real and Virtual [59]. With real concept drift, changes occur in the data stream that cause the decision boundary to change. With virtual concept drift, changes occur in the data stream that do not change the decision boundary.

There are a number of existing surveys, the major ones are by Gama in 2014 [59] and Ditzler [45] in 2015. More recently, in 2018, Khamassi brings together related ideas from different disciplines for addressing concept drift [95] and explains that the type of concept drift that new classes cause is called Class Prior Concept Drift and has only been used as a term since 2015 [95]. Originally, novel class emergence was considered as a type of real concept drift. Novel class emergence is also known as concept evolution in data stream literature. Therefore, terminology as detailed in Section 3.6 is: (1) Concept evolution is class prior concept drift of the novel class emergence type; and (2) concept drift is real concept drift (there are changes in input data that change existing class boundaries without forming new class boundaries) and virtual concept drift (there are changes in input data with class boundaries staying the same). When novel classes appear in a data stream in accumulated instances, this data drift is called concept evolution [43]. When there is a distribution change within the existing classes only and no new classes arise, this data drift is called concept drift [57]. In this thesis we have termed outlier detection, concept evolution and concept drift as data discrepancies.

2.2 Open-Set Recognition

This section focuses upon open-set recognition. This is the setting for our proposed method *DeepStreamOS* in Chapter 4 to demonstrate outlier discrepancy detection. Open-set recognition extends the traditional classification task by recognising instances from known classes while also detecting instances that belong to unknown or novel classes. This concept acknowledges that in real-world scenarios, models may encounter instances that were not seen during training. It aims to provide accurate predictions for known classes and properly handle instances from new or untrained classes by labelling them as 'unknown' [13].

Open-Set recognition methods can be split into two types of models: Generative and discriminative. In the generative models, training data is added in order to augment the open space. In discriminative models, a border is attempted to be created around the known classes to separate them from the open space [63]. Discriminative methods can be statistical, deep learning based or a combination of the two. The first open-set detection methods were statistical and used Support Vector Machines (SVM). However, SVMs separate the classes with hyperplanes and were not sufficient to discriminate the open space on their own.

Subsequently, Extreme Value Theory (EVT) was added to SVMs [168], improving the outcome. EVT estimates the probability of the instance being an outlier with respect to each class and uses a rejection threshold. Rudd proposed EVM [162] which is derived from EVT and is the radial probability of inclusion of a point with respect to the class of another. In combination methods, EVT is now a common post-processing step. However, EVT methods use thresholds and when the unknown image is only slightly different from the known images there is more of a risk of miss-classifying known images as unknown. Thresholds also require tuning. To improve upon this, distance measures can also be used and recently, clustering has been applied to EVM [82].

Deep learning based methods either use outputs from the deep neural network under test or incorporate deep learning in other processing steps. The simplest method is thresholding the Softmax scores of the DNN. However, this is rejecting uncertain predictions, rather than unknown classes [13]. Subsequently, OpenMax [13]

used activation patterns from the final hidden layer of the DNN. The theory being that the output of the penultimate layer of the network is far away from the feature space of the images the network was trained on and are easier to capture. EVT was then applied to determine if the instance belongs to a class. Following OpenMax, DOC [172] used output layers of Sigmoids rather than Softmax to reduce the open space risk by tightening the decision boundaries of Sigmoid functions with Gaussian fitting. More recently Generative Adversarial Networks (GANs) are used to generate instances to fill the open space. For instance, Classification-Reconstruction learning for Open-Set Recognition (CROSR) [203] reconstructs input samples from low dimensional latent representations. However, any technique that utilises deep learning in the processing requires considerable computational resources and requires additional parameters to tune the system.

Generative, discriminative, statistical and deep learning based open-set methods are all ongoing areas of research with recent practical applications in 3D object recognition [14] and research in further directions with Positive-Unlabelled learning and Open-Set adaptation [115]. In safety-critical applications such as for the automotive industry, the computational cost during inference is particularly important as there are limited computational resources and real-time constraints. Detection of unknowns is used to trigger a safe fallback mode where a different monitoring system is employed [136] therefore, fast detection of the unknowns is required.

In summary, deep learning based open-set methods are increasing. Activations from within the DNN are used and they tend to be from the final hidden layer and are applied to an alternative output layer than the traditional Softmax layer to reduce the open-space risk. Detecting unknowns is used in safety-critical applications, indicating that they need to be detected promptly. However, in the open-set classification field, the focus has not been on speed.

2.3 Data Discrepancy Detection and Adaptation for Streaming Images

This section focuses upon concept evolution and concept drift detection and adaptation to these discrepancies. This is the setting for our proposed methods of *AdaDeepStream* in Chapter 5 and *DeepStreamEnsemble* in Chapter 6.

Existing approaches for DNN adaptation to concept evolution and concept drift are limited as in the streaming field, the focus has been on other types of traditional machine learning classifiers and have been mainly assessed with low-dimensional data [56]. The slow uptake of addressing DNNs in the streaming literature results from the specific challenges that DNNs pose. DNNs are typically implemented for high dimensional unstructured data such as text, images or audio due to the superior non-linear classification boundaries they are able to attain, as previously explained in Section 1.2.2. To achieve this, DNNs require large amounts of data, and high data dimensionality contributes to the adaptation latency being large [205]. DNN adaptation can also cause catastrophic forgetting, where originally known classes are forgotten in the presence of new classes [122]. DNNs also require balanced classes for training which are not available in streaming scenarios when novel classes are emerging. This is exacerbated by DNNs requiring a larger amount of data for training as compared to other types of machine learning models.

To achieve online DNN adaptation, solutions often require prior selection of specialised DNN architectures or loss functions, enforcing retraining of the DNN. For instance, [192] and [15] require extra layers to be added to the CNN prior to training. Therefore, in a number of solutions, data discrepancy detection and adaptation cannot be retrospectively applied without completely retraining the DNN on the initial data. This is particularly an issue in DNNs that take a long time to train. The requirement of online adaptation may not always be realised at the time of original implementation of the DNN system. If such a system is not in place, metrics would need to be manually monitored, data collected and labelled, and a new model statically trained.

To handle high dimensional data, the data is typically transformed into a different representation to improve the separation of the classes [85, 191, 60, 195]. Clustering is commonly used in data discrepancy detection [101, 195, 60, 42, 73, 22, 207]. It is an implicit method of drift detection meaning that the drift is monitored over a number of instances before drift is declared. This is in contrast to explicit drift detection where the change is detected and immediately reported. Implicit methods of drift detection tends to result in a delay of data discrepancy detection.

The following sections further explore the solutions available for concept evolution detection and concept drift detection. Methods that are reviewed here are designed for, or have been successfully used with CNNs and image data.

2.3.1 Concept Evolution

There are numerous methods that are able to detect concept evolution (also known as streaming novel class detection) and adapt to images. However, to the best of our knowledge, Reactive Sub-space Buffer (RSB) [101] is the only method that detects novel classes and directly adapts a classifying CNN. Other systems have DNNs involved in the detection of novel classes such as CNN based Prototype Ensemble (CPE) [195] and Convolutional open-world multi-task image Stream classifier with Intrinsic similarity Metrics (CSIM) [60].

There are a larger number of other methods that adapt to novel classes in images that do not involve any kind of DNN which are: EMC [42], SAND [73], SENNE [22], KNNENS [207], ECSMiner [123], SEEN [212], SENCForest [131], SACCOS [61], SENCMas [132] and Echo-D [74]. Table 2.1 shows a summary of approaches for novel class detection and adaptation for images. From this, it can be seen that ten out of the thirteen methods use clustering.

For the CNN adaptation, if the method contains a CNN, the CNN usage is defined, with 'Classify' signifying that the CNN is the main image classifying CNN and 'Detect' signifies that a CNN is used in the detection of concept evolution. In the CNN Adaptation column, 'Full' signifies that all layers of the CNN are adapted. The only method that adapts the classifying CNN is RSB, which retrains the entire CNN with identified concept evolution instances that it has stored in memory.

TABLE 2.1: Overview of the approaches for concept evolution detection and adaptation solutions for streaming images

Method		Detection Method	Detection Type	CNN Usage	CNN Adaptation
CPE	[195]	Clustering	Implicit	Detect	Full
CSIM	[60]	Clustering	Implicit	Detect	Full
Echo-D	[74]	Clustering	Implicit	-	-
ECSMiner	[123]	Clustering	Implicit	-	-
EMC	[42]	Clustering	Implicit	-	-
KNNENS	[207]	Clustering Ensemble	Implicit	-	-
RSB	[101]	Clustering	Implicit	Classify	Full
SACCOS	[61]	Clustering Graph	Implicit	-	-
SAND	[73]	Outliers	Implicit	-	-
SEEN	[212]	Clustering Forest	Implicit	-	-
SENCForest	[131]	Forest	Implicit	-	-
SENC-MaS	[132]	Matrix Sketching	Implicit	-	-
SENNE	[22]	Clustering Ensemble	Implicit	-	-

2.3.2 Concept Drift

Some of the systems specified for detection of concept evolution also detect concept drift. Of those that detect both, some may use different methods for each type of detection. Table 2.2 provides an overview of leading concept drift and adaptation solutions for streaming images. To the best of our knowledge, in the concept drift field, RSB [101] and Change Detection Test-CNN (CDT-CNN) [44] are the only methods that detect concept drift and adapt the classifying CNN. RSB uses the same clustering method to detect concept evolution and concept drift. It uses centroid clustering, and a reactive subspace buffer tracks drift. The CNN adaptation is memory-based and stores diverse samples only. This method uses implicit drift detection suggesting a slow response to changing data. Whereas, CDT-CNN detects concept drift via CUSUM [142], a sequential analysis technique that is used to detect changes in a process or signal. The authors of CDT-CNN express that the drift detection is not the contribution, but the adaptation of the CNN. When the drift is detected, the layers of the CNN that have become obsolete due to the concept drift are identified and only those layers are retrained. CPE [195] detects concept drift using a clustering method but employs a CNN in the detection of concept drift in images but does not directly adapt a classifying CNN.

There are a larger number of other methods that adapt to concept drift in images that do not involve a DNN: Evolving Micro-Clusters (EMC) [42] which is a clustering method; Semi-Supervised Adaptive Novel Class Detection (SAND) [73] where the concept drift technique differs from the concept evolution detection technique and uses a clustering ensemble; Enhanced Classifier for Data Streams with novel class Miner (ECSMiner) [123] which also uses a differing concept drift detection technique via an ensemble; Semi-supervised Adaptive ClassifiCation Over data Stream (SACCOS) [61] which is a clustering graph technique and Efficient handling of concept drift and concept evolution over Stream Data (Echo-D) [74] which uses a clustering ensemble for concept drift detection.

Most solutions for adapting to concept drift in images use implicit clustering. Only CDT-CNN and ECSMiner's concept drift detection method does not use clustering. CDT-CNN uses an off-the-shelf drift detector based on error detection and ECSMiner's drift detection aspect uses an error detection based ensemble.

The Deep Unsupervised Domain Adaptation (UDA) field, has also yielded promising results regarding natural image processing, video analysis, natural language processing, time-series data analysis, and medical image analysis [114]. Batch normalisation is added to the layers of the DNN for fast adaptation with reduced overfitting. Batch Normalisation-Norm (BN-Norm) [15] recomputes the normalisation statistics during test time [15]. Test ENTropy (TENT) [192] re-estimates these statistics but also optimises transformation batch normalisation parameters using a single backpropagation pass during prediction [192]. This would be fast with regards to the adaptation of the CNN but may be limited in the types of data drift it can successfully adapt to. Table 2.2 shows a summary of the approaches for concept drift detection and adaptation. It can be seen that of the ten concept drift detection techniques, six of them use clustering. If the method contains a CNN, the CNN usage is defined. In this case, we have four methods that adapt a classifying CNN, and one method that uses a CNN to detect concept drift.

In summary, There are limited methods for combined data discrepancy detection and CNN adaptation methods for streaming images. This is because DNNs have not been widely studied in streaming scenarios due to their high adaptation latency, their catastrophic forgetting of previous classes when adapted to novel classes, their

TABLE 2.2: Overview of the approaches for concept drift detection and adaptation solutions for streaming images

Method		Detection Method	Detection Type	CNN Usage	CNN Adaptation
BN-Norm	[15]	Statistics	Implicit	Classify	SA, Partial
CDT-CNN	[44]	Change Detection	Explicit	Classify	Partial
CPE	[195]	Clustering	Implicit	Detect	Full
Echo-D	[74]	Clustering Ensemble	Implicit	-	-
ECSMiner	[123]	Ensemble	Implicit	-	-
EMC	[42]	Clustering	Implicit	-	-
RSB	[101]	Clustering	Implicit	Classify	Full
SACCOS	[61]	Clustering Graph	Implicit	-	-
SAND	[73]	Clustering Ensemble	Implicit	-	-
TENT	[192]	Statistics	Implicit	Classify	SA, Partial

SA: Specific Architecture

requirement of balanced classes and their need for large amounts of training data as compared to other machine learning models. Most solutions use clustering as the discrepancy detection method, and transform the high dimensional input data into a lower dimensional representation. Clustering requires a number of accumulated instances before drift is declared, which is an implicit method of detection and can be slower than explicit methods. Where explicit concept drift detection is used, it is an off-the-shelf method.

For the CNN adaptation, the few existing methods retrain some or all of the CNN layers, with the exception of methods in the related field of UDA where extra batch normalisation layers are inserted into the network. However, this changes the architecture of the network and requires that the CNN is trained from scratch, meaning that this method cannot be applied to a pretrained standard CNN. It is notable that none of these solutions employ the use of activations from within the CNN. In concept drift, it is also notable that concept drift is applied via synthetic augmentation of the images to apply effects such as blur, noise and illumination. We have identified that CNN adaptation has not been widely studied within combined discrepancy detection and CNN adaptation methods, thus in the following section, we investigate online CNN adaptation methods from other fields.

2.4 Online Convolutional Neural Network Adaptation

In this section, we review online CNN adaptation techniques, with OCL being the major contributing field, and streaming learning for DNNs being another relatively new contributing field. OCL solutions tend to be trained on large incremental batches, whereas streaming learning trains DNNs one instance at a time and can be evaluated at any point instead of only at the end of a batch. The term OCL is often used interchangeably with lifelong learning and incremental learning [122]. In the OCL field, the aim is to accumulate and preserve knowledge without forgetting any previous data [122], known as catastrophic forgetting, where the network cannot perform well on previously seen data after updating with new data [69]. The terminology in this field is such that streaming data is provided as tasks, where for instance, a task may include new classes (class incremental) or data non-stationarity (domain incremental) [122].

Methods in the OCL field to mitigate catastrophic forgetting are based on: Memory, Regularisation, Knowledge Distillation and Parameter Isolation [122]. Regularisation techniques help prevent catastrophic forgetting by making small changes to the network's parameters. This can be done by adding penalty terms to the loss function or by modifying the gradient of parameters during optimisation [30, 98, 116, 157, 206]. Memory-based techniques store a subset of samples from previous tasks. These samples can be used for replay while training on a new task or for regularisation purposes [30, 18, 1, 31, 98, 149, 116, 7, 156, 154, 89, 92, 157, 8, 206, 210]. However, these methods can be infeasible if storing raw samples is not possible due to privacy or storage concerns [122].

Knowledge Distillation is a technique for transferring knowledge between networks. It has been widely used in continual learning methods [208, 156, 89, 110, 210], where it is often considered as a type of regularisation. However, it can be difficult to balance the regularisation with the current learning when learning from a long stream of data [122]. Parameter-isolation techniques prevent interference between tasks by allocating different parameters to each task with no architecture change [1, 77, 154] or with a Dynamic Architecture (DA) change [108, 202, 161], adding specific

architecture and hence parameters for new tasks. Generative replay is an alternative to memory-based techniques. It involves training a deep generative model to generate pseudo-data that mimics past data for replay. However, it can take a long time to train these models, and they may not be suitable for more complex datasets. Therefore, they are out of scope for this thesis. Most methods are based on memory, although there has been a trend towards combining methods [122].

The OCL field has two sub-fields of: Online Class Incremental (OCI) which is analogous to concept evolution, where new classes are added; and Online Domain Incremental (ODI) which is analogous to concept drift, where changes in existing classes occur. Methods that excel in performance in the Online Class Incremental (OCI) setting are Incremental Classifier and Representation Learning (iCARL) [156], Learning without Forgetting (LwF) [110], Maximally Interfered Retrieval (MIR) [8] and Experience Replay (ER) [31] [122]. LwF is regularisation and knowledge distillation based and all others are memory-based. LwF uses outputs from past tasks as an effective regulariser and knowledge distillation [85] to preserve knowledge from past tasks and uses a teacher/student model. However, the teacher/student model means it has reliance on relatedness between the old and the new data and may not perform well if the distributions of the old and new data are different [110]. iCARL creates a training set by mixing all of the samples in the memory buffer and the current unseen samples, adjusts the loss function to address class imbalance between old and new classes and uses a nearest-class-mean (NCM) classifier instead of a Softmax classifier. iCARL has the best performance with a small memory buffer on small datasets. However it has slow inference time. ER simply trains the model with the unseen data batch and memory mini-batches, has an efficient training time and has outperformed other approaches [31]. MIR selects memory samples according to the loss increases given the estimated parameter update based on the incoming unseen window and updates the DNN on this sample plus the original unseen window. It performs well on large scale datasets with a large memory buffer [122].

There are also a number of "tricks" that can be applied that can enhance existing adaptation methods. These address class imbalance due to the strong bias towards the known classes. Examples of the strongest ones are Review Trick (RV) [25], which uses an additional tuning step with a small learning rate and a memory buffer and

NCM [127], which replaces the fully connected layer and the Softmax layer with an NCM Classifier. However, inference time increases with the growth of the memory size whereas for RV, the training time increases with the growth of the memory size but it is more memory-efficient than NCM [122].

The same methods that are used in the OCI field are employed in the ODI field [122]. The highest performing DNN adaptation methods in the ODI field are based on memory. The notable difference is that the knowledge distillation method of LwF [110] is not successful in the ODI setting. Methods that excel in performance are iCARL, MIR and ER [122]. However iCARL has slow inference time.

Streaming Learning for DNNs is a relatively young field with papers emerging from 2019 [76]. In streaming learning, each unique example is seen only once, the ordering of the stream is arbitrary and the DNN being trained can be evaluated at any point within the data stream, rather than the incremental batch learning scenario, where a model learns from a series of large collections of labeled samples [77]. In the streaming literature, attention is given to which parts of the network should be static and which parts should be plastic [45], sample shortage, and the time to train the DNN [94, 205]. Methods of streaming learning for DNNs are ExStream [76], DeepSLDA [77] and REMIND [78]. ExStream is a memory-based method using a compressed set of prototypes to reduce memory size. In DeepSLDA, the use of deep Streaming Linear Discriminant Analysis [144] is explored for training the output layer of a CNN incrementally, which may be preferable in limited memory/computational resources scenarios as it does not store previous examples. REMIND [78] stores compressed feature maps in memory.

The UDA field, as first mentioned in Section 2.3.2 contains a method called Unsupervised Continual Learning for Gradually Varying domain adaptation (UCL-GV) [183]. Here, UDA is brought closer to the continuous learning paradigm with clustering, episodic memory replay with buffer management and a contrastive loss is incorporated for better alignment of the buffer samples. Due to its connection with continuous learning, it is listed in Table 2.3.

Table 2.3 provides an overview of the online CNN adaptation techniques, how they mitigate catastrophic forgetting, which parts of the CNN is adapted ('Full' or 'Partial') and whether a specific initial architecture (SA) is required or whether the

architecture is dynamically updated (DA). It has been stated, and is logical to assume that methods that retrain the entire network or that dynamically update the DNN architecture can be slow to adapt [164]. Therefore, initially, it appears that partial updating with no architecture changes is preferable. However, this does not take into account parameters such as the number of retraining epochs or if the learning rate is reduced, which could speed up the optimisation process and therefore the adaptation process. Dynamically Expandable Networks (DEN) [202] as listed in Table 2.3, adds small networks to the existing structure to accommodate new classes. The speed of this method depends on how efficiently the partial section of the DNN is selected, created and trained. In CDT-CNN [44], the CNN adaptation is partial but the layers to be updated are dynamically selected which may be slow. If there is an existing system that has a standard DNN which takes a long time to train, it would not be preferable to retrain on a new specialised architecture, but rather be able to apply the adaptation to the existing DNN. Hence, it is difficult to judge whether an adaptation technique will be fast simply from the section of network that is updated and what architecture changes are employed. The optimum selection of a method also depends on the desired characteristics of the system i.e. if a pretrained network with the constraint of no retraining is preferable.

There are differences in the focus of the concept evolution/concept drift detection and adaptation fields and the OCL fields. Research in the OCL field focuses on accumulating and preserving knowledge without forgetting any previous data [122]. True-labelled samples of the classes are often used, which results in an artificially high performance. In a real-world scenario, even partially labelling a data stream using humans can be expensive [59] and impractical due to the need for domain experts and manual labelling. This is in contrast to the Concept Evolution and Concept Drift fields where focus is upon the changing data, taking into account only some previous data and using minimal true-labelled samples. The fields of concept evolution/concept drift and OCL require bringing together as they offer complimentary views. The following section provides a combined overview of these aforementioned streaming and online continuous learning methods in a unique taxonomy.

TABLE 2.3: Overview of the approaches for online CNN adaptation

Method		Catastrophic Forgetting Mitigation	CNN Adaptation
A-GEM	[30]	Memory, Regularisation	Full
CBO	[18]	Memory	Full
CCG	[1]	Memory, Parameter Isolation	SA, Partial
CN-DPM	[108]	Parameter Isolation (DA)	DA, Partial
DeepSLDA	[77]	Memory, Parameter Isolation	Partial
DEN	[202]	Parameter Isolation (DA)	DA, Partial
DMC	[208]	Knowledge Distillation	Full
ExStream	[76]	Memory	Partial
ER	[31]	Memory	Full
EWC	[98]	Memory, Regularisation	Full
GDumb	[149]	Memory	Full
GEM	[116]	Memory, Regularisation	Full
GSS	[7]	Memory	Full
iCARL	[156]	Memory, Knowledge Distillation	Full
iTAML	[154]	Memory, Parameter Isolation	SA, Partial
LUCIR	[89]	Memory, Knowledge Distillation	SA, Full
LwF	[110]	Regularisation, Knowledge Distillation	SA, Full
MEFA	[92]	Memory	SA, Full
MER	[157]	Memory , Regularisation	Partial
MIR	[8]	Memory	Full
REMIND	[78]	Memory	Partial
SI	[206]	Memory , Regularisation	Full
TreeCNN	[161]	Parameter Isolation (DA)	DA, Partial
UCL-GV	[183]	Memory	SA, Full
WA	[210]	Memory, Knowledge Distillation	SA, Full

SA: Specific Architecture, DA: Dynamic Architecture

2.5 Discrepancy Detection and Adaptation Taxonomy

This section provides an overview of discrepancy detection and adaptation for streaming images and online CNN adaptation techniques. In Figure 2.1, we provide a diagram of the reviewed study areas, their methods and how they relate to each other. In Figure 2.2, we categorise the solutions for discrepancy detection and adaptation of streaming images into their use of CNNs, which parts of the CNN they adapt and what discrepancy detection method they use. Alongside, we also categorise the online CNN adaptation methods into their adaptation and catastrophic forgetting (CF) mitigation method. Figure 2.3 summarises the categories for detection and adaptation and Table 2.4 shows an amalgamated overview.

Figure 2.1 shows the available solutions for concept evolution and concept drift for detection of and adaptation to streaming images. The methods in blue show the solutions that adapt a classifying CNN. The pink methods use a CNN in the detection of concept evolution or concept drift. The yellow methods show the solutions where no CNNs are used. Some of the methods can be applied to both concept evolution and concept drift, with some adopting different solutions within one framework (as indicated via the asterisk). This shows that in the streaming literature, there are limited solutions that adapt the classifying CNN. As shown in blue, in the OCL and streaming learning fields, there are many solutions that adapt CNNs. The Deep UDA field has methods overlapping concept drift and OCL and the streaming learning in CNNs overlap into the OCL field.

TABLE 2.4: Overview of the approaches for data discrepancy detection with CNN adaptation and online CNN adaptation solutions for streaming images

Algorithm		Detects	Detection Method	Detection Type	Catastrophic Forgetting Mitigation	CNN Adaptation
Drift Detection and Adapting Classification CNN						
BN-Norm	[15]	CD	Statistics	Implicit	Parameter Isolation	SA, Partial
CDT-CNN	[44]	CD	Change Detection	Explicit	Parameter Isolation	Partial
RSB	[101]	CD/CE	Clustering	Implicit	Memory	Full
TENT	[192]	CD	Statistics	Implicit	Parameter Isolation	SA, Partial
Online CNN Adaptation						
A-GEM	[30]	-	-	-	Memory, Regularisation	Full
CBO	[18]	-	-	-	Memory	Full
CCG	[1]	-	-	-	Memory, Parameter Isolation	SA, Partial
CN-DPM	[108]	-	-	-	Parameter Isolation (DA)	DA, Partial
DeepSLDA	[77]	-	-	-	Memory, Parameter Isolation	Partial
DEN	[202]	-	-	-	Parameter Isolation (DA)	DA, Partial
DMC	[208]	-	-	-	Knowledge Distillation	Full
ExStream	[76]	-	-	-	Memory	Partial
ER	[31]	-	-	-	Memory	Full
EWC	[98]	-	-	-	Memory, Regularisation	Full
GDumb	[149]	-	-	-	Memory	Full
GEM	[116]	-	-	-	Memory, Regularisation	Full
GSS	[7]	-	-	-	Memory	Full
iCARL	[156]	-	-	-	Memory, Knowledge Distillation	Full
iTAML	[154]	-	-	-	Memory, Parameter Isolation	SA, Partial
LUCIR	[89]	-	-	-	Memory, Knowledge Distillation	SA, Full
LwF	[110]	-	-	-	Regularisation, Knowledge Distillation	SA, Full
MEFA	[92]	-	-	-	Memory	SA, Full
MER	[157]	-	-	-	Memory , Regularisation	Partial
MIR	[8]	-	-	-	Memory	Full
REMIND	[78]	-	-	-	Memory	Partial
SI	[206]	-	-	-	Memory , Regularisation	Full
TreeCNN	[161]	-	-	-	Parameter Isolation (DA)	DA, Partial
UCL-GV	[183]	-	-	-	Memory	SA, Full
WA	[210]	-	-	-	Memory, Knowledge Distillation	SA, Full

SA: Specific Architecture, DA: Dynamic Architecture

Figure 2.2 shows a taxonomy for data discrepancy detection and adaptation solutions for streaming images and online CNN adaptation, categorising the type of

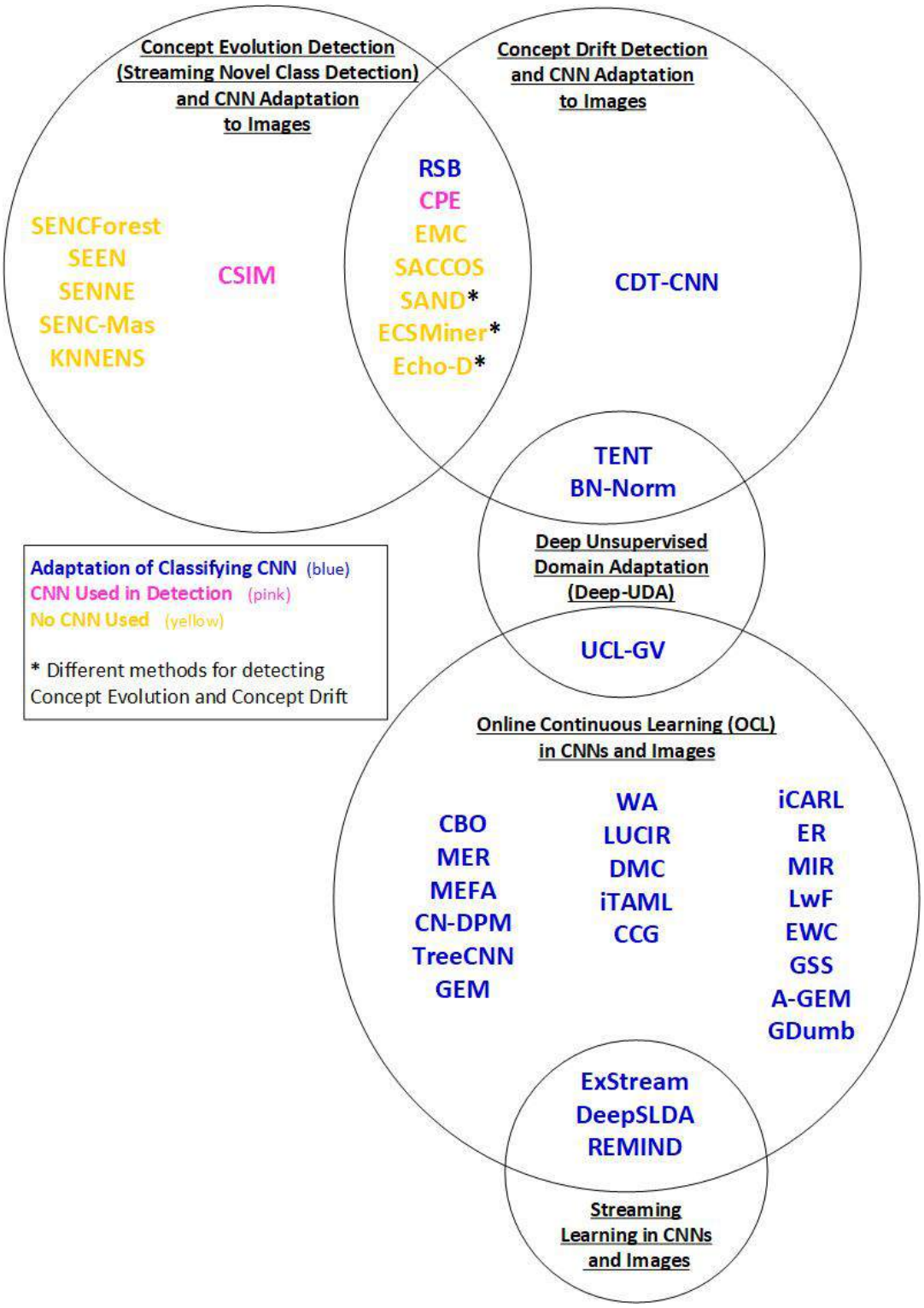


FIGURE 2.1: Discrepancy Detection and CNN Adaptation Methods

CNN usage; whether it is adapting the classifying CNN, whether it uses the CNN in detection, or if no CNN is involved. If a CNN is involved, the type of adaptation is categorised, as to whether it retrains the entire CNN, or only retrains part of the CNN, also displaying the catastrophic forgetting mitigation technique. Figure 2.3

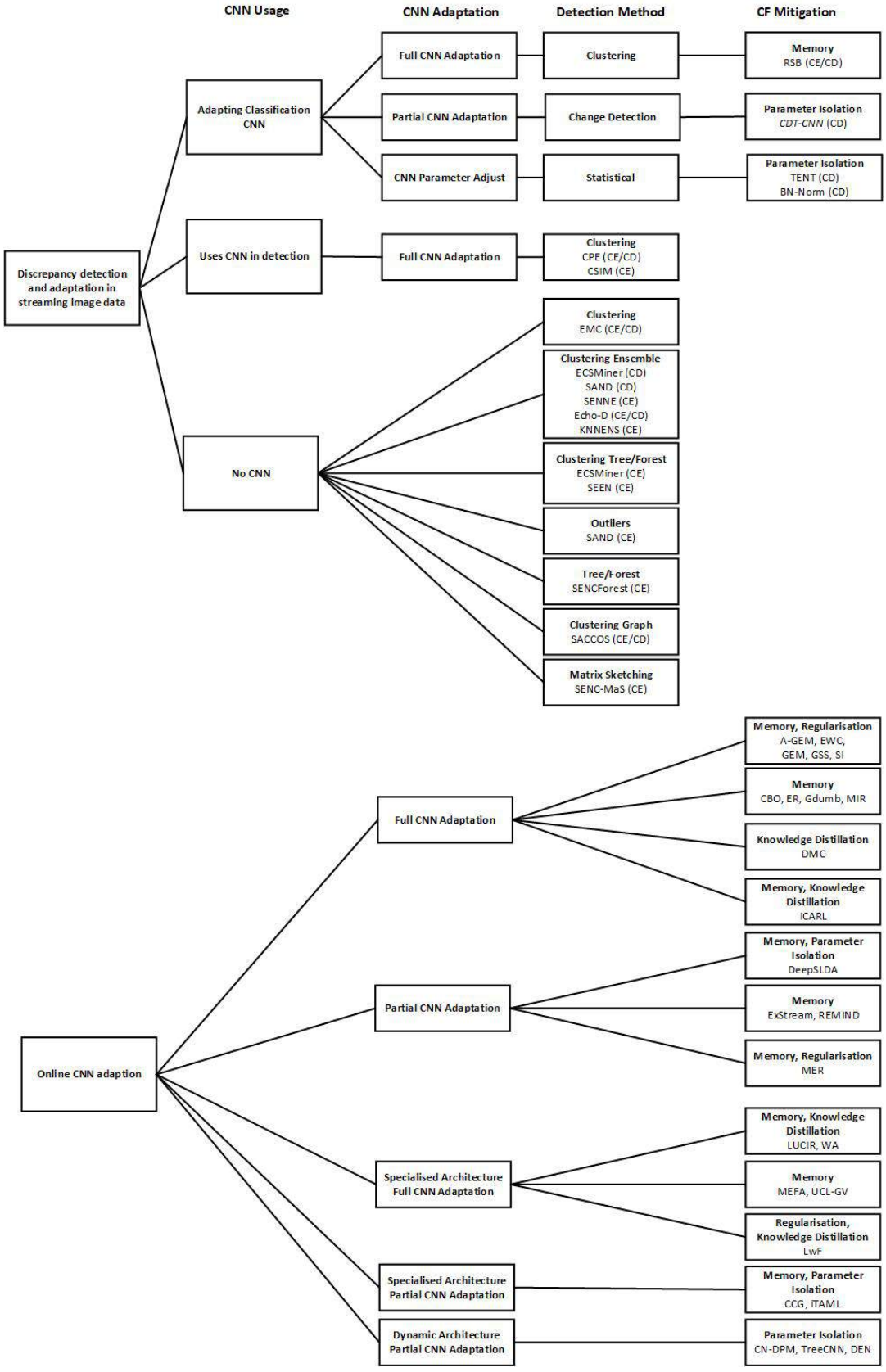


FIGURE 2.2: Proposed taxonomy for data discrepancy detection and adaptation solutions for streaming images

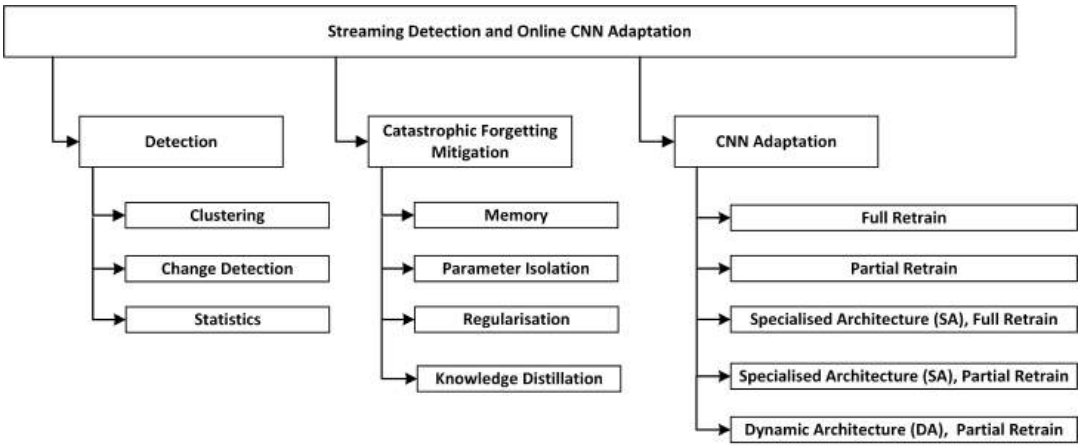


FIGURE 2.3: Proposed taxonomy for streaming detection and online CNN adaptation solutions

summarises all of the discrepancy detection and CNN adaptation techniques into their categories of detection, catastrophic forgetting mitigation and CNN adaptation. Table 2.4 summarises this information against the methods, also indicating the detection type of explicit or implicit. It can be seen that there are a limited number of detection and CNN adaptation techniques. Only one method uses explicit detection, with the majority using implicit techniques. It can also be seen that there are an abundance of CNN adaptation techniques using varying methods for CNN adaptation.

2.6 Discussion

Studying and grouping the existing solutions for streaming data discrepancy detection and adaptation for images reveals research gaps, providing a basis for introducing innovative approaches. Categorising the available solutions based on their discrepancy detection technique and their CNN involvement has assisted in identifying these gaps. For the combined data discrepancy detection and adaptation to streaming image data, the gaps are: (1) There are a lack of solutions for detecting data discrepancies in images and adapting a classifying CNN. This is because CNNs have not been a focus of study in the streaming field as they have a large adaptation latency and suffer from catastrophic forgetting more so than other machine learning models; and (2) there are limited solutions offering explicit data discrepancy detection. The existing solutions are tailored towards clustering and clustering ensembles.

We see no reason why explicit methods of discrepancy detection have not received as much attention, other than the lack of focus on discrepancy detection and CNN adaptation in the streaming scenario.

Focusing upon the CNN adaptation, there are CNN adaptation techniques available in the OCL field where data is applied in large incremental batches and the DNN adapts to these batches. The gaps are: (1) The streaming field has very limited solutions for adapting CNNs; and (2) the OCL field offers DNN adaptation techniques but they have not been applied to the concept evolution and concept drift fields to offer a more dynamic analysis. To the best of our knowledge, there is only one method that directly adapts the classifying DNN and attempts to bring together the OCL and concept drift fields, namely RSB [101]. However, this method uses implicit drift detection, it has a slow response to changing data, and analysis with respect to drift patterns was not performed.

Additionally, a further gap identified in the streaming data discrepancy detection and CNN adaptation field is that none of the detection methods use DNN activations. When reviewing the open-set recognition field, it was identified that activations from the final hidden layer are used in some solutions. Activations are also employed in other fields which is explored in Sections 3.4.1 and 3.4.2. The activations are higher dimensional than the already high dimensional input data of images. However, they have the potential to provide more information than the input data alone.

When an image is processed through a CNN, many feature maps are produced, which are smaller representations of the image, each containing features that have been extracted from the original image such as edges, textures or backgrounds. Section 3.3 details this process. This natural feature extraction process that takes place within the hidden layers of DNNs provides a much richer number of features than the original image, enabling more detailed analysis. For instance, analysing the image data will only be able to determine at a higher level if there is a discrepancy, such as identifying that it is open-set or an outlier, and then may only be able to distinguish the more extreme cases. However, analysing the feature maps (the values of which are called activations, collectively known as activation patterns, as described in detail in Section 3.3) provides an opportunity to extract alternative aspects of the

data than is possible via purely using the image data. Each image has its own way of activating the neural network and producing activation patterns. With further processing these patterns become usable in other machine learning models, for which we coin the term of activation classification footprint.

At the output of a DNN, classification is determined by a probability distribution across all classes. Two images can be of the same class and be correctly classified, but have different activation classification footprints. These images could represent different sub-types in a class. If the final hidden layer activations were monitored, this type of change in the data would not be captured. Monitoring the activation classification footprint has the potential to capture this.

Consider the scenario where we have a system that identifies human faces, the faces may be of differing sizes, which the model is trained upon and the model generalises well. However, at inference time, an image may arise that has a very big or very small face as compared to any of the training images, still this can be recognised by the model and is classified as a face. The activation classification footprint gives the opportunity to detect if anything has activated in this way before and thus detect a change in data.

We propose to create activation classification footprints from the naturally arising activation patterns from within the hidden layers of DNNs. These activation classification footprints have the potential to detect more discrepancies than is possible via the input data. This makes DNN activations a promising area to be applied to streaming data discrepancy detection and CNN adaptation.

Some CNN adaptation methods use specialised architecture or loss functions, requiring a classification system to specifically use these architectures from the outset as the models need to be trained on these architectures. However, in real-world applications, DNNs would be trained on large amounts of data and it may be infeasible to retrain new models with this data. Thus, it would be preferable to provide a method that can be applied to existing pretrained DNNs without the need to retrain, thus transforming a standard DNN into a discrepancy detecting adaptable DNN.

For the majority of concept drift detection methods, concept drift is applied via synthesised augmentation of the images such as artificially created new background, blur, noise, illumination and occlusion. However, in real-world systems, items in the

images are likely to change over time in a different way, such as photographed wild birds with different backgrounds [129] or a cat monitoring system, where there is a change in location of the monitored items such as the litter box or the cat changes from a kitten into a cat [35]. Therefore changes in classes concerning real images rather than synthesised ones demonstrate more real-world usability.

To address these gaps, this thesis proposes for the first time, explicit data discrepancy detection and DNN adaptation for streaming data using activation classification footprints. This thesis firstly proposes *DeepStreamOS* to address outlier data discrepancy detection using activation classification footprints. Next, *AdaDeepStream* is proposed to address concept evolution detection and DNN adaptation using activation classification footprints. Finally, *DeepStreamEnsemble* is proposed to address concept drift detection and DNN adaptation using activation classification footprints. *AdaDeepStream* and *DeepStreamEnsemble* also bring together the fields of concept evolution/concept drift with OCL by analysing DNN adaptation techniques in a dynamic streaming environment by applying drift patterns. Application of concept drift is via the introduction of previously unseen real-world images of new sub-classes, thus using real-world images as opposed to synthetically augmented ones.

2.7 Summary

This chapter offered an overview of different approaches for data discrepancy detection, starting with open-set recognition and progressing onto concept evolution and concept drift detection with CNN adaptation, focusing on CNNs and images. Due to a lack of solutions for CNN adaptation in the streaming field, the fields of OCL and streaming learning for DNNs were investigated with respect to CNN adaptation techniques. A novel taxonomy of the approaches for discrepancy detection and CNN adaptation in streaming images has been presented to simplify the understanding of the current literature. The contributions of this thesis with respect to the literature were discussed, demonstrating the originality of the proposed methods. The following chapter provides background information in order to understand the proposed solutions.

Chapter 3

Background for Proposed Solutions

3.1 Introduction

This chapter provides necessary background information in order to understand the proposed solutions, accompanied by the formalisation and theoretical explanation of the algorithms employed in these solutions. As this thesis is concerned with high dimensionality data, Section 3.2 provides an overview of the problems associated with this. Subsequently, in Section 3.3, CNNs are explained as they are used in our experimental study, with a focus on the activations that are output from the internal layers of the CNN. In Section 3.4, the use of DNN activations and methods of reducing them is concisely reviewed. This is followed by the basis of our activation reduction methods which are JS-Divergence and a DNN-based CBIR descriptor method.

To detect our discrepancies, we employ streaming machine learning models. Section 3.5 provides an overview of these methods. For our outlier detection solution as described in Chapter 4, we use an existing streaming clustering method of Micro-cluster Based Continuous outlier Detection (MCOD), for which an overview is given. For concept evolution and concept drift detection, we introduce the streaming classifiers that are employed in the solutions in Chapters 5 and 6 respectively. We provide background information for decision trees, leading onto Hoeffding trees. We introduce the Self Adjusting Memory k-Nearest Neighbours (SAM-kNN) streaming classifier as used in Chapter 5.

To provide an increased understanding of concept evolution and concept drift, Section 3.6 gives a comprehensive overview of what they are and the difference between them, followed by a concise review on existing methods of detection. Finally, an overview of the Drift Detection Method (DDM) [58] is given in Section 3.6.2 as this features in our solution in Chapter 5.

3.2 The Curse of Dimensionality

Our experiments focus on the use of images, which are high dimensional. However, it is not the image data that will be used in our proposed solutions, but the activations from the CNN. These activations are also high dimensional therefore, we provide an overview of the problems associated with high dimensional data in this section. Bellman introduced the curse of dimensionality [12]. It occurs in multiple fields, machine learning being one of them. It encompasses the challenges that arise when the number of features in a dataset increases. The curse of dimensionality can make data analysis more complex and lead to the issues of increased computational complexity, data sparsity, increased data volume, overfitting and difficulty in visualisation as data points become approximately equidistant from each other [12]. As the number of dimensions increase, algorithms that excel in low-dimensional spaces may become slow and impractical. Data sparsity means that there may be large regions of empty space between data points, such that it is harder to accurately model the relationships between data points, leading to poor accuracy predictions. Increased data volume may mean that a much larger amount of data is required to cover the feature space, increasing storage requirements. The risk of overfitting is increased, where a model fits well to specific characteristics involving i.e. random variations to the data, which is not representational of the underlying patterns and does not generalise well to unseen data. The increase in dimensionality in distance-based algorithms such as clustering is particularly a problem as the distance becomes less meaningful, as all data points tend to be roughly equidistant from each other. It is increasingly difficult to visualise and interpret data when the dimensions increase. Most people can only visualise data up to three dimensions. Visualising data in many dimensions can be challenging, making it harder to gain insights from

the data.

To mitigate the curse of dimensionality, dimensionality reduction techniques are often used, such as Principal Component Analysis (PCA) [197], Uniform Manifold (UMAP) [125] or t-distributed Stochastic Neighbor Embedding (t-SNE) [189], which reduces the number of dimensions while preserving the most important information. These techniques can help simplify the data and improve the performance of machine learning models. Additionally, feature selection and feature engineering can be used to reduce the dimensionality of the dataset by selecting the most relevant features or creating new meaningful features. The following section provides background information about the activations from a CNN.

3.3 Convolutional Neural Networks

As CNNs are used in our experimental study, this section provides an overview of how they operate, with a focus on explaining the activations from within them. In computer vision, DNNs (specifically CNNs) have excelled in image classification [104, 79, 173, 90]. Figure 3.1 shows the architecture of a commonly used CNN called VGG16 [173]. Simonyan and Zisserman introduced VGG16 in 2015, and despite its age, it remains extensively utilised today, very recently in domains such as plant identification, medical diagnosis, visual speech recognition, and manufacturing processes [201, 135, 163, 167]. The adaptive methods in this thesis aim to provide a wrapper to existing systems such that the standard CNN becomes adaptive, these systems may not be using the latest designs of networks, thus VGG16 is a suitable choice as it is widely used. While this thesis employs VGG16, it also employs MobileNet [90] but is not restricted to these models alone. VGG16 consists of convolutional blocks, followed by classification layers. For the remainder of this thesis, convolutional block will be simply referred to as block. Each block consists of a few convolutional layers (typically one to three) and a max pooling layer. The convolutional layers apply a set of filters to the input data which extract features such as edges from the original image. The max pooling layer simply reduces the size. The classification layers are fully connected layers, where each output dimension depends on each input dimension and each neuron is connected to each neuron

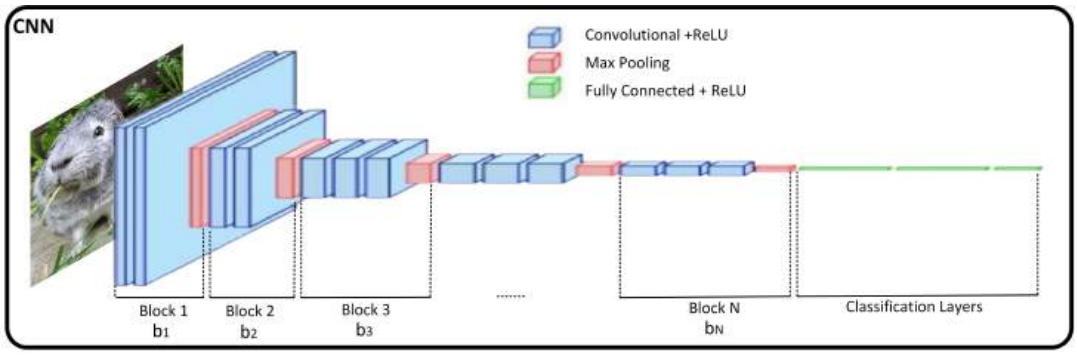


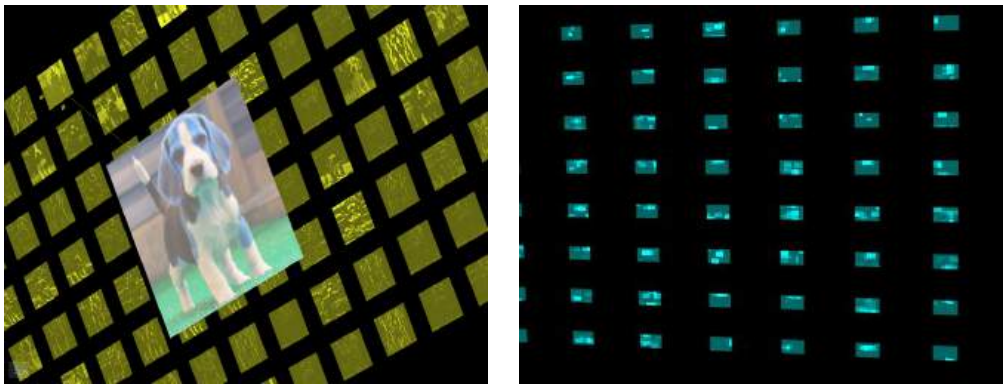
FIGURE 3.1: VGG16 CNN Architecture

in the following layer. A simple diagram of convolutional layers followed by a fully connected layer, leading to three output classifications is shown in Figure 3.3. The fully connected layers transform the final convolved outputs into a final number that corresponds to a classification label.

Training a Deep Neural Network (DNN) involves several key steps. The process begins with the forward propagation of inputs through the network layers to generate an output. This output is then passed through a Softmax layer, which converts the raw output into a probability distribution over the predicted classes, ensuring that the sum of all probabilities equals 1. The difference between the predicted output and the actual label is calculated using a loss function, which quantifies the error of the prediction. Backpropagation, guided by the chain rule of calculus, is then used to compute the gradients of the loss function with respect to the weights and biases in the network, effectively determining how much each parameter contributed to the error. The learning rate, a hyperparameter, determines the step size during the optimisation process when updating the network's parameters. This entire process is repeated for a specified number of iterations or epochs. An optimisation algorithm, such as Stochastic Gradient Descent (SGD) [158] or Adam [97], is used to adjust the parameters (weights and biases) in the direction that minimises the loss function, thereby improving the model's predictions over time.

Figure 3.2 shows examples of feature maps from TensorSpace, a neural network 3D visualisation framework [184]. The example images are taken from the AlexNet [103] CNN, which is similar to VGG16. Figure 3.2 (a) shows the initial image of a dog. Behind the dog, in yellow is shown the output of each filter from the first block. These are known as feature maps or channels. In this thesis, we shall refer to them as

feature maps. Each feature map is a representation of the original image, each with specific characteristics that have been picked out by each filter. For instance, some of the feature maps show vertical edges and others show horizontal edges. Each pixel from each feature map is an activation value. In the first block, the feature maps preserve nearly all of the information found in the original image. Deeper into the layers, these feature maps take on a more abstract nature and become less visually interpretable. They start encoding more advanced concepts like angles and corners. Figure 3.2 (b) shows some of the feature maps from the final block. These carry increasingly less information about the visual contents of the image, and increasingly more information related to the class of the image. These feature maps from within the hidden layers of the CNN demonstrate the natural feature extraction capability that image classifying CNNs have.



(a) First Convolutional Block

(b) Final Convolutional Block

FIGURE 3.2: Examples of feature maps from block outputs

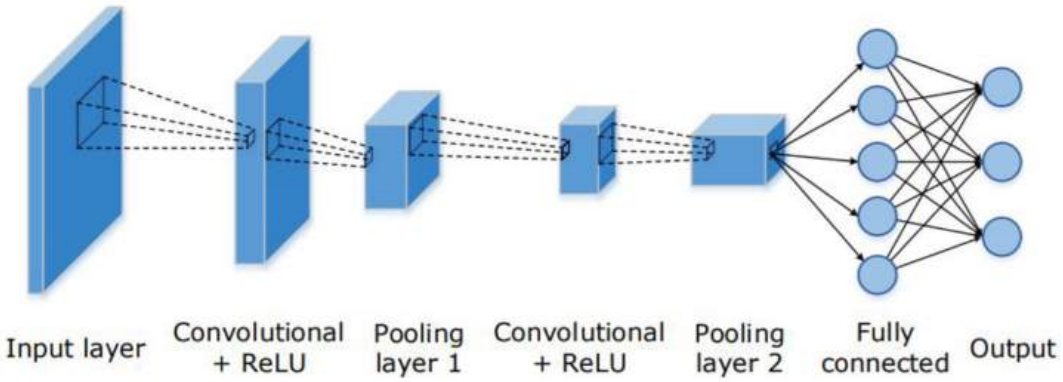


FIGURE 3.3: Simple Representation of Convolutional and Fully Connected Layers

Convolution of an image is depicted in Figure 3.4. The input image is multiplied by a filter (also known as a kernel or a feature detector. For this thesis, we shall refer to it as a filter). The image is multiplied by a filter being placed over sections of the image and is moved by a pre-defined 'stride' In the example, the applied filter is shown in yellow with a thick border and the values it is multiplied by (the values of the filter) are shown in red in the bottom right hand corner of each element. The stride of the filter is 1, as in the second iteration, the filter has moved one place to the right.

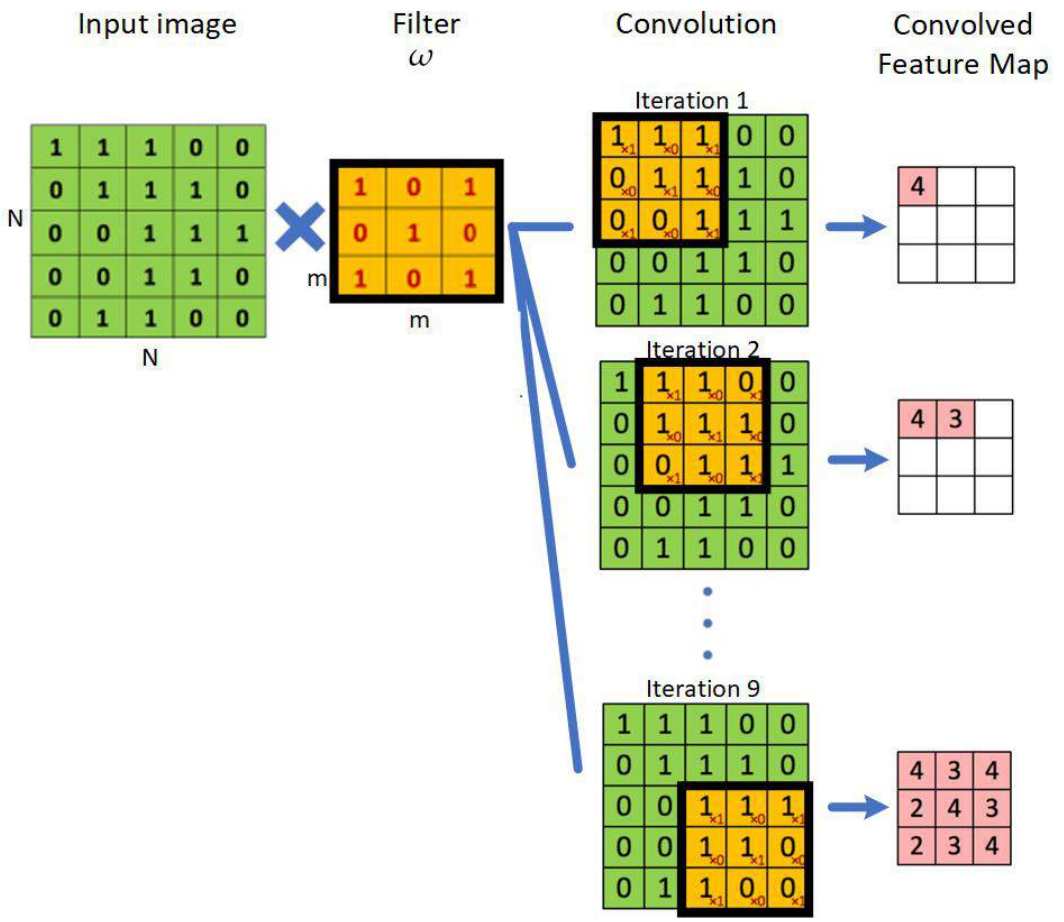


FIGURE 3.4: Graphical Representation of Convolution

We have an input (neuron layer) of $N \times N$ (this is the input image in the first convolutional layer) and we have an $m \times m$ filter ω. In order to compute the output of the convolution, we multiply the output from the previous layer with the filter elementwise in strides across, then down the input matrix. At each stride, the area the filter is located on the input matrix is multiplied by the filter and these multiplied values are summed. This reduces the size of the output matrix unless the padding is

set to 'valid', which adds a user-specified number of padding elements around the outside of the matrix such that the output of the convolution remains the same size as the input (most common) or is larger. The example we show in Figure 3.4 is with padding set to 'same' which means no padding elements are added, hence why the convolved feature map is smaller than the input matrix. Lastly, the activation function σ is applied. In VGG16, this is achieved via a ReLU layer as shown in Figure 1.3 (c), where if the input > 0, the input value remains the same, otherwise the output is zero. Equation 3.1 represents the output of a convolutional layer with the activation function, applied:

$$y_{ij}^{\ell} = \sigma(\sum_{a=0}^{m-1}\sum_{b=0}^{m-1} \omega_{ab} y_{(i+a)(j+b)}^{\ell-1}) \tag{3.1}$$

Following the convolutional layer is commonly a max-pooling layer, which simply takes a pre-defined region of size $k \times k$ and outputs a single value, which is the maximum value in that region, also known as sub-sampling. Table 3.1 shows the output dimensions for each layer in the VGG16 CNN with 1000 classification outputs and the number of activations pertaining to each layer. This gives an indication that there are a huge amount of activations per layer. The activation values output from these convolutional, max pooling and fully connected layers are the activations that are used in varying ways in this thesis.

Given the utilisation of activations in our methods and the substantial volume of values generated per layer, it is advantageous to test these methods on deep learning architectures with a reduced number of layers to maintain the tractability of our experimental work. While more recent architectures, such as MobileNet, feature a greater number of hidden layers and consequently produce a higher number of output features, their execution demands more time and memory. Thus, VGG16 provides a suitable architecture for demonstrating our methods in an afforded amount of time and computing resources.

During this thesis, the CNN is adapted via the following cross-entropy cost function:

$$-\frac{1}{M}\sum_{x=1}^{M}\sum_{c=1}^{L} {y_c}^{(x)} log {p_c}^{(x)} \tag{3.2}$$

Where M is the number of instances in the training set, x is the training set instance,

TABLE 3.1: Calculation of the number of activations for VGG16 CNN

Layer	Dimensions	Number of Activations
Input	$224 \times 224 \times 3$	
Block 1 Conv 1	$224 \times 224 \times 64$	3211,264
Block 1 Conv 2	$224 \times 224 \times 64$	3211,264
Block 1 Max Pool	$112 \times 112 \times 128$	1605,632
Block 2 Conv 1	$112 \times 112 \times 128$	1605,632
Block 2 Conv 2	$112 \times 112 \times 128$	1605,632
Block 2 Max Pool	$56 \times 56 \times 256$	802,816
Block 3 Conv 1	$56 \times 56 \times 256$	802,816
Block 3 Conv 2	$56 \times 56 \times 256$	802,816
Block 3 Conv 3	$56 \times 56 \times 256$	802,816
Block 3 Max Pool	$28 \times 28 \times 512$	401,408
Block 4 Conv 1	$28 \times 28 \times 512$	401,408
Block 4 Conv 2	$28 \times 28 \times 512$	401,408
Block 4 Conv 3	$28 \times 28 \times 512$	401,408
Block 4 Max Pool	$14 \times 14 \times 512$	100,352
Block 5 Conv 1	$14 \times 14 \times 512$	100,352
Block 5 Conv 2	$14 \times 14 \times 512$	100,352
Block 5 Conv 3	$14 \times 14 \times 512$	100,352
Block 5 Max Pool	$7 \times 7 \times 512$	25,088
Fully Connected	4096	4096
Fully Connected	4096	4096
Fully Connected	1000	1000
Output	1000	
	Total:	16,482,816

L is the number of classes, c is the class, y is a one-hot encoded vector of ones and zeros (one for each class), y_c is the y value of the class and p_c is the model's prediction for that class (the output of the Softmax layer for class c).

Now that we have an understanding of activations within a CNN, the following sections provide a review of how these activations have been extracted and used to gain useful information in discrepancy detection fields, the DNN inspection field and possible methods of reducing them, leading onto our choices for activation extraction and reduction.

3.4 DNN Activations

The proposed solutions in Chapters 4, 5 and 6 all use DNN activations. For one input instance into a DNN, a huge number of activations are produced at each hidden layer within the deep neural network. Section 3.3 explains and exemplifies this. Activation data requires reducing before it can be meaningfully used in subsequent analysis stages. Activations are used in various discrepancy detection fields and are also commonly used in the DNN Inspection field. Activations have been widely used in the field of Visual Interpretation of deep neural networks to determine which neurons are related to specific image features to explain how the neural network is

arriving at its classification. How the network arrives at its classification is out of the scope of this thesis as we are only interested in identifying the important neurons in a data instance's classification. The following sub-sections investigate the use of activations in various discrepancy detection fields and in visual interpretation and also how this activation data can be reduced.

3.4.1 Activation Usage in Discrepancy Detection

In discrepancy detection in DNNs, there are a number of different fields pertaining to the detection of different types of data discrepancies. We have already reviewed one of them - open-set. The others are out-of-distribution (OOD), anomaly detection and adversarial detection. We focus on the use of DNN activations in these fields. OOD refers to instances that lie outside of the data distribution that the model was trained on and aims to identify instances that are substantially different from the training distribution. Anomaly detection, is the detection of instances that deviate significantly from the majority of the data. Adversarial detection involves identifying instances that have been specifically created to mislead and exploit vulnerabilities in a model's decision-making process [68]. The following paragraphs review where DNN activations have been exploited in the aforementioned fields.

The main detection area that activations are used in is OOD and only recently, since 2019, has this been explored. In OOD detection, areas involving DNN activations are: Deriving OOD scores based on analysis of activations; and uncertainty estimation methods that operate on internal activations. These analysis scores and estimates are used to monitor DNNs in safety-critical domains such as self-driving cars [34, 75, 83] and cybersecurity malware classification [37], or explaining patient medical diagnosis. In [147] a simple statistic analysis of the penultimate layer activations is employed to detect rare sub-classes in the data as they may reduce the DNN performance. In [34, 75, 83], the activation outputs of the hidden layers are monitored for DNN safety signalling.

In [83], Henzinger uses k-means clustering and abstraction boxes. In [75], Hashemi builds a Gaussian model for every neuron independently, establishing safety intervals for every neuron based on the mean and the standard deviation of the previously built models and a voting mechanism to improve the decisions correctness.

In [34], Cheng uses binary activations patterns and the hamming distance for OOD detection. However, this has significant efficiency issues for use in current DNN architectures and is improved upon in [138]. The type of DNN activation function is important for OOD uncertainty calibration and an alternative bespoke activation function is suggested in [128]. There are benefits to using the outputs of various hidden layers and not just the final layer, as these authors conclude [148, 141]. Olber proposes the use of binary neuron activation patterns using convolutional layers [138]. It is however, specific to networks using ReLU activation functions. ReAct [178], reduces model overconfidence on OOD data by modifying the activations of the final hidden layer to bring the overall activation pattern closer to the known activation patterns and thus reduce the overconfidence score. ASH [46] removes about 90% of a sample's activation at a late layer and the remaining 10% is lightly adjusted and used. The shaping is applied at inference time, and does not require any statistics calculated from training data.

The use of activations within the anomaly detection field is less prevalent, with [175] showing that the hidden activation values contain information useful to distinguish between normal and anomalous samples by combining three DNNs in an anomaly network and based on the activation values in the target network, the alarm network decides if the given sample is normal.

Adversarial detection involves identifying instances that have been specifically created to mislead and exploit vulnerabilities in a model's decision-making process [68]. Qui [150] is using activation based back propagation algorithm to detect an image's effective path. This author notes that it needs to be kept in mind that different methods or more computationally expensive methods may be required for adversarial detection. Activations and the link between the activation layers are used to map a path of neurons inside the deep neural network. There are adversarial images that are close to the feature space of the original training instances, with only a few pixels alteration in some cases [177]. Here, a layer in the deep network is used where the activations will be far away from the training samples, where unknown images become outliers in an open set recognition problem [13]. Carrara [24] shows that hidden layer activations can be used to reveal incorrect classifications

caused by adversarial attacks. In SafetyNet [120], Lu states that "Adversarial attacks work by producing different patterns of activation in late stage ReLUs to those produced by natural examples". Therefore, SafetyNet quantises the final ReLU activation layer of the model and constructs a binary support vector machine with a radial basis function kernel as a classifier. Carrara [23] proposed a hypothesis that intermediate representations of adversarial examples undergo a distinct evolution with respect to clean inputs. The relative positions of internal activations of points representing the dense regions of the feature space is encoded. The detector is a binary classifier constructed on top of the pretrained network, taking as inputs, the encoded relative positions of internal activations of points representing the dense regions of the feature space for adversarial examples and clean inputs. It should be noted that adversarial detection is harder to detect than open-set as it often involves small perturbations to the input data. It is out of the scope of this thesis but the review on activations usage has shown that activations are successfully used within this sub-field and point to successful usage in other discrepancy detection fields.

Table 3.2 shows a summary of activation usage in discrepancy detection methods. 'Final' denotes that only the activations from the final hidden layer are used. 'Multi' denotes that activations from multiple hidden layers are employed in the solution. From this, we can see that there is a mixture of the amount of layers that are used.

Focusing upon the activations as used in other discrepancy detection paradigms, we studied the field of activation usage in data discrepancy detection. In open-set detection (where an extra classification option of 'unknown' is added to the output of a DNN) in Section 2.2, we discovered that typically, only the final layer of of DNN activations are used. In OOD, anomaly detection and adversarial detection, we find more examples of use, ranging from just the final layer to intermediate and all layers, indicating that other layers also carry important information about the images. Activation data is also used in the DNN Inspection field for the purpose of identifying which neurons are responsible for which parts of image classification. This is reviewed in Section 3.4.2.

TABLE 3.2: Overview of the approaches for using activations in discrepancy detection

Layer	Description	Method
Final	Statistic analysis of penultimate layer activations to detect rare sub-classes	[68]
Final	Modifies the activations of the final hidden layer to bring the overall activation pattern closer to the known activation patterns to reduce the overconfidence score	[147]
Final	90% of a sample's activation at a late layer is removed, the remaining 10% is lightly adjusted	[138]
Final	Quantises the last ReLU activation layer of the model and builds a binary support vector machine with a radial basis function kernel classifier	[46]
Multi	Binary activations patterns and the hamming distance for OOD detection	[203]
Multi	A sampling-free approach to approximate uncertainty estimates that rely on noise injection at training time. Further simplification specifically for convolutional neural networks using ReLU activation functions.	[83]
Multi	Comparison study that concludes that there are benefits to using the outputs of various hidden layers	[37]
Multi	Based on inferring Gaussian models of neuron activation values of some of the neurons and layers	[14]
Multi	Binary activation patterns and hamming distance	[115, 34]
Multi	Bespoke Activation Function and activations	[75]
Multi	Comparison of activations between networks	[128]
Multi	Backpropagation using activations to extract the image's effective path	[148]

3.4.2 DNN Inspection

In the DNN inspection field, there have been many approaches to the identification of the most important activations and these have been recently surveyed showing that this is an important area and forms part of explainable AI [4], [21]. In [21], Table 1 shows that neuron activations are used for explainers for DNNs in different fields such as images and text based DNNs. Approaches to using neuron activations can be summarised as: Top k percent of activations in each layer, the activation magnitude, average activations and clustering, nearest neighbour and backpropagation.

The first approach of top k percent activations is used in the Summit paper [88], which uses the activation of channels in a CNN (to determine i.e edges, shapes, texture) and applies global max pooling to reduce the data. It uses the activations of the channels and is appropriate to CNNs only. This method only does a forward pass through the network to obtain the activations therefore, it is low on computation which is required in a streaming environment. Activation Magnitude and Matrix Factorisation is utilised by Olah [137] and uses the magnitude of the neuron activations and represents them as a cube and breaks them up using matrix factorisation to get more meaningful groups of neurons, however matrix factorisation is computationally expensive as it has to be explicitly calculated for each image, and is therefore

not suitable for a streaming environment. Average activation and clustering is used in [113], where the average activation of each neuron in the activation layer is used (average is taken for all instances with the same class), then clustered and a number of neurons from each cluster is selected. This is more suited to a static environment as a number of instances from the same class are required, which would not be available in a streaming scenario with an emerging novel class.

In ActiVis [93], the average activation for each neuron for all instances in a class are used, but presented to the user for visualisation. The nearest neighbour approach is used in [145] where nearest neighbour is applied to the activation outputs of each hidden layer. Locality Sensitive Hashing (LSH) function is used to reduce the data dimensionality so it is suitable for use in the nearest neighbour representation, but this is computationally expensive and unsuitable for a streaming environment. Backpropagation is used in [166, 150]. The latter is applicable to both CNNs and fully connected networks. This describes an effective critical path of weights and neurons that lead to the final predicted path and uses an activation-based back propagation algorithm to extract the effective path. This requires a backward pass through the network which is computationally expensive and not appropriate in a streaming environment.

In summary, usage of neuron activations requires limitation of which neurons are used (i.e. only use a particular layer(s) or channel(s), or general reduction of the activation such as in LSH to be suitable for post processing. Papers [145, 88, 33] use the last activation layer only as a method of data reduction as this is the most representative of the classification. Given this, a summary of general data reduction techniques is required.

3.4.3 Activation Reduction

Using DNN activations implies data reduction of some form. This could be just selecting a fewer number of layers from the network, or filtering the number of activations selected from each layer, using a statistical calculation between activations, or using a general data reduction technique or a mixture of some or all. For instance, a method could be using a distance measure on each layer such as L2Norm to get a single value to represent the activations in each layer.

Popular general methods of data reduction are Independent Component Analysis (ICA), Principle Component Analysis (PCA), autoencoders, restricted Boltzmann machines and graph-embedding. Hinton describes using selectively initialised autoencoders as better than PCA [84]. PCA and ICA are used for linear transformations but when we are concerned with deep neural networks, they are not linear transformations. PCA is restricted to linear dimensionality reduction whereas autoencoders enable linear and non-linear transformations. The choice of the autoencoder depends on the nature of the data, for instance, convolution networks are preferred for image datasets and Long Short Term Models (LSTM) produce good results for sequential data.

In summary, analysis of activations within discrepancy detection fields show that it is feasible and advantageous to use activations from within multiple hidden layers of DNNs. In the DNN inspection field, activations are typically used in conjunction with machine learning methods such as clustering or kNN directly on layers or feature maps of the DNN to extract them. However, these are computationally expensive and the speed and computation aspect has not been a focus of this field. Some methods of data reduction can also be computationally expensive such as LSH or autoencoders which also require parameter tuning, or the reduction methods are only suitable for linear transformations such as PCA.

Another avenue is statistical measures that can differentiate between probability distributions. Popular measures are Kullback-Leibler (KL) Divergence [55], Jensen-Shannon (JS) divergence) [80, 171] and Kolmogorov-Smirnov (KS) Test [194]. The Jensen-Shannon divergence is based on the KL divergence and measures the similarity between two probability distributions and has previously been used with DNNs. It has been used with DNN neurons and random forests to calculate the importance of a neuron [80] and applied within kernels of Support Vector Machines [171] where Sharma expressed concern over whether JS-Divergence provides adequate separation when the difference between input distributions is subtle. The KS-Test is used to determine whether two sets of data came from the same distribution. Both the JS-Divergence and the KS-Divergence are used to compare distributions, the JS-Divergence measures the difference between two probability distributions, whereas

the KS-Test determines whether two samples could come from the same distribution. Empirically, in our setting, JS-Divergence was proven to be more successful than the KS-Test. Therefore, the JS-Divergence is focused upon with Section 3.4.4 formalising this.

Given that we are focusing on computationally inexpensive reduction methods and that our experimental methodology is based on images, then another avenue of investigation opens up, which is that of image descriptors. Image descriptors are a compact representation of the visual features of the content of images and can be used to distinguish one image from another by encoding useful information [70]. There has been a recent advance in descriptor generation by analysing activations of DNNs [176]. Section 3.4.5 describes this in detail and is the basis of an activation reduction method used in Chapter 6.

3.4.4 Jensen Shannon Divergence

JS-Divergence is a smooth symmetric measure of the similarity between two probability distributions based on the Kullback-Leibler divergence (KL-Divergence). KL-Divergence D_{KL} is based on entropy (a quantification of how much information is in data). H is the entropy and if log_2 is used, entropy is the minimum number of bits it would take to encode the information. For instance, if p is the probability distribution of the activations in hidden layer 1, we get an entropy measure of H bits from hidden layer 1 via Equation 3.3. The KL-Divergence combines two probability distributions and calculates the difference of the log values for each. For instance, if q is our distribution of the activations from hidden layer 2, KL-Divergence calculates how much information is lost when p is compared with q. Equation 3.4 formalises this. However, KL divergence cannot be used to measure the distance between two distributions as it is not symmetric. The JS-Divergence D_{JS} calculates a normalised score that is symmetrical as shown in Equation 3.5.

$$[!h]H = -\sum_{i=1}^{N} p(x_i) \cdot \log p(x_i) \tag{3.3}$$

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot (\log p(x_i) - \log q(x_i)) \tag{3.4}$$

$$D_{JS}(p||q) = \frac{1}{2}D_{KL}(p||\frac{p+q}{2} + \frac{1}{2}D_{KL}(q||\frac{p+q}{2}) \tag{3.5}$$

The application of the JS-Divergence measurement to the hidden layers of a CNN is discussed in the following paragraphs.

The JS-Divergence-based measurement is applied to a single instance presented to the CNN. The activations of this instance undergo a transformation: they are normalised, and if the layers under comparison differ in size, the smaller layer is padded with zeros. This normalisation ensures the activations range between 0 and 1 and are of equivalent size, making them suitable for the JS-Divergence calculation. Individually, the hidden layer activations for a single instance do not represent a probability distribution. However, after transformation, they become amenable to distributive measures, allowing us to use these activations to derive a value that distinguishes the activation data between any two layers under consideration. JS-Divergence is a symmetrical similarity measurement and these properties can lend themselves well to the activations after the above transformation process.

In this approach, we are not comparing neurons between each layer on a one-to-one basis, as would be the case with distance-based measures such as Euclidean or Manhattan distances. Instead, we employ a distribution-based calculation which does not assume this relationship of the neurons. This calculation is performed consistently for each instance, enabling direct comparison across instances. As the distribution of the input data changes, so too does the distribution of activations within the hidden layers of the CNN, impacting our inter-layer measurements. Further details on the distributions of activations in the hidden layers are provided in the following paragraph.

As an example, the VGG16 CNN uses ReLU activation functions, whose outputs are greater than zero as shown in Figure 1.3 (c). However, biases can be applied to neurons, which may make the output of the activation function negative. Thus, the activations of each layer output are normalised to be between 0 and 1. This also makes the method applicable to other networks that use different activation functions that may produce negative output values.

The deep neural network's latent representations from within the hidden layers of the deep neural network are used in detecting out of distribution samples [145,

107]. The output of the final hidden layer of a DNN into the SoftMax output layer is a probability distribution as DNNs learn the distribution of the training data and the output layer uses this probability distribution to generate the output class. The internal latent representations are also created from the distributions that are learnt at training time. Thus, the activations of the hidden layers are not random data but derived from the training of the deep neural network. In [140], Ong investigates the distributions of the activations of the hidden layers. Figure 3.5 shows an overview of the distributions of activations at the last convolutional layer for one class (image from [140]).

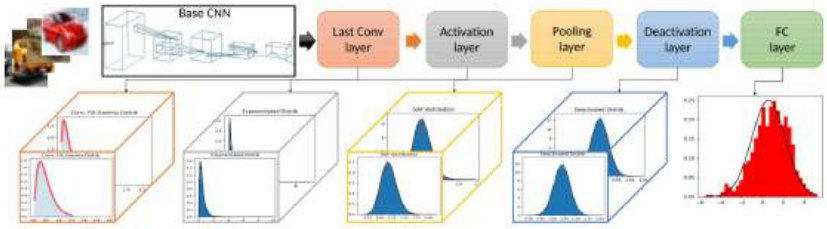


FIGURE 3.5: Distributions of the activations at the final convolution layer of VGG16

The Jensen-Shannon Divergence features within activation reduction methods in Chapters 4 and 5.

3.4.5 DNN Image Retrieval Descriptors

CBIR is intended for content-based image retrieval [109]. One method of image retrieval is to create descriptors for images using deep neural networks [176]. It is based on obtaining neural codes from fully connected layers activations. CBIR progresses this by using the information contained in convolutional layers. However, the number of neurons in the convolutional part is large and most of them do not contribute significantly to the final classification. Therefore the most significant neuron activations only are extracted in order to provide extra information about the image such as background textures or colour distribution that is present in the convolutional layers [176].

As shown in Figure 3.1, each block in the CNN (b_1 to b_N) consists of convolutional layers followed by a max pooling layer. Figure 3.6 shows the max pooling layer and the first convolutional layer of block N, the last block in the VGG16 model as shown

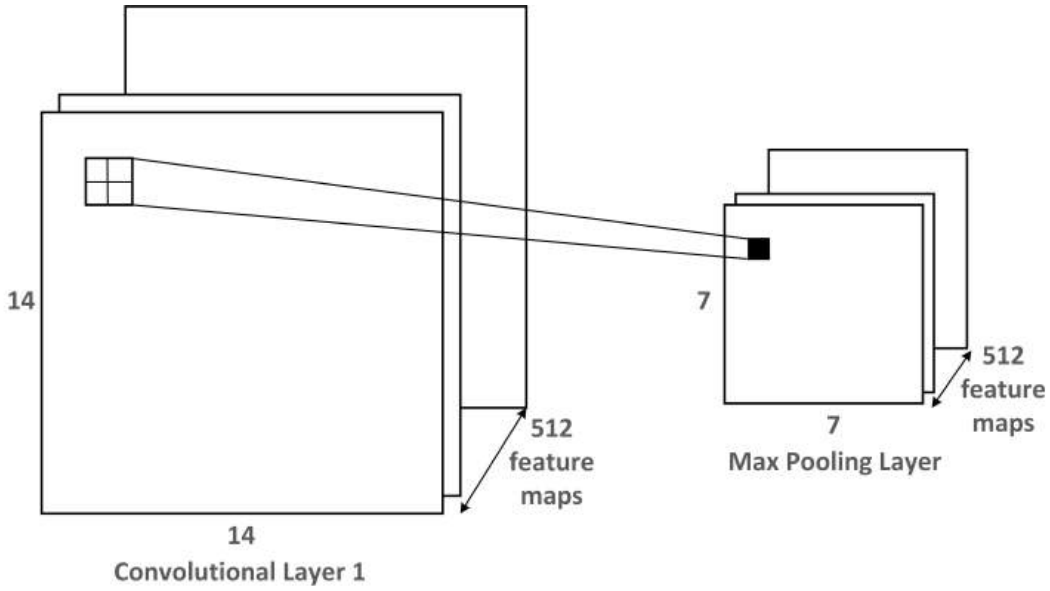


FIGURE 3.6: Representation of an important neuron projected back into the a convolutional layer

in Figure 3.1. Using this as an example of how the activations are reduced for each CNN block: The largest value from the corresponding position in each feature map in the max pooling layer is stored, giving a 7 x 7 one-dimensional matrix. A correspondingly sized matrix is created that stores a 1 if the neuron is over a threshold value, otherwise 0 (significance matrix, z). The neurons over the threshold value are projected back into the first convolutional layer, where the number of neuron values obtained is scaled up by the factor of difference between the max pooling layer and the convolutional layer. The convolutional neuron values in each projected area are summed. This is repeated for each feature map in the convolutional layer. Corresponding neuron locations on the other feature maps in the first convolutional layer are averaged to give one value per feature map, thus keeping the output dimensionality fixed. This single characteristic value for each feature map is calculated as in Equation 3.6:

$$\omega_k = \frac{\sum_H^{i=1} \sum_W^{j=1} (z)_{ij} (a_k)_{ij}}{\sum_H^{i=1} \sum_W^{j=1} (z)_{ij}} \tag{3.6}$$

Where ω_k is the single characteristic value for feature map k, z is the significance matrix for the block, a_k are the activation values for feature map k. H is the height and W is the width of the feature map and i and j specify matrix elements.

In Section 3.4 we have reviewed DNN activations and identified two methods on which to base their reduction in order to produce activation classification footprints. These footprints are then applied to streaming machine learning models, which are detailed in Section 3.5.

3.5 Streaming Machine Learning Models

In this section, we outline the streaming machine learning models that are used in our proposed solutions. Firstly, Section 3.5.1 gives the background of the streaming clustering model that is employed in the the proposed solution for outlier detection *DeepStreamOS*, in Chapter 4. This is followed by the streaming decision tree in Section 3.5.2, which is used in the proposed solutions *AdaDeepStream* and *DeepStreamEnsemble* in Chapters 5 and 6 respectively. Lastly, in Section 3.5.3, the background is given for the nearest neighbour model which is used to assist in the DNN adaptation in the *AdaDeepStream* solution in Chapter 5.

3.5.1 Micro-cluster-based Continuous Outlier Detection

MCOD is used within the outlier detection solution *DeepStreamOS* proposed in Chapter 4. Outlier detection methods that have been used in real-time streams are MCOD and AnyOut, with MCOD being an established outlier detection technique [72]. Tran [186] compares exact-Storm, Abstact-C, LUE, DUE, COD, MCOD and Thresh-LEAP and determines that MCOD demonstrates the superior performance among all of the algorithms because of its micro-cluster abilities.

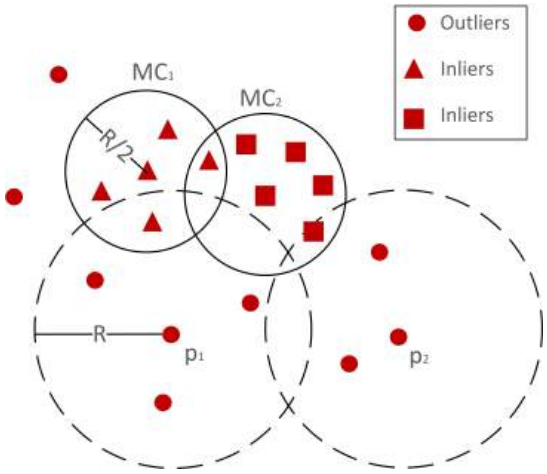


FIGURE 3.7: Example MCOD clusters for k=4 in an MCOD clusterer

Figure 3.7 shows the concept of MCOD. MCOD is based on a micro-clustering technique that takes parameters of: (1) Radius (R) of the micro-cluster (MC), (2) the minimum number of instances to form a micro-cluster (k) and (3) window size – the number of instances considered in the algorithm (W). If there are $k+1$ instances of p within $R/2$ of an MC, p becomes a member of that MC. If p is within $2R/2$ of any cluster, it becomes an outlier of those MCs. MCOD uses the centre of the micro clusters to perform its calculations, which makes it computationally efficient.

3.5.2 Hoeffding Tree

The Hoeffding tree is used within the concept evolution and concept drift detection solutions *AdaDeepStream* and *DeepStreamEnsemble* respectively, proposed in Chapters 5 and 6. The Hoeffding tree [91] is an incremental decision tree that uses the Hoeffding bound and is capable of learning from data streams. It is established and well used [47]. Before delving into the Hoeffding Tree, we first give some background for the standard decision tree.

Decision Trees

Decision trees are used for both classification and regression. They recursively partition the training instances into smaller subsets based on the values of input features, assigning a target value or class label to each subset. Examples of well known decision trees are CART [19] and ID3 [151]. CART (Classification and Regression Trees) employs the Gini Index (for classification only) as the metric, while ID3 (Iterative Dichotomiser 3) utilises the Entropy function and information gain as metrics. These classic decision tree learners operate with static data and necessitate the simultaneous storage of all training instances in memory, thereby constraining the number of instances they can learn from.

Figure 3.8 shows the basic concepts of a decision tree. This consists of a Root Node; the topmost node of the tree, which represents the entire dataset. The Internal Node, which are nodes other than the root and leaf nodes and they represent feature tests. Leaf Node, which are terminal nodes that hold the final predicted class or value. A node can be split into two or more child nodes. A criterion measure is used to determine the quality of a split (e.g., the Gini impurity or entropy). The

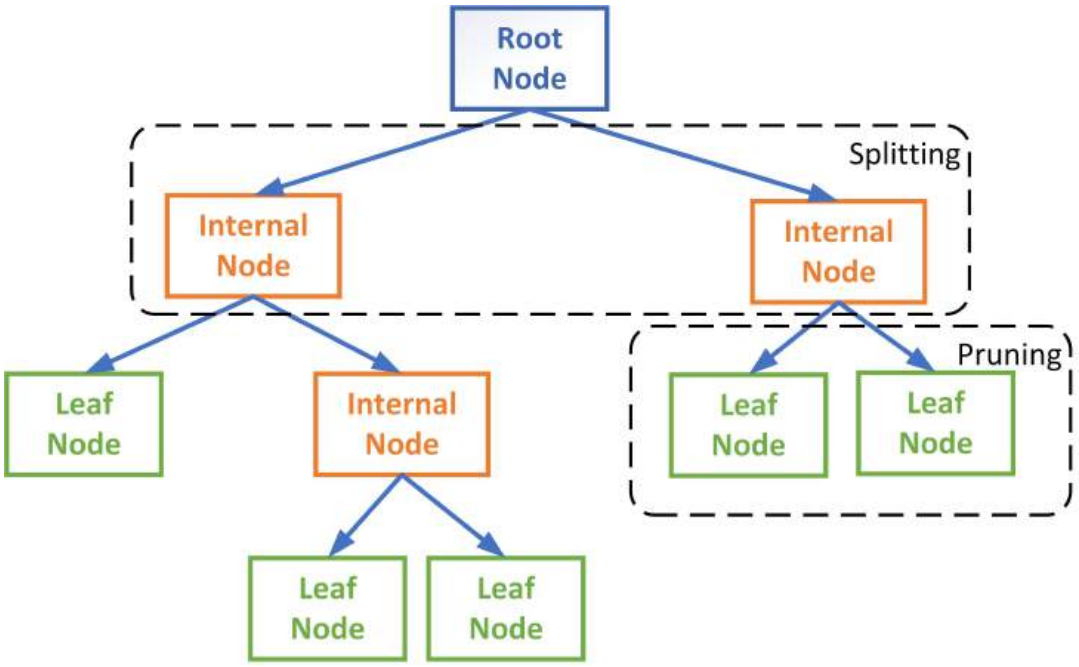


FIGURE 3.8: Decision Tree Architecture

Gini impurity measures the probability of incorrectly classifying a randomly chosen element in the dataset. For a binary classification problem with classes labeled 0 and 1, the Gini impurity at node t is calculated as in Equation 3.7:

$$Gini(t) = 1 - (p_0^2 + p_1^2) \tag{3.7}$$

Where p_0 is the proportion of class 0 instances in node t, and p_1 is the proportion of class 1 instances in node t. The entropy measures the impurity or disorder in a set of instances. It is calculated as in Equation 3.8:

$$Entropy(t) = -p_0 \log_2(p_0) - p_1 \log_2(p_1) \tag{3.8}$$

p_0 and p_1 are the same as defined for the Gini impurity (p_0 is the proportion of class 0 instances in node t, and p_1 is the proportion of class 1 instances in node t).

The information gain represents the reduction in entropy or impurity achieved by a split. Given a parent node p and its child nodes $c_1, c_2, ..., c_k$, the information gain IG for a split is calculated as in Equation 3.9:

$$IG(p) = Entropy(p) - \sum_{i=1}^{k} \frac{|c_i|}{|p|} \cdot Entropy(c_i) \tag{3.9}$$

Where $|c_i|$ is the number of instances in child node c_i and $|p|$ is the number of instances in the parent node p.

Construction of the decision tree involves selecting the best split at each internal node based on a criterion (e.g., Gini impurity or entropy). This procedure is iteratively carried out until a stopping criterion is satisfied, such as reaching a maximum depth, attaining a minimum number of samples per leaf, or reaching a minimum impurity threshold. Pruning is used to prevent overfitting by removing nodes that do not provide significant information gain. It involves removing nodes from the tree while maintaining its generalisation capability.

In summary, decision trees use a hierarchical structure to make predictions based on feature tests. The choice of split is based on criteria like Gini impurity or entropy, and the tree can be pruned to avoid overfitting. Decision trees are interpretable and easy to visualise, making them a popular choice in various applications. However, classic decision trees require knowledge of the entire dataset. For data stream classification, the amount of data is infinite and the arriving instances can only be processed once. In classic decision tree algorithms, this would cause a very high volume of data, making them unsuitable for streaming data.

Hoeffding Tree Formalisation

The Hoeffding tree is an incremental decision tree which uses a small number of instances, n to decide on a splitting attribute. The Hoeffding bound is used to guarantee that the splitting attribute is the correct choice with probability $1 - \delta$. Assuming that the information gain is used as the splitting measure and we have a real-valued random variable r, then the range R will be $log\ c$ (where c is the number of classes). The Hoeffding bound states that, with probability $1 - \delta$, the true mean of variable r is at least $r - \varepsilon$, where ε is defined in Equation 3.10.

$$\varepsilon = \sqrt{\frac{R^2 ln(1/\delta)}{2n}} \tag{3.10}$$

$$\Delta G \geq \Delta \bar{G} - \varepsilon > 0 \tag{3.11}$$

where ΔG is the change in information gain between the two best attributes in the data stream, and $\Delta\bar{G}$ is the change in information gain between the two best attributes in the small set of examples. The Hoeffding bound guarantees that with probability $1-\delta$, the small sample is representative of the data stream as expressed in Equation 3.11. Examples are accumulated from the stream until ε becomes smaller than $\Delta\bar{G}$, then the node is split using the current best attribute and subsequent examples are passed to the new leaves. This gives a very fast and efficient decision tree.

An extension of the Hoeffding Tree is the Hoeffding Adaptive Tree [16]. It supplements the Hoeffding Tree by monitoring the branches of trees using a drift detector [17], commonly ADWIN [17] for accuracy. If the accuracy decreases, the branch is replaced with a new branch, if the new branch is more accurate. It is used in our proposed solution in Chapter 5.

3.5.3 Self Adjusting Memory k-Nearest Neighbours

SAM-kNN is leveraged within the concept evolution and DNN adaptation solution *AdaDeepStream*, to assist in DNN adaptation as proposed in Chapter 5. SAM-kNN [117] is a popular Self Adjusting Memory (SAM) model for the k Nearest Neighbor (kNN) [160, 118]. It is based on the kNN algorithm [38, 49] but can handle heterogeneous concept drift, i.e., different types of concept drift occurring at different rates. To identify concept drift, SAM-kNN uses a biologically inspired memory model. This memory model is able to detect changes in the distribution of the data, even if the changes are small or gradual. Figure 3.9 shows the architecture and is adapted from [117]. SAM-kNN operates by maintaining two separate memories: The short-term memory (STM) and the long term memory (LTM). Incoming examples are stored within the STM. Previous information from former concepts that conflict with information from the current concept, is transferred from the short-term memory (STM) to the long-term memory (LTM). Knowledge accumulation is condensed whenever the available space is filled. Both models are taken into account during predictions, depending on their historical performances.

The STM memory M_{ST} represents the current concept and is a dynamic sliding

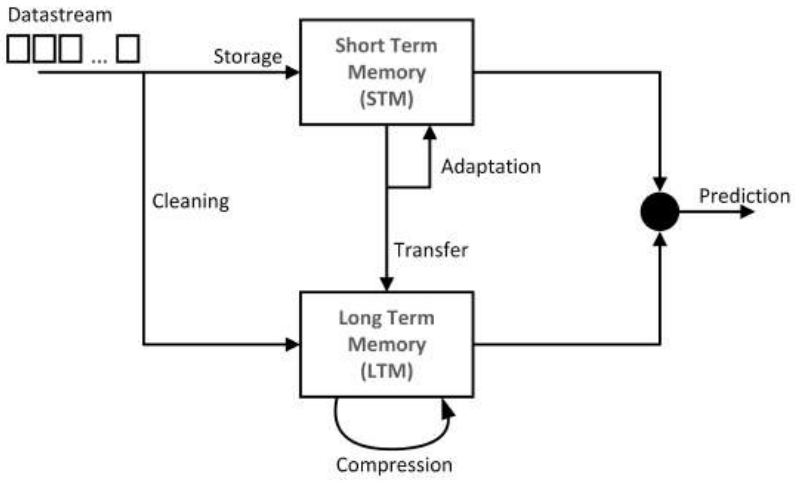


FIGURE 3.9: SAM Architecture

window containing the most recent m examples of the data stream. The LTM memory M_{LT} preserves a compressed version of all former information which is not contradicting those of the STM, a set of p points. There is also a combined memory M_C which is the union of both memories with size $m + p$. There is a distance weighted kNN classifier for each memory M_{ST}, M_{LT} and M_{CM}. The kNN model assigns a label for a given point x. Weights w_{ST}, w_{LT} and w_C represent the accuracy of the corresponding model on the current concept. The overall prediction of the model relies on the sub-model with the highest weight. The model is adapted incrementally at each timestep. During adaptation, the following parameters are adjusted: The size of the STM; the data points in the LTM; the weights w_{ST}, w_{LT} and w_C. The model has the following hyper-parameters: The number of neighbours k; the minimum length of the STM; and the maximum number of stored examples (STM and LTM combined).

A concept change is not explicitly detected, but the size is changed such that the interleaved test-train error of the remaining STM is minimised. This relies on the fact that a model trained on internally consistent data yields less errors and the remaining instances are assumed to represent the current concept or be sufficiently close to it. This is achieved by evaluating differently sized STMs and adopting the one with the minimum interleaved test-train error. The interleaved test-train error efficiently uses every example for test and training in the original order rather than cross-validation which requires multiple repetitions over the same data to provide a

stable estimation of the error.

The LTM contains all data for former concepts that is consistent with the STM. This requires cleaning of the LTM for each seen example. When drift is detected, the size of the STM is reduced by transferring as much data to the LTM as possible. Before this is transferred, examples are deleted from the discarded samples from the STM. Whenever the STM is shrunk, the discarded set is transferred to the LTM after cleaning. Set A are the instances leftover in the STM (old concept) that is to be transferred to the LTM. Set B is each incoming instance that has been stored from the stream as 'cleaning' instances. For each example in B (x_i, y_i), the nearest k neighbors of x_i in B is found and the ones with label y_i are selected to define a threshold. x_i is then applied to A and the nearest k neighbours which are less than the threshold defined from B are removed from A. Set B is also cleaned.

When the size limit of the LTM is reached, the LTM instances are condensed to a sparse knowledge representation via clustering, allowing longer conservation than simply discarding instances. For every class label, the corresponding data points are grouped, then the clustering algorithm kMeans++ [10] is used with a reduced number of clusters as compared to the original number of classes, producing prototypes representing the compressed original data.

In summary, the error of different sizes of the STM is monitored to detect drift. Meanwhile incoming instances from the data stream are also stored to be used in the 'cleaning' process. When the STM error becomes large, the leftover samples (old concept) are transferred to the LTM but before they go in, they are cleaned. Instances in the transferred STM memory are removed if they are not close enough to the 'cleaning' samples stored from the stream. Instances in the LTM are compressed to a sparse knowledge representation via kMeans++ clustering.

The disadvantages of SAM-kNN are that it needs to maintain two separate models, the STM and LTM, which means it requires more memory than some other models. This can be a problem for datasets with a large number of features or a large number of data points. It also needs to update its models dynamically, depending on the data that is being processed which can be computationally expensive for large datasets or for datasets with a large number of features. Now that we have an overview of streaming classifiers that are employed in our proposed solutions,

Section 3.6 focuses on the definitions of the data that is applied to our proposed solutions.

3.6 Concept Evolution and Concept Drift Definition

Concept evolution is the data discrepancy that our solution *AdaDeepStream* is focused on in Chapter 5. Concept drift is the data discrepancy that our solution *DeepStreamEnsemble* is focused on in Chapter 6. Hence, this section formally defines the terms and explains the differences between them.

Bayesian decision theory is commonly employed to describe classification procedures, taking into account the prior probability distribution of classes (referred to as p(y)) and the class conditional probability distribution (referred to as p(X | y)) [57, 95]. The classification decision relies on the posterior probabilities linked to the classes. The posterior probability associated with class c_i, given instance X, is determined via Equation 3.12:

$$P(c_i|X) = \frac{P(c_i) \cdot P(X|c_i)}{P(X)} \tag{3.12}$$

Where $P(X) = \sum_{i=1}^{m} P(c_i) \cdot P(X|c_i)$ is the prior probability distribution. If concept drift occurs between times t_0 and t_1, this gives Equation 3.13:

$$\exists X : p_{t_0}(X, y) \neq p_{t_1}(X, y) \tag{3.13}$$

Figure 3.10 shows the different types of concept drift. In the literature, Real and Virtual concept drift are collectively referred to as 'concept drift'. Class prior concept drift pertains to changes in the class prior probability. This form of drift may manifest as class imbalance, the emergence of novel classes, or the fusion of existing classes. In previous surveys on concept drift [86, 57, 45], Class Prior Concept Drift was not considered as a distinct type of drift. It is categorised as virtual drift when it leads to class imbalance without affecting the decision boundaries. It is regarded as real drift when it causes prior class evolution and thus affects decision boundaries.

The term "class prior concept drift" was first introduced in [96]. Treating it as a distinct type of drift is pertinent in various real-world applications. For example, in

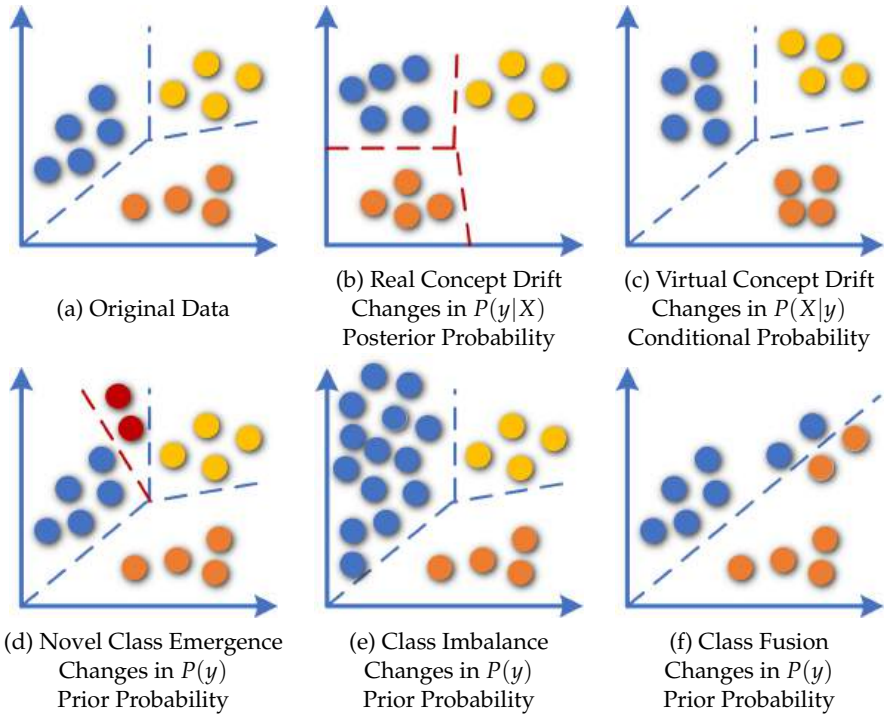


FIGURE 3.10: Types of Concept Drift

the biomedical field, certain viruses may acquire resistance to previously effective antibiotics. Consequently, the class of resistant viruses may undergo evolution over time and become more dominant. In such a scenario, the class prior distribution has altered, signifying the occurrence of a class prior concept drift [96]. In the literature, novel class emergence is often considered separately as concept evolution. Thus, in this thesis, the applied concept drift is real and virtual concept drift. Concept evolution is novel class emergence.

An example of concept evolution (or novel class emergence) applied in this thesis is where a DNN is trained on images of cats and dogs, then a frog is presented to the network. An example of concept drift (real or virtual) is where a DNN is trained on images of super-classes of i.e. Animals and Transport, where the Animals category consists of sub-classes of cats and dogs and the super-class of Transport consists of sub-classes of Trucks and Aeroplanes. A frog is presented to the network, it's super-class is Animal. With this example, it is likely that this would not cause a decision boundary change, as the super-classes Animal and Transport have quite different characteristics. This is virtual concept drift, the input data has changed, but the class boundaries have not. If the super-classes were more similar, this may manifest in

class boundary changes, resulting in real concept drift.

A practical example of real concept drift is in satellite images, where land may be categorised as agricultural, but over time, development creeps in and it becomes urban; there has been a genuine change in the data and the model requires updating. Virtual concept drift can occur when there are changes in the data, but the actual terrain is the same. i.e. if atmospheric conditions change, or a sensor fails, causing the image to change. In one pass, it is sunny and the terrain is classified as agricultural, however, a change in atmospheric conditions may cause the model to misclassify the image on the next pass. The model's performance degrades, but the underlying terrain usage has not changed, the differences are due to atmospheric conditions. Now that concept evolution and concept drift has been described and formalised, we move onto reviewing existing drift detection methods in Section 3.6.1.

3.6.1 Drift Detection Review

Our solution concerning concept evolution in Chapter 5 involves the use of an existing drift detection method. In Section 2.3.2, we reviewed concept drift methods that were specifically for concept drift detection and adaptation for images. In this section we review drift detection methods only. They operate on detecting the differences between two inputs provided to them, and they signal possible changes.

Concept drift for online learning have been surveyed several times: (Widmer and Kubat 1996 [196]; Tsymbal 2004 [187]; Quionero-Candela et al. 2009 [150]; Zliobaite et al. 2012 [213]; Gama et al. 2014 [57]). Concept drift detection algorithms can be categorised into (1) sequential, (2) adaptive windowing and (3) statistical [119, 200, 6].

Sequential based methods analyse whether the drift has occurred and predict the drift based on the accuracy evaluation. Examples are Cumulative Sum (CUSUM) [142] and Geometric Moving Average (GMA) [159]. CUSUM calculates the difference of observed values from the mean and raises an alarm when it is significantly different. GMA uses a forgetting factor to weight the latest data and a threshold is used to tune the false alarm rate. However, these algorithms require tuning of the parameter values, resulting in a trade-off between false alarms and detecting true drifts.

Windowing methods use fixed or dynamic windows that summarise information; that information is then used to make a comparison between the previously summarised windows and the current window to detect drift. Examples of adaptive windowing methods are Adaptive Windowing (ADWIN) [17] and Kolmogorov Smirnov Windows (KSWIN) [152]. ADWIN uses sliding windows of variable size and if two windows are found that have distinctly different averages, then the data distribution is deemed to have changed. KSWIN is based on the Kolmogorov statistical test and has no assumption of the underlying data distribution. Another adaptive windowing method is HDDM [54] which is based on Hoeffding bounds and monitors the data distribution of different time windows using probability inequalities instead of the probability distribution function. It compares the moving averages to detect the drifts and uses a forgetting scheme to find the weight of moving averages in the data stream. However, it needs to explore with different weighting schemes for application to real-world problems [6].

Statistical methods are based on parameters like mean and standard deviation to predict drift. Examples are DDM [58] and EDDM [11]. DDM is based on the Probably Automatically Correct (PAC) learning model [58]. It assumes the binomial distribution and uses the standard deviation to detect drift and works well on abrupt drift. EDDM improves upon DDM by increasing the detection of gradual drift whilst maintaining a good abrupt concept drift detection rate by using a distance error rate instead of the classifiers error rate. The statistical methods are generally faster than the sequential or adaptive windowing methods, and although DDM is one of the older algorithms, it is one of the most accurate and fastest [66].

We conducted experiments on concept drift detection from the activations of the deep neural network using streaming decision tree/drift detector combinations on seven drift patterns as defined in Chapters 5 and 6. The test data instances are applied as in Figures 5.4 and 5.5. We applied ADWIN, KSWIN, HDDM, EDDM and DDM drift detectors. We measured the F1-Score for detection, using the following metrics for the F1-Score calculation: True positives are defined as images that belong to the new class and were correctly identified as concept drift. False positives are defined as images that belong to an existing class and were incorrectly identified as concept drift. True negatives are defined as images that belong to an existing

class and were correctly identified as such. We found that the DDM and EDDM drift detectors significantly outperformed the ADWIN, KSWIN, HDDMA and HDDMW methods. DDM and EDDM are statistical control based drift detectors whilst ADWIN, KSWIN, HDDMA and HDDMW are adaptive windowing methods that monitor the data distribution of different time windows. Overall, in our setting, the DDM method produced improved results as compared to EDDM. We also experimented with combining the more successful drift detection methods in an ensemble. The ensemble results were only a slight improvement on DDM. Due to the simplicity of using one method as opposed to an ensemble for little performance gain, DDM was favoured. Hence, the drift detection method selected to be used as the drift detector in *AdaDeepStream* in Chapter 5 is DDM. Section 3.6.2 details and formalises this method.

3.6.2 Drift Detection Method

DDM (Drift Detection Method) [58] is used as the drift detection method in our proposed solution *AdaDeepStream* in Chapter 5. It is a statistical method, it assumes the binomial distribution and uses the standard deviation to detect drift. It is based on the PAC learning model [58], whereby the error rate will decrease as the number of analysed samples increases, as long as the data distribution is stationary. If a rise in the error rate is identified that surpasses a calculated threshold, it triggers either a change or a warning of a change. The detection threshold is computed using two statistics, acquired when $(p_i + s_i)$ is minimum: The minimum recorded error rate (p_{min}) and the minimum recorded standard deviation (s_{min}). At instance i, the detection algorithm employs the error rate at instant i (p_i) and the standard deviation at instant i (s_i). The conditions for entering the warning zone are shown in Equation 3.14 and the conditions for detecting change are shown in Equation 3.15:

$$\text{if } p_i + s_i \geq p_{min} + 2 * s_{min} \rightarrow \text{warning} \tag{3.14}$$

$$\text{if } p_i + s_i \geq p_{min} + 3 * s_{min} \rightarrow \text{change} \tag{3.15}$$

DDM is able to detect abrupt and gradual drift but not very slow gradual drifts [11] as examples are stored for an extended period and the drift level can take a considerable duration to be triggered, requiring a large amount of memory. It is also sensitive to noise.

3.7 Summary

This chapter discussed the background and theoretical definitions of the algorithms used as the basis of the novel methods of *DeepStreamOS* (Chapter 4), *AdaDeepStream* (Chapter 5) and *DeepStreamEnsemble* (Chapter 6) proposed in this thesis. The experimental study involves image data as the input data and CNNs as the DNN. Hence, a background to the curse of dimensionality has been provided along with CNNs and their activations. Subsequently, the streaming machine learning models employed in our proposed solutions have been explained and formalised. *DeepStreamOS* uses the JS-Divergence as a tool to reduce DNN activations, and MCOD to detect outliers. *AdaDeepStream* uses the JS-Divergence in a different way, and a DNN-based image retrieval descriptor generation method to reduce DNN activations. The Hoeffding Tree classifier is used with the activations and DDM to detect concept evolution. SAM-kNN is used to assist the DNN adaptation process. *DeepStreamEnsemble* also uses the DNN-based image retrieval descriptor generation method to reduce DNN activations but applies it in a different way, with Hoeffding trees in an ensemble for the detection and adaptation processes. The definitions for concept evolution and concept drift were provided to deepen understanding of these discrepancy types. These background methods are referenced in the following three contribution chapters.

Chapter 4

DeepStreamOS: Open-Set Classification in DNNs

In the previous chapter, the necessary background to understand the proposed solutions, along with the theoretical explanation of the algorithms employed in these solutions is provided. In this chapter, we propose a novel outlier detection method using JS-Divergence and MCOD to address the second objective of this thesis as outlined in Section 1.5: **Design, develop and evaluate an Open-Set (outlier) discrepancy detection and DNN adaptation method.** The work discussed in this chapter is published as a paper titled "DeepStreamOS: Fast Open-Set classification for convolutional neural networks" in the "Elsevier Pattern Recognition Letters" journal [26]. The code is available at https://github.com/chambai/DeepStreamOS.

4.1 Introduction

DNNs predict classifications based on the data they are trained on and have achieved state-of-the-art performance in classification tasks [105, 180]. However, when the unseen instances are presented and they deviate from the training set distribution, they can be incorrectly classified. This is problematic in safety critical systems such as autonomous vehicles, flight control, medical image classification or medical sensor analysis. For instance, DNNs have been used widely to develop autonomous vehicles, and various safety incidents have been reported in the media such as Google's self driving car hitting a bus [106] and a Tesla driver's fatal crash [199]. Fast detection of unknown classes in these circumstances is essential.

The basic recognition categories for classes in open-set classification are asserted

in [168], where *unknown unknown* classes are defined as classes unseen in training and also not having any meta information during training such as semantic or attribute information [63]. In this chapter, we aim to detect *unknown unknowns* but shall refer to them as unknowns. This research focuses on widely used convolutional neural networks (CNNs) of VGG16 [173] and MobileNet [90], which classify images, utilising image data from CIFAR-10 [103] and Fashion-MNIST [198] datasets. We produce open-set unknown images by withholding classes from the dataset during training, then apply unseen instances of known and withheld unknown classes during testing. There are two different types of unknown images applied; those that introduce a novel class (termed Class data), and those that introduce a new sub-type of image to an existing class (termed Sub-Class data).

We analyse instances by utilising activations from the hidden layers within the DNN, reducing the activations and applying this to a streaming analysis method for outlier detection. Apart from our own previous work [27], to our knowledge, there have been no studies analysing the activation data with streaming analysis techniques. We represent multi-layer activations from the DNN using Jensen-Shannon Divergence (JS-Divergence) and apply open-set classification using MCOD (Micro-cluster-based Continuous Outlier Detection) streaming outlier detection [99] to provide fast analysis and detection of unknown instances.

We compare our system (*DeepStreamOS*) to leading DNN open-set classification solution, OpenMax [13] and Extreme Value Machine, EVM [162]. These comparison methods have been selected as they are similar to our method in that they are lightweight and do not augment the training data. Both OpenMax and EVM have been used as comparison for other successful open-set classification methods such as [41, 203] and show high F1-Scores under the datasets and networks they have been tested on. We show our methods effectiveness in comparison to OpenMax and EVM via F1-Scores and the efficiency via the speed per instance during inference.

We can summarise our contributions in this chapter as follows:

1. We apply a statistical method to quantify the difference in activation distribution between any two consecutive hidden layers of DNNs to get a dynamic trajectory of activations.

2. We use fast interpretation of the reduced activations via a stream-based outlier detection method to detect open-set images.

The rest of this chapter is organised as follows: In Section 4.2 we present a system description including formalisation and implementation details of the *DeepStreamOS* components and methodology. In the experimental study in Section 4.3 we specify the experimental setup specific to this chapter. In Section 4.4, we evaluate and analyse *DeepStreamOS* on Class and Sub-Class data from the CIFAR-10 and Fashion-MNIST datasets. The same data and DNNs are applied to open-set DNN solutions OpenMax and EVM and the results are compared.

4.2 DeepStreamOS System Description

This section details our *DeepStreamOS* system components and their interactions via descriptions and algorithms, presenting our JS-Divergence based activation reduction method and our concept evolution detection method as *DeepStreamOS*. Figure 4.1 shows the *DeepStreamOS* system and Table 4.1 lists the symbols used to describe our system.

TABLE 4.1: Summary of symbols

Symbol	Description	Symbol	Description
h	Layer number	n	Number of known classes
i	Instance	p	Predicted class
j	JS-divergence value	r	Outlier result
J	JS-divergence set	R	Radius of MCOD micro-cluster
k	Min inst per micro-cluster	S	A set of inlier/outlier decisions
l	Layer	W	MCOD window size
M	MCOD clusterer	y	True class
MC	MCOD micro-cluster		

Prerequisites for the system are: (1) A trained DNN that Open-Set Classification is being applied to, and (2) the data instances that the DNN is trained on. To train *DeepStreamOS*, the training instances are presented to the DNN. The activations of each training instance are extracted, then the activation data proceeds through two stages: (1) Activation reduction and (2) streaming analysis setup. In Figure 4.1, an image i is applied and the activations are extracted. For the activation data reduction, we calculate the JS-Divergence between each pair of consecutive hidden layers

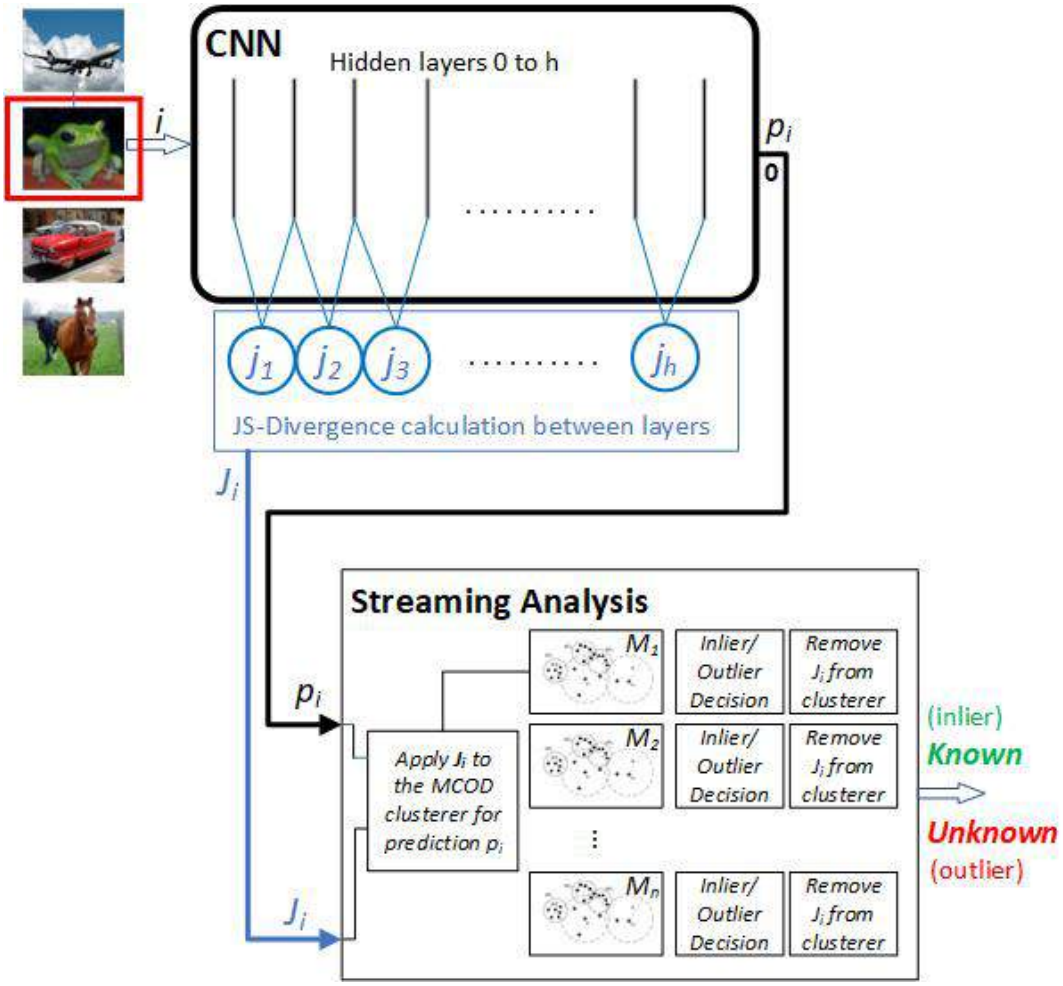


FIGURE 4.1: *DeepStreamOS* System Overview.

(j_1 to j_h). Equations 3.4, 3.3 and 3.5 show how JS-Divergence is calculated. These values are stored in set J_i to provide a dynamic representation of the activations of a DNN, termed an activation classification footprint. An MCOD clusterer M contains all of the JS-Divergence training instance footprints for one known class. There are M_1 to M_n MCOD clusterers, one for each known class. In this example, the CNN is trained on airplanes, automobiles and horses. An unknown image of a frog is applied, the JS-Divergence is calculated per pair of hidden layers (j_1 to j_h) and stored as one set, J_i. The predicted class, p_i is 0 (airplane). J_i is applied to the MCOD clusterer for class 0 (M_1). Outlier analysis is applied to M_1. If it is an inlier, it is *known*. If it is an outlier, it is *unknown*. The following sections provide details and algorithms for the activation reduction, training and inference of our system.

4.2.1 Activation Reduction

Activation reduction is achieved via the JS-Divergence measure, which is explained in detail in Section 3.4.4. To recap, JS-Divergence is a method of measuring the similarity between two probability distributions. The activation reduction method is shown in Algorithm 1. For brevity in our algorithms, we refer to JS-Divergence as JSD. For each layer, starting at the second layer, the JS-Divergence is calculated between the current layer and the previous layer (lines 1 to 6). To compare layers using JS-Divergence, the number of activations utilised from each neighbouring layer needs to be the same size. The layer is flattened (line 4), the largest layer size in the CNN is selected and if the layer sizes are smaller, they are padded with zero's (line 5), then the JS-Divergence is calculated between the neighbouring layers (line 6). This yields one value between each activation layer per data instance. These values are stored in set J_i (line 7). These values represent activation classification footprints for the instances as discussed in Section 2.6. The same process is used during training and inference.

Algorithm 1 *DeepStreamOS* Activation Reduction

Input: Pretrained DNN on n classes
Input: Instance i
Output: J_i: Set of JS-Divergence calculations between layers

```
1:  layer number, h = 0
2:  for each layer, l_i do                          ▷ at layer level
3:      if h > 0 then                               ▷ skip first layer
4:          flatten(l_i)                            ▷ make layer activations 1D
5:          pad(l_i,l_i − 1)                        ▷ make layer activations same size
6:          j_l = jsdiverge(l_i,l_i − 1)            ▷ calc JSD between consecutive layers
7:          Add j_l to JS-Divergence set, J_i       ▷ store JSD value
8:      end if
9:      h += 1
10: end for
```

4.2.2 Outlier Detection

The streaming outlier detection method based on MCOD [99] is used for the streaming analysis. MCOD is a streaming clustering algorithm and is explained in detail in Section 3.5.1. Before outlier detection can occur, the MCOD clusterers need to be trained. As shown in Figure 4.1, there is one MCOD clusterer M per known class.

Reduced activation instances are added to the MCOD clusterer that represents their classification label. Algorithm 2 details this procedure. For each training instance, the DNN prediction is acquired (line 2). If the DNN predicted value matches the true value of the training instance, then the activation classification footprint is generated (line 4) as described in Algorithm 1. Subsequently, the activation classification footprint data is added to the MCOD clusterer pertaining to the class of the instance (line 6). This results in n trained MCOD clusterers M_1 to M_n. Micro clusters (MC) may be formed within the MCOD clusterer (M), however, during the training phase we are not interested in these micro clusters as we do not require inlier/outlier decisions. W (MCOD window size) is set to the number of training instances for that class plus one and the effect of varying k (the minimum number of instances to form an MCOD micro-cluster) and R (the radius of the MCOD micro-cluster) is evaluated to determine optimum values.

Algorithm 2 *DeepStreamOS* Training

Input: Pretrained DNN on n classes
Input: Empty MCOD clusterers $\mathbf{M}_1 ... M_n$ for n known classes
Output: n Trained MCOD clusterers $\mathbf{M}_1 ... M_n$

1: **for** *training instance*, i and its true value, y **do** ▷ at instance level
2: p_i = **getPredictedClass**(i) ▷ DNN predicted class
3: **if** $p_i = y_i$ **then** ▷ only add instance to MCOD if correctly predicted
4: J_i = **getJSD**(i) ▷ get JSD values
5: M_y = MCOD clusterer for true class y_i
6: **addToClusterer**(M_y,J_i) ▷ add reduced instance to MCOD clusterer
7: **end if**
8: **end for**

Algorithm 3 *DeepStreamOS* Inference

Input: Pretrained DNN on n classes
Input: Trained MCOD clusterers $\mathbf{M}_1 ... M_n$ for n known classes
Output: S: A set of results

1: **for** *unseen instance*, i **do** ▷ at instance level
2: J_i = **getJSD**(i) ▷ get reduced activations (footprint)
3: p_i = **getPredictedClass**(i) ▷ DNN predicted class
4: M_p = MCOD clusterer for class p_i
5: r_i = **addToClusterer**(M_p,J_i) ▷ determine if outlier
6: Add r_i to results, R
7: **removeFromClusterer**(M_p,J_i) ▷ remove instance from MCOD
8: **end for**

The algorithm for the open-set classification (inference) is presented in Algorithm 3. For previously unseen instances arriving at the DNN, the activation classification pattern is extracted (line 2) as detailed in Algorithm 1. The DNN predicted class of the instance is obtained (line 3) and the reduced instance is added to the MCOD clusterer for the predicted class (line 5), giving an outcome r, as to whether the reduced instance is an inlier or an outlier with respect to the MCOD clusterer it was added to. If it is an inlier, the instance is considered to be known and if it is an outlier, the instance is considered as unknown. The instance is then removed from the MCOD clusterer (line 7) so that the clusterer only contains the training data and the next instance is not affected by the previous instance.

4.3 Experimental Methodology

This section provides details of the experiments conducted with details of the specific setup implemented for the proposed method in this chapter.

4.3.1 Datasets

For the experiments we use two datasets; CIFAR-10 [103] and Fashion-MNIST [198]. The CIFAR-10 dataset consists of 10 different classes of 32×32 colour images. In total there are 50000 training images and 10000 test images. The Fashion-MNIST dataset consists of 10 different classes of 28×28 greyscale images. In total there are 60000 training images and 10000 test images. Each of the classes is assigned a class number ranging from 0 to 9. These classes can also be split into categories: Transport and Animal for CIFAR-10 and Footwear and Clothing for Fashion-MNIST. These datasets were selected as they naturally lend themselves to these super-class categorisations. Fashion-MNIST images originate from photographs of fashion items, shot by professional photographers, demonstrating different aspects of the product, for instance, front and back. Examples of these images are shown in Figure 4.2 (a) (image adapted from [198]). In order to explore real photographic images rather than illustrated, artificially generated or altered images, the CIFAR-10 dataset is employed. CIFAR-10 provides a similar amount of instances to Fashion-MNIST. The CIFAR-10 dataset was manually selected and annotated by humans who were asked to ensure the images were photo-realistic [103]. An example of these images is

shown in Figure 4.2 (b) (image adapted from [103]). CIFAR-10 is considered a hard dataset with variations in colour, illumination, and background [165]. These datasets are widely used as benchmark datasets in image classification [165, 170, 20].

(a) Fashion-MNIST (b) CIFAR-100

FIGURE 4.2: Examples of images for Fashion-MNIST and CIFAR-10

Tables 4.2 and 4.3 lists the classes and their assigned categories.

4.3.2 Data Combinations

Two types of data combinations are experimented with; we shall call them (1) Class and (2) Sub-Class. For the Class data combination, the DNNs are trained on the original class labels. For example, the network will be trained on classes Ship and Truck and class Dog is applied as an open-set class, so the open-set class has a novel class label compared to that which the network was trained on. For the Sub-Class data combination, the DNNs are trained on the category labels. For example, the

TABLE 4.2: CIFAR-10 Classes

Class ID	Class Name	Category
0	Airplane	Transport
1	Automobile	Transport
2	Bird	Animal
3	Cat	Animal
4	Deer	Animal
5	Dog	Animal
6	Frog	Animal
7	Horse	Animal
8	Ship	Transport
9	Truck	Transport

TABLE 4.3: Fashion-MNIST Classes

Class ID	Class Name	Category
0	T-Shirt/Top	Clothing
1	Trouser	Clothing
2	Pullover	Clothing
3	Dress	Clothing
4	Coat	Clothing
5	Sandal	Footwear
6	Shirt	Clothing
7	Sneaker	Footwear
8	Bag	Footwear
9	Ankle Boot	Footwear

network will be trained on classes Cat, Horse, Ship and Truck, but instead of using these class labels, class labels of Animal, Animal, Transport and Transport are used. The Frog class is applied as the open-set data, which has a class label of Animal, so the input data changes without the class label changing; a Sub-Class has been applied as the open-set data.

For the Fashion-MNIST dataset, the class of Bag has been included in the Footwear category as it is made of similar material and is commonly purchased with footwear. Tables 4.4, 4.5, 4.6 and 4.7 show the selected combinations and which category they relate to. The Data ID column is the combination of classes represented as *known classes - unknown classes*. The combinations have been selected such that there are groups of 2, 4 and 6 known classes and incrementing numbers of unknown classes.

The x-axis of Figures 4.3, 4.4, 4.5 and 4.6 are data combinations, the nomenclature of which is *known-unknown* classes. The known class IDs are specified first, separated with a hyphen, then the unknown class IDs are specified. For instance, in Figure 4.3 the class combination of *01-6* signifies that the known classes are Airplane and Automobile and the unknown class is Frog. The class IDs for the datasets can be found in Tables 4.2 and 4.3.

4.3.3 Experimental Settings

The DNNs that have been applied to the *DeepStreamOS* system are the widely used models; VGG16 [173] and MobileNet [90]. We have utilised transfer-learning from ImageNet weights and then trained each of these networks on the known classes specified in the Data ID column of Tables 4.4, 4.5, 4.6 and 4.7. Both of these DNNs are CNNs. All hidden layers that have measurable outputs in Keras [36] are utilised. VGG16 has 20 hidden measurable layers and MobileNet has 88 hidden measurable layers.

For *DeepStreamOS*, the MCOD parameters k (the minimum number of neighbours required to form an MCOD micro cluster) and R (the radius of the micro cluster) require selecting. Based on empirical results and results from previous work [27], k is set to 80. We conducted a parameter tuning exercise to investigate the effect that the R (the radius of the micro cluster) has on the F1-Score. This parameter tuning was conducted on a subset of Class and Sub-Class data combinations from

Figures 4.3 to 4.6, trained and inferred in the same way as the method described in this chapter. For information, the results are shown in Appendix B.3. The selected values are shown in the bar chart titles in Figures 4.3 to 4.6. For the test data, there are 1000 instances per class. Combinations of known and unknown classes are applied in equal proportions. For instance, if the CIFAR-10 combination of *0189-23* is applied then there would be 4000 known classes and 2000 unknown classes. The number of instances in each of the known classes were reduced by half to provide a balanced amount of unknown to known instances.

As stated in Section 4.1, we compare our system to the leading open-set classification solutions of OpenMax [13] and Extreme Value Machine, EVM [162]. An overview of these methods follows:

1. **OpenMax** uses activation vectors to estimate the probability of deep network failure. From correctly classified training samples, activations from the final hidden layer are used to calculate a Mean Activation Vector (MAV). The distances between the training samples and their corresponding class MAVs are computed and used to fit a separate Weibull (an Extreme Value theory (EVT) probabilistic model) distribution for each class. The values of the activation vector are then redistributed according to the score of the Weibull distribution fit, and these redistributed values are used to compute an activation for unknown classes. The class probabilities for known classes and unknown classes are calculated by applying Softmax again, but this time on the newly redistributed activation vectors [13].

2. **EVM** uses EVT and the margin distribution to further estimate the probability of sample inclusion in each class. EVT provides the functional form for the radial probability of inclusion of a point in a class with respect to its other surrounding classes. The points and distributions that best summarise each class (the points that are the least redundant with respect to one another) give a compact probabilistic representation of the boundary for each class in terms of its extreme vectors. This provides a nonlinear classifier with radial inclusion functions that are similar to Radial Basis Function kernels but have variable bandwidths and skew [162].

These comparison methods are similar to our *DeepStreamOS* method in that they do not augment the training data; they are discriminative rather than generative methods, and therefore, do not generate data to fill the open space [63]. Similar to our method, OpenMax uses activations from within the network, but only uses the last layer whereas we use a trajectory of activations across all layers. EVM uses statistical methods only. Similarly to OpenMax, it employs EVT but also uses the margin distribution to improve upon OpenMax. These are both successful methods used for comparison in open-set literature [41, 203]. In summary, OpenMax and EVM are both statistical-based methods and focus on EVT, discriminating between objects via extreme features rather than average ones and both rely on distances. OpenMax uses the activations of the final hidden layer of a DNN, whereas EVM uses the image data directly.

No incorrectly classified instances are removed during testing in order to more closely simulate real-world applications. *DeepStreamOS* is compared to OpenMax and EVM open-set methods using identical data. The F1-Score metric with out of sample as the positive class is used as analysed in [81], leading to the following definitions: True positives are unknown images that are classified as unknown. False positives are known images that are classified as unknown. False negatives are unknown images that are classified as known.

To compare *DeepStreamOS* with OpenMax, we are using a modified version of OpenMax that we have adapted to work with our data and DNNs. The parameters of alpha and tail required modification for our data and optimum values for these were found empirically to be 2 and 9, respectively by running a subset of data combinations (two from each data combination for each number of trained classes for class and sub-class data) with a range of values between 1 and 20 for both alpha and tail. To compare with EVM we are using the EVM PyPI installer associated with the research paper [162]. The tail size parameter was tuned and optimum values were found to be 40 for all Class data combinations, 60 for CIFAR-10 Sub-Class data combinations and 20 for Fashion-MNIST Sub-Class data combinations. A subset of data combinations was selected (two from each data combination for each number of trained classes for class and sub-class data) with a range of values between 10 and

100. The distance function of cosine was selected as it yielded slightly improved results over Euclidean distance. There is an option to apply a threshold to the amount of training data as this speeds up the training process. This has been set to 1 to use all of the training data in order to compare directly with our method and OpenMax. The output of EVM is the probability of the unseen instance belonging to each trained class. The class with the maximum probability is taken. If the probability is greater than 0.5, the instance is marked as belonging to that class (known). If the probability is less than 0.5, the instance is deemed not to belong to any of the classes (unknown). Our system is operating on HP Spectre x360 Intel i7-8750H CPU, 8GB RAM.

4.4 Experimental Results

In this section, we discuss our thorough experimental study, evidencing the efficacy of the proposed method.

Table 4.8 shows the average F1-Scores and the standard deviation for each DNN, data combination type and dataset. For Class data combinations, Table 4.8 shows that *DeepStreamOS* achieves F1-Scores, between 0.660 and 0.709. In figures 4.3 and 4.4, *DeepStreamOS* generally performed less well in the combinations that only had two known classes. The VGG16 network shows lower F1-Scores for Fashion-MNIST and comparative F1-Scores for the CIFAR-10 as compared to MobileNet, probably because there are less layers in VGG16 than MobileNet, and therefore less JS-Divergence calculations contributing to the MCOD clustering. When the number of known classes remains the same and the number of unknown classes are increased, typically there is a drop in the F1-Score when the largest amount of unknown classes are applied i.e. Figure 4.4, combination *5789-012346* and is most prevalent in the VGG16 network/Fashion data in Figure 4.4. DeepStreamOS performed well on the known classes *0189* in figure 4.3 and the known classes *5789* in Figure 4.4. These both contained a single Transport/Animal or Clothing/Footwear category respectively.

For Sub-Class data combinations, Table 4.8 shows that DeepStreamOS achieves F1-Scores between 0.662 and 0.720. In Figures 4.5 and 4.6, DeepStreamOS achieves

consistently higher F1-Scores for all Sub-Class data combinations than OpenMax. A decrease in the F1-Score can be seen in Figure 4.5 for the four known classes of *0123* for VGG16 network/CIFAR-10 data and less so in Figure 4.6 for known classes *012579* for the VGG16 network. Again, indicating that DeepStreamOS is less accurate on the smaller VGG16 network. However, the number of classes is limited to only two categories of Transport/Animal and Clothing/Footwear in this Sub-Class data combination scenario.

DeepStreamOS outperforms OpenMax in 97% of data combinations and EVM in 43%, with *DeepStreamOS* outperforming EVM for the Fashion-MNIST dataset as shown in Figure 4.6. The overall average F1-Score of *DeepStreamOS* exceeds EVM by 2.1% and OpenMax by 29.6%. Using the Wilcoxon Signed-Rank test, the difference between the F1-Score of *DeepStreamOS* and that of OpenMax over the 170 tested data combinations is statistically significant. The p-value is less than 0.00001 which is less than 0.05 significance level, suggesting the acceptance of the alternative hypothesis that true location shift is not equal to 0. These results indicate that *DeepStreamOS* performs more accurately on networks with more layers and on a mixture of Transport/Animal and Clothing/Footwear categories. However, when there are only two known classes or when the number of unknown classes are increased, performance decreases.

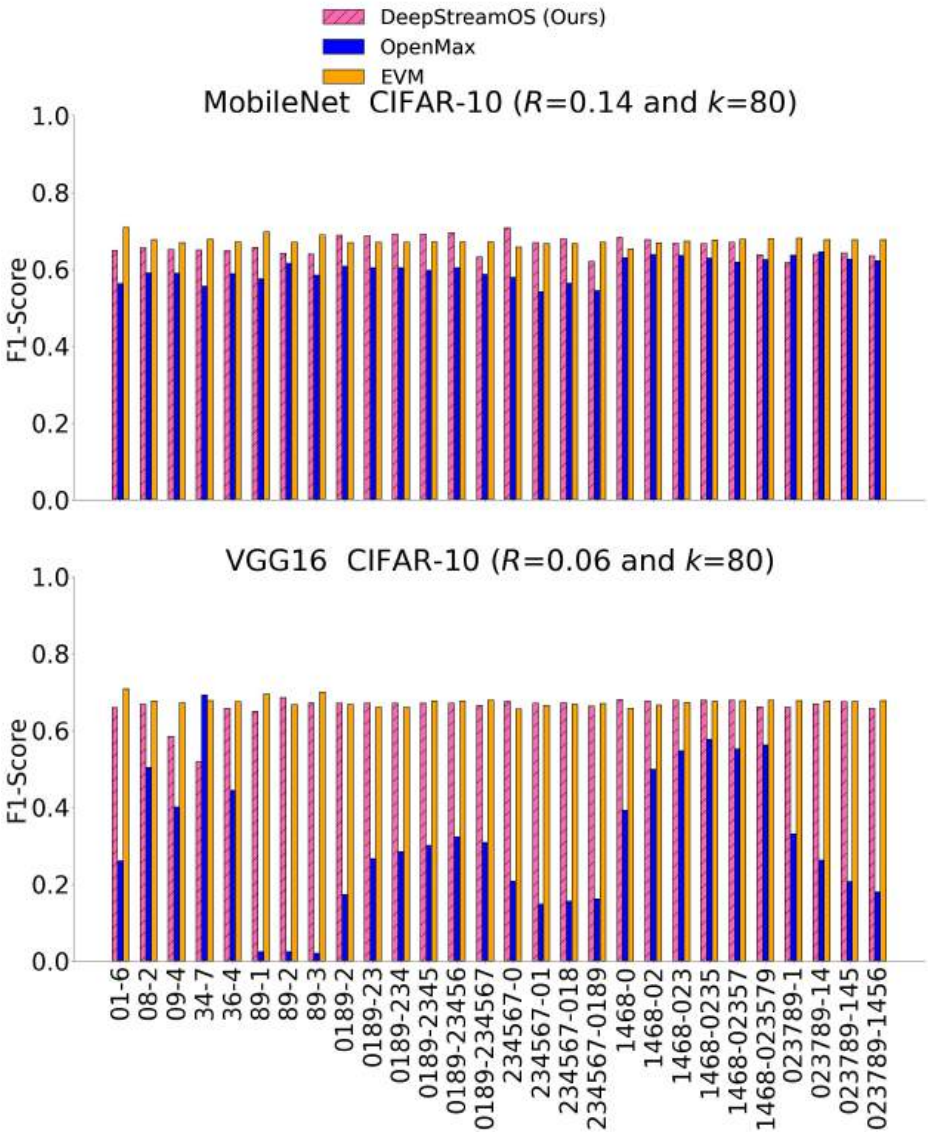


FIGURE 4.3: F1-Scores for CIFAR-10 Class data combinations.

TABLE 4.4: CIFAR-10 Category Combinations (Where Data ID represents *known classes-unknown classes*, T=Transport, A=Animal)

Data ID	Category Combination	Data ID	Category Combination
01-6	TT-A	234567-0	AAAAAA-T
08-2	TT-A	234567-01	AAAAAA-TT
09-4	TT-A	234567-018	AAAAAA-TTT
34-7	AA-A	234567-0189	AAAAAA-TTTT
36-4	AA-A	1468-0	TAAT-T
89-1	TT-T	1468-02	TAAT-TA
89-2	TT-A	1468-023	TAAT-TAA
89-3	TT-A	1468-0235	TAAT-TAAA
0189-2	TTTT-A	1468-02357	TAAT-TAAAA
0189-23	TTTT-AA	1468-023579	TAAT-TAAAAT
0189-234	TTTT-AAA	023789-1	TAAATT-T
0189-2345	TTTT-AAAA	023789-14	TAAATT-TA
0189-23456	TTTT-AAAAA	023789-145	TAAATT-TAA
0189-234567	TTTT-AAAAAA	023789-1456	TAAATT-TAAA

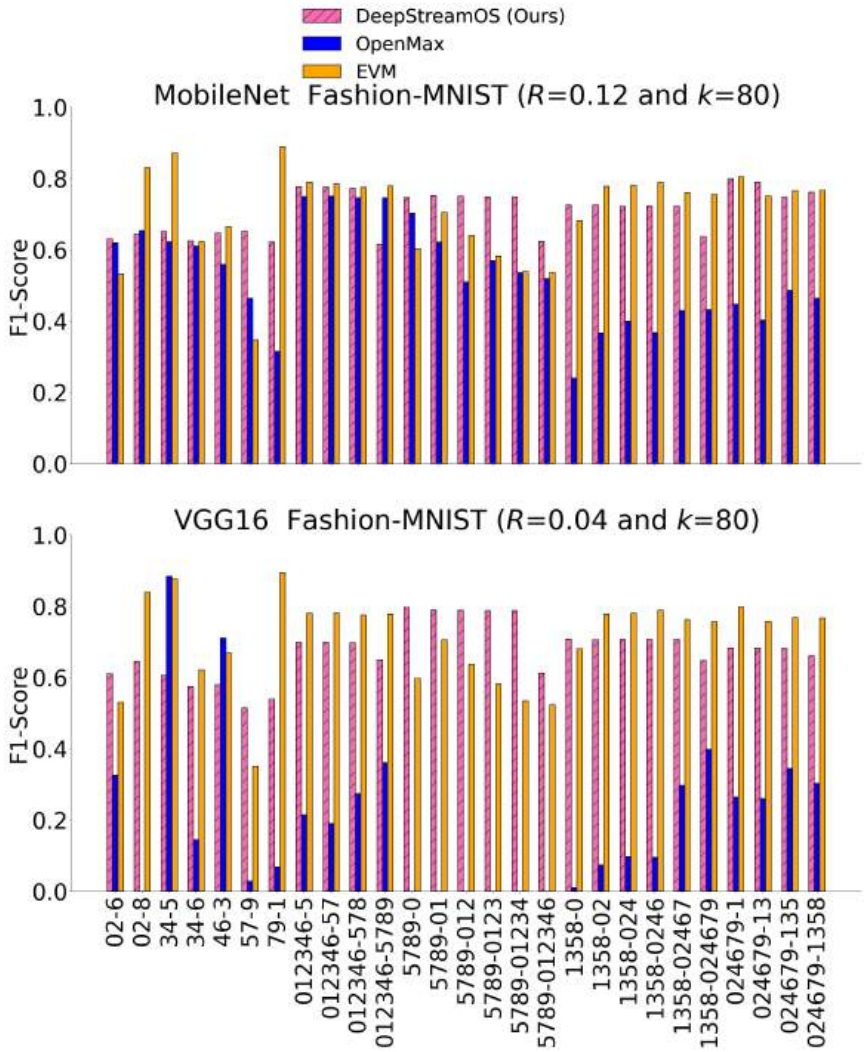


FIGURE 4.4: F1-Scores for Fashion-MNIST Class data combinations.

TABLE 4.5: Fashion-MNIST Category Combinations (Where Data ID represents *known classes-unknown classes*, C=Clothing, F=Footwear)

Data ID	Category Combination	Data ID	Category Combination
02-6	CC-C	5789-0123	FFFF-CCCC
02-8	CC-F	5789-01234	FFFF-CCCCC
34-5	CC-F	5789-012346	FFFF-CCCCCC
34-6	CC-F	1358-0	CCFF-C
46-3	CC-C	1358-02	CCFF-CC
57-9	CC-C	1358-024	CCFF-CCC
79-1	FF-F	1358-0246	CCFF-CCCC
012346-5	CCCCCC-F	1358-02467	CCFF-CCCCC
012346-57	CCCCCC-FF	1358-024679	CCFF-CCCCCC
012346-578	CCCCCC-FFF	024679-1	CCCCFF-C
012346-5789	CCCCCC-FFFF	024679-13	CCCCFF-CC
5789-0	FFFF-C	024679-135	CCCCFF-CCF
5789-01	FFFF-CC	024679-1358	CCCCFF-CCFF
5789-012	FFFF-CCC		

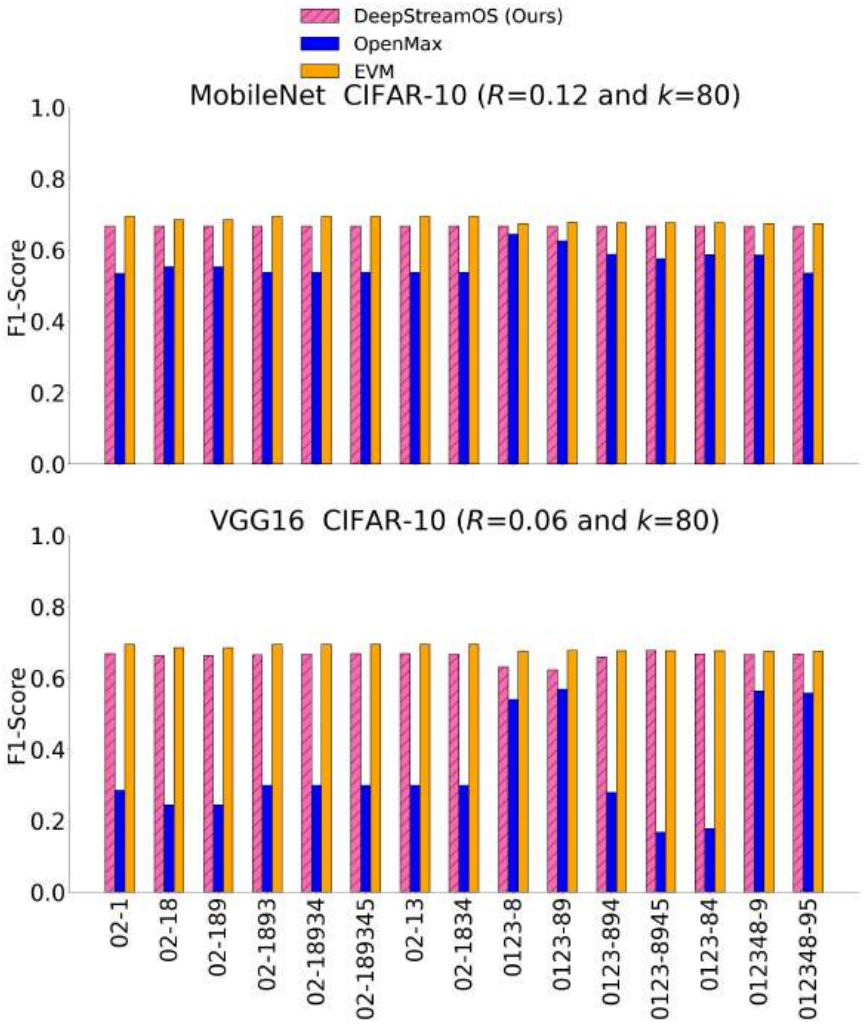


FIGURE 4.5: F1-Scores for CIFAR-10 Sub-Class data combinations.

TABLE 4.6: Sub-Class Category Combinations (Where Data ID represents *known classes-unknown classes*, T=Transport, A=Animal)

Data ID	Category Combination	Data ID	Category Combination
02-1	TA-T	0123-8	TTAA-T
02-18	TA-TT	0123-89	TTAA-TT
02-189	TA-TTT	0123-894	TTAA-TTA
02-1893	TA-TTTT	0123-8945	TTAA-TTAA
02-18934	TA-TTTTT	0123-84	TTAA-TA
02-189345	TA-TTTTTT	012348-9	TTAAAT-T
02-13	TA-TA	012348-95	TTAAAT-TA
02-1834	TA-TTAA		

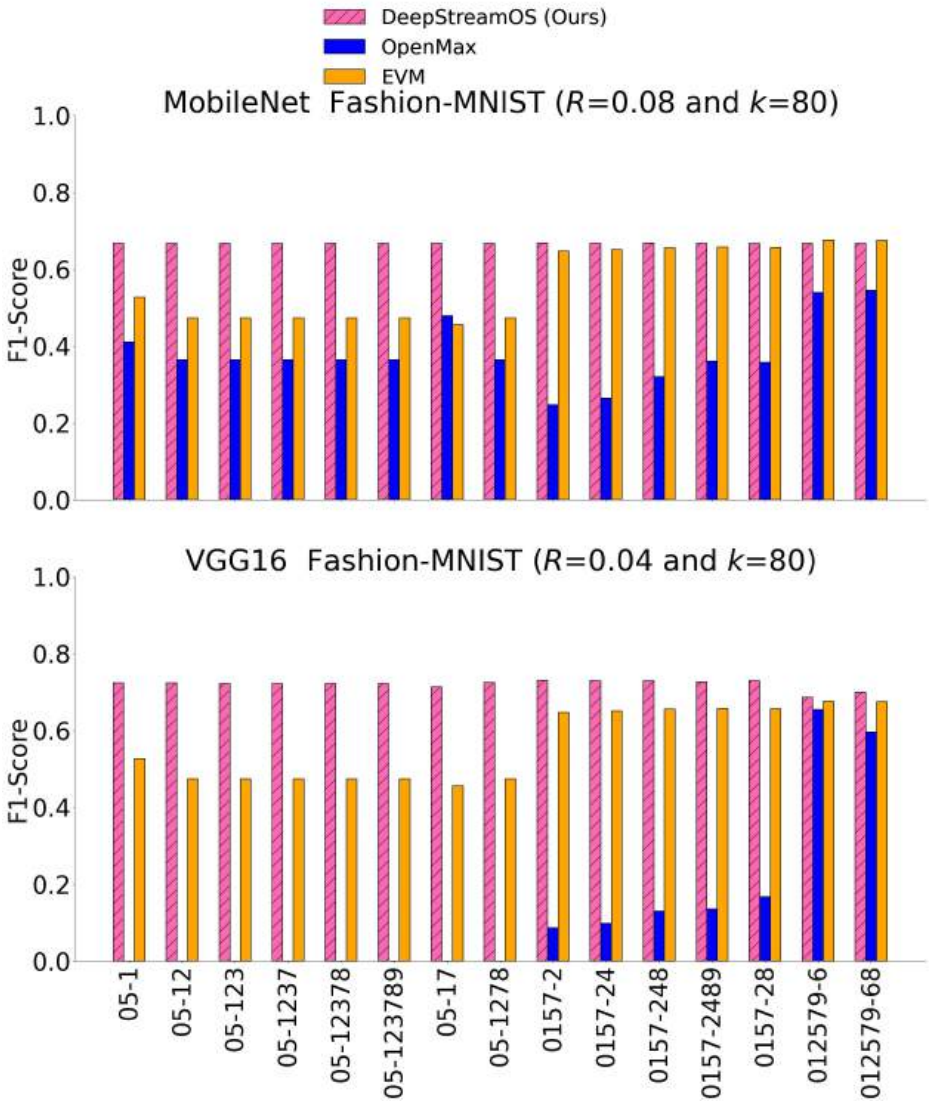


FIGURE 4.6: F1-Scores for Fashion-MNIST Sub-Class data combinations.

TABLE 4.7: Fashion-MNIST Sub-Class Category Combinations(Where Data ID represents *known classes-unknown classes*, C=Clothing, F=Footwear)

Data ID	Category Combination	Data ID	Category Combination
05-1	CF-C	0157-2	CCFF-C
05-12	CF-CC	0157-24	CCFF-CC
05-123	CF-CCC	0157-248	CCFF-CCF
05-1237	CF-CCCF	0157-2489	CCFF-CCFF
05-12378	CF-CCCFF	0157-28	CCFF-CF
05-123789	CF-CCCFFF	012579-6	CCCFFF-C
05-17	CF-CF	012579-68	CCCFFF-CF
05-1278	CF-CCFF		

DeepStreamOS consistently performs well on sub-class data combinations, however data is limited. *DeepStreamOS* outperforms OpenMax in all scenarios and outperforms EVM in scenarios involving the Fashion-MNIST dataset. The initial CNN accuracies are included in Appendix B.1. The CNN classifier accuracy could affect *DeepStreamOS* as the training of the MCOD clusterers is dependent on how many correct training instances are provided from the CNN. Appendix B.2 investigates this.

Table 4.9 shows the average time to process an instance in milliseconds. The timings are performed on a subset of at least 50% of data combinations within each category in Table 4.8. The average speed of detection for *DeepStreamOS* is 6ms per instance. This is the time per instance, measured in batches of 100 instances. It is the duration it takes to extract the activations, calculate the JS-Divergence between each layer and process them through the MCOD outlier detection algorithm to produce an inlier/outlier result. The average time taken for OpenMax and EVM to calculate the outcome for each instance is 29ms and 599ms, respectively. *DeepStreamOS* is 5 times faster than OpenMax and 100 times faster than EVM.

For OpenMax and EVM, the more known classes there are, the longer it takes to process an instance. However, for *DeepStreamOS*, timings remain consistent, indicating that *DeepStreamOS* is more suitable for up-scaling than OpenMax or EVM.

TABLE 4.8: Average and standard deviation (SD) for dataset F1-Scores

			F1-Score		
DNN	Data Type	Dataset	DeepStreamOS (Ours)	OpenMax	EVM
MobileNet	Class	CIFAR-10	0.660 (0.024)	0.600 (0.029)	**0.674 (0.011)**
MobileNet	Class	Fashion-MNIST	**0.709 (0.061)**	0.531 (0.142)	**0.709 (0.124)**
MobileNet	Sub-Class	CIFAR-10	0.667 (0.000)	0.564 (0.035)	**0.684 (0.009)**
MobileNet	Sub-Class	Fashion-MNIST	**0.667 (0.000)**	0.381 (0.085)	0.562 (0.095)
VGG16	Class	CIFAR-10	0.661 (0.033)	0.314 (0.181)	**0.674 (0.012)**
VGG16	Class	Fashion-MNIST	0.677 (0.076)	0.198 (0.221)	**0.708 (0.125)**
VGG16	Sub-Class	CIFAR-10	0.662 (0.015)	0.342 (0.141)	**0.684 (0.009)**
VGG16	Sub-Class	Fashion-MNIST	**0.720 (0.095)**	0.124 (0.212)	0.562 (0.095)

TABLE 4.9: Average and standard deviation (SD) for time per instance

			Time per Instance (ms)		
DNN	Data Type	Dataset	DeepStreamOS (Ours)	OpenMax	EVM
MobileNet	Class	CIFAR-10	**10.5 (4.6)**	21.3 (5.6)	632.8 (279.5)
MobileNet	Class	Fashion-MNIST	**6.6 (3.5)**	33.6 (10.8)	1049.0 (750.5)
MobileNet	Sub-Class	CIFAR-10	**4.6 (1.2)**	15.1 (0.3)	520.7 (34.3)
MobileNet	Sub-Class	Fashion-MNIST	**4.9 (1.5)**	18.2 (0.3)	176.2 (12.1)
VGG16	Class	CIFAR-10	**5.8 (3.2)**	45.4 (11.4)	1087.5 (532.6)
VGG16	Class	Fashion-MNIST	**5.3 (3.4)**	43.9 (11.0)	669.2 (233.9)
VGG16	Sub-Class	CIFAR-10	**3.3 (2.5)**	24.6 (2.5)	436.8 (19.7)
VGG16	Sub-Class	Fashion-MNIST	**3.3 (0.8)**	27.8 (2.3)	216.9 (15.6)

4.5 Summary

In this chapter, we proposed a novel method for outlier detection using DNN activations. This method, called *DeepStreamOS*, extracts the DNN activations from each hidden layer of the DNN and uses the statistical comparison method of JS-Divergence to obtain a value indicating the difference between two consecutive layers. This is obtained for each set of neighbouring layers, providing a set of JS-Divergence values per instance, termed an activation classification footprint. The footprints are then provided to the streaming analysis aspect, which consists of a number of MCOD clusterers (one for each known class). The JS-Divergence values for the instance is provided to the MCOD clusterer that the DNN predicted. The instance is examined to ascertain if it is an outlier. If it is an outlier, it is considered an unknown instance. If it is an inlier it is considered a known member of that class.

To examine the proposed method, a set of experiments with two benchmark image datasets were conducted. Data class combinations were selected for the system to be trained on and other classes were withheld in order to be applied as outliers. Firstly at the class level, then at the sub-class level. At the class level, novel class labels arose. At the sub-class level, no new class labels arose, but novel sub- classes were applied, thus affecting the input data of known classes. Data combinations of known and unknown classes were applied to two DNNs with a varying number of hidden layers.

The results showed that the overall average F1-Score of *DeepStreamOS* exceeds EVM by 2.1% and OpenMax by 29.6%. *DeepStreamOS* outperformed the state-of-the-art solutions, OpenMax in 97% of the data combinations, and EVM in 43% of

combinations. With further investigation, it was discovered that the performance of *DeepStreamOS* is compromised when there are only two known classes or when the number of unknown classes are increased. Thus, performance improves when there are a larger amount of known classes and a lower amount of unknown classes. This scenario is common in real-world applications such as in medical image analysis, where a system would be pretrained on large amounts of known data and a few novel changes would occur, i.e. chest x-rays where a new respiratory disease is emerging. *DeepStreamOS* performs more accurately on networks with an increased number of hidden layers and a mixture of categories, rather than unknown instances from the same categories. Using the Wilcoxon Signed-Rank test, the difference between the F1-Score of the compared methods is statistically significant. With regards the overall average F1-Score, *DeepStreamOS* exceeds OpenMax by 30% and EVM by 2%. With respect to the average speed of detection for an instance, *DeepStreamOS* by far exceeds the compared methods, being five times faster than OpenMax and one hundred times faster than EVM. Compared to EVT-based methods or deep learning-based methods, *DeepStreamOS* provides fast inference with only two parameters that require tuning for the MCOD outlier detection. This is comparable to the number of parameters that require tuning in systems only involving EVT and considerably less than if a DNN such as a GAN or autoencoder is involved in the processing.

The results indicate that DNN activation can be used with streaming machine learning methods to detect outliers. JS-Divergence reduces the DNN activations into a small amount of values per instance (the activation classification footprint). To improve the detection performance, in the next chapter, we propose the *AdaDeepStream* method with a modified JS-divergence activation reduction method with more values per instance, and an alternative activation reduction method based on content-based image retrieval. We also move away from outlier detection, towards an accuracy volatility-based discrepancy detection method. We detect concept evolution whereby we apply novel classes in drift patterns and extend further to include DNN adaptation in order to provide a complete detection and adaptation solution, thus satisfying the third objective from Section 1.5.

Chapter 5

AdaDeepStream: Streaming DNN Adaptation to Concept Evolution

In the previous chapter, we described a novel method called *DeepStreamOS* which provides outlier detection for novel classes and sub-classes, addressing the second objective as defined in Section 1.5 of this thesis. This chapter proposes the *AdaDeepStream* concept evolution detection and DNN adaptation method. It progresses the work in this thesis into concept evolution discrepancy detection, further investigating the activation reduction, with two new methods developed and analysed, and adding DNN adaptation, satisfying objective three: **Design, develop and evaluate a concept evolution discrepancy detection and DNN adaptation method**. The work presented in this chapter is published as a paper titled "AdaDeepStream: Streaming Adaptation to Concept Evolution in Deep Neural Networks" in the "Springer Applied Intelligence" journal [28].
The code is available at https://github.com/chambai/AdaDeepStream.

5.1 Introduction

DNNs are widely used and have achieved state-of-the-art performance in static data classification tasks [105, 180]. However, data evolves in real-world scenarios and standard DNNs are not responsive to changing data. DNNs only recognise classes they are trained on. Therefore, novel classes are attributed to known labels from the training data. This will result in incorrectly classified instances. Non-responsiveness to changing data could be dangerous in safety-critical applications such as autonomous vehicles [143] and medical sensor analysis [3]. For instance,

deep neural networks have been used widely to develop autonomous vehicles and various safety incidents have been reported in the media such as Google's self driving car hitting a bus [106] and a Tesla driver's fatal crash [199]. Detection and adaptation of novel classes in these circumstances is essential. In this chapter we focus on the key challenge of detecting novel classes emerging in streaming images and CNN adaptation to this.

In dynamically changing and non-stationary environments, data drift can manifest when out of distribution instances occur. When novel classes appear in a data stream in accumulated instances, this data drift is called concept evolution [43]. When there is a distribution change within the existing classes only and no new classes arise, this data drift is called concept drift [57]. This chapter focuses on concept evolution only.

Detecting and adapting to concept evolution in CNNs can be problematic. In the streaming environment there is no prior knowledge of the novel classes to assist in the initial training process [209]. Image data is high dimensional which makes it harder to detect concept evolution than for lower dimensional datasets [195]. In CNN adaptation, high data dimensionality contributes to the adaptation latency being large [205]. DNN adaptation can cause catastrophic forgetting (where originally known classes are forgotten in the presence of new classes) [122]. DNNs require balanced classes for training which are not available in streaming scenarios. DNNs require a larger amount of data for training as compared to other types of machine learning models. To achieve online CNN adaptation, solutions often require prior selection of specialised DNN architectures, loss functions or knowledge distillation [202, 156, 110, 8]. This means that novel class detection and adaptation cannot be retrospectively applied without retraining the CNN. This is particularly an issue in CNNs that take a long time to train. The requirement of online adaptation may not always be realised at the time of original implementation of the CNN system. If such a system is not in place, metrics would need to be manually monitored, data would need to be collected, labelled and another model statically trained before being updated on the system, which may not be achievable when the system is online. Therefore, easily implementable solutions to upgrade existing trained CNNs are required. At the very least, an image recognition system should be able to

detect and adapt to novel classes in a data stream. For an image recognition system involving a DNN, the two fields of Concept Evolution and Online Class Incremental (OCI) are involved. OCI is a subset of the Online Continuous Learning (OCL) field concerning novel classes only. OCI typically trains a newly initialised DNN in an incremental manner, applying all instances of one class at a time [122]. Research in this field focuses on accumulating and preserving knowledge without forgetting any previous data [122], true-labelled samples of the classes are often used, which results in artificially high performance. In a real-world scenario, even partially labelling a data stream using humans can be expensive [59] and impractical due to the need for domain experts and manual labelling. This is in contrast to the Concept Evolution field where focus is upon the changing data, taking into account only some previous data and using minimal true-labelled samples. The Concept Evolution and OCI fields require bringing together as they offer complimentary views. We apply concept evolution patterns of abrupt, gradual, incremental and reoccurring as shown in Figure 1.4.

Existing approaches for CNN adaptation to concept evolution are limited as the focus has been on lower dimensional data [56]. To handle high dimensional data, the data is transformed into a different representation to improve the separation of the classes [85, 191, 60, 195]. Clustering is commonly used in novel class detection [101, 195, 60, 42, 73, 22, 207]. It is an implicit method of drift detection. The drift is monitored over a number of instances before drift is declared. This is in contrast to explicit drift detection where the change is detected and immediately reported. Implicit methods of drift detection result in a delay of concept evolution detection. In data stream classification, there are approaches for novel class detection in images where DNNs are used in the detection process [60, 81, 111, 5]. However, considerably less where a CNN is adapted to the novel classes [101]. This means that CNN adaptation has not been studied in depth in the concept evolution field. On the contrary, the OCI field has a large number of CNN adaptation techniques [156, 110, 31, 76, 8]. However, they have not been applied to the more dynamic concept evolution setting. Only one paper, [101] started to address this by unifying OCL with concept drift and concept evolution adaptation [101]; however, data drift patterns were not applied. Our own previous work uses the activations from within a DNN

with streaming clustering and detects random outliers [26], but had limitations with memory [27]. An autoencoder was employed as the activation reduction method, requiring a large amount of memory and computational power during the offline training phase. Thus, a new solution is required that addresses concept evolution detection and CNN adaptation to images with the following attributes: (1) Transformation of the image data into a different representation. (2) Explicit concept evolution detection. (3) A solution that can be applied to existing CNNs with no need to retrain the CNN. (4) Analysis with respect to concept evolution patterns.

Our system (*AdaDeepStream*) aims to provide a wrapper whereby pretrained standard CNNs can be enabled to explicitly detect and adapt to concept evolution in images. The concept evolution detection and CNN adaptation is facilitated by the use of activations from within the deep neural network, applied to streaming machine learning models. Our system detects change in an unsupervised manner by comparing the predictions of the CNN with predictions from a streaming classifier using the internal activations of the CNN. Only instances that are detected at the beginning of the change are true-labelled to minimise human interaction. The aim is to adapt a CNN within seconds. This research focuses on image classification via the widely used VGG16 Convolutional Neural Network [173]. Datasets CIFAR-10, CIFAR-100 [103] and Fashion-MNIST [198] are used. Concept evolution patterns are produced by withholding classes from the dataset during training, then applying unseen instances of known and withheld classes during testing. We analyse unseen instances by utilising activations from the hidden layers within the deep neural network. We reduce the activations using two methods (1) an extension of our Jensen-Shannon Divergence (JS-Divergence) as used in [26], altered to also calculate JS-Divergence between each layer and the final layer. This is referred to as JSDL for the remainder of this chapter. (2) a modified Content-Based Image Retrieval (CBIR) descriptor generation method [176], referred to as DS-CBIR for the remainder of this chapter. We apply the activations to a Hoeffding Adaptive Tree [16] streaming classifier. The difference between the Hoeffding adaptive classifier and the CNN is used to detect concept evolution via Drift Detection Method (DDM) [58]. Once concept evolution is detected, the CNN is adapted via our method, termed DSAdapt. We substitute our DNN adaptation method with four leading methods from the OCI

field: Incremental Classifier and Representation Learning (iCARL) [156], Learning without Forgetting (LwF) [110], Experience Replay (ER) [31, 76] and Maximally Interfered Retrieval (MIR) [8] augmented with Review Trick (RV) [25]. We perform an extensive empirical study, comparing these and our DSAdapt method based on accuracy, speed of inference, and speed of adaptation. Our entire system, *AdaDeepStream*, is an offline training and online inference method. Its effectiveness is shown in comparison to Reactive Subspace Buffer (RSB) [101] and CNN based Prototype Ensemble (CPE) [195] also through accuracy, speed of inference and speed of adaptation.

The CBIR technique [176] is modified to remove the threshold and further reduce the activations by splitting them into equal sections and averaging each section. To the best of our knowledge, applying CBIR to concept evolution detection is unique. JSDL uses the JS-Divergence statistical difference measure in a novel way between layers of a deep neural network. To the best of our knowledge, apart from our own previous work in [27, 26]; using the reduced activations for concept evolution detection is unique. Our CNN adaptation (DS-Adapt) is novel. Reduced activations provide a buffer of training and true-labelled instances, assisting in addressing class imbalance and catastrophic forgetting. To the best of our knowledge, using activations with streaming machine learning models for CNN adaptation is novel. A summary of the contributions of this chapter is as follows:

1. Heuristics for activation reduction of deep neural networks via DS-CBIR and JSDL to apply to concept evolution detection.

2. Concept evolution detection using neural network activations and streaming machine learning models.

3. CNN adaptation involving neural network activations and streaming machine learning models.

4. Analysis of OCI CNN adaptation techniques in a concept evolution setting.

This chapter is organised as follows: In Section 5.2, we present a description of the system that includes formalisation and implementation details of the *AdaDeepStream* components and methodology. In the experimental study in Section 5.3 we

specify the specific experimental setup details for this method. In Section 5.4, we evaluate *AdaDeepStream* with four other CNN adaptation methods and two combined novel class detection and CNN adaptation solution.

5.2 AdaDeepStream System Description

This section details our *AdaDeepStream* system components and their interactions via descriptions and algorithms. We present our two activation reduction methods (JSDL and DS-CBIR), our drift detection method, our CNN adaptation method (DS-Adapt), and how the application of the four substituted CNN adaptation methods is achieved.

Firstly, some fundamental concepts are given. For the definition of a data stream, let $D = \{(x_i, y_i)\}_1^\infty$ where (x_i, y_i) is an instance that has arrived at timestep t. y_i is the true class label of the instance. Concept evolution are new classes that emerge during inference. They are not present in the initial training data. For data stream classification with novel class detection: Let D_{init} be a training set of M examples: $D_{\text{init}} = \{(x_i, y_i)\}_1^M$ where the class labels are: $y_i \in Y = \{1, 2, \ldots, L\}$ and Y are the known class labels. An initial model is built with: $f : X \to Y$. The model f, is then used to predict the class labels of incoming instances if it belongs to a known class, or flag up a change. Each change detection is considered a possible new class in that window. The model is then adapted to include the novel class. In the following algorithm descriptions, index notation is used. For instance, x_{ih} is the ith instance in hidden layer h. Table 5.1 lists the commonly used symbols. For brevity in the descriptions, the Hoeffding Adaptive Tree streaming activation classifier is referred to as HAT, JS-Divergence is referred to as JSD and the SAM-kNN streaming clusterer is referred to as SAM.

Figure 5.1 shows the proposed *AdaDeepStream* system. The prerequisites for the system are: (1) A trained CNN on which the detection and adaptation is being applied, and (2) the data instances on which the deep neural network is trained. *AdaDeepStream* is an offline-online system in that it is trained offline and processes unseen instances online. To train *AdaDeepStream*, the training instances are presented to the CNN. The activations of each training instance are extracted, then

TABLE 5.1: Summary of main symbols

Symbol	Description	Symbol	Description
A	Reduced activations	K	SAM class predictions
B	Change detection window buffer	L	Number of classes
b	Convolutional block	N	Number of convolutional blocks
C	CNN	P	CNN predicted class labels
C_{adapt}	Adapted CNN	R	Drift detector outputs
D	Data stream	S	HAT class predictions
E	Class buffer	s	Sample from class buffer
G	Number of windows in window buffer	U	Window buffer
H	Number of hidden layers	V	Max num activation vals per layer
h	Hidden layer number	W	Number of change detection groups
i	Instance number	w	Window number
J	Activations	x	Instance data
J_{max}	Max activation vals per channel	Y	True class labels
J_{conv}	Activation values of conv layer		

the activation data proceeds through 3 stages: (1) Activation reduction, (2) drift detection setup and (3) adaptation setup. Activation reduction is achieved via JSDL (Section 5.2.1) or via DS-CBIR (Section 5.2.2). The HAT and SAM are trained with reduced activation data: $A_{\text{init}} = \{(a_i, y_i)\}_1^M$. Where A_{init} is a training set of M samples of activation data, where a_i is an instance of reduced activation data. The class buffer is initialised with image and activation data: $E_{\text{init}} = \{(x_i, a_i, y_i)\}_1^M$. Where E_{init} is a training set of M samples of image and activation data. At inference time, a window of image data, D_w is applied. The reduced activations are extracted via JSDL or DS-CBIR to give A_w. Each instance in A_w is classified by the HAT to give a window of HAT predictions, S_w. The CNN predictions, P_w are compared with S_w. Instances where the HAT and CNN predictions match are assigned as 0. Instances where the HAT and CNN prediction do not match are assigned as 1. These are provided to the DDM drift detector. If the drift detector identifies changes, the window data, (A_w, D_w) is passed to the adaptation phase. In Figure 5.1, units involving the adaptation phase are shown in orange (that is, the HAT, Drift Change Instances, Previous Window Buffer, SAM, Class Buffer, Train Classifier Layers and the classification layers of the CNN). The window that change was detected in is true labelled. The SAM and the class buffer are updated with the true labelled window data. Windows prior to the drift detection are collected from the previous window buffer. Samples are taken from the class buffer for each class detected in the previous

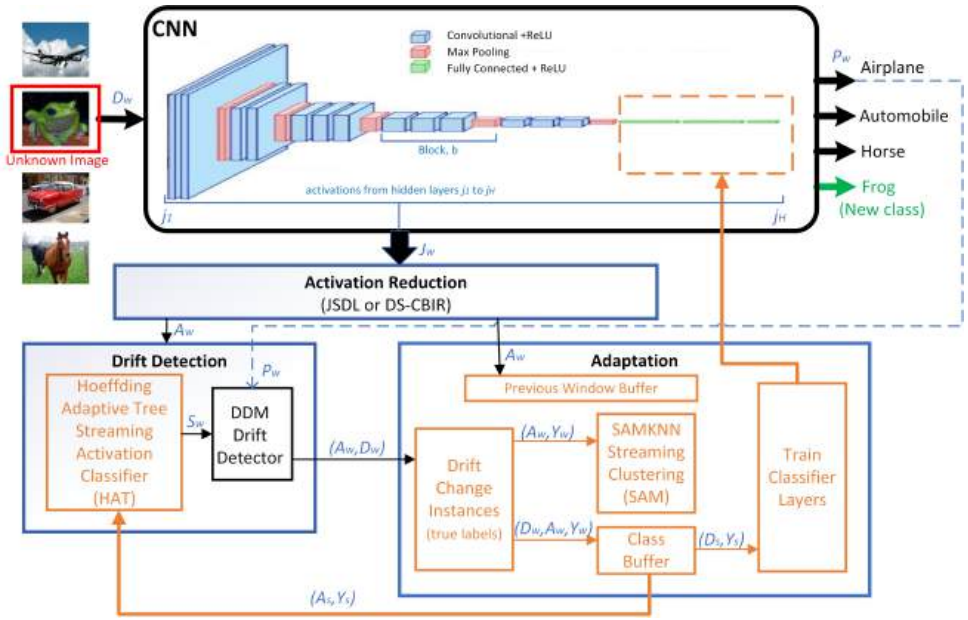


FIGURE 5.1: *AdaDeepStream* System overview. An unknown image of a frog is presented to the CNN, the activations are extracted, drift is detected and the CNN is adapted to recognise the new class. CNN image adapted from [52].

window buffer to assist in mitigating catastrophic forgetting. The sampled image data (D_s, Ys) is applied to the adaptation of the CNN. Only the CNN classification layers are trained as shown by the dashed orange box in the CNN. The representations learned in the early layers provide a generalisation that remains even when new data is learned [204]. If the CNN is adapting when new windows, D_w are arriving, the instance labels stored in the previous window use the prediction from SAM instead of from the CNN as SAM has already been updated with the novel classes. While the CNN is adapted, the HAT is adapted with the activation instances from the class buffer (A_s, Y_s) to recognise i.e. the Frog class. From this point on the frog is a recognised class and does not trigger the drift detector.

5.2.1 Activation Reduction - JSDL

We calculate the JS-Divergence between each pair of consecutive hidden layers and between each layer and the final hidden layer. Details of JS-Divergence statistical difference measure can be found in Section 3.4.4. The JS-Divergence calculates a normalised score that is symmetrical. The JS-Divergence has been selected as the statistical difference measure as it, or the KL-Divergence has been used in conjunction with DNN activations [139, 188].

Algorithm 4 JSDL

Input: One window of image data, D_w
Input: CNN, C expressed in hidden layers: $h \in C = \{1, 2, \ldots, H\}$
Output: A_w: One window of reduced activations

1: let V be the max number of activations in all layers of C
2: let H be the number of the final hidden layer
3: **for** $x_i \in D_w$ **do**
4: let layer number, $h = 0$
5: **for** $h \in C$ **do**
6: $J_{ih} \leftarrow$ getActivations(h) ▷ get activations for current instance and hidden layer
7: **if** $h > 1$ **then**
8: $J_{ih} \leftarrow$ **flatten**(J_{ih}) ▷ convert activations into 1-D array
9: $J_{ih} \leftarrow$ **pad**(J_{ih}, J_{in-1}, V) ▷ pad adjacent activation layers to the same size
10: $J_{ih} \leftarrow$**jsdiverge**(J_{ih}, J_{ih-1}) ▷ calculate JSD for adjacent layers
11: $A_w \leftarrow$ **jsdiverge**(J_{ih}, J_{iH}) ▷ calculate JSD for current and last layer
12: **end if**
13: h = h + 1
14: **end for**
15: **end for**
16: Return A_w

Other statistical measures of Kolmogorov-Smirnov and cosine similarity were experimented with. However, JS-Divergence provided superior results. To compare layers using JS-Divergence, the number of activations utilised from each neighbouring layer needs to be the same. The largest layer size in the CNN is selected and if the layer sizes are smaller, they are padded with zero's, then the JS-Divergence is calculated between the neighbouring layers. This yields one value between each activation layer per data instance. The algorithm for our JSDL implementation is presented in Algorithm 4. In the overall *AdaDeepStream* system (Algorithm 6), JSDL would be called in Line 2 via the **reduceActivations** method.

For previously unseen instances arriving at the deep neural network, the activations for one layer at a time is extracted via **getActivations** (Line 6). The JS-Divergence is calculated for two neighbouring layers; therefore, the first layer is skipped as shown in Line 7. The layers are flattened into a 1-D array via **flatten** in (Line 8) and padded to the size of the largest layer in the network in **pad** (Line 9) and the JS-Divergence is calculated between the neighbouring layers in **jsdiverge** (Line 10). This JS-Divergence calculation process is repeated for the current layer and the final layer in **jsdiverge** (Line 11). This is repeated for all layers. The set of reduced activations (A_w) for the window is returned.

Algorithm 5 DS-CBIR

Input: One window of image data, D_w
Input: CNN, C expressed in convolutional blocks: $b \in C = \{1, 2, \ldots, N\}$
Output: A_w: One window of reduced activations

1: **for** $x_i \in D_w$ **do**
2: **for** $b \in C$ **do**
3: let b_p be the pooling layer of block b
4: **for** each channel, c of b_p **do**
5: $J_{\max} \leftarrow$ **max**(b_p) ▷ get max channel value in pooling layer
6: $J_{\text{conv}} \leftarrow$ **getConvLayer**(b) ▷ get conv layer 1 values
7: **end for**
8: $J_b \leftarrow J_{\max} + J_{\text{conv}}$
9: **end for**
10: $J_H \leftarrow$ **getActivations**(H) ▷ get final hidden layer activations
11: $J_b \leftarrow$ **sectionAvg**$(J_b, 16)$ ▷ get the average value for 16 sections
12: $J_H \leftarrow$ **sectionAvg**$(J_H, 32)$ ▷ get the average value for 32 sections
13: $A_w \leftarrow J_b + J_H$
14: **end for**

5.2.2 Activation Reduction - DS-CBIR

A CBIR method [176] is intended for content-based image retrieval. It creates descriptors for images using deep neural networks. It is based on obtaining neural codes from fully connected layers activations, using the information contained in convolutional layers. However, the number of neurons in the convolutional part is large and most of them do not contribute significantly to the final classification. Therefore the most significant neuron activations only are extracted in order to provide extra information about the image such as background textures or colour distribution that is present in the convolutional layers [176]. We have modified this CBIR method for use within *AdaDeepStream* to extract the most useful activations from the network such that we can utilise it in our streaming classifier. The algorithm for our CBIR implementation is presented in Algorithm 5, we have called this DS-CBIR. This is invoked in the overall *AdaDeepStream* system (Algorithm 6) Line 2, via the **reduceActivations** method.

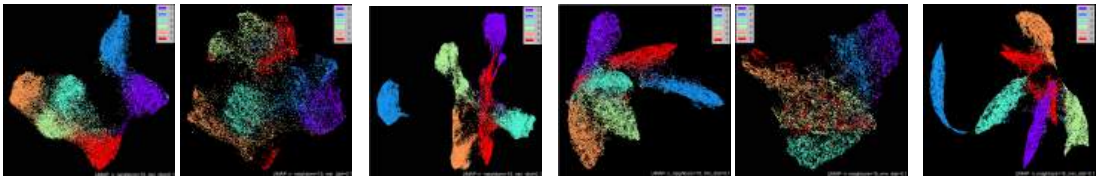
For previously unseen instances arriving at the deep neural network, the activations for one block at a time is extracted, where a block in a CNN consists of convolutional layers, then a pooling layer. For each channel in the block, the maximum activations for the pooling layer is extracted via **max** (Line 5). Corresponding

values are obtained from the first convolutional layer in the block via the **getConvLayer** method (Line 6). This is repeated for each block in the network. Due to the constraint on the number of input features the streaming classifier can accept, we only extract the values from the first convolutional layer in each block. The original CBIR descriptor generation paper [176] used a threshold to save on computing time, reasoning that as ReLU (Rectified Linear Unit) activation functions were used, then processing under an activation threshold of 0.5 was not advantageous. A description of this method can be found in Section 3.4.5. As our system is designed to be flexible for different types of CNNs, we removed this threshold which, on our system, did not incur a significant increase in computing time but improved the clustering of the activations. Lines 2 to 10 is the original CBIR algorithm [176]. Additionally, the combined output of the max pooling layer activations and the convolutional layer activations from each block are reduced further via the **sectionAvg** method (Line 11) where set J_b is split into 16 equal parts and the average for each part is calculated. The activations for the last hidden layer J_H are extracted and these are also reduced into 32 values in the same way (Line 12). These reduced sets are combined (Line 13) and returned as the set of reduced activations (activation classification footprints) for the window. This is required so that the number of values presented to the streaming classifiers are within the range of the number of input features accepted by the streaming classifier.

To achieve DS-CBIR, the convolutional blocks need to be identified for the CNN. This is done automatically by *AdaDeepStream*, but may need manually adjusting, whereas JSDL simply uses all of the networks layers. Figure 5.2 shows UMAP [125] representations of the reduced activations. From this we can see that Fashion-MNIST activation data potentially has superior clustering and separation of the classes than the CIFAR-10 and CIFAR-100 data. Both of these activation reduction methods provide activation classification footprints as discussed in Section 2.6 into our concept evolution detection method, described in the following section.

5.2.3 Concept Evolution Detection

The well known drift detection method of DDM [58] is used within the concept evolution detection mechanism for drift detection. The drift detection method in



(a) CIFAR-10 JSDL (b) CIFAR-100 JSDL (c) Fashion-MNIST JSDL (d) CIFAR-10 DS-CBIR (e) CIFAR-100 DS-CBIR (f) Fashion-MNIST DS-CBIR

FIGURE 5.2: UMAP representations of reduced activation training data for six classes in the CIFAR-10, CIFAR-100 and Fashion-MNIST datasets

AdaDeepStream can be substituted for any other drift detection method. An empirical study was conducted with other drift detectors, for which the results are included in Appendix A. As only drift detection and not more information such as severity or region of the drift is required, error-based drift detection methods only were analysed. Statistical control-based and adaptive window-based drift detectors were selected. The statistical based drift detectors of DDM [58] and EDDM [11] outperformed the adaptive window-based drift detectors of ADWIN [17], KSWIN [152], HDDMA and HDDMW methods [54]. DDM was significantly faster than EDDM, therefore it was selected.

After the activations have been reduced, they are supplied to a Hoeffding Adaptive Tree streaming classifier which is established and well used [16]. Further information on this classifier is provided in Section 3.5.2. The Hoeffding Adaptive Tree is trained on the reduced activations obtained from the training data. Algorithm 6 shows the overall *AdaDeepStream* process at inference time. For previously unseen instances arriving in the data stream, the prediction is obtained from the CNN, **cnnPredict** (Line 3). The reduced activations are extracted and a prediction from these is obtained from the Hoeffding Adaptive Tree Classifier, **hatPredict** (Line 4). The predictions are then compared. If they match, 0 is provided to the drift detector, otherwise 1 is provided (Lines 5 to 15). The drift detector returns a 'W' if a change warning is detected, and returns a 'C' if a change is detected. This is stored for later use (Line 14). If a warning or a change is detected in the window, then the true labels for that window are obtained, **getTrueValues** (Line 17). In reality, the images in this window would be displayed to the user for them to label them with the true values. As only the windows where the change was detected are displayed to the user, there

Algorithm 6 *AdaDeepStream*

Input: Windows of image data D
Input: Pretrained CNN, C on L classes
Input: Pretrained Hoeffding Adaptive Tree on activations of L classes
Output: P: A set of CNN predicted classes for data stream

1: **for** $D_w \in D$ **do**
2: $A_w \leftarrow$ **reduceActivations**(D_w) ▷ reduce activation via JSDL or DS-CBIR
3: $P_w \leftarrow$ **cnnPredict**(D_w) ▷ get CNN predictions
4: $S_w \leftarrow$ **hatPredict**(A_w) ▷ get HAT predictions
5: **for** $x_i \in D_w$ **do**
6: let a_i be the reduced activations for x_i
7: let p_i be the CNN predicted class for x_i
8: let s_i be the HAT predicted class for x_i
9: **if** $p_i = s_i$ **then** ▷ check if CNN and HAT predictions match
10: $d_i = 0$ ▷ 0 if match
11: **else**
12: $d_i = 1$ ▷ otherwise 1
13: **end if**
14: $R_w \leftarrow$ **driftDetector**(d_i) ▷ detect drift change
15: **end for**
16: **if** 'W' or 'C' is in R_w **then** ▷ drift detector returns 'W' for warning or 'C' for change
17: $Y_w \leftarrow$ **getTrueValues**(D_w) ▷ get true values for the drift detection window
18: $Y \leftarrow Y + Y_n$ ▷ add new labels to known classes
19: $C_{\text{adapt}} \leftarrow$ **adaptDnn**(D_w, A_w, Y_w) ▷ adapt CNN via DS-Adapt or another method
20: **else**
21: Let U be the window buffer and G be the number of windows in U
22: **if** CNN adaptation in progress **then**
23: $K_w \leftarrow$ **samPredict**(A_w) ▷ use SAM predictions if CNN adapting
24: $U \leftarrow (D_w, A_w, K_w)$
25: **else**
26: $U \leftarrow (D_w, A_w, P_w)$ ▷ use CNN predictions
27: **end if**
28: **if** $G > 2$ **then** ▷ only store 2 windows
29: $U \leftarrow U - U_0$ ▷ remove oldest window
30: **end if**
31: **end if**
32: $P \leftarrow P_w$
33: **end for**

is less labelling than if all the true values are used. The next step is adaptation (Line 19). Image data D_w, reduced activation data A_w and their true labels Y_w are provided to the **adaptDNN** method. This method can be any CNN adaptation method. Our CNN adaptation algorithm, DSAdapt, is described in Algorithm 7. We substituted this adaptation with four other state-of-the-art methods. The results are shown in Section 5.4. If there is no drift detected, the previous G windows are stored in U (Lines 28 and 29). If there is CNN adaptation occurring in the background, the window is re-predicted via the SAM clusterer, **samPredict** and stored (Lines 23 and 24). As the SAM clusterer is the first to be updated with true labels, this improves the

Algorithm 7 DSAdapt

Input: One window of image data: D_w
Input: One window of reduced activation data A_w
Input: One window of true labels: Y_w
Input: CNN to be adapted, C
Input: Buffer of two previous windows, U
Output: Adapted CNN, C_{adapt}

1: let W be the total number of change detection groups in buffer, B
2: $B \leftarrow U + (D_w, A_w, Y_w)$ ▷ save the current change detection and previous two windows
3: **if** $W > 5$ **then** ▷ if there's more than 5 groups of change detection buffers
4: $\quad B \leftarrow B - B_0$ ▷ remove oldest group
5: **end if**
6: **samPartialFit**(A_w, Y_w) ▷ add true labelled data to SAM
7: $K_w \leftarrow$ **samPredict**(B)
8: Let E be the class buffer
9: $E \leftarrow E + (D_w, A_w, Y_w)$ ▷ Add true labelled data to class buffer
10: $E \leftarrow E - (D_0, A_0, Y_0)$ ▷ remove oldest instance
11: **for** class l in $(K_w + Y_w)$ **do** ▷ get 100 samples per class from buffer
12: $\quad (D_s, A_s, Y_s) \leftarrow$ **getClassBufferSample**$(l, 100)$
13: **end for**
14: $(D_s, A_s, Y_s) \leftarrow (D_s, A_s, Y_s) + B$ ▷ add change detection windows to samples
15: copy C and train with (D_s, Y_s) for 3 epochs ▷ adapt CNN
16: **hatPartialFit**(A_s, Y_s) ▷ adapt HAT
17: replace C with adapted CNN, C_{adapt}
18: repeat steps 1 to 17 on background thread

accuracy of the stored windows whilst the CNN adaptation is awaited. If there is no CNN adaptation occurring, the window is stored with the CNN predictions (Line 26). SAM-kNN is a popular Self Adjusting Memory (SAM) model for the k Nearest Neighbor (kNN) [160, 118]. It operates on a window of instances which is set to 1000 instances; therefore, the memory usage will not increase over time.

5.2.4 Adaptation

CNN adaptation is based on transfer learning with clustering, a class buffer and some memory of previous instances. Our CNN adaptation method DSAdapt is described in Algorithm 7.

When a warning of a change or a change is detected, the window of data within which it is detected is provided to the CNN adaptation method. The current window and the previous two windows are added to a buffer (Line 2). These are the change detection windows. If the buffer exceeds 5 groups of change detection windows, the first group in the buffer is removed (Line 4). This provides some 'memory'

for the CNN adaptation. True-labelled instances are supplied to the SAM-kNN clusterer [117] (Line 6). The buffer instances are re-predicted via the SAM-kNN Streaming Classifier (Line 7) to give K_w. True-labelled instances are also supplied to a class buffer that stores training data instances of the original image data and the equivalent activation data (Line 9). Each time a window is added to the class buffer, the oldest window is removed (Line 10), ensuring that the size of the class buffer does not increase. Each class in K_w and the true values for the change detection window Y_w is randomly sampled from the class buffer. One hundred instances (or if there is not enough, the maximum number of instances that are available) are randomly selected from the class buffer (Lines 11 to 13). The change detection windows, B are added to the sampled class buffer data (Line 14). The CNN is adapted for 3 epochs (Line 15). The current CNN is replaced with the adapted CNN (Line 17). Once the adaptation has been triggered, it continues adapting (Lines 1 to 17) on a background thread, using the current instance window and previous windows predicted via the SAM-kNN Streaming Clusterer (Line 18), replacing the true values Y_w with the predicted values from the clusterer K_w. In summary, to achieve a balance between avoiding catastrophic forgetting and remembering recent data only, windows pertaining to the last five change detections are stored. Only the classes found in these change detection windows are sampled from the class buffer. Therefore, not all previous classes that have ever arrived are used in the adaptation, but only more recent ones. As a true-labelled instance is added to the class buffer, the oldest instance is removed, ensuring the size of the class buffer does not increase over time. Catastrophic forgetting is partially mitigated by remembering the recent classes only. This is intentional as in the concept evolution field it is not the aim to remember all classes ever seen.

Figure 5.3 shows the adaptation in more detail where the numbers depict the following actions: 1. True label the change detection window. 2. Store the two previous windows and the change detection window in the change detection buffer. 3. If the change detection buffer contains more than 5 buffers, remove the oldest. 4. Add true-labelled change detection window image data to image buffer. 5. Remove the oldest instances from the image class buffer. 6. Add true-labelled change detection window activation data to SAMKNN. 7. Add true-labelled change detection

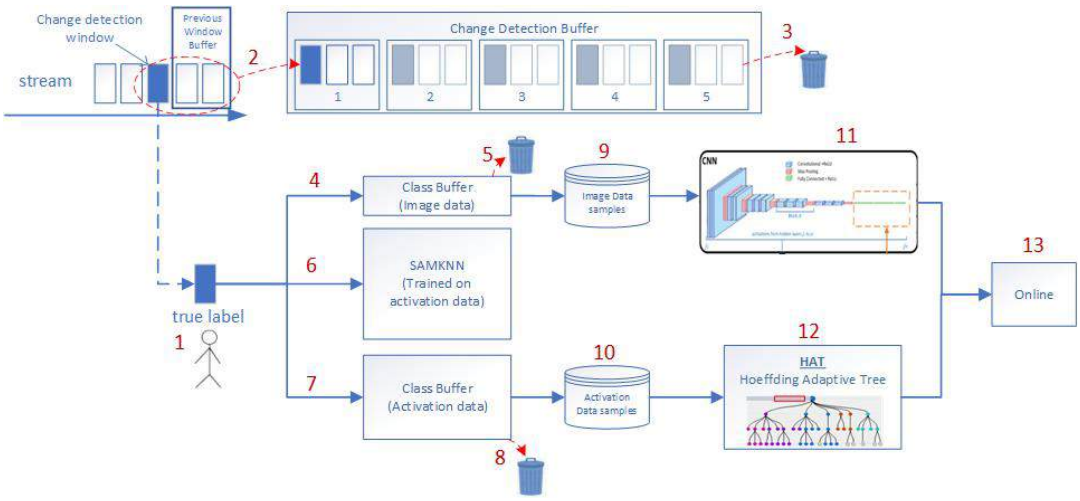


FIGURE 5.3: Overview of the DSAdapt adaptation method

window activation data to class buffer. 8. Remove the oldest instances from the activation class buffer. 9. Get image data samples (100 per class that is present in the change detection buffer). 10. Get activation data samples (same instances as in number 9). 11. Train the CNN offline. 12. Train the HAT offline. 13. Move the CNN and HAT online at the same time.

5.3 Experimental Methodology

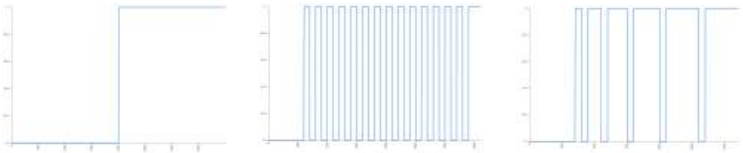
This section provides details of the experiments conducted and the specific setup implemented for the proposed method in this chapter.

5.3.1 Datasets

We use datasets CIFAR-10, CIFAR-100 [103] and Fashion-MNIST [198]. Information regarding the CIFAR-10 and Fashion-MNIST dataset can be found in Section 4.3.1. The CIFAR-100 dataset consists of 100 different classes of 32×32 colour images. The 100 classes are grouped into 20 super-classes. The super-classes are used in this chapter and are listed in Table 5.2. In total there are 5000 training images and 1000 test images. There are 2500 training images per super-class, and 500 test images per super-class. CIFAR-100 has the same attributes as CIFAR-10, as outlined in Section 4.3.1, specifically that it is a widely used benchmark dataset of photo-realistic images considered a hard classification scenario. It provides a larger range of super-classes and sub-classes, but with less images per class than CIFAR-10.

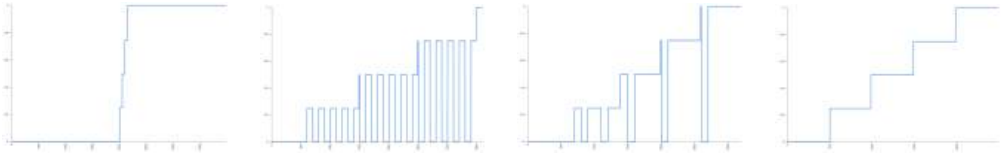
5.3.2 Data Combinations

Three combinations of eight trained classes and two novel classes have been selected from each dataset. An empirical study was conducted on the effectiveness of our drift detection mechanism on different combinations of pairs of novel classes. It was found that the drift detection was more effective when the classes differed in their categories. The results of this study can be found in Appendix C.2. For CIFAR-10, the classes can be split into categories of Transport and Animals, and for Fashion-MNIST, the classes can be split into categories of Clothing and Footwear. Therefore, to give a more rounded analysis, novel class combinations were selected as (1) a pair containing a mix of categories, (2) a pair containing classes from the same category (Animals for CIFAR-10 and Clothing for Fashion-MNIST) and (3) a pair containing classes from the other category (Transport for CIFAR-10 and Footwear for Fashion-MNIST). The selected combinations are shown in Table 5.2. For CIFAR-100, the class combinations were randomly selected. The trained and novel class identifiers are listed with a key of the class labels.



(a) Abrupt (b) Reoccurring (c) Gradual

FIGURE 5.4: Temporal types of concept evolution patterns



(a) Abrupt (b) Reoccurring (c) Gradual (d) Incremental

FIGURE 5.5: Categorical types of concept evolution patterns

For the test data, the instances are applied as in Figures 5.4 and 5.5. The x-axis is the number of images, and the y-axis is the cumulative number of concept evolution classes that have been introduced into the data stream. For instance, in Figures 5.4 and 5.5, when the line is at zero, this represents that only images the CNN has been trained on are in the data stream, and when the line is at 1, all concept evolution classes have been applied to the stream. Figure 5.5 shows the categorical patterns.

TABLE 5.2: Class data combinations

Trained Classes	Novel Classes	Class Identification Key
	Fashion-MNIST	
0-1-2-3-5-6-8-9	4-7	0 = T-Shirt/Top
2-3-4-5-6-7-8-9	0-1	1 = Trousers
0-1-2-3-4-6-7-8	5-9	2 = Pullover
0-1-2-3-4-6	5-7-8-9	3 = Dress
4-5-6-7-8-9	0-1-2-3	4 = Coat
2-3-4-6-8-9	0-1-5-7	5 = Sandal
0-1-8-9	2-3-4-5-6-7	6 = Shirt
0-1-5-6	2-3-4-7-8-9	7 = Sneaker
0-1-2-7	3-4-5-6-8-9	8 = Bag
6-9	0-1-2-3-4-5-7-8	9 = Ankle Boot
1-8	0-2-3-4-5-6-7-9	
0-5	1-2-3-4-6-7-8-9	
	CIFAR-10	
0-1-2-3-4-6-7-8	5-9	0 = Airplane
2-3-4-5-6-7-8-9	0-1	1 = Automobile
0-1-2-4-5-6-8-9	3-7	2 = Bird
0-1-2-3-6-7	4-5-8-9	3 = Cat
2-3-4-5-6-7	0-1-8-9	4 = Deer
0-1-4-5-8-9	2-3-6-7	5 = Dog
5-7-8-9	0-1-2-3-4-6	6 = Frog
0-2-6-7	1-3-4-5-8-9	7 = Horse
1-2-3-4	0-5-6-7-8-9	8 = Ship
0-7	1-2-3-4-5-6-8-9	9 = Truck
1-8	0-2-3-4-5-6-7-9	
2-3	0-1-4-5-6-7-8-9	
	CIFAR-100	
1-3-4-5-6-7-10-11-13-17	8-18	0 = Aquatic Mammals
0-1-3-5-11-12-15-17-18-19	2-7	1 = Fish
1-5-7-8-9-14-15-16-17-18	0-10	2 = Flowers
4-7-8-9-11-14-15-17-18-19	2-5-6-16	3 = Food Containers
0-2-4-5-7-9-13-14-15-18	10-11-12-19	4 = Fruit and Vegetables
5-6-7-10-11-12-14-17-18-19	1-4-8-9	5 = Household Electrical Devices
1-2-4-6-7-9-11-16-18-19	0-8-10-12-13-17	6 = Household Furniture
0-1-6-7-8-9-11-12-17-18	2-4-5-15-16-19	7 = Insects
0-2-3-5-8-9-10-11-12-19	4-6-13-15-17-18	8 = Large Carnivores
0-9-10-11-12-13-14-16-17-18	2-3-4-5-6-8-15-19	9 = Large Man-Made Outdoor Things
0-2-3-4-6-8-9-13-16-19	1-10-11-12-14-15-17-18	10 = Large Natural Outdoor Scenes
2-5-10-11-13-15-16-17-18-19	1-3-6-7-8-9-12-14	11 = Large Omnivores and Herbivores
4-5-8-12-14-15-16-17-18-19	0-1-2-3-6-7-9-10-11-13	12 = Medium-Sized Mammals
1-2-3-4-5-6-7-8-12-13	0-9-10-11-14-15-16-17-18-19	13 = Non-Insect Invertibrates
0-1-7-8-9-10-11-14-18-19	2-3-4-5-6-12-13-15-16-17	14 = People
		15 = Reptiles
		16 = Small Mammals
		17 = Trees
		18 = Vehicles 1
		19 = Vehicles 2

There are four steps, representing that there were four concept evolution classes that have been cumulatively applied at different times in the data stream. No incorrectly classified instances are removed during testing in order to simulate real-world applications.

5.3.3 Experimental Settings

We experiment with two different methods of activation reduction: JSDL and DSCBIR. The CNN that has been applied to the proposed system is VGG16, using transfer learning from ImageNet weights and trained on three combinations of classes for each data setup. All hidden layers that have measurable outputs in PyTorch [146] are used. We selected a well-known pretrained network and applied transfer learning as this is a common scenario in real-world applications.

The CNN adaptation of *AdaDeepStream* is substituted with four different adaptation methods: (1) LwF [110] (2) iCARL [156], (3) ER [31] and (4) MIR [8], augmented with trick RV [25]. These are successful methods in the OCI setting. Some methods are adjusted to be appropriate for the single-head setting.

Single-head [32] configuration is where all classes (previously known and novel classes) have a single shared output layer and do not need to know the class label [122]. The alternative is multi-head where more than one output layer is created as the model adapts and extra information is required to select the correct head. Our method is a single-head configuration and we will therefore compare with the single-head implementations. Each of these methods has been adjusted to receive only the true-labelled windows of instances when drift was detected from *AdaDeepStream* and to remove prior knowledge of the number of novel classes. The following hyper-parameters, as specified in [122] are set for all of the CNN adaptation methods: Learning rate = 0.01, epochs = 3, weight decay = 0 and memory buffers = 5000 (the mid-range that was used in the survey paper [122]). An overview of each of these methods follows:

1. LwF (Learning without Forgetting) is a regularisation and knowledge distillation-based method [85]. As previously explained in Section 2.4, in OCL, tasks are data changes. For instance, a new task can be the addition of a novel class

into the data. A student model is created for the new task and the original model becomes the teacher model. The authors describe it as a combination of distillation networks [85] and fine tuning [64], where fine tuning initialises with parameters from an existing network trained on related data and trains for new data with a lower learning rate on part of the network. Distillation networks takes a more complex ensemble of networks and applies the parameters to a simpler network to produce the same outputs. LwF adds task specific parameters for a new task and learns parameters that work well on old and new tasks, using images and labels from only the new task and no data from existing tasks. A variant of LwF is used, LwF.MC that only has one head (where all tasks share the same output head) [122].

2. iCARL (Incremental Classifier and Representation Learning) is a memory-based method. A training set is constructed by mixing all the samples in the memory buffer and the current task samples. The loss function has a classification loss to help the model correctly predict the novel classes and a knowledge distillation loss to prompt the model to reproduce the outputs from the previous model for old classes. It uses binary cross-entropy for each class to handle the imbalance between old and new classes. Originally, a Nearest Class Mean (NCM) classifier is used with the memory buffer to predict labels for test images [127], which means it looks for a subset of samples whose mean of latent features have the closest Euclidean distance to the mean of all samples in this class; however, this method requires all samples from all classes, and therefore cannot be applied in the online setting. Therefore, the modified version from [122] with reservoir sampling [190] is used instead of NCM [156].

3. ER (Experience Replay) is the use of a small memory, termed episodic memory, that stores examples from previous tasks and then replays these examples when training for future tasks. It jointly trains on both the examples from the current task and examples stored in the episodic memory [31].

4. MIR (Maximally Interfered Retrieval) is a recent memory-based method that performs an estimated parameter update based on the incoming mini-batch using stochastic gradient descent. It uses the samples from the memory buffer

that have the largest estimated loss increases and mixes them with the incoming mini-batch. In this contribution, MIR is augmented with RV (Review Trick) [25], as RV can provide an improvement and is more efficient in memory than NCM [122]. RV reduces class imbalance by applying a fine-tuning step with a small learning rate, using a balanced subset from the memory buffer and the training set [121]. As recommended, We use RV from [121] with a learning rate 10 times smaller than the training learning rate.

The entire *AdaDeepStream* system is compared to RSB [101] and CPE [195]. These were selected due to their use of CNNs. Due to a shortage of methods detecting discrepancies and directly adapting a CNN, we have RSB, which does directly adapt a CNN, and we also have CPE, which uses a CNN in its detection method, but does not adapt a main classifying CNN.

1. RSB (Reactive Subspace Buffer) is a memory-based method. It has centroid-driven memory and stores diverse samples of incrementally arriving instances. It is based on experience replay and combines class-based prototype centroid-driven memory with a reactive sub space buffer that tracks drift occurrences in previously seen classes and adapts clusters accordingly [101]. It traces the dominant class in each of the clusters and can switch labels of the clusters or split them when local changes are detected. User configurable parameters are set to: Max centroids = 10, max size of class buffer = 100, window size = 100. The window size is set to the same size as our system, which is a generally accepted size in streaming machine learning as it is large enough to perform statistics on [65]. The class buffer is set to the same size as our system, which provides a comparable environmental setting and means the amount of training data is manageable in data combinations with a larger number of classes. Max centroids were tuned from a range of 1 to 20 on a subset of data (two from each data combination for each number of trained classes for class and sub-class data). All other parameters are as recommended in the paper [101].

2. CPE (CNN based Prototype Ensemble) is cluster-based and projects the images into a learned discriminated feature representation called prototypes. This improves the intra-class compactness and expand inter-class separateness in the

output feature representation to enable the robustness of novel class detection. When a pre-defined number of novel class instances are detected, the CNN retrains [195]. Contrary to our system and RSB, the CNN is used in the detection of the novel classes, not as an overall image classifying CNN. Recommended settings from the paper are used for the training of the CNN: 1000 instances from each class of training data, number of novel classes that are accumulated before CNN retraining = 1000, learning rate = 0.001 [195]. Tuning was performed for the threshold of the prototypes, manually tuning between 1 and 100, and a value of 5.0 was found to produce the best results.

The system is running on AMD Ryzen Threadripper PRO 3955WX 16-Cores 3.90 GHz, 256 GB RAM with NVIDIA RTX A6000 GPU.

5.4 Experimental Results

In this section, we discuss our thorough experimental study, evidencing the efficacy of the proposed method. The results are discussed in terms of accuracy for each of the activation reduction methods and subsequently, timings for the time per instance and DNN adaptation time.

Experiments were conducted on three datasets (CIFAR-10, CIFAR-100 and Fashion-MNIST). Each using two different activation reduction methods (DS-CBIR and JSDL). CIFAR-100 accuracy plots are shown in Figures 5.6 and 5.7. Plots for all other datasets are included in Appendix C.3. The RSB and CPE results are displayed within the DS-CBIR plots for ease of comparison, but they do not use any activation reduction method. Figures 5.6 and 5.7 show how the accuracy of the models vary with an increasing number of applied novel classes for the CIFAR-100 dataset. Generally, the accuracy of all methods tend to decrease when the number of novel classes are increased. This is more prominent in the DS-CBIR results (Figure 5.6) than it is in the JSDL results (Figure 5.7), where the accuracies are more erratic. This pattern is also prominent in the other dataset results in Appendix C.3. This indicates that DS-CBIR is more consistent at detecting the changes in drift than JSDL.

In the Temporal-Reoccurring plots in Figures 5.6, 5.7 and Appendix C.3, higher accuracies are reported than for the other concept evolution patterns. This can be

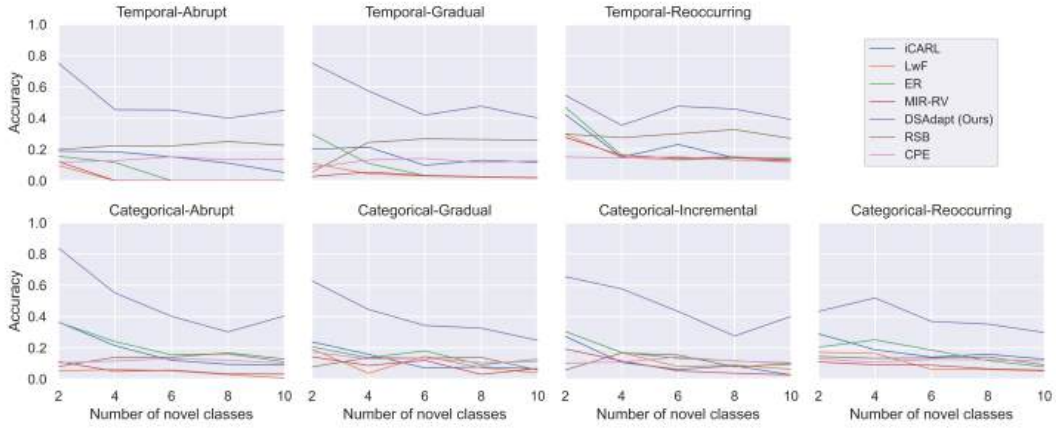


FIGURE 5.6: Number of novel classes against accuracy for DS-CBIR VGG16 CNN, CIFAR-100 for all concept evolution patterns.

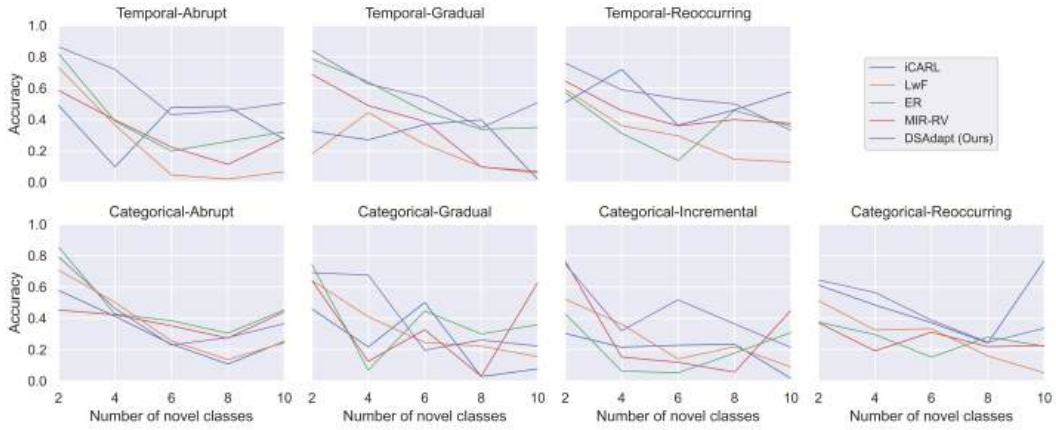


FIGURE 5.7: Number of novel classes against Accuracy for JSDL VGG16 CNN, CIFAR-100 for all concept evolution patterns.

seen for both DS-CBIR and JSDL. This could be due to there being a more diverse range of old and new classes in less windows in this pattern as compared to the other patterns. It is temporal, so the known classes are applied randomly rather than incrementally. The novel classes occur in small blocks alongside the known classes. This more diverse data is applied to the adaptation.

As DS-CBIR is more stable than JSDL, we focus on the DS-CBIR activation reduction method. Table 5.3 shows the average accuracy after adaptation for DS-CBIR, all datasets and all concept evolution patterns. The equivalent data for the JSDL activation reduction method is included in Appendix C.3, Table C.4. The data combinations with a large number of trained classes and a small number of novel classes

TABLE 5.3: Average accuracy after CNN adaptation for each concept evolution pattern for DS-CBIR activation reduction.

Reduction/ Dataset	Method	Cat. Abr.	Tem. Abr.	Cat. Gra.	Tem. Gra.	Cat. inc.	Cat. Reo.	Tem. Reo.
DS-CBIR CIFAR-10 8 Trained 2 Novel classes	iCARL	0.304	0.308	0.171	0.174	0.154	0.056	0.442
	LwF	0.044	0.080	0.236	0.127	0.354	0.187	0.437
	ER	0.548	0.453	0.459	0.677	0.457	0.377	**0.694**
	MIR-RV	0.070	0.090	0.131	0.113	0.162	0.112	0.491
	RSB	0.359	0.416	0.343	0.429	0.270	0.386	0.630
	CPE	0.546	0.470	0.458	0.481	0.372	0.460	0.398
	ADS (ours)	**0.888**	**0.819**	**0.543**	**0.807**	**0.732**	**0.702**	0.598
DS-CBIR CIFAR-100 10 Trained 2 Novel classes	iCARL	0.365	0.190	0.238	0.201	0.274	0.290	0.425
	LwF	0.053	0.097	0.193	0.112	0.305	0.172	0.298
	ER	0.363	0.157	0.207	0.298	0.305	0.204	0.471
	MIR-RV	0.112	0.122	0.143	0.027	0.193	0.110	0.278
	RSB	0.080	0.200	0.078	0.055	0.058	0.145	0.299
	CPE	0.104	0.12	0.175	0.084	0.102	0.131	0.153
	ADS (ours)	**0.838**	**0.752**	**0.629**	**0.753**	**0.655**	**0.433**	**0.547**
DS-CBIR MNIST-Fashion 8 Trained 2 Novel classes	iCARL	0.681	0.116	0.235	0.569	0.227	0.103	0.626
	LwF	0.180	0.049	0.106	0.115	0.183	0.102	0.461
	ER	0.806	0.606	0.317	**0.907**	0.324	0.166	0.728
	MIR-RV	0.18	0.167	0.198	0.205	0.001	0.038	0.547
	RSB	0.771	0.656	0.606	0.644	0.682	0.687	**0.816**
	CPE	0.925	**0.897**	**0.93**	0.898	**0.907**	**0.861**	0.815
	ADS (ours)	**0.964**	0.661	0.831	0.884	0.860	0.807	0.568

Legend: Highest values in bold.

are shown in these tables. This scenario is more usually seen in real-world applications. For instance, a model would be pretrained on known data, and see a few novel classes arriving, such as in medical image analysis where identification of changes in chest x-rays is required. A deep neural network would be trained on many known diseases, and one or two new variations resulting in different chest x-ray results would occur, rather than many new manifestations in a short amount of time.

For DS-CBIR, CIFAR-10 and CIFAR-100, Our method (ADS) has exceeded all others, except in one scenario of Temporal-Reoccurring. The accuracies for Fashion-MNIST are generally higher for all methods. This is probably due to the superior intra-class cohesion and inter-class separation of the reduced activation data, as demonstrated in the UMAP [125] representations in Figure 5.2. Our method is among the best; however, CPE excels on the Fashion-MNIST dataset. The standard deviation of the methods ranges from 0.116 to 0.684, our method has the highest. The implicit drift detection methods of RSB and CPE have standard deviations of 0.116 and 0.091 respectively, which are the lowest. Therefore, the higher standard deviation may indicate that the explicit nature of the drift detection has an effect on the consistency of our method. There is a connection between how successful the CNN adaptation is and how successful the drift detection is, as the drift detection is

based on the difference in predictions between the CNN and the Hoeffding Adaptive Tree Streaming Classifier. Therefore, the less successful the adaptation is, the more unstable the drift detection will be.

On average, of the different concept evolution patterns applied, our DSAdapt method outperforms the other adaptation methods. The two best-performing methods are DSAdapt (ours) and ER, which are both memory-based and use the standard cross-entropy loss. The regularisation and knowledge distillation-based method, LwF did not perform well in our scenarios. CPE only performs well on the Fashion-MNIST dataset. This method does not have a CNN that classifies images, but has a CNN that learns a feature representation of the images. Therefore, it is not a direct comparison and our results suggest it prefers data that has superior intra-class cohesion and inter-class separation. After *AdaDeepStream*, the next highest performing adaptation method is ER and the next best performing combined concept evolution and CNN adaptation method is CPE. On average, in our scenarios in Table 5.3, *AdaDeepStream* outperforms ER by 27% and CPE by 24%.

The Wilcoxon Signed-Rank test is used to analyse the difference between the accuracies of methods. The difference between the accuracy of *AdaDeepStream* and that of ER over the 272 tested data patterns is statistically significant. The p-value is less than 0.00001, which is less than 0.05 significance level, suggesting the acceptance of the alternative hypothesis that true location shift is not equal to 0. The same is true for CPE and RSB. Therefore, *AdaDeepStream* with our DSAdapt adaptation method significantly outperforms the next-best substituted CNN adaptation method of ER and *AdaDeepStream* significantly outperforms the drift detection and adaptation methods of CPE and RSB. The initial CNN accuracies are included in Appendix C.1

Tables 5.4 and 5.5 summarise the timings for the inference time and the time to adapt, respectively and compare timings of the activation reduction methods of DS-CBIR and JSDL. The RSB and CPE overall comparison methods are listed in these tables under the DS-CBIR column for ease of comparison. However, they do not use an activation reduction method. The time to process one instance in milliseconds is shown in Table 5.4. This is the time per instance, measured in batches of 100 instances. ER has the fastest time per instance with 7.1ms, with *AdaDeepStream*

TABLE 5.4: Time per instance (ms) and rank.

	DS-CBIR		JSDL	
	Time (ms)	Rank	Time (ms)	Rank
iCARL	266.5 (82.2)	6	329.5 (92.5)	6
LwF	8.1 (7.7)	3	**14.2 (8.0)**	**1**
ER	**7.1 (2.8)**	**1**	16.3 (13.5)	3
MIR-RV	7.2 (2.4)	2	16.3 (13.6)	4
RSB	19.9 (7.8)	5	-	-
CPE	545.2 (290.1)	7	-	-
ADS (Ours)	8.5 (2.1)	4	15.7 (7.0)	2

Legend: Standard deviation in brackets. Lowest values in bold.

TABLE 5.5: Adaptation time (s) (with standard deviation in brackets) and rank. Lowest values are in bold

	DS-CBIR		JSDL	
	Time (s)	Rank	Time (s)	Rank
iCARL	78.095 (45.3)	6	115.373 (53.2)	6
LwF	6.095 (7.1)	2	21.463 (23.4)	3
ER	6.774 (6.1)	3	16.976 (18.2)	2
MIR-RV	36.000 (17.9)	5	56.256 (38.1)	5
RSB	**1.995 (0.7)**	**1**	-	-
CPE	420.000 (123.0)	7	-	-
ADS (Ours)	7.690 (1.9)	4	**7.816 (1.2)**	**1**

Legend: Standard deviation in brackets. Lowest values in bold.

having the fourth fastest at 8.5ms. This is close to the fastest results given that CPE and iCARL are 545.2ms and 266.5ms respectively. From the adaptation time, ours is fourth fastest at 7.69 seconds. The fastest is RSB at 1.995s, with CPE being the slowest at 420s. However, RSB adapts on each data window and each individual adaptation does not appear to have a big effect and it takes a number of windows before an increase in the accuracy is seen. Discounting RSB, *AdaDeepStream* is close to the fastest adaptation time of the next fastest methods of LwF and ER. From Tables 5.4 and 5.5, DS-CBIR outperforms JSDL with regards to the time per instance, and adaptation time.

5.5 Summary

In this chapter, we proposed a method called *AdaDeepStream* which detects concept evolution and adapts a DNN to this. Two DNN activation reduction methods were analysed, one using an extended JS-Divergence method as compared to *DeepStreamOS* in Chapter 4, and one using content-based image retrieval descriptor generation via DNN activations. The output of these were input into our novel concept evolution detection method which uses a Hoeffding adaptive tree and accuracy volatility-based detection method. The DNN adaptation uses a partial network update and DNN activations to assist in a memory based method to mitigate catastrophic forgetting.

To evaluate the proposed method, a set of experiments were conducted with three benchmark image datasets. Classes were selected for the system to be trained on and other classes were withheld in order to be applied as concept evolution via novel classes in seven different patterns. These data patterns were applied in various class combinations to the commonly used VGG16 DNN. *AdaDeepStream* was compared to two state-of-the-art detection and DNN adaptation systems and the DNN adaptation method was compared with four leading OCI DNN adaptation methods.

The results showed that of the two activation reduction methods: JSDL and DS-CBIR, DS-CBIR is more consistent at detecting the changes in drift than the JSDL method. *AdaDeepStream* exceeded other methods on the accuracy for CIFAR-10 and CIFAR-100 data with all methods responding well to the Fashion-MNIST data. However, our method did generally have higher standard deviations. This may be because the drift detection is based on the difference in predictions between the CNN and the streaming classifier and the less successful the adaptation is, the more unstable the drift detection will be. Another contributing factor could be the relatively small amount of data we are extracting from the activations. Our adaptation component (DSAdapt) outperformed the other leading OCI DNN adaptation methods significantly as confirmed by the Wilcoxon Signed-Rank test. The overall average accuracy of the more real-world scenarios presented in Table 5.3 show that *AdaDeepStream* outperforms the next highest combined detection and adaptation method of CPE by 24% and the next highest OCI DNN adaptation method of

ER by 27%. In terms of timings, *AdaDeepStream* is amongst the fastest compared methods on both instance inference time and DNN adaptation.

AdaDeepStream is able to detect concept evolution using DNN activations and adapt a DNN to this. DS-CBIR was the most successful activation reduction method. Although an improvement on our previous *DeepStreamOS* method, there is still a relatively small amount of activations extracted. We have shown this is sufficient for concept evolution detection however, in the next chapter we are detecting concept drift which is a more challenging problem as the changes in data that require detecting are smaller. In the next chapter, we propose the *DeepStreamEnsemble* method with a further modified CBIR method which extracts multi-layer activations and uses them in an ensemble, providing the input into our novel drift detection method and DNN adaptation method assisted by an activation ensemble. We detect concept drift whereby we apply novel sub-classes in drift patterns and perform DNN adaptation, thus satisfying the fourth objective from Section 1.5.

Chapter 6

DeepStreamEnsemble: Streaming DNN Adaptation to Concept Drift

The previous chapter presented the novel concept evolution detection and DNN adaptation method *AdaDeepStream* which built upon the *DeepStreamOS* method described in Chapter 4. *AdaDeepStream* addresses the third objective as stated in Section 1.5. In this chapter, we propose the *DeepStreamEnsemble* method to address the final objective, objective four of this thesis which is to: **Design, develop and evaluate a concept drift discrepancy detection and DNN adaptation method**. To achieve this more challenging objective, *DeepStreamEnsemble* uses DNN activations within ensembles. The work presented in this chapter is submitted as a paper titled "DeepStreamEnsemble: Streaming Adaptation to Concept Drift in Deep Neural Networks" to the "IEEE Transactions on Big Data, Special Issue on Stream Data Learning and its Applications" journal.
The code is available at https://github.com/chambai/DeepStreamEnsemble.

6.1 Introduction

The same issues outlined in Chapter 5 (Streaming DNN adaptation to Concept Evolution) also apply to Streaming DNN adaptation to Concept Drift. In summary, standard DNNs only recognise classes they were trained on, but in many real-world applications, the data evolves over time. When the data evolves, the DNN can attribute incorrect labels to the data with no indication of this.

In this research, we focus on detection of and DNN adaptation to concept drift in high dimensional data. Concept drift is data drift where changes in the input

data occur, but no new class labels arise [57]. We perform an experimental study using image data as the high dimensional data and a CNN as the type of DNN. We manifest the concept drift as new sub-classes in images. For instance, an image classification system is trained on a class of Flower that consists of Poppies and Roses. Concept drift is applied by adding images of Sunflowers in to the data stream. The overall class of Flower has not changed, but the input data for that class has.

As with concept evolution, DNN adaptation in a streaming scenario to concept drift encounters the same issues with respect to the DNN adaptation. To recap from Section 5.1, the issues are, that DNNs have a longer latency than other machine learning models, they suffer from catastrophic forgetting of previously known classes, class imbalance when new data is arising and they require more data as compared to other types of machine learning models.

In systems concerned with the detection of concept drift and adaptation of DNNs, the domains of Concept Drift and Online Domain Incremental (ODI) play crucial roles. ODI, a sub-field of Online Continuous Learning (OCL), addresses non-stationarity in data without changes in class labels [122]. As with the OCI (Online Class Incremental) field introduced in Section 5.1, ODI incrementally trains a newly initialised DNN, one class at a time [122] and focuses on retaining all past data, relying on true labelled samples leading to over inflated performance, whereas the concept drift field focuses on the most recent changing data, using minimal true labels, without the focus on retaining all past data, thus the two fields of Concept Drift and ODI offer differing perspectives.

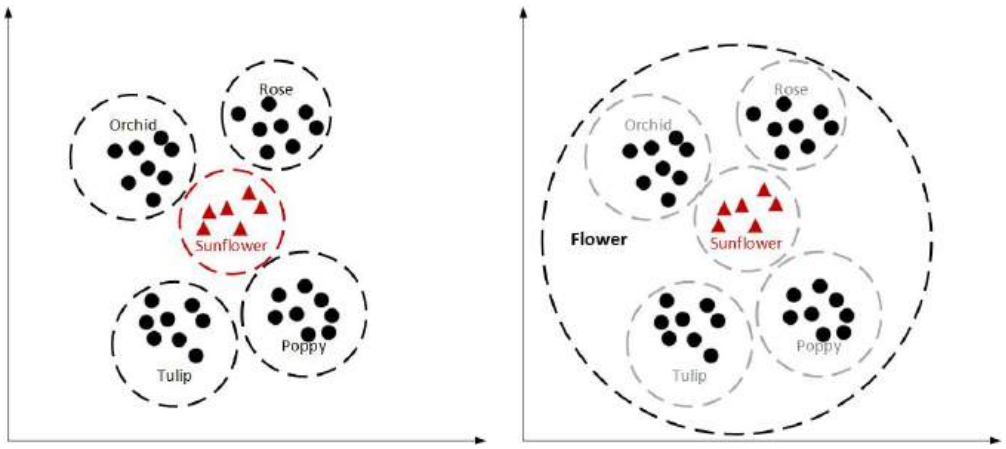
Current approaches to DNN adaptation in response to concept drift are constrained in the same way as for concept evolution, focusing on structured low dimensional data [56] [56]. Methods tend to be a subset of those for concept evolution, as a number of methods combine concept evolution and drift detection solutions. Many use clustering [101, 195, 60, 42, 73], with some utilising DNNs in the drift detection mechanism [60, 81].

The same DNN adaptation techniques that are applied in the OCI field are also applied in the ODI fields [156, 110, 31, 8]. These have not previously been applied in a concept drift setting. [101] initiated the integration of ODI with concept drift but did not explore the application of concept drift patterns.

In the Concept Drift and Online Domain Incremental (ODI) fields, concept drift typically involves changes such as new backgrounds, blur, noise, illumination, and occlusion [122, 101, 195]. These synthetic alterations to images may not faithfully replicate real-world scenarios. Hence, a novel approach is required to tackle concept drift detection and adapt deep neural networks to high dimensional data, with the specific characteristics of: (1) Transformation of data into an alternative representation. (2) Explicit concept drift detection. (3) Applicability to existing DNNs without the need for retraining. (4) Analysis concerning concept drift patterns. (5) Analysis concerning emerging novel sub-classes within a super-class.

Our system, called *DeepStreamEnsemble*, is designed to serve as a wrapper enabling pretrained standard DNNs to explicitly detect concept drift and adapt within seconds. This method involves offline training and online inference. To analyse unseen instances, we leverage neuron activations from within the hidden layers of the DNN. Using a Content-Based Image Retrieval (CBIR) DNN descriptor generator method [176], we reduce these activations and apply to a Hoeffding Tree [91] classifier ensemble for each block of hidden layers in the DNN. The difference in predictions between the Hoeffding Tree ensemble and the DNN serves as the basis for concept drift detection through our innovative method, referred to as DSE-DD, which relies on accuracy volatility. Upon detecting concept drift, our novel adaptation method, termed DSE-Adapt, is employed to adapt the DNN.

Detecting concept drift is more challenging than detecting concept evolution as the differences are subtler within-class changes, rather than completely novel classes. Figure 6.1 shows a comparison of class boundaries for concept evolution and concept drift. In (a) it can be seen that during concept evolution, the emerging class of Sunflower arises with its own class boundary. In (b) the class boundary is now the super-class of Flower. The sub-class of Sunflower emerges in the centre of the super-class, making it more challenging to detect than an entirely new class. Thus, this chapter provides a unique solution for concept drift detection and DNN adaptation to streaming data. More advanced methods of DNN activation extraction, concept drift detection and DNN adaptation using streaming machine learning ensembles are proposed in order to address the more challenging problem of concept drift.



(a) Concept Evolution (b) Concept Drift

FIGURE 6.1: Class boundaries for concept evolution and concept drift.
Legend: Black (round) are known instances. Red (triangles) are unknown instances. Black dashed line is a class boundary. Grey dotted line is a sub-class boundary.

We focus our experimental methodology on CNNs as the DNN and images as the high dimensional data. To the best of our knowledge, *DeepStreamEnsemble* is the first streaming concept drift detection and CNN adaptation system to focus on accuracy volatility using CNN activations for concept drift detection. Its use of activations from different sections of the CNN, and a streaming machine learning ensemble is unique. We present a novel activation reduction method based on a CBIR technique, which is the first time CBIR has been involved as an input for concept drift detection. We individually substitute concept drift detection and CNN adaptation modules whereby the user can independently select these components. We provide the first analysis of abrupt, gradual and reoccurring drift patterns to the ODI field. For the first time, we apply concept drift as sub-class images to streaming concept drift detection and CNN adaptation systems, rather than synthetic augmentations. The main contributions presented in this chapter are:

1. Reduction of DNN activations based on a CBIR technique, utilised as input to concept drift detection.

2. Explicit concept drift detection method using multi-layer DNN activations and streaming machine learning ensemble.

3. DNN adaptation method using multi-layer DNN activations and streaming machine learning ensemble.

4. Unification of ODI and concept drift fields by analysing ODI DNN adaptation methods in a sub-class concept drift setting via drift patterns.

This chapter is organised as follows: In section 6.2 we present a description of the system that includes formalisation and implementation details of the *DeepStreamEnsemble* components and methodology. In the experimental study in Section 6.3 we specify the experimental setup. In Section 6.4 we evaluate and analyse *DeepStreamEnsemble* against eleven other methods. Section 6.5 summarises our findings.

6.2 DeepStreamEnsemble System Description

This section elaborates on the components of our *DeepStreamEnsemble* system and how they interact, providing detailed descriptions, algorithms and formalisations. It introduces our neuron activation reduction method (Block-CBIR), drift detection method (DSE-DD), and CNN adaptation method (DSE-Adapt), along with an explanation of how the substituted drift detection and CNN adaptation methods are applied.

The definition of a data stream as in Section 5.2 also applies for this chapter, except that sub-classes are emerging during inference. The DNN is trained via the cross-entropy cost function as described by Equation 3.2.

Each detected change signifies a potential new sub-class within the current window. The model f, is adjusted to incorporate the identified new sub-class. Index notation is employed in the algorithm descriptions; for example, D_w denotes the wth window in the data stream. Commonly used symbols are listed in Table 6.1. To streamline the descriptions, the Hoeffding Tree Block Ensemble is abbreviated as HTBE, and the Adaptation Ensemble Buffer is referred to as AEB.

The proposed *DeepStreamEnsemble* system is illustrated in Figure 6.2. The system requires two prerequisites: (1) A CNN fitted with training data (C), which is the subject of the concept drift detection and adaptation, and (2) data samples used to train the deep neural network (D_{init}). *DeepStreamEnsemble* operates as an offline-online

TABLE 6.1: Summary of main symbols

Symbol	Description	Symbol	Description
A	Reduced Activations	N	Number of convolutional blocks
B	Change Detection buffer	P	CNN predicted class labels
b	Convolutional block	q	Window count
C	CNN	S	HTBE class predictions
C_{adapt}	Adapted CNN	s	Sample from class buffer
c	Class	U	Previous window buffer
D	Data stream	V	Set of accuracy differences
d	Drift detection result	v	Accuracy difference
E	Class buffer	W	Number of change detection groups in B
K	AEB predicted classes	w	Window number
k	Feature map	x	Image data
L	Number of classes	Y	True class labels
M	Number of training set samples		

system, undergoing offline training and subsequently processing unseen instances online.

Referring to Figure 6.2, *DeepStreamEnsemble* comprises of four modules: The CNN, Activation Reduction, Concept Drift Detection and Adaptation. The activations for each convolutional block in the CNN are extracted (b_1 to b_N) into J_w. Each block consists of convolutional layers followed by a max pooling layer. The CNN activations are reduced via Block-CBIR in the activation reduction module by extracting the most important neurons in the max pooling and convolutional layers and are provided to the concept drift detection as A_w in Figure 6.2. The concept drift detection requires the activation reduction output from the Block-CBIR module, the Hoeffding Tree Block Ensemble (HTBE) predictions and accuracy volatility change detection between the predictions of the CNN (P_w) and the Hoeffding as determined by the DSE-DD module. The intuition behind the concept drift detection is that the CNN is a natural feature extractor. Each layer learns to detect different features from the input image. The initial layers learn to detect simpler patterns such as edges and textures, while deeper layers recognise more complex features such as shapes or entire objects [105]. These learned features are represented via feature maps (the values of which are called activations). Analysing these feature maps provides an opportunity to extract alternative aspects of the data than is possible via purely using the image data. Each image has its own way of activating the neural network and producing activation patterns. The CNN is a strong classifier, and the Hoeffding Tree

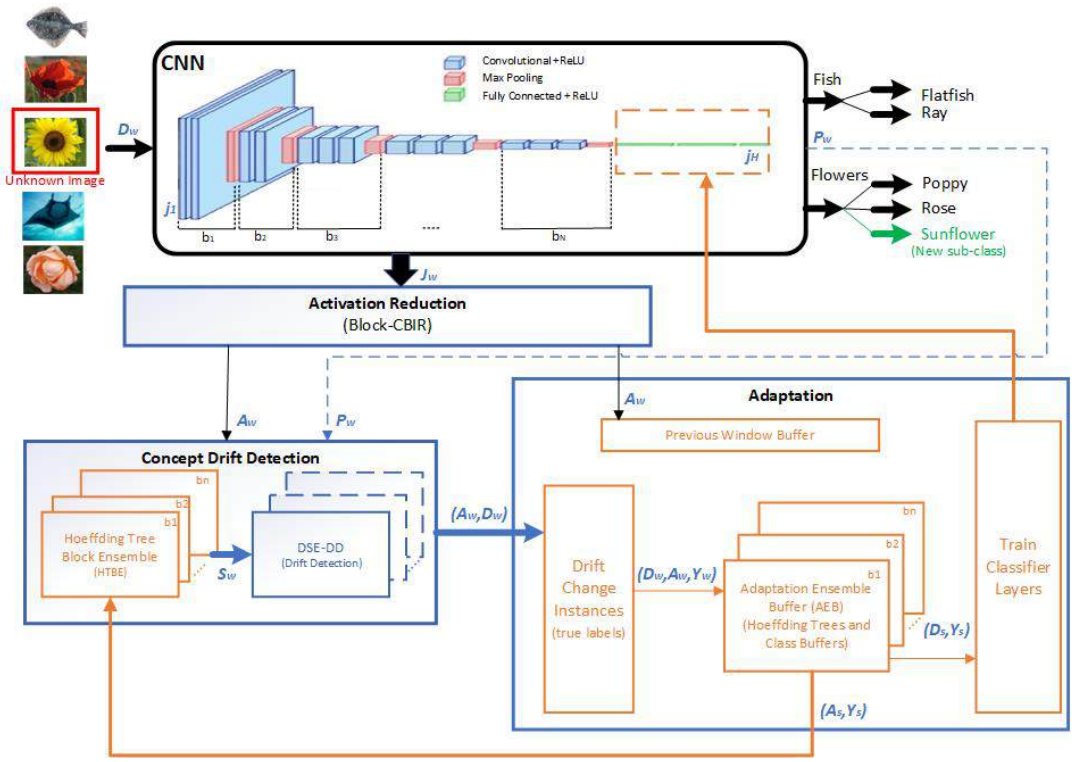


FIGURE 6.2: Overview of the *DeepStreamEnsemble* system. An unknown image featuring a Sunflower is input to the CNN, activations are extracted from various convolutional blocks, concept drift is identified, adapting the CNN for the new Sunflower sub-class within the Flowers category.

is a weaker classifier, but adaptive. The Hoeffding Tree [91] is a streaming classifier and operates on the minimal amount of data required to make a decision. The Hoeffding Trees will adapt to the reduced activations of the CNN more often than the CNN is adapted. This assists in the Hoeffding Tree diverging further from the CNN, contributing to the accuracy volatility between the CNN and the Hoeffding Trees. There is one Hoeffding tree for each convolutional block (b_1 to b_N) as shown in Figure 6.2. Using the accuracy volatility between the prediction of the DNN and the prediction of the Hoeffding Trees for each block provides monitoring of changes in these latent representations from the CNN. The HTBE predictions for all blocks (S_w) are provided to the change detection element (labelled as DSE-DD in Figure 6.2). The DSE-DD calculates the total accuracy between the CNN and the HTBE predictions for the current window of data (w). The change in accuracy between windows is monitored in order to detect drift.

If change is detected, the window of data is provided to the adaptation phase (A_w, D_w) where it is true-labelled (shown in Figure 6.2 via the Drift Change Instances

element) and fitted to the AEB (one for each block b_1 to b_N) with image data, activation data and true labels (D_w, A_w, Y_w). Each block in the AEB consists of a Hoeffding Tree and a class buffer containing activation data. The Hoeffding Tree and the class buffers only contain training data and true-labelled instances. There is also a class buffer for the image data. The CNN is adapted offline, whilst instances continue to be processed. Instances arriving between the time the change was detected and the time it takes to adapt the CNN are re-predicted via the AEB. Windows prior to the detected drift change are stored in the previous window buffer as shown in Figure 6.2 as the Previous Window Buffer element with data (A_w), re-predicted via the AEB and used in the adaptation. Only the fully connected layers of the CNN are adapted via the Train Classifier Layers element in Figure 6.2 using data samples for each class, sourced from the class buffers in the AEB (D_s, Y_s). The HTBE Hoeffding trees are updated offline with the same samples of data as the CNN (except it is activation data instead of image data (A_s,Y_s)). When the CNN has completed adaptation, the system is updated. Algorithm 8 provides a system overview. Algorithm 9 details the activation reduction. Algorithm 10 details the concept drift detection and Algorithm 11 details the adaptation.

Algorithm 8 shows the overall process of *DeepStreamEnsemble*. In summary, for each window of image data, the activations are reduced (Line 2) via our method Block-CBIR. Drift detection is performed (Line 5) using the CNN and the HTBE predictions via our method DSE-DD. If drift is detected then true values of the windows in which drift was detected are obtained (Line 7). The DNN is adapted (Line 8) and the block ensembles are adapted with the new true values (Line 9). If the CNN is adapting, predictions are obtained from the AEB (Line 13). Otherwise, the predictions are obtained from the CNN (Line 16). At Line 5 (conceptDriftDetection), an alternative concept drift detection method can be invoked. At Line 8, (adaptDnn), an alternative CNN adaptation method can be invoked. Further information about the substituted concept drift detection and adaptation methods can be found in Section 6.3.

Algorithm 8 *DeepStreamEnsemble* Overview

Input: Windows of image data D
Input: Pretrained CNN, C on L classes
Input: Pretrained Hoeffding Tree on activations of L classes
Output: P: A set of CNN predicted classes for data stream, D

for $D_w \in D$ **do**
  $A_w \leftarrow$ **reduceActivations**(D_w) ▷ Block-CBIR or another method
  $P_w \leftarrow$ **cnnPredict**(D_w) ▷ get CNN predictions
  $S_w \leftarrow$ **htbePredict**(A_w) ▷ get HTBE predictions
  $r \leftarrow$**conceptDriftDetection**(A_w) ▷ DSE-DD or another method
  if $r = 1$ **then**
    $Y_w =$ **getTrueValues**(D_w) ▷ Get true values for the drift window
    adaptDnn(D_w, A_w, Y_w) ▷ DS-Adapt or another method
    adaptEnsembles(D_w, A_w, Y_w) ▷ Adapt HTBE
  else
    Let U be the window buffer and Let G be the number of windows in U
    if CNN adaptation in progress **then**
      $K_w \leftarrow$ **aebPredict**(A_w) ▷ use AEB predictions
      $U \leftarrow (D_w, A_w, K_w)$
    else
      $U \leftarrow (D_w, A_w, P_w)$ ▷ use CNN predictions
    end if
    if $G > 2$ **then** ▷ only store 2 windows
      $U \leftarrow U - U_0$ ▷ remove oldest window
    end if
  end if
  $P \leftarrow P_w$
end for

6.2.1 Activation Reduction

CBIR [176] is intended for content-based image retrieval. It creates descriptors for images using deep neural networks. It is based on obtaining neural codes from fully connected layers activations. CBIR takes this one stage further by using the information contained in convolutional layers. However, the number of neurons in the convolutional layers is large and most of them do not contribute significantly to the final classification. Therefore the most significant neuron activations only are extracted in order to provide extra information about the image such as background textures or colour distribution that is present in the convolutional layers [176]. A description of this method can be found in Section 3.4.5. We have modified CBIR to use within *DeepStreamEnsemble* to extract the most useful activations from the network such that we can utilise it in our streaming classifier. The algorithm for our CBIR implementation is presented in Algorithm 9. We have called this Block-CBIR to distinguish it from the original. This is invoked in the overall *DeepStreamEnsemble*

system (Algorithm 8) Line 2, via the **reduceActivations** method.

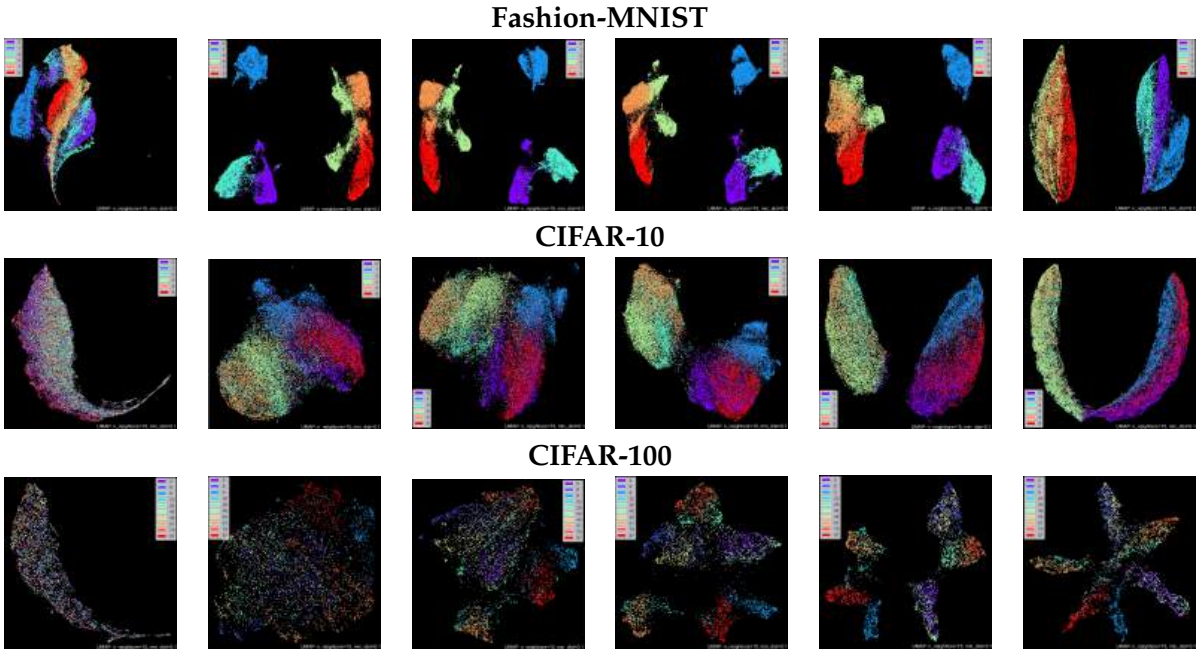
Algorithm 9 Activation Reduction: Block-CBIR

Input: One window of image data, D_w
Input: CNN, C expressed in convolutional blocks: $b \in C = \{1, 2, \ldots, N\}$
Output: A_w: One window of reduced activations

1: **for** $x_i \in D_w$ **do**
2: **for** $b \in C$ **do**
3: let b_p be the pooling layer of block b
4: **for** each channel, c of b_p **do**
5: $J_{max} \leftarrow$ **max**(b_p) ▷ max channel value in pooling layer
6: $J_{conv} \leftarrow$ **getConvLayer**(b) ▷ get conv layer 1 values
7: **end for**
8: $J_b = J_{max} + J_{conv}$
9: $A_b \leftarrow$ **sectionAvg**$(J_b, 150)$ ▷ get the average value for 150 sections
10: **end for**
11: $J_H \leftarrow$ **getActivations**(H) ▷ get final hidden layer activations
12: $A_H \leftarrow$ **sectionAvg**$(J_H, 32)$ ▷ get the average value for 32 sections
13: $A_w \leftarrow A_b + A_H$
14: **end for**

For previously unseen instances arriving at the deep neural network, the activations for one block at a time are extracted, where a block in a CNN consists of convolutional layers, and a pooling layer. For each value of each channel of the pooling layer in the block, the maximum activation value across the channels for each location in the pooling layer is extracted and stored (Line 5). Corresponding values are obtained from the first convolutional layer in the block via the **getConvLayer** method (Line 6). This is repeated for each block in the network.

In this work, there is less of a constraint on the number of features as each block's activations are processed individually. However, only the first convolutional layer in each block is used as, empirically, this provided superior results than using both convolutional layers. The original CBIR paper [176] used a threshold to save on computing time, reasoning that as ReLU (Rectified Linear Unit) activation functions were used, then processing under an activation threshold of 0.5 was not advantageous. As our system is designed to be flexible, we removed this threshold so that other types of activation functions could be used within applied CNNs, which, on our system, did not incur a significant increase in computing time but improved the clustering of the activations. The combined output of the max pooling layer activations and the convolutional layer activations from each block are reduced further via



(a) Block 1 (b) Block 2 (c) Block 3 (d) Block 4 (e) Block 5 (f) Final

FIGURE 6.3: UMAP representations of reduced activation training data for each block and final hidden layer of VGG16 CNN

the **sectionAvg** method (Line 9) where set J_b is split into 150 parts and the average for each part is calculated. The activations for the last hidden layer J_H are extracted and converted into 32 values in the same way (Line 12). These reduced sets are combined (Line 13) and returned as the set of reduced activations for the window.

To achieve Block-CBIR, the convolutional blocks need to be identified for the CNN. For VGG16 there are five convolutional blocks. Figure 6.3 shows an example of UMAP [125] representations of the reduced activations for the convolutional blocks and final hidden layer of VGG16 CNN for the Fashion-MNIST, CIFAR-10 and CIFAR-100 datasets. Sub-classes of 0-1-2-7-5-9 are shown for Fashion-MNIST. Sub-classes of 0-1-2-3-4-8 are shown for CIFAR-10. Sub-classes 2-8-9-22-26-46-48-61-79-87 are shown for CIFAR-100. The sub-classes are explained in detail in Table 6.2. From the UMAP images in Figure 6.3, the latter blocks (i.e. Block 5) show improved separation as compared to the earlier blocks (i.e. Block 1). Fashion-MNIST demonstrates superior separation in all blocks as compared to CIFAR-10 and CIFAR-100, with CIFAR-10 displaying the least separation and CIFAR-100 displaying fewer instances per class. This activation reduction method provides activation classification footprints, as discussed in Section 2.6, into our concept drift detection method described

in the following section.

6.2.2 Concept Drift Detection

Our Concept drift detection method (DSE-DD) is based on an ensemble of the activation data from the blocks of the CNN, the prediction of the CNN and the accuracy volatility of each CNN block. The following paragraph details this method.

Algorithm 10 Drift Detection: DSE-DD

Input: Window of reduced activation data, A_w
Input: CNN, C expressed in convolutional blocks, $b \in C = \{1, 2, \ldots, N\}$
Input: CNN class predictions for window, P_w
Input: HTBE predictions for window, S_w
Input: Pretrained HTBE on activations and L classes; 1 for each conv. block, b
Output: r: 1 if window contains drift, otherwise 0

1: **for** $b \in C$ **do** ▷ For each block in the CNN
2: $a \leftarrow$ **calcRunAcc**(S_w, P_w) ▷ accuracy for CNN block & CNN predictions
3: $V_b \leftarrow$ max(a) - min(a) ▷ store the accuracy difference for the block
4: **end for**
5: v_w = sum(V_b) ▷ sum the accuracy difference for all blocks
6: **if** $q > 3$ **then**
7: **if** $v_w >$ average($\{v_0, v_1, v_2\}$) **then** ▷ is window > avg. of first 3 windows
8: $r = 1$ ▷ drift occurred
9: **else**
10: $r = 0$ ▷ no drift occurred
11: **end if**
12: **else**
13: $V \leftarrow v_w$ ▷ store accuracy difference for this window
14: $r = 0$ ▷ no drift occurred
15: **end if**

Concept drift detection is achieved via our novel method, DSE-DD and is detailed in Algorithm 10. A window of reduced activation data A_w is applied to the HTBE, which is trained on reduced activations and L classes. The data from each block in the CNN is applied to its own Hoeffding Tree. The Hoeffding Tree classifier is established and well used [91]. The HTBE provides predictions for each convolutional block (b_1 to b_N) of the CNN. The running accuracy of the predictions for each of the blocks and the CNN prediction is calculated in **calcRunAcc** (Line 2). When the HTBE has been trained, the training samples are predicted via the HTBE and the CNN. These predicted values are stored and are used in the accuracy calculation of the unseen windows in **calcRunAcc**. The difference between the maximum and

minimum accuracy of the window is calculated (Line 3) and summed (Line 5). Under the scenario of new sub-classes, the earlier blocks in the CNN tend to be more transient than the latter blocks, thus these early blocks provide a valid input into the drift detection. As super-class predictions from the CNN are being used instead of true values, increasing or decreasing accuracy could indicate concept drift. Therefore any change in the accuracy is measured, regardless of whether it is an increase or decrease. A threshold is required and is calculated from the accuracy volatility difference of the first three windows of the data stream. If the accuracy difference for the subsequent test data windows is greater than this threshold, drift is declared (Lines 6 to 15).

6.2.3 Adaptation

Algorithm 11 DNN Adaptation: DSE-Adapt

Input: One window of image data: D_w
Input: One window of reduced activation data A_w
Input: One window of true labels: Y_w
Input: CNN to be adapted, C
Input: Buffer of two previous windows, U
Output: Adapted CNN, C_{adapt}

1: let W be the total number of change detection groups in buffer, B
2: $B \leftarrow U + (D_w, A_w, Y_w)$ ▷ save current and previous two windows
3: **if** $W > 5$ **then** ▷ check number of change detection groups
4: $B \leftarrow B - B_0$ ▷ remove oldest group
5: **end if**
6: **aebPartialFit**(D_w, A_w, Y_w) ▷ add true labelled data to AEB
7: $K_w \leftarrow$ **aebPredict**(B) ▷ re-predict buffer windows via AEB
8: Let E be the class buffer
9: **for** $b \in E$ **do**
10: $E_b \leftarrow E_b + (D_w, A_w, Y_w)$ ▷ Add true labelled data to class buffer
11: $E_b \leftarrow E_b - (D_0, A_0, Y_0)$ ▷ remove oldest instance
12: **end for**
13: **for** class l in $(K_w + Y_w)$ **do** ▷ get 100 samples per class from buffer
14: $(D_s, A_s, Y_s) \leftarrow$ **getClassBufferSample**(l, 100)
15: **end for**
16: $(D_s, A_s, Y_s) \leftarrow (D_s, A_s, Y_s) + B$ ▷ add drift windows to samples
17: copy C and train with (D_s, Y_s) for 3 epochs ▷ adapt CNN
18: **htbePartialFit**(A_s, Y_s) ▷ adapt HTBE with sampled data
19: replace C with adapted CNN, C_{adapt}

Our CNN adaptation method (DSE-Adapt) is based on partial update of the

CNN, an ensemble class buffer and some memory of previous instances, and is described in Algorithm 11. When a change is detected, that window of data is provided to the CNN adaptation method - the original image data (D_w), the activation data (A_w) and the true class values (Y_w). The current window and the previous two windows are added to a change detection buffer (Line 2). If the buffer exceeds 5 windows, the first window in the buffer is removed (Lines 3 and 4). This means that data previous to the change is also provided to the CNN adaptation and some data from previous change detection points are supplied (if available), thus providing some memory for the CNN adaptation. True-labelled instances are fitted to the AEB which is an ensemble, one for each CNN convolutional block plus the final layer, each containing a Hoeffding Tree. This is the same architecture as for the ensemble in the drift detection stage but only contains training data and true labelled samples (Line 6). The two windows of data that are collected prior to the drift detection window may contain incorrect CNN predictions due to the change in data. These instances are re-predicted via **aebPredict** (Line 7). The predictions from the AEB are weighted according to the training accuracy of each block of the CNN, thus **aebPredict** returns the prevailing prediction, taking into account the block accuracies. The current window true value instances are added to the class buffers (Line 10). A sample of 100 instances (or the maximum number of instances that are available) for each class that occurs in the true-labelled window and in the previous two windows is extracted from the class buffer (Lines 13 to 15) and used for CNN adaptation. The CNN is adapted for three epochs (Line 17). The HTBE is adapted (Line 18). The current CNN is replaced with the adapted CNN (Line 19).

6.3 Experimental Methodology

This section provides details of the experiments conducted, with details of the specific setup implemented for the proposed method in this chapter.

6.3.1 Datasets

We utilise the Fashion-MNIST [198], CIFAR-10, and CIFAR-100 datasets [103]. Information regarding the CIFAR-10 and Fashion-MNIST dataset can be found in Section 4.3.1 and information regarding CIFAR-100 can be found in Section 5.3.1. The 100

sub-classes are grouped into 20 super-classes, as shown in Table 6.2, along with the sub-class identifiers in brackets.

6.3.2 Data Combinations

Sub-class data combinations are applied. For example, on CIFAR-10, for a data combination of trained sub-classes 0-1-2-3 and novel sub-classes 8-9-4-5, the network will be trained on classes Airplane, Automobile, Bird and Cat. However, instead of using these labels, class labels of Transport, Transport, Animal and Animal are assigned. The Ship, Truck, Deer and Dog classes are applied as the novel sub-class data. For Fashion-MNIST and CIFAR-10, the data combinations have been selected such that there are groups of 2, 4 and 6 known classes and incrementing numbers of unknown sub-classes as shown in Table 6.3.

For CIFAR-100, two scenarios have been selected:

1. Trained on 5, 10 and 20 super-classes (2 sub-classes from each super-class), adding 2 randomly selected novel sub-classes, each within a trained super-class.

2. Trained on 5, 10 and 20 super-classes (2 sub-classes from each super-class), adding 3 novel sub-classes per selected sub-class.

Scenario (1) involves numerous known classes and a limited amount of concept drift classes, resembling typical real-world applications. For example, a model trained on extensive data might encounter a low number of emerging concept drift classes, as seen in i.e. medical image analysis. In this scenario, a DNN is fitted with data for various known diseases, with only a couple of new variations appearing over time. Scenario (2) represents a larger influx of concept drift classes. Table 6.3 provides details of three class combinations for scenarios (1) and (2).

6.3.3 Experimental Settings

The CNN utilised in the proposed system is VGG16, trained from initialisation with ImageNet weights; a common practice in real-world applications. Data instances are applied in patterns, the same as for concept evolution, as shown in Figures 5.4 and 5.5, where the number of images are shown on the x-axis, and the cumulative

TABLE 6.2: Super-classes and sub-classes

Super-Class Names	Sub-Class Names and Sub-Class Identifiers
	Fashion-MNIST
Footwear	Sandal, Sneaker, Bag, Ankle Boot (5, 7, 8, 9)
Clothing	T-Shirt/Top, Trouser, Pullover, Dress, Coat, Shirt (0, 1, 2, 3, 4, 6)
	CIFAR-10
Transport	Airplane, Automobile, Ship, Truck (0, 1, 8, 9)
Animal	Bird, Cat, Deer, Dog, Frog, Horse (2, 3, 4, 5, 6, 7)
	CIFAR-100
Aquatic Mammals	Beaver, Dolphin, Otter, Seal, Whale (4, 30, 55, 72, 95)
Fish	Aquarium Fish, Flatfish, Ray, Shark, Trout (1, 32,67, 73, 91)
Flowers	Orchids, Poppies, Roses, Sunflowers, Tulips (54, 62, 70, 82, 92)
Food Container	Bottles, Bowls, Cans, Cups, Plates (9, 10, 16, 28, 61)
Fruit and Vegetables	Apples, Mushrooms, Oranges, Pears, Sweet Peppers (0, 51, 53, 57, 83)
Household Electrical Devices	Clock, Computer Keyboard, Lamp, Telephone, Television (22, 39, 40, 86, 87)
Household Furniture	Bed, Chair, Couch, Table, Wardrobe (5, 20, 25, 84, 94)
Insects	Bee, Beetle, Butterfly, Caterpillar, Cockroach (6, 7, 14, 18, 24)
Large Carnivores	Bear, Leopard, Lion, Tiger, Wolf (3, 42, 43, 88, 97)
Large Man-Made Outdoor Things	Bridge, Castle, House, Road, Skyscraper (12, 17, 37, 68, 76)
Large Natural Outdoor Scenes	Cloud, Forest, Mountain, Plain, Sea (23, 33, 49, 60, 71)
Large Omnivores and Herbivores	Camel, Cattle, Chimpanzee, Elephant, Kangaroo (15, 19, 21, 31, 38)
Medium-Sized Mammals	Fox, Porcupine, Possum, Raccoon, Skunk (34, 63, 64, 66, 75)
Non-Insect Invertebrates	Crab, Lobster, Snail, Spider, Worm (26, 45, 77, 79, 99)
People	Baby, Boy, Girl, Man, Woman (2, 11, 35, 46, 98)
Reptiles	Crocodile, Dinosaur, Lizard, Snake, Turtle (27, 29, 44, 78, 93)
Small Mammals	Hamster, Mouse, Rabbit, Shrew, Squirrel (36, 50, 65, 74, 80)
Trees	Maple, Oak, Palm, Pine, Willow (47, 52, 56, 59, 96)
Vehicles 1	Bicycle, Bus, Motorcycle, Pickup Truck, Train (8, 13, 48, 58, 90)
Vehicles 2	Lawn-mower, Rocket, Streetcar, Tank, Tractor (41, 69, 81, 85, 89)

number of introduced concept drift sub-classes in the data stream are shown on the y-axis. A line at zero signifies that only images the CNN has been trained on have been applied. Whilst a line at 1 indicates all concept drift sub-classes have been applied to the stream. In categorical patterns, Figure 5.5 (a), (b), (c), and (d) show steps, representing the cumulative application of thirty concept drift classes in the data stream. In order to simulate real-world applications more closely, no incorrectly classified instances are removed during testing.

The drift detection mechanism of *DeepStreamEnsemble* is replaced with six distinct drift detection methods which are high performing, widely used and from a range of statistical, windowing and novelty detection methods, each tuned for optimal results. The adaptation aspect of *DeepStreamEnsemble* is substituted with two different CNN adaptation methods. These are successful methods in the Online Domain Incremental (ODI) setting [122]. Each of these methods has been adjusted such that they only receive true labels for the window in which the drift was detected by *DeepStreamEnsemble*. To allow for a direct comparison of the substituted adaptation methods, the drift detection has been simulated so that all methods are adapting to

TABLE 6.3: Sub-class data combinations

Trained Sub-Classes	Novel Sub-Classes
Fashion-MNIST	
0-5	1-2
0-5	1-2-3
0-5	1-2-3-7
0-5	1-2-3-7-8
0-5	1-2-3-7-8-9
0-1-5-7	2-4
0-1-5-7	2-4-8
0-1-5-7	2-4-8-9
0-1-2-5-7-9	6
0-1-2-5-7-9	6-8
CIFAR-10	
0-2	1-8
0-2	1-8-9
0-2	1-8-9-3
0-2	1-8-9-3-4
0-2	1-8-9-3-4-5
0-1-2-3	8-9
0-1-2-3	8-9-4
0-1-2-3	8-9-4-5
0-1-2-3-4-8	9
0-1-2-3-4-8	9-5
CIFAR-100 Scenario 1 - Two novel sub-classes applied	
7-10-11-13-14-16-17-19-20-21-25-29-30-32-33-35-37-39-40-42-43-44-45-48-49-50-51-52-53-55-56-62-63-64-65-67-69-70-77-81	71-97
0-1-2-3-4-5-6-8-9-12-15-22-23-24-26-27-34-36-38-41-47-54-61-71-75-76-80-83-87-89-90-91-92-93-94-95-96-97-98-99	10-52
18-24-28-31-38-46-57-58-59-60-61-66-68-71-72-73-74-75-76-78-79-80-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99	27-75
18-24-31-38-59-60-68-71-73-76-78-82-85-88-89-91-92-93-96-97	52-70
14-16-18-21-25-28-31-37-40-49-60-65-67-68-73-74-77-79-84-86	33-94
13-17-19-21-29-37-42-43-44-45-48-50-62-63-64-65-69-70-77-81	27-75
2-8-9-22-26-46-48-61-79-87	13-40
14-18-55-56-59-72-77-79-81-85	24-45
18-24-66-72-75-82-84-92-94-95	14-30
CIFAR-100 Scenario 2 - Three novel sub-classes per super-class applied	
2-8-9-22-26-46-48-61-79-87	10-11-13-16-28-35-39-40-45-58-77-86-90-98-99
14-18-55-56-59-72-77-79-81-85	4-6-7-24-26-30-41-45-47-52-69-89-95-96-99
18-24-66-72-75-82-84-92-94-95	4-5-6-7-14-20-25-30-34-54-55-62-63-64-70
18-24-31-38-59-60-68-71-73-76-78-82-85-88-89-91-92-93-96-97	1-3-6-7-12-14-15-17-19-21-23-27-29-32-33-37-41-42-43-44-47-49-52-54-56-62-67-69-70-81
14-16-18-21-25-28-31-37-40-49-60-65-67-68-73-74-77-79-84-86	1-5-6-7-9-10-12-15-17-19-20-22-23-24-26-32-33-36-38-39-45-50-61-71-76-80-87-91-94-99
13-17-19-21-29-37-42-43-44-45-48-50-62-63-64-65-69-70-77-81	3-8-12-15-26-27-31-34-36-38-41-54-58-66-68-74-75-76-78-79-80-82-85-88-89-90-92-93-97-99
7-10-11-13-14-16-17-19-20-21-25-29-30-32-33-35-37-39-40-42-43-44-45-48-49-50-51-52-53-55-56-62-63-64-65-67-69-70-77-81	0-1-2-3-4-5-6-8-9-12-15-18-22-23-24-26-27-28-31-34-36-38-41-46-47-54-57-58-59-60-61-66-68-71-72-73-74-75-76-78-79-80-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99
0-1-2-3-4-5-6-8-9-12-15-22-23-24-26-27-34-36-38-41-47-54-61-71-75-76-80-83-87-89-90-91-92-93-94-95-96-97-98-99	7-10-11-13-14-16-17-18-19-20-21-25-28-29-30-31-32-33-35-37-39-40-42-43-44-45-46-48-49-50-51-52-53-55-56-57-58-59-60-62-63-64-65-66-67-68-69-70-72-73-74-77-78-79-81-82-84-85-86-88
18-24-28-31-38-46-57-58-59-60-61-66-68-71-72-73-74-75-76-78-79-80-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99	0-1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-19-20-21-22-23-25-26-27-29-30-32-33-34-35-36-37-39-40-41-42-43-44-45-47-48-49-50-51-52-53-54-55-56-62-63-64-65-67-69-70-77-81

the same instances. The memory buffers are set to the mid-range value applied in the OCL survey paper [122]. The complete *DeepStreamEnsemble* system is contrasted with three other detection and DNN adaptation systems. They are evaluated via accuracy and the intensity of concept drift, where the intensity is the number of sub-classes applied as concept drift. *DeepStreamEnsemble* continues processing images whilst adaptation takes place. The image datasets used in this chapter are relatively small in streaming terms. Therefore, during adaptation the stream processing speed is reduced such that adaptation can occur and remaining data instances are available subsequent to the adaptation. In real world scenarios there would not be a limitation on the amount of streaming data.

1. DDM (Drift Detection Method) [58] is a statistical method, it assumes the binomial distribution and uses the standard deviation to detect drift. The following tuning parameters were set: Min num instances $= 30$ (minimum number of instances so change can be detected, tuned between 10 and 50), warning level=4, out control level $= 2$. The warning level and control level were tuned between 1 and 4.

2. ADWIN (Adaptive Windowing) [17] is a windowing method that summarises information from previous windows and then compares this with summarised information from the current window. It uses sliding windows of variable size and if two windows are found that have distinctly different averages, then the data distribution is deemed to have changed. The following tuning parameters were set: Delta $= 0.5$ after tuning at values of 0.0001, 0.002, 0.05 and 0.5 across all data combinations.

3. HDDMW (Hoeffding Drift Detection Method using the statistical W-test) [54] is a windowing method that summarises information from previous windows and then compares this with summarised information from the current window. It relies on Hoeffding bounds and monitors the data distribution within various time windows by employing probability inequalities rather than the probability distribution function. The method contrasts moving averages to identify drifts and incorporates a forgetting scheme to determine the weight of moving averages in the data stream. The following tuning parameters were

set: Drift confidence=0.001, warning confidence=0.01, lambda option=0.09. Drift confidence and warning confidence were tuned around the values: 0.0001, 0.001, 0.005, 0.1 and lambda option from 0.01 to 0.1 across all data combinations.

4. KSWIN (Kolmogorov–Smirnov Windowing) [152] is a windowing method that summarises information from previous windows and then compares this with summarised information from the current window. It is based on the Kolmogorov statistical test and has no assumption of the underlying data distribution. The following tuning parameters were set: Alpha=0.0001, window size=50, stat size=30. Alpha was tuned between 0.001 and 0.01, window size between 20 and 200, and stat size between 10 and 50 across all data combinations.

5. MINAS (MultIclass learNing Algorithm for data Streams) [50] is a clustering method where each recognised class is represented by a collection of micro-clusters, which provide a statistical summary of the data. In the online phase, new micro-clusters can be established or removed. Unseen instances are either assigned to an existing cluster or designated as unknown, being stored in short-term memory. Once a sufficient number of examples accumulate in short-term memory, they undergo clustering, resulting in the creation of a new set of micro-clusters. Each newly formed micro-cluster is subjected to evaluation, and non-cohesive or unrepresentative ones are eliminated. The valid new micro-clusters are then assessed to determine whether they represent an extension of a recognised class or a novelty pattern. These valid micro-clusters are incorporated into the decision model and utilised in the classification of new examples. The following tuning parameters were set: min short mem trigger=10, min examples cluster=5. Min short mem trigger was tuned between 10 and 100 and min examples cluster was tuned between 1 and 10 across all data combinations.

6. OCDD (One Class Drift Detector) [71] employs a one-class learner featuring a sliding window. The one-class classifier is utilised to gauge the distribution of the emerging concept, determining whether new samples belong to

the existing concept or deviate as outliers. Samples categorised as outliers are recognised as data from the emerging concept. A drift is signalled based on the percentage of outliers identified within the sliding window. This iterative process persists until no further new data is encountered. The following tuning parameters were set: Nu = 0.001, size=10, percent=0.1. Nu was tuned between 0.001 and 0.9, size was tuned between 10 and 1000, and percent was tuned between 0.1 and 0.9 across all data combinations.

DDM is a statistical method, ADWIN, HDDMW and KSWIN are all windowing methods, but use different information to summarise the windows. MINAS is a micro-clustering statistical summary method and OCDD is a one-class learner with a sliding window. These methods provide a range of drift detection techniques. The F1-Score metric with out of sample as the positive class is used as analysed in [81], leading to the following definitions: True positives are unknown images that are correctly classified. False positives are known images that are incorrectly classified. False negatives are unknown images that are identified as known.

The adaptation aspect of *DeepStreamEnsemble* is substituted with two different CNN adaptation methods: (1) ER [31] and (2) MIR [8] augmented with trick RV [25]. These are successful methods in the Online Domain Incremental (ODI) setting from the comprehensive survey [122]. Each of these methods has been adjusted to receive only the true-labelled windows of instances when the drift was detected from *DeepStreamEnsemble*. To allow for a direct comparison of the substituted adaptation methods, the drift detection has been simulated so that all methods are adapting to the same instances. The memory buffers are set to 5000. This is the mid-range value applied in the OCL survey paper [122]. An overview of each of these methods follows:

1. ER (Experience Replay) [190]. See Section 5.3.3 for details of this method.
2. MIR (Maximally Interfered Retrieval) [8]. See Section 5.3.3 for details of this method.

The entire *DeepStreamEnsemble* system is compared to RSB [101], TENT [192] and CPE [195]. They are evaluated via accuracy and the intensity of concept drift, where

the intensity is the number of sub-classes applied as concept drift. *DeepStreamEnsemble* continues processing images whilst adaptation takes place. An overview of each of the combined drift and adaptation methods follows:

1. RSB (Reactive Subspace Buffer). See Section 5.3.3 for details of this method.

2. TENT (Test ENTropy) [192] requires batch normalisation layers after the max pooling layers in the CNN. The normalisation statistics of the batch normalisation layers are adjusted online at each batch to reduce generalisation errors by optimising channel-wise affine transformations to minimise entropy. The following tuning parameters were set: Learning rate $= 0.01$ on a subset of data (two from each data combination for each number of trained classes for class and sub-class data).

3. CPE (CNN based Prototype Ensemble). See Section 5.3.3 for details of this method. Recommended settings from the paper [195] are applied for the training of the CNN: 1000 instances from each class of training data, number of novel classes that are accumulated before CNN retraining = 1000, learning rate = 0.001 [195]. Tuning was performed for the threshold of the prototypes and a value of 3.0 was found to produce superior results.

The system is running on AMD Ryzen Threadripper PRO 3955WX 16-Cores 3.90 GHz, 256 GB RAM with NVIDIA RTX A6000 GPU.

6.4 Experimental Results

In this section, we present our comprehensive experimental analysis, substantiating the effectiveness of both the suggested concept drift detection method and the adaptation approach. The outcomes are scrutinised based on the F1-Score for concept drift detection and accuracy for adaptation and overall comparisons. Timing considerations encompass time per instance and CNN adaptation time.

Table 6.4 displays the average F1-Score and standard deviation for each drift pattern. For concept drift detection, our approach (DSE-DD) demonstrated superior performance over DDM, ADWIN, HDDMW, KSWIN, MINAS, and OCDD across all

drift pattern scenarios, with the exception of CIFAR-10 data. This is because *DeepStreamEnsemble* detects changes in data. It detects when the unknown classes were first introduced but not the whole range of the unknown classes. CIFAR-10 data has more instances per class than CIFAR-100 therefore, in CIFAR-100, drift is detected for more of the range of unknown instances, resulting in higher F1-Scores. The full investigation is included in Appendix D.2. Notably, our method exhibited proficiency in detecting Categorical Gradual and Categorical Reoccurring drift patterns, while MINAS and OCDD exhibited higher efficacy in handling the remaining patterns. MINAS and OCDD are implicit methods and incur an additional processing time of over 20ms per instance compared to our method. The other explicit methods tend to have a high false positive rate, therefore reducing their F1-Score. From Table 5.4, our drift detection method ranks fourth in the timings, exhibiting only 1.05ms difference per instance compared to the top-performing DDM method. However, despite this slight disparity, our approach significantly outperforms the latter concerning F1-Score evaluation. The most time-consuming approaches are the implicit methodologies employed by MINAS and OCDD.

For CNN adaptation, for the Fashion-MNIST dataset, our method (DSE-Adapt) outperformed ER and MIR-RV for all drift patterns except for the Categorical Abrupt pattern. For the CIFAR-10 dataset, our method performed well on Categorical Incremental and Categorical Reoccurring drift patterns, with the accuracy difference between ours and the next best-performing adaptation technique being small. All methods performed less well on the CIFAR-100 dataset. This concurs with the UMAP images in Figure 6.3 which indicate that Fashion-MNIST data has superior separation compared to CIFAR data.

The CIFAR-100 dataset only has 100 images per class, whereas CIFAR-10 and Fashion-MNIST have 1000 test images per class, providing the CNN more samples to train from. Table 5.5 shows that our adaptation method is eight times faster than the leading ODI method of ER, and seven times faster than MIR-RV. Our method also achieves the highest overall adaptation accuracy.

Our system, *DeepStreamEnsemble* has been compared with three other systems: RSB, TENT and CPE. Table 6.5 presents the accuracy following the application of adaptation for each drift pattern. *DeepStreamEnsemble* outperforms the three other

TABLE 6.4: Drift detection module comparison. The average F1-Score for each drift pattern.

	Categorical Abrupt	Temporal Abrupt	Categorical Gradual	Temporal Gradual	Categorical Incremental	Categorical Reoccurring	Temporal Reoccurring
				Fashion-MNIST			
DSE-DD (Ours)	**0.838 (0.209)**	**0.791 (0.222)**	**0.615 (0.308)**	**0.799 (0.235)**	**0.825 (0.173)**	**0.625 (0.186)**	**0.826 (0.188)**
DDM	0.088 (0.111)	0.077 (0.090)	0.084 (0.094)	0.080 (0.082)	0.159 (0.111)	0.160 (0.132)	0.103 (0.112)
ADWIN	0.366 (0.194)	0.384 (0.206)	0.470 (0.232)	0.421 (0.202)	0.430 (0.184)	0.411 (0.157)	0.403 (0.182)
HDDMW	0.079 (0.129)	0.044 (0.052)	0.052 (0.095)	0.033 (0.050)	0.087 (0.069)	0.127 (0.143)	0.050 (0.063)
KSWIN	0.246 (0.33)	0.251 (0.314)	0.395 (0.308)	0.209 (0.3)	0.341 (0.316)	0.294 (0.228)	0.198 (0.261)
MINAS	0.633 (0.198)	0.632 (0.198)	0.381 (0.185)	0.632 (0.198)	0.633 (0.199)	0.475 (0.092)	0.632 (0.198)
OCDD	0.632 (0.198)	0.632 (0.198)	0.381 (0.185)	0.632 (0.198)	0.632 (0.198)	0.475 (0.092)	0.632 (0.198)
				CIFAR-10			
DSE-DD (Ours)	0.352 (0.300)	0.411 (0.335)	**0.564 (0.239)**	0.395 (0.270)	0.563 (0.245)	**0.521 (0.171)**	0.379 (0.254)
DDM	0.147 (0.089)	0.143 (0.108)	0.174 (0.108)	0.134 (0.108)	0.225 (0.132)	0.203 (0.105)	0.172 (0.100)
ADWIN	0.171 (0.153)	0.168 (0.116)	0.290 (0.239)	0.231 (0.158)	0.222 (0.149)	0.217 (0.126)	0.215 (0.133)
HDDMW	0.237 (0.122)	0.207 (0.086)	0.230 (0.216)	0.191 (0.119)	0.321 (0.121)	0.323 (0.114)	0.209 (0.123)
KSWIN	0.044 (0.051)	0.09 (0.130)	0.224 (0.157)	0.083 (0.094)	0.243 (0.182)	0.142 (0.124)	0.098 (0.113)
MINAS	**0.632 (0.204)**	**0.633 (0.199)**	0.384 (0.187)	**0.633 (0.197)**	0.63 (0.2)	0.474 (0.089)	**0.635 (0.199)**
OCDD	**0.632 (0.198)**	0.632 (0.198)	0.381 (0.185)	0.632 (0.198)	**0.632 (0.198)**	0.475 (0.092)	0.632 (0.198)
				CIFAR-100			
DSE-DD (Ours)	**0.584 (0.226)**	**0.626 (0.281)**	**0.580 (0.298)**	**0.564 (0.211)**	**0.658 (0.244)**	**0.667 (0.192)**	**0.673 (0.237)**
DDM	0.236 (0.219)	0.035 (0.087)	0.317 (0.395)	0.186 (0.194)	0.222 (0.222)	0.206 (0.303)	0.101 (0.231)
ADWIN	0.273 (0.179)	0.275 (0.186)	0.251 (0.160)	0.271 (0.194)	0.313 (0.202)	0.320 (0.185)	0.340 (0.217)
HDDMW	0.431 (0.308)	0.391 (0.313)	0.460 (0.310)	0.451 (0.259)	0.412 (0.300)	0.517 (0.268)	0.460 (0.314)
KSWIN	0.233 (0.137)	0.234 (0.188)	0.326 (0.214)	0.19 (0.104)	0.352 (0.155)	0.462 (0.193)	0.347 (0.198)
MINAS	0.506 (0.286)	0.506 (0.286)	0.291 (0.206)	0.506 (0.287)	0.506 (0.286)	0.379 (0.172)	0.506 (0.286)
OCDD	0.506 (0.286)	0.506 (0.286)	0.29 (0.205)	0.506 (0.286)	0.506 (0.286)	0.379 (0.172)	0.506 (0.286)

Legend: The highest values are in bold. The standard deviation is in brackets.

methods in all drift scenarios. Figure 6.4 shows how the accuracy changes with the intensity of concept drift for a selection of scenarios. For *DeepStreamEnsemble*, as the intensity of concept drift increases, there is a small decline in accuracy; nevertheless, it remains superior to other methods. This trend is particularly pronounced in CIFAR-100 data, due to a higher number of applied sub-classes. All methods exhibit

improved performance on the Fashion-MNIST data. Conversely, the Categorical-Incremental drift pattern poses a more challenging scenario for these systems, as can be seen by the slightly poorer performance across all methods. The Wilcoxon Signed-Rank test is employed to assess the distinction in accuracies among methods. Analyzing the difference between the accuracy of *DeepStreamEnsemble* and the next highest-performing method, RSB, across 266 tested data combinations and patterns, has a p-value below 0.00001. This value, being less than the 0.05 significance level, supports the acceptance of the alternative hypothesis that the true location shift is not equal to 0. Consequently, *DeepStreamEnsemble* demonstrates a significant superiority over RSB.

TABLE 6.5: Overall system comparison. The average accuracy after adaptation has been applied for each drift pattern.

	Categorical Abrupt	Temporal Abrupt	Categorical Gradual	Temporal Gradual	Categorical Incremental	Categorical Reoccurring	Temporal Reoccurring
			Fashion-MNIST				
DeepStreamEnsemble (Ours)	**0.970 (0.037)**	**0.971 (0.037)**	**0.950 (0.060)**	**0.977 (0.028)**	**0.963 (0.043)**	**0.977 (0.025)**	**0.983 (0.018)**
RSB	0.930 (0.078)	0.94 (0.081)	0.910 (0.071)	0.931 (0.067)	0.930 (0.071)	0.927 (0.051)	0.926 (0.024)
TENT	0.761 (0.169)	0.764 (0.169)	0.769 (0.012)	0.819 (0.128)	0.547 (0.022)	0.815 (0.040)	0.865 (0.091)
CPE	0.920 (0.092)	0.946 (0.049)	0.927 (0.067)	0.943 (0.043)	0.872 (0.125)	0.944 (0.045)	0.933 (0.053)
			CIFAR-10				
DeepStreamEnsemble (Ours)	**0.947 (0.022)**	**0.949 (0.022)**	**0.942 (0.011)**	**0.948 (0.016)**	**0.953 (0.016)**	**0.944 (0.015)**	**0.948 (0.015)**
RSB	0.930 (0.030)	0.934 (0.032)	0.879 (0.059)	0.918 (0.039)	0.927 (0.037)	0.891 (0.043)	0.921 (0.026)
TENT	0.706 (0.173)	0.705 (0.173)	0.722 (0.026)	0.757 (0.131)	0.506 (0.024)	0.757 (0.045)	0.793 (0.107)
CPE	0.707 (0.057)	0.691 (0.105)	0.708 (0.080)	0.731 (0.056)	0.655 (0.034)	0.733 (0.037)	0.754 (0.036)
			CIFAR-100				
DeepStreamEnsemble (Ours)	**0.574 (0.180)**	**0.577 (0.191)**	**0.670 (0.154)**	**0.631 (0.170)**	**0.559 (0.182)**	**0.681 (0.146)**	**0.670 (0.155)**
RSB	0.508 (0.106)	0.517 (0.101)	0.533 (0.035)	0.559 (0.08)	0.516 (0.167)	0.584 (0.075)	0.592 (0.079)
TENT	0.422 (0.193)	0.429 (0.194)	0.501 (0.158)	0.479 (0.181)	0.277 (0.195)	0.497 (0.148)	0.513 (0.157)
CPE	0.306 (0.199)	0.310 (0.177)	0.330 (0.148)	0.321 (0.161)	0.311 (0.191)	0.337 (0.149)	0.335 (0.154)

Legend: Standard deviation in brackets. Highest values in bold.

The adaptation time cannot be compared directly for the combined concept drift detection and adaptation systems as the comparison methods do not explicitly adapt.

TABLE 6.6: Drift detection time (ms) and rank.

	Time (ms)	Rank
DSE-DD (Ours)	64.56 (31.5)	4
DDM	**63.51 (25.2)**	1
ADWIN	63.64 (24.9)	2
HDDMW	63.85 (25.6)	3
KSWIN	64.96 (26.7)	5
MINAS	86.88 (38.78)	7
OCDD	78.91 (26.2)	6

Legend: Standard deviation in brackets. Lowest values in bold for timings.

TABLE 6.7: Adaptation Time and accuracy after adaptation for ODI methods.

	Adaptation Time (s)	Accuracy
DSE-Adapt (Ours)	**20.546 (18.998)**	**0.854 (0.158)**
ER	164.061 (414.814)	0.841 (0.131)
MIR-RV	145.735 (246.299)	0.784 (0.159)

Legend: Standard deviation in brackets. Lowest values in bold for timings. Highest values in bold for accuracies.

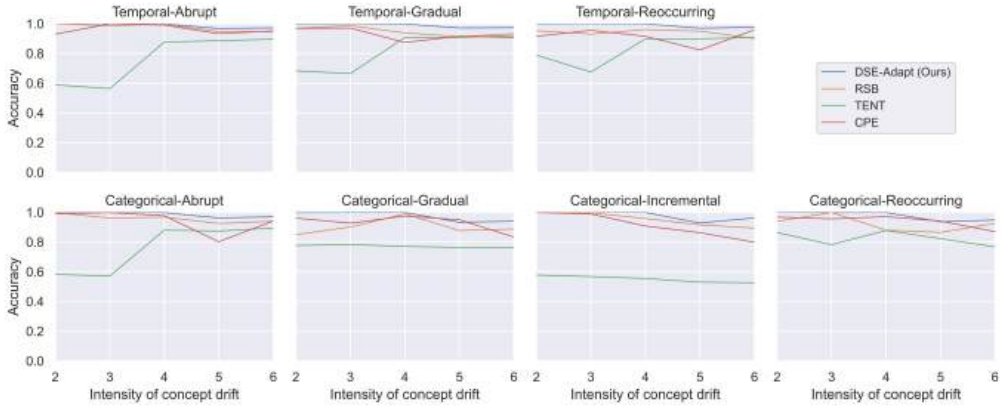
TABLE 6.8: Time per instance and accuracy after adaptation for combined concept drift detection and adaptation methods. DSE is *DeepStreamEnsemble*.

	Time per instance (ms)	Accuracy
DSE (Ours)	18.355 (17.842)	**0.846 (0.019)**
RSB	18.263 (10.605)	0.795 (0.184)
TENT	**5.229 (5.208)**	0.638 (0.167)
CPE	344.313 (321.603)	0.653 (0.021)

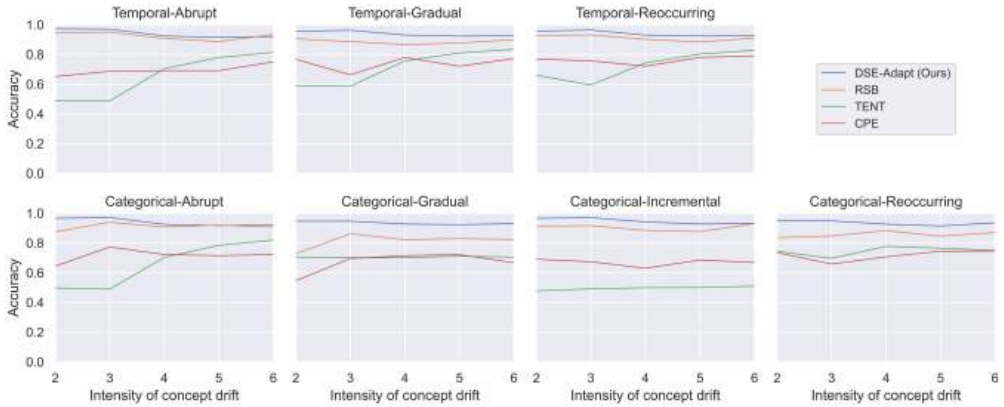
Legend: Standard deviation in brackets. Lowest values in bold for timings. Highest values in bold for accuracies.

As *DeepStreamEnsemble* adapts offline, the time per instance can be compared. Table 6.8 shows the time per instance in milliseconds. *DeepStreamEnsemble* takes a similar amount of time to process an instance as the next best performing method, RSB does. However, our system outperforms RSB on accuracy. TENT is the fastest but differs in the way it adapts as it employs batch normalisation layers which means the CNN architecture has to be updated and retrained before use. TENT demonstrates the lowest accuracy. CPE has the longest duration and presents a low accuracy in comparison to ours. From Table 6.4, our method outperforms other leading drift detection methods by between 8% and 46% on F1-Score. From Table 6.5 our method

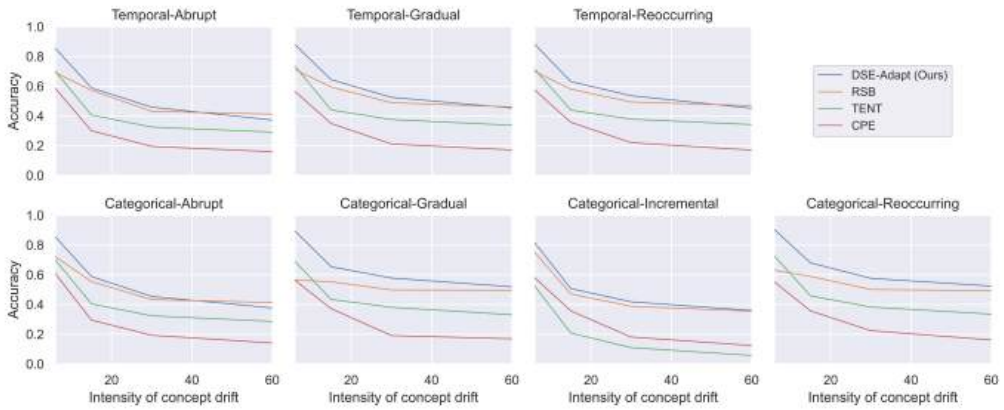
outperforms leading combined drift detection and adaptation methods by between 5% and 20% on accuracy. Overall, *DeepStreamEnsemble* provides the highest average accuracy with timings comparable to the next highest performing method. The initial CNN accuracies are included in Appendix D.1



(a) Fashion-MNIST 2 trained classes



(b) CIFAR-10 2 trained classes



(c) CIFAR-100 averaged accuracies for 4, 10, 20 and 40 trained classes

FIGURE 6.4: Intensity of concept drift against accuracy

6.5 Summary

In this chapter, we proposed a method called *DeepStreamEnsemble* which detects concept drift and adapts a DNN to this. A DNN activation based image descriptor method was used as the basis to produce activation classification footprints for each convolutional block of a CNN and supplied to an ensemble of Hoeffding tree classifiers. The output of this ensemble was input into our novel concept drift detection method which monitors the accuracy volatility between the DNN predictions and the Hoeffding tree ensemble. The DNN adaptation uses a partial network update and an activation ensemble to assist in a memory based method to mitigate catastrophic forgetting.

To evaluate the proposed method, a set of experiments were conducted with three benchmark image datasets. Super-classes were selected for the system to be trained on and sub-classes of the super-classes were withheld in order to be applied as concept drift via novel sub-classes in seven different patterns. These data patterns were applied in various class combinations to the commonly used VGG16 DNN. *DeepStreamEnsemble* was compared to six leading concept drift detection methods, two leading ODI DNN adaptation methods and three combined concept drift detection and DNN adaptation methods.

The results showed that our drift detection method (DSE-DD) overall outperforms all other methods in terms of accuracy. For drift detection, our method excels across all patterns for Fashion-MNIST and CIFAR-100 and across the Categorical Gradual and Categorical Reoccurring patterns for CIFAR-10. On CIFAR-10, our method excels above all other explicit detection methods, with the implicit novelty detection methods achieving greater accuracy, but with increased duration. For adaptation, our method (DSE-Adapt) excels for Fashion-MNIST data and for Categorical Incremental and Categorical Reoccurring patterns for CIFAR-10. However, it exhibits reduced performance on CIFAR-100. A contributory factor to CIFAR-100 lower accuracies is that it does not have as many instances per class. This indicates that the optimum CNN adaptation method may vary depending on the dataset.

In this chapter, by presenting *DeepStreamEnsemble*, we have shown that our method is able to detect concept drift using DNN activations and adapt a DNN to this and

have therefore addressed the final objective from Section 1.5. The next chapter provides a brief synopsis of the work presented in this thesis, draws conclusions and identifies important directions for future research.

Chapter 7

Conclusion

This thesis studies discrepancy detection and DNN adaptation in streaming scenarios. Firstly addressing outlier discrepancy detection in an open-set scenario, then addressing concept evolution and concept drift discrepancies via drift patterns with DNN adaptation. The aims and objectives identified in this thesis are concerned with reviewing the literature for the state-of-the-art methods and proposing novel methods to improve upon the existing approaches and expand upon the existing analysis.

The aims of this thesis have been fulfilled by addressing the objectives. The first aim is: **Detecting data discrepancies in DNNs via neural activations for data streams** and has been addressed via objectives 1, 2, 3 and 4 from Section 1.5. The second aim is: **Adapting DNNs in the presence of data discrepancies for data streams** and has been addressed via objectives 1, 3 and 4 from Section 1.5. The following paragraphs give a summary of the chapters, demonstrating how they have achieved the objectives.

Chapter 1 provides an introduction to data discrepancies, DNNs, streaming classification and the main challenges in these areas of research. The proposal of employing DNN activations as an alternative representation of the input data is also posed, along with the definition of the problems, aims and objectives of this thesis.

Chapter 2 addresses the first objective from Section 1.5, which is to **critically review data discrepancy detection and DNN adaptation methods**. The focus of this review is upon CNNs as the DNN, and images as the input data. The topics of open-set recognition and data discrepancy detection with adaptation for streaming images is reviewed and analysed. A taxonomy of these subjects is given in Section

2.5. From this analysis, it becomes clear that there is a lack of discrepancy detection and CNN adaptation methods for streaming images. Solutions that do exist often use implicit clustering drift detection where a number of drift instances must arrive before declaring drift. This is in contrast to explicit drift detection where i.e. statistical tests are performed to provide a more immediate response. We saw no reason for the limited number of explicit discrepancy detection methods other than the lack of focus on discrepancy detection and CNN adaptation in streaming scenarios. DNN adaptation is not widely studied in the streaming literature as compared to traditional machine learning models as DNNs take longer to adapt, require more data, suffer from class imbalance, and the adaptation of DNNs causes catastrophic forgetting.

The OCL field offers many DNN adaptation techniques, our review focused on CNNs and it was found that little analysis had been performed with respect to OCL CNN adaptation solutions in the streaming literature. Furthermore, OCL solutions had not been analysed in terms of drift patterns.

High dimensional data is often transformed into a different representation to analyse and use it. Investigation into the use of DNN activations in Sections 2.2 and 3.4.1 revealed that some methods in open-set recognition and other related detection fields have successfully employed activations. Section 2.6 discussed that they can provide more information than the original input data as CNNs act as natural feature extractors. From this, we coined our own term of 'activation classification footprints' that represent the input images via unique neural activation patterns, presenting information that is not discernible in the original image.

For the concept drift detection and CNN adaptation methods, it was found that concept drift is typically applied in the form of new background, blur, noise, illumination and occlusion. These are all synthetically applied drifts and do not represent real-world situations. It was also identified that as DNNs can take an extended period to train for real-world systems. It would be advantageous to have a system that could be applied to an existing DNN without completely retraining it. Thus transforming a standard DNN into a discrepancy detecting, adapting DNN. These identified gaps made it clear that a new approach for discrepancy detection and DNN adaptation for streaming data was required with the following attributes: (1)

Transformation of the data into a different representation; (2) explicit concept evolution detection; (3) explicit concept drift detection; (4) classifying DNN adaptation in a streaming environment; (5) transforming a pretrained standard DNN into a discrepancy detecting adapting DNN; (6) analysis with respect to concept drift patterns; and (7) analysis with respect to novel sub-class real images arising within a super-class. Hence, this thesis focused on achieving these attributes.

Chapter 3 reviews the required background and theoretical explanation of the algorithms employed in the novel methods proposed in this thesis. The second objective of this thesis, which is to **design, develop and evaluate an Open-Set (outlier) discrepancy detection method** is addressed in Chapter 4 by proposing the *DeepStreamOS* method for outlier discrepancy detection. This method provides the groundwork for Chapter 5 where the concept evolution discrepancy detection approach is introduced and the system is expanded to include DNN adaptation. This results in a novel method termed *AdaDeepStream* and achieves the third objective of this thesis, which is to **design, develop and evaluate a concept evolution discrepancy detection and DNN adaptation method**. Finally, Chapter 6 tackles the more complex problem of concept drift discrepancy detection and DNN adaptation, producing a novel method called *DeepStreamEnsemble*. This addresses the fourth objective which is to **design, develop and evaluate a concept drift discrepancy detection and DNN adaptation method**. The following section summarises these contributions with attention to the identified attributes.

7.1 Summary of Contributions

In Chapter 4, *DeepStreamOS*: Open-Set classification in DNNs, we proposed a method for addressing outlier discrepancy detection. This involved extracting activations from within the CNN into a different representation to the input data to provide activation classification footprints and using these to explicitly detect outliers at the instance level. Novel class outliers and novel sub-class outliers were applied. Experiments have shown that fast open-set classification is achievable using *DeepStreamOS*. This was proven using two different types of data, one where novel classes are introduced, and one where new sub-types of images are introduced into existing classes.

Our method explicitly detected outliers at the instance level. We successfully detected both types of data, which indicates that it is more suited to deep neural networks with a larger number of layers. *DeepStreamOS* was compared to OpenMax and EVM where, in terms of effectiveness via the F1-Score, it outperformed OpenMax in all scenarios and outperformed EVM on Fashion-MNIST data, overall exceeding EVM by 2% and OpenMax by 30%. *DeepStreamOS* considerably outperformed both OpenMax and EVM in terms of speed. *DeepStreamOS* is 5 times faster than OpenMax and 100 times faster than EVM with scope to up-scale to more classes without affecting the time to process an instance. We have applied our method to pretrained CNNs and image data in this paper, however the inference method is not specific to CNNs or images and could be applied to other types of deep neural networks and data.

In Chapter 5 we introduced *AdaDeepStream*: Streaming DNN adaptation to concept evolution. We proposed a method for addressing concept evolution discrepancy detection. This contribution also places CNN adaptation methods from the OCI field in the Concept Evolution field. This involved extracting activations from within the CNN to provide activation classification footprints. This was investigated for two different activation extraction methods (JSDL and DS-CBIR), where JSDL built upon methods used in *DeepStreamOS*. Explicit concept evolution detection was performed using a well established drift detector and CNN adaptation with assistance from the activation classification footprints. Our *AdaDeepStream* adaptation method overall outperforms other leading OCI adaptation methods on accuracy by 27% when placed in the concept evolution scenario with limited true-labelled data. *AdaDeepStream* also outperforms other combined drift detection and CNN adaptation systems (RSB and CPE) in accuracy by 24%. From the two novel activation reduction methods presented, DS-CBIR produces more stable results than JSDL. *AdaDeepStream* performs well on all concept evolution patterns whilst other methods show an improvement on the Temporal-Reoccurring pattern, possibly due to a more diverse range of classes arriving in the drift detection windows. Compared to the other methods, *AdaDeepStream* also performs well on data with less intra-class cohesion and inter-class separation. However, it is less stable compared to the implicit drift detection comparison methods. This could indicate that the explicit drift

detection and the use of CNN predictions in the drift detection method has an effect on the consistency of our method. Therefore, a good drift detection method is important. The speed of inference and adaptation of *AdaDeepStream* is comparable with the fastest adaptation methods. *AdaDeepStream* is a memory-based CNN adaptation method as is the next best performing method, ER. In this scenario of small datasets, it indicates that the simple memory-based methods achieve good results.

In Chapter 6 we introduced *DeepStreamEnsemble*: Streaming DNN adaptation to concept drift. We proposed a method for addressing concept drift discrepancy detection. This involved extracting activations from within the CNN to provide activation classification footprints for each convolutional block of the CNN and applied to an ensemble of classifiers. This is in contrast to *AdaDeepStream*, where there is one classifier in total. Explicit concept evolution detection was performed using our novel accuracy volatility drift detection method. CNN adaptation was achieved via a memory-based method and assistance from the activation classification footprints. Other bespoke drift detection and adaptation mechanisms may be substituted and can be applied to existing pretrained CNNs to extend their functionality into an autonomously adapting CNN to concept drift. Thus providing a flexible system where the detection and adaptation components can be individually substituted. A thorough analysis has been performed in a concept drift pattern scenario, with image data and CNNs. Drift is applied as evolving sub-classes as opposed to the usual concept drift scenario of noise, blurring or occlusion. *DeepStreamEnsemble* overall outperforms other leading methods of detecting concept drift by 8% on the F1-Score, and other leading methods of CNN adaptation by 5% on accuracy. It is seven times faster than other ODI CNN adaptation methods and overall exceeds all adaptation methods on accuracy. For overall comparison with CNN image drift detection and adapting systems, our method outperforms in all drift scenarios. Via our substitution of individual extraction and adaptation mechanisms, we have shown that there could be potential performance gains in using our novel unsupervised drift detection system with other CNN adaptation methods however, the adaptation will not be as fast.

All three contributing methods operate on pretrained CNNs. *DeepStreamOS* does

not employ a specialised output layer and both the *AdaDeepStream* and *DeepStreamEnsemble* methods perform CNN adaptation which does not require a specialised architecture or loss function. Thus, to apply these methods requires no retraining of the image classifying CNN.

7.2 Future Directions

The novel methods presented in this thesis pave the way for future research in the streaming high dimensional data discrepancy detection and DNN adaptation field. Recommendations include, but are not limited to the following: (1) Further explorations into measurements between the layers of deep neural networks; (2) apply to other types of deep neural networks; (3) apply to different real-world domains; and (4) providing mechanisms to assist in the true labelling of samples.

1. **Further explorations into measurements between the layers of deep neural networks**: Two of our methods used the statistical similarity measure of JS-Divergence between layers. Alternatively, there are other similarity methods that have been investigated specifically for comparing representations between neural network layers [100].

2. **Apply to other types of deep neural networks**: Our JS-Divergence based activation reduction methods use a similarity measure between hidden layers of the DNN. This type of measurement has the potential to be network agnostic. The more successful CBIR descriptor-based activation reduction method requires more knowledge about the network, and although specifically aimed at CNNs, it is promising to apply a modified version of this to other types of DNNs to determine the most activated neurons and project them back into the preceding layer to extract the important neurons. One recent type of neural network is the Vision Transformer (ViT) [48]. They are used for image recognition amongst other tasks. The ViT represents an input image as a sequential series of image patches and can learn high-quality intermediate representations [153], making it an interesting candidate for our activation reduction methods.

3. **Apply to different real-world domains**: Apply to other types of different real-world domains such as x-ray datasets with novel or evolving diseases i.e. COVID-19 [39], histology [9] and automotive vision [62].

4. **Providing mechanisms to assist in the automatic labelling of novel classes**: Our proposed methods only require true labels of the initial novel classes and sub-classes that cause the change detection. This is much less than most systems. The true labelling of samples is time-consuming and usually requires the use of a domain-expert. To automatically label and generate new samples to assist in mitigating class imbalance would provide a more practical system [102], for instance, using methods that leverage generative adversarial networks [211, 193].

7.3 Concluding Remarks

In conclusion, data discrepancy detection and adaptation in DNNs in streaming scenarios is an important and ongoing area of research. The main challenges in this field are fast discrepancy detection, efficiently adapting DNNs when they suffer from class imbalance, catastrophic forgetting and DNNs requirement for large amounts of training data. This thesis progresses this research area and offers a unique perspective by proposing the use of DNN activations in streaming machine learning models for discrepancy detection and DNN adaptation. This is a step forward towards realising fully adaptive continuous deep learning systems.

Appendix A

Drift Detector Experiments

TABLE A.1: Experimental Results for MobileNet DNN, Extremely Fast Decision Tree Methods

DNN	Dataset	Drift	Data Setup	DSCD exfastadwin (Ours)	DSCD exfastddm (Ours)	DSCD exfasteddm (Ours)	DSCD exfasthddma (Ours)	DSCD exfasthddmw (Ours)	DSCD exfastkswin (Ours)
mobilenet	fashion	cat-abr	M1	0	0.919	0.02	0	0	0
mobilenet	fashion	cat-abr	M3	0.039	0.095	0.675	0	0	0.02
mobilenet	fashion	cat-abr	M2	0	1	0	0	0	0
mobilenet	fashion	cat-abr	M4	0	1	0.02	0	0	0
mobilenet	fashion	tem-abr	M1	0	0.485	0.02	0	0	0
mobilenet	fashion	tem-abr	M3	0.039	1	0.773	0.81	0	0
mobilenet	fashion	tem-abr	M2	0	0.039	0	0	0.131	0
mobilenet	fashion	tem-abr	M4	0	1	0	0	0	0
mobilenet	fashion	cat-gra	M1	0	0.246	0.958	0	0	0
mobilenet	fashion	cat-gra	M3	0	1	0.02	1	0	0
mobilenet	fashion	cat-gra	M2	0	0	0	0	0	0
mobilenet	fashion	cat-gra	M4	0	0.974	0.02	0	0	0
mobilenet	fashion	tem-gra	M1	0	1	0.02	0	0	0
mobilenet	fashion	tem-gra	M3	0	0.507	0.788	0	0	0
mobilenet	fashion	tem-gra	M2	0	0.995	0	0	0	0
mobilenet	fashion	tem-gra	M4	0	0.625	0.02	0	0	0
mobilenet	fashion	cat-inc	M1	0	1	0.02	0	0	0
mobilenet	fashion	cat-inc	M3	0.02	1	1	0	0	0
mobilenet	fashion	cat-inc	M2	0	0.039	0.02	0	0	0
mobilenet	fashion	cat-inc	M4	0	1	0.02	0	0	0
mobilenet	fashion	tem-out	M1	0	0.611	0.639	0	0	0
mobilenet	fashion	tem-out	M3	0	0	0.043	0	0	0
mobilenet	fashion	tem-out	M2	0	0.582	0	0	0	0
mobilenet	fashion	tem-out	M4	0	0.235	0.036	0	0	0
mobilenet	fashion	cat-reo	M1	0	0.684	0.485	0	0	0
mobilenet	fashion	cat-reo	M3	0.02	0.883	1	0	0	0
mobilenet	fashion	cat-reo	M2	0	0.039	0	0	0	0
mobilenet	fashion	cat-reo	M4	0	1	0.02	0	0	0
mobilenet	fashion	tem-reo	M1	0	1	0.02	0	0	0
mobilenet	fashion	tem-reo	M3	0	0.99	1	0	0	0
mobilenet	fashion	tem-reo	M2	0	0.039	0	0	0	0
mobilenet	fashion	tem-reo	M4	0	1	0.02	0	0.276	0
mobilenet	CIFAR-10	cat-abr	C1	0	0	1	0.276	0.113	0.02
mobilenet	CIFAR-10	cat-abr	C2	0	0	1	1	0	0
mobilenet	CIFAR-10	cat-abr	C3	0	1	1	0	0.077	0.02
mobilenet	CIFAR-10	tem-abr	C1	0	0	1	0	0.165	0.02
mobilenet	CIFAR-10	tem-abr	C2	0.02	0	1	0.98	0.291	0.039
mobilenet	CIFAR-10	tem-abr	C3	0	1	0.165	0	0.214	0
mobilenet	CIFAR-10	cat-gra	C1	0.039	0	0.87	0.909	0.095	0.02
mobilenet	CIFAR-10	cat-gra	C2	0.02	1	1	1	0.276	0
mobilenet	CIFAR-10	cat-gra	C3	0.04	0.995	1	0.077	0.246	0.02
mobilenet	CIFAR-10	tem-gra	C1	0	0	0.889	0	0.198	0.02
mobilenet	CIFAR-10	tem-gra	C2	0	0.059	1	0	0	0.02
mobilenet	CIFAR-10	tem-gra	C3	0.02	1	0.81	0.857	0.183	0
mobilenet	CIFAR-10	cat-inc	C1	0	1	0.693	0	0.319	0.02
mobilenet	CIFAR-10	cat-inc	C2	0.039	1	0.425	0.462	0	0.02
mobilenet	CIFAR-10	cat-inc	C3	0.058	0.182	0.246	0.413	0	0
mobilenet	CIFAR-10	tem-out	C1	0	0.385	0.075	0.073	0.167	0.042
mobilenet	CIFAR-10	tem-out	C2	0	0.771	0.74	0	0.23	0
mobilenet	CIFAR-10	tem-out	C3	0	0.438	0.723	0	0.192	0
mobilenet	CIFAR-10	cat-reo	C1	0.02	1	0.667	0	0	0
mobilenet	CIFAR-10	cat-reo	C2	0.058	0.901	0.773	1	0.261	0.02
mobilenet	CIFAR-10	cat-reo	C3	0	0.936	0.658	0	0.058	0.02
mobilenet	CIFAR-10	tem-reo	C1	0	1	1	0	0.095	0
mobilenet	CIFAR-10	tem-reo	C2	0.02	1	0.895	0	0	0
mobilenet	CIFAR-10	tem-reo	C3	0	0	0	0	0	0

TABLE A.2: Experimental Results for VGG16 DNN, Extremely Fast Decision Tree Methods

DNN	Dataset	Drift	Data Setup	DSCD exfastadwin (Ours)	DSCD exfastddm (Ours)	DSCD exfasteddm (Ours)	DSCD exfasthddma (Ours)	DSCD exfasthddmw (Ours)	DSCD exfastkswin (Ours)
vgg16	fashion	cat-abr	M1	0	0	0	0	0	0
vgg16	fashion	cat-abr	M3	0	0.02	0.02	0	0	0
vgg16	fashion	cat-abr	M2	0.039	0.611	1	0	0.058	0.02
vgg16	fashion	cat-abr	M4	0	0.02	0	0	0	0
vgg16	fashion	tem-abr	M1	0	0	0	0	0	0
vgg16	fashion	tem-abr	M3	0.02	1	0.02	0	0	0
vgg16	fashion	tem-abr	M2	0	0.02	1	1	0	0
vgg16	fashion	tem-abr	M4	0	0.02	0	0	0	0
vgg16	fashion	cat-gra	M1	0	0	0	0	0	0
vgg16	fashion	cat-gra	M3	0	0.496	0.02	0	0	0
vgg16	fashion	cat-gra	M2	0.04	0.907	0	0.286	0	0
vgg16	fashion	cat-gra	M4	0	0.336	0	0	0	0
vgg16	fashion	tem-gra	M1	0	0	0	0	0	0
vgg16	fashion	tem-gra	M3	0	1	0.02	0	0	0
vgg16	fashion	tem-gra	M2	0	0.305	0	0	0	0
vgg16	fashion	tem-gra	M4	0	0.02	0	0	0	0
vgg16	fashion	cat-inc	M1	0	0.02	0.02	0	0	0
vgg16	fashion	cat-inc	M3	0	0.742	0.02	0	0	0
vgg16	fashion	cat-inc	M2	0	0.387	1	0.077	0	0.02
vgg16	fashion	cat-inc	M4	0	0.02	0	0	0	0
vgg16	fashion	tem-out	M1	0	0	0	0	0	0
vgg16	fashion	tem-out	M3	0	0.675	0.675	0	0	0
vgg16	fashion	tem-out	M2	0	0.71	0.569	0	0	0
vgg16	fashion	tem-out	M4	0	0.05	0	0	0	0
vgg16	fashion	cat-reo	M1	0	0	1	0	0	0
vgg16	fashion	cat-reo	M3	0	0.907	0	0	0	0
vgg16	fashion	cat-reo	M2	0.039	0.964	1	0.276	0.198	0.02
vgg16	fashion	cat-reo	M4	0	0.02	0	0	0	0
vgg16	fashion	tem-reo	M1	0	0	0	0	0	0
vgg16	fashion	tem-reo	M3	0	1	0.02	0	0	0
vgg16	fashion	tem-reo	M2	0	0.23	1	0	0	0
vgg16	fashion	tem-reo	M4	0	0	0	0	0	0
vgg16	CIFAR-10	cat-abr	C1	0.02	1	1	0.919	0.261	0.02
vgg16	CIFAR-10	cat-abr	C2	0	0	1	0	0	0
vgg16	CIFAR-10	cat-abr	C3	0.039	0	0	0.387	0.148	0.039
vgg16	CIFAR-10	tem-abr	C1	0.02	1	0.824	0	0.058	0.02
vgg16	CIFAR-10	tem-abr	C2	0	1	1	0	0	0
vgg16	CIFAR-10	tem-abr	C3	0.058	0.718	1	0	0.182	0
vgg16	CIFAR-10	cat-gra	C1	0.02	0.995	1	0.835	0.291	0.02
vgg16	CIFAR-10	cat-gra	C2	0	1	1	0	0.182	0.02
vgg16	CIFAR-10	cat-gra	C3	0.04	0.83	1	0	0	0.02
vgg16	CIFAR-10	tem-gra	C1	0	0	1	0	0.276	0.02
vgg16	CIFAR-10	tem-gra	C2	0	1	0.864	0	0	0
vgg16	CIFAR-10	tem-gra	C3	0	1	1	0.131	0.248	0.02
vgg16	CIFAR-10	cat-inc	C1	0.039	1	1	0.374	0.095	0
vgg16	CIFAR-10	cat-inc	C2	0	0.98	0.02	0	0	0.02
vgg16	CIFAR-10	cat-inc	C3	0	0	0.844	0.261	0.077	0.02
vgg16	CIFAR-10	tem-out	C1	0	0	0.808	0	0.167	0.039
vgg16	CIFAR-10	tem-out	C2	0	0.732	0.756	0	0	0
vgg16	CIFAR-10	tem-out	C3	0	0.756	0.638	0	0.281	0.039
vgg16	CIFAR-10	cat-reo	C1	0	1	0.582	0.291	0.361	0.02
vgg16	CIFAR-10	cat-reo	C2	0	0	0.995	0	0.148	0.02
vgg16	CIFAR-10	cat-reo	C3	0.02	0	0	0	0.261	0.02
vgg16	CIFAR-10	tem-reo	C1	0	1	1	0.925	0	0
vgg16	CIFAR-10	tem-reo	C2	0	0	1	0.039	0	0
vgg16	CIFAR-10	tem-reo	C3	0.039	1	0	0.621	0.246	0.02

TABLE A.3: Experimental Results for MobileNet DNN, Hoeffding Decision Tree Methods

DNN	Dataset	Drift	Data Setup	DSCD hoeffadwin (Ours)	DSCD hoeffddm (Ours)	DSCD hoeffeddm (Ours)	DSCD hoeffhddma (Ours)	DSCD hoeffhddmw (Ours)	DSCD hoeffkswin (Ours)
mobilenet	fashion	cat-abr	M1	0	1	0.658	0.925	0	0
mobilenet	fashion	cat-abr	M3	0	0.4	0.02	0	0	0
mobilenet	fashion	cat-abr	M2	0	0.039	0	0	0	0
mobilenet	fashion	cat-abr	M4	0	1	0.02	0	0	0
mobilenet	fashion	tem-abr	M1	0	0.876	0.02	0	0	0
mobilenet	fashion	tem-abr	M3	0	1	0.947	0.93	0.077	0
mobilenet	fashion	tem-abr	M2	0	0.02	0	0	0	0
mobilenet	fashion	tem-abr	M4	0	0.87	0.02	0	0	0
mobilenet	fashion	cat-gra	M1	0.039	1	1	0.995	0	0
mobilenet	fashion	cat-gra	M3	0	1	0.817	0	0	0
mobilenet	fashion	cat-gra	M2	0	0	0	0	0	0
mobilenet	fashion	cat-gra	M4	0	0.078	0.02	0	0	0
mobilenet	fashion	tem-gra	M1	0	1	0.058	0	0	0
mobilenet	fashion	tem-gra	M3	0	1	0.02	0	0	0
mobilenet	fashion	tem-gra	M2	0	0.02	0.02	0	0	0
mobilenet	fashion	tem-gra	M4	0	0.571	0	0	0	0
mobilenet	fashion	cat-inc	M1	0.024	0.507	1	0.347	0	0
mobilenet	fashion	cat-inc	M3	0.039	1	1	0	0	0
mobilenet	fashion	cat-inc	M2	0	0.039	0	0	0	0
mobilenet	fashion	cat-inc	M4	0	0.23	0	0	0	0
mobilenet	fashion	tem-out	M1	0	0.701	0.4	0	0	0
mobilenet	fashion	tem-out	M3	0	0.742	0.71	0	0	0
mobilenet	fashion	tem-out	M2	0	0.278	0	0	0	0
mobilenet	fashion	tem-out	M4	0	0.04	0	0	0	0
mobilenet	fashion	cat-reo	M1	0	0.742	0.02	0	0	0
mobilenet	fashion	cat-reo	M3	0	0.758	0.02	0	0	0
mobilenet	fashion	cat-reo	M2	0	0.039	0	0	0	0
mobilenet	fashion	cat-reo	M4	0	0.425	0	0	0	0
mobilenet	fashion	tem-reo	M1	0	1	0.876	0	0	0
mobilenet	fashion	tem-reo	M3	0	0.964	0.02	0	0	0
mobilenet	fashion	tem-reo	M2	0	0.02	0	0	0	0
mobilenet	fashion	tem-reo	M4	0	1	0	0	0	0
mobilenet	CIFAR-10	cat-abr	C1	0	0	1	1	0.182	0
mobilenet	CIFAR-10	cat-abr	C2	0	0	0	0	0	0
mobilenet	CIFAR-10	cat-abr	C3	0	1	0	0.667	0.131	0
mobilenet	CIFAR-10	tem-abr	C1	0	0	1	0	0.131	0
mobilenet	CIFAR-10	tem-abr	C2	0	0	0.964	0	0	0
mobilenet	CIFAR-10	tem-abr	C3	0	1	0.851	0	0.113	0
mobilenet	CIFAR-10	cat-gra	C1	0	0.862	1	0	0	0
mobilenet	CIFAR-10	cat-gra	C2	0	0.821	0.907	0.02	0	0
mobilenet	CIFAR-10	cat-gra	C3	0	0.9	1	0.095	0.182	0.02
mobilenet	CIFAR-10	tem-gra	C1	0	1	1	0	0.039	0.02
mobilenet	CIFAR-10	tem-gra	C2	0	0.15	0.93	0	0	0.02
mobilenet	CIFAR-10	tem-gra	C3	0	0	0.969	0	0.096	0.02
mobilenet	CIFAR-10	cat-inc	C1	0.02	1	0.601	0	0.246	0.02
mobilenet	CIFAR-10	cat-inc	C2	0	1	0	0.02	0	0
mobilenet	CIFAR-10	cat-inc	C3	0.039	1	1	0.058	0.182	0
mobilenet	CIFAR-10	tem-out	C1	0	0	0.748	0.172	0.125	0
mobilenet	CIFAR-10	tem-out	C2	0	0.702	0.781	0	0.08	0
mobilenet	CIFAR-10	tem-out	C3	0	0.769	0.703	0	0.232	0
mobilenet	CIFAR-10	cat-reo	C1	0	1	0.507	0	0.165	0
mobilenet	CIFAR-10	cat-reo	C2	0	1	0.953	0	0	0
mobilenet	CIFAR-10	cat-reo	C3	0	1	0.895	0	0.261	0
mobilenet	CIFAR-10	tem-reo	C1	0	1	0.788	0	0.214	0
mobilenet	CIFAR-10	tem-reo	C2	0	1	1	0	0	0
mobilenet	CIFAR-10	tem-reo	C3	0	0.895	1	0.058	0.077	0

TABLE A.4: Experimental Results for VGG16 DNN, Hoeffding Decision Tree Methods

DNN	Dataset	Drift	Data Setup	DSCD hoeffadwin (Ours)	DSCD hoeffddm (Ours)	DSCD hoeffeddm (Ours)	DSCD hoeffhddma (Ours)	DSCD hoeffhddmw (Ours)	DSCD hoeffkswin (Ours)
vgg16	fashion	cat-abr	M1	0	0	0	0	0	0
vgg16	fashion	cat-abr	M3	0	0.02	0.02	0	0	0
vgg16	fashion	cat-abr	M2	0	0.995	0	1	0	0
vgg16	fashion	cat-abr	M4	0	0	0	0	0	0
vgg16	fashion	tem-abr	M1	0	0	0	0	0	0
vgg16	fashion	tem-abr	M3	0	1	0.02	0	0	0
vgg16	fashion	tem-abr	M2	0	0.734	0.02	0	0	0
vgg16	fashion	tem-abr	M4	0	1	0	0	0	0
vgg16	fashion	cat-gra	M1	0	0	0	0	0	0
vgg16	fashion	cat-gra	M3	0	0.216	0	0	0	0
vgg16	fashion	cat-gra	M2	0	0.02	1	0	0	0
vgg16	fashion	cat-gra	M4	0	0.02	0	0	0	0
vgg16	fashion	tem-gra	M1	0	0	0	0	0	0
vgg16	fashion	tem-gra	M3	0	0.113	0	0	0	0
vgg16	fashion	tem-gra	M2	0	1	0	0	0	0
vgg16	fashion	tem-gra	M4	0	0	0	0	0	0
vgg16	fashion	cat-inc	M1	0	0.02	0	0	0	0
vgg16	fashion	cat-inc	M3	0	0.742	0.02	0	0	0
vgg16	fashion	cat-inc	M2	0	1	0	0	0.058	0
vgg16	fashion	cat-inc	M4	0	0	0	0	0	0
vgg16	fashion	tem-out	M1	0	0	0	0	0	0
vgg16	fashion	tem-out	M3	0	0.538	0.594	0	0	0
vgg16	fashion	tem-out	M2	0	0.75	0	0	0	0
vgg16	fashion	tem-out	M4	0	0.039	0	0	0	0
vgg16	fashion	cat-reo	M1	0	0	0	0	0	0
vgg16	fashion	cat-reo	M3	0	1	0	0	0	0
vgg16	fashion	cat-reo	M2	0	0.02	0.02	0	0	0
vgg16	fashion	cat-reo	M4	0	0	0	0	0	0
vgg16	fashion	tem-reo	M1	0	0.02	0	0	0	0
vgg16	fashion	tem-reo	M3	0	0.561	0.02	0	0	0
vgg16	fashion	tem-reo	M2	0	1	0.02	0	0	0
vgg16	fashion	tem-reo	M4	0	0	0	0	0	0
vgg16	CIFAR-10	cat-abr	C1	0	1	1	0	0.131	0
vgg16	CIFAR-10	cat-abr	C2	0	1	1	0	0	0
vgg16	CIFAR-10	cat-abr	C3	0	0.529	0.837	0	0.23	0
vgg16	CIFAR-10	tem-abr	C1	0	0.113	1	0.974	0.425	0
vgg16	CIFAR-10	tem-abr	C2	0	1	1	0	0	0
vgg16	CIFAR-10	tem-abr	C3	0	0	1	0.131	0.214	0
vgg16	CIFAR-10	cat-gra	C1	0	0.821	1	1	0.132	0.02
vgg16	CIFAR-10	cat-gra	C2	0	1	1	0	0	0
vgg16	CIFAR-10	cat-gra	C3	0.02	0	0.571	0.601	0	0.02
vgg16	CIFAR-10	tem-gra	C1	0	1	1	0	0.246	0
vgg16	CIFAR-10	tem-gra	C2	0	1	1	0	0	0
vgg16	CIFAR-10	tem-gra	C3	0	1	1	0.059	0.2	0.02
vgg16	CIFAR-10	cat-inc	C1	0	1	0.701	0.83	0.198	0.02
vgg16	CIFAR-10	cat-inc	C2	0.02	1	0.851	0.953	0	0
vgg16	CIFAR-10	cat-inc	C3	0.02	0.925	1	0.592	0.261	0
vgg16	CIFAR-10	tem-out	C1	0	0.763	0.648	0	0.323	0
vgg16	CIFAR-10	tem-out	C2	0	0.792	0.717	0	0	0
vgg16	CIFAR-10	tem-out	C3	0	0.733	0.538	0	0.246	0
vgg16	CIFAR-10	cat-reo	C1	0	1	0.844	0.214	0.4	0
vgg16	CIFAR-10	cat-reo	C2	0	1	1	0	0	0
vgg16	CIFAR-10	cat-reo	C3	0	0	1	0	0.261	0
vgg16	CIFAR-10	tem-reo	C1	0	1	0	0	0.276	0
vgg16	CIFAR-10	tem-reo	C2	0	1	1	0	0	0
vgg16	CIFAR-10	tem-reo	C3	0	1	0.639	0	0.077	0

Appendix B

Additional Results for Chapter 4 - DeepStreamOS

B.1 DNN Accuracies

Tables B.1 and B.2 show the training classification accuracy of the DNNs prior to adaptation, where true positives are the correctly predicted class and super-class respectively.

TABLE B.1: Initial training accuracies for novel class DNNs

Trained Classes	DNN Accuracy
MobileNet CIFAR-10	
0-1	0.947
0-8	0.858
0-9	0.911
3-4	0.876
3-6	0.865
8-9	0.909
0-1-8-9	0.791
2-3-4-5-6-7	0.643
1-4-6-8	0.874
0-2-3-7-8-9	0.758
MobileNet Fashion-MNIST	
0-2	0.956
3-4	0.936
4-6	0.870
5-7	0.969
7-9	0.968
0-1-2-3-4-6	0.817
5-7-8-9	0.958
1-3-5-8	0.981
0-2-4-6-7-9	0.857
VGG16 CIFAR-10	
0-1	0.966
0-8	0.935
0-9	0.963
3-4	0.913
3-6	0.904
8-9	0.962
0-1-8-9	0.908
2-3-4-5-6-7	0.777
1-4-6-8	0.952
0-2-3-7-8-9	0.878
VGG16 Fashion-MNIST	
0-2	0.973
3-4	0.961
4-6	0.909
5-7	0.983
7-9	0.975
0-1-2-3-4-6	0.865
5-7-8-9	0.977
1-3-5-8	0.989
0-2-4-6-7-9	0.886

TABLE B.2: Initial training accuracies for novel sub-class DNNs

Trained Classes	DNN Accuracy
MobileNet - CIFAR-10	
0-2	0.936
0-1-2-3	0.956
0-1-2-3-4-8	0.962
MobileNet - Fashion-MNIST	
0-5	1.000
0-1-5-7	1.000
0-1-2-5-7-9	1.000
VGG16 - CIFAR-10	
0-2	0.956
0-1-2-3	0.966
0-1-2-3-4-8	0.971
VGG16 - Fashion-MNIST	
0-5	1.000
0-1-5-7	1.000
0-1-2-5-7-9	1.000

B.2 DNN Prediction Accuracy Investigation Results

This section investigates how DNN classifier accuracy affects *DeepStreamOS*. The experiment comprises of changing the accuracy of the DNN to 33%, 50%, 70%, 90% and 100%. Ideally, the DNN should be trained to these specific accuracies and the neuron activations extracted from these. However, this is not possible as the DNN starts training from initialisation at accuracies of approximately 0.8 accuracy for the smaller number of classes. Therefore, the accuracies have been achieved by altering the prediction of the DNN after it has been predicted and replacing predicted values with suitable random values to achieve the desired accuracy. This means that the activations extracted (which are used to create the MCOD clusterers) are representative of a more accurately trained CNN than the altered predictions provide. Therefore accuracy results are probably higher than they would be if the CNN was able to be trained to such a low accuracy. It should be noted that during training, the MCOD clusterers are initialised with correctly predicted training instances only. Therefore, as the applied DNN accuracy reduces, so do the amount of training instances stored in each MCOD cluster. Nonetheless, this experiment can give an indication to the behaviour of the system at lower DNN accuracies.

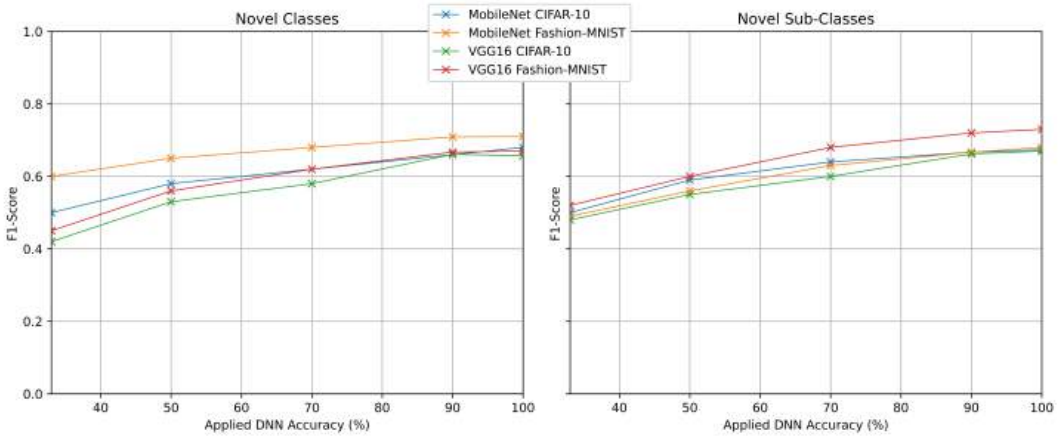


FIGURE B.1: Variation of F1-Score with DNN prediction accuracy

Figure B.1 shows the variation of the F1-Score (where true positives are unknown classes classified as unknown) with the applied accuracies for novel classes and novel sub-classes. Overall, the lower the trained accuracy of the CNN, the lower

the detection rate of unknowns in the open-set system. The unknown detection accuracy reduces at a lower rate than the applied DNN accuracy. This is because less training data was used to initialise the MCOD clusterers with the reduction rate possibly being lessened by the activations originating from a DNN with a higher accuracy.

B.3 Results of Parameter Investigation

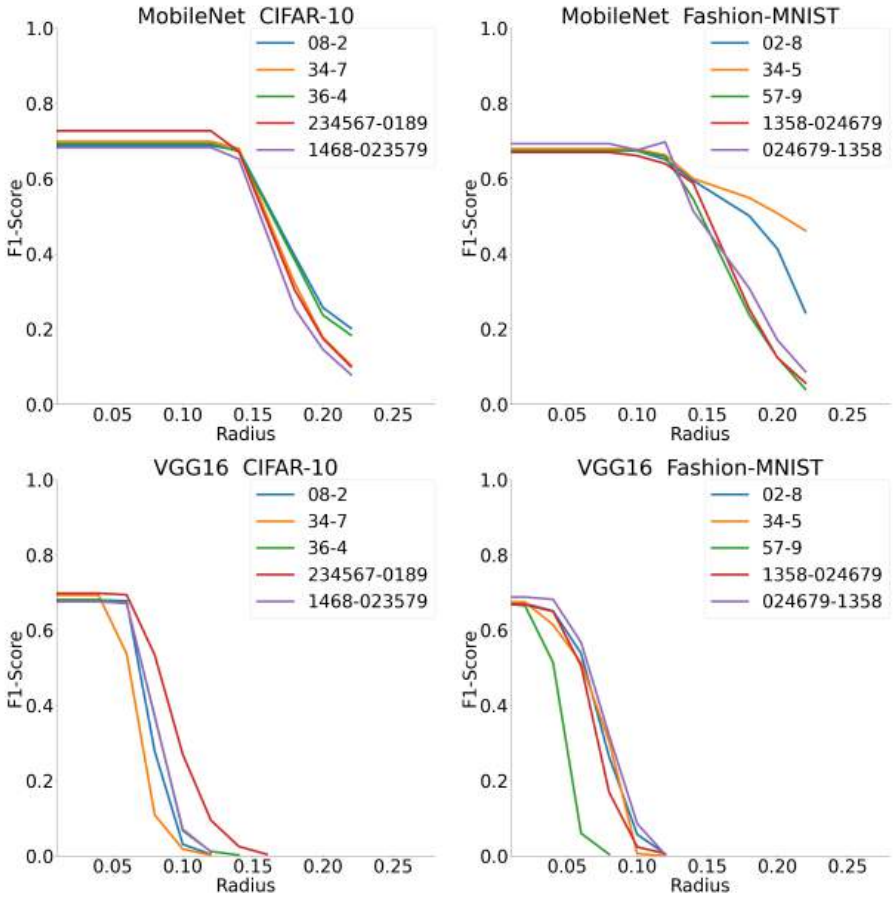


FIGURE B.2: Radius, *R* against F1-Score for class data combinations.

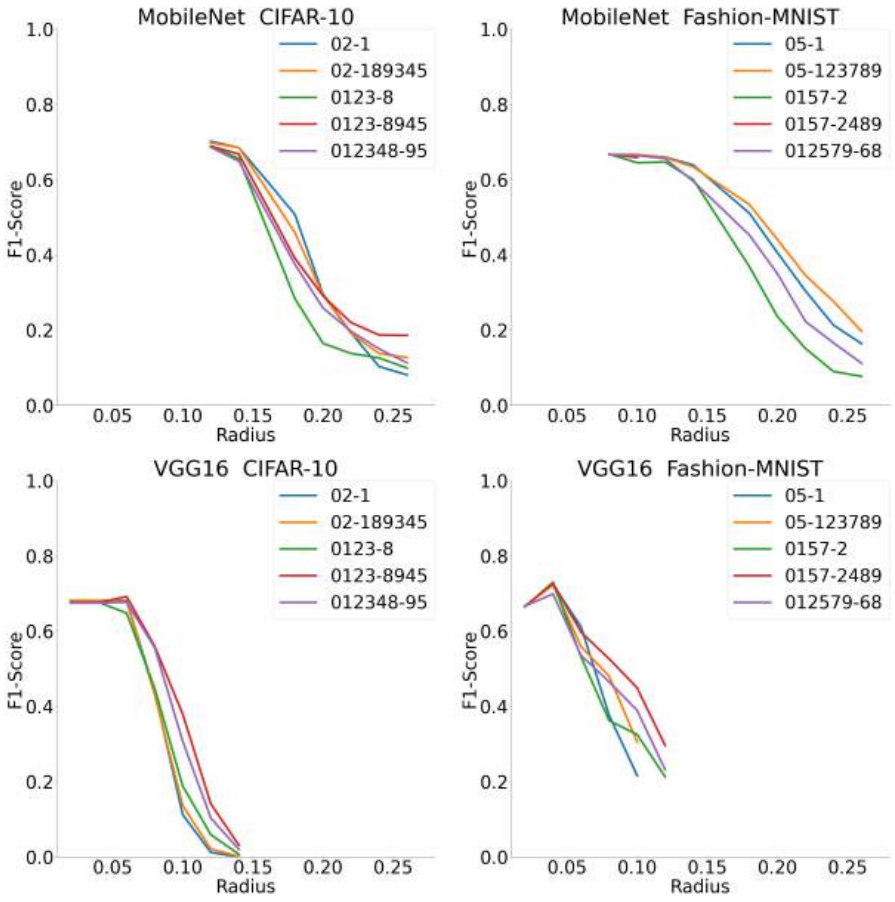


FIGURE B.3: Radius, *R* against F1-Score for sub-class data combinations.

Appendix C

Additional Results for Chapter 5 - AdaDeepStream

C.1 DNN and Streaming Classifier Accuracies

Table C.1 shows the training classification accuracy of the DNNs and the streaming classifiers prior to adaptation, where true positives are the correctly predicted class.

TABLE C.1: Training accuracies for VGG16 DNN and Hoeffding Tree Streaming Classifier (SC)

Trained Classes	DNN	SC (JSDL)	SC (CBIR)
CIFAR-10			
0-1-2-3-4-6-7-8	0.879	0.808	0.809
2-3-4-5-6-7-8-9	0.983	0.749	0.850
0-1-2-4-5-6-8-9	0.899	0.809	0.810
0-1-2-3-6-7	0.905	0.824	0.823
2-3-4-5-6-7	0.976	0.825	0.862
0-1-4-5-8-9	0.994	0.890	0.892
5-7-8-9	0.947	0.876	0.874
0-2-6-7	0.926	0.823	0.881
1-2-3-4	0.908	0.849	0.840
0-7	0.979	0.970	0.947
1-8	0.980	0.950	0.947
2-3	0.912	0.895	0.908
Fashion-MNIST			
0-1-2-3-5-6-8-9	0.940	0.923	0.902
2-3-4-5-6-7-8-9	0.995	0.944	0.962
0-1-2-3-4-6-7-8	0.918	0.916	0.873
0-1-2-3-4-6	0.958	0.891	0.907
4-5-6-7-8-9	0.985	0.967	0.965
2-3-4-6-8-9	0.926	0.914	0.911
0-1-8-9	0.995	0.959	0.970
0-1-5-6	0.948	0.945	0.930
0-1-2-7	0.987	0.962	0.965
6-9	1.000	0.998	0.992
1-8	0.999	0.996	0.996
0-5	1.000	0.996	0.996
CIFAR-100			
1-3-4-5-6-7-10-11-13-17	0.816	0.683	0.682
0-1-3-5-11-12-15-17-18-19	0.733	0.635	0.634
1-5-7-8-9-14-15-16-17-18	0.805	0.643	0.644
4-7-8-9-11-14-15-17-18-19	0.798	0.692	0.690
0-2-4-5-7-9-13-14-15-18	0.795	0.708	0.690
5-6-7-10-11-12-14-17-18-19	0.815	0.660	0.664
1-2-4-6-7-9-11-16-18-19	0.792	0.689	0.696
0-1-6-7-8-9-11-12-17-18	0.774	0.668	0.672
0-2-3-5-8-9-10-11-12-19	0.777	0.689	0.687
0-9-10-11-12-13-14-16-17-18	0.769	0.654	0.651
0-2-3-4-6-8-9-13-16-19	0.806	0.627	0.634
2-5-10-11-13-15-16-17-18-19	0.790	0.634	0.634
4-5-8-12-14-15-16-17-18-19	0.751	0.639	0.640
1-2-3-4-5-6-7-8-12-13	0.748	0.576	0.612
0-1-7-8-9-10-11-14-18-19	0.778	0.660	0.657

C.2 Drift detection on pairs of novel classes

Tables C.2 and C.3 show the F1-Score for the drift detection of *AdaDeepStream* as applied to eight trained classes and varying combinations of two unknown classes. This was performed for the Temporal-Abrupt drift pattern. The tables are ordered via the F1-Score from highest to lowest and the mix of categories are listed in the 'Class Category' column. From this it can be seen that the mix of categories dominate the higher F1-Scores, with the single categories predominately in the lower portions of the tables.

TABLE C.2: Drift detection on pairs of novel classes for VGG16 CIFAR-10

Trained Classes	Unknown Class Numbers	F1-Score	Unknown Class Name 1	Unknown Class Name 2	Class Category
0-1-2-3-4-6-7-8	5-9	0.977	dog	truck	Mix
0-2-3-4-5-7-8-9	1-6	0.971	automobile	frog	Mix
0-2-3-4-6-7-8-9	1-5	0.969	automobile	dog	Mix
0-2-3-5-6-7-8-9	1-4	0.966	automobile	deer	Mix
0-1-2-3-4-5-6-8	7-9	0.965	horse	truck	Mix
0-1-2-3-4-6-7-9	5-8	0.963	dog	ship	Mix
0-2-3-4-5-6-8-9	1-7	0.960	automobile	horse	Mix
0-1-2-4-5-6-7-9	3-8	0.954	cat	ship	Mix
0-1-2-3-4-5-6-9	7-8	0.951	horse	ship	Mix
0-1-2-3-4-5-7-8	6-9	0.950	frog	truck	Mix
0-1-2-3-4-5-7-9	6-8	0.941	frog	ship	Mix
0-1-2-4-5-6-7-8	3-9	0.933	cat	truck	Mix
0-2-4-5-6-7-8-9	1-3	0.932	automobile	cat	Mix
0-1-2-3-5-6-7-9	4-8	0.925	deer	ship	Mix
0-1-3-4-5-6-7-8	2-9	0.923	bird	truck	Mix
1-2-3-4-5-6-8-9	0-7	0.914	airplane	horse	Mix
1-2-3-4-6-7-8-9	0-5	0.891	airplane	dog	Mix
1-2-4-5-6-7-8-9	0-3	0.879	airplane	cat	Mix
1-3-4-5-6-7-8-9	0-2	0.869	airplane	bird	Mix
1-2-3-5-6-7-8-9	0-4	0.869	airplane	deer	Mix
0-1-3-4-5-6-7-9	2-8	0.866	bird	ship	Mix
2-3-4-5-6-7-8-9	0-1	0.851	airplane	automobile	Transport
1-2-3-4-5-6-7-8	0-9	0.826	airplane	truck	Transport
1-2-3-4-5-7-8-9	0-6	0.813	airplane	frog	Mix
0-1-2-4-5-6-8-9	3-7	0.813	cat	horse	Animals
0-1-3-4-5-7-8-9	2-6	0.792	bird	frog	Animals
0-1-2-3-4-6-8-9	5-7	0.743	dog	horse	Animals
0-1-3-4-5-6-8-9	2-7	0.714	bird	horse	Animals
1-2-3-4-5-6-7-9	0-8	0.704	airplane	ship	Transport
0-1-3-5-6-7-8-9	2-4	0.590	bird	deer	Animals
0-1-2-3-4-5-8-9	6-7	0.539	frog	horse	Animals
0-1-2-3-6-7-8-9	4-5	0.521	deer	dog	Animals
0-1-3-4-6-7-8-9	2-5	0.517	bird	dog	Animals
0-1-2-4-5-7-8-9	3-6	0.513	cat	frog	Animals
0-1-4-5-6-7-8-9	2-3	0.505	bird	cat	Animals
0-1-2-4-6-7-8-9	3-5	0.502	cat	dog	Animals
0-1-2-3-5-7-8-9	4-6	0.499	deer	frog	Animals
0-1-2-3-5-6-8-9	4-7	0.495	deer	horse	Animals
0-2-3-4-5-6-7-8	1-9	0.474	automobile	truck	Transport
2-3-4-5-6-7-8-9	1-2	0.438	automobile	bird	Mix
0-1-2-5-6-7-8-9	3-4	0.296	cat	deer	Animals

TABLE C.3: Drift detection on pairs of novel classes for VGG16 Fashion-MNIST

Trained Classes	Unknown Class Numbers	F1-Score	Unknown Class Name 1	Unknown Class Name 2	Class Category
0-1-3-4-6-7-8-9	2-5	1.000	Pullover	Sandal	Mix
0-1-2-3-5-6-8-9	4-7	1.000	Coat	Sneaker	Mix
0-1-2-3-5-6-7-8	4-9	1.000	Coat	Ankle boot	Mix
1-2-3-4-5-6-7-8	0-9	0.999	T-shirt	Ankle boot	Mix
0-2-3-4-5-6-7-8	1-9	0.999	Trouser	Ankle boot	Mix
0-1-2-4-5-6-7-8	3-9	0.999	Dress	Ankle boot	Mix
0-1-2-3-4-5-8-9	6-7	0.999	Shirt	Sneaker	Mix
0-1-2-4-5-6-8-9	3-7	0.997	Dress	Sneaker	Mix
0-1-2-3-6-7-8-9	4-5	0.996	Coat	Sandal	Mix
0-1-2-3-4-7-8-9	5-6	0.991	Sandal	Shirt	Mix
0-1-2-3-4-5-7-8	6-9	0.988	Shirt	Ankle boot	Mix
1-2-3-4-6-7-8-9	0-5	0.986	T-shirt	Sandal	Mix
0-1-2-3-4-5-6-7	8-9	0.985	Bag	Ankle boot	Footwear
0-1-2-4-6-7-8-9	3-5	0.978	Dress	Sandal	Mix
0-1-3-4-5-6-7-8	2-9	0.977	Pullover	Ankle boot	Mix
0-1-2-3-4-6-7-9	5-8	0.970	Sandal	Bag	Footwear
2-3-4-5-6-7-8-9	0-1	0.969	T-shirt	Trouser	Clothing
0-2-3-4-5-6-7-9	1-8	0.965	Trouser	Bag	Mix
0-2-3-4-5-7-8-9	1-6	0.962	Trouser	Shirt	Clothing
1-2-3-4-5-6-8-9	0-7	0.941	T-shirt	Sneaker	Mix
0-2-3-5-6-7-8-9	1-4	0.938	Trouser	Coat	Mix
0-1-3-4-5-6-8-9	2-7	0.929	Pullover	Sneaker	Mix
0-1-2-3-4-5-6-9	7-8	0.919	Sneaker	Bag	Footwear
0-2-3-4-6-7-8-9	1-5	0.907	Trouser	Sandal	Mix
0-1-3-4-5-6-7-9	2-8	0.883	Pullover	Bag	Mix
0-1-2-3-5-6-7-9	4-8	0.850	Coat	Bag	Mix
0-1-2-5-6-7-8-9	3-4	0.808	Dress	Coat	Clothing
1-2-3-4-5-7-8-9	0-6	0.768	T-shirt	Shirt	Clothing
0-1-2-4-5-6-7-9	3-8	0.680	Dress	Bag	Mix
0-1-2-3-4-6-7-8	5-9	0.637	Sandal	Ankle boot	Footwear
1-2-3-4-5-6-7-9	0-8	0.601	T-shirt	Bag	Mix
0-2-4-5-6-7-8-9	1-3	0.570	Trouser	Dress	Clothing
0-1-2-3-4-5-7-9	6-8	0.523	Shirt	Bag	Mix
0-1-2-3-4-5-6-8	7-9	0.498	Sneaker	Ankle boot	Footwear
0-1-3-5-6-7-8-9	2-4	0.492	Pullover	Coat	Clothing
1-2-3-5-6-7-8-9	0-4	0.485	T-shirt	Coat	Clothing
1-3-4-5-6-7-8-9	0-2	0.484	T-shirt	Pullover	Clothing
0-1-3-4-5-7-8-9	2-6	0.476	Pullover	Shirt	Clothing
0-1-2-3-5-7-8-9	4-6	0.460	Coat	Shirt	Clothing
0-1-2-4-5-7-8-9	3-6	0.456	Dress	Shirt	Clothing
1-2-4-5-6-7-8-9	0-3	0.447	T-shirt	Dress	Clothing
2-3-4-5-6-7-8-9	1-2	0.447	Trouser	Pullover	Clothing
0-1-4-5-6-7-8-9	2-3	0.270	Pullover	Dress	Clothing
0-1-2-3-4-6-8-9	5-7	0.224	Sandal	Sneaker	Footwear

C.3 Novel Class Accuracy Results

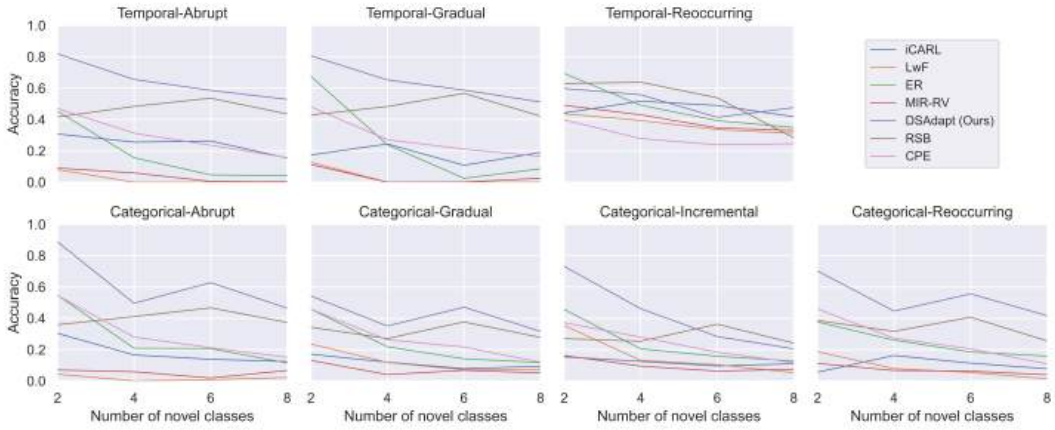


FIGURE C.1: Number of novel classes against accuracy for DS-CBIR, VGG16 CNN, CIFAR-10 for all concept evolution patterns.

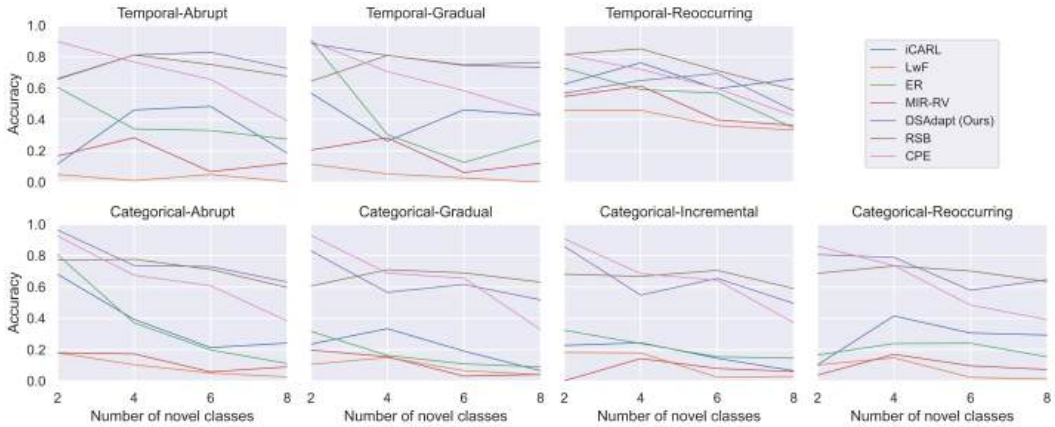


FIGURE C.2: Number of novel classes against accuracy for DS-CBIR, VGG16 CNN, Fashion-MNIST for all concept evolution patterns.

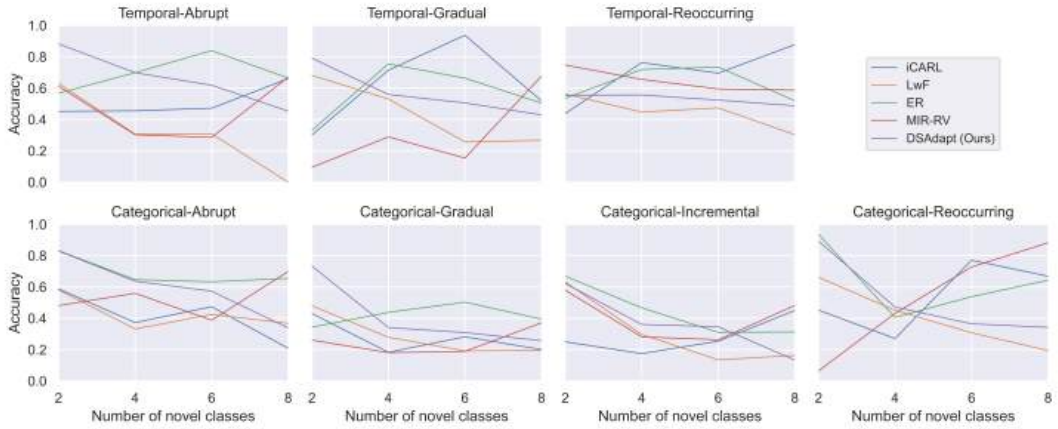


FIGURE C.3: Number of novel classes against accuracy for JSDL, VGG16 CNN, CIFAR-10 for all concept evolution patterns.

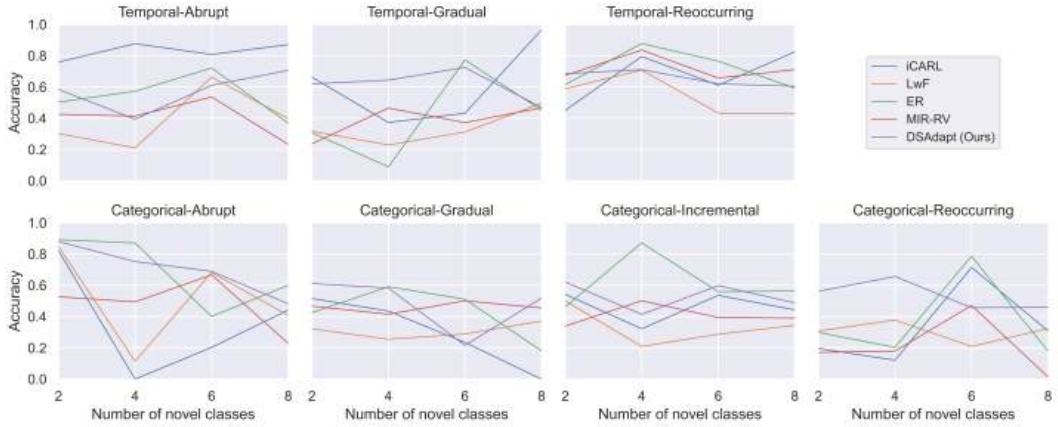


FIGURE C.4: Number of novel classes against accuracy for JSDL, VGG16 CNN, Fashion-MNIST for all concept evolution patterns.

TABLE C.4: Average accuracy after CNN adaptation for each concept evolution pattern for JSDL activation reduction. Highest values are in bold.

Reduction/ Dataset	Method	Cat. Abr.	Tem. Abr.	Cat. Gra.	Tem. Gra.	Cat. inc.	Cat. Reo.	Tem. Reo.
JSDL CIFAR-10 8 Trained 2 Novel classes	iCARL	0.588	0.452	0.431	0.301	0.250	0.452	0.438
	LwF	0.582	0.631	0.480	0.682	0.632	0.662	0.563
	ER	**0.832**	0.568	0.343	0.331	**0.669**	**0.936**	0.535
	MIR-RV	0.481	0.613	0.261	0.095	0.580	0.064	**0.748**
	ADS (ours)	0.830	**0.883**	**0.734**	**0.792**	0.622	0.893	0.555
JSDL CIFAR-100 10 Trained 2 Novel classes	iCARL	0.578	0.491	0.460	0.324	0.303	0.612	0.508
	LwF	0.708	0.732	0.642	0.180	0.522	0.512	0.591
	ER	**0.856**	0.821	**0.744**	0.787	0.429	0.376	0.572
	MIR-RV	0.453	0.586	0.642	0.688	**0.765**	0.372	0.645
	ADS (ours)	0.793	**0.864**	0.691	**0.842**	0.746	**0.644**	**0.760**
JSDL MNIST-Fashion 8 Trained 2 Novel classes	iCARL	0.821	**0.759**	0.516	**0.663**	0.543	0.196	0.450
	LwF	0.848	0.302	0.322	0.317	0.501	0.308	0.589
	ER	**0.892**	0.503	0.425	0.308	0.462	0.299	0.615
	MIR-RV	0.527	0.425	0.467	0.236	0.339	0.171	0.674
	ADS (ours)	0.880	0.584	**0.611**	0.620	**0.621**	**0.562**	**0.685**

Appendix D

Additional Results for Chapter 6 - DeepStreamEnsemble

D.1 DNN and Streaming Classifier Accuracy

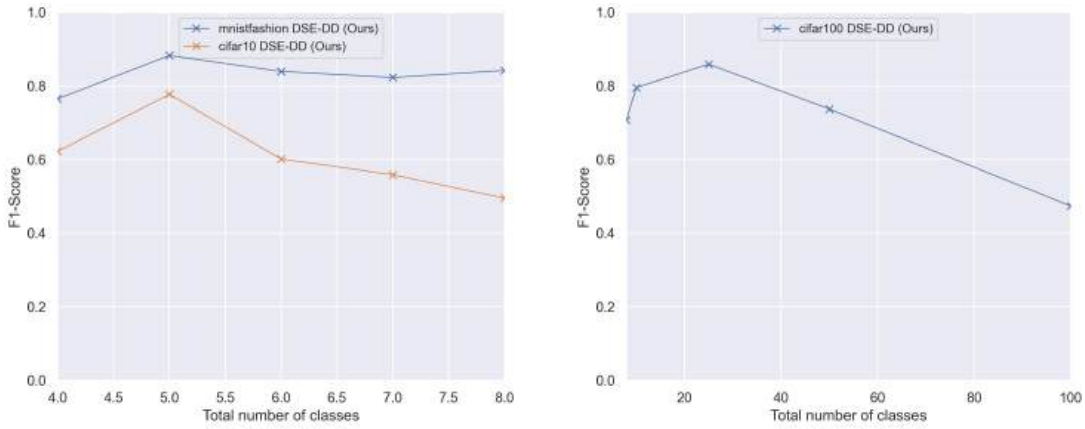
Table D.1 shows the training classification accuracy of the DNNs prior to adaptation, where true positives are the correctly predicted super-class.

TABLE D.1: Initial training accuracies for VGG16 DNN and Hoeffding Tree Streaming Classifiers (SC0 to SC5)

Trained Classes	DNN	SC0	SC1	SC2	SC3	SC4	SC5
CIFAR-10							
0-2	0.957	0.636	0.837	0.885	0.946	0.979	0.98
0-1-2-3	0.966	0.618	0.848	0.886	0.952	0.987	0.981
0-1-2-3-4-8	0.967	0.618	0.903	0.93	0.979	0.987	0.996
Fashion-MNIST							
0-5	1.000	0.822	0.996	0.997	1.000	0.998	0.999
0-1-5-7	1.000	0.869	0.998	1.000	1.000	0.999	0.998
0-1-2-5-7-9	1.000	0.883	0.998	0.998	1.000	1.000	1.000
CIFAR-100							
7-10-11-13-14-16-17-19-20-21-25-29-30-32-33-35-37-39-40-42-43-44-45-48-49-50-51-52-53-55-56-62-63-64-65-67-69-70-77-81	0.712	0.089	0.459	0.59	0.775	0.766	0.642
0-1-2-3-4-5-6-8-9-12-15-22-23-24-26-27-34-36-38-41-47-54-61-71-75-76-80-83-87-89-90-91-92-93-94-95-96-97-98-99	0.714	0.07	0.473	0.602	0.801	0.75	0.655
18-24-28-31-38-46-57-58-59-60-61-66-68-71-72-73-74-75-76-78-79-80-82-83-84-85-86-87-88-89-90-91-92-93-94-95-96-97-98-99	0.76	0.088	0.471	0.615	0.827	0.838	0.683
18-24-31-38-59-60-68-71-73-76-78-82-85-88-89-91-92-93-96-97	0.856	0.178	0.579	0.694	0.877	0.893	0.74
14-16-18-21-25-28-31-37-40-49-60-65-67-68-73-74-77-79-84-86	0.784	0.162	0.524	0.639	0.863	0.875	0.748
13-17-19-21-29-37-42-43-44-45-48-50-62-63-64-65-69-70-77-81	0.691	0.184	0.557	0.668	0.792	0.706	0.713
2-8-9-22-26-46-48-61-79-87	0.862	0.226	0.599	0.751	0.876	0.903	0.832
14-18-55-56-59-72-77-79-81-85	0.831	0.252	0.626	0.737	0.886	0.849	0.813
18-24-66-72-75-82-84-92-94-95	0.889	0.304	0.711	0.797	0.898	0.933	0.86
31-38-57-83	0.976	0.598	0.884	0.886	0.967	0.98	0.968
59-79-96-99	0.97	0.555	0.872	0.902	0.934	0.975	0.925
78-86-87-93	0.971	0.541	0.863	0.885	0.972	0.988	0.967

D.2 CIFAR-10 and CIFAR-100 Drift Detection Analysis

This section analyses the difference between the drift detection results for CIFAR-10 and CIFAR-100 from Table 6.4. CIFAR-10 and CIFAR-100 offer two different experimental cases. CIFAR-10 contains 10 classes, which can be naturally split into two super-classes of Animal and Transport. CIFAR-100 contains 20 super-classes, which are divided into 100 sub-classes. The class combinations are specified in Table 6.3. CIFAR-10 and Fashion-MNIST investigate a larger amount of data per sub-class (1000 test instances) with a small number of sub-classes applied. Conversely, CIFAR-100 investigates a smaller amount of data per sub-class (500 test instances) with a larger number of sub-classes applied. Figure D.1 shows how the F1-Score varies with the total number of sub-classes for each dataset. The F1-Score definition is such that true positives are the correct identification of unknown instances. The CIFAR-100 data combinations begin at eight sub-classes, and the F1-Score for this lies between the F1-Score for the CIFAR-10 dataset and the Fashion-MNIST datasets as can be seen in Figure D.1 (b). To allow for direct comparison between the datasets, we compare drift detection plots for a CIFAR-10 and CIFAR-100 data combination that have the same number of known and unknown sub-classes.



(a) CIFAR-10 and Fashion-MNIST (b) CIFAR-100

FIGURE D.1: Total number of classes against F1-Score for Fashion-MNIST, CIFAR-10 and CIFAR-100

Figure D.2 shows an example of CIFAR-10 drift detection for the known classes

of 0, 1, 2 and 3, with unknown classes 8 and 9 applied. Figure D.3 shows an example of CIFAR-100 drift detection for the known classes of 78, 86, 87 and 93, with unknown classes 22 and 44 applied. From these examples, it can be seen that the CIFAR-10 data has more instances than the CIFAR-100 data and does not detect established drift, but only changes. With more data available, the system has time to settle down such that there are no longer volatility changes, whereas, with a smaller amount of data, more of the data is perceived as changing. Thus, the CIFAR-10 F1-Score results are lower than the CIFAR-100 results.

Figure D.4 shows UMAP [125] representations of the known and unknown sub-classes aforementioned CIFAR-10 and CIFAR-100 examples for each block of the CNN. It can be seen that the inter-class separation and the intra-class cohesion are similar in these examples, but with less data points.

In summary, the *DeepStreamEnsemble* drift detection method detects changes in data, not the entire range of novel sub-classes. This is not an indicator of how well the adaptation of the system will perform as not all of the unknown data is required for successful adaptation. Less change detection windows means less true-labelling resources are required. As shown in Table 6.5, CIFAR-10 performs well with respect to the classification accuracy after adaptation.

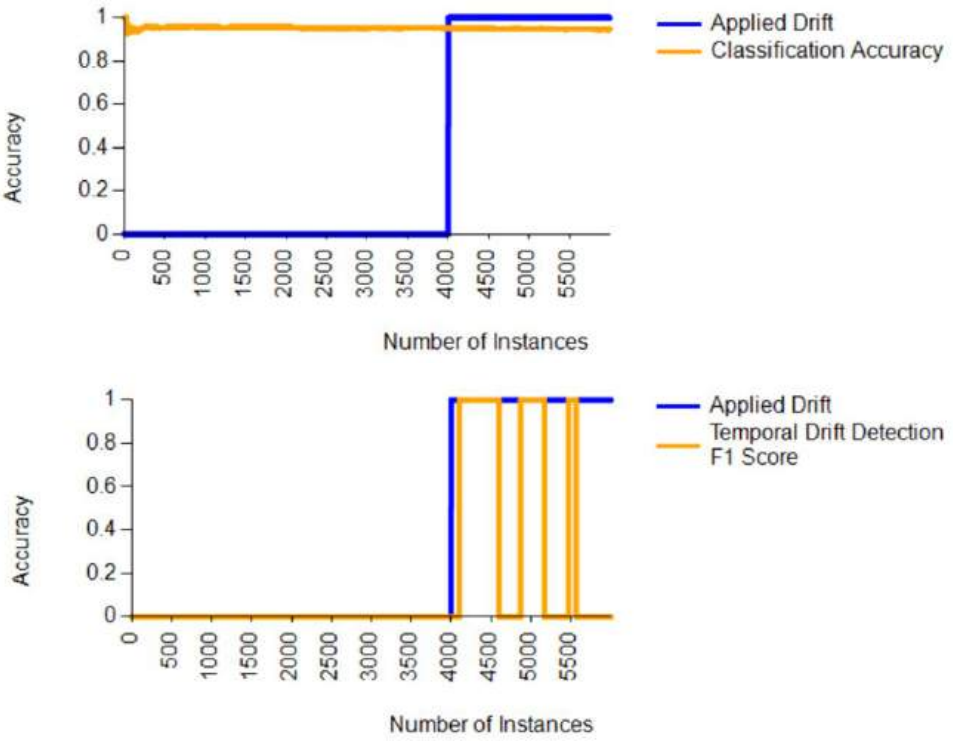


FIGURE D.2: Drift detection example for CIFAR-10

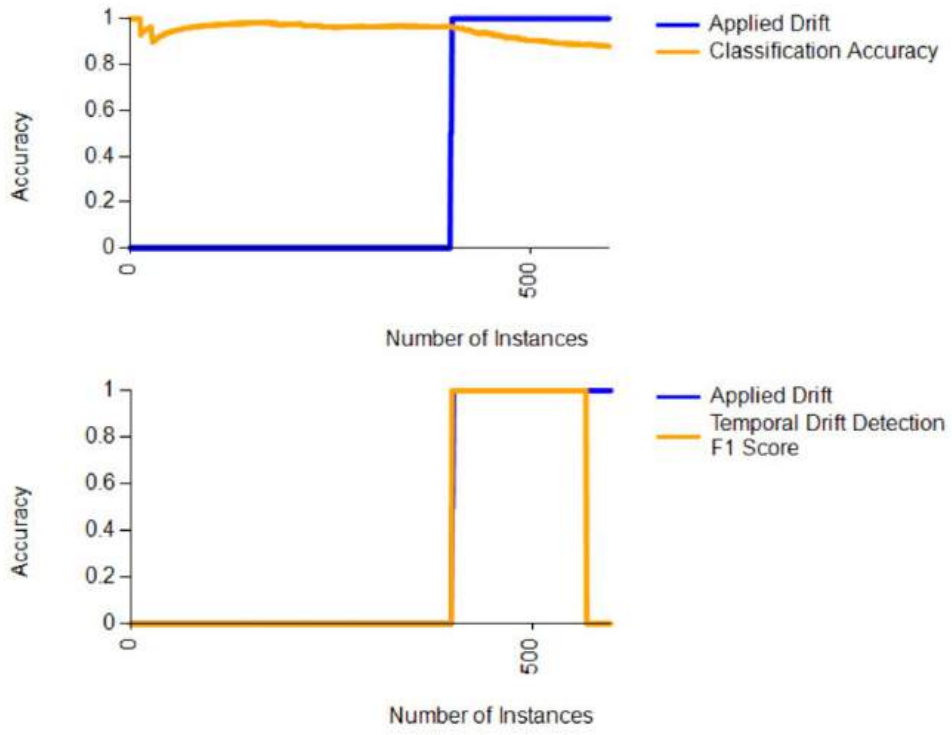
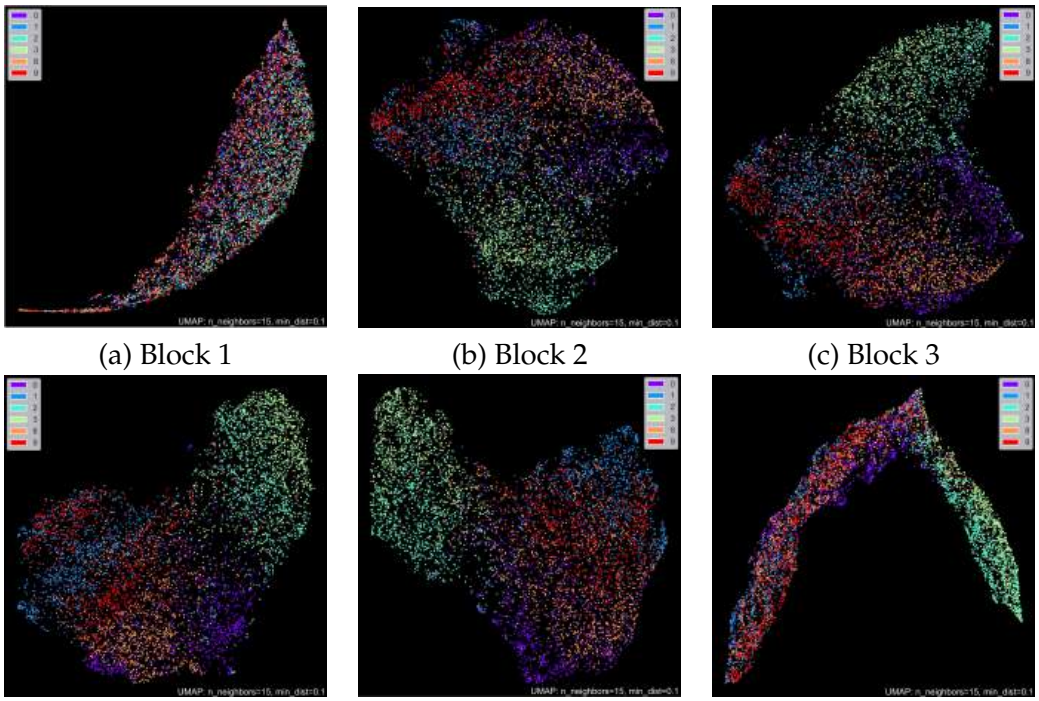


FIGURE D.3: Drift detection example for CIFAR-100

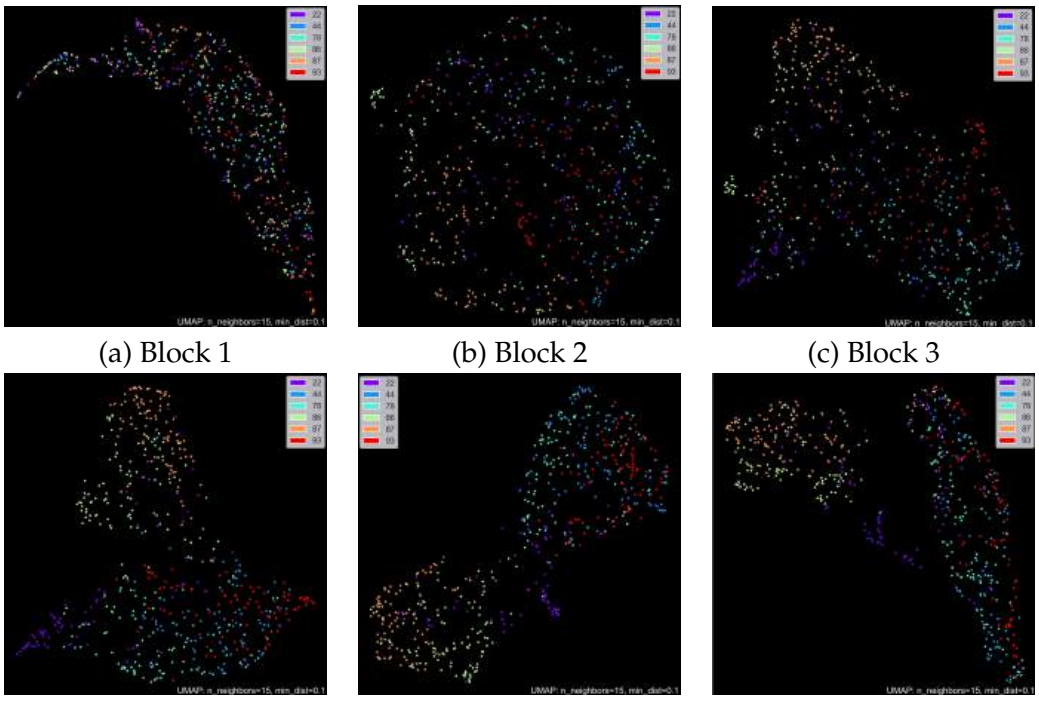
CIFAR-10



(a) Block 1 (b) Block 2 (c) Block 3

(d) Block 4 (e) Block 5 (f) Final

CIFAR-100



(a) Block 1 (b) Block 2 (c) Block 3

(d) Block 4 (e) Block 5 (f) Final

FIGURE D.4: UMAP representations of reduced activation training data for each block and final hidden layer of VGG16 CNN

Bibliography

[1] Davide Abati et al. “Conditional Channel Gated Networks for Task-Aware Continual Learning”. In: 2020, pp. 3931–3940.

[2] Aisha Abdallah, Mohd Aizaini Maarof, and Anazida Zainal. “Fraud detection system: A survey”. In: *Journal of Network and Computer Applications* 68 (June 2016), pp. 90–113. ISSN: 1084-8045.

[3] Zahraa S. Abdallah et al. “Activity Recognition with Evolving Data Streams: A Review”. In: *ACM Computing Surveys* 51.4 (July 2018), 71:1–71:36. ISSN: 0360-0300.

[4] Amina Adadi and Mohammed Berrada. “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)”. en. In: *IEEE Access* 6 (2018), pp. 52138–52160. ISSN: 2169-3536.

[5] M. Adimoolam et al. “A Novel Technique to Detect and Track Multiple Objects in Dynamic Video Surveillance Systems”. eng. In: *International Journal of Interactive Multimedia and Artificial Intelligence* (June 2022). ISSN: 1989-1660.

[6] Supriya Agrahari and Anil Kumar Singh. “Concept Drift Detection in Data Stream Mining : A literature review”. en. In: *Journal of King Saud University - Computer and Information Sciences* (Dec. 2021). ISSN: 1319-1578.

[7] Rahaf Aljundi et al. “Gradient based sample selection for online continual learning”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[8] Rahaf Aljundi et al. “Online Continual Learning with Maximal Interfered Retrieval”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[9] Guilherme Aresta et al. "BACH: Grand challenge on breast cancer histology images". In: *Medical Image Analysis* 56 (Aug. 2019), pp. 122–139. ISSN: 1361-8415.

[10] David Arthur and Sergei Vassilvitskii. "k-means++: the advantages of careful seeding". In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*. Ed. by Nikhil Bansal, Kirk Pruhs, and Clifford Stein. SIAM, 2007, pp. 1027–1035. URL: http://dl.acm.org/citation.cfm?id=1283383.1283494.

[11] Manuel Baena-García et al. "Early Drift Detection Method". In: *4th ECML PKDD international workshop on knowledge discovery* (Jan. 2006).

[12] RE Bellman and SE Dreyfus. *Applied dynamic programming*. 2015.

[13] Abhijit Bendale and Terrance E. Boult. "Towards Open Set Deep Networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 1563–1572.

[14] Ayush Bhardwaj et al. "Empowering Knowledge Distillation via Open Set Recognition for Robust 3D Point Cloud Classification". en. In: *Pattern Recognition Letters* 151 (Nov. 2021), pp. 172–179. ISSN: 0167-8655.

[15] Kshitij Bhardwaj et al. "Benchmarking Test-Time Unsupervised Deep Neural Network Adaptation on Edge Devices". In: *2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. May 2022, pp. 236–238.

[16] Albert Bifet and Ricard Gavaldà. "Adaptive Learning from Evolving Data Streams". en. In: *Advances in Intelligent Data Analysis VIII*. Ed. by Niall M. Adams et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 249–260. ISBN: 978-3-642-03915-7.

[17] Albert Bifet and Ricard Gavaldà. "Learning from Time-Changing Data with Adaptive Windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining*. Proceedings. Society for Industrial and Applied Mathematics, Apr. 2007, pp. 443–448. ISBN: 978-0-89871-630-6.

[18] Zalán Borsos, Mojmir Mutny, and Andreas Krause. "Coresets via Bilevel Optimization for Continual Learning and Streaming". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 14879–14890.

[19] Leo Breiman. *Classification and Regression Trees*. New York: Routledge, Oct. 2017. ISBN: 978-1-315-13947-0.

[20] Lorenzo Brigato et al. "Image Classification With Small Datasets: Overview and Benchmark". In: *IEEE Access* 10 (2022). Conference Name: IEEE Access, pp. 49233–49250. ISSN: 2169-3536.

[21] Vanessa Buhrmester, David Münch, and Michael Arens. "Analysis of Explainers of Black Box Deep Neural Networks for Computer Vision: A Survey". en. In: *Machine Learning and Knowledge Extraction* 3.4 (Dec. 2021), pp. 966–989. ISSN: 2504-4990.

[22] Xin-Qiang Cai et al. "Nearest Neighbor Ensembles: An Effective Method for Difficult Problems in Streaming Classification with Emerging New Classes". In: *2019 IEEE International Conference on Data Mining (ICDM)*. ISSN: 2374-8486. Nov. 2019, pp. 970–975.

[23] Fabio Carrara et al. "Adversarial examples detection in features distance spaces". In: 2018, pp. 1–10.

[24] Fabio Carrara et al. "Adversarial image detection in deep neural networks". en. In: *Multimedia Tools and Applications* 78.3 (Feb. 2019), pp. 2815–2835. ISSN: 1573-7721.

[25] Francisco M. Castro et al. "End-to-End Incremental Learning". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 233–248.

[26] Lorraine Chambers and Mohamed Medhat Gaber. "DeepStreamOS: Fast open-Set classification for convolutional neural networks". en. In: *Pattern Recognition Letters* 154 (Feb. 2022), pp. 75–82. ISSN: 0167-8655.

[27] Lorraine Chambers, Mohamed Medhat Gaber, and Zahraa S. Abdallah. "DeepStreamCE: A Streaming Approach to Concept Evolution Detection in Deep Neural Networks". In: *arXiv:2004.04116 [cs, stat]* (Apr. 2020). arXiv: 2004.04116.

[28] Lorraine Chambers, Mohamed Medhat Gaber, and Hossein Ghomeshi. "AdaDeepStream: streaming adaptation to concept evolution in deep neural networks". In: *Applied Intelligence* 53.22 (Nov. 2023), pp. 27323–27343. ISSN: 1573-7497.

[29] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM Computing Surveys* 41.3 (July 2009), 15:1–15:58. ISSN: 0360-0300.

[30] Arslan Chaudhry et al. "Efficient Lifelong Learning with A-GEM". en. In: *7th international conference on learning representations*. New Orleans, LA, USA, May 2019.

[31] Arslan Chaudhry et al. "On Tiny Episodic Memories in Continual Learning". In: *33rd Conference on Neural Information Processing Systems*. Vancouver, Canada: NeurIPS, June 2019.

[32] Arslan Chaudhry et al. "Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 532–547.

[33] Bryant Chen et al. "Detecting Backdoor Attacks on Deep Neural Networks by Activation Clustering". en. In: *Workshop on Artificial Intelligence Safety 2019 co-located with the Thirty-Third AAAI Conference on Artificial Intelligence*. 00004. Honolulu, Hawaii: CEUR-WS.org, 2019, p. 8.

[34] Chih-Hong Cheng, Georg Nührenberg, and Hirotoshi Yasuoka. "Runtime Monitoring Neuron Activation Patterns". In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. ISSN: 1558-1101. Mar. 2019, pp. 300–303.

[35] Yonggi Cho et al. "Multi-Cat Monitoring System Based on Concept Drift Adaptive Machine Learning Architecture". en. In: *Sensors* 23.21 (Jan. 2023). Number: 21 Publisher: Multidisciplinary Digital Publishing Institute, p. 8852. ISSN: 1424-8220.

[36] Francois Chollet. *keras*. original-date: 2015-03-28T00:35:42Z. June 2021. URL: https://github.com/keras-team/keras (visited on 06/18/2021).

[37] Scott E. Coull and Christopher Gardner. "Activation Analysis of a Byte-Based Deep Neural Network for Malware Classification". In: *2019 IEEE Security and Privacy Workshops (SPW)*. May 2019, pp. 21–27.

[38] T. Cover and P. Hart. "Nearest neighbor pattern classification". In: *IEEE Transactions on Information Theory* 13.1 (Jan. 1967). Conference Name: IEEE Transactions on Information Theory, pp. 21–27. ISSN: 1557-9654.

[39] Dominic Cushnan et al. "An overview of the National COVID-19 Chest Imaging Database: data quality and cohort analysis". In: *GigaScience* 10.11 (Nov. 2021), giab076. ISSN: 2047-217X.

[40] Abhinandan S. Das et al. "Google news personalization: scalable online collaborative filtering". In: *Proceedings of the 16th international conference on World Wide Web*. WWW '07. New York, NY, USA: Association for Computing Machinery, May 2007, pp. 271–280. ISBN: 978-1-59593-654-7.

[41] Akshay Raj Dhamija, Manuel Günther, and Terrance Boult. "Reducing Network Agnostophobia". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 9157–9168.

[42] Salah Ud Din and Junming Shao. "Exploiting evolving micro-clusters for data stream classification with emerging class detection". en. In: *Information Sciences* 507 (Jan. 2020), pp. 404–420. ISSN: 0020-0255.

[43] Salah Ud Din et al. "Data stream classification with novel class detection: a review, comparison and challenges". en. In: *Knowledge and Information Systems* 63.9 (Sept. 2021), pp. 2231–2276. ISSN: 0219-3116.

[44] Simone Disabato and Manuel Roveri. "Learning Convolutional Neural Networks in presence of Concept Drift". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. ISSN: 2161-4407. July 2019, pp. 1–8.

[45] Gregory Ditzler et al. "Learning in Nonstationary Environments: A Survey". In: *IEEE Computational Intelligence Magazine* 10.4 (Nov. 2015). 00315, pp. 12–25. ISSN: 1556-6048.

[46] Andrija Djurisic et al. “Extremely Simple Activation Shaping for Out–of–Distribution Detection”. In: *International Conference on Learning Representations (ICLR) 2023*. ICLR, Sept. 2023.

[47] Pedro Domingos and Geoff Hulten. “Mining high-speed data streams”. en. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*. Boston, Massachusetts, United States: ACM Press, 2000, pp. 71–80. ISBN: 978-1-58113-233-5.

[48] Alexey Dosovitskiy et al. “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. In: ICLR, Oct. 2020.

[49] Sahibsingh A. Dudani. “The Distance-Weighted k-Nearest-Neighbor Rule”. In: *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6.4 (Apr. 1976), pp. 325–327. ISSN: 2168-2909.

[50] Elaine Ribeiro de Faria, André Carlos Ponce de Leon Ferreira Carvalho, and João Gama. “MINAS: multiclass learning algorithm for novelty detection in data streams”. en. In: *Data Mining and Knowledge Discovery* 30.3 (May 2016), pp. 640–680. ISSN: 1573-756X.

[51] Paul Fergus and Carl Chalmers. “Introduction to Deep Learning”. en. In: *Applied Deep Learning: Tools, Techniques, and Implementation*. Ed. by Paul Fergus and Carl Chalmers. Computational Intelligence Methods and Applications. Cham: Springer International Publishing, 2022, pp. 141–171. ISBN: 978-3-031-04420-5. DOI: 10.1007/978-3-031-04420-5_6.

[52] Max Ferguson et al. “Automatic localization of casting defects with convolutional neural networks”. In: *2017 IEEE International Conference on Big Data (Big Data)*. Dec. 2017, pp. 1726–1735.

[53] D Forsyth. *Computer Vision: a modern approach*. EN. Archive Location: world. Prentice hall, 2011.

[54] Isvani Frías-Blanco et al. “Online and Non-Parametric Drift Detection Methods Based on Hoeffding’s Bounds”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.3 (Mar. 2015). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 810–823. ISSN: 1558-2191.

[55] Björn Friedrich, Taishi Sawabe, and Andreas Hein. "Unsupervised statistical concept drift detection for behaviour abnormality detection". en. In: *Applied Intelligence* 53.3 (Feb. 2023), pp. 2527–2537. ISSN: 1573-7497.

[56] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. "A Survey of Classification Methods in Data Streams". en. In: *Data Streams: Models and Algorithms*. Ed. by Charu C. Aggarwal. Advances in Database Systems. Boston, MA: Springer US, 2007, pp. 39–59. ISBN: 978-0-387-47534-9.

[57] João Gama et al. "A survey on concept drift adaptation". en. In: *ACM Computing Surveys* 46.4 (Mar. 2014), pp. 1–37. ISSN: 03600300.

[58] João Gama et al. "Learning with Drift Detection". en. In: *Advances in Artificial Intelligence – SBIA 2004*. Ed. by Ana L. C. Bazzan and Sofiane Labidi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 286–295. ISBN: 978-3-540-28645-5.

[59] GamaJoão et al. "A survey on concept drift adaptation". EN. In: *ACM Computing Surveys (CSUR)* (Mar. 2014).

[60] Yang Gao et al. "Adaptive Image Stream Classification via Convolutional Neural Network with Intrinsic Similarity Metrics". en. In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. London: ACM, Aug. 2018.

[61] Yang Gao et al. "SACCOS: A Semi-Supervised Framework for Emerging Class Detection and Concept Drift Adaption Over Data Streams". In: *IEEE Transactions on Knowledge and Data Engineering* 34.3 (Mar. 2022). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1416–1426. ISSN: 1558-2191.

[62] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for autonomous driving? The KITTI vision benchmark suite". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. ISSN: 1063-6919. June 2012, pp. 3354–3361.

[63] Chuanxing Geng, Sheng-Jun Huang, and Songcan Chen. "Recent Advances in Open Set Recognition: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1–1. ISSN: 1939-3539.

[64] Ross Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: 2014, pp. 580–587.

[65] Heitor Murilo Gomes et al. "Machine learning for streaming data: state of the art, challenges, and opportunities". In: *ACM SIGKDD Explorations Newsletter* 21.2 (Nov. 2019), pp. 6–22. ISSN: 1931-0145.

[66] Paulo M. Gonçalves et al. "A comparative study on concept drift detectors". en. In: *Expert Systems with Applications* 41.18 (Dec. 2014). 00073, pp. 8144–8156. ISSN: 0957-4174.

[67] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. en. MIT Press, Nov. 2016. ISBN: 978-0-262-33737-3.

[68] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *3rd International Conference on Learning Representations*. San Diego, CA, USA, May 2015.

[69] Ian J. Goodfellow et al. "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks". In: *International Conference on Learning Representations (ICLR) 2014*. Banff, Canada: ICLR, Mar. 2015.

[70] A. Ardeshir Goshtasby. "Image Descriptors". en. In: *Image Registration: Principles, Tools and Methods*. Ed. by A. Ardeshir Goshtasby. Advances in Computer Vision and Pattern Recognition. London: Springer, 2012, pp. 219–246. ISBN: 978-1-4471-2458-0.

[71] Ömer Gözüaçık and Fazli Can. "Concept learning using one-class classifiers for implicit drift detection in evolving data streams". en. In: *Artificial Intelligence Review* 54.5 (June 2021), pp. 3725–3747. ISSN: 1573-7462.

[72] Diana Haidar and Mohamed Medhat Gaber. "Data Stream Clustering for Real-Time Anomaly Detection: An Application to Insider Threats". en. In:

Clustering Methods for Big Data Analytics: Techniques, Toolboxes and Applications. Ed. by Olfa Nasraoui and Chiheb-Eddine Ben N'Cir. Unsupervised and Semi-Supervised Learning. 00001. Cham: Springer International Publishing, 2019, pp. 115–144. ISBN: 978-3-319-97864-2.

[73] Ahsanul Haque, Latifur Khan, and Michael Baron. "SAND: Semi-Supervised Adaptive Novel Class Detection and Classification over Data Stream". en. In: *Thirtieth AAAI Conference on Artificial Intelligence*. Feb. 2016.

[74] Ahsanul Haque et al. "Efficient handling of concept drift and concept evolution over Stream Data". In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. May 2016, pp. 481–492.

[75] Vahid Hashemi et al. "Gaussian–Based Runtime Detection of Out–of–distribution Inputs for Neural Networks". en. In: *Runtime Verification*. Ed. by Lu Feng and Dana Fisman. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 254–264. ISBN: 978-3-030-88494-9.

[76] Tyler L. Hayes, Nathan D. Cahill, and Christopher Kanan. "Memory Efficient Experience Replay for Streaming Learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. ISSN: 2577-087X. May 2019, pp. 9769–9776.

[77] Tyler L. Hayes and Christopher Kanan. "Lifelong Machine Learning With Deep Streaming Linear Discriminant Analysis". In: 2020, pp. 220–221.

[78] Tyler L. Hayes et al. "REMIND Your Neural Network to Prevent Catastrophic Forgetting". en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Vol. 12353. Series Title: Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 466–483. ISBN: 978-3-030-58597-6.

[79] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: 2016, pp. 770–778.

[80] Mohsen Heidari, Mohammad Hossein Moattar, and Hamidreza Ghaffari. "Forward propagation dropout in deep neural networks using Jensen–Shannon and random forest feature importance ranking". In: *Neural Networks* 165 (Aug. 2023), pp. 238–247. ISSN: 0893-6080.

[81] Dan Hendrycks and Kevin Gimpel. "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: *5th International Conference in Learning Representations*. Toulon, France, Apr. 2017.

[82] James Henrydoss et al. "Enhancing Open-Set Recognition using Clustering-based Extreme Value Machine (C-EVM)". In: *2020 IEEE International Conference on Big Data (Big Data)*. Dec. 2020, pp. 441–448.

[83] Thomas A Henzinger, Anna Lukina, and Christian Schilling. "Outside the Box: Abstraction-Based Monitoring of Neural Networks". en. In: *European Conference on Artificial Intelligence*. Santiago de Compostela, 2020.

[84] G. E. Hinton. "Reducing the Dimensionality of Data with Neural Networks". en. In: *Science* 313.5786 (July 2006), pp. 504–507. ISSN: 0036-8075, 1095-9203.

[85] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. "Distilling the knowledge in a neural network". In: *NIPS deep learning and representation learning workshop*. 2015.

[86] T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. "Learning from streaming data with concept drift and imbalance: an overview". en. In: *Progress in Artificial Intelligence* 1.1 (Apr. 2012), pp. 89–101. ISSN: 2192-6360.

[87] F. M. Hohman et al. "Visual Analytics in Deep Learning: An Interrogative Survey for the Next Frontiers". In: *IEEE Transactions on Visualization and Computer Graphics* (2018). 00041, pp. 1–1. ISSN: 1077-2626.

[88] Fred Hohman et al. "Summit: Scaling Deep Learning Interpretability by Visualizing Activation and Attribution Summarizations". In: *IEEE Transactions on Visualization and Computer Graphics* 26.1 (Jan. 2020). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 1096–1106. ISSN: 1941-0506.

[89] Saihui Hou et al. "Learning a Unified Classifier Incrementally via Rebalancing". In: 2019, pp. 831–839.

[90] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *arXiv:1704.04861 [cs]* (Apr. 2017). arXiv: 1704.04861.

[91] Geoff Hulten, Laurie Spencer, and Pedro Domingos. "Mining time-changing data streams". en. In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '01*. San Francisco, California: ACM Press, 2001, pp. 97–106. ISBN: 978-1-58113-391-2.

[92] Ahmet Iscen et al. "Memory-Efficient Incremental Learning Through Feature Adaptation". en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 699–715. ISBN: 978-3-030-58517-4.

[93] M. Kahng et al. "ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models". In: *IEEE Transactions on Visualization and Computer Graphics* 24.1 (Jan. 2018). 00000, pp. 88–97. ISSN: 1077-2626.

[94] Alex Kantchelian et al. "Approaches to adversarial drift". In: *Proceedings of the 2013 ACM workshop on Artificial intelligence and security*. AISec '13. New York, NY, USA: Association for Computing Machinery, Nov. 2013, pp. 99–110. ISBN: 978-1-4503-2488-5.

[95] Imen Khamassi et al. "Discussion and review on evolving data streams and concept drift adapting". en. In: *Evolving Systems* 9.1 (Mar. 2018), pp. 1–23. ISSN: 1868-6486.

[96] Imen Khamassi et al. "Self-Adaptive Windowing Approach for Handling Complex Concept Drift". en. In: *Cognitive Computation* 7.6 (Dec. 2015), pp. 772–790. ISSN: 1866-9964.

[97] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations*. San Diego, CA, USA: ICLR, May 2015.

[98] James Kirkpatrick et al. "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13 (Mar. 2017). Publisher: Proceedings of the National Academy of Sciences, pp. 3521–3526.

[99] M. Kontaki et al. "Continuous monitoring of distance-based outliers over data streams". In: *2011 IEEE 27th International Conference on Data Engineering*. 00098. Apr. 2011, pp. 135–146.

[100] Simon Kornblith et al. "Similarity of Neural Network Representations Revisited". en. In: *Proceedings of the 36th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, May 2019, pp. 3519–3529.

[101] Lukasz Korycki and Bartosz Krawczyk. "Class-Incremental Experience Replay for Continual Learning under Concept Drift". en. In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. Nashville, TN, USA: IEEE, June 2021, pp. 3644–3653. ISBN: 978-1-66544-899-4.

[102] Łukasz Korycki and Bartosz Krawczyk. "Combining Active Learning and Self-Labeling for Data Stream Mining". en. In: *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*. Ed. by Marek Kurzynski, Michal Wozniak, and Robert Burduk. Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, 2018, pp. 481–490. ISBN: 978-3-319-59162-9. DOI: 10.1007/978-3-319-59162-9_50.

[103] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. University of Toronto, 2009.

[104] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (May 2017), pp. 84–90. ISSN: 0001-0782.

[105] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". en. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 1476-4687.

[106] D Lee. "Google self-driving car hits a bus". en-GB. In: *BBC News* (Feb. 2016).

[107] Kimin Lee et al. "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., 2018, pp. 7167–7177.

[108] Soochan Lee et al. "A neural dirichlet process mixture model for task-free continual learning". en. In: *International Conference on Learning Representations*. 2020.

[109] Xiaoqing Li, Jiansheng Yang, and Jinwen Ma. "Recent developments of content-based image retrieval (CBIR)". In: *Neurocomputing* 452 (Sept. 2021), pp. 675–689. ISSN: 0925-2312.

[110] Zhizhong Li and Derek Hoiem. "Learning without Forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.12 (Dec. 2018). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 2935–2947. ISSN: 1939-3539.

[111] Shiyu Liang, Yixuan Li, and R. Srikant. "Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks". In: *5th International Conference in Learning Representations*. Toulon, France, Apr. 2017.

[112] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical Image Analysis* 42 (Dec. 2017), pp. 60–88. ISSN: 1361-8415.

[113] M. Liu et al. "Towards Better Analysis of Deep Convolutional Neural Networks". In: *IEEE Transactions on Visualization and Computer Graphics* 23.1 (Jan. 2017). 00126, pp. 91–100. ISSN: 1077-2626.

[114] Xiaofeng Liu et al. *Deep Unsupervised Domain Adaptation: A Review of Recent Advances and Perspectives*. arXiv:2208.07422 [cs, eess]. Aug. 2022.

[115] Mohammad Reza Loghmani, Markus Vincze, and Tatiana Tommasi. "Positive-unlabeled learning for open set domain adaptation". en. In: *Pattern Recognition Letters* 136 (Aug. 2020), pp. 198–204. ISSN: 0167-8655.

[116] David Lopez-Paz and Marc' Aurelio Ranzato. "Gradient Episodic Memory for Continual Learning". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[117] Viktor Losing, Barbara Hammer, and Heiko Wersing. "KNN Classifier with Self Adjusting Memory for Heterogeneous Concept Drift". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. ISSN: 2374-8486. Dec. 2016, pp. 291–300.

[118] Viktor Losing, Barbara Hammer, and Heiko Wersing. "Tackling heterogeneous concept drift with the Self-Adjusting Memory (SAM)". en. In: *Knowledge and Information Systems* 54.1 (Jan. 2018), pp. 171–201. ISSN: 0219-3116.

[119] J. Lu et al. "Learning under Concept Drift: A Review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (Dec. 2019). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 2346–2363. ISSN: 1558-2191.

[120] Jiajun Lu, Theerasit Issaranon, and David Forsyth. "SafetyNet: Detecting and Rejecting Adversarial Examples Robustly". In: 2017, pp. 446–454.

[121] Zheda Mai et al. "Batch-level Experience Replay with Review for Continual Learning". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. Seattle, USA: IEEE, July 2020.

[122] Zheda Mai et al. "Online continual learning in image classification: An empirical survey". en. In: *Neurocomputing* 469 (Jan. 2022), pp. 28–51. ISSN: 0925-2312.

[123] Mohammad Masud et al. "Classification and Novel Class Detection in Concept-Drifting Data Streams under Time Constraints". In: *IEEE Transactions on Knowledge and Data Engineering* 23.6 (June 2011), pp. 859–874. ISSN: 2326-3865.

[124] John McCarthy et al. "A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence, August 31, 1955". en. In: *AI Magazine* 27.4 (Dec. 2006). Number: 4, pp. 12–12. ISSN: 2371-9621.

[125] Leland McInnes, John Healy, and James Melville. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. Sept. 2020.

[126] McKinsey. *What is IoT: The Internet of Things explained | McKinsey*. Aug. 2022. URL: https://www.mckinsey.com/featured-insights/mckinsey-explainers/what-is-the-internet-of-things (visited on 03/23/2023).

[127] Thomas Mensink et al. "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.11 (Nov. 2013). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 2624–2637. ISSN: 1939-3539.

[128] Lassi Meronen, Christabella Irwanto, and Arno Solin. "Stationary Activations for Uncertainty Calibration in Deep Learning". In: *Advances in Neural Information Processing Systems*. Vol. 33. Curran Associates, Inc., 2020, pp. 2338–2350.

[129] Golnaz Moallem et al. "An explainable deep vision system for animal classification and detection in trail-camera images with automatic post-deployment retraining". In: *Knowledge-Based Systems* 216 (Mar. 2021), p. 106815. ISSN: 0950-7051.

[130] Mehdi Mohammadi et al. "Deep Learning for IoT Big Data and Streaming Analytics: A Survey". In: *IEEE Communications Surveys Tutorials* 20.4 (2018), pp. 2923–2960. ISSN: 2373-745X.

[131] Xin Mu, Kai Ming Ting, and Zhi-Hua Zhou. "Classification Under Streaming Emerging New Classes: A Solution Using Completely-Random Trees". In: *IEEE Transactions on Knowledge and Data Engineering* 29.8 (Aug. 2017). Conference Name: IEEE Transactions on Knowledge and Data Engineering, pp. 1605–1618. ISSN: 1558-2191.

[132] Xin Mu et al. "Streaming Classification with Emerging New Class by Class Matrix Sketching". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 31.1 (Feb. 2017). Number: 1. ISSN: 2374-3468.

[133] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. en. "O'Reilly Media, Inc.", Sept. 2016. ISBN: 978-1-4493-6990-3.

[134] National Highway Traffic Safety Administration NHTSA. *Summary Report: Standing General Order on Crash Reporting for Level 2 Advanced Driver Assistance Systems*. en. May 2022. URL: https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-06/ADAS-L2-SGO-Report-June-2022.pdf (visited on 09/17/2023).

[135] G. S. Nijaguna et al. "Quantum Fruit Fly algorithm and ResNet50-VGG16 for medical diagnosis". In: *Applied Soft Computing* 136 (Mar. 2023), p. 110055. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2023.110055.

[136] Julia Nitsch et al. “Out-of-Distribution Detection for Automotive Perception”. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. Sept. 2021, pp. 2938–2943.

[137] Chris Olah et al. “The Building Blocks of Interpretability”. en. In: *Distill* 3.3 (Mar. 2018). 00081, e10. ISSN: 2476-0757.

[138] Bartłomiej Olber et al. “Detection of Out-of-Distribution Samples Using Binary Neuron Activation Patterns”. en. In: 2023, pp. 3378–3387.

[139] Eng-Jon Ong, Sameed Husain, and Miroslaw Bober. “Understanding the Distributions of Aggregation Layers in Deep Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 1–15. ISSN: 2162-2388.

[140] Eng-Jon Ong, Sameed Husain, and Miroslaw Bober. “Understanding the Distributions of Aggregation Layers in Deep Neural Networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* 35.4 (Apr. 2024). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 5536–5550. ISSN: 2162-2388.

[141] Yaniv Ovadia et al. “Can you trust your model’ s uncertainty? Evaluating predictive uncertainty under dataset shift”. In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[142] E. S. Page. “Continuous Inspection Schemes”. In: *Biometrika* 41.1/2 (1954). Publisher: [Oxford University Press, Biometrika Trust], pp. 100–115. ISSN: 0006-3444.

[143] Sankar K. Pal et al. “Deep learning in multi-object detection and tracking: state of the art”. en. In: *Applied Intelligence* 51.9 (Sept. 2021), pp. 6400–6429. ISSN: 1573-7497.

[144] Shaoning Pang, S. Ozawa, and N. Kasabov. “Incremental linear discriminant analysis for classification of data streams”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 35.5 (Oct. 2005). Conference Name:

IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), pp. 905–914. ISSN: 1941-0492.

[145] Nicolas Papernot and Patrick McDaniel. "Deep k-Nearest Neighbors: Towards Confident, Interpretable and Robust Deep Learning". In: *arXiv:1803.04765 [cs, stat]* (Mar. 2018). arXiv: 1803.04765.

[146] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems*. Vol. 32. Curran Associates, Inc., 2019.

[147] Colin Paterson, Radu Calinescu, and Chiara Picardi. "Detection and Mitigation of Rare Subclasses in Deep Neural Network Classifiers". In: *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. Aug. 2021, pp. 9–16.

[148] Janis Postels et al. "Sampling-Free Epistemic Uncertainty Estimation Using Approximated Variance Propagation". In: 2019, pp. 2931–2940.

[149] Ameya Prabhu, Philip H. S. Torr, and Puneet K. Dokania. "GDumb: A Simple Approach that Questions Our Progress in Continual Learning". en. In: *Computer Vision – ECCV 2020*. Ed. by Andrea Vedaldi et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2020, pp. 524–540. ISBN: 978-3-030-58536-5.

[150] Yuxian Qiu et al. "Adversarial Defense Through Network Profiling Based Path Extraction". In: *IEEE Conference on Computer Vision and Pattern Recognition*. Long Beach, CA, USA: Computer Vision Foundation / IEEE, June 2019.

[151] J. R. Quinlan. "Induction of decision trees". en. In: *Machine Learning* 1.1 (Mar. 1986), pp. 81–106. ISSN: 1573-0565.

[152] Christoph Raab, Moritz Heusinger, and Frank-Michael Schleif. "Reactive Soft Prototype Computing for Concept Drift Streams". en. In: *Neurocomputing* 416 (Nov. 2020), pp. 340–351. ISSN: 0925-2312.

[153] Maithra Raghu et al. "Do Vision Transformers See Like Convolutional Neural Networks?" In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 12116–12128.

[154] Jathushan Rajasegaran et al. "iTAML: An Incremental Task-Agnostic Meta-learning Approach". en. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, June 2020, pp. 13585–13594. ISBN: 978-1-72817-168-5.

[155] The MIT Press Reader. *Surveillance, Companionship, and Entertainment: The Ancient History of Intelligent Machines*. en. Nov. 2021. URL: https://thereader.mitpress.mit.edu/the-ancient-history-of-intelligent-machines/ (visited on 09/20/2023).

[156] Sylvestre–Alvise Rebuffi et al. "iCaRL: Incremental Classifier and Representation Learning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2017, pp. 2001–2010.

[157] Matthew Riemer et al. "Learning to Learn without Forgetting by Maximizing Transfer and Minimizing Interference". en. In: *International Conference on Learning Representations*. 2019.

[158] Herbert Robbins and Sutton Monro. "A Stochastic Approximation Method". In: *The Annals of Mathematical Statistics* 22.3 (1951). Publisher: Institute of Mathematical Statistics, pp. 400–407. ISSN: 0003-4851.

[159] S. W. Roberts. "Control Chart Tests Based on Geometric Moving Averages". In: *Technometrics* 42.1 (Feb. 2000), pp. 97–101. ISSN: 0040-1706.

[160] Martha Roseberry, Bartosz Krawczyk, and Alberto Cano. "Multi-Label Punitive kNN with Self-Adjusting Memory for Drifting Data Streams". In: *ACM Transactions on Knowledge Discovery from Data* 13.6 (Nov. 2019), 60:1–60:31. ISSN: 1556-4681.

[161] Deboleena Roy, Priyadarshini Panda, and Kaushik Roy. "Tree-CNN: A hierarchical Deep Convolutional Neural Network for incremental learning". In: *Neural Networks* 121 (Jan. 2020), pp. 148–160. ISSN: 0893-6080.

[162] E. M. Rudd et al. "The Extreme Value Machine". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.3 (Mar. 2018). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 762–768. ISSN: 1939-3539.

[163] Shashidhar Rudregowda et al. "Visual Speech Recognition for Kannada Language Using VGG16 Convolutional Neural Network". en. In: *Acoustics* 5.1 (Mar. 2023). Number: 1 Publisher: Multidisciplinary Digital Publishing Institute, pp. 343–353. ISSN: 2624-599X.

[164] Sid Ryan et al. "Deep Learning Versus Conventional Learning in Data Streams with Concept Drifts". In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. Dec. 2019, pp. 1306–1313.

[165] Mohammadreza Salehi et al. "A Unified Survey on Anomaly, Novelty, Open-Set, and Out-of-Distribution Detection: Solutions and Future Challenges". In: *Transactions on Machine Learning Research* 2022.2022 (Nov. 2022).

[166] W. Samek et al. "Evaluating the Visualization of What a Deep Neural Network Has Learned". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.11 (Nov. 2017). 00168, pp. 2660–2673. ISSN: 2162-237X.

[167] Soma Sarker, Sree Nirmillo Biswash Tushar, and Heping Chen. "High accuracy keyway angle identification using VGG16-based learning method". In: *Journal of Manufacturing Processes* 98 (July 2023), pp. 223–233. ISSN: 1526-6125.

[168] Walter J. Scheirer, Lalit P. Jain, and Terrance E. Boult. "Probability Models for Open Set Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.11 (Nov. 2014). 00199, pp. 2317–2324. ISSN: 1939-3539.

[169] Jeffrey C. Schlimmer and Richard H. Granger. "Incremental learning from noisy data". en. In: *Machine Learning* 1.3 (Sept. 1986). 00598, pp. 317–354. ISSN: 1573-0565.

[170] Neha Sharma, Vibhor Jain, and Anju Mishra. "An Analysis Of Convolutional Neural Networks For Image Classification". In: *Procedia Computer Science*. International Conference on Computational Intelligence and Data Science 132 (Jan. 2018), pp. 377–384. ISSN: 1877-0509.

[171] Piyush K. Sharma and Adrienne Raglin. "IoT: Smart City Parking Solutions with Metric-Chisini-Jensen-Shannon Divergence based Kernels". In: *MILCOM 2019 - 2019 IEEE Military Communications Conference (MILCOM)*. ISSN: 2155-7586. Nov. 2019, pp. 324–330.

[172] Lei Shu, Hu Xu, and Bing Liu. "DOC: Deep Open Classification of Text Documents". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017.

[173] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *ICLR 2015*. San Diego, Apr. 2015.

[174] SITNFlash. *The History of Artificial Intelligence*. en-US. Aug. 2017. URL: https://sitn.hms.harvard.edu/flash/2017/history-artificial-intelligence/.

[175] Philip Sperl, Jan-Philipp Schulze, and Konstantin Böttinger. "Activation Anomaly Analysis". en. In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Frank Hutter et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 69–84. ISBN: 978-3-030-67661-2.

[176] Paweł Staszewski et al. "A New Approach to Descriptors Generation for Image Retrieval by Analyzing Activations of Deep Neural Network Layers". In: *IEEE Transactions on Neural Networks and Learning Systems* (2021), pp. 1–8. ISSN: 2162-2388.

[177] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. "One Pixel Attack for Fooling Deep Neural Networks". In: *IEEE Transactions on Evolutionary Computation* 23.5 (Oct. 2019). Conference Name: IEEE Transactions on Evolutionary Computation, pp. 828–841. ISSN: 1941-0026.

[178] Yiyou Sun, Chuan Guo, and Yixuan Li. "ReAct: Out-of-distribution Detection With Rectified Activations". In: *Advances in Neural Information Processing Systems*. Vol. 34. Curran Associates, Inc., 2021, pp. 144–157.

[179] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning, second edition: An Introduction*. en. Google-Books-ID: uWV0DwAAQBAJ. MIT Press, Nov. 2018. ISBN: 978-0-262-35270-3.

[180] Christian Szegedy et al. "Inception-v4, inception-ResNet and the impact of residual connections on learning". In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI'17. San Francisco, California, USA: AAAI Press, Feb. 2017, pp. 4278–4284.

[181] R Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[182] Péter Szikora and Nikolett Madarász. "Self-driving cars — The human side". In: *2017 IEEE 14th International Scientific Conference on Informatics*. Nov. 2017, pp. 383–387.

[183] Abu Md Niamul Taufique, Chowdhury Sadman Jahan, and Andreas Savakis. "Unsupervised Continual Learning for Gradually Varying Domains". en. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. New Orleans, LA, USA: IEEE, June 2022, pp. 3739–3749. ISBN: 978-1-66548-739-9.

[184] TensorSpace. *TensorSpace Playground*. 2019. URL: https://tensorspace.org/html/playground/index.html (visited on 10/26/2023).

[185] NLR The National Law Review. *Autonomous or Self-Driving Vehicle Safety Risks*. en. May 2021. URL: https://www.natlawreview.com/article/dangers-driverless-cars (visited on 04/03/2024).

[186] Luan Tran, Liyue Fan, and Cyrus Shahabi. "Distance-based outlier detection in data streams". en. In: *Proceedings of the VLDB Endowment* 9.12 (Aug. 2016). 00032, pp. 1089–1100. ISSN: 21508097.

[187] Alexey Tsymbal. *The Problem of Concept Drift: Definitions and Related Work*. Tech. rep. 00895. Trinity College Dublin, 2004.

[188] Furkan Ulger et al. "Fine-Grained Classification of Solder Joints With \alpha-Skew Jensen–Shannon Divergence". In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* 13.2 (Feb. 2023), pp. 257–264. ISSN: 2156-3985.

[189] L Van der Maaten and Geoffrey Hinton. "Visualizing data using t-SNE". In: *Journal of Machine Learning Research* 9.11 (2008).

[190] Jeffrey S. Vitter. "Random sampling with a reservoir". In: *ACM Transactions on Mathematical Software* 11.1 (Mar. 1985), pp. 37–57. ISSN: 0098-3500.

[191] Apoorv Vyas et al. "Out-of-Distribution Detection Using an Ensemble of Self Supervised Leave-out Classifiers". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 550–564.

[192] Dequan Wang et al. "Tent: Fully Test-Time Adaptation by Entropy Minimization". en. In: *International Conference on Learning Representations*. Jan. 2023.

[193] Yaxing Wang et al. "Transferring GANs: generating images from limited data". In: 2018, pp. 218–234.

[194] Zhixiong Wang and Wei Wang. "Concept Drift Detection Based on Kolmogorov—Smirnov Test". en. In: *Artificial Intelligence in China*. Ed. by Qilian Liang et al. Lecture Notes in Electrical Engineering. Singapore: Springer, 2020, pp. 273–280.

[195] Zhuoyi Wang et al. "Robust High Dimensional Stream Classification with Novel Class Detection". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. ISSN: 2375-026X. Apr. 2019, pp. 1418–1429.

[196] Gerhard Widmer and Miroslav Kubat. "Learning in the Presence of Concept Drift and Hidden Contexts". en. In: *Machine Learning* 23.1 (Apr. 1996), pp. 69–101. ISSN: 1573-0565.

[197] Svante Wold, Kim Esbensen, and Paul Geladi. "Principal component analysis". In: *Chemometrics and Intelligent Laboratory Systems*. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists 2.1 (Aug. 1987), pp. 37–52. ISSN: 0169-7439.

[198] Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion–MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: *arXiv:1708.07747* (Sept. 2017).

[199] D Yadron and D Tynan. *Tesla driver dies in first fatal crash while using autopilot mode*. en. Section: Technology. June 2016. URL: http://www.theguardian.com/technology/2016/jun/30/tesla-autopilot-death-self-driving-car-elon-musk (visited on 06/28/2021).

[200] Myuu Myuu Wai Yan. "Accurate detecting concept drift in evolving data streams". en. In: *ICT Express* 6.4 (Dec. 2020), pp. 332–338. ISSN: 2405-9595.

[201] Le Yang et al. "A new model based on improved VGG16 for corn weed identification". English. In: *Frontiers in Plant Science* 14 (July 2023). Publisher: Frontiers. ISSN: 1664-462X. DOI: 10.3389/fpls.2023.1205151.

[202] Jaehong Yoon et al. "Lifelong Learning with Dynamically Expandable Networks". en. In: *International Conference on Learning Representations* (2018), p. 11.

[203] Ryota Yoshihashi et al. "Classification-Reconstruction Learning for Open-Set Recognition". In: 2019, pp. 4016–4025.

[204] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems*. Vol. 27. Curran Associates, Inc., 2014.

[205] Liheng Yuan et al. "Recent Advances in Concept Drift Adaptation Methods for Deep Learning". en. In: *Thirty-First International Joint Conference on Artificial Intelligence*. Vol. 6. ISSN: 1045-0823. July 2022, pp. 5654–5661.

[206] Friedemann Zenke, Ben Poole, and Surya Ganguli. "Continual Learning Through Synaptic Intelligence". en. In: *Proceedings of the 34th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2017, pp. 3987–3995.

[207] Jianjun Zhang et al. "KNNENS: A k-Nearest Neighbor Ensemble-Based Method for Incremental Learning Under Data Stream With Emerging New Classes". In: *IEEE Transactions on Neural Networks and Learning Systems* (2022). Conference Name: IEEE Transactions on Neural Networks and Learning Systems, pp. 1–8. ISSN: 2162-2388.

[208] Junting Zhang et al. "Class-incremental Learning via Deep Model Consolidation". en. In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. Snowmass Village, CO, USA: IEEE, Mar. 2020, pp. 1120–1129. ISBN: 978-1-72816-553-0.

[209] Zhuo Zhang et al. "Dataset and Baselines for IID and OOD Image Classification Considering Data Quality and Evolving Environments". en. In: *International Journal of Interactive Multimedia and Artificial Intelligence* 8.Special Issue

on AI-driven Algorithms and Applications in the Dynamic and Evolving Environments (2023). ISSN: 1989-1660.

[210] Bowen Zhao et al. "Maintaining Discrimination and Fairness in Class Incremental Learning". In: 2020, pp. 13208–13217.

[211] Miaoyun Zhao, Yulai Cong, and Lawrence Carin. "On Leveraging Pretrained GANs for Generation with Limited Data". en. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, pp. 11340–11351.

[212] Yong-Nan Zhu and Yu-Feng Li. "Semi-Supervised Streaming Learning with Emerging New Labels". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34.04 (Apr. 2020). Number: 04, pp. 7015–7022. ISSN: 2374-3468.

[213] Indrė Žliobaitė, Mykola Pechenizkiy, and João Gama. "An Overview of Concept Drift Applications". In: *Big Data Analysis: New Algorithms for a New Society. SBD*. Vol. 16. Springer, Jan. 2016, pp. 91–114. ISBN: 978-3-319-26987-0.

[214] Alaettin Zubaroğlu and Volkan Atalay. "Data stream clustering: a review". en. In: *Artificial Intelligence Review* 54.2 (Feb. 2021), pp. 1201–1236. ISSN: 1573-7462.