A Novel Machine Learning Approach For File Fragments Classification

Alia Algurashi

A thesis presented for the degree of
Doctor of Philosophy

School of Computing Sciences
University of East Anglia
United Kingdom
November, 2022

Abstract

Identifying types of manipulated or corrupted file fragments in isolation from their context is an essential task in digital forensics. In traditional file type identification, metadata, such as file extensions and header and footer signatures, is used. Traditional metadata-based approaches do not work where metadata is missing or altered, therefore some alternative strategies and approaches need to be applied or developed to solve the problem.

One approach is to apply some statistical techniques to extract features from the binary contents of file fragments and then use them as inputs for classification algorithms. This results in high dimensionality, causing learning and classification to be time-consuming. Another approach is deep learning neural networks, which extract features automatically. File fragment classification is further complicated by the high number of possible file classes. Also, some container file types, such as Powerpoint (PPT) include data belonging to other file types, such as JPEG, which can confuse the classification algorithms.

In this thesis, we developed a hybrid method to address high feature dimensionality. We use filters and wrappers to reduce the number of features. We explored the possible hierarchical relationships between file classes and we represent them with a hierarchy tree to help narrow the uncertainties for challenging file types. We proposed a novel hybrid approach that combines hierarchical models with feature selection to improve the accuracy of file fragment classification. We also explored the use of deep learning techniques for this task.

We test our methods using a benchmark dataset - GovDocs. The results from hybrid feature selection show a reduction in the number of features from 66,313 to 11–32, and provide improved accuracy compared to methods using all features. The accuracy increased from 69% using random forest to 75% using the DAG tree. We incorporate the hybrid feature selection into hierarchical modelling to generate trees that use only the most discriminative features. We find that these models outperformed classical machine-learning approaches. Finally, using deep learning for file fragment classification provided the highest accuracy of all techniques explored, obtaining accuracies of 86%.

A Novel Machine Learning Approach For File Fragments Classification

Alia Algurashi

Acknowledgements

My faith in God is ultimately what has enabled me to complete this dissertation.

Words cannot express my gratitude to my supervisor Dr. Wenjia Wang for his invaluable guidance and feedback. Also, I would like to extend my thanks to my second supervisor Professor Beatriz De La Iglesia for her advice and comments. Additionally, this endeavour would not have been possible without generous support from the Saudi government, who financed my research.

My gratitude to my family knows no bounds. My parents, my husband and my father-in-law have given me huge financial and emotional support, and my sisters, my brothers and my daughters have kept my spirits and motivation high during the whole process.

Lastly, I'd like to thank my research colleagues for their moral support, as well as the friends I made while attending the University of East Anglia. Thanks should also go to the high-performance computing team and the technical support staff who helped and inspired me.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Introduction

In today's world of data, digital devices generate, store, process, and transfer immense amounts of data. Data is usually stored in computers as sequences of bytes which are called files. Digital files on modern disks are stored in fixed-length blocks called sectors that range in size from hundreds to thousands of bytes. Figure 1.1 is a diagram showing how a computer disk is divided into tracks (A) and each track is further subdivided into sectors (B). Each sector contains a fixed amount of data, typically 512 bytes for hard disk drives (HDDs) and 2048 bytes for CDs and DVDs. A computer file larger than a sector can be saved in many sectors that are rarely consecutive but are usually spread out all over the disk. An operating system uses the file system to control how these files are stored and retrieved. Consequently, when the disk is corrupted and the file system is lost, it will be very difficult to read the data of the file. In this case, it is necessary to recognise the type of file in order to correctly represent the binary data of files.

File types differ depending on how their data have been organised and saved. This means that each file type has its own data structure, which is normally associated with the file name as an extension. For example, PDF, DOC, EXE, TXT, etc. The increasing usage of digital media storage by the public has been accompanied by an increase in the rate of cyber-crimes, which has highlighted the need for digital tools to monitor, control, and recover digital data.

The existing tools for identifying file types use either a file extension or a file signature. A file signature is a sequence of bytes in a standardised file format that indicates the file type.

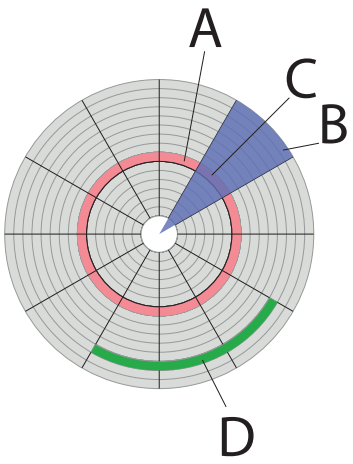

Figure 1.1: Computer disk structure showing how a storage disk is divided into tracks and sectors, as follows:

A) Track

B) Geometrical sector

C) Disk sector

D) Cluster

Table 1.1: Examples of different file signatures

File extension	Signature	Description
JPEG	FF D8 FF E0	JPEG image
ZIP	57 69 6E 5A 69 70	WinZip compressed archive
DOCX	50 4B 03 04 14 00 06 00	MS Office 2007 document
ASF	30 26 B2 75 8E 66 CF 11	Windows Media Audio — Video File

Table 1.1 shows some examples of file type signatures. These signatures are usually located in the first few bytes, as shown in Figure 1.2.

A lot of software applications rely on the operating system that uses file signatures to identify file types by matching a predefined database of well-known file signatures with the signature of an unknown file. The most common application that uses file signatures as a file type identifier is the file command on the Linux operating system [Moody and Erbacher, 2008]. Such file metadata can be easily manipulated, either by malicious software or by end users. Additionally, files do not usually store sequences on discs, but as separated

fragments, where the operating system is responsible for keeping track of file fragments that belong to specific files. When a file system is lost, either deliberately, or by mistake, file type identification becomes difficult, especially for those fragments from the middle of the file, where they do not include any metadata to link them to their parent file types. Therefore, a traditional approach for file type identification would not work well in the case where metadata is manipulated, missing or if the file is heavily fragmented [Underwood, 2009].

An alternative approach to file type classification has been developed, which relies on finding patterns in the binary contents of a file. This approach is called content-based file type identification. McDaniel and Heydari [2003] introduced the idea of analysing the binary content of a file to find a pattern that distinguishes file types, rather than using a file extension or binary signature. Since 2003, many studies have proposed different methodologies to solve the problem of file fragment classification using binary file contents. Most of these methods apply statistical analysis to the byte frequency distribution of fragments to distinguish file types. Others combine statistical analysis with machine learning techniques in an attempt to solve the problem. Although they have addressed the problem of classifying a complete file, it is still the case that file fragment classification is a significant challenge.

1.2 Problem Statement and Motivation

Roussev defined the classification of file fragments as the process of identifying the true file type to which a file fragment belongs [Roussev and Quates, 2013]. The identification of file fragments without relying on file metadata is essential in many digital security tasks, for example, a forensic analyst, who analyses and collects information from sizeable storage, needs a tool that gives a statistical summary of the storage container. While time is a critical factor in digital forensics, this summary will help in making a quick decision if the storage requires further analysis or not. In order to give such an overview, data fragments in the media storage need to be classified accurately. Another example where the ability

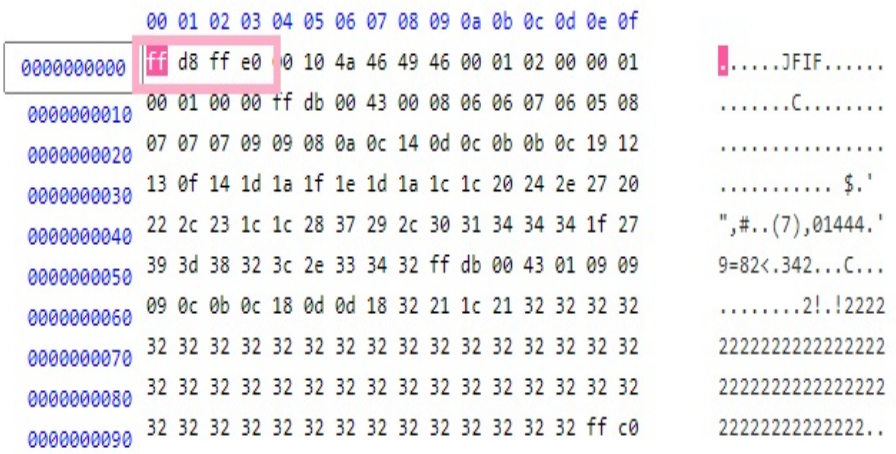

Figure 1.2: An example of a JPEG file header, highlighting the first four bytes which are the same for all JPEG files.

to identify the file type of a data fragment is needed would be a military network where the types of traffic permitted on the network are limited. Monitoring of the network traffic would allow unpermitted traffic to be detected and dealt with. By using file fragment classification in this way, data types can be evaluated without the need for an individual to view the potentially sensitive information contained within the data. A third example where file fragments classification is important is for file carving applications, where it is a useful technique for digital forensics investigations and for simple data recovery. In order to obtain files from raw disk images, file carvers use a database of headers and footers (essentially, strings of bytes at predictable offsets) for specific file types, regardless of the type of file system on the disk image. It is important to note that file carving is possible even if the file system metadata has been lost, so classifying file fragments accurately would significantly help to improve the precision of file carving [Richard III and Roussev, 2005]. The classification of file fragment types is challenging, because integrated metadata that is usually located in a file header and helps identify file types, would not always be available. For example, in fragments originating from the middle of a file, metadata is not present. Additionally, a wide variety of existing file types makes file fragment classification more

complicated. Whereas these file types range from the basic and primitive file types, such as ASCII or JPEG, to the more complicated container files, such as Portable Document Format (PDF), archive files like TAR and ZIP, or compounded archive files [Roussev and Quates, 2013]. Researchers have developed approaches to identify the file types of file fragments without relying on a file's metadata. In summary, previous approaches analyse the patterns contained within the binary contents of each file type and build classifiers based upon this information. However, such binary-content analysis methods produce vast quantities of byte frequency-based features, which can be noisy and uninformative in terms of discriminating the many different file types. Additionally, none of the previous methods were able to correctly classify file fragments with high levels of accuracy, in the absence of their metadata.

1.3 Research Questions

The research questions that we aim to answer are:

1. Does the binary content of the file fragment have a pattern that represents its file type? To study this, we will create experiments to find such patterns in file fragments.

2. Can file fragments be represented as structured data with specific features? To explore this issue, we will review previous work and examine how it represents file fragments as structured data.

3. What are the most suitable features for representing file fragments? This will be studied by creating experiments to analyse file fragment features and using machine learning techniques to find out which features are the most representative.

4. Is there any specific classifier that can classify file fragments based on their binary content correctly and reliably? For this purpose, we will create experiments that compare different classification algorithms that employ file fragment binary content and evaluate their performance.

5. Can hierarchical modelling improve the detection rate for file fragment types? In order to accomplish this, we will create two different hierarchical models to classify file fragments and evaluate them.

6. Can deep learning approaches improve the accuracy of file fragment classification? For this purpose, we will generate two deep learning models and evaluate them.

1.4 Research Aim and Objectives

The main aim of this research is to develop a machine learning based method to accurately classify the type of file fragments. To reach this aim, this research needs to achieve the following specific research objectives:

- To review and analyse critically the relevant literature.
- To create a structured and unstructured dataset of file fragments to evaluate this research methodology.
- To extract various sets of statistical features and quantify their usefulness.
- To explore the use of hybrid hierarchical modelling for file fragment classification problems.
- To explore different deep learning methods for file fragments classification.
- To evaluate the efficiency and effectiveness of our approaches using statistical measures.

1.5 The Novelty Of The Proposed Research

There is no universal method that can accurately classify file fragments. Although some of these studies provide good accuracy as a result of only considering a limited number of file classes, other studies achieved high detection rates by relying on metadata that explicitly indicates which type of file they belonged to. Other studies that do not include metadata in

their experiments were either unsuitable for application to forensics or did not perform to a good degree of accuracy. Therefore, there is a need to develop new methods to classify file fragments that work well in the scenarios described. Our proposed research is completely novel in that it uses hybrid feature selection during hierarchical modelling. This work also represents the first use of the transformer deep learning approach to the file fragment classification problem. We also compare the results of this approach to the CNN method used by others to provide a fair comparison with the existing techniques.

1.6 Contribution

This research provides several contributions including a novel method for file fragment classification. The following summarises the contributions of this research:

1. For file fragments, we developed a hybrid feature selection approach in order to analyse and reduce large structural features. In addition to reducing the number of fragment features needed for classification, this method is also effective in improving classification accuracy. This work was published in International Conference on Innovative Techniques and Applications of Artificial Intelligence, 2019 [Algurashi and Wang, 2019].

2. File Fragments dataset. We have created a novel dataset which can be used to fairly assess the performance of the different approaches explored here. This dataset will also be useful to other researchers conducting similar research. The dataset includes 13 different file types where each file is divided into fragments and each fragment has large features. This data set is available on GitHub Algurashi [2023].

3. We developed a novel hybrid hierarchical model for file fragment classification. Our hybrid hierarchical models have improved the accuracy of file fragment classification compared to other feature-based machine learning approaches. This work was submitted to IEEE Transactions on Information Forensics and Security, 2022.

4. We developed CNN and transformer models for file fragment classification. Deep learning outperforms traditional feature-based machine learning for classifying file fragments. An article has been written about this work and will soon be submitted to IEEE Transactions on Information Forensics and Security.

1.7 Thesis Outlines

The structure of this Thesis is outlined as follows

- Chapter 1: Introduction - describes the research with a focus on motivation, aims, process and contribution.
- Chapter 2: Background - explains the background of different methods, techniques, and algorithms used for file fragment classification.
- Chapter 3: Literature Review - represents the literature of the related work. It will focus on machine learning methods used in file fragments classification
- Chapter 4: Methodology - represents the research methodology and design. In addition, it will describe the data set used in our experiments.
- Chapter 5: Data Representation and Feature Engineering - Presents the process of file transformation into fragments. Also describes and evaluates the feature extraction processes considered. In addition, it explains the methodology used in feature selection including experiments that evaluate the effect of feature selection on classification performance.
- Chapter 6: Hierarchical Approach for file fragment classification - Provides the background of a hierarchy classifier. Includes two types of hierarchical structures, which are Binary Tree (BT) and A Directed Graph (DAG).

- Chapter 7: Deep Learning Approach for File Fragment Classification: Includes two types of deep learning methods, which are a Convolutional Neural Network (CNN) and the transformer.
- Chapter 8: Final Evaluation - provides a statistical comparison of the different structures, strategies, mechanisms, and techniques that are discussed in this thesis.
- Chapter 9: Conclusion and Future Work - The thesis ends by presenting conclusions, discussing the main findings for the work undertaken, and finally discussing the novel contributions and potential directions for future work.

1.8 Summary

In this introductory chapter, we presented a general overview and the research problem background. The motivation for the research includes three real-world scenarios of file fragment classification. Followed by presenting the gap in the previous work and the adopted research methodology. Then the main research questions are listed followed by the main objectives of the research have been defined. Finally, we provide a list of our contributions, including a published paper and a publicly available data set of file fragments. In the next chapter, we provide the reader with a general background of the methods and techniques used for file fragment classification.

2 Background

2.1 Introduction

As noted earlier in the introduction to this thesis, the most reliable approach to file fragment classification in the case of missing informative data is to analyse the binary content of the fragments. A great deal of work has been done in order to solve file fragment classification using binary content. Most previous work on file fragment classification has used Natural Language Processing (NLP) techniques by treating file fragments as a sequence of text.

NLP techniques automatically process natural languages, such as speech and text, by software and algorithms. NLP dates back more than fifty years and has evolved from linguistics with the advent of modern computers [Kornai, 2010]. The NLP approach offers some very powerful mechanisms for extracting patterns from text, such as N-gram analysis. It is important to note that not only statistical methods are used, but NLP methods also employ machine learning techniques to find patterns and use them for classification.

The purpose of this chapter is to guide the reader through the background of different methods, techniques, and algorithms that have been used or may be suitable for file fragments classification tasks. Section 2.2 explains how NLP techniques can be used to process file fragment data. Section 2.3 explains the statistical approaches that can be used to classify file fragments. Section 2.4 details the most frequent machine learning approaches used for file fragment classification. Then, in Section 2.5, we describe several feature selection methods and explain why they are important for the file fragment classification problem. Finally, the summary of this chapter is presented in Section 2.6.

2.2 Processing File Fragments Dataset

To apply NLP techniques for file fragment classification there are two ways to deal with file fragments data, either as structured data or as unstructured data:

1. Structured Data

   The format of structured data is predefined and is highly specific. It is usually presented as a number of rows (data examples) and a number of columns (data features). File fragments can be presented as structured data by extracting features. The most commonly used feature for file fragments is the N-gram. N-gram is a probabilistic approach used to predict the probability for the following item in a sequence (e.g. word, syllable, phoneme etc.) depending on the previous item. N-gram models are now widely used in statistical natural language processing [Manning and Schutze, 1999]. Looking at the binary sequences contained within file fragments and considering them as sequences of text allows NLP techniques to be applied and can be used to distinguish file fragments and assign them to their original file type. In this way, if each byte contains 8 bits, then each can be represented as a number ranging from 0 to 255. Bytes Frequencies Distribution (BFD) is the counting of occurrences of each byte value. BFD calculation is similar to N-gram, whereas the BFD is considered a special type of N-gram (1-gram). In this application, 'N' refers to a number of bytes in a sequence rather than a number of words. Researchers use this approach because they expect that there is a characteristic byte pattern for each type of file. Figure 2.1 shows an example of BFD for different file types. For example, the DOC file has higher frequencies between 0-127, which is the frequency of ASCII characters. Another example, in the HTML file, the bytes representing the characters "<" and ">" will occur more frequently than other byte values. In other words, the consistent frequency values in the file byte frequency distribution are considered strong indicators of the file type.

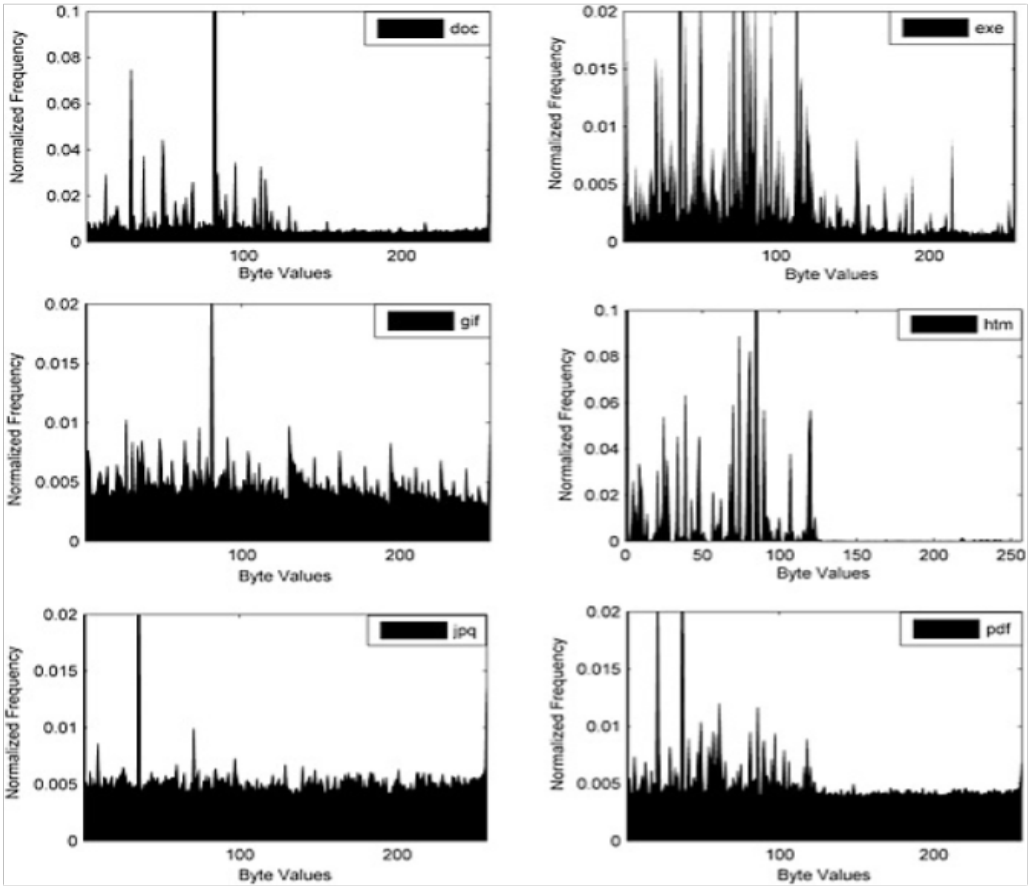

Figure 2.1: Binary distribution of some common file types. X axis: bytes (0 to 255), Y axis: normalised frequency of byte values (in %)[Li et al., 2005]. Some file types have unique binary distributions.

2. Unstructured Data

   There are many types of unstructured data that are stored in their native formats. In the case of file fragments, the native format of the data is a sequence of bytes. Unstructured data should be represented in numerical form. A number of methods can be used to numerically represent text (fragments bytes). The most common methods are bag-of-words, one_hot vector and a Word2Vec. Bag-of-words represents the text as a collection of words without regard to the order in which they appear in the text and only their frequency is retained. As a result, representing text as a bag is not sufficient when there is potential information in word order. The second most common method

is one_hot encoding where text is represented as binary vectors that map each text element to integer values. Then, each integer value is represented by a binary vector consisting of all zero values except the index of the integer, which is indicated by a one. However, there are two main disadvantages of this approach. Firstly, the storage of these data is inefficient and therefore a large amount of memory is required to process these data. Additionally, this approach does not capture the relationship between words, as it does not provide information about the context in which they occur. Lastly, Word2Vec, sometimes called word embedding, is a numerical representation of text that represents words with similar meanings by real-valued vectors. Typically, each word (or token) in a text is represented as a multidimensional vector. In the NLP case, the weight matrix is moved horizontally across the sentence one step at a time, capturing a specific number of words at a time. Each weight matrix is referred to as a filter, and each filter consists of an activation function, similar to that of a feed-forward neural network. In deep neural networks, the activation function, ReLU (Rectified Linear Unit), is mostly used due to its mathematical properties. According to the general logic of the filters, each filter is capable of detecting different features of text, and the deeper the filter the more likely it is to identify more complex features. As an example, a network's very first filters are likely to detect simple features, while its very last filters might be able to detect specific types of text. Each of these filters has a weight which is learnt by back-propagation during training, so generation of the filters is data-driven rather than hard-coded. Interestingly, Word2Vec embeddings allow for exploring interesting mathematical relationships among words. In this regard, this approach is an effective way to represent file fragments, where each byte in the fragment is represented by a real-valued vector in a high-dimensional space. Similar to words in the text, bytes that have a similar context in the vector space will also have a similar representation in their file content. The deep learning toolbox, Keras, offers an embedding layer that can be used for neural networks on text data that allows for a more expressive representation of fragments than more traditional methods, such as

bag-of-words, where the relationships between bytes (or tokens) are ignored, or bigram and trigram approaches, where the context is rigidly defined. The real-valued vector representation for words can be learnt during the neural network training procedure. The input data must be integer encoded, which means that every byte is replaced by a unique number ranging from 0 to 255. The embedding layer is initialised with random weights and will learn an embedding for all of the bytes (words) in the training set. Therefore, it can be used as a component of a deep learning model in which the embedding is also learnt together with the model itself. Embedding layers are primarily employed for their efficiency in converting sparse binary features (one_hot encoded words in the dictionary) into compact real-valued representations. As a result, we were able to reduce a single byte (256 binary features in 1-hot encoding) to between 16 and 64 real-valued features. The embedding layer is the first hidden layer within a network. Three arguments are required to generate the layer:

- input_dim: The size of the byte values in the fragment's data. The size of the bytes is 256, and their value ranges from 0-to-255.
- output_dim: Describes the size of the vector space in which bytes will be embedded. Defining the size of the output vectors from this layer for each byte. The size may be 32 or 100, for example.
- input_length: This corresponds to the length of any input sequence in a Keras model. In our data, the input dimension is 512 bytes, which is the size of the fragment.

In the next sections, we will introduce the different approaches and methods to classify file fragments.

2.3 Statistical Method For File Fragments Classification

In the statistical approach to classifying file fragments, each file type is represented as vectors to be compared. An unknown file fragment is classified based on distance to the nearest file vector type. Researchers create file vectors using a wide range of statistical features that have been extracted from file fragments. N-gram or Byte Frequencies Distribution (BFD) of file fragments is one of the most well-known features used for this approach. In addition, other statistical file features are derived from N-gram (or BFD), for example the mean and standard deviation of BFD, entropy, and so on. More details of these statistical features are presented in Chapter 3. The calculation of the distance between a file type vector and an unknown file fragment is obtained using a distance metric. The most commonly used distance metrics in the published literature for file fragment classification include the Manhattan distance and the Mahalanobis distance.

2.4 Machine Learning Approach For File Fragments Classification

Since 2006, the classification of file fragments has changed from threshold-based metrics and has shifted to supervised and unsupervised machine learning methods. Here, we discuss the most commonly used algorithms in file classification cases. Specifically, supervised algorithms include K-Nearest-Neighbour (KNN), Support Vector Machines (SVMs), Artificial Neural Networks (ANNs), random forest, Linear Discriminant Analysis/functions (LDA)and deep learning methods.

2.4.1 K-Nearest-Neighbour

The K-Nearest Neighbour (KNN) algorithm is a supervised machine learning algorithm that can be used for both classification and regression prediction. The KNN algorithm uses 'feature similarity' to predict values for new data points, which means that the new data

point is assigned a value based on how closely it matches the training set. Following are the steps in the KNN algorithm.

1. Choose the value of K, that is, the number of nearest data points. The value of K is any integer greater than 0.

2. Perform the following for each point in the test data:

   (a) The distance between each row of training data and the test data can be calculated using any of the following distance metrics: Euclidean, Manhattan, or Hamming distances. Euclidean distance is the most commonly used method to calculate the distance.

   (b) Sort the distances according to their values.

   (c) From the sorted array, select the top K rows.

   (d) The test data point will now be assigned a class based on the most frequent class in the top K rows.

Here is an example of how the KNN algorithm works. Let us consider the following dataset, which can be visualised as shown in Figure 2.2.

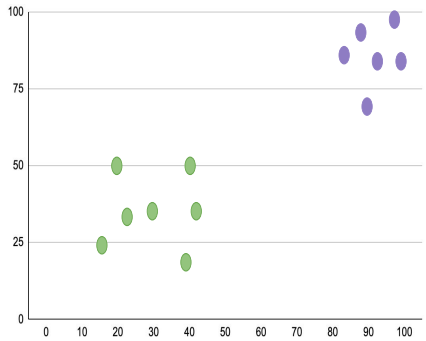

Figure 2.2: Visualised example of a two-class dataset, with data points from each class shown in green and purple respectively.

Figure 2.3: Shows a dataset that contains two distinct clusters of data. For visualisation purposes, one cluster is shown in purple and the other in green.

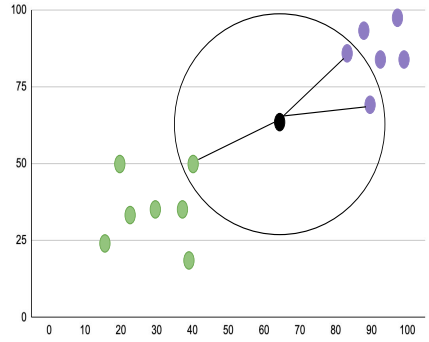

Figure 2.3: Two cluster dataset with the addition of a new test data-point show in black. A circle indicates the three nearest neighbours to the test data point. The majority of the nearest neighbours belong to the purple class, and therefore the test data-point would be assigned to that class.

Figure 2.3 shows the addition of a new test data point, shown as a black dot at point (60, 60). The task is to classify this data point as belonging to either the purple or red classes. Assuming K = 3, the three nearest data points to the test data point are found. As two of the three K-nearest data points belong to the red class, the black dot will be assigned to the red class.

2.4.2 Linear Discriminant Analysis (LDA)

In supervised classification problems, linear discriminant analysis, normal discriminant analysis, or discriminant function analysis is commonly used to reduce dimensionality. To explain it another way, these methods allow features in a higher-dimensional space to be projected into a lower-dimensional space.

For example, consider two classes that have multiple features and we need to separate them efficiently. Figure 2.4 shows that if only one of the features is considered, the classes are

not completely separated, which means that there is overlap in that feature space. Thus, we would like to add features so that the classes can be separated, and therefore classified more accurately.

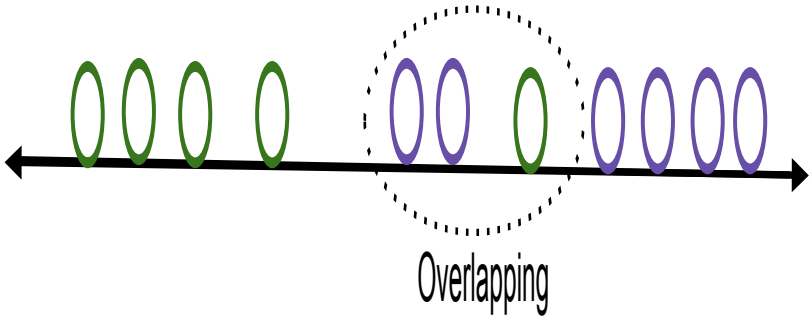

Figure 2.4: Example visualisation of a two class and one feature dataset, shown on a 1D axis. There is overlap between the classes, highlighted by a black circle.

For illustration consider two sets of data points that belong to two different classes that we would like to classify. Figure 2.5 illustrates that when the data points are plotted on a 2D plane, there is no straight line that can completely separate the two classes of the data points. To maximise the separability between the two classes, Linear Discriminant Analysis (LDA) is used, which reduces the 2D graph into a 1D graph. LDA is used here to create a new axis and project data into a new space in such a way as to maximise the separation between the two categories, thereby reducing the 2D space into a 1D space.

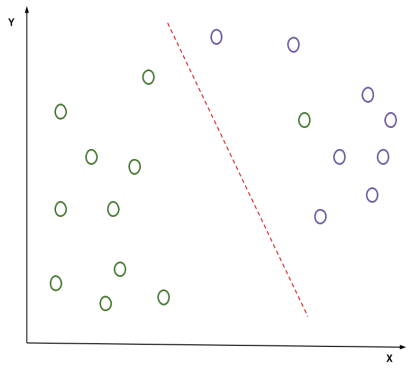

Figure 2.5: Example visualisation of a two class and two feature dataset, where there is no perfect separation between the classes. LDA is used to find a new axis (red dotted line) that maximises the separation between the classes and minimises the separation within the classes.

.

LDA creates a new axis based on two criteria:

1. To maximise the distance between the means of the two classes (e.g. increase inter-class variance).
2. Reduce the distance between the data points within each class as much as possible (e.g. decrease the intra-class variance).

As shown in Figure 2.6, a new axis (in red) is generated and plotted in the 2D graph to maximise the distance between the means of the two classes while minimising the variation within each class.

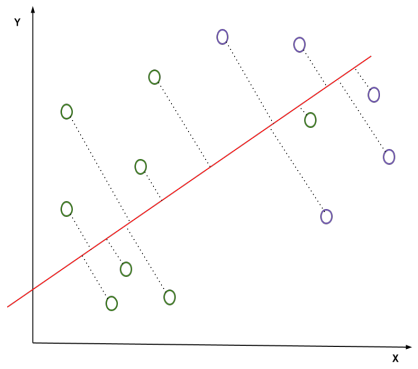

Figure 2.6: Visualisation of projecting 2D data onto a 1D axis using LDA.

To put it simply, this newly generated axis increases the separation between the data points of the two classes. Using the criteria mentioned above.

2.4.3 Support Vector Machine (SVM)

Support Vector Machines (SVMs) are machine learning algorithms that are used to solve two-group classification problems. SVM models are able to categorise new text after receiving sets of labelled training data. An SVM algorithm aims to find a hyperplane in an N-1 dimensional space (N is the number of features) that distinguishes data points by their characteristics.

For example, let us consider that we have two classes, green and purple, and our data has two features, x and y. The objective is to develop a classifier that, given a pair of (x, y) coordinates, outputs whether the data point belongs to the green or purple class. Figure 2.7 illustrates the labelled training data

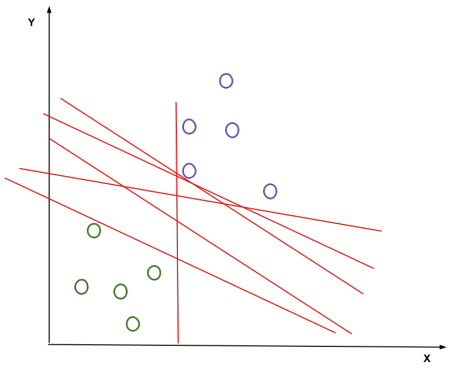

Figure 2.7: Illustration of how labelled training data can be separated using different lines.

A support vector machine uses these data points to calculate a hyperplane that best separates the data. The dimensionality of the hyperplane N-1, where N is the number of features. For example, for data containing two feature dimensions, the hyperplane would be a 1D line. Using the resulting hyperplane, which in the case of Figure 2.7 is a line, everything that falls on one side of the line will be classified as green, data that falls on the other side will be classified as purple. Figure2.7 shows that many hyperplanes can exist between two classes in 2-dimensional space. So, how does the SVM decide on the most effective hyperplane? For SVM, the optimal hyperplane is the one that maximises the margins between the two classes and the hyperplane. Figure 2.8 shows the hyperplane with the greatest distance from the nearest element of each class.

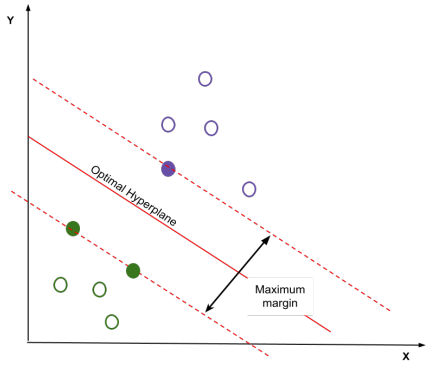

Figure 2.8: Illustration of a hyperplane with the greatest distance from the nearest element of each class.

2.4.4 Random Forest

One of the most popular tree-based supervised learning algorithms is random forest. A random forest combines hundreds of decision trees and then trains each tree on a different set of observations. It makes the final predictions by combining the predictions of each tree. There are several advantages to using random forests for classification tasks. Individual decision trees tend to overfit to the training data, but random forests can mitigate this problem by combining the different predictions made by separate decision trees. In this way, random forests can give a better prediction accuracy than a single decision tree Palczewska et al. [2014]. Figure 2.9 illustrates how the random forest algorithm works, by completing the following steps:

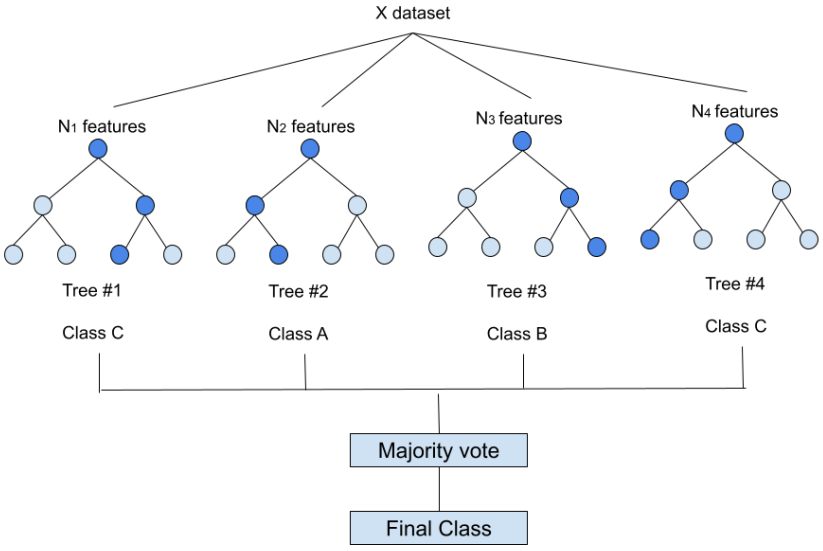

Figure 2.9: Random forest illustration. A decision tree is created for random features and random samples, and a majority vote is used to classify each test sample.

1. Random samples are selected from the x dataset.

2. The algorithm will generate a decision tree for each sample selected. Based on each decision tree created, it will generate a prediction result

3. For classification, a voting will then be conducted for each predicted outcome.

4. Finally, the algorithm will classify the sample as belonging to the class that has the most votes across all trees (the class with the majority of votes).

2.4.5 Neural Networks

Neural networks simulate the cognitive function of the brain [Gurney, 2018]. The human brain encodes knowledge by sending signals within a complex network of approximately 100 billion neurons [Bishop et al., 1995]. A neuron's ability to alter its response to inputs is effectively how they learn. The response of many individual neurons is combined to perform more complex tasks, such as image recognition or time-series prediction. Artificial neural networks are composed of nodes connected by network links, similar to the human brain [Russell, 2010]. Input signals x_1 to x_r are combined in a weighted sum and a bias b is added to determine the net-input z to a given node. See equation below.

$$z = \sum_{i=1}^{r} w_{ij} x_i + b$$

(2.1)

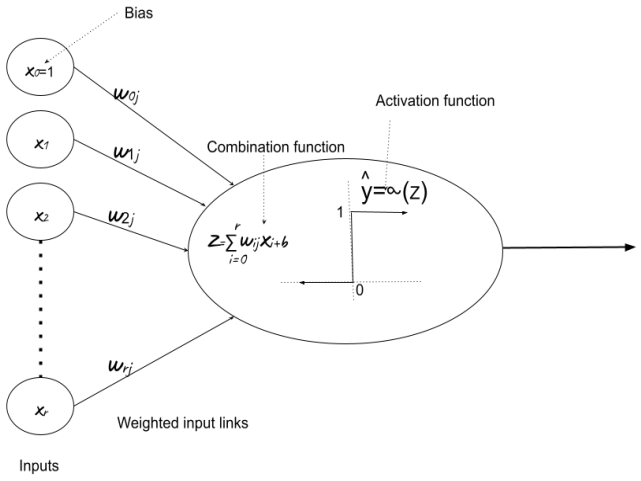

Figure 2.10: Illustration of a perceptron node in a Neural Network. Weighted node inputs are summed and then an activation function determines the output of the node.

A feed-forward neural network consists of multiple perceptrons interconnected in such a way that the output of one perceptron becomes the input for another [Bishop et al., 1995]. Signals can only pass in one direction in a feed-forward network. Multilayer Perceptron (MLP) neural networks are the most widely studied and used feed-forward neural networks [Gurney, 2018]. This type of neural network is composed of three or more layers: an input layer, one or more hidden layers, and an output layer. Each layer consists of a set of nodes, which are highly interconnected in a feed-forward manner:

A connection is made between every node in the input layer and every node in the first hidden layer, then this is repeated until every node in the second hidden layer is connected to every node in the output layer. As input signals propagate through the neural network towards the output layer, a vector of connection weights is maintained.

2.5 Deep Learning

2.5.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are used to learn to differentiate documents on classification problems using word embeddings as a method of representing words [Goldberg, 2016]. According to Goldberg [2016], CNNs are effective at document classification since they are able to pick up salient features (e.g. tokens or sequences of tokens) in a way that is independent of their position within input sequences Essentially, the network architecture consists of three parts:

1. Word embedding: When different words that have similar meanings (based on their usage) also have a similar representation.
2. Convolutional Model: Feature extraction model that uses word embeddings to learn to extract salient features from documents.
3. Fully-Connected Model: Interpreting the extracted features in terms of predicting the outcome.

According to Chen [2015], a single layer CNN can perform well with document classification, with different kernel sizes across different filters to allow grouping of word representations at different scales.

2.5.2 Transformer

In a recent study Vaswani et al. [2017] proposes a transformer as a novel feature extractor for Neural Machine Translation (NMT) tasks and finds that it performs better than RNNs in many cases. Tang et al. [2018] hypothesises that the reason the transformer has strong performance is that it has the ability to extract semantic features and to capture long-range dependencies from the data. For example, transformers can learn relationships between words that are far apart in a text, allowing them, in a way, to encode the context

and meaning of a word. By contrast, RNNs may encode only very short-term contextual information, usually one word before or one word after a given word. Confirming this, it has been demonstrated by means of experiments that transformers outperform RNNs in terms of word sense disambiguation. Transformer architecture follows an encoder-decoder structure, yet it does not rely on recurrence or convolutions in order to generate a convolutional output. Essentially, the encoder on the left half of the transformer architecture maps input sequences to continuous representations, which are compact and highly informative. Generally, encoders have 2 main layers that can be stacked as necessary:

1. In the first layer, there is a multi-head self-attention mechanism. The multiple-head mechanism has implemented h heads that each receives a (different) linearly projected version of the queries, keys, and values and that the heads are then used to generate h outputs in parallel, which are then used to generate a final result.

2. The second layer is a fully connected feed-forward network, which consists of two linear transformations with a Rectification Linear Unit (ReLU) in between $ffN(x) = ReLU(w_1x + b_1)w_2 + b_2$. Each of the layers of the transformer encoder applies the same linear transformations to all words in the input sequence. However, each layer utilises different weights (W1, W2) and bias parameters (b1, b2) when doing so. Additionally, each of these layers has a residual connection. There is also a normalisation layer, layernorm(x), which normalises the sum computed between the layer input, x, and the output generated by the layer itself, $layer(x) : layernorm(x + layer(x))$
   Considering that the transformer architecture does not make use of recurrence, it cannot capture any information about the relative positions of the words in the sequence. By injecting positional encodings into the input embeddings, this information can be added.
   Positional encoding vectors are generated using sine and cosine functions of different frequencies and have the same dimensions as the input embeddings. Then, they are

simply summed to inject the positional information into the embeddings.

2.6 Feature Selection

Feature selection also called variable selection or attribute selection, is the process of determining the most relevant attributes. Using feature selection techniques helps classification problems in 4 ways.

1. By reducing data dimensionality and model complexity.
2. By reducing model training time.
3. By helping to avoid overfitting during model training.
4. By providing insight to the researcher, aiding understanding of the data.

There are two main strategies for feature selection, filters and wrappers. In this research, we applied both filters and wrappers. These methods will be explained in the following sections.

2.6.1 Filters

The filter method selects features without relying on any learning algorithm. It selects the relevant features in isolation, without considering the effect on the classification accuracy of the learning model. The filter works by searching for and finding the most related features. The process starts by choosing the searching method to determine the direction of the search. Then it assigns a relevance score for each feature using statistically based measures. The higher the score, the more relevant the feature is. Figure 2.11 illustrates how the filter process works.

The filtering methods are divided by the statistical measures they use. The statistical measures must be carefully chosen based on the data type of the input variable and the output, or response variable. Broadly speaking, there are four main measures:

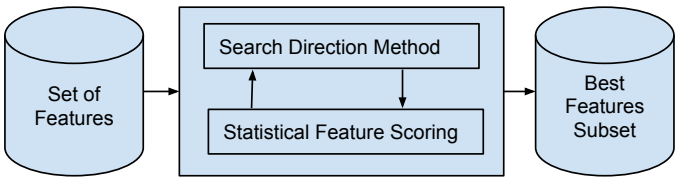

Figure 2.11: Filter process illustration. Feature selection through related feature search.

1. Distance Measures: This method assigns weights to each feature individually and then ranks them based on their relevance value. The features that have a weighted score higher than the specific threshold value are considered good features. The most common algorithm is Relief, which was proposed by Kira et al. [1992]. Relief scores features by calculating the distances between all combinations of feature pairs. Kononenko et al. [1997] proposed the ReliefF algorithm for application to multi-classes problems. However, all Relief family algorithms have a significant limitation; they cannot capture redundancy among features, they only capture the relevancy of features to the target. Therefore, for high-dimensional data where many redundant features may exist, relief algorithms are not the best choice for big data feature selection [Urbanowicz et al., 2018].

2. Information Measures: This measure quantifies the information that can be gained from each feature. For example, the information gained from features f_1 is determined by splitting the dataset according to the given classes and calculating the reduction in entropy. Feature f_1 is preferred over feature f_2 if the information gained from f_1 is greater than that from f_2 [Liu and Yu, 2005]. Equation 2.2 shows how information

gain is calculated for each feature.

$$InfoGain(Class, Attribute) = H(Class) - H(Class|Attribute) \tag{2.2}$$

There are more statistical measures of information, such as ANOVA, F-test, and mutual information. Choosing the most appropriate measure is based on your understanding of the data and the problem you are addressing.

3. Dependency Measures: The correlation-based measure estimates the ability to predict the value of one feature from another. Good features are those highly associated with the classes [Liu and Yu, 2005]. Feature correlation can be calculated by Pearson's correlation, Spearman's rank coefficient, Kindall's rank coefficient, etc.

4. Consistency Measures: The inconsistency measure aims to remove features whose values are similar, irrespective of the class under consideration. For example, a feature, f_1, which holds the value 5 for both classes A and B, provides little information to discriminate the classes. Focus and Las Vegas are examples of this filter method. Both are designed to do an exhaustive search of the feature space until they find the minimum combination of features that divides the training data into classes. Again, this filter method is not ideal for processing datasets with high dimensionality [Liu and Yu, 2005].

2.6.2 Wrapper

While the filter methodologies treat data features independently from the learning algorithm, wrapper methodologies rely on selecting the best subset of features with the guidance of the learning algorithm. Figure 2.12 illustrates the general idea of how the wrapper works. The wrapper algorithm usually starts with a given subset of features F_0, as it can be an empty set, a full set, or any subset of randomly selected features. Then the algorithm searches through the feature space using a particular search strategy: sequential,

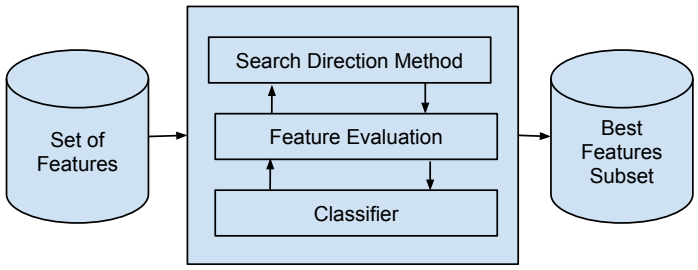

Figure 2.12: Wrapper process illustration. Guided feature subset selection.

exponential, or randomised search. For each subset of generated features F_i, the algorithm evaluates it using a learning model. F_i is modified by adding or removing features based on the performance of the learning model [Kabir et al., 2010]. Wrappers differ depending on the search method used, for example, sequential search methods where the search scheme adds or removes features sequentially. Eliminating or adding features can be done in a forward or backward direction [Devijver and Kittler, 1982].

- Forward selection is the simplest algorithm. It starts with an empty set and adds one feature (or set of features) at a time until all features are considered. The process of adding features depends on the performance achieved by the base learner.
- Backward elimination: Backward elimination works exactly inverse to forward selection. It starts with a complete set, drops a feature (or set of features) and observes the performance of the base learner.

2.6.3 Hybrid Feature Selection

In theory, the best subset of features can be found by employing an exhaustive search and evaluation function, but this method has an exponential computing complexity ($2n, n = numberoffeatures$) and is unrealistic when the number of features is large. Therefore,

the best way to use the wrapper technique with a huge number of features is by dividing the selection process into two steps, which are called the hybrid feature selection method. hybrid feature selection method utilised both filters and wrappers in sequence [Singh and Silakari, 2009].

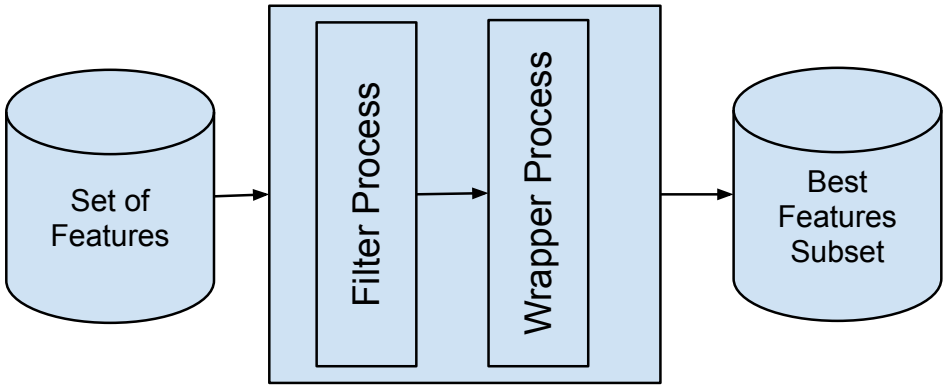

Figure 2.13: Hybrid feature selection process illustration. Sequential integration of filters and wrappers.

2.7 Summary

This chapter presented a general understanding of the statistical and machine learning approaches applicable to the file fragment classification problem. We demonstrated how NLP techniques can be applied to file fragment classification and how they can be used to treat file fragment data in a structured or unstructured form. We also discussed how statistical methods can be used to solve the file fragment classification problem. In addition, we described the most common machine learning algorithms used to solve file fragment classification problems, which included the KNN, SVM, LDA, NN, CNN, transformer, and random forest algorithms. Moreover, we discussed feature selection techniques for classification problems such as ours and why they are important in this field. The next chapter

presents a literature review, providing details on previous work in the area of file fragment classification.

3 Literature Review

3.1 Introduction

This chapter reviews earlier work on file fragment classification. Section 3.2 studies the use of statistical analysis methods to solve file fragment classification problems. Section 3.3 explains previous work that used machine learning methods to solve the file fragment classification problem, including supervised and unsupervised techniques and deep learning methods. Finally, a summary of this chapter is presented in Section 3.4.

3.2 Statistical Approach For File Fragments Classification

Schultz et al. [2001] were the first to suggest an N-gram analysis of the file binary content to detect malicious virus code. He created a UNIX mail filter that detected malicious Windows executables. BFD N-gram is used as the set of features to represent each executable code.

McDaniel and Heydari [2003] introduced the idea of presenting files as vectors to be compared. They suggested fingerprints for file type identification and proposed three algorithms. They believed that the different file types have different fingerprints, which contain typical features that are specific to their type. In their research, the file fingerprint calculation is derived from the byte frequency distribution of the whole file. The authors developed three different algorithms to address the file type classification problem. Their first algorithm was Byte Frequency Analysis (BFA). In this approach, they generated the frequency distribution of all byte values (0-255) of a file's binary content, and before averaging the BFD of all sample files, they normalised these values. This average of distributions is called the file

fingerprint for a certain file type, and then unknown files are recognised by calculating their BFD and comparing them to all fingerprints that have been previously created. The BFA algorithm of McDaniel and Haidari suffered from low accuracy (27.50%), so they extended the BFA algorithm and incorporated the byte order into the file fingerprint which led to the development of the second algorithm, Byte Frequency Cross-Correlation (BFC). In the BFC algorithm, they used two-dimensional arrays to record the co-occurrence of each byte pair in a file. The difference between the same byte in different files is summed and the cross-correlation between all byte pairs is computed. The BFC algorithm works well for certain file types, such as HTML, where some ASCII values of the file type have higher occurrences, such as "<>". This improved the accuracy to 45.83%. In the authors' last algorithm, the File Header/Trailer (FHT), they used only the file header and trailer as inputs for their calculation. This increased the accuracy to over 95%. However, other researchers [Garfinkel et al., 2010] believe that the McDaniel and Haidari algorithms are not suitable for file fragment classification, where such highly informative signatures are not usually present.

Li et al. [2005] followed the approach used by McDaniel and Haidari and introduced file fingerprints using a unigram analysis of block contents. They built three different models: single centroid (one model for each file type), multi centroid (multi-model for each file type), and exemplar files. The multi centroid model is used instead of the single centroid (average) that was used by Haidari. In the exemplar files model, they ran several tests using the n-byte of the file (where n is the first 20, 200, 500 or 1000 bytes), as well as for the whole file. The authors used the mean and standard deviation of the byte frequency to represent the centroid of the file types and used the Mahalanobis distance in the file model comparison. In the exemplar file model, they used Manhattan distances to compare the BFD of the exemplar file with the given file. Their solution achieved an accuracy of almost 100% when they only used the first 20 bytes. In other words, they used the file signature information, which is usually located in the first few bytes, to recognise the file type for the classification of whole files. It was not intended to be used for fragments that originated

from the middle of files and would therefore not be helpful for the task of file fragment classification, where there is no signature or metadata present.

3.2.1 N-gram And Other Statistical Measures

In order to aid in the processing of file fragments, some researchers add more statistical measures. This section will present the previous work done on file fragment classification using N-gram analysis and other statistical measures driven from it.

Karresand and Shahmehri [2006b] introduced a method for file fragment classification which they called Oscar. They used a centroid-byte frequency similar to Li's work. However, in their case, the centroid is presented by two vectors of length 256, which are the mean and standard deviation of byte value frequency distribution. They used a 4KB fragment size. Their approach has been optimised for JPEG files, where specific markers are explicitly used to improve the accuracy. The Oscar method can recognise JPEG fragments with a 97.90% accuracy level, but this exceptional result is limited to JPEG fragments, and it could not be achieved with other file types because they lack the characteristic structural information present in JPEG files. Also, they did not exclude metadata in their experiments. The authors [Karresand and Shahmehri, 2006a] improved the Oscar method and considered a new metric which is the Rate of Change (RoC), where the difference between two consecutive byte values is found. In this way, the byte order is taken into consideration. Karresand and Shahmehris's approach is mainly suited to the identification of JPEG fragments. The JPEG file format contains "0xFF 0x00" as an initial mark for all metadata tags. This byte-pair will increase the RoC of almost any JPEG file and provides special identification characteristics for them. The other file types would not have this special feature, which explains why they obtain a poor detection accuracy of less than 12% for other file types.

In Erbacher and Mulholland [2007], the researchers used a sliding window algorithm that covered powers of 2 from 64 bytes to 16K bytes. Thirteen statistics were chosen to determine the different characteristics of the file types and display them graphically to visually identify

seven file types, including DOC, EXE, JPEG, PDF, PPT, XLS, and ZIP. In analysing the thirteen statistical features for the file and data types, the average, kurtosis, distribution of averages, standard deviation, and distribution of standard deviations proved to be sufficient to effectively distinguish the different types of data. The additional statistics did not add value to the analysis. Among all the data types they examined, JPEG files were the most organised. Still, they only performed small tests consisting of five files for each type.

Hall et al. [2006] utilised the entropy and compressibility metrics for the file classification problem. They measured data entropy and compressibility of 90-byte windows of the file. Then they plotted the average and standard deviation of entropy and compressibility for each file type. The unknown file type was determined using the point-by-point delta, Pearson's rank-order correlation measurement, and some other measures for curve fitting to find the similarity. No average detection rate was offered. However, the technique fails to identify file types accurately but will help in differentiating the type of data contained within the file, such as compressed versus uncompressed data. Only 12% of compressed fragments were correctly identified and no other results were provided.

Moody and Erbacher [2008] extended the work of Karresand and Shahmehri [2006a] by considering the five most beneficial statistical features stated by Erbacher and Mulholland. Moody and Erbacher did their study on 200 files of 8 different types. They reported that their method was only able to differentiate simple file types. For example, it could distinguish the difference between textual files, such as CSV and HTML. They achieved a detection rate of 74.2%, including high detection rates for text-type files, that is, CSV (100%), HTML (100%), and TXT (80%).

Penrose et al. [2013] focused on the problem of distinguishing compressed from encrypted fragments. 9 of the 15 tests in the NIST statistical test suite [Rukhin et al., 2001] were used to construct a classifier. As a rule, encrypted data fragments (encrypted using AES) should

be indistinguishable from random data, while compressed data using standard methods (such as lossless compression methods) should exhibit a bias. The researchers used three compression tools (in order of increasing compression ability): ZIP, BZ2, and 7ZIP; and fragment sizes of 4, 8, and 16KB. For 16KB deflate-compressed chunks (using ZIP), the accuracy of the method reached 92%. For 16KB BZ2-compressed chunks it reached 72%. For 7ZIP-compressed chunks, the best accuracy was 12%, for 4KB chunks. The interpretation of the results is that as compression quality increases and redundant elements of the encoding are eliminated, the output stream can no longer be distinguished from random or encrypted data. Consequently, they state that any classification method that relies on statistical measures will fail and only specialised techniques will provide reliable results.

3.3 Machine Learning Approach For File Fragments Classification

Since 2006, the classification of file fragments has changed from threshold-based metrics and has shifted to supervised and unsupervised machine learning methods. We review the most commonly used algorithms in file classification cases. Specifically, supervised algorithms include Support Vector Machines (SVMs), artificial neural networks, random forest, genetic programming, and linear discriminant analysis/functions.

3.3.1 K-Nearest-Neighbour

A selection of previous work has focused on using KNN as the method for file fragment classification. Now, we will discuss these studies in turn. Axelsson [2010a], Axelsson [2010b] and Axelsson et al. [2013] presents a method to classify file fragments, using the Normalised Compression Distance (NCD) algorithm combined with the K-Nearest-Neighbour (KNN)algorithm. By using the compression of two fragments and their concatenations, he measures the distance between a pair of fragments and finds the comparison ratio and

repeatability. The fragment is classified by computing its distance from sample fragments representing different types of files and associating those fragments with the type of file represented in the closest sample fragment. In [Axelsson, 2010a], Axelsson used a dataset consisting of 150 files of 17 file types. Using a fragment size of 512 bytes, he got an average accuracy of around 50%. In [Axelsson, 2010b], Alexsson used the freely available corpus of forensic research data by Garfinkel and selected 28 file types in the experiments. Using a fragment size of 512 bytes, he achieved accuracies between 32% and 36% when the value of k ranged from 1 to 10. Although Axelsson's results were not very impressive, his results revealed some interesting patterns in the misclassifications. He found that large numbers of files were misclassified as docx files. The reason for this might lie in the fact that docx files are container files, with specially structured contents.

In [Axelsson et al., 2013], Alexsson used a dataset comprising 50 files of 28 file types, with 4 fragment sizes of 512, 1024, 1536 and 2048 bytes. According to Alexsson, the use of different block sizes does not have any significant effect on the accuracy of file fragment analysis by the NCD algorithm. Additionally, choosing a compressor based on its compression ratio or repeatability has no significant impact on performance.

Conti et al. [2010] considered a novel way to classify file fragments by seeking to recognise the primitive data type for the file object, rather than classifying the whole file or fragments. They employed 14000 fragments, each 1024 bytes in size, and from 14 different file types. They constructed fragment signatures using four statistical measurements: Shannon entropy, Hamming weight, chi-square, and mean value of bytes. They compared these signature vectors to unknown file fragments using KNN, with the Euclidean distance as the distance metric. They achieved 98.55% classification accuracy for random/ compressed/encrypted fragments, 100% for Base64 encoded fragments, 100% for unencoded fragments, 96.7% for machine code (ELF and PE) fragments, 98.7% for text fragments, and 82.5% for Bitmap fragments. However, the classifier did not perform as well when applied

to real-world binary data, especially when it contained fragments of a previously unstudied primitive type, even one with a closely related structure. They also classified fragments according to types at a very coarse granularity.

3.3.2 Linear Discriminant Analysis (LDA)

Linear Discriminant Analysis (LDA) has been used as a classification method for the file fragment classification problem. The following is a summary of previous work in this area. Veenman [2007] proposed using BFDs, entropy and the Kolmogorov complexity features for LDA algorithms, to classify disk clusters. He included 11 different file types in his study and the size of the fragments he used was 4069 bytes. His most accurate results were achieved with HTML and JPEG, which gave accuracy levels of 99% and 98% respectively. Lower accuracy levels were produced with ZIP (18%) and executable files (78%). Thus, the average accuracy he obtained was 45%. An advantage of his work is that he excluded file metadata from his experiments and his results detail how some files are commonly confused with certain other file types. Such file types would require further work to differentiate them to a good degree.

Ahmed et al. [2010] further developed the methods of Ahmed et al. [2009] but also introduced some new methods. In creating their multi-centroid model, they clustered files with similar BFD regardless of the type of data. Based on these assumptions, different file types may have similar byte frequency distributions; however, files of the same type may have different byte frequency distributions. LDA was used to create a discriminant function for each fragment type within each cluster. The cosine similarity metric was found to be more accurate than the Mahalanobis distance. An unknown fragment was first assigned to a cluster using cosine similarity. A fragment would be classified as the same type if all files in a cluster were that type. If not, then LDA was applied to find the closest type match in the cluster. A total of ten different types of files were used, but compressed and encrypted files were excluded. In the training and test sets, 100 files of each type were included. The

header information was included in whole files rather than file fragments. An accuracy of 77% was obtained.

3.3.3 Support Vector Machine (SVM)

Following is a summary of the studies that have used Support Vector Machines (SVMs) in the file fragment classification problem. Li et al. [2006] utilised SVMs for file classification. Their method relies on using the file header bytes for the process of identifying the files, five types of which were included. They reported accuracy levels ranging between 61% and 100%. Although Li et al. [2006] inspired subsequent researchers to use a machine learning algorithm for file classification problems, their model could not be generalised to work with file fragment classification, because their method made use of file header bytes.

Li et al. [2011] claimed that the classification of file fragments of many file types can achieve high accuracy, but that the classification of high-entropy file fragments is challenging and had, up to that point in time, produced poor detection accuracy. Therefore, in their research, they highlighted the recognition of high entropy file fragments. Li et al. proposed using an SVM to classify high entropy fragments, based on BFD. Using private datasets, their study involved only four file types (JPEG, MP3, DLL and PDF). Each file type had 880 files, and each file was divided into fragments which were 4096 bytes, where the first and last fragments were eliminated to ensure that metadata were excluded. Although they reported an 81% overall accuracy level, they did not take into consideration other high entropy file types, such as compressed files. Moreover, they used large fragment sizes, which may have made the classification task easier.

Sportiello and Zanero [2011] sought to build models for nine different file types, using SVMs. They used 27 statistical features in their file type modelling process, including BFD, entropy and Lempel-Ziv complexity. In their experiment, they divided files into fragments which were 512 bytes in size. However, their report did not mention whether they excluded file

headers and footers. Although they reported a high accuracy rate, ranging from 71.1% to 98.1%, they did not attempt to do any multi-class classification.

Sportiello and Zanero [2012] also investigated the application of SVMs to identify the file types of file fragments for eight file types. In their study, the authors discussed the impact of certain features associated with the block classification process as a whole. Their classification framework was based on a one-to-many approach, with a classifier trained for each file type. They explored features including the BFD, the frequencies of the differences between two consecutive bytes, Word Frequency Distribution (WFD), the mean byte value, entropy, and Lempel-Ziv complexity. A combination of BFD, Information Entropy and complexity gave the best results across the majority of file types, ranging from 87% for EXE files to 96.9% for MP3 files.

Fitzgerald et al. [2012] considered 9000 fragments of 512 bytes, from 24 file types taken from govdoc. They utilised an SVM to classify these fragments, based on unigram and bigram byte features. They reported an overall accuracy of 47.5%. They reported that the accuracy for compressed file prediction averaged 21.8%. This could be explained by the fact that the classification of high entropy files is a more challenging task due to the lack of repeating patterns resulting from the compression process.

Roussev and Quates [2013] claimed that no approach can effectively classify data fragments of high-entropy file types. They developed a deflate analyser to automatically discover deflate coded data and to generate a number of useful statistics. Microsoft Office files are used for their analysis, and also DLL, EXE, MP3 and PNG. They also quantify the relationship between fragment size and the expected optimal classification accuracy and conclude that larger block sizes lead to more accurate classification accuracy.

Beebe et al. [2013] developed the sceadan algorithm. They combined unigram (BFD) and bigram, as Fitzgerald did, to use as input to the SVM classifier. Additionally, they used 11 statistical measurements as fragment features. Their experiments included 38 different file types, which produced 10,000 fragments of 512 bytes. A 73% detection rate was reported.

In other work, Beebe et al. [2016] employed hierarchical class-to-type modelling. Working with 52 file types, they decided on a hierarchy of two layers, initially separating the file types into six classes and training classifiers for each class. The authors again used the four feature sets adopted in their previous work. K-means clustering was used, and a classification accuracy of 74.1% was achieved.

Nguyen et al. [2014] studied the impact of data fragment size on the identification of file types. This paper examines the results of applying machine learning techniques to identify types of files in data fragments of different sizes, with the aim of finding the minimum size needed for file type recognition. They use SVM with BFD as a feature to recognise the type of data fragment. For the SVM, linear kernels and multiclass (one-against-all) were selected. In terms of file types, they split files into different fragment sizes, ranging from 50 bytes to 512 bytes, from the 165 MB dataset. The total number of 512-byte fragments was 338889. 1000 fragments were randomly selected for each file type from the 264000 fragments dataset to build classification models. Thus, the total number of data fragments for 22 file types was 22000 each time. Initially, they performed experiments on 512 bytes, which is a common size for work in this field. Using fragment sizes greater than or equal to 200 bytes, the file type recognition accuracy is greater than 60% and does not change significantly with increased fragment sizes. For fragment sizes smaller than 200 bytes, the accuracy is below 60% and decreases dramatically as the fragment size is reduced. Thus, file type recognition cannot be accurate if the data fragment is smaller than 200 bytes.

Bhatt et al. [2020] also used hierarchy-based file fragment classification, examining fourteen file types, with 1000 fragments from each. Classification was performed using an SVM. Ten features extracted included bigram distribution, mean byte value, contiguity, entropy, hamming weight and Kolmogorov complexity. Information gain, a feature ranking metric, was also used. They achieved a mean classification accuracy of 67.78%. Simple file classification (e.g. TXT, LOG and HTML) could be achieved with relative ease, although complex files (e.g. ZIP, PPT and PDF) proved more challenging.

3.3.4 Neural Networks and Deep Learning

The following studies used feedforward neural networks for file fragment classification. Harris [2007] utilised a neural network to identify file types. He split files into 512-byte blocks, and only the first 10 bytes of each file were analysed. The byte value and the BDF were then used as input to a neural network. Harris focused exclusively on image files and included five different image file types in his work (BMP, GIF, JPEG, PNG and TIFF). They reported detection rates ranging from 1% (for GIF files) to 50% (TIFF) with raw filtering, and from 0% (GIF) to 60% (TIFF) without filtering. However, Harris's results were no better than those existing techniques which used file signature for file classification problems. Their neural network architecture was a simple, shallow network that contained a single layer of neurons.

Amirani et al. [2008] employed a neural network to classify six different file types (the DOC, PDF, EXE, JPEG, HTML, and GIF). They used BFD as an initial feature. They then applied Principal Component Analysis (PCA) and the auto-associated neural network to the 256 features, which came from BFD, to generate a dataset with a reduced number of features. Subsequently, they trained a three-layer Multilayer Perceptron (MLP) for file type identification. They achieved an excellent result, with a 98.33% detection rate. Despite the good results these researchers obtained, their method cannot be generalised to work with fragmented files, as they considered the whole file, including the metadata.

Calhoun and Coles [2008] extended Veenman's research by building a classification model, using byte frequency, entropy, and 20 more statistical features, and using LDA to recognise file types. Additionally, they considered common substring and sequence statistical approaches to determine file types, by finding the longest substring that is common between blocks of the same file type. They validated their work using two tests. In the first test, they considered fragments that were 896 bytes in size, from files where the first 128 bytes had been chunked and removed. The second test involved considering fragments that were 512 bytes in size from files where the first 512 bytes had been chunked and removed. Calhoun

and Coles classified type files only on a pairwise basis. This may explain the good results they achieved, which showed an average of 88.3% accuracy. As a result, their technique could not easily be generalised to real-world fragment classification problems.

Ahmed et al. [2010] used a neural network to recognise file types. Their approach contained three main phases. First, they used cosine similarity in the compression of BFD file content. The second phase involved the use of a divide-and-conquer technique, in which the BDF of similar fragments were grouped regardless of their types. In the third phase, an MLP neural network is used to classify file types. Irfan Ahmed et al evaluated their approach using 2000 files, consisting of 10 file types and reported a detection rate of 90.19%. However, once again, their approach was not proven to work with fragmented files, because they considered the whole file, including information about the file type.

Wang et al. [2018] used Convolutional Neural Networks (CNNs) to classify file fragments. Essentially, they train one layer of CNNs on top of an embedded layer. Feature-based algorithms such as SVM, KNN, and XGboost were compared to CNNs for classification accuracy. They applied these algorithms to 7 different fragment sizes which are 64 bytes, 128 bytes, 256 bytes, 251 bytes, 1KB, 2KB and 4KB. For fragments extracted from the middle of files, the accuracy generally improved by increasing the size of the fragments. They also found that CNNs outperformed the other methods, for example, for 512 fragments the accuracies achieved by CNN, Xgboost, SVM and KNN were 67%, 65%, 64% and 60%, respectively.

Vulinović et al. [2019] explored the use of Feedforward Neural Networks (FFNNs) and CNNs for the classification of file fragments. They studied 19 file types. The FFNNs were trained using byte histograms and byte pair histograms, whereas the CNN were trained on blocks containing 512 bytes of data taken from the GovDocs1 dataset. The average accuracy with CNN is 54% and for FFNN 82%.

Mittal et al. [2020] used CNNs for the large-scale classification of file fragments. They use one-dimensional CNN architecture, where it is built on embedding layers and fellow by

dense layer. They created new datasets of 75 file types and achieved an average accuracy of 66.5% for 512 fragment size and 77.5 for 4096 fragment size.

3.3.5 Mixed Methods For File Fragments Classification

Ahmed [2010] compared the detection rate of file types for a number of commonly used classifiers (neural network, LDA, K-means, KNN, decision tree, SVM). They used the BFD of the initial byte sequence of each file as an input feature. They stated that for high-entropy file fragments, SVM is probably the most appropriate classifier to obtain the best accuracy, while neural networks and KNN perform best for low-entropy file fragments. Content sampling techniques were used to reduce classification time.

Kattan et al. [2010] tested a novel approach based on genetic programming. From the Internet, 120 examples of each of the five file types (JPEG, GIF, TXT, PDF, EXE) were downloaded at random. Compressed or encrypted files were not included. File headers may have been included because the analysis was done on whole files rather than fragments. The BFD was first extracted using PCA and then processed by a multi-layer auto-associative neural network to produce file fingerprints. By reducing correlated features into a set of uncorrelated 'principal components', PCA removes redundancy from the data. The file fingerprint is created by a multi-layer auto-associative neural network. To classify unknown file types, a 3-layer network was used with these file fingerprints. Only 30 files of each type were tested. There were 98% true positives reported in a confusion matrix. Including the file, headers may be the reason for the high detection rate.

Gopal et al. [2011] compared the performance of the SVM using N-gram features, and K-nearest-neighbours, with several commercial off-the-shelf tools, such as Libmagic, Trid, Outsidein and Droid. They explored file classification under four different file situations of damage and segmentation: First is the ideal case where there is no damage and the files are complete, without missing bytes. The second case is where the signature bytes are removed and no additional bytes are missing. The third case is where the signature

bytes are removed and additional bytes are missing at random locations. The fourth and final case is where the files are fragmented to specific size. The authors considered 36000 files of 316 file types, taken from a publicly available data corpus. They reported that the classification of file fragments is much more difficult than the classification of complete files. Therefore, using SVM with bigram, the accuracy they achieved on 512-byte fragments was about 33%. With 4096 byte fragments, the accuracy was 40%.

Ahmed et al. [2011] extended their work in [Ahmed, 2010] by including bigrams as features for six different classifiers: LDA, K-means, KNN, decision tree, SVM, and neural network. Randomly, 500 files of each of the 10 file types were collected from the Internet. They concluded that KNN is the most accurate and least time-consuming algorithm for file fragment classification.

Amirani et al. [2013] extended the work of Kattan et al. [2010] to detect file fragments. The original version used an ANN as the final stage classifier and an SVM as the intermediate stage classifier. Randomly, 200 files of each of the six file types were collected from the Internet, which are DOC, EXE, GIF, HTML, JPEG and PDF. Half of the files were used for training and the other half for testing. Each file was fragmented at a random starting point and a fragment of 1000 bytes or 1500 bytes was taken for analysis. They found that the SVM classifier gave better results than the ANN for file fragments of both 1000 and 1500 bytes. Detection of PDF files produced 89% true positives with fragments of 1500 bytes. Possibly, this indicates that the PCA has been unduly influenced by the file header.

The literature in this chapter has been summarised in the following three tables.

Table 3.1 summarises the statistical features that have been used by other researchers in the previous literature. The first column is the feature name. The second column is the feature definition and calculation equation, if needed. The last column is a reference to the studies that use each statistical feature in their work.

Table 3.1: Definitions and usage of the features.

Feature Name	**The Description**	**Reference**
unigram	sometimes called a Byte Frequency Distribution (BFD). Each byte in a file is represented as a numerical value from 0x00 to 0xFF. The BFD extraction in our study counted the occurrence of each byte value in the fragment, producing a vector of 256 features for each instance.	Schultz et al. [2001] McDaniel and Heydari [2003] Li et al. [2005] Karresand and Shahmehri [2006b] Hall et al. [2006] Al-Sadi et al. [2013] Karresand and Shahmehri [2006a] Calhoun and Coles [2008] Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020] Moody and Erbacher [2008]
bigram	A bigram vector can be extracted for each instance. A bigram is similar to a unigram, but it takes the sequence of each two consecutive bytes into consideration. Hence, a word two bytes in length is represented as a numerical value ranging from 0x0000-0xFFFF. To find the bigram vector, we counted the occurrences of each word value (two bytes) in each instance, resulting in a 65,536 dimensional vector.	Fitzgerald et al. [2012] Beebe et al. [2013]

Continued on next page

Table 3.1: Definitions and usage of the features. (Continued)

Feature Name	**The Description**	**Reference**
ROC	The rate of change is the difference between two consecutive byte values. By this measure, the byte order is taken into consideration.	Karresand and Shahmehri [2006b] Calhoun and Coles [2008]
LAFA	Low ASCII Frequency Average is the mean of byte frequencies with values ranging between 0x00-0x1F in the fragment.	Calhoun and Coles [2008] Beebe et al. [2013]
MAFA	The Medium ASCII Frequency Average is the mean of byte frequencies with values ranging between 0x20-0x7F in the fragment.	Calhoun and Coles [2008] Beebe et al. [2013]
HAFA	The High ASCII Frequency Average is the mean of byte frequencies with values ranging between 0x80-0xFF in the fragment.	Calhoun and Coles [2008] Beebe et al. [2013]

Continued on next page

Table 3.1: Definitions and usage of the features. (Continued)

Feature Name	**The Description**	**Reference**
Unigram Entropy	In information theory, entropy measures the randomness of data; the more random a set of bytes, the higher the entropy. For this research, we used Shannon's classic formula to find the entropy of unigram byte values. $H(x_i) = -\sum_{j=1}^{n_i} P(x_{ij}) log_b P(x_{ij})$ where x_i is each possible data value and $P(x_{ij})$ is the probability mass function for the value.	Hall et al. [2006] Calhoun and Coles [2008] Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020]
Bigram Entropy	Similar to unigram entropy, except it measures the randomness of the two byte values.	Beebe et al. [2013] Bhatt et al. [2020]
Fragment Mean	Is defined as the average of all byte values. In a file fragment, the mean of fragment bytes value is the sum of all bytes divided by the fragment size. For this research, we used the equation: $x_i = \frac{1}{n_i} \sum_{j=1}^{n_i} x_{ij}$	Karresand and Shahmehri [2006b] Al-Sadi et al. [2013] Moody and Erbacher [2008] Calhoun and Coles [2008] Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020]

Continued on next page

Table 3.1: Definitions and usage of the features. (Continued)

Feature Name	**The Description**	**Reference**
Standard Deviation	The standard deviation of the byte values is the standard deviation of bytes in the fragments. $S_i = \sqrt{\frac{1}{n_i-1}\sum_{j=1}^{n_i}(x_{ij}-\bar{x_i})^2}$	Karresand and Shahmehri [2006b] Alsadi et al. [2013] Moody and Erbacher [2008] Calhoun and Coles [2008] Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020]
Standard Kurtosis	It measures whether data is heavily or lightly tailed when compared with data which has a normal distribution. This determines the extent of peaks in the byte values. $K_i = \frac{1}{n_i-1}\frac{\sum_{j=1}^{n_i}(x_{ij}-\bar{x_i})^4}{s_i^4}$	Karresand and Shahmehri [2006b] Beebe et al. [2013]
Standard Skewness	The measurement of asymmetry of the byte value distribution graph when related to the mean. $G_i = \frac{1}{n_i-1}\frac{\sum_{j=1}^{n_i}(x_{ij}-\bar{x_i})^3}{s_i^3}$	Beebe et al. [2013]
Average Contiguity	The difference between the values of each two consecutive bytes. $C_i = n_i\frac{\sum_{j=1}^{n_i}(x_{ij}-\bar{x_i})^4}{(\sum_{j=1}^{n_i}(x_{ij}-\bar{x_i})^2)^2}$	Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020]
Maximum Byte Streak	The length of the longest streak of repeating bytes in the block.	Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020]

Continued on next page

Table 3.1: Definitions and usage of the features. (Continued)

Feature Name	**The Description**	**Reference**
Hamming Weight	Total number of set bits divided by total number of bits in the block.	Calhoun and Coles [2008] Fitzgerald et al. [2012] Beebe et al. [2013] Bhatt et al. [2020]
Mean Absolute Deviation	The statistical measure of dispersion at byte level. $MAD_i = \frac{1}{n_i}\sum_{j=1}^{n_i}\lvert x_{ij} - \bar{x_i}\rvert$	Beebe et al. [2013]
Delta Standard Deviation	It is the standard deviation of the byte frequencies. $D_{S_B} = Pr((B+1) > S_j \geq B)$	Moody and Erbacher [2008] Calhoun and Coles [2008]

Table 3.2 summarises the methods that previous researchers have used. The first column includes the method name; the second column includes references to the studies that used each method in their file fragment classification work.

Table 3.2: Machine learning methods that previous studies have used.

Method Name	**Used by (Reference)**
KNN	[Schultz et al., 2001], McDaniel and Heydari [2003], Li et al. [2005], Karresand and Shahmehri [2006b], Karresand and Shahmehri [2006a], Mokhov and Debbabi [2008], Ahmed et al. [2009], Cao et al. [2010] ,Axelsson [2010a], Axelsson [2010b], Ahmed [2010], Conti et al. [2010], Gopal et al. [2011], Ahmed et al. [2011]
LDA	Veenman [2007], Calhoun and Coles [2008], Li et al. [2011], Ahmed et al. [2011]
Neural Networks	Amirani et al. [2008], Ahmed et al. [2010], Ahmed [2010], Ahmed et al. [2011], Amirani et al. [2013]
SVM	Li et al. [2006], Li et al. [2011],Li et al. [2011], A28, A29,Ahmed et al. [2011], Fitzgerald et al. [2012],Sportiello and Zanero [2012], Amirani et al. [2013], Beebe et al. [2013], Qiu et al. [2014], Bhat et al. [2021]
others	Memon and Pal [2006], Hall et al. [2006], Erbacher and Mulholland [2007], Moody and Erbacher [2008], Mokhov and Debbabi [2008],Li et al. [2011], Kattan et al. [2010], Roussev and Quates [2013], Penrose et al. [2013]

A summary of the various sources of data used in previous studies is shown in Table 3.3. The first column lists the sources of data and in the second column is a listing of the study references.

Table 3.3: Sources of data used by previous studies.

Data Source	Used by (Reference)
Personal, search engine	Schultz et al. [2001],Memon and Pal [2006],Hall et al. [2006], Li et al. [2006],Veenman [2007],Amirani et al. [2008], Ahmed et al. [2010],Li et al. [2011],Sportiello and Zanero [2011],Ahmed et al. [2011], Sportiello and Zanero [2012],Amirani et al. [2013],Axelsson et al. [2013], Beebe et al. [2016],Mittal et al. [2020]
Garfinkel corpus Garfinkel et al. [2009]	Axelsson [2010a],Axelsson [2010b],Ahmed [2010] ,Gopal et al. [2011],Fitzgerald et al. [2012],Beebe et al. [2013] ,Qiu et al. [2014],Beebe et al. [2016],Bhatt et al. [2020]
Not given	McDaniel and Heydari [2003],Li et al. [2005] ,Hickok et al. [2005],Karresand and Shahmehri [2006b],Karresand and Shahmehri [2006a], Erbacher and Mulholland [2007],Calhoun and Coles [2008], Moody and Erbacher [2008],Ahmed et al. [2009],Cao et al. [2010], Conti et al. [2010],Kattan et al. [2010],Roussev [2010]

Table 3.4: File fragment size used by previous studies.

Fragment Size	Used by (Reference)
Complete File	Schultz et al. [2001],Memon and Pal [2006], McDaniel and Heydari [2003],Li et al. [2005], Amirani et al. [2008],Ahmed et al. [2009], Cao et al. [2010],Ahmed et al. [2010]

Continued on next page

Table 3.4: File fragment size used by previous studies. (Continued)

Fragment Size	Used by (Reference)
4096 bytes	Karresand and Shahmehri [2006b],Veenman [2007], Erbacher and Mulholland [2007],Li et al. [2011],Sportiello and Zanero [2011],Roussev and Quates [2013]
1024 bytes	Erbacher and Mulholland [2007],Calhoun and Coles [2008] ,Conti et al. [2010],Amirani et al. [2013],Roussev and Quates [2013],A50
512 bytes	Karresand and Shahmehri [2006a],Erbacher and Mulholland [2007], Axelsson [2010a],Axelsson [2010b],Gopal et al. [2011],Sportiello and Zanero [2011],Fitzgerald et al. [2012],Sportiello and Zanero [2012],Beebe et al. [2013],Bhatt et al. [2020]
Less than 512 bytes	Hall et al. [2006],Erbacher and Mulholland [2007], Moody and Erbacher [2008],Ahmed [2010],Kattan et al. [2010], Ahmed et al. [2011]
Larger than 4096	Erbacher and Mulholland [2007],Roussev and Quates [2013]

Continued on next page

Table 3.4: File fragment size used by previous studies. (Continued)

Fragment Size	**Used by (Reference)**
Header or Header and Footer bytes	Hickok et al. [2005],Li et al. [2006],Ahmed et al. [2011]

3.4 Summary

This section has presented a literature review of the techniques commonly used for file fragment classification. We have discussed the feature engineering and classification methods applied to this problem, as well as the datasets used and the specifics of the experiments employed to evaluate the techniques (e.g. fragment size, fragment location, data types, etc.). A detailed list of previous work using these approaches was provided in three tables. Although many existing approaches to file fragment classification can quickly identify the overall type of a document, few of them can reliably and accurately classify file fragments from a large number of file type classes, or if only considering fragments that originate from the middle of a file. Following from this literature review, in the next chapter we will detail the research methodology framework that we will adopt to develop and test new methods for file fragment classification.

4 Research Methodology

4.1 Introduction

The general purpose of this chapter is to describe the framework we will use to address the file fragment classification problem. Section 4.2 introduces the general methodology framework, including describing each component in the framework, which includes data representation, modelling and evaluation.

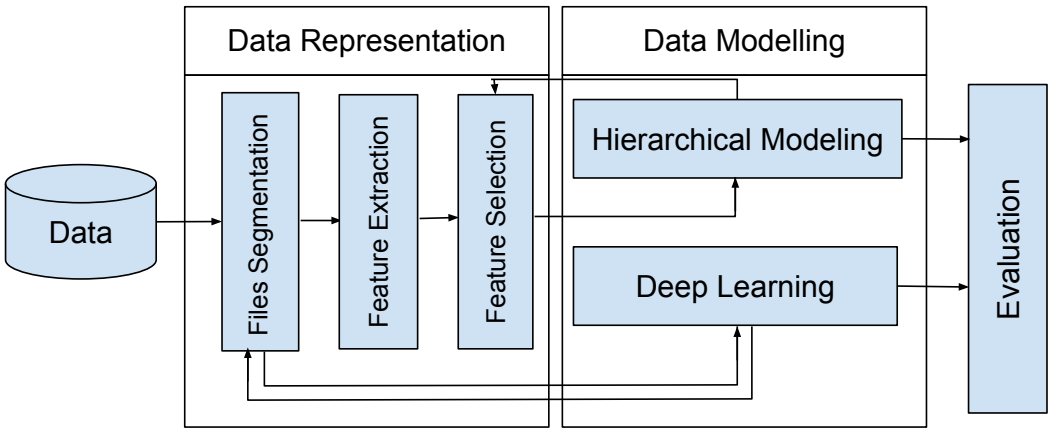

Figure 4.1: Research methodology. Comprising four components: Data, data representation, modelling, and evaluation.

4.2 Research Methodology Framework

Based on the literature review chapter, it is clear that there is no single and general method that could accurately and reliably classify file fragments. Some of these studies provide a

good level of accuracy as they only consider a limited number of file classes. Other studies achieved high detection rates by relying on metadata, which explicitly helps to indicate different file types. However, this is not the case for fragmented files where metadata is unavailable. Lastly, the accuracy level will be higher when small datasets are used and when large fragment sizes are used. However, some studies that exclude metadata and involve a large number of file classes either achieved a poor accuracy level or are not appropriate for application to forensics. Therefore, there is a need for developing new methods to classify file fragments reliably and accurately because each of the existing strategies have their own strengths in different areas. The existing approaches also have their own weaknesses, and none of them performs universally well in all situations. To address this, we propose the use of two different approaches, hierarchical and deep learning methods, to achieve more accurate and reliable classification of data file fragments. Figure 4.1 shows our proposed research framework for classifying structured and unstructured data file fragments. It consists of three main components: Data Representation, modelling, and evaluation. The purpose of each element will be explained in detail in the following sections.

4.2.1 Data Representation

This component includes four main steps: Data collection, data cleaning, data segmentation, and feature engineering. Each will be explained in the following sub-sections:

Data Collecting

The dataset we built for this research is derived from publicly accessible file corpora [Garfinkel et al., 2009] that are freely available online. According to Collobert et al. [2011], using a publicly available dataset not only allows scientific tests to be repeated, but also provides researchers with a benchmark to compare their results with previous work in this area. Therefore, our file fragments dataset has been derived from the gov doc file corpus, which is a website designed to support computer forensics education and research. Memory dumps, desk images, and network packets contained in digital corpora are all available for

free and can be used by researchers without prior approval. Further details of the dataset will be discussed in Chapter 5.

Data Cleaning

This research is mainly concerned with the classification of file fragments in the absence of their metadata. Therefore, we must simulate file fragments that do not contain meta information. As a result, we remove the metadata from the files in the dataset. Chapter 5, Section 5.2.1, discusses this process in more detail.

Data Segmentation

We are also in need of a data segmentation process since our main concern is the classification of the file fragments. For this reason, we segment our files into fragments of specific sizes. The process of segmentation, as well as the size of the fragments, are described in detail in Chapter 5, Section 5.2.1.

Feature Engineering

Feature engineering in this research includes two steps:

1. Feature extraction is the process of transforming raw data into features. In this research, the feature extraction stage generates specific features as file fragment attributes.

2. The feature selection process follows feature extraction to determine and involve the most informative features. Several feature selection techniques have been applied in this research. The details of data representation and experiments are presented in Chapter 5.

4.2.2 Modelling

In machine learning, modelling refers to using a machine learning algorithm and running it over data to create a machine learning model. To accomplish the objectives of this study, we investigated two different approaches to modelling: Hierarchical modelling and deep learning modelling.

Hierarchical

In this component, a number of machine learning algorithms will be used to develop models that can classify the dataset created in the previous component. Our hierarchical modelling includes hybrid feature selection when training classifiers. Two different hierarchical approaches were considered, which are Binary Tree (BT) and Directed Acyclic Graph (DAG). The details of these novel hybrid hierarchical modelling approaches for file fragment classification will be explained in Chapter 6.

Deep Learning

As noted in Chapter 2, the classification of file types is similar to the text mining task in some ways. Specifically, file fragments, which are a sequence of bytes, can be considered as similar to a sequence of characters, words, or text. Therefore, applying deep learning techniques that have been used for text mining may also work well for file fragment classification. As we have mentioned before, finding representative features is one of the most challenging tasks that affects the accuracy of classification. Thus, deep learning with unstructured data will allow the methods to determine the most distinctive features without supervision, from the raw data (raw data being the binary content of file fragments in our task). Further research on deep learning and how to employ it to our problem will be discussed in Chapter 7.

4.2.3 Evaluation

This section describes the existing methods and measures that will be used to evaluate the results of this research methodology.

Data Partition Strategies for Training and Testing

Hold-out: This method divides a dataset randomly into three subsets which are:

1. Training set: this portion of the dataset is used to build predictive models.
2. Validation set: this portion of the dataset is used to check the performance of the model built in the training phase.
3. Test set: this portion of the dataset of unseen examples is used to estimate the future performance of a model.

Evaluation of Effectiveness

1. Confusion Matrices: A numerical and graphical representation of the classification results in terms of correct and incorrect classifications. Allows the identification of classes that are commonly confused with each other during testing.
    - Binary class confusion Matrices The binary confusion matrix is 2x2 matrix containing the actual and predicted values from the proposed classier. An example of a binary class confusion matrix is shown in Table 4.1

Table 4.1: An example of a binary confusion matrix.

		Target (Actual Value)	
		Positive	Negative
Model (Predicted Value)	Positive	TP	FP
	Negative	FN	TN

- TP is the number of correct predictions for the positive cases.
- FP is the number of incorrect predictions for the positive cases.

- FN is the number of incorrect prediction for the negative cases.
- TN is the number of correct predictions for the negative cases.

- Multi-class Confusion Matrices

  The binary confusion matrix is less complicated than the multi-class confusion matrix. An example of a confusion matrix including four different classes is shown in Table 4.2. The intersection of the actual class and its predicted class is presented by TP (The true positive values).

Table 4.2: An example of a multi-class confusion matrix.

		Predicted			
		a	b	c	d
Actual	a	TPa	Eab	Eac	Ead
	b	Eba	TPb	Ebc	Ebd
	c	Ecs	Ecb	TPc	Ecd
	d	Eda	Edb	Edc	TPd

2. Average Accuracy

   The accuracy of model's prediction can be calculated from its confusion matrix with the following equation.

$$Acc = \frac{TP + TN}{TP + FP + FN + TN}$$

3. Precision

   The precision of a prediction is the ratio of correctly predicted positive observations to the total number of correctly predicted positive observations.

$$Precision = \frac{TP}{TP + FP}$$

4. Sensitivity (Recall)

   The recall is determined by the ratio of correctly predicted positive observations to

the total number of positive observations in the actual class.

$$Recall = \frac{TP}{TP + FN}$$

5. F1

   It is a weighted average of precision and recall. As a result, this score accounts for both false positives and false negatives. F1 is not as intuitively understandable as accuracy, but it is often more useful than accuracy, especially with uneven class distributions. Accuracy works best if false positives and false negatives have a similar cost. It is better to consider both precision and recall if the cost of false positives and false negatives is very different.

$$F1Score = \frac{2 * (Recall * Precision)}{(Recall + Precision)}$$

Significance Tests

Statistical analysis is necessary to assess the performance of the proposed method in term of being significantly better than other methods. We will use the Wilcoxon-test and the Friedman test to evaluate the performance of the proposed ensemble against other methods.

1. Wilcoxon-Test

   Pairwise comparisons can be performed using Wilcoxon signed-rank tests [39 aliyah]. This test calculates the differences in the effectiveness of the two considered classifiers, ranks them according to the calculated differences, and compares their positive and negative ranks. We use a control algorithm to provide a baseline for our comparisons and set the statistical significance threshold at 0.05 (alpha = 0.05).

2. Friedman-Test

   The Friedman test can be used to determine whether a statistically significant differ-

ence in performance is observed across several classifiers applied to different datasets. However, the test does not reveal which classifier has produced the best performance. Instead, the Friedman test is supposed to be followed by a post-hoc test, which will determine which classifiers have produced results which are significantly different from the others. The post-hoc test we use is the Nemenyi test. In this test, a critical difference diagram is generated, which requires results from a control classifier to generate a ranking for each classifier being tested. These rankings are then used to determine whether the results of two classifiers are significantly different, according to a critical difference parameter, which is calculated as

$$CD = q_a \sqrt{\frac{k(k+1)}{6N}}$$

A critical value is easily obtained from any statistics textbook, k is the number of classifiers included, and N is the number of samples.

4.3 Summary

In this chapter, we have outlined the general framework that we will follow in this research. We have described each component of the framework individually, including data preparation, modelling, and evaluation. The next chapter will describe the data preparation and feature engineering techniques used in this work.

5 Data Preparation and Feature Engineering

5.1 Introduction

This chapter provides an overview of how we generated the dataset that was used in the experiments presented in this thesis. This includes the steps taken to prepare the data and engineer the features. Throughout this chapter, we will discuss how files are transformed into features for application to machine learning algorithms. Section 5.2 describes how to construct a dataset that is suitable for machine learning algorithms by converting a set of files into a structured dataset. This is followed by Section 5.3, Experiments, which provides an overview of this chapter's experiments and how we select the most appropriate filters, machine learning algorithms and the steps to achieve this chapter's objectives. Then Section 5.4 provides an overview of the results of the experiments and the data obtained to evaluate this chapter's methodology. Lastly, Section 5.5 summarises this chapter's findings.

5.1.1 Contributing Publications

The work in this chapter forms part of a publication presented at the International Conference on Innovative Techniques and Applications of Artificial Intelligence [Algurashi and Wang, 2019].

5.2 Data Preparation and Feature Engineering

This work is the first part of our thesis methodology stated in Chapter 4. Data preparation, feature selection, and evaluation are the three major steps in our procedure, as shown in Figure 5.1. Each step is described in detail in the following sections.

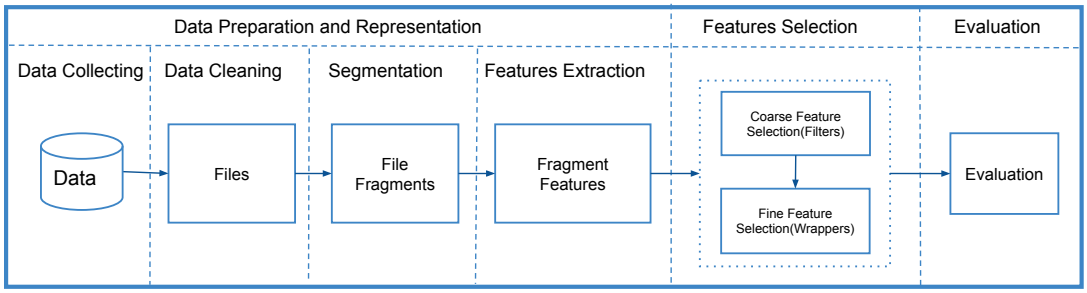

Figure 5.1: Data preparation and feature engineering with coarse and fine filtering steps.

5.2.1 Data Preparation and Representation

In data preparation, unstructured data is transformed into a structured form. This section discusses the four main steps that have been taken for data preparation, including data collecting, data cleaning, data segmentation, and feature extraction.

Data Collecting

As noted earlier in Chapter 4, we use the public file objects corpus [Calhoun and Coles, 2008]. A sample of 13 types of files is included in our experiments. The selection of file types was based on the relevance and popularity of those file types. Table 5.1 provides a static description of our initial file samples that have been used in this chapter. The total number of files used was 2600. This resulted in the processing of 81.94 GB of data.

Table 5.1: Dataset summary: File types, amounts, and sizes used in the research.

		Files size		
File type	Number of file	Total in MB	Max in KB	Min in KB
CSV	200	44.9	7,612	2
DOC	200	44500	13,838	22
HTML	200	3.12	465	1
JPEG	200	638	2,590	4
LOG	200	2290	16,459	5
PDF	200	73.4	4,915	7
PNG	200	5	976	4
PPT	200	84.4	5,753	34
TXT	200	71,000	28,705	1
WAV	200	398	391,000	12
XSL	200	2000	441,772	147
ZIP	200	31900	791,706	169,376
all	2600	81936.828		

Data Cleaning

The data cleansing process identifies incomplete, incorrect, inaccurate, or irrelevant file samples and replaces, modifies, or deletes dirty data. Files with less than 512 bytes were excluded. We manually reviewed the files to verify that the file extension matched the file content, as some files might have been manipulated to display the incorrect extension.

Files Segmentation

File fragmentation, as discussed in Chapter 1, may pose a significant problem for researchers in identifying file types. File signatures (magic numbers) can be used as direct methods to recognise data types, either from their headers or from their headers and footers. Files that have not been fragmented can be classified by looking at the few bytes in the file header and/or footer to determine the type of file. The complete un-fragmented file is not our concern in this research. In line with the literature review chapter, the purpose of this study is to classify file fragments that might be missing their metadata. For that purpose, the given set of files needs to be chunked into fragments of a defined size. Garfinkel et al. [2009] reported that using the smallest sector size for research is the most appropriate and

generalised option. This is because larger fragments allow the learning model to incorporate more information, which may not be representative of fragments of a smaller size. As 512 bytes is the smallest possible size of a sector, we chose to use the worst-case scenario and fragment files into 512-byte chunks. We developed a program that partitions files into 512-byte chunks, taking into account removing the first two bytes that may contain metadata. Every file is read and divided into 512-byte chunks, then stored separately.

Feature Extraction

In machine learning, feature extraction is the task of processing raw data to generate a structured dataset consisting of rows of samples and columns of attributes. A computer file is an object that consists of a sequence of bytes. To remind the reader, each byte contains 8 bits, hence a byte can be represented as a numerical value ranging from 0 to 255. There are many different statistical measures that can be calculated for file fragments. Table 3.1 in Chapter 3 summarises these statistical measures that have been used across 34 different studies to represent file fragment features. The unigram or Byte Frequency Distribution (BFD) is the most frequently used attribute. In this work, we employ 11 vectors of the feature. Each vector varies in size. For example, the BFD feature comprises 256 features. A program has been developed to obtain statistical features. Algorithm 1 illustrates that the BFD for each fragment in our dataset is obtained by finding the frequency number for each byte value, where we loop over each fragment and count the occurrence numbers of each byte value, resulting in an array of 256 elements for each fragment.

Algorithm 1 Byte Frequency Distribution (BFD)

Input: fragment ▷ file fragment of size 512
Output: Frequency distribution of file fragments
1: *Initialisation* :
2: $BFD = [256]$
3: $B = 0$
4: **function** LOOP($fragment$)
5: $BFD[fragment[B]] + 1$
6: $B = B + 1$
7: **return** BFD
8: **end function**

Algorithm 1 shows how the program loops across each file fragment and finds the frequency bigram. It is similar to BFD, but it takes into account the sequence of each two consecutive bytes. Hence, a word of two bytes in length is represented as a numerical value ranging from 0x0000-0xFFFF. To find the bigram vector, we counted the occurrences of each word value (two bytes) in each instance, resulting in a 65,536-dimensional vector.

Algorithm 2 describes how the program loops through each fragment and finds the difference of each consecutive byte, providing an array of 511 elements in size. These values are used as features of the fragments. These values are represented as R0, R1... R511. R1 refers to the difference between the first and second byte values, while R2 refers to the difference between the byte values of the second and third bytes and so on.

Algorithm 2 Rate Of Change (ROC) in byte value

Input: fragment ▷ file fragment of size 512
Output: The rate of change in file fragments
1: *Initialisation* :
2: $RoC = [511]$
3: $r = 0$
4: **while** $r \leq 512$ **do** ▷
5: $RoC[r] = absolute(fragment[r] - fragment[r + 1])$
6: $r = r + 1$
7: **end while**
8: **return** RoC ▷

Likewise, we calculate the vectors of all the other features listed in table 3.1 Each vector has a different size. There are 256 features in BFD, 65,536 features in bigram, and 255 features in frequency of rate of change, all of which are combined with single vector features (entropy, standard deviation, mean, etc.) to form all together 66,313 features. All these data are stored in databases. Each time we need a sample of file fragments, we select it randomly from the file fragment database. For this chapter experiment, a random sample of 300 fragments from each of 14 file types was selected to provide a balanced dataset, giving 3,900 samples with 66,313 features in total.

5.2.2 Feature Selection

In this work, the hybrid approach was proposed to handle the large dataset and overcome the limitations of both a dependent filter and a dependent wrapper, to provide a usable, effective and accurate feature set.

Our hybrid feature selection model will be built using the filter method as the first step. First, we must answer three questions: what is the most subtle filter for our data, what baseline learning algorithm is the most appropriate for our data, and at which point we must cut the ranked feature set. On the basis of our previous discussion, we saw that the filtering method differs according to the statistical tests they use. Consequently, the most useful filter for our data would be information gain and mutual information gain. We excluded the others since they do not work well with a large number of features, as we have more than 66K in our case. Furthermore, a baseline algorithm must be selected to evaluate the performance of the selected subsets of features and decide which should be used as the base learner.

5.3 Experiments

5.3.1 Preliminary Experiments (Finding Suitable Filters and Learning Algorithms)

In this section, we present an overview of our preliminary experimental setup. Figure 5.2 illustrates the design of the filtering step experiment. For selecting features, two-process filtering and wrappering are used. To move forward to the wrapper step, there are three questions to answer. First, what is the best filter for our data? Next, can we determine which learning-based algorithm will work best with our data? The final question is, after ranking our data with a filter-based selected in question one, at what point should we cut features for the wrapper? For example, should we include the first 100 or 1000 or exactly how many of the ranked features should be used?

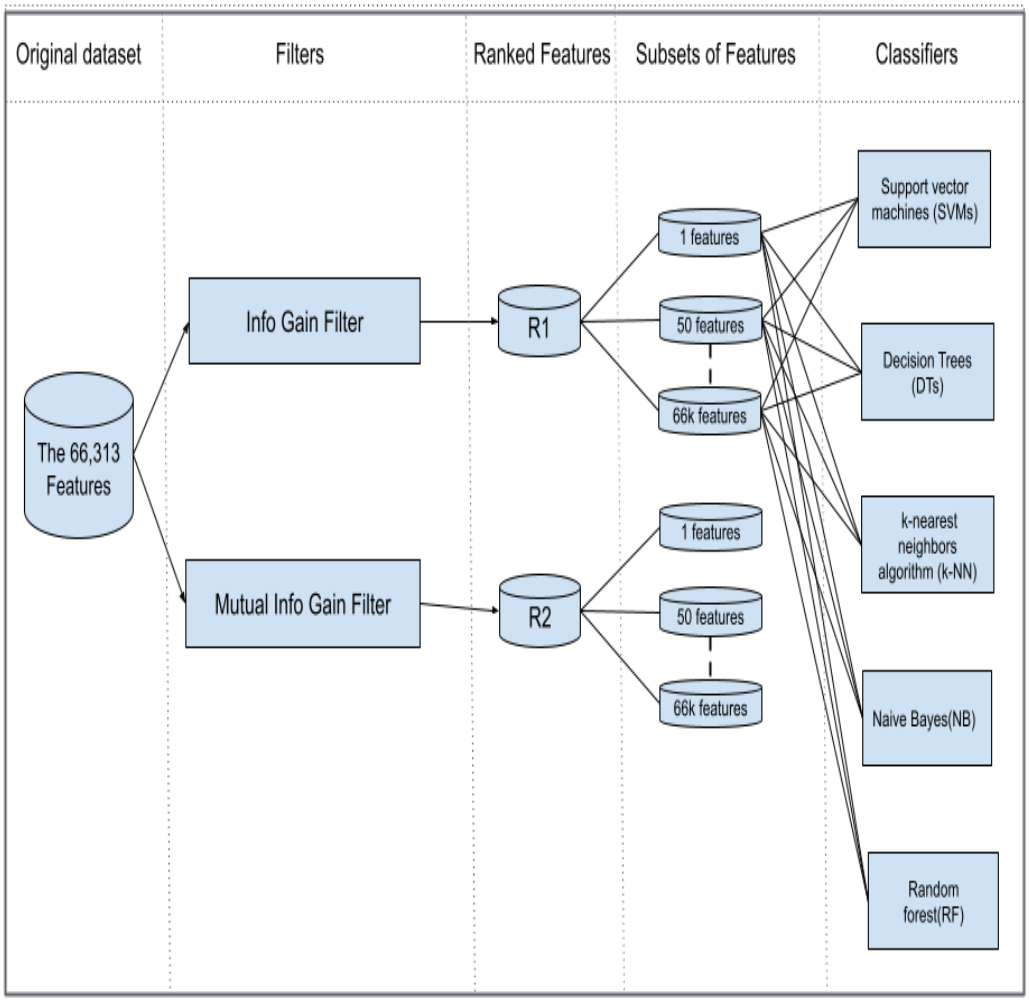

Figure 5.2: Preliminary experiments illustration. Identifying the best filter, learning-based algorithm, and optimal feature subset for the wrapper process.

According to the filter section in this thesis, since our data has numerical input features (file fragment statistical features) and categorical output (file types), we selected two different filters: information gain and mutual information gain, as these are the most appropriate filters for our dataset.

Next, we used the best-first search algorithm with the two selected filters to rank the features from most to least relevant. The two output-ranked features are divided into subsets, each

including a different number of features. It ranged from the first most relevant feature to the 66,313th feature, using a step size of 50.

It is necessary to decide which algorithm base learner we will use to evaluate the performance of the subsets of the features selected by the two filters. We used five learning algorithms: Decision Trees (DT), K-Nearest Neighbours (KNN), Support Vector Machine (SVM), Naive Bayes (NB), and Random Forest (RF). These five algorithms were chosen because they are well known, commonly appear in the literature, and are very different in their approaches to classification. Through these steps, we can answer the previous three questions.

5.3.2 Hybrid Feature Selection

In the previous experiment, we decided which filter to use, the most appropriate base classifier to adopt, and the threshold at which we should cut the ranked filtered features for input into the wrapper step. Figure 5.3 illustrates the steps of the hybrid feature selection experiment. The first step in this experiment is to apply a wrapper to these ranked features and to see how that could improve the accuracy. The recursive feature elimination wrapper was used because of its simplicity over other types of wrapper. In this step, we apply the wrapper to 13 feature subsets for each of the 13 file classes. To evaluate our hybrid feature selection method, we compared it with RF classifiers that used the top-selected ranked feature filters using the chosen filters. Again, the dataset was divided into training and testing subsets at a ratio of 60:40. The evaluation measures used were average accuracy.

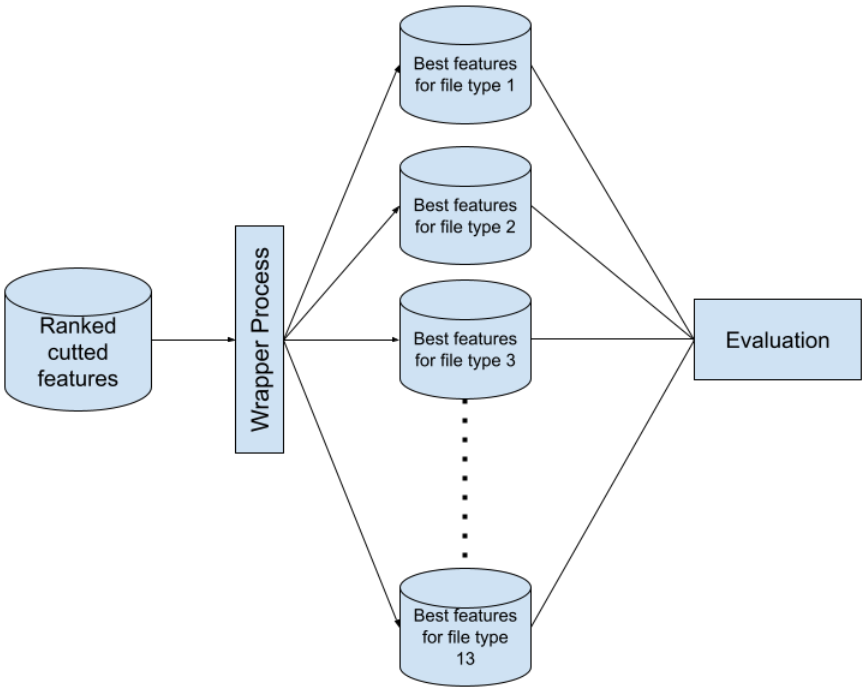

Figure 5.3: Hybrid feature selection experiments illustration. Applying wrapper to ranked features for improved accuracy.

5.4 Results

5.4.1 Preliminary Experiments For Finding Suitable Filters and Learning Algorithms

This section explains the evaluation results obtained from the preliminary experiments. Figures 5.4 and 5.5 show the accuracy achieved by DT, KNN, SVM, NB, and RF that used the 1326 subsets of features, using two different filter types. Figure 5.4 relates to the results obtained using the information gain filter. Note that the 1326 subsets are derived from splitting the total number of features (66,313) into intervals of 50 features. The results show that most classifiers achieve greater accuracy as the number of features increases. The RF

classifier achieves the best accuracy compared to the other four classifiers tested. However, this result does not help our objective of reducing the number of required features.

Figure 5.5 shows the results of using the mutual information gain filter to rank the features. These results are quite different from those obtained in Figure 5.4. The results show a much more consistent level of performance as additional features are included, and the mutual information gain filter actually achieves better accuracy with fewer features. The five classifiers perform similarly when they use the mutual information gain filter. However, the five classifiers achieved their best accuracy levels at different cut-off thresholds. For example, RF achieved its best accuracy using only 900 features of the 66,313 features, which means that 69% of the original features might be considered redundant and we can discard them. In summary, the mutual information gain filter is shown to be a better method of ranking features for feature selection than the information gain filter.

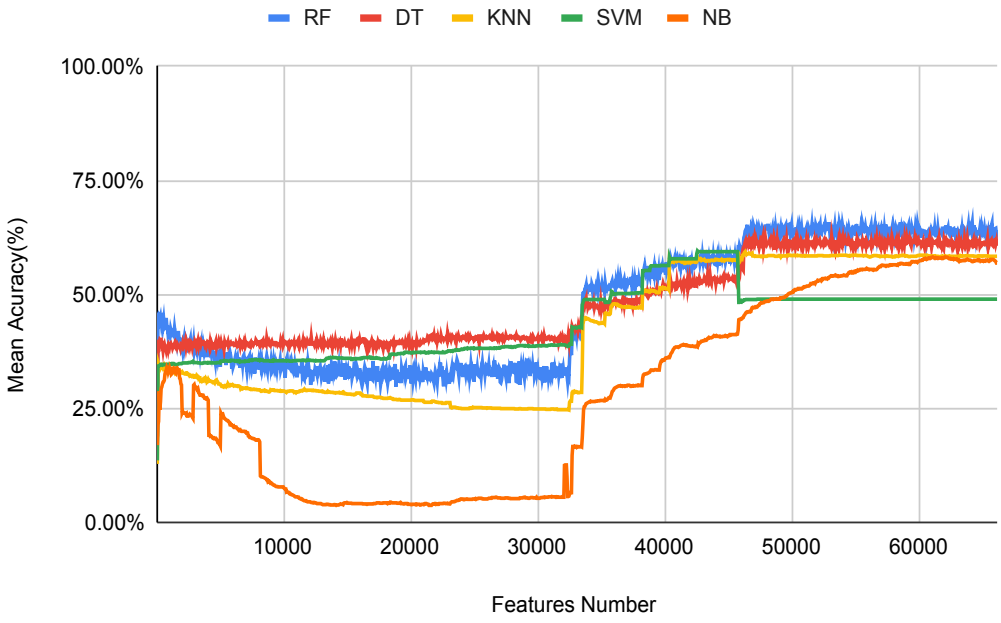

Figure 5.4: A comparison of the accuracy achieved by DT, KNN, SVM, NB, and RF using the information gain filter.

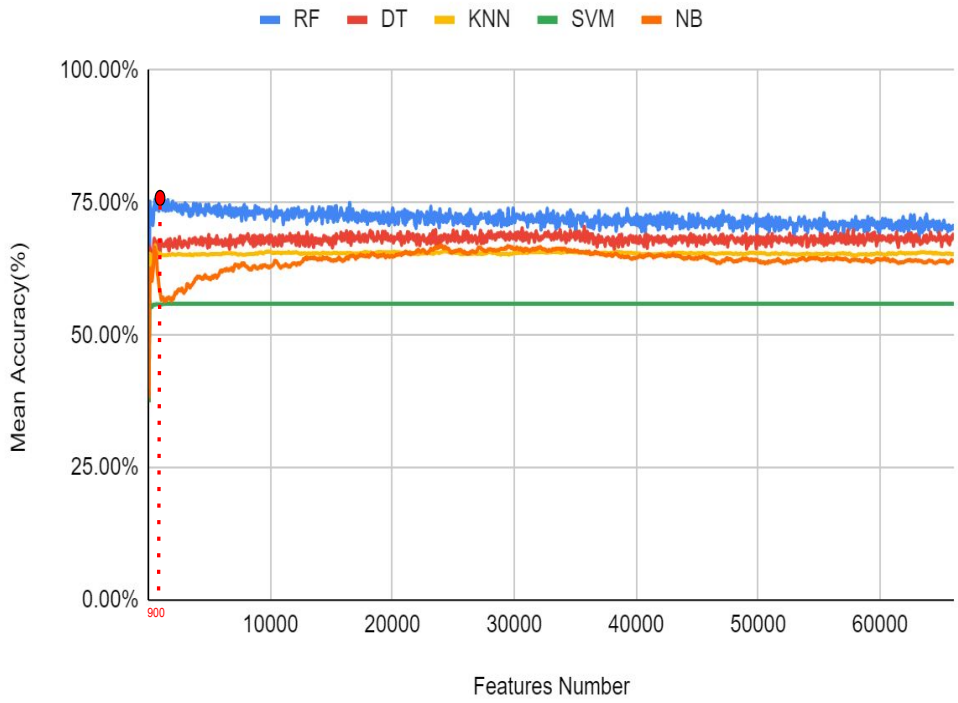

Figure 5.5: A comparison of the accuracy achieved by DT, KNN, SVM, NB, and RF using the mutual information gain filter.

Figure 5.6 presents the average accuracy of the five base classifiers across all subsets of features tested using the two filters. This graph indicates that the classifiers generally performed better when using features ranked by the mutual information filter, and the best performance has been achieved using the random forest classifier.

Following these results, the decision regarding the most appropriate filter method to use for feature selection and the appropriate learning model can be finalised. A mutual information gain filter and random forest based classifier would be suitable methods for applying to our experiment. The next section will explain the hybrid feature selection experiment and provide its results.

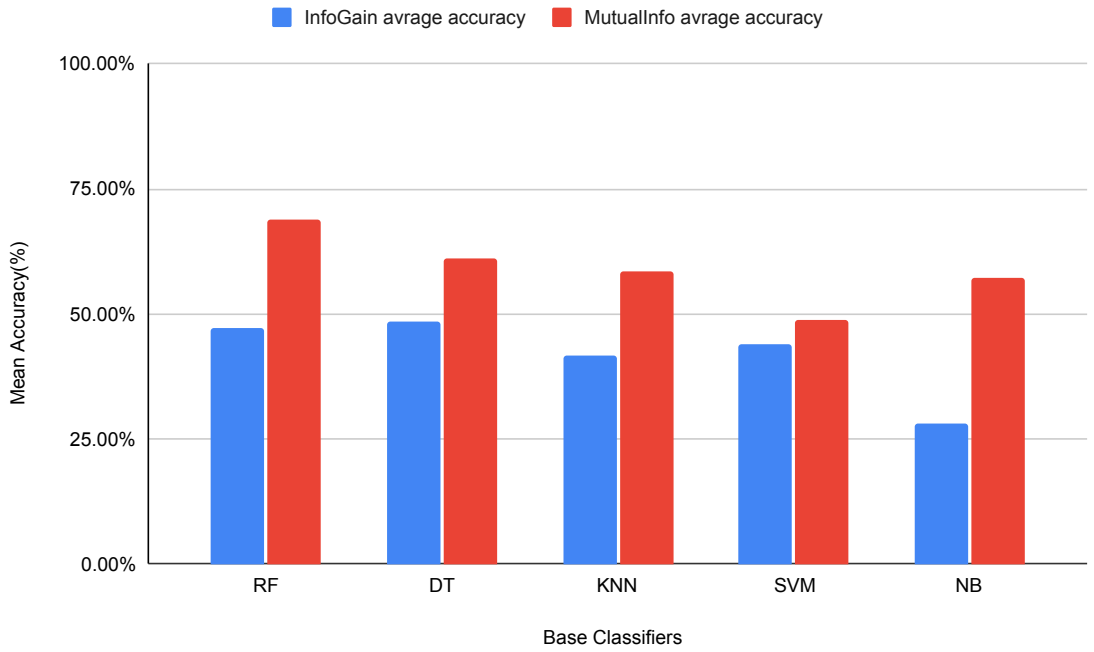

Figure 5.6: A comparison between the average performance of five base models using features selected by information gain and the mutual information gain filter.

Hybrid Feature Selection

From the previous section, we decided to use the mutual information gain filter and random forest as a base classifier. So, the features have been ranked using the mutual information gain filter and we use the only first 900 features, as it is the point where random forest achieved its highest accuracy in our preliminary experiments. The next step is to apply a wrapper to these 900 features and see if that can further improve the accuracy. Recursive feature elimination wrapper was considered because of its simplicity over other approaches. In this step, we selected the best 11-50 feature subsets from each of the 13 file classes.

To evaluate our hybrid feature selection method, we compared it with RF classifiers that used all 66,313 features, top 900 ranked feature filters using mutual information gain filter and a hybrid feature selection method. 42 classifiers were trained using the 3 subsets of features produced in the previous step (66K, 900, 11-50). Again, the dataset was divided

into training and testing subsets in a 60:40 ratio. The evaluation measure used was the average accuracy. Figure 5.7 shows the mean classification accuracy of our test results for the classifiers that were trained with three sets of features. The first one contains all features (66,313 features), the second one includes only 900 of the features selected by the mutual information gain filter, and the third one includes features selected by the wrapper.

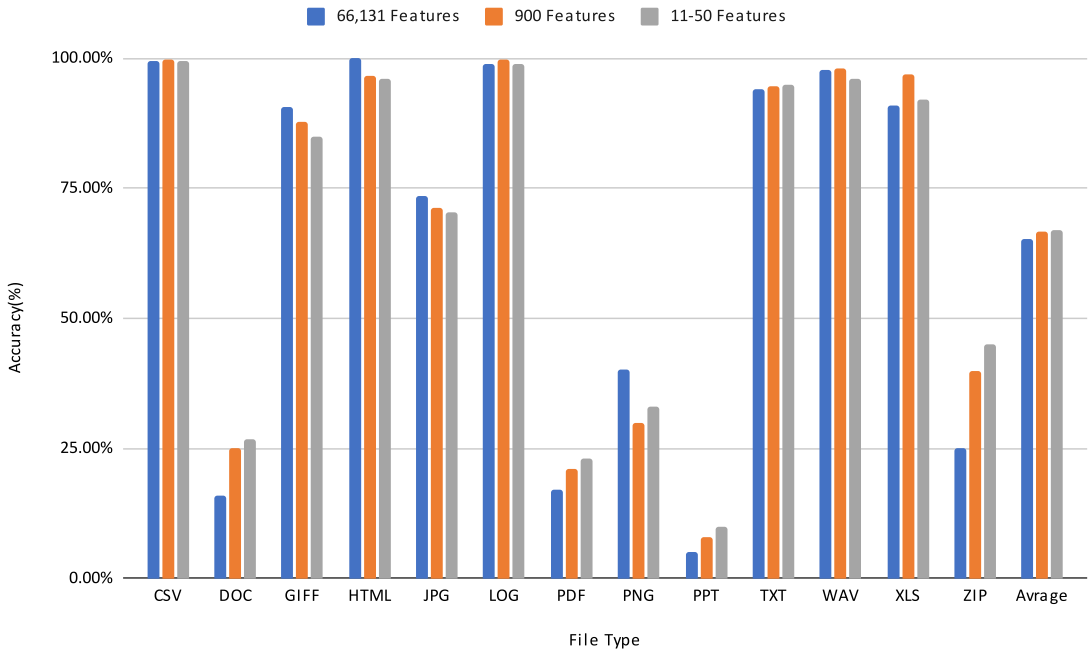

Figure 5.7: The accuracy of each file type using all features, the first 900 features selected by the mutual information gain filter and the hybrid selected features.

In general, our findings indicate that the learning algorithms functioned better and in a similar manner when they were applied to a smaller number of the selected features. For CSV files, we achieved a 99.43% mean accuracy with the full set of the features. This increased to 99.53% when using hybrid selected features. The same patterns were observed in most other types of file classification. For PDF files, we also saw a strong improvement in classification accuracy. At first, we achieved 16.96% accuracy with all the features, which rose to 21.33% with 900 selected features. Finally, when the hybrid selected features were

used, our method achieved 23.25% - the highest accuracy. Similarly, the ZIP, and DOC file types achieved their best classification accuracy using Hybrid selected features.

Some file types, such as HTML fragments, experienced a slight drop in accuracy. With the full set of features, we achieved 100% accuracy for HTML. When using 900 selected features, this decreased to 96.57%, and again when using hybrid selected features, it decreased again to 96.00%. Similarly, the JPEG and GIF file types show similar behaviour. However, considering that the number of features was reduced from 66,313 to only 50, about 99.96% of the features were removed. This means that our hybrid method was effective in getting rid of the majority of irrelevant features and was accurate in selecting a smaller number of very informative and useful features.

5.5 Summary

The purpose of this chapter was to demonstrate how files can be transformed into statistical features that can be inputted and processed by machine learning algorithms. We also showed how hybrid feature selection can help improve the detection rate of file fragments. We provided background to allow understanding of the feature selection methods, and described how we chose the most suitable method for our research. In summary, our results indicate that employing our hybrid feature selection method, a two-stage coarse-to-fine selection process, is able to reduce the dimensionality of the dataset significantly, as well as improve the accuracy of file fragment classification in most cases. In light of our findings, we recommend using the mutual information gain filter to rank the relevant features of each file, as this improved the performance of the classifiers. Furthermore, we recommend that a wrapper be used with a forward selection approach in order to save time (as this approach is typically faster than backward selection). In the next chapter, entitled Using hybrid feature selection and hierarchical modelling for file fragment classification, we will examine if this can improve file fragment classification accuracy.

6 A Novel Hierarchical Hybrid Approach For File Fragment Classification

6.1 Introduction

We focused our attention in the previous chapter on finding the best features for the classification of file fragments. In addition, we generated classifiers for each file type (specialised classifiers) and compared the accuracy achieved by each model. However, it is not straightforward to create a general model for file fragment classification and to combine multiple one-against-all predictions for multiclass classification. One way to combine specialised classifiers is by integrating them into a hierarchical model. This chapter considers two hierarchical models for solving file fragment classification. We investigate how integrating hybrid feature selection into hierarchical models can enhance the accuracy of file fragment classification and how putting the type of file in the hierarchy tree might narrow the possibilities when faced with a challenging type of file. This chapter is organised into five sections. Section 6.2 illustrates the general hierarchical hybrid modelling, including three steps, which are data representation, feature selection, and modelling. The details of each of these steps are discussed separately in the following subsections. Section 6.3 then describes the experiment that has been carried out in this chapter to validate the hierarchical approach for classifying file fragments that have been described in this chapter. The results of the experiments conducted on hierarchical approaches are presented in Section 6.4. The conclusions of the findings of the chapter are presented in Section 6.5.

6.2 Hierarchical Hybrid Modelling

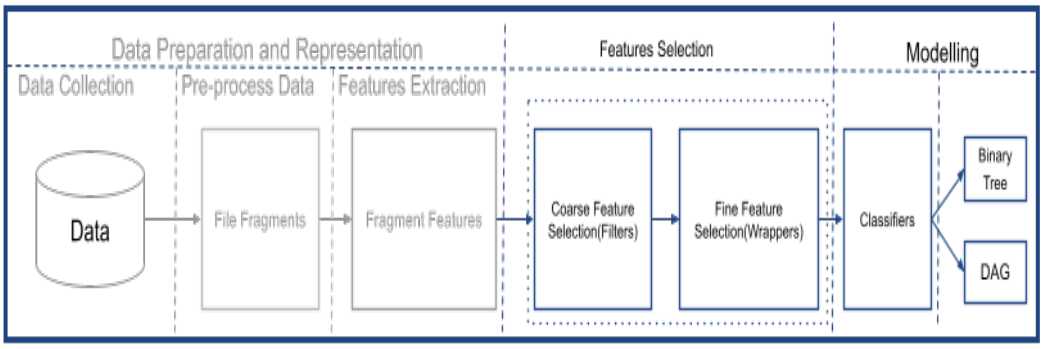

Figure 6.1: Illustration of hierarchical hybrid modelling that includes data representation, feature selection, and modelling.

Hierarchical classification refers to grouping things based on a hierarchy, or a set of levels and orders. File types can be arranged in a hierarchical manner, with general classes at the top followed by specific classes further down the hierarchy. This is opposed to the classic classification approach, where each example is simply categorised into one final class label.

The hierarchical approach contains multiple classifiers, with the exact number and arrangement of those classifiers dependent on the technique adopted. An effective method is to have a local classifier at each node, allowing the multi-specialised classifier to make a decision [Ramírez-Corona et al., 2016]. For example, if we have a four-class classification problem, a classic approach is best served by features that perfectly separate these four classes. In contrast, each local classifier in the hierarchical approach only needs to distinguish three or fewer classes to work effectively. To remind the reader, the two main challenges facing the researcher for the file fragments classification problem are the high dimensionality of statistical features and the vast number of available file types. The issue of dimensionality has been solved in the previous chapter using the hybrid feature selection method, and the

second challenge is now the focus of this chapter. The primary goal of this chapter is to demonstrate that higher classification accuracies may be achieved by producing coarse to fine (general to specific) classification models. As a result, we are investigating the integration of hybrid feature selection into hierarchical models with the following objectives:

1. Determine the hierarchical relationship between file classes.
2. Break down the problem by limiting the number of classes at each node (the fewer the number of classes, the better the classification might be).
3. Combine local classifiers to reach a decision rather than relying on a single decision by one model.
4. Consider selecting features for the local classifier instead of using the same features for all node classifiers; this may aid in classification.

Binary Tree (BT) and DAG were proposed as two common hierarchical structures. This chapter, Hierarchical Modelling, is the third part of our thesis methodology stated in Chapter 4. We explain the strategy that we use to accomplish the goal of this chapter. Figure 6.1 shows how to achieve the hierarchical hybrid modelling of this chapter, including three main steps. The first is the data representation, which was detailed in Chapter 5. The second component is the feature selection, where we will build upon the findings of Chapter 5 and more details are presented in Section 6.2.2. Lastly, we have the modelling section, which is the main focus of this chapter; in Section 6.2.3, we describe how we implemented the two hierarchical models (BT, DAG) and the dataset we used to evaluate them.

6.2.1 Data Representation

Refer to Chapter 5, Section 5.2.1.

6.2.2 Feature Selection

In Chapter 5, we discussed that selecting the best feature using the recursive feature elimination algorithm is both time and resource intensive when applied to a large number of features. Due to this, we followed the coarse-to-fine approach described in the Chapter 5 methodology section. We found that mutual information gain filter and random forest are the most appropriate algorithms for filtering our data. 66,313 features were ranked from most important to least important and we chose the first 900 features based on the findings in Chapter 4. Up to this point, and based on the research methodology, the recursive feature elimination algorithm technique was used at each node of our hierarchical models of interest: BT and DAG. The salience of the 900 features selected previously by the coarse filters was evaluated using this recursive feature elimination algorithm with a forward selection approach. At each node of the hierarchy, the 900 features were then further reduced using the recursive feature elimination algorithm (wrapper technique).

6.2.3 Modelling

Binary Tree (BT)

The BT model can start learning at the root of the tree, and the grouping method divides records into two clusters. Figure 6.2 is an example of a 6-class BT classification problem. Each node holds a label of the group class. The training process continues to perform classification operations until it can accurately assign labels. Note that Figure 6.2 is just an example of a binary tree; non-leaf child nodes may end up with overlapping classifications because the adopted clustering algorithms may assign records belonging to the same class to different clusters. In some ways, our binary tree hierarchy is similar to the concept of decision trees. Using this tree, we are able to identify similarities between different file types that could lead to a more specialised classification. We now have a more limited range of possible file types from which to produce the final classification.

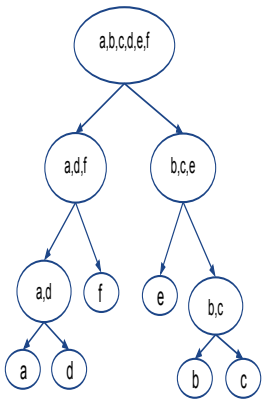

Figure 6.2: A six class BT. The root node is the top of the tree, containing all data groups. Each node has two child nodes, further splitting the data into groups of two. The tree terminates at leaf nodes when the number of classes is 1.

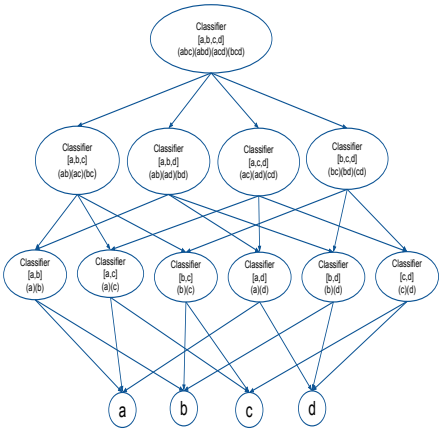

Figure 6.3: A four class DAG tree. A hierachical structure where each node consists of a machine learning classifier. There is a node and classifier for every combination of classes. The number of classes reduces by 1 as the tree progresses down from the root node.

Directed Acyclic Graph (DAG)

For the DAG structure, the tree arranges the classifiers into hierarchical structures where each node in the DAG tree consists of a machine learning classifier. DAG, like BT, requires a grouping mechanism to generate a hierarchical structure. We explored the use of K-means for this function but found that it led to similar classes being clustered together early in the hierarchy, meaning that entire branches of the tree ended up focused around very similar classes. Instead, we use a combination mechanism to cover all possible class groupings. Figure 6.3 shows an example of a four-class DAG tree. The root node classifies the input into four class groups. It is important to note that such groups are not identified directly. Instead, all possible groups are considered using a combination method. The combination method starts at the root node and finds all possible subclasses of size $|C| - 1$. Following the tree downward, the subset size decreases by 1. The process ends when the combination size is two. At this point, the number of classifiers that will be trained to build the DAG

model can be calculated as $NumberOfClassifiers = 2^N - N - 1$, where N represents the number of class labels in the dataset. For each node in the hierarchy, a classifier is trained. During testing, each level of the tree will classify a single test example as belonging to a particular grouping of classes. As we travel down the tree, the grouping becomes finer, until a single class label is assigned by the binary classifier at the leaf node.

6.3 Experiments

In this section, we describe the experiments that were carried out to verify the hierarchical methodology adopted. The dataset described in Chapter 5 is used to test the effectiveness of the models.

6.3.1 Dataset

Please refer to Chapter 5 Section 5.2.1.

6.3.2 Feature Selection

Two procedures are involved in this step. The selection process begins with a coarse procedure, followed by a fine procedure. The coarse selection process was applied once for all file fragments, whereas the fine selection procedure followed the hierarchical structure to find the best features representing each file's group at each node. To remind the reader, the coarse selection ranks features according to their correlation to the target class, and the best-first searching algorithm was used to rank the relevant features from the most to the least relevant. The question is at which point we should cut the ranked features set and include a subset for further selection (fine selection). To decide on the appropriate cutting point, we performed an experiment to generate different subsets of features starting from the first important feature and moving up to the 66,313th feature, in 50 feature intervals. A random forest classifier evaluates the accuracy achieved using different sets of features. Figure 6.4 shows the accuracy achieved by the 8,178 classifiers, trained using the 8,178 subsets

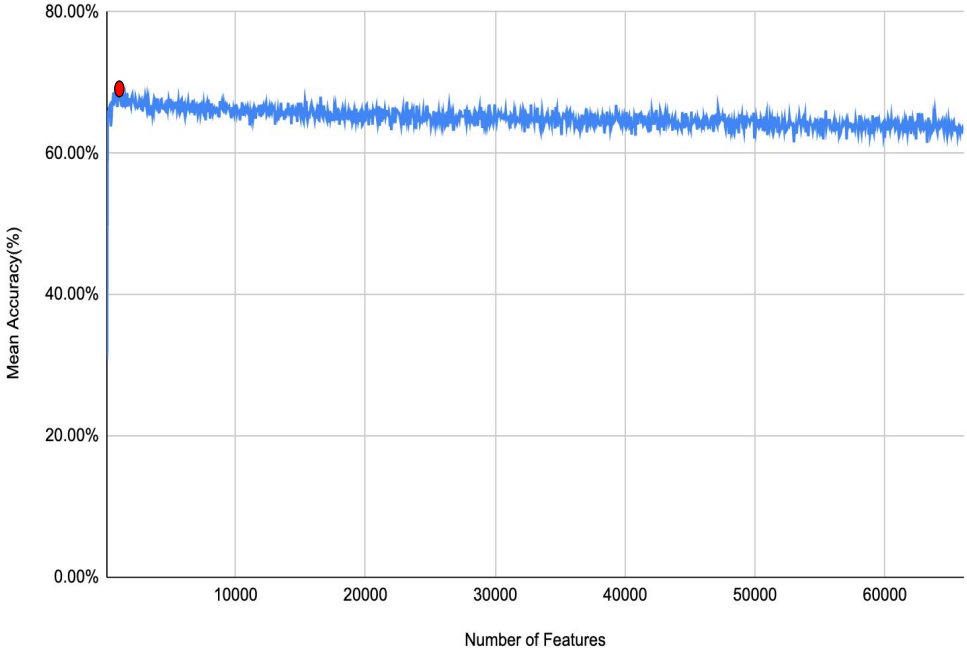

Figure 6.4: The mean accuracy of random forests using different sets of features ranging from the most important feature up to all 66,313 features.

of features. The best classification accuracy was achieved using the first 900 features, so the remaining features were considered redundant and discarded before the next step. Thus, 73% of the original features were removed. Consistent with our research methodology, the fine feature selection process using the wrapper technique was used at each node of the hierarchical models: BT and DAG tree. The forward selection algorithm is the most widely used wrapper algorithm among researchers, as it is relatively simple compared to other approaches. The salience of the 900 features selected by the coarse filters was evaluated using this recursive feature elimination algorithm, using a forward selection approach. Figure 6.4 shows the mean accuracy of random forests using different sets of features ranging from the most important feature up to the 66,313th feature. Features were ranked using the mutual info gain filter. It should be noted that, in figure 6.4, the red point represents the accuracy achieved by using the 900 most relevant features.

6.3.3 Binary Tree (BT)

To create this hierarchy, a classifier must be trained at each node and a clustering mechanism must be applied to group the input data. K-means is the most commonly used clustering mechanism. Additionally, because this research uses a fixed value of$k(k = 2)$, it is the simplest clustering method. The idea of such a clustering algorithm is, at each node of the hierarchy, to group the available class labels into two disjoint groups (clusters) so that the classes within each group share some similar byte-level characteristics. Algorithm 3 presents a BT hierarchy generation algorithm. We assume:

1. Every node in the tree is assigned a classifier
2. Left: refers to the left branch of the hierarchy
3. Right: refers to the right branch of the hierarchy.

The initial input to the algorithm, D, is the entire training dataset, which includes examples of file fragments from all file type classes. As the initial number of classes is greater than two, the procedure for generating the node is as follows: K-means clustering is used to divide the data into two temporary sets, $D1$ and $D2$. Feature selection is applied to $D1$ and $D2$, and the output is assigned the temporary labels $G1$ and $G2$. A classifier is then trained to discriminate between $G1$ and $G2$. This forms the root node. $G1$ and $G2$ are passed separately to the start of the algorithm to generate new left- and right-hand nodes. This process continues recursively until the number of classes at a node is less than or equal to two. When it is equal to two, the clustering step is omitted and the wrapper feature selection method is applied to the node's input data, followed by the generation of a classifier. The process terminates when the number of classes at a node is equal to one, at which point the node is a leaf node that will simply return a single class label. Once the tree has been generated, new unseen data examples can be classified as described in Algorithm 4. As with the training procedure, the testing process is recursive. With each recursion, the algorithm is called with a parameter, e, the example to be classified. The

classification begins at the root node. Based on the class label returned by the classifier, the classification continues to the next level of nodes. The choice of whether to traverse to the left or right node depends on the class label returned. The process continues until a leaf node is reached, at which time the label returned by the node is the final classification for the test example.

Algorithm 3 Generate_BinaryTree

Input: D = a structure file fragments training dataset
Output: File fragments BT classifier

Initialisation :
C := number of fragments classes
$d \subseteq D$
procedure GENERATE_BINARYTREE(D)
    if $C == 1$ **then**
        Return the node as a leaf node holding the corresponding class label
    else if $C == 2$ **then**
        Feature_selector ← Apply wrapper feature selection and create a new set of features, d, to distinguish between $c1$ and $c2$
        Binary_classifier ← Train a binary classifier on d
        New_node ← Feature_selector, Binary_classifier
        New_node.left ← Generate_BinaryTree($d1$)
        New_node.right ← Generate_BinaryTree($d2$)
    else if $C > 2$ **then**
        Cluster the data examples into two groups, $G1$, $G2$ using K-means
        Relabel the dataset examples to their corresponding groups, either $G1$ or $G2$
        Apply feature selection to create a new feature set, d, to distinguish between $G1$ and $G2$
        Binary_classifier ← Train a binary classifier on d
        New_node ← Feature_selector, Binary_classifier
        New_node.left ← Generate_BinaryTree($G1$)
        New_node.right ← Generate_BinaryTree($G2$)
    end if
end procedure

Algorithm 4 BT classifier

Input:

e ▷ e is unseen file fragment example

N ▷ N is the pointer to the current node in the hierarchy(root node at beginning)

Output: The predicted file type label

function BT CLASSIFIER(e,N)

    C= Classification result for e fragment using classifier at node N

    if C == 1 **then** return C

    else if C==2 **then** C ∈ N.rightClassGroup then return (BT Classifier(e,N.rightClassGroup))

    else return (BT Classifier(e,N.leftClassGroup))

    end if

end function

6.3.4 Directed Acyclic Graph (DAG)

Algorithm 5 shows in detail how DAGs are generated. D is the training dataset, and C is the number of class labels. The training set D includes all examples, where each example has a label that indicates what type of file it is. For our data, where there are 13 file types, k=C-1 and C=13 at the top of the hierarchy, and the nodes at bottom of the DAG tree have k=2. The set of size k class combinations (C_k) was calculated at each level of the tree. Using the 13 file types at the root of the hierarchy (i=0, where i represents the level number), the combination mechanism found all possible combinations of size 12. Moving down the tree, i increases by one, and consequently, the combination size decreases by one. This was repeated until the combination size reached two. At each level, we loop through this set, doing the following:

- From the training, we found the subset E_i that contained the combination $C_i \epsilon C_r$
- Used wrapper to select features
- Generated a classifier $G_i usingE_i$
- Added the classifier to the classifier dictionary

Once the DAG tree has been generated, the testing algorithm works as follows: Starting at the root node, the testing example will undergo the same processing stages as the training

data (e.g. feature selection). The classifier at the node will then be applied to the data and this will provide an initial classification label for the test data. The label assigned will determine which node the data is passed to in the next level of the tree (i.e. the left- or right-hand node). This process will continue down the tree until a leaf node is reached. The leaf node will then provide a binary classification, the result of which determines the final class label assigned to the test data.

Algorithm 5 Creation of a DAG tree for file fragments classification

Input: D ▷ D is a structure file fragments training dataset
Output: File fragments BT classifier

Initialisation : n = number of fragments classes
d $\subseteq$ D
C= Set of Classes
Start
k=n−1
procedure GENERATET_DAG(D)
    for each node in DAG tree **do**
        C_k^n = Combinations of size k in C

        for each element in C_k^n **do**
            create a subset (E_i) of training example that class labels are $C_i \in C_k$
            Apply wrapper feature selection to the training setE_i
            Generate a classifier M_i using E_i
            Add classifier to the classifier dictionary
        end for
        $k = k - 1$
        $n = n - 1$
    end for
end procedure

6.4 Results

6.4.1 Binary Tree (BT) Results

This section presents the BT hierarchy results for the classification of file fragments involving the K-means clustering technique. Figure 6.5 shows how our BT arranges file types in the hierarchy. Our BT successfully arranged file types into groups that appear to be meaningful

from a human perspective. The root node of our hierarchy separated the office files from all other types with an accuracy of 99%. Then, the classifiers on the next-level nodes were able to separate text files from a number of other files with 100.00% accuracy (i.e. image, video, PDF, and archive files). Within the text type group node, the algorithm separated HTML files from CSV, LOG and TXT with an accuracy of 100.00%. In addition, the algorithm separated GIF files from JPEG, PDF, PNG, WAV and ZIP with an accuracy of 100.00%. However, the model could not achieve further separation at some deeper hierarchy nodes. This could be because certain file types with similar binary content have been grouped together, which is one of the drawbacks of the BT hierarchy model. For example, DOC and PPT share similar byte-level representations and are therefore difficult to distinguish using this model. Our overall mean accuracy was 73.6% after training the BT on fragments of size 512-bytes.

As shown in Table 6.1, our macro-average F1 measure was 0.73. The highest accuracy was achieved for LOG files, CSV files, TXT files, and WAV files. The experimental results of the classification using the BT hierarchy are provided in Figure 6.9. The lowest accuracy was obtained using PPT, PDF, and PNG fragments, which were grouped into the same node. As the BT did not learn to distinguish these classes at earlier nodes in the tree, they continued to misclassify these classes further down the tree. Our BT model outperformed other comparable BT classifiers reported in previous work. In particular, Bhatt et al's BT model obtained an average classification accuracy of 67.78%, which was achieved using a traditional RF classifier. However, the precise training and testing split used in their experiments differed from those used here, which may explain some of the difference in accuracy [Bhat et al., 2021]. Generally, BT works better with simple file classes like TXT, CSV, etc., while it performs poorly with complex file classes such as PPT, ZIP, and PDF. Our evaluation strongly suggests that the BT approach is insufficient to solve the file fragment classification problem, especially for the challenging file types. These issues could be due to the fact that the K-means clustering algorithm groups similar file classes together, leading to successive

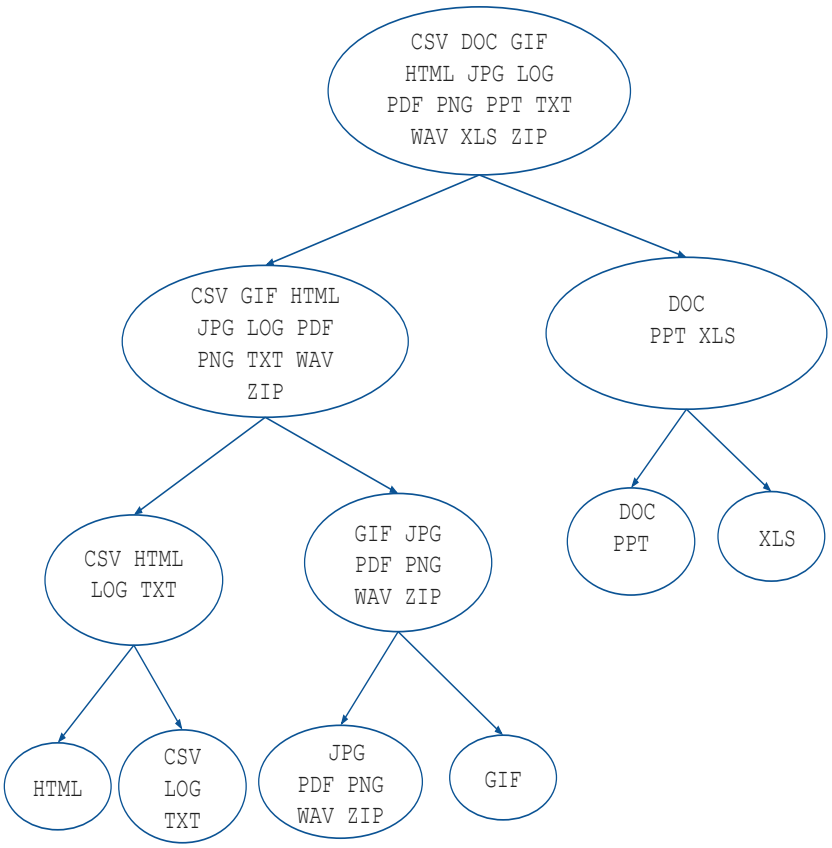

Figure 6.5: The BT hierarchy experiment outputs and how it arranges the types of files.

misclassifications further down the hierarchy. To overcome the problems associated with the BT approach, we proposed using the DAG method.

Figure 6.6 displays the confusion matrix of the BT Model. The high values on the diagonal of the matrix show that many of the classes are classified with a very high degree of accuracy. For example, CSV and LOG files are classified with near perfect accuracy. The values in the off-diagonal parts of the matrix show where confusions occurred. The most significant confusions are observed for PDF, PNG, PPT and ZIP files. This result is to be expected, as all of these file types are encoded using compression, meaning that much of their unique

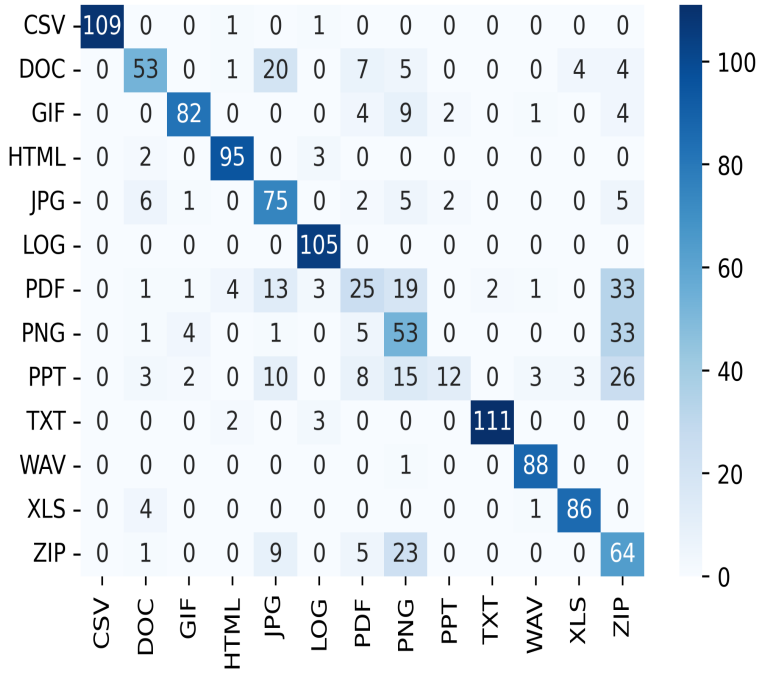

Figure 6.6: The confusion matrix for the classification of the files fragments using the BT Model.

Table 6.1: Classification results for the BT model.

File Type	F1 Measure	Precision	Recall	Accuracy(%)
CSV	0.99	1.00	0.98	98.20%
DOC	0.64	0.76	0.55	55.32%
GIF	0.85	0.92	0.79	79.41%
HTML	0.94	0.93	0.94	94.00%
JPEG	0.68	0.58	0.81	81.25%
LOG	0.96	0.92	1.00	100.00%
PDF	0.38	0.51	0.30	30.39%
PNG	0.46	0.41	0.54	53.61%
PPT	0.19	0.60	0.11	10.98%
TXT	0.97	0.98	0.97	96.55%
WAV	0.95	0.92	0.99	98.88%
XLS	0.94	0.93	0.95	94.51%
ZIP	0.48	0.39	0.64	63.73%
Mean	0.73	0.76	0.74	73.60%

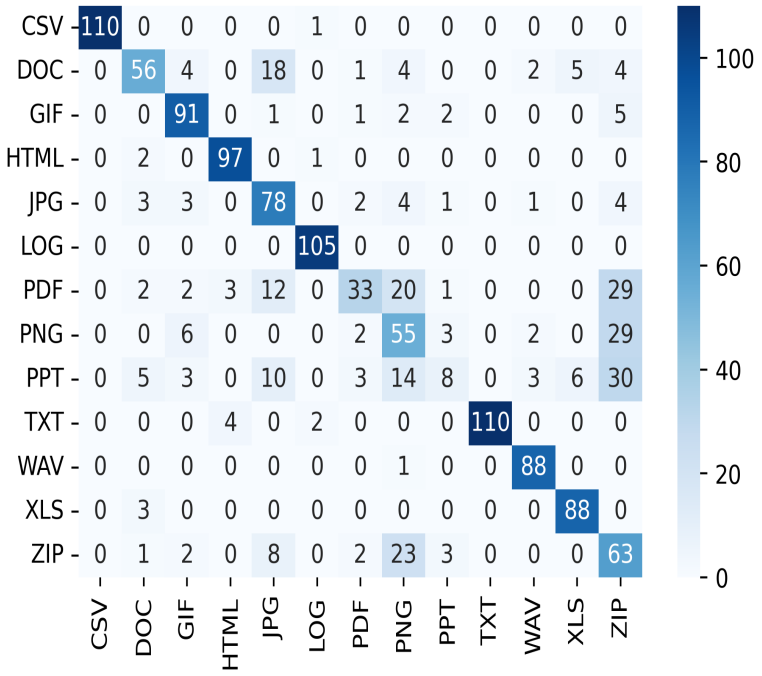

Figure 6.7: The confusion matrix for the classification of the files fragments using the DAG tree model.

byte information has been removed in order to reduce redundancy and file size. Therefore, some fragments from these files may look very similar when observed at the byte level.

6.4.2 Directed Acyclic Graph (DAG) Results

This section presents the DAG hierarchy results for file fragment classification. Similar to BT training, DAG classifier training used 60% of the dataset, while testing used the remaining 40%. Using a combination of techniques, the training started at the tree's root node, and the labels were distributed among the nodes using grouping. At the root node, input classes were grouped into sets of the same size. In this study, the fourteen file classes were divided into fourteen subsets, each consisting of thirteen classes. As the algorithm progresses down the tree, the subset sizes decreased by one at every level. This process ended when the combination size reached two. 8,178 classifiers were trained to create this DAG model. Using the 512-byte fragment size, we achieved an overall mean accuracy of 75.16%. According to Table 6.2, our macro-averaged F1-measure was 0.73. The highest accuracies were achieved for LOG, CSV, TXT, HTML, WAV and XLS files. DOC, GIF, PNG and PDF files have shown some improvement over the BT approach. However, the accuracies of the classification of ZIP and PPT have declined by almost 2% for. ZIP files and 1.2% for PPT files.

Figure 6.7 illustrates the DAG classification confusion matrix. The most difficult file type to classify is PPT files, which are misclassified frequently as ZIP, PNG, JPEG and less frequently as DOC, GIF, or PDF. The reason for this is because all of these file types involve a form of compression. The types of compression used are either lossy or lossless. For example, PNG and ZIP files both use lossless compression to reduce file size, which may explain why these file types are frequently confused. The DOC format uses the DEFLATE compression format, which is also lossless, and again this is one of the file types most often confused with PPT files.

Table 6.2: Classification results for DAG tree model.

File Type	F1 Measure	Precision	Recall	Accuracy(%)
CSV	1.00	1.00	0.99	99.10%
DOC	0.67	0.78	0.60	59.57%
GIF	0.85	0.82	0.89	89.22%
HTML	0.95	0.93	0.97	97.00%
JPEG	0.70	0.61	0.81	81.25%
LOG	0.98	0.96	1.00	100.00%
PDF	0.45	0.75	0.32	32.35%
PNG	0.50	0.45	0.57	56.70%
PPT	0.16	0.44	0.10	9.76%
TXT	0.97	1.00	0.95	94.83%
WAV	0.95	0.92	0.99	98.88%
XLS	0.93	0.89	0.97	96.70%
ZIP	0.47	0.38	0.62	61.76%
Mean	0.73	0.76	0.74	75.16%

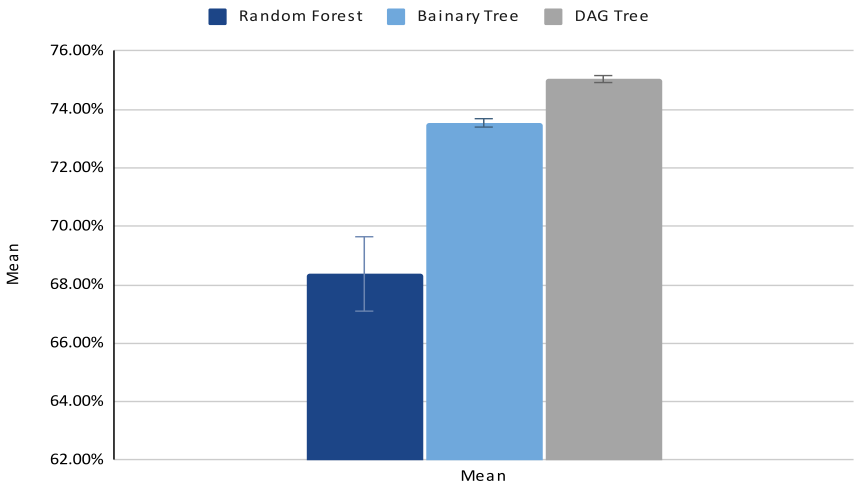

Figure 6.8: The mean accuracy of random forests, binary trees, and DAG trees. The error bars represent +/− 1 standard deviation from the mean.

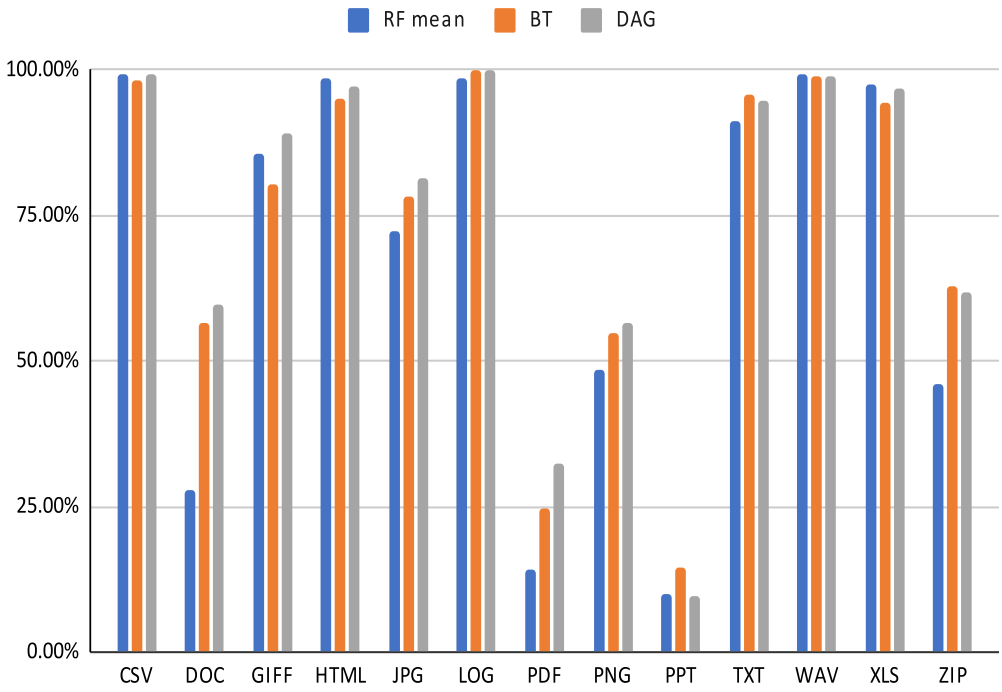

Figure 6.9: Accuracies of random forests, binary trees, and directed acyclic graph models.

JPEG files have achieved a fairly good accuracy of 81.25%, with a small number of examples being misclassified as ZIP, PNG, PDF and DOC. JPEG uses lossy compression rather than lossless compression, which may be why the accuracy of JPEG identification is higher than for other compressed file types. The small number of misclassified JPEGs may be because PDF and DOC files can actually embed JPEG files within them, making some fragments in those classes appear more similar to JPEGs. Although ZIP files are one of most difficult file types, our DAG was able to achieve an accuracy of 61.76%.

6.4.3 Comparison with Baseline Work

Figure 6.8 shows the mean classification accuracy for the thirteen file types using random forest classification, BT classification, and DAG classification. Compared to the baseline classifier (random forest), both hierarchy models achieved higher accuracy. The error bars

for the hierarchical models show that there is less variation in the average accuracy across five experimental repeats, compared to the random forest model, which shows a higher variation in mean accuracy. This means that the classification is more consistent across separate runs for the hierarchical models than with the random forest. The DAG model achieved the highest accuracy at 75.16%. However, DAG does not outperform BT for all 13 types of files, as shown in Figure 6.9. Some file types showed better accuracy with BT, whereas others produced higher accuracies with DAG. The BT and DAG hierarchy models were most beneficial to the DOC, JPEG, PDF, PNG and ZIP file types. For example, the accuracy of random forest, BT, and DAG for the DOC file was 27.84%, 55.32%, and 59.57%, respectively. As expected, due to these types involving a form of file compression, most of the incorrectly classified PNG file fragments were misclassified as ZIP or GIF file fragments. All three of these file types are typically compressed using a lossless technique. In our analysis, we noticed that files compressed using the same compression method were typically confused with each other.

6.5 Summary

File fragment classification is a crucial task for security fields. This work explored two hierarchical approaches for file fragment classification, called BT and DAG tree. We used a hybrid feature selection technique. We hypothesised that specialised feature selection at each node supports the node classifier to better distinguish between node classes than is possible using general features for all file types.

In this work, we used a dataset comprising 13 file types, with 300 fragments for each file, and using a fragment size of 512 bytes. Compared to previous studies, our models provided comparable classification accuracies for most file types. The mean accuracy of the DAG tree is higher than obtained using BT and random forest. However, that is not the case for each file type's classification accuracy. The container files, such as PDF, are misclassified with files that might be included within them, such as image files (e.g. PNG, JPEG and

GIF files). On the other hand, some image files might be misclassified with ZIP, PDF and PPT, either because they have used the same compression method, or again the fragments from these files might include images.

BT was successful in arranging files in a logical hierarchy model. However, both the BT and DAG tree results show that the feature-based approach for file fragment classification has perhaps reached the limit of its capabilities using these techniques. Following this, in the next chapter, we will look at a different approach for file fragment classification. Specifically, we will explore the application of Deep Neural Networks (DNNs) to this problem. This approach does not require the use of feature engineering and allows us to approach the problem in a more abstract way, involving unstructured data.

7 deep learning For File Fragments Classification

7.1 Introduction

In earlier chapters, we focused on finding the best statistical features that could differentiate between file types. However, we observed that the extracted features failed to distinguish between some file types, specifically complex files such as ZIP and PPT. The data used for this research consists of file fragments, each containing 512 bytes. We could treat this data as a text file, in which each sentence (fragment) consists of 512 words (bytes), and each word is one byte long. By considering our data in this way, we can take advantage of Natural Language Processing (NLP) techniques to solve the classification problem for file fragments. The advantage of this approach is that we do not have to worry about extracting the most representative features. Instead, we can use unstructured fragment data directly. The main focus of this chapter is to further investigate using NLP deep learning methods to solve file fragment classification. This chapter is divided into five sections. Section 7.2 illustrates the procedure framework of this chapter. Section 7.3 explains the strategy we used to test the effectiveness of this chapter's methodology, including the experimental design adopted and the data used to evaluate the methodology. Section 7.4 explains the result of the proposed methods. Finally, Section 7.5 summarises this chapter.

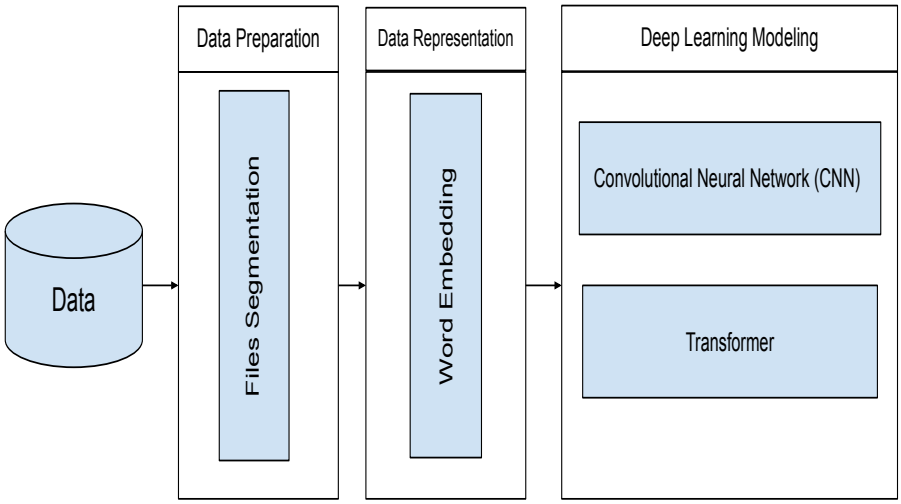

Figure 7.1: Deep learning modelling consists of five components: Data, data preparation, data representation and modelling.

7.2 Deep learning For File Fragment Classification Procedure

Figure 7.1 illustrates the procedure framework of this chapter, which consists of five components: data, data preparation, data representation, and modelling. Each component will be discussed in more detail in the following sections.

Data Preparation

As explained in Chapter 4, data preparation steps include loading the data, cleaning the data to remove any unnecessary information and separating the data into training and test sets. As in the previous chapters, we downloaded 13 different file types from the Granfinkel corpus [Garfinkel et al., 2009]. Two hundred files from each file type were randomly chosen.

Subsequently, the files were cleaned of their metadata and fragmented into 512-byte parts using a fragmentation method. The fragments were then stored in a database. Unlike our previous dataset described in Chapters 5 and 6, in this chapter we use a larger number of file fragments in our dataset. The reason for this is that deep learning approaches are well known to produce better results with larger quantities of training data [Chen and Lin, 2014]. A detailed explanation of the use of larger datasets in deep learning is provided in Appendix A.

7.2.1 Data Representation

In Natural Language Processing (NLP), data representation is the numerical representation of text data used in Machine Learning algorithms. Several methods can be used to numerically represent text. The most common method is a bag-of-words, one_hot vector and a Word2Vec. Bag-of-words represent the text as a collection of words without regard to the order in which they appear in the text, and only their frequency is retained. As a result, representing text as a bag is not sufficient when there is potential information in word order. The second common method is one_hot encoding where text is represented as binary vectors that map each text element to integer values. Then, each integer value is represented by a binary vector consisting of all zero values except the index of the integer, which is indicated by a one. However, there are two main disadvantages of this approach. Firstly, the storage of this data is inefficient and therefore a large amount of memory is required to process this data. Additionally, this approach does not capture the relationship between words, as it does not provide information about the context in which they occur. Lastly, Word2Vec, sometimes called word embedding, is a numerical representation of text that represents words with similar meanings using real-valued vectors. Typically, each word (token) in a text is represented as a multi-dimensional vector. In the NLP case, the weight matrix is moved horizontally across the sentence one step at a time, capturing specific numbers of words at a time. Each weight matrix is referred to as a filter, and each filter consists of an activation function, similar to that of a feedforward neural network. In

deep neural networks, the activation function, ReLU (Rectified Linear Unit), is mostly used due to its mathematical properties. According to the general logic of the filters, each filter is capable of detecting different features of text, and the deeper the filter, the more likely it is to identify more complex features. As an example, a network's very first filters are likely to detect simple features, while its very last filters might be able to detect specific types of text. Each of these filters has a weight, which is learnt by back-propagation during training, so generation of the filters is data-driven rather than hard-coded. Interestingly, Word2Vec embeddings allow for exploring interesting mathematical relationships among words. In this sense, this approach is an effective way to represent file fragments, where each byte of the fragment is represented by a real-valued vector in a high-dimensional space. Similarly to words in the text, bytes that have a similar context in the vector space will also have a similar representation in their file content. The deep learning toolbox, Keras, offers an embedding layer that can be used for neural networks on text data that allows for a more expressive representation of fragments than more traditional methods, such as bag-of-words, where the relationships between bytes (or tokens) are ignored, or bigram and trigram approaches, where the context is rigidly defined. The real-valued vector representation for words can be learnt during the training procedure of a neural network. The input data must be integer encoded, which means that every byte is replaced by a unique number ranging from 0 to 255. The embedding layer is initialised with random weights and will learn an embedding for all of the bytes (words) in the training set. Therefore, it can be used as a component of a deep learning model in which the embedding is also learnt together with the model itself. Embedding layers are primarily employed for their efficiency in converting sparse binary features (1-hot encoded-words in the dictionary) into compact real-valued representations. As a result, we were able to reduce a single byte (256 binary features in 1-hot encoding) to 16-64 real-valued features. The embedding layer is the first hidden layer within a network. Three arguments are required to generate the layer:

- input_dim: The size of the byte values in the fragment's data. The bytes values size is 256, and their value ranges from 0-to-255.

- output_dim: Describes the size of the vector space in which bytes will be embedded. Defining the size of the output vectors from this layer for each byte. The size may be 32 or 100, for example.
- input_length: This corresponds to the length of any input sequences in a Keras model. In our data, the input dimension is 512, which is the fragment size.

7.2.2 Modelling

In terms of deep learning classifiers, we chose to use the most common, which is the Convolutional Neural Network (CNN), and a recently developed approach called transformer. In the following sections, we will discuss these two types of deep learning neural networks and how they can be applied to our file fragment classification problem.

CNN

Our CNN architecture is based on the approach described by Collobert et al. [2011]. In their work, they present a single end-to-end neural network model with convolutional and pooling layers for use across a range of fundamental natural language processing problems. Our CNN consists of the following layers:

1. Embedding layer: To represent fragments as vectors.
2. Convolution layer: To perform the convolution operation, CNN has a convolution layer with several filters.
3. Rectified Linear Unit (ReLU): ReLU layers are used in CNNs for performing operations on elements. They produce rectified feature maps.
4. Pooling layer: Next, the rectified feature map is fed into a pooling layer. Pooling is a way to reduce the size of the feature map by downsampling. A pooling layer flattens the resulting two-dimensional arrays from the pooled feature map into a single, long, continuous, linear vector.

5. Fully connected layer: When the pooling layer's flattened matrix is fed as an input, it forms a fully connected layer that classifies and identifies the file fragments.

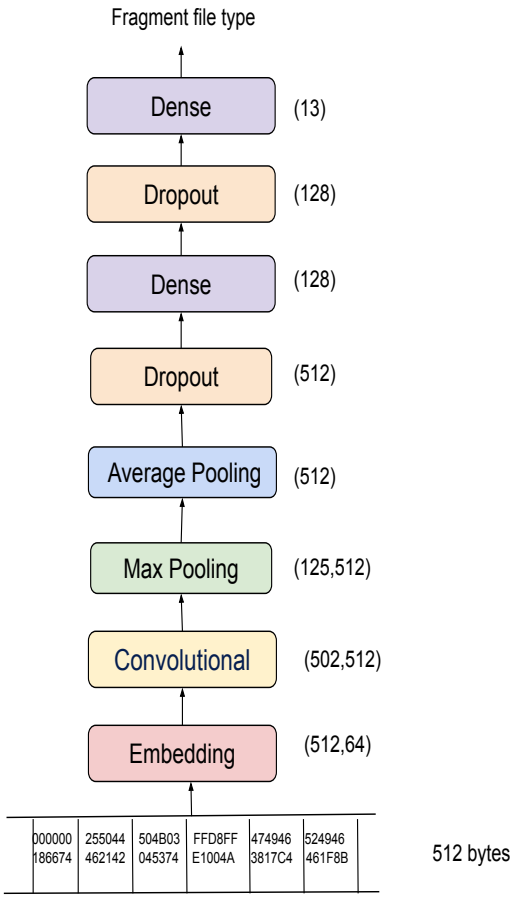

Figure 7.2: Illustration of the proposed CNN architecture.

Figure 7.2 shows that our models are fed raw byte blocks as input in a similar manner to NLP [Mikolov et al., 2013]. Our work treats file fragments as a sequence of bytes and aims to capture the characteristic patterns of byte transitions that are indicative of many file types.

Transformer

Transformers typically include an encoder and a decoder. In our work, we only require the encoder process. Figure 7.3 illustrates the transformer we have built. As described

before, the encoder includes two main layers, which are multi-head attention and the feed-forward neural network. In our work, we feed fragment data as a sequence of bytes. We then use a positional encoding method to add byte positions to the sequence data. Three values are calculated by the positional encoding, and these are the Key (K), Query (Q), and Values (V). Each vector value multiplies with a weighted matrix which is learnt through the training process. Query and key vectors are used to calculate "Score". Then the results go through soft max operation to normalise the scores. Then the output is sent to the feedforward neural network. Then it normalised again before doing pooling and dropout. The result would be linearised and normalised to get the file classes.

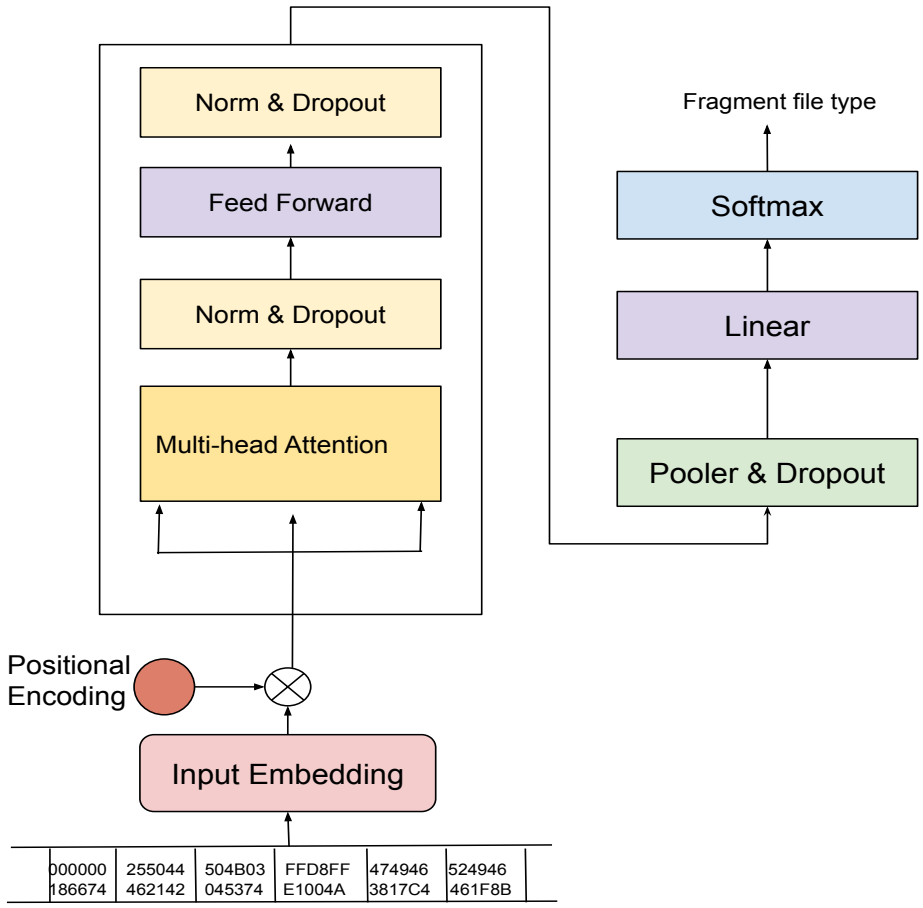

Figure 7.3: Illustration of the proposed transformer architecture.

7.3 Experiments

The following sub-sections detail the two experiments undertaken in this work and their results. The dataset we use in both of these experiments is described in the Data Preparation section of this chapter. To ensure that the dataset was balanced in terms of the number of examples of each type of file, we randomly selected 102,400 fragments from each type. This resulted in a total of 1,331,200 examples, which were randomly shuffled and distributed equally between the training (80%), validation (10%), and hold-out testing (10%) groups.

7.3.1 CNN Experiment

Design Space and Hyperparameter Optimisation

The general network architecture is the same as described in Section 7.2.2. In this work, we train our networks to 50 epochs, using a batch size of 32. We use the Adam optimiser with a learning rate of 0.3, and the categorical cross-entropy loss function. The network includes a drop-out layer, set to 0.3, to avoid overfitting. However, after hand-tuning the general architecture of the CNN, we defined five hyperparameters which we optimised manually using the trail-error method. For each hyperparameter, we chose a range of reasonable values to test:

1. Embedding size: Dimensionality of trainable, real values in embedding space 16, 32, 64, 128
2. Convolution kernel (filter): The size of the convolutional kernel (the same for all layers) 16, 32, 64, 128, 256, 512
3. Stride size: The amount of data to be skipped between each kernel operation 1, 3, 11, 27, 35
4. Dense units: The number of units in the hidden dense layer 64, 128, 256

5. Block size: The number of CNN layers 1, 2, 3

Figure 7.4, 7.5, 7.6, 7.8 and 7.7 the impact of different hyperparameters on validation accuracy for 512 bytes file fragment input. Table 7.1 shows the design of the CNN network and the default values of the hyperparameters. For each of the 5 hyperparameters, we vary a target parameter using the ranges described previously, whilst keeping the remaining 4 parameters fixed at the values shown in Table 7.1.

Table 7.1: Different CNN hyperparameters for layers:
[E]Embedding(<embedding vector length>)
[C1D]Convolution(<filtersize,stride>)
[MP]max-pooling(<size>)
[AP]average-pooling
[F] fully connected(<#units>)
[D] dropout(<dropout value >)
[F]flatten.

Row Number	**Architecture**
1	E(x)-C1D(128,27)-MP(4)-AP-D(0.5)-F(256)-D(0.5)-F(13)
2	E(64)-C1D(x,27)-MP(4)-AP-D(0.5)-F(256)-D(0.5)-F(13)
3	E(64)-C1D(128,x)-MP(4)-AP-D(0.5)-F(256)-D(0.5)-F(13)
4	E(64)-C1D(128,27)-MP(4)-AP-D(0.5)-F(x)-D(0.5)-F(13)
5	E(64)-C1D(128,27)-MP(4)-AP-D(0.5)-F(256)-D(0.5)-F(13)

Figure 7.4 shows the result of testing different values for the size of the embedding vector. The results show that 64 is the optimal size for the embedding vector, where an accuracy of 85.04% is achieved. The lowest accuracy was obtained using an embedding size of 16, giving an accuracy of 83.67%. The accuracy increases with embedding size, up to a size of 64. Above 64, the accuracy begins to fall, with an accuracy of 84.62% when using an embedding

size of 128. Therefore, an embedding size of 64 is sufficient to represent a 512-byte fragment vector.

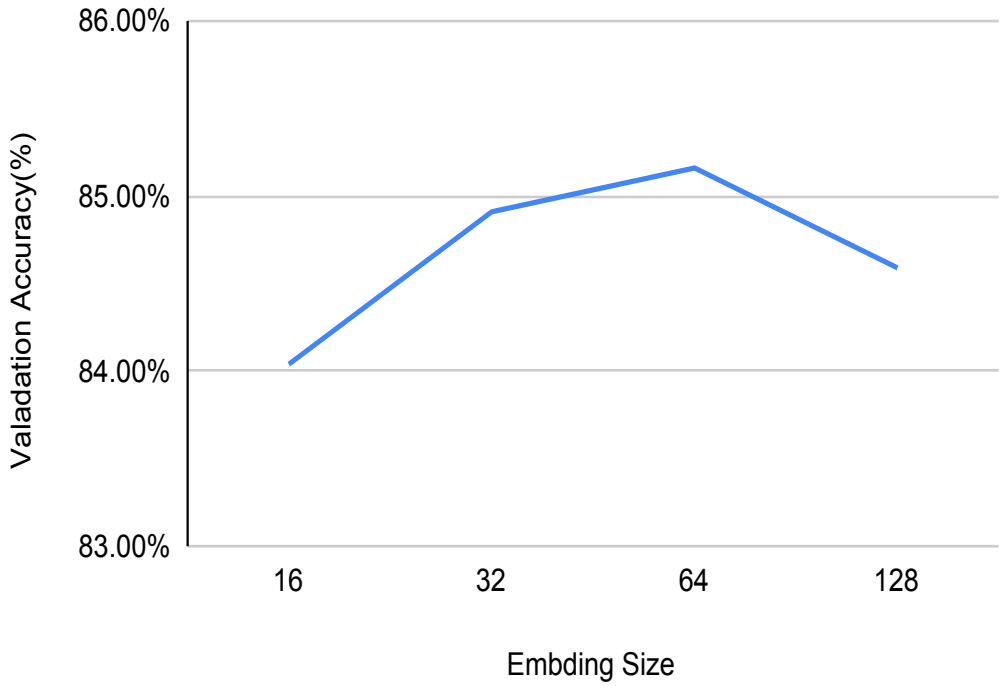

Figure 7.4: Illustration of the effect of the different embedding sizes on CNN validation accuracy.

The convolutional kernel (or filter) is the number of sequences taken into account at each point of the convolutional process. As before, we keep the remaining four hyperparameters fixed when varying the value of the convolutional kernel parameter. The second row of Table 7.1 shows the fixed values used for the remaining four parameters, with 'x' denoting the filter size. Figure 7.5 illustrates the validation accuracies obtained for different filter sizes. The results show that the validation accuracy is positively correlated with an increase in the filter size. However, the accuracy starts to level off at about 128, and peaks at 512, with a maximum accuracy of 86.55%. Although higher accuracies may be achieved with even larger filter sizes, we could not increase the size of the filters beyond the input size in our experiments, which is 512 bytes (words).

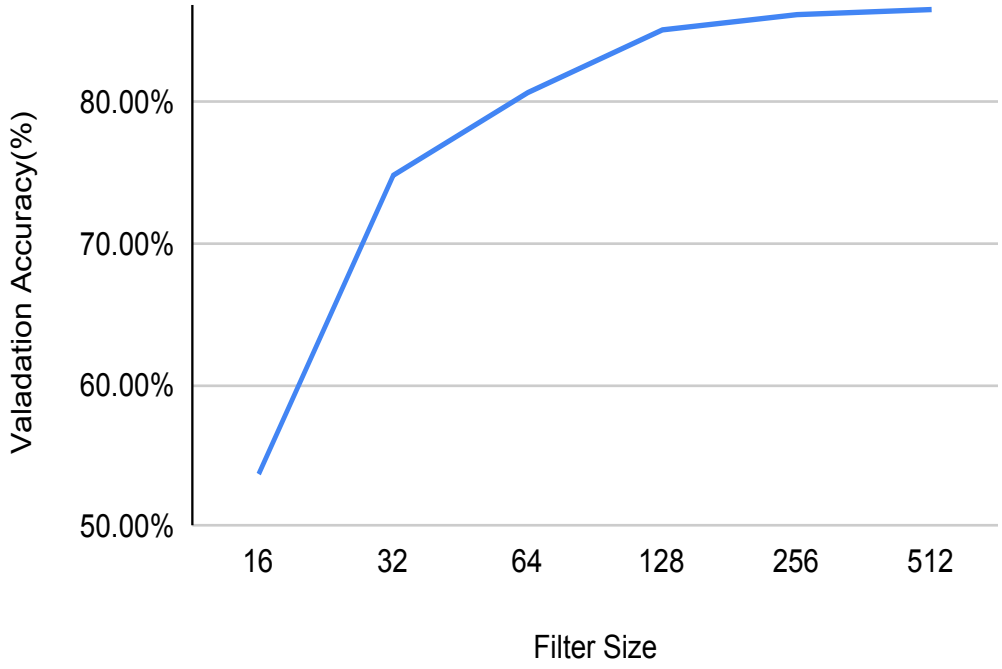

Figure 7.5: Illustration of the effect of the different filter sizes on CNN validation accuracy.

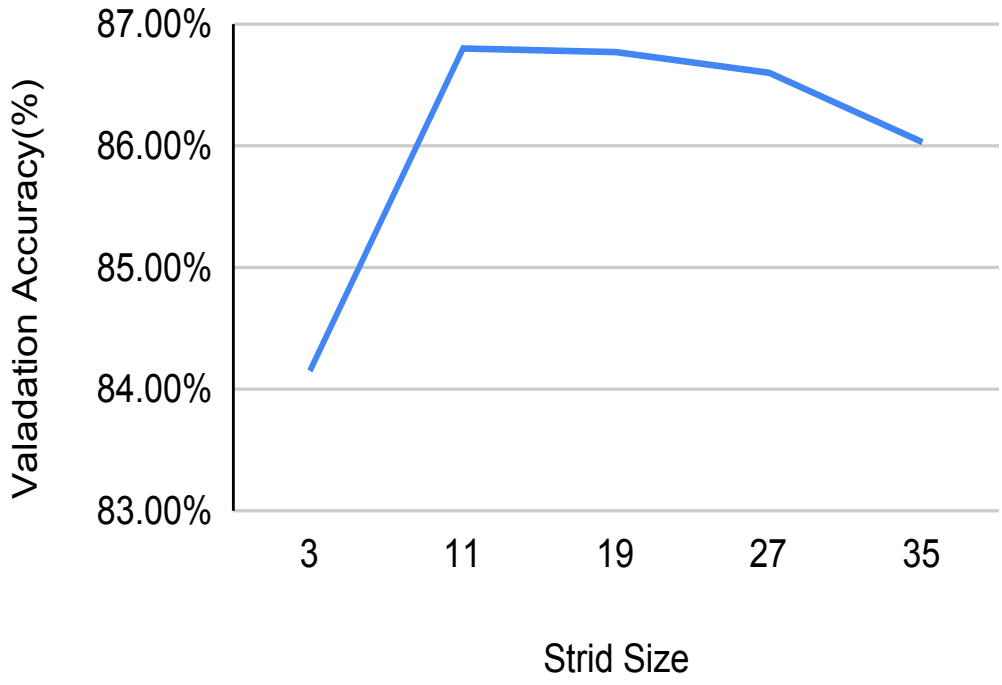

Figure 7.6: Illustration of the effect of the different stride sizes on CNN validation accuracy.

The stride value is the number of bytes shifted over the input matrix by the convolutional kernel. The filters are moved one byte at a time when the stride is 1. For a stride of 2 bytes, the filters are shifted by 2 bytes at a time. We tried five different values for the stride, which are 3, 11, 18, 27, 35, and fixed the other four hyperparameters as shown in row 3 of Table 7.1. Figure 7.6 shows the results of varying the stride value. The lowest accuracy was obtained using a stride of 3, giving an accuracy of 84.15%. At a stride of 11, the accuracy increased to 86.80%. Beyond this value, the accuracy started to decrease again, suggesting that 11 is the optimal value for this hyperparameter.

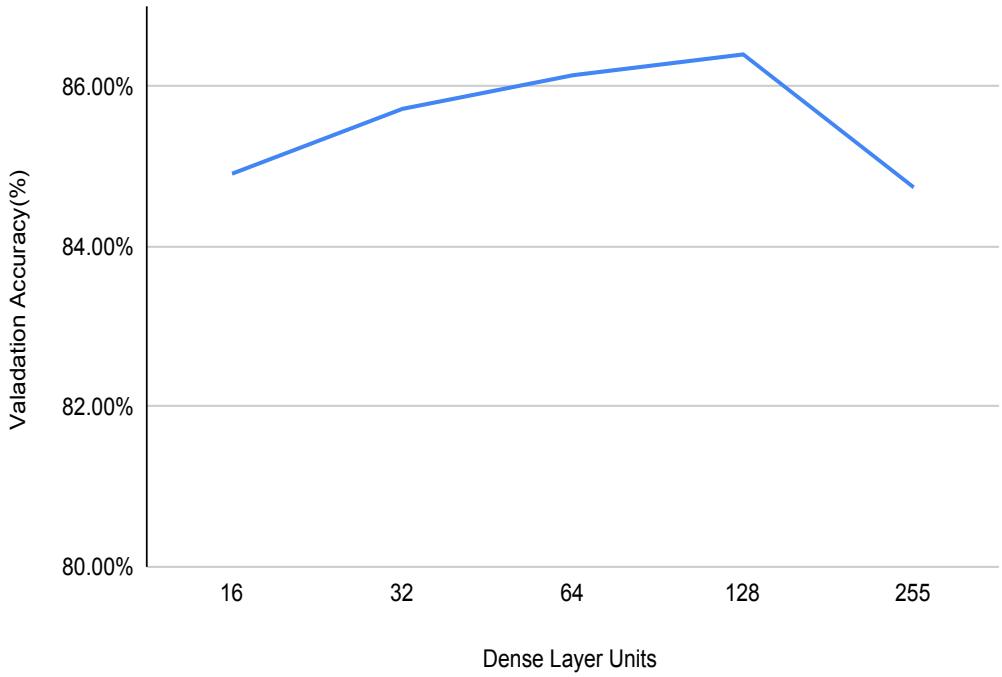

Figure 7.7: Illustration of the effect of the different number of dense layer units on CNN validation accuracy.

The dense size refers to the number of units contained within the dense layer of our network. We vary the number of units as follows: 16, 32, 64, 128, 255. All other hyperparameters were fixed, according to the values shown in row 4 of Table 7.1. Figure 7.7 shows how the validation accuracy changes as the number of dense units increases. The results show that the validation accuracy increases with the number of units, up to a value of 128 units. Beyond this, the accuracy starts to drop. The maximum accuracy of 86. 4% is achieved at 128 units, and therefore this is the optimal value for the number of dense units in our work.

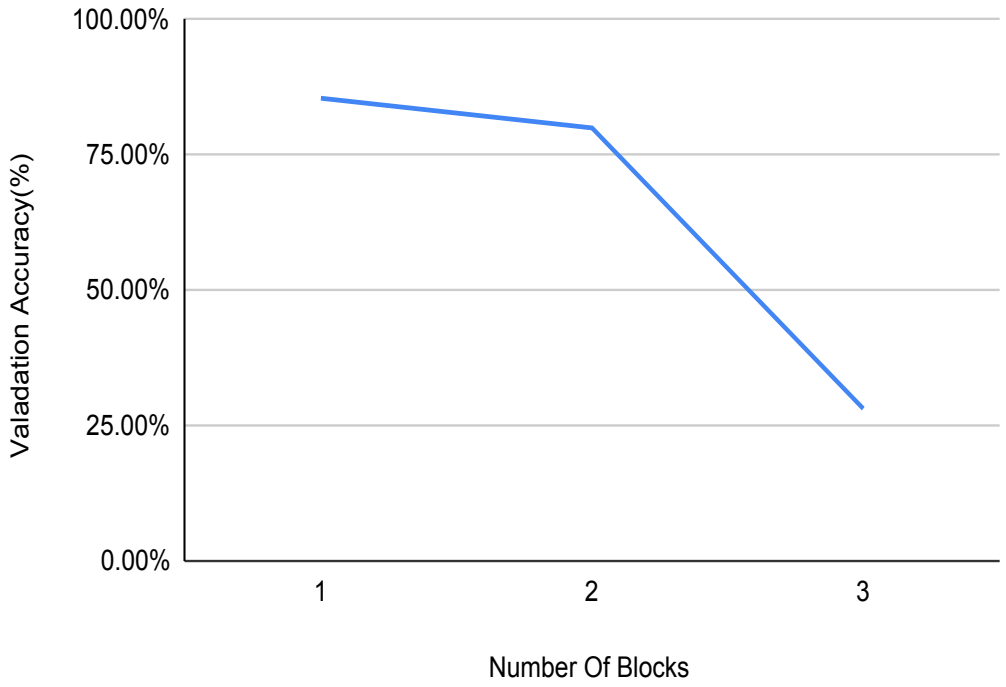

Figure 7.8: Illustration of the effect of the different number of blocks on CNN validation accuracy.

The number of blocks in a CNN can be varied. This is commonly known as the depth of CNN. In our work, we explored using 3 different CNN depths: 1, 2, 3. We fixed all four remaining parameters, as shown in row 5 of Table 7.1. Figure 7.8 illustrates the effect on validation accuracy of changing the number of CNN blocks. One layer of CNN was enough to achieve good accuracy for the validation set. Increasing the number of CNN blocks beyond one did not give any improvements to the accuracy, instead showing a sharp decline in accuracy beyond 2 blocks, dropping to 28.03%.

7.3.2 Transformer Experiment

Design Space and Hyperparameter Optimisation

The general network architecture is the same as described in Section 7.2.2. For this network, we train to 50 epochs, using a batch size of 32. We use the Adam optimiser with a learning

rate of 0.3, and the categorical cross-entropy loss function. The network includes a drop-out layer, set to 0.3, to avoid over-fitting. However, after hand tuning the general architecture of the transformer, we defined five hyperparameters which we optimised manually using the trail-error method. For each hyperparameter, we chose a range of reasonable values to explore:

1. Embedding size: Dimensionality of trainable real values in embedding space 16, 32, 64, 128
2. Feedforward dimension: The number of units in hidden layers in the feedforward network inside the transformer encoder 16, 32, 64, 128, 256, 512, 1024, 2048
3. Number of attention heads: The number of head layers 10, 20, 30

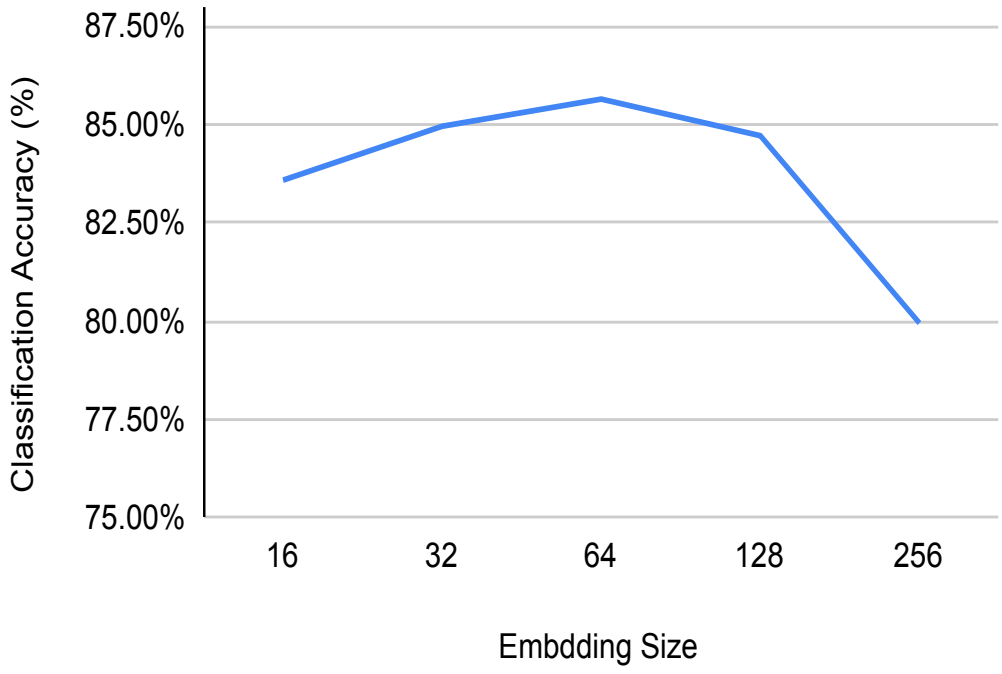

Figure 7.9: Illustration of the effect of the different embedding sizes on rransformer validation accuracy.

Figure 7.9 illustrates the effect of the different embedding sizes on the transformer validation accuracy. The validation accuracy increases with Embedding size. An accuracy of 83.58% is obtained with a size of 16, and the accuracy peaks at 85.64% with an embedding size of 64. Beyond 64, the validation accuracy starts to fall again, suggesting that 64 is the optimal embedding size for our work.

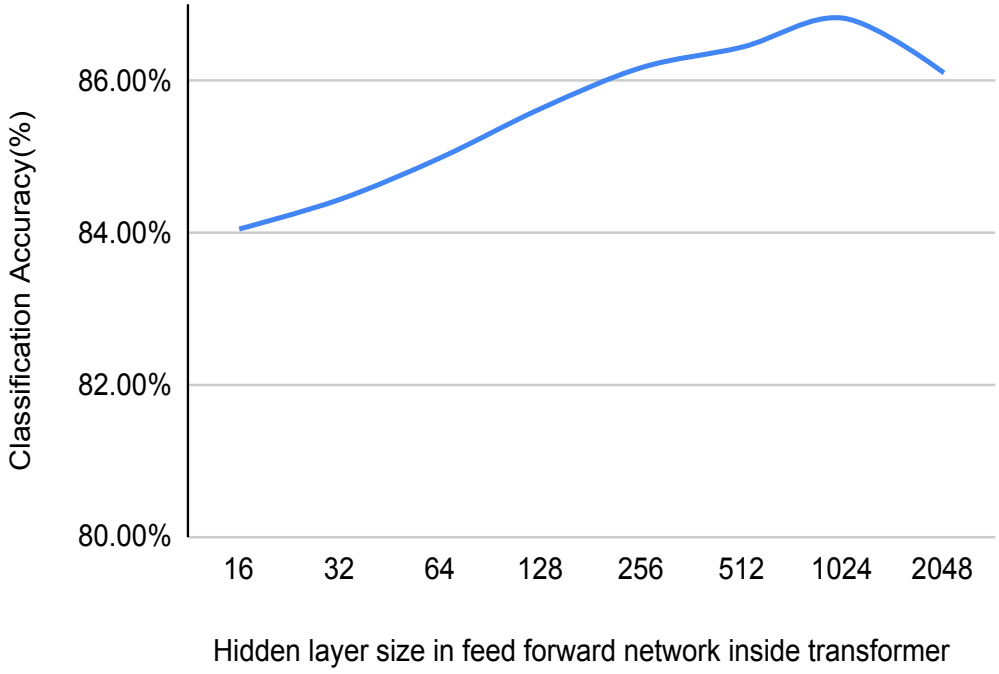

Figure 7.10: Illustration of the effect of the different hidden layer sizes on transformer validation accuracy.

Figure 7.10 shows the effect of the different hidden layer units on transformer validation accuracy. The validation accuracy increases with hidden layer size, starting at an accuracy of 84.05% with a size of 16. The accuracy continues to increase with the hidden layer size, although the rate of increase appears to slow beyond a size of 256. The highest validation accuracy of 86.83% was achieved using a hidden layer size of 1024. Increasing the layer size to 2048 gave a reduction in classification accuracy, suggesting that the optimal value is a layer size of 1024. For this reason, we used a hidden layer size of 1024 in our work.

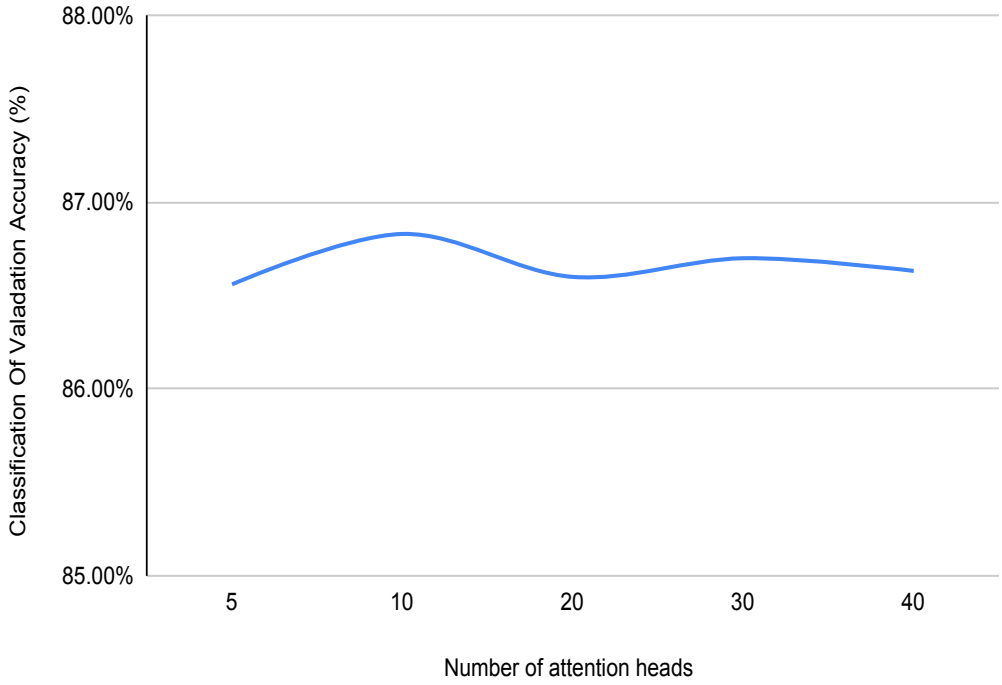

Figure 7.11: Illustration of the effect of the different number of attention heads on transformer validation accuracy.

Figure 7.11 shows the effect of the different number of attention heads on transformer validation accuracy. The highest accuracy of 86.83% was obtained using 10 attention heads. The accuracy decreased to 86.6% when the number of attention heads increased to 20, before increasing again when using 30 attention heads. However, the increase observed at 30 attention heads was still lower than the accuracy produced at 10 heads, and so a value of 10 attention heads was selected for our work.

7.4 Results

7.4.1 CNN Results

In the previous experimental tuning of the hyperparameters, we determined the best values of each hyperparameter, as shown in Table 7.2. Next, in this section, we present the results

Table 7.2: Classification results for the CNN experiment. Classification performance is shown separately for the 13 file types, as well as a combined mean across all file types. The F1 measure, precision, recall and classification accuracy are the performance metrics presented.

File type	F1	precision	recall	accuracy
JPEG	0.874	0.808	0.953	95.28%
GIF	0.939	0.964	0.915	91.50%
PNG	0.825	0.863	0.790	78.99%
ZIP	0.603	0.449	0.916	91.61%
DOC	0.921	0.967	0.878	87.83%
PPT	0.632	0.778	0.532	53.25%
XLS	0.996	0.999	0.994	99.36%
PDF	0.409	0.774	0.278	27.96%
TXT	0.979	0.992	0.966	96.65%
HTML	0.994	0.999	0.990	98.95%
LOG	1.000	1.000	1.000	99.98%
CSV	0.999	1.000	0.998	99.75%
WAV	1.000	1.000	1.000	99.95%
Mean	0.859	0.892	0.862	86.23%

of our experimental evaluation using these optimal parameters. Table 7.2 presents the F1, precision, recall and accuracy of each file type when using the CNN with test data. The highest accuracies were obtained by XLS, LOG, CSV and WAV with the accuracy close to 100% for these classes. PDF was the worst performing file type, with an accuracy of around 28%. However, the ZIP file achieved fairly good classification accuracy despite being one of the most difficult file types to identify. For PPT, an accuracy of 10.98% was achieved using the previous feature-based approach, but using the CNN improved the classification of PPT files to 53%. Similarly, the accuracies of JPEG, GIF, PNG and DOC improved compared to the results obtained using the DAG feature-based approach (Refer to Chapter 5). The accuracies of these four classes increased from 81%, 89%, 57% and 60% to 95%, 91%, 78% and 88%, respectively. The mean accuracy across all file types was 86.23%, which represents a very good level of accuracy, well above the levels of chance.

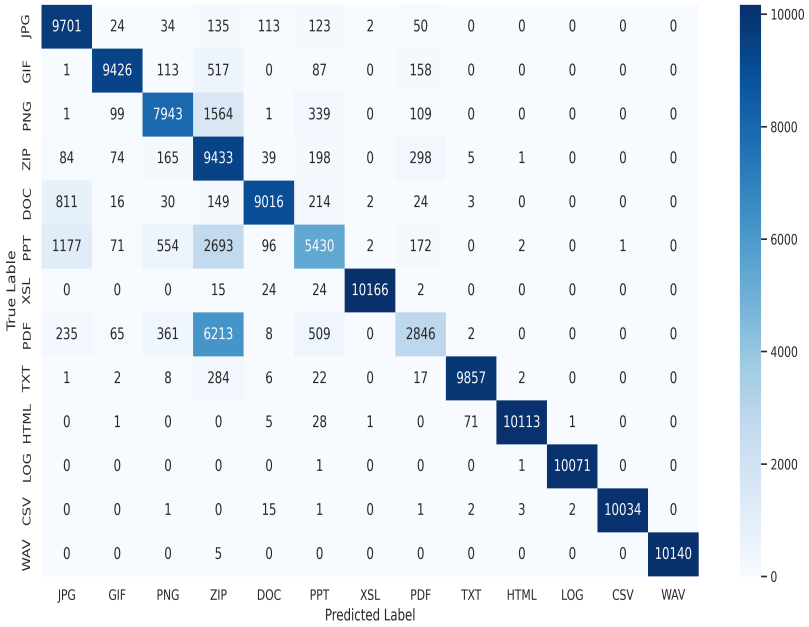

Figure 7.12: The confusion matrix of file fragments classification using CNN.

Figure 7.12 shows a confusion matrix for this experiment. Generally, the confusion matrix contains the highest values along its diagonal, which means that many classes have been successfully detected. However, the off-diagonal values in the matrix represent confusions in the classification. For example, the WAV file is mostly classified correctly, except for 5 instances, in which it was confused with the ZIP file type. By contrast, although the PDF file is correctly classified on 2846 occasions, the matrix shows that it is commonly confused with the ZIP file type, and less frequently with JPEG, PNG and PPT file types. PPT and PNG are also confused with the ZIP file type, although less frequently than for PDF files. The Microsoft Office file types DOC and PPT are sometimes confused with JPEG file fragments.

7.4.2 Transformer Results

Using the best hyperparameters determined in Section 6.3.2, in this section, we will present the results of file fragment classification experiments using the transformer technique. Table 7.3 presents the F1, precision, recall and accuracy of each file type when using the transformer with test data. The highest accuracies were obtained by XSL, LOG, CSV and WAV, all of which had accuracies of 100%. The lowest accuracy was obtained for PPT, at 50.32%, with ZIP and PDF also achieving relatively low accuracies of 64.36% and 50.65%, respectively.

The mean accuracy across all file types was 86%, which is the same accuracy achieved by the CNN approach, although the exact distribution of performance differed between file types. Specifically, although some of the file types performed slightly better in the CNN approach (e.g., the accuracy of PPT reduced slightly from 53.25% to 50.32% using transformer), generally the results were more consistent across the thirteen file types. For example, the accuracy of the PDF file type increased from 27. 96% to 50. 65%. Additionally, the results across the different file types appeared more stable when exploring values for the hyperparameters, when compared to the same process for the CNN approach.

Table 7.3: Classification results for the transformer experiment. Classification performance is shown separately for the 13 file types, as well as a combined mean across all file types. The F1 measure, precision, recall and classification accuracy are the performance metrics presented.

file type	F1	precision	recall	accuracy
JPEG	0.90	0.88	0.92	91.98%
GIF	0.97	0.96	0.97	96.86%
PNG	0.72	0.66	0.79	78.53%
ZIP	0.56	0.49	0.64	64.36%
DOC	0.91	0.90	0.93	93.07%
PPT	0.62	0.79	0.50	50.32%
XLS	1.00	1.00	0.99	99.48%
PDF	0.56	0.62	0.51	50.65%
TXT	0.98	0.99	0.96	95.91%
HTML	0.99	0.99	0.99	98.87%
LOG	1.00	1.00	1.00	99.97%
CSV	1.00	1.00	1.00	99.86%
WAV	1.00	1.00	1.00	99.99%
Mean	0.86	0.87	0.86	86.14%

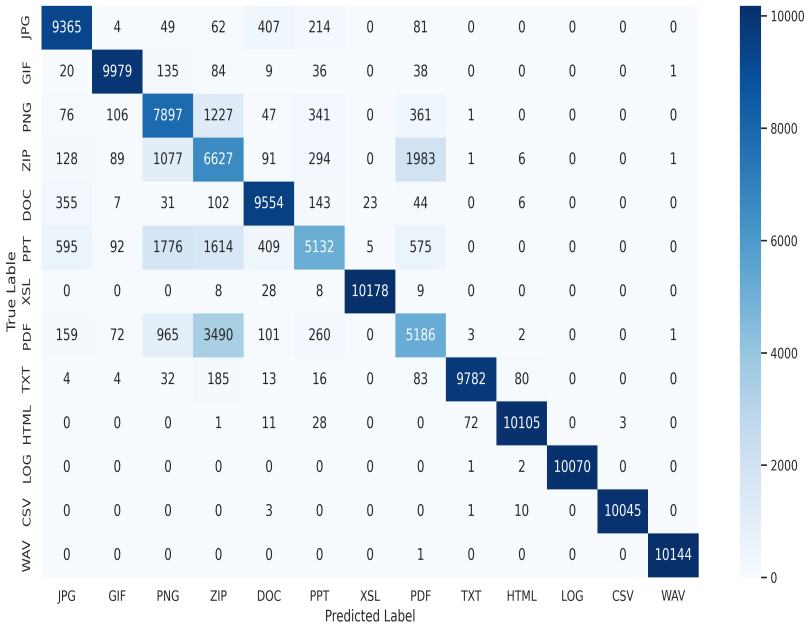

Figure 7.13: The confusion matrix of file fragments classification using transformer.

Figure 7.13 shows a confusion matrix for the transformer experiment. Similar to the CNN results, the confusion matrix contains the highest values along its diagonal, showing that many file types were identified with high accuracy. However, there are a relatively small number of confusions. The highest levels of confusion can be seen for PNG, ZIP, PPT and PDF. For example, PPT test files were commonly confused with JPEG, PNG, ZIP and PDF. Similarly, PDF files were most often confused with JPEG, PNG, ZIP and PPT. The smallest number of confusions was given by the LOG file type, which was confused only 3 times out of 10073 classifications.

7.4.3 Discussion

The results of the CNN and transformer are very similar in terms of the file types that are either hard or easy to classify. This result is not surprising as both are fundamentally similar, being neural network approaches. In addition, the data used in the experiments is the same, meaning that some files that are hard to classify using one method may be because the file is inherently difficult to classify. For example, text files are generally easier to classify as they can contain highly structured and informative data, such as tags (< or >) in the case of HTML, or repetitive trace information in the case of LOG files. By contrast, compressed files are inherently difficult to classify because compression algorithms aim to remove repeated patterns which provide essential information to file fragment classification algorithms. Overall, the mean accuracy is similar between both transformer and CNN, but the transformer is more robust to hyper parameter selection, and gives more even classification performance across the file types.

7.5 Summary

In this chapter, we explored the use of deep learning methods for the task of file fragment classification. Specifically, we used NLP deep learning techniques, whereby file fragments are considered like text data, with each fragment representing a sentence containing 512

words (or bytes). We looked at two approaches, CNN and transformer. Both approaches improved classification accuracy over the hierarchical techniques explored in the previous chapter, with the mean classification accuracy increasing from 75% to 86.14% using the transformer model, and 86.23% with the CNN. Despite the higher accuracy, the pattern of classification errors is similar to that observed previously. For example, complex files such as PPT remain challenging to distinguish from certain other file types. In the next chapter we will compare these results with those obtained using hierarchical modelling.

8 Evaluation and Discussion

8.1 Introduction

The purpose of this chapter is to provide a statistical comparison of the different approaches proposed in this thesis for the classification of file fragments, based on evaluation results presented earlier in Chapters 4, 5 and 6. Essentially, what we are trying to do is to determine first whether the results presented in these earlier chapters were statistically significant and not simply a result of chance, and to compare the results of the different approaches that were proposed.

For comparing classifiers, several statistical tests are available; the question is which one should be used. Janez Demsar conducted a comprehensive theoretical and practical study of the available statistical tests for comparing machine learning algorithms [Demšar, 2006]. According to this study, non-parametric statistical tests are recommended for classification algorithm comparison purposes. More specifically, the Wilcoxon signed-rank test was recommended for comparing two classifiers, while the Friedman test with a corresponding post-hoc test was recommended for more than two classifiers. Consequently, these tests were adopted here to conduct the desired comparisons.

The rest of this chapter is organised as follows: Section 8.2 presents the comparisons with respect to the hierarchical models, which are binary tree and DAG. Section 8.3 then provides an evaluation of non-feature-based deep learning models (CNN and transformer models). Finally, a summary of this chapter is presented in Section 8.4.

8.2 Statistical Evaluation of Hierarchical File Fragments Classification Models

A statistical comparison is presented here between the random forest as a base classifier and the binary tree and DAG hierarchical classification models presented in Chapter 6. To generate the binary tree and DAG tree classification models, the choice of the base classifier and the data feature selection technique can affect the classification accuracy. Based on the evaluation presented in Chapter 5, random forest was found to be the most effective classifier for generating a binary tree and a DAG tree. In terms of feature selection, the mutual information gain filter outperformed the information gain filter for the file fragment classification task. This section aims to determine whether this classification result is statistically significant.

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
Binary Tree - Random Forest	Negative Ranks	0[a]	.00	.00
	Positive Ranks	10[b]	5.50	55.00
	Ties	0[c]		
	Total	10		

a. Binary Tree < Random Forest
b. Binary Tree > Random Forest
c. Binary Tree = Random Forest

Test Statistics[a]

	Binary Tree - Random Forest
Z	-2.803[b]
Asymp. Sig. (2-tailed)	.005

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Figure 8.1: Wilcoxon signed-rank test of binary tree and random forest.

Figure 8.1 shows a statistical comparison between two classifier models which are binary tree and random forest. According to the outcome of the Wilcoxon signed-rank test, there was a significant difference in the performance between the two models. The binary tree model was found to be statistically more effective than the random forest for the file fragment classification task. With respect to the binary tree, a rank of z=-2.666 and p(.008)<.05. This confirms that the binary tree performed statistically significantly better than the random forest approach.

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
DAG Tree - Random Forest	Negative Ranks	0[a]	.00	.00
	Positive Ranks	10[b]	5.50	55.00
	Ties	0[c]		
	Total	10		

a. DAG Tree < Random Forest
b. DAG Tree > Random Forest
c. DAG Tree = Random Forest

Test Statistics[a]

	DAG Tree - Random Forest
Z	-2.803[b]
Asymp. Sig. (2-tailed)	.005

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Figure 8.2: Wilcoxon signed-rank test of random forest and DAG tree.

Figure 8.3 gives a statistical comparison between two classifier models, which are random forest and DAG tree. According to the outcome of the Wilcoxon signed-rank test, there was a significant difference in the performance between the two models. DAG model was found to be significantly better than random forest. With respect to DAG, a rank of z=-2.689 was obtained, with p(.007)<.05. These results confirm that the DAG performed statistically significantly better than random forest.

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
DAG Tree - Binary Tree	Negative Ranks	0[a]	.00	.00
	Positive Ranks	10[b]	5.50	55.00
	Ties	0[c]		
	Total	10		

a. DAG Tree < Binary Tree
b. DAG Tree > Binary Tree
c. DAG Tree = Binary Tree

Test Statistics[a]

	DAG Tree - Binary Tree
Z	-2.803[b]
Asymp. Sig. (2-tailed)	.005

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Figure 8.3: Wilcoxon signed-rank test of binary tree and DAG tree.

A statistical comparison between binary tree and DAG tree is presented in Figure 8.3. The results of a Wilcoxon signed-rank test revealed that the performance of these two models is statistically significantly different. Specifically, the DAG model was found to outperform the binary tree, with a rank of z=-2.689 being obtained for p(.005)<.05. Therefore, the results from the DAG were significantly better than those obtained by the binary tree.

8.3 Deep Learning File Fragments Classification Model Statistical Evaluation

A statistical comparison is presented here between the random forest as a base classifier and the CNN and transformer models presented in Chapter 7.

Based on the evaluation presented in Chapter 7, CNN and transformer were found to be the most effective classifiers for file fragment classification. This section aims to determine whether this classification result is statistically significant.

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
CNN - Random Forest	Negative Ranks	0[a]	.00	.00
	Positive Ranks	10[b]	5.50	55.00
	Ties	0[c]		
	Total	10		

a. CNN < Random Forest
b. CNN > Random Forest
c. CNN = Random Forest

Test Statistics[a]

	CNN - Random Forest
Z	-2.803[b]
Asymp. Sig. (2-tailed)	.005

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Figure 8.4: Wilcoxon signed-rank test of random forest and CNN.

Figure 8.4 shows a statistical comparison between two classifier models which are random forest and CNN. According to the outcome of the Wilcoxon signed-rank test, there was a significant difference in the performance between the two models. The CNN model was found to be statistically more effective than the random forest for the file fragment classification task. With respect to CNN, a rank of z=-2.803 and p(.005)<.05 was obtained. This confirms that the CNN performed statistically better than the random forest approach.

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
Transformer - Random Forest	Negative Ranks	0[a]	.00	.00
	Positive Ranks	10[b]	5.50	55.00
	Ties	0[c]		
	Total	10		

a. Transformer < Random Forest
b. Transformer > Random Forest
c. Transformer = Random Forest

Test Statistics[a]

	Transformer - Random Forest
Z	-2.803[b]
Asymp. Sig. (2-tailed)	.005

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Figure 8.5: Wilcoxon signed-rank test of random forest and transformer.

Figure 8.5 presents the results of a statistical analysis of the results obtained by the random forest and transformer classifiers. A Wilcoxon-signed rank test was performed for this analysis. The results of this test showed that the difference in performance between the classifications of both models was statistically significant. Therefore, the transformer is superior to the random forest classifier for the file fragment classification task. The results of the test gave a rank of z=-2.803 and p(.005)<0.5.

Wilcoxon Signed Ranks Test

Ranks

		N	Mean Rank	Sum of Ranks
Transformer – CNN	Negative Ranks	4[a]	4.13	16.50
	Positive Ranks	5[b]	5.70	28.50
	Ties	1[c]		
	Total	10		

a. Transformer < CNN
b. Transformer > CNN
c. Transformer = CNN

Test Statistics[a]

	Transformer – CNN
Z	-.711[b]
Asymp. Sig. (2-tailed)	.477

a. Wilcoxon Signed Ranks Test
b. Based on negative ranks.

Figure 8.6: Wilcoxon signed-rank test CNN and transformer.

The analysis so far has shown that both the CNN and transformer outperform the baseline random forest classifier. Next, we will determine if the results obtained by the CNN and transformer are statistically significantly different and, therefore, whether either classifier is better than the other. Figure 8.6 provides the results of this statistical analysis, comparing the results of classifications produced by the random forest and transformer classifiers. As before, a Wilcoxon signed-rank test was completed. The test did not reveal a statistically significant difference between the CNN and transformer classifications. This means that although both approaches are better than the random forest classifier, they could not be described as better than each other. A rank of z=-0.711 and p(.477)>0.05 were produced by this test.

8.4 Comparison

This section provides a general statistical comparison between all models in this thesis, using the Friedman test. Figure 8.7 shows a statistical comparison between all models in this thesis. According to the outcome of the Friedman test, there was a significant difference in the performance between the models. The table headed 'Ranks' provides a ranking of the classification performance for the 5 models tested. The results in this table show that the baseline random forest model performs worst out of all models. The two hierarchical models are ranked next, followed by the two neural network approaches. The transformer model is confirmed to be the highest ranked model in terms of classification performance.

Descriptive Statistics

	N	Mean	Std. Deviation	Minimum	Maximum
Random Forest	10	.68436590201	.00852513380	.67065044308	.69980000000
Binary Tree	10	.73392409400	.00374015601	.72863660035	.74087223077
DAG Tree	10	.74812409400	.00491532447	.73925462562	.75627953846
CNN	10	.862770	.0048651	.8516	.8680
Transformer	10	.864400	.0025543	.8605	.8683

Friedman Test

Ranks

	Mean Rank
Random Forest	1.00
Binary Tree	2.00
DAG Tree	3.00
CNN	4.45
Transformer	4.55

Test Statistics[a]

N	10
Chi-Square	38.211
df	4
Asymp. Sig.	<.001

a. Friedman Test

Figure 8.7: Friedman signed-rank test of all models.

Then we generate a critical difference diagram 8.8 based on the Wilcoxon-Holm method to detect pairwise significance. The diagram ranks classification performance from highest to lowest, with lower ranks representing better performance. From the diagram, we can

also determine whether the results from the models are statistically significantly different from each other. The figure shows that the results from the RF, BT and DAG models are significantly different from each other, and their results are significantly poorer than those obtained using either neural network approach (transformer and CNN). Therefore, the transformer and CNN models are confirmed to outperform the baseline and hierarchical approaches. Finally, although the ranking in Figure 8.8 shows that the transformer outperforms the CNN model, there is no significant difference between the output of these two models.

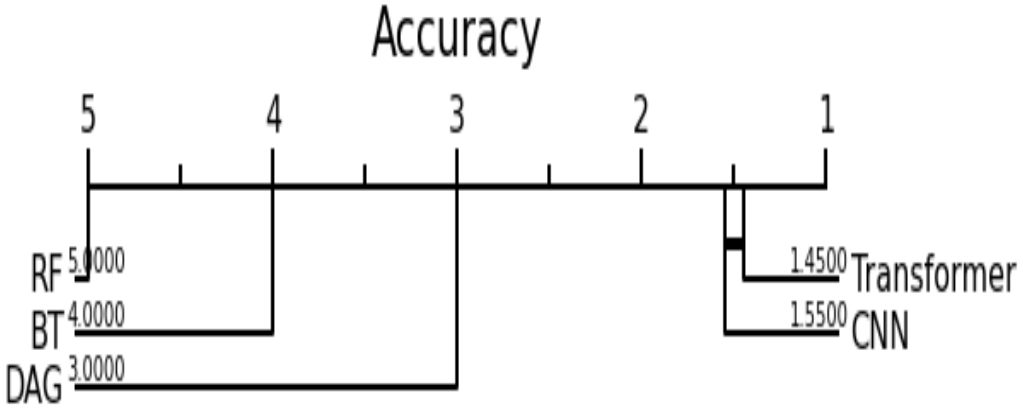

Figure 8.8: Critical difference diagram shows the statistical difference between the models. The bold line connection classifiers means that they are not statistically different.

8.5 Efficiency

Although classification accuracy is a very important aspect to discuss when evaluating the performance of a classification algorithm, the time taken for the algorithm to run is also valuable to consider. For example, if one algorithm offers only marginal accuracy improvements over another, but takes much longer to run, then it is arguably a less practical solution, even though it offers measurably better classification accuracy. In this section, we present the time taken for each of our algorithms to run through training and testing and compare them in the context of their classification performance.

Table 8.1: The training and testing time processes for random forest, binary tree, DAG tree, CNN and transformer.

The Model	Training Time/seconds	Testing Time/seconds
Random Forest	2.30	0.08
Binary Tree	21.37	161.62
DAG Tree	31,620	788
CNN	8,327.89	9.81
Transformer	26,927.45	67.99

Algorithms trained and tested using normal CPU compute capability were done so on an Intel(R) Xeon(R) Processor, model number E5-2670 v3, operating at 2.30GHz. These models include random forest, binary tree and DAG tree. The CNN and transformer models were trained and tested using Graphical Processing Unit (GPU) technology, for which an NVIDIA Quadro RTX 6000 GPU was used.

Table 8.1 shows the results of timing the training and testing processes for each of the methods explored in this thesis. The results show that the fastest method was the random forest, which takes only a few seconds to train and less than a second to test the data. However, random forest was shown to be the poorest performing approach in this work. The binary tree took longer to test and train, and also took longer to train than other methods producing superior classification results. Of the better performing approaches, DAG, CNN and transformer, the transformer achieved some of the highest classification accuracies but also took the second longest to train. Its classification results were marginally worse than those obtained using the CNN, but the CNN approach took less than a third of the time to train, and less than ten seconds to test. Although the DAG tree provided good classification accuracies, both its testing and training times were the longest of all methods explored here. It should be mentioned that although training times are greater for these better performing methods, training time is a less important consideration when it comes to the practical application of these methods to file forensic tasks. For these reasons, the CNN approach is arguably the most practical method for file fragment classification, offering

the most reasonable training and testing times, whilst providing the highest classification performance.

8.6 Summary

This chapter has provided the statistical comparison of the random forest, binary tree, DAG tree, CNN and transformer methods. For the feature-based approaches, there was a statistically significant difference between the results obtained by the random forest, binary tree and DAG tree. The DAG tree achieved the best accuracy compared to the other feature-based approaches. The results obtained by the transformer and CNN methods were not statistically significantly different. However, both of these neural network approaches were significantly different to the feature-based approaches. These neural network approaches were also found to be the most efficient in terms of classification accuracy. Although their training durations were among some of the longest we recorded, their testing durations were the best, with the exception of the random forest. Finally, the CNN approach was shown to be the best overall approach in terms of classification accuracy and testing duration.

9 Conclusion and Further Work

9.1 Introduction

This is the final chapter in this thesis and it provides an overview of the research described in this thesis. Here, we will provide a summary of the main findings and contributions, as well as some potential directions for future work. The remainder of this section is organised as follows: Section 9.2 provides a brief summary of the work presented, while Section 9.3 presents the main findings in light of the research question and discusses them in relation to the issues outlined in Chapter 1. Finally, Section 9.4 offers some suggestions for future research.

9.2 Summary of the Work

Several approaches have been proposed in this thesis to solve the problem of classifying file fragments. The two main approaches are hierarchical and deep learning modelling. The hierarchy modelling comprises a set of base classifiers held within the nodes of the hierarchy (one classifier per node). Nodes near the root hold classifiers are designed to discriminate between groups of class labels, while the leaves hold classifiers are designed to distinguish between individual class labels. Two types of hierarchy (structures) were considered, Binary Tree (BT) hierarchies and Directed Acyclic Graph (DAG) hierarchies. As these hierarchical modelling approaches are feature-based, we focus on finding the most suitable features that represent file fragments. In Chapter 5, we studied the use of hybrid feature selection for this task and found that the mutual information gain filter outperformed the information gain filter. We used a wrapper with a forward selection approach, because this approach is more efficient, in that it is faster to run than the backward selection method.

In Chapter 6, we explored the idea that specialised feature selection at each node would improve the classifiers' ability to discriminate between the different file types of file fragments.

BT proved to be able to order files into a logical hierarchy. In general, the DAG tree outperformed the BT for the classification task, achieving 75.16% compared to 73.60%. However, the results of both approaches suggested that we may have reached a performance limit using these more feature-based, traditional techniques. Motivated by this, in Chapter 7 we examined the application of Deep Neural Networks (DNNs) to this task. As DNNs do not require the use of feature selection techniques, we were able to explore this problem using unstructured data (the native format of the data), with the expectation that the deep learning methods would extract the most informative features by themselves within the Embedding layer.

We explored two popular approaches, which are commonly applied to Natural Language Processing tasks. Namely, the 1D Convolutional Neural Network (CNN) including an embedded layer, and the transformer model. The results of our experiments revealed that the deep learning approaches outperformed the techniques evaluated in previous chapters. CNN and transformer obtained similar results, in the region of 86% classification accuracy. We identified that there appeared to be file types that proved challenging to classify using both the traditional and the deep learning approaches, suggesting that these classes are inherently difficult to distinguish.

9.3 Main Findings

This section presents the main findings of the research work presented in this thesis. As originally stated in Chapter 1, the main research questions to be addressed were:

1. Does the binary content of file fragments contain patterns that represent their file type? Based on the academic literature, researchers believe that there are patterns for each file type. Moreover, on the basis of our experiments, we found that some

file fragment types have very clear patterns that distinguish them from other file fragments, whereas some other file types do not reliably contain an easily identifiable pattern.

2. Can file fragments be represented as structured data with specific features? The literature review showed that researchers commonly extract different statistical features such as unigrams, bigrams and entropy, which help represent the file fragments in a feature-based approach. We also used these techniques in this work, again confirming that file fragments can be represented in this way.

3. What are the most suitable features for representing file fragments? We designed experiments to select the most representative features using different feature selection techniques. To establish the most suitable features, we first gathered a list of the most frequently used features in file fragment classification research. We generated 66,313 features for each fragment, from which we aimed to identify the most suitable ones. To find the most suitable ones, we explored multiple techniques, and the results of our findings showed that using hybrid feature selection was the best approach. We used filters and wrappers in sequence. We found that retaining the top 900 features of the full set of 66,313 features gave the best results. This implies that 73% of the original features are redundant and can be discarded. Further filtering using the wrapper applied to the 900 features showed that between 11 and 15 features are enough to represent each file fragment.

4. Is there a specific classifier that can classify file fragments based on their binary content correctly and reliably? Based on the experiments presented in Chapter 5, in which we explored 5 different base classifiers, including decision tree, K-nearest neighbour, support vector machines, naive Bayes, and random forest. The results of these experiments showed that random forest provided the highest degree of classification accuracy out of the 5 classifiers tested. However, the maximum accuracy was approximately 69%, indicating that there is room for improvement.

5. Can hierarchical modelling improve the detection rate for file fragment types? The answer to this question was addressed in the experiments described in Chapter 6. The three hierarchical approaches evaluated were random forest, Binary Tree (BT), and Directed Acyclic Graph (DAG). Both hierarchical models achieved higher accuracy compared to the baseline, random forest model. DAG tree produced the highest classification accuracy of 75.16%. However, DAG does not outperform BT for all file types. In addition, BT was successful in arranging file types in a logical hierarchical structure. These results, and the extensive work undertaken to improve the selection of appropriate features, suggest that we may have reached the limit of classification accuracy for these hierarchical feature-based approaches. To achieve a better accuracy, it is necessary to use an alternative approach.

6. Can deep learning improve the accuracy of file fragment classification?

   In Chapter 7 we turned our attention to a non-feature-based approach. Specifically, we looked at using CNNs and the transformer architecture. The use of these deep learning methods gave superior classification accuracy to feature-based approaches. Although the maximum accuracy obtained using the previous techniques was 75%, accuracies of 86% were achieved using both the transformer and CNN. Although the overall accuracy increased using these deep learning approaches, analysis of the confusion matrices relating to the experiments showed that while the detection of some file types improved, others remained challenging to discriminate reliably.

9.4 Limitations and Further Work

Hierarchical modelling supports the idea of using a specialised approach for file fragment classification. In other words, the most effective method could be to create a specialised model for each file type, then ensemble them to generate an aggregate of the decisions. So it appears that the logical next step of the research will be to develop an approach to classification, known as a sequence ensemble. This approach will take advantage of the

relative strengths of different classification techniques. For example, the BT model could be used in the first stage, in which all file types are initially classified. Following this, the more complex file types could be processed by the deep learning method as the second classification stage.

In addition to the point above, a possible direction for future work could be to develop a new DAG Tree in which the classifiers at each node are deep learning models rather than more traditional models. This would essentially create a hierarchy of deep learning models, which could improve upon the results obtained in this work.

The parameters used in the neural network architectures in this work (CNN and transformer) were selected through manual tuning. Although we searched the parameter space to find good parameters, we only explored a small fraction of the entire space, and there may still be better parameters that we have not tested. Therefore, the results obtained in this work may be improved by using more complex network architectures or different hyperparameters. Another potential area for improvement is that our dataset includes some data types which are not entirely unique. For example, PPT files can themselves contain other file formats, such as images, text, videos and sound files. This problem could be solved by first extracting the embedded data from the container files and then training classifiers on the altered dataset.

The deep learning approach explored in this work relied on significant volumes of training data to generate classification models. While this is common for deep learning approaches, which typically require vast quantities of data to train effective models, we have not tested our approach using less data. It would be interesting to see how reducing the quantity of training data affects the classification accuracy, as this would reduce the time taken to train the models.

In developing the code for our DAG Tree approach, we were concerned with ensuring that the code worked correctly and, to a lesser degree, efficiently. Increasing the number of file classes leads to an exponential increase in the complexity of the processing required. As

the algorithm is largely arranged around a loop containing independent calculations and functions, it may be possible to parallelise this code to make it execute more efficiently.

10 Bibliography

Ahmed, I. (2010). *Fast And Accurate Content Classification Techniques For Malware Detection And File Type Identification.* PhD thesis, Ajou University.

Ahmed, I., Lhee, K.-s., Shin, H., and Hong, M. (2009). On improving the accuracy and performance of content-based file type identification. In *Australasian Conference on Information Security and Privacy*, pages 44–59. Springer.

Ahmed, I., Lhee, K.-s., Shin, H., and Hong, M. (2010). Content-based file-type identification using cosine similarity and a divide-and-conquer approach. *IETE Technical Review*, 27(6):465–477.

Ahmed, I., Lhee, K.-S., Shin, H.-J., and Hong, M.-P. (2011). Fast content-based file type identification. In *IFIP International Conference on Digital Forensics*, pages 65–75. Springer.

Al-Sadi, A., Yahya, M. B., and Almulhem, A. (2013). Identification of image fragments for file carving. In *World Congress on Internet Security (WorldCIS-2013)*, pages 151–155. IEEE.

Algurashi, A. (2023). File fragments dataset. `https://github.com/AliaAlgurashi/File_Fragments_Dataset/`. (Accessed on 09/08/2023).

Algurashi, A. and Wang, W. (2019). Hybrid feature selection method for improving file fragment classification. In *International Conference on Innovative Techniques and Applications of Artificial Intelligence*, pages 379–391. Springer.

Amirani, M. C., Toorani, M., and Beheshti, A. (2008). A new approach to content-based file type detection. In *2008 IEEE Symposium on Computers and Communications*, pages 1103–1108. IEEE.

Amirani, M. C., Toorani, M., and Mihandoost, S. (2013). Feature-based type identification of file fragments. *Security and Communication Networks*, 6(1):115–128.

Axelsson, S. (2010a). The normalised compression distance as a file fragment classifier. *Digital Investigation*, 7:S24–S31.

Axelsson, S. (2010b). Using normalized compression distance for classifying file fragments. In *2010 International Conference on Availability, Reliability and Security*, pages 641–646. IEEE.

Axelsson, S., Bajwa, K. A., and Srikanth, M. V. (2013). File fragment analysis using normalized compression distance. In *IFIP International Conference on Digital Forensics*, pages 171–182. Springer.

Beebe, N., Liu, L., and Sun, M. (2016). Data type classification: Hierarchical class-to-type modeling. In *IFIP International Conference on Digital Forensics*, pages 325–343. Springer.

Beebe, N. L., Maddox, L. A., Liu, L., and Sun, M. (2013). Sceadan: Using concatenated n-gram vectors for improved file and data type classification. *IEEE Transactions on Information Forensics and Security*, 8(9):1519–1530.

Bhat, A., Likhite, A., Chavan, S., and Ragha, L. (2021). File fragment classification using content based analysis. In *ITM Web of Conferences*, volume 40. EDP Sciences. Article number 03025.

Bhatt, M., Mishra, A., Kabir, M. W. U., Blake-Gatto, S., Rajendra, R., Hoque, M. T., and Ahmed, I. (2020). Hierarchy-based file fragment classification. *Machine Learning and Knowledge Extraction*, 2(3):216–232.

Bishop, C. M. et al. (1995). *Neural networks for pattern recognition*, page 116. Oxford University Press.

Calhoun, W. C. and Coles, D. (2008). Predicting the types of file fragments. *Digital Investigation*, 5:S14–S20.

Cao, D., Luo, J., Yin, M., and Yang, H. (2010). Feature selection based file type identification algorithm. In *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 3, pages 58–62. IEEE.

Chen, X.-W. and Lin, X. (2014). Big data deep learning: challenges and perspectives. *IEEE Access*, 2:514–525.

Chen, Y. (2015). Convolutional neural network for sentence classification. Master's thesis, University of Waterloo.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537.

Conti, G., Bratus, S., Shubina, A., Sangster, B., Ragsdale, R., Supan, M., Lichtenberg, A., and Perez-Alemany, R. (2010). Automated mapping of large binary objects using primitive fragment type classification. *Digital Investigation*, 7:S3–S12.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research*, 7:1–30.

Devijver, P. A. and Kittler, J. (1982). *Pattern recognition: A statistical approach*, page 41. Prentice Hall.

Erbacher, R. F. and Mulholland, J. (2007). Identification and localization of data types within large-scale file systems. In *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE'07)*, pages 55–70. IEEE.

Fitzgerald, S., Mathews, G., Morris, C., and Zhulyn, O. (2012). Using NLP techniques for file fragment classification. *Digital Investigation*, 9:S44–S49.

Garfinkel, S., Farrell, P., Roussev, V., and Dinolt, G. (2009). Bringing science to digital forensics with standardized forensic corpora. *Digital Investigation*, 6:S2–S11.

Garfinkel, S., Nelson, A., White, D., and Roussev, V. (2010). Using purpose-built functions and block hashes to enable small block and sub-file forensics. *Digital Investigation*, 7:S13–S23.

Goldberg, Y. (2016). A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420.

Gopal, S., Yang, Y., Salomatin, K., and Carbonell, J. (2011). Statistical learning for file-type identification. In *2011 10th international conference on machine learning and applications and workshops*, volume 1, pages 68–73. IEEE.

Gurney, K. (2018). *An introduction to neural networks*, page 234. CRC Press.

Hall, G. A. et al. (2006). Sliding window measurement for file type identification. Technical report, ManTech Security and Mission Assurance.

Harris, R. M. (2007). Using artificial neural networks for forensic file type identification. Master's thesis, Purdue University.

Hickok, D. J., Lesniak, D. R., and Rowe, M. C. (2005). File type detection technology. In *Proceedings from the 38th Midwest Instruction and Computing Symposium*. http://www.micsymposium.org/.

Kabir, M. M., Islam, M. M., and Murase, K. (2010). A new wrapper feature selection approach using neural network. *Neurocomputing*, 73(16-18):3273–3283.

Karresand, M. and Shahmehri, N. (2006a). File type identification of data fragments by their binary structure. In *Proceedings of the IEEE Information Assurance Workshop*, pages 140–147. IEEE.

Karresand, M. and Shahmehri, N. (2006b). Oscar-file type and camera identification using the structure of binary data fragments. In *1st Conference on Advances in Computer*

Security and Forensics (ACSF 2006) 13-14 July 2006, Liverpool, UK, page 11. School of Computing and Mathematical Sciences, John Moores University.

Kattan, A., Galván-López, E., Poli, R., and O'Neill, M. (2010). Gp-fileprints: File types detection using genetic programming. In *European Conference on Genetic Programming*, pages 134–145. Springer.

Kira, K., Rendell, L. A., et al. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, volume 2, pages 129–134. AAAI Press.

Kononenko, I., Šimec, E., and Robnik-Šikonja, M. (1997). Overcoming the myopia of inductive learning algorithms with relieff. *Applied Intelligence*, 7(1):39–55.

Kornai, A. (2010). *Mathematical linguistics*, page 248. Springer Science & Business Media.

Li, B., Wang, Q., and Luo, J. (2006). Forensic analysis of document fragment based on svm. In *2006 International Conference on Intelligent Information Hiding and Multimedia*, pages 236–239. IEEE.

Li, Q., Ong, A., Suganthan, P., and Thing, V. (2011). A novel support vector machine approach to high entropy data fragment classification. In *Proceedings of the South African Information Security Multi-Conf (SAISMC)*, pages 236–247. University of Plymouth.

Li, W.-J., Wang, K., Stolfo, S. J., and Herzog, B. (2005). Fileprints: Identifying file types by n-gram analysis. In *Proceedings from the Sixth Annual IEEE SMC Information Assurance Workshop*, pages 64–71. IEEE.

Liu, H. and Yu, L. (2005). Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):491–502.

Manning, C. and Schutze, H. (1999). *Foundations of statistical natural language processing*, page 250. MIT Press.

McDaniel, M. and Heydari, M. H. (2003). Content based file type detection algorithms. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003*, page 10. IEEE.

Memon, N. and Pal, A. (2006). Automated reassembly of file fragmented images using greedy algorithms. *IEEE Transactions on Image Processing*, 15(2):385–393.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2–4, 2013, Workshop Track Proceedings*.

Mittal, G., Korus, P., and Memon, N. (2020). Fifty: large-scale file fragment type identification using convolutional neural networks. *IEEE Transactions on Information Forensics and Security*, 16:28–41.

Mokhov, S. A. and Debbabi, M. (2008). File type analysis using signal processing techniques and machine learning vs. file unix utility for forensic analysis. In *IMF 2008–IT Incident Management & IT Forensics*, pages 73–85. Gesellschaft für Informatik eV.

Moody, S. J. and Erbacher, R. F. (2008). Sádi-statistical analysis for data type identification. In *2008 Third international workshop on systematic approaches to digital forensic engineering*, pages 41–54. IEEE.

Nguyen, K., Tran, D., Ma, W., and Sharma, D. (2014). The impact of data fragment sizes on file type recognition. In *2014 10th International Conference on Natural Computation (ICNC)*, pages 748–752. IEEE.

Palczewska, A., Palczewski, J., Marchese Robinson, R., and Neagu, D. (2014). Interpreting random forest classification models using a feature contribution method. In *Integration of reusable systems*, pages 193–218. Springer.

Penrose, P., Macfarlane, R., and Buchanan, W. J. (2013). Approaches to the classification of high entropy file fragments. *Digital Investigation*, 10(4):372–384.

Qiu, W., Zhu, R., Guo, J., Tang, X., Liu, B., and Huang, Z. (2014). A new approach to multimedia files carving. In *2014 IEEE International Conference on Bioinformatics and Bioengineering*, pages 105–110. IEEE.

Ramírez-Corona, M., Sucar, L. E., and Morales, E. F. (2016). Hierarchical multilabel classification based on path evaluation. *International Journal of Approximate Reasoning*, 68:179–193.

Richard III, G. G. and Roussev, V. (2005). Scalpel: A frugal, high performance file carver. In *Proceedings of the 2005 Digital Forensics Research Workshop (2005)*, pages 1–10. DFRWS.

Roussev, V. (2010). Data fingerprinting with similarity digests. In *IFIP International Conference on Digital Forensics*, pages 207–226. Springer.

Roussev, V. and Quates, C. (2013). File fragment encoding classification—an empirical approach. *Digital Investigation*, 10:S69–S77.

Rukhin, A., Soto, J., Nechvatal, J., Smid, M., Barker, E., Leigh, S., Levenson, M., Vangel, M., Banks, D., Heckert, A., et al. (2001). *A statistical test suite for random and pseudorandom number generators for cryptographic applications*, volume 22. US Department of Commerce, Technology Administration, National Institute of.

Russell, S. J. (2010). *Artificial intelligence a modern approach*, page 693. Pearson Education, Inc.

Schultz, M. G., Eskin, E., Zadok, E., Bhattacharyya, M., and Stolfo, S. (2001). MEF: Malicious email filter: A UNIX mail filter that detects malicious windows executables. In *Proceedings of USENIX Annual Technical Conference - FREENIX Track. Boston, MA: June 2001*. The USENIX Association.

Singh, S. and Silakari, S. (2009). An ensemble approach for feature selection of cyber attack dataset. *International Journal of Computer Science and Information Security*, 7(2):297–302.

Sportiello, L. and Zanero, S. (2011). File block classification by support vector machine. In *2011 Sixth international conference on availability, reliability and security*, pages 307–312. IEEE.

Sportiello, L. and Zanero, S. (2012). Context-based file block classification. In *IFIP International Conference on Digital Forensics*, pages 67–82. Springer.

Tang, G., Müller, M., Rios, A., and Sennrich, R. (2018). Why self-attention? a targeted evaluation of neural machine translation architectures. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4263–4272. Association for Computational Linguistics.

Underwood, W. (2009). Extensions of the unix file command and magic file for file type identification. Technical report, Georgia Tech Institute of Technology.

Urbanowicz, R. J., Meeker, M., La Cava, W., Olson, R. S., and Moore, J. H. (2018). Relief-based feature selection: Introduction and review. *Journal of biomedical informatics*, 85:189–203.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, volume 30, pages 5998–6008. The MIT Press.

Veenman, C. J. (2007). Statistical disk cluster classification for file carving. In *Third international symposium on information assurance and security*, pages 393–398. IEEE.

Vulinović, K., Ivković, L., Petrović, J., Skračić, K., and Pale, P. (2019). Neural networks for file fragment classification. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1194–1198. IEEE.

Wang, Y., Su, Z., and Song, D. (2018). File fragment type identification with convolutional neural networks. In *Proceedings of the 2018 International Conference on Machine Learning Technologies*, pages 41–47. Association for Computing Machinery, New York, United States.

A Comparison of Dataset Sizes for Deep Learning Approach

In this appendix, we aim to provide supporting evidence for the utilisation of a larger dataset in the context of the deep learning chapter (Chapter 7). While Chapter 6 employed a dataset consisting of 300 examples for each of the 13 file types, Chapter 7 utilises an expanded version of the dataset specifically designed for deep learning approaches.

In Chapter 6, the dataset was carefully curated to include a representative sample of 300 examples of each of the 13 file types under investigation. This dataset served as the foundation for conducting experiments and analysing various aspects related to the research objectives outlined in that chapter. However, in Chapter 7, we expanded on the dataset used in Chapter 6, as training deep learning approaches typically requires more data than traditional machine learning techniques. The augmented dataset encompasses an increased number of examples specifically tailored for deep learning analysis. By incorporating a larger dataset, we aim to demonstrate the enhanced efficiency and performance of deep learning models compared to the traditional approaches presented in Chapter 6.

In this appendix, we present a detailed examination of the deep learning approach applied to the original Chapter 6 dataset. By utilising this smaller version of the dataset, we aim to demonstrate the comparative advantages and limitations of deep learning methodologies when faced with limited training data. Through a detailed analysis, we provide insights into the impact of dataset size on the effectiveness and applicability of deep learning techniques within this research domain.

Through this appendix, readers will gain a comprehensive understanding of the decision to expand the dataset for deep learning purposes and will be able to evaluate the merits of using a larger dataset for enhancing the efficiency and performance of deep learning models in this work.

A.1 CNN Experiment

Employing the same Convolutional Neural Network (CNN) structure as described in Chapter 7, along with the same set of hyperparameters, we obtained the following classification results using the smaller dataset used in Chapter 6.

Table A.1 presents a detailed analysis of the F1 score, precision, recall, and accuracy metrics for each file type when testing using the Convolutional Neural Network (CNN) trained on the smaller version of the dataset (The dataset used in Chapter 6). In particular, the highest accuracies were achieved for the LOG file type, with an accuracy score approaching 100%. PNG was the worst-performing file type, demonstrating an accuracy of approximately 16%. The second lowest accuracy was observed for the PPT file type, achieving an accuracy of around 17%. PDF was the third-lowest accuracy, with a score of 28%. However, when a larger dataset was used for training the CNN, significant improvements were observed in the classification accuracy of several file types (Chapter 7). Specifically, the classification accuracies for JPEG, GIF, PNG, ZIP, DOC, PPT, XLS, PDF, TXT, HTML, CSV, and WAV file types all increased relative to the results obtained using the smaller dataset (Table A.2). For a complete description of the results using the larger dataset, refer to Table 7.2 in Chapter 7. These improvements in accuracy indicate that the CNN model, coupled with the larger dataset, resulted in more accurate classification for these file types. Notably, the classification accuracy for JPEG, GIF, XSL, ZIP, TXT, CSV, WAV and HTML files was significantly improved, surpassing 90%. Overall, the mean accuracy across all file types saw a substantial improvement, rising from 64.95% to 86.23%. This improvement underlines the

Table A.1: Classification results for the CNN experiment using the smaller dataset. Classification performance is shown separately for the 13 file types, as well as a combined mean across all file types. The F1 measure, precision, recall and classification accuracy are the performance metrics presented.

File Type	F1 Score	Precision	Recall	Accuracy
JPEG	0.555	0.545	0.566	56.59%
GIF	0.553	0.491	0.632	63.24%
PNG	0.237	0.397	0.169	16.91%
ZIP	0.258	0.21	0.333	33.33%
DOC	0.643	0.606	0.686	68.60%
PPT	0.256	0.5	0.172	17.24%
XLS	0.97	0.977	0.963	96.27%
PDF	0.233	0.2	0.279	27.88%
TXT	0.862	0.806	0.926	92.59%
HTML	0.867	0.914	0.825	82.52%
LOG	1	1	1	100.00%
CSV	0.939	0.975	0.906	90.63%
WAV	0.992	1	0.985	98.48%
Mean	0.644	0.663	0.649	64.95%

effectiveness of using a larger dataset in combination with the CNN model, as it contributed to a notable increase in overall accuracy across a wide range of file types.

A.2 Transformer Experiment

By utilising the transformer structure outlined in Chapter 7, and employing the same set of hyperparameters, the following results were obtained with the smaller dataset used in Chapter 6. Table A.3 presents an analysis of the F1 score, precision, recall, and accuracy metrics for each file type when utilising the transformer model with training and test data from the smaller dataset used in Chapter 6. Notably, the highest accuracies were achieved for the LOG and WAV file types, with accuracies approaching 98%. Conversely, the PPT file type gave the poorest performance, with an accuracy of approximately 14%. The ZIP file type gave the second-lowest accuracy, achieving around 24%. Additionally, the PDF file type displayed a relatively low accuracy of 26%.

Table A.2: The comparison of CNN classification results using the small dataset and large dataset.

File Type	CNN Classification Accuracy Using The Smaller Dataset	CNN Classification Accuracy Using The Large Dataset
JPEG	56.59%	95.28%
GIF	63.24%	91.50%
PNG	16.91%	78.99%
ZIP	33.33%	91.61%
DOC	68.60%	87.83%
PPT	17.24%	53.25%
XLS	96.27%	99.36%
PDF	27.88%	27.96%
TXT	92.59%	96.65%
HTML	82.52%	98.95%
LOG	100.00%	99.98%
CSV	90.63%	99.75%
WAV	98.48%	99.95%
Mean	64.95%	86.23%

The use of a larger training dataset with the transformer model led to significant improvements in the classification accuracies of several file types. Table A.4 shows the accuracy scores for the JPEG, GIF, PNG, ZIP, DOC, PPT, XLS, PDF, TXT, HTML, CSV, and WAV file types. The results show that the accuracy of these file types increased relative to the results obtained using the smaller dataset. For a full description of the results of these experiments, refer to Table 7.3 in Chapter 7.

These improvements highlight the effectiveness of the transformer model when used with a larger training dataset. This results in higher classification accuracies across a range of file types. Notably, the classification accuracies for JPEG, GIF, DOC, XLS, ZIP, TXT, CSV, WAV, and HTML files showed a large improvement, surpassing 90% accuracy.

Overall, the mean accuracy across all file types showed a substantial improvement, rising from 64.08% to 86.14%. This improvement emphasises the effectiveness of employing a larger dataset in conjunction with the transformer model, as it significantly enhances the overall classification across a wide range of file types.

Table A.3: Classification results for the transformer experiment using the small dataset. Classification performance is shown separately for the 13 file types, as well as a combined mean across all file types. The F1 measure, precision, recall and classification accuracy are the performance metrics presented.

File Type	F1 Score	Precision	Recall	Accuracy
JPEG	0.595	0.561	0.633	63.30%
GIF	0.589	0.679	0.520	51.96%
PNG	0.314	0.277	0.364	36.36%
ZIP	0.208	0.181	0.244	24.36%
DOC	0.661	0.629	0.697	69.75%
PPT	0.237	0.667	0.144	14.43%
XLS	0.791	0.949	0.679	67.89%
PDF	0.265	0.269	0.260	26.04%
TXT	0.906	0.891	0.922	92.17%
HTML	0.882	0.820	0.953	95.35%
LOG	0.820	0.701	0.989	98.89%
CSV	0.971	1.000	0.944	94.38%
WAV	0.986	0.991	0.982	98.20%
Mean	0.633	0.663	0.641	64.08%

Table A.4: The comparison of the transformer classification results using small dataset and large dataset.

File Type	Transformer Classification Accuracy Using The Smaller Dataset	Transformer Classification Accuracy Using The Large Dataset
JPEG	63.30%	91.98%
GIF	51.96%	96.86%
PNG	36.36%	78.53%
ZIP	24.36%	64.36%
DOC	69.75%	93.07%
PPT	14.43%	50.32%
XLS	67.89%	99.48%
PDF	26.04%	50.65%
TXT	92.17%	95.91%
HTML	95.35%	98.87%
LOG	98.89%	99.97%
CSV	94.38%	99.86%
WAV	98.20%	99.99%
Mean	64.08%	86.14%

In conclusion, the results presented in this Appendix have demonstrated that large improvements in classification accuracy are observed when training a CNN using a larger volume of training data. This is to be as expected, as it is commonly understood that deep learning methods generally require larger amounts of data to achieve good classification performance, compared to more traditional machine learning approaches. However, traditional approaches can outperform deep learning techniques in cases where there is limited data available for training. Therefore, the choice of which approach to adopt may depend on the availability of data for training, be that limited by the cost or practicality or data acquisition. These results have just justified the use of the larger dataset in Chapter 7, compared to the smaller dataset used in earlier chapters.