PONTIFICAL CATHOLIC UNIVERSITY OF RIO GRANDE DO SUL
SCHOOL OF TECHNOLOGY
COMPUTER SCIENCE GRADUATE PROGRAM

NEURAL NETWORKS APPLIED TO FILE FRAGMENT CLASSIFICATION

ATILA LEITES ROMERO

Thesis submitted to the Pontifical Catholic University of Rio Grande do Sul in partial fulfillment of the requirements for the degree of Master in Computer Science.

Advisor: Prof. Dr. Avelino Francisco Zorzo

Porto Alegre
2020

Ficha Catalográfica

L533n Leites Romero, Atila

Neural networks applied to file fragment classification / Atila Leites Romero . – 2020.
53 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientador: Prof. Dr. Avelino Francisco Zorzo.

1. computer forensics. 2. data carving. 3. machine learning. 4. neural networks. 5. file fragment classification. I. Zorzo, Avelino Francisco. II. Título.

Atila Leites Romero

Neural networks applied to file fragment classification

This Thesis has been submitted in partial fulfillment of the requirements for the degree of Master of Computer Science, of the Graduate Program in Computer Science, School of Technology of the Pontifícia Universidade Católica do Rio Grande do Sul.

Sanctioned on ________ _______, 20________.

COMMITTEE MEMBERS:

“If we are offered several hypotheses, we should begin our considerations by striking the most complex of them with our sword.”
(Isaac Asimov and Robert Silverberg [AS91])

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nivel Superior – Brasil (CAPES) – Finance Code 001.

I would also like to acknowledge the support provided by the Pontifical Catholic University of Rio Grande do Sul (PUCRS) and by the National Institute of Science and Technology on Forensic Sciences (INCT) during the preparation of this dissertation.

REDES NEURAIS APLICADAS A CLASSIFICAÇÃO DE FRAGMENTOS DE ARQUIVOS

RESUMO

Classificação de fragmentos de arquivos visa identificar o tipo original de um arquivo de onde um bloco de dados foi extraído. As técnicas de aprendizado de máquina, e as redes neurais em particular, têm o potencial de aprimorar esta área, porque o suporte a um novo tipo de arquivo usando métodos tradicionais é trabalhoso e não é automático. Embora estudos sobre redes neurais aplicadas à classificação de fragmentos de arquivos tenham mostrado bons resultados, para aplicar essas contribuições em cenários reais é necessário um melhor entendimento de como esses métodos respondem a um aumento no número de tipos de arquivos suportados e quais são as principais fontes de erros desta abordagem.

Este trabalho foca nos desafios da aplicação de redes neurais para classificação de fragmentos de arquivos e está organizado em três partes. Na primeira, os valores de acurácia de alguns tipos de redes neurais são comparados, classificando fragmentos de arquivos extraídos do *dataset* Govdocs1, usando suas extensões de arquivo como classes. Os resultados sugerem que os tipos de arquivo que compõem o *dataset* têm um papel mais relevante nos resultados do que os detalhes de arquitetura dos modelos de redes neurais. Na segunda parte, a influência do número de classes na acurácia é explorada. A conclusão é que o número de classes no *dataset* é relevante, mas menos importante que os tipos de extensões selecionadas para compô-lo. Finalmente, um procedimento é criado para testar a hipótese de que parte dos erros de classificação de fragmentos de arquivos pode ser explicada pela incapacidade dos modelos de distinguir dados de alta entropia de dados aleatórios. A conclusão é que, para o modelo usado, a incapacidade de distinguir dados de alta entropia de dados aleatórios pode explicar apenas 1/3 dos erros. Isso sugere que, para pesquisas futuras sobre classificação de fragmentos de arquivos, a busca por um procedimento para rotular automaticamente estruturas internas de arquivos pode ser uma abordagem promissora.

Palavras-Chave: computação forense, *data carving*, aprendizado de máquina, redes neurais, *long short-term memory*, camadas convolucionais, classificação de fragmentos de arquivo, dados de alta entropia.

NEURAL NETWORKS APPLIED TO FILE FRAGMENT CLASSIFICATION

ABSTRACT

File fragment classification aims to identify the original type of file from which a given block of data was extracted. Machine learning techniques, and neural networks in particular, have the potential to improve this field, because the support of a new file type using traditional methods is laborious and not automatic. While studies on neural networks applied for file fragment classification have shown good results, to apply those contributions on real scenarios a better understanding is required on how those methods respond to an increase in number of supported file types and what the are the main sources of errors of this approach.

This work focus on the challenges of applying neural networks for file fragment classification and it is organized into three parts. In the first part, the accuracy values of some types of neural networks are compared, classifying file fragments taken from the Govdocs1 dataset, using their file extensions as class labels. The results suggest that the file types composing the dataset have a more relevant role in the results than the architectural details of the neural network models. In the second part, the influence of the number of classes on accuracy is explored. The conclusion is that the number of classes in the dataset is relevant, but less important than the types of extensions selected to compose the dataset. Finally, a procedure is devised to test the hypothesis that part of the errors of file fragment classification can be explained by the inability of the models to distinguish high entropy data from random data. The conclusion is that, for the used model, the inability to distinguish high entropy data from random data can only account for about 1/3 of the errors. This suggests that, for future researches on file fragment classification, the search for a procedure to automatically label inner file data structures may be a promising approach.

Keywords: computer forensics, data carving, machine learning, neural networks, long short-term memory, convolutional layers, file fragment classification, high entropy data.

LIST OF FIGURES

Figure 1.1 – Research parts 14
Figure 4.1 – Learning curves of some models (validation accuracy vs. time) 28
Figure 4.2 – Validation accuracy of models 28
Figure 4.3 – CLD vs. other studies 29
Figure 5.1 – Validation accuracy by number of classes 35
Figure 5.2 – Validation accuracy of models trained with pair of classes 35
Figure 5.3 – PCA of accuracy of models trained with pair of classes 36
Figure 5.4 – PCA of accuracy of models trained with pair of classes - detail 36
Figure 6.1 – Complement of entropy measure for 28 file types 45

LIST OF TABLES

Table 2.1 – Summary of data carving categorization schemes 20
Table 3.1 – File fragment classification studies using the Govdocs1 dataset 23
Table 3.2 – The Govdocs1 dataset and file types used in each study 24
Table 4.1 – Details of 14 models trained on 28 classes 26
Table 5.1 – Data fed to the PCA algorithm 39
Table 6.1 – Complement of entropy measure 44

LIST OF ACRONYMS

BLSTM – Bidirectional Long Short-Term Memory

CNN – Convolutional Neural Network

DANN – Distributed Adaptative Neural Network

ELM – Extreme Learning Machine

GB – Gygabyte

GPU – Graphics Processing Unit

GRE – Gene Regulatory Engine

KNN – k-Nearest Neighbors

LSTM – Long Short-Term Memory

MLP – Multilayer Perceptron

PCA – Principal Component Analysis

RAM – Random-access Memory

RNN – Recurrent Neural Network

SVM – Support Vector Machine

CONTENTS

1 INTRODUCTION **12**

2 BACKGROUND **16**

2.1 MULTILAYER PERCEPTRON 16

2.2 CONVOLUTIONAL NEURAL NETWORK 17

2.3 LONG SHORT-TERM MEMORY 18

2.4 DATA CARVING 19

2.5 CONCLUSION 20

3 RELATED WORK **21**

4 EXPLORATORY RESEARCH OF MODELS BASED ON ACCURACY **25**

4.1 METHOD 25

4.2 RESULTS 27

4.3 DISCUSSION 30

5 DESCRIPTIVE RESEARCH ON INFLUENCE OF NUMBER OF CLASSES . . . **32**

5.1 METHOD 32

5.2 RESULTS 34

5.3 DISCUSSION 37

6 EXPERIMENT ON RANDOM DATA DETECTION **40**

6.1 METHOD 41

6.2 RESULTS 43

6.3 DISCUSSION 45

7 CONCLUSION **47**

7.1 FUTURE WORK 48

REFERENCES **49**

1. INTRODUCTION

In a forensic context, file recovery is a frequent task that can be motivated by several situations, like a physical media malfunction, an intentional attempt to hide data, or the need to access deleted or older versions of files. When a file system no longer provides the physical location of a file on the media, data carving is often the only procedure capable of retrieving this content.

Data carving is a forensic process that attempts to recover files without previous information of where the file starts or ends [Gar07]. To accomplish this task, a program has to analyze a source of raw data, searching for patterns indicating a known file type and making attempts to locate and reconstruct each of its constituent parts. That process commonly disregards the file system [Vee07], being able to recover deleted files from unallocated areas, but faces the problem of fragmentation [Vee07] [PM09]: in many cases, files are not written sequentially on disk and deleted files may have missing parts.

Data carving is frequently used in forensic environments, but it may also be beneficial in other areas, such as reverse engineering, network traffic analysis, and data mining. This observation is related to the fact that many types of data sources contain embedded files. Therefore, they may be used as input to a data carving process. This includes network traffic, memory dumps, hard drive images, and files containing other files.

Furthermore, data carving may resort to many techniques as file fragment classification, whole file classification, and fragment reassembling. File fragment classification aims to identify the original type of file from which a given block of data was extracted. It is a more difficult task than whole file classification, because it cannot rely on structures of the beginning and end of the files, which are generally easier to recognize. Reassembling is the attempt to reconstruct a file from pieces that may be out of order and mixed with pieces from other files, and it is a procedure that depends on the correct output of the file fragment classification task.

Normally, computer users do not need to deal with hard disk sectors directly and have contact only with the already mounted file system, which presents directories and files for them. But to present such a view, the operating system has to interpret the raw media data, which is simply a stream of data blocks. For example, the first blocks of a drive usually contain a partition table, indicating ranges of blocks belonging to each partition. Inside a partition, the operating system expects to find a file system. The file system stores metadata about each file or directory and keeps an index indicating the position of each file on disk. This way, when a user accesses a file, the operating system uses the file system to find where the file is stored on the disk, accessing those areas directly and returning its content to the user, who sees the returned data as the file content. When a file is deleted, the corresponding index entry is erased, while the actual content of the file may be left untouched to avoid disk access. In this circumstance, a data carving procedure may successfully retrieve the file, even if the file system cannot. One common data carving approach that does not deal with fragmentation consists of searching for headers and footers. To retrieve the file, a software using this approach would sequentially read each drive sector, find a known header and save the following sectors until a footer is found or a size limit is reached.

Although researches are providing good alternatives in data carving, the algorithms used in practice by available data carving software are generally still manually coded, using fixed byte sequences found on headers and footers of the files. The amount of different file types combined with the slow process of manually coding each of those patterns makes the development of data carving software an overwhelming task [MH03].

The amount of work required to support the vast amount of file types in existence is arguably the main obstacle to implement new technologies on data carving software. The forensic community would benefit from research that could make the task of supporting the carving of a new file type easier. Machine learning techniques have the potential to achieve that goal because they can replace the step of manually encoding a structure parser by automatically recognizing patterns in large amounts of data.

The range of common file types, files that are most likely to be relevant in a piece of evidence, like images, videos, and documents, do not change frequently. Because of that, the available tools could so far be kept up to date, despite the work required to include a new file type.

But there are situations when this is not enough. This can happen when the relevant file type is proprietary, uncommon, or newly created. As an example, some video surveillance solutions use proprietary video formats that are not recognized by data carving software. In this situation, it would be beneficial to use a tool that, using examples of files of that particular kind, could create a customized model able to identify and retrieve this file type.

One strategy to solve this problem could be the use of neural networks to provide an automatic classification of file fragments. Some works already explored this approach, but the practice of using the file extension as labels for each of the file fragments may introduce errors when multiple file types use the same data structures. Thus, the plain search for better models without tackling the problem of inner data labeling may not be enough to achieve higher accuracy results.

Therefore, this work explores some of the challenges of applying neural networks in file fragment classification, attempting to answer the following research questions:

Q1. How do different neural network models compare to each other in terms of training performance and quality of results for file fragment classification?

Q2. How does the accuracy of neural network models changes relative to the number of classes in file fragment classification?

Q3. Can the inability of the models to distinguish high entropy data from random data explain part of the file fragment classification errors? If so, to what extent?

The initial goal of the first research question was to identify the most promising models on file fragment classification, but an apparent limit was found on how far these models could be improved. This led to the second research question, that explores the influence of the number of classes on the accuracy of the models. While the number of classes seems to influence the accuracy of the trained models, the choice of which file types are included in the dataset has a bigger impact because file types that contain images or use compression were found to have an expressive negative effect on accuracy. Motivated by these results, the third research question was formulated to test the hypothesis that part of the observed errors is caused by the inability of the models to distinguish high entropy data from random data.

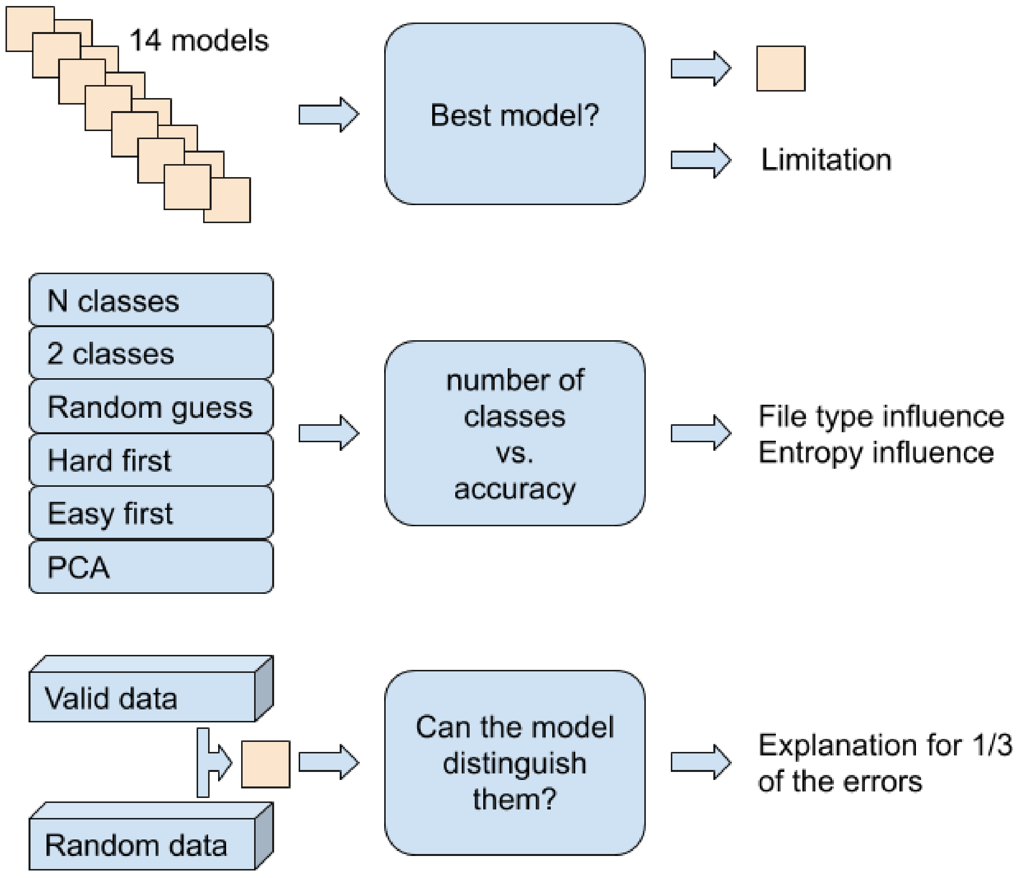

Figure 1.1 – Overview of the three parts of the research.

These parts are depicted in Figure 1.1. The first research, represented by the label “Best model?”, evaluates 14 models, pictured as orange squares. The two results are a single selected model and the label “Limitation”, representing the apparent limit found on accuracy improvements. The second research is represented in the middle, where the boxes in the left indicate the different types of input and methods that were applied, while the text in the right refers to the type of results that the research provided, on how file type and entropy influence accuracy limits. The last line represents the third research, where for each model trained (the orange square) valid data and random data were used as input. The right side indicates the result obtained, an explanation for about 1/3 of the errors.

In this work, only neural networks are taken into account, but other machine learning approaches could also be applied, like Support Vector Machines (SVM) [FMMZ12] and k-Nearest Neighbors (kNN) [Axe10]. This restriction was motivated by the success that neural networks have shown in other fields, like image classification [MBB+92] and speech recognition [GMH13].

The remainder of this document is organized as follows. Chapter 2 describes some key concepts relevant to this work. Chapter 3 analyzes current research on file fragment classification. Chapter 4 addresses the first research question, comparing the accuracy of fourteen neural network models. Chapter 5 addresses the second research question, exploring the influence of the number of classes on the accuracy of the models. Chapter 6 addresses the third research question, testing the hypothesis that part of the observed errors is caused by the inability of the models to distinguish high entropy data from random data. Finally, Chapter 7 concludes the research and includes suggestions for future work.

2. BACKGROUND

This chapter describes some key concepts of neural networks and data carving. Section 2.1 describes the Multilayer Perceptron, a simple neural network that shows the main core concepts and techniques used in other more complex neural networks. Section 2.2 describes the Convolutional Neural Network, a type of neural network actively used in image classification. Section 2.3 describes the Long Short-Term Memory, a neural network that can label sequences of inputs. Section 2.4 describes how some authors classify existing data carving techniques.

2.1 Multilayer Perceptron

Multilayer Perceptron (MLP) [Ros58] is a feedforward artificial neural network with at least three layers, in which all layers are fully-connected, meaning that all nodes of a layer are connected to all nodes of the next. The term feedforward implies that the connections between the nodes of the considered network do not form cycles, thus the layers are processed sequentially in order, without feedback loops.

One of the applications of these networks is classification, where given enough labeled input samples they can be trained to classify the inputs in a closed group of categories, often refered as “classes”.

The output of each layer is the result of a matrix multiplication of the input and a set of weights plus a bias vector, also subject to an optional activation function, as shown in Equation 2.1, where y is the output vector of the layer, x is the input vector, W is the set of weights matrix, b is the bias vector, and a is the activation function.

$$y = a.(x.W + b) \tag{2.1}$$

The training consists of a search for the weights that would minimize the error between the predicted output and the correct label of the instance. This error function is also known as loss function or cost function. Some authors prefer to distinguish between the two using the term “loss” to refer to a single training example and “cost” to refer to the average of losses in the training set.

An essential characteristic of this cost minimization procedure is that it is not random. Given the high number of parameters that can be adjusted, a random search for the lowest cost would be prohibitively expensive. The success of neural networks in several fields and applications would not be possible without the backpropagation algorithm

[RHM86]. This algorithm uses the gradient of the network formulas to propagate the error in the network's output back to each node, assessing how much each weight contributes to this error, and adjusting its value to minimize the error.

When the neural network output results for multiple classes, it is usual to use one-hot encoding in the input and to apply the softmax function in the last layer of the neural network. Softmax applies the exponential function on the vector and then normalizes it. In one-hot encoding, a value is converted into a vector where the number of elements equals the number of desired classes. Only one of the elements is set to one, corresponding to the class of the instance, while the others are set to zero.

Some relevant loss functions for multi-class neural networks are categorical cross-entropy, binary cross-entropy, and mean squared error. Assuming the input of the neural network uses one-hot encoding, categorical cross entropy is $-log(\hat{y}_i)$, where $\hat{y}_i$ is the neural network output for the correct class i. Binary cross entropy is similar, but penalizes errors in all classes instead of only the correct one. Mean squared error, like binary cross-entropy, also consider all classes simultaneously, but with a different formula. It is the mean of the results of $(\hat{y}_i - y_i)^2$, where y is the correct vector, $\hat{y}$ is the predicted vector, and i is a vector component, corresponding to a class.

To train the neural network, the dataset may be processed multiple times. Each of those cycles is usually called an epoch. Epochs can be further divided into batches, where the adjustments derived from the backpropagation are applied more frequently.

The training stopping condition is arbitrary. Some choices are to use a fixed number of epochs, to train for a fixed period of time, or to stop after a fixed number of epochs without improvement above certain threshold.

2.2 Convolutional Neural Network

Convolutional networks [LBD+89] [LB95] use a sliding window over the input. The convolutional kernel is a matrix with the size of that window. It processes the input matrix dividing it in small parts, processing each part using the same procedure described in the multilayer perceptron section, and concatenating the results of each part. The size of the step used to move the window while processing parts of the input matrix is the stride. If it is equal to the kernel size, the input matrix is divided and processed without overlap. To enforce that the final output has the same size of the input, padding may be used, which is a technique that increases the input matrix size, adding elements to its borders.

One of the advantages of convolutional layers is that the number of trainable parameters is reduced in comparison with a fully connected layer. And the linear transformation that the kernel does is uniformly applied across the entire input.

When used to process images, the input of the convolution is a matrix with two dimensions. But, in other applications, as file fragment classification, it is possible to use a one dimension convolution, since the core concept of the convolutional processing can be understood as a simple division of the input in parts, that are then processed separately, concatenating results at the end, a concept that can be applied on any number of dimensions.

After the convolutional layer, sometimes a pooling layer is added. It is a special type of convolution that applies a predetermined operation instead of a matrix multiplication with trainable weights. The two most common operations used in pooling layers are max and average. Max pooling returns the highest number of the values being processed, while average pooling uses the average of these values. Pooling layers offer a fast way to achieve translation invariance [Hin87], which means that the detection of patterns is not affected by where in the input they occur. The drawback is that it introduces information loss, since the output is smaller than the input.

2.3 Long Short-Term Memory

Recurrent Neural Network (RNN) are neural networks where part of the input of a layer uses its previous output [HS97]. Long Short-Term Memory [HS97] (LSTM) is a type of Recurrent Neural Network architecture that addresses the problem of vanishing or exploding gradients through time. This problem occurs in regular RNNs because the signal transmitted from one time-step to another is influenced at each interaction, both on forward and backward propagation phases, limiting the potential interaction between distant time-steps. The solution offered by LSTM introduces the concept of a memory cell, adding gates units to each node. The initial proposal included only input and output gates [HS97]. The input gate is responsible for regulating if the cell internal state will be affected by the layer input. Similarly, the output gate regulates if the internal state will influence the output of the layer. The concept of a forget gate was introduced later [GSC00], to control whether the internal state of the cell should be reset.

Equations 2.2 to 2.6 describe a typical LSTM network. At each time step t, x_t is the input vector, c_t is the cell's internal state and h_t is the output state. Both c_t and h_t influence the next time step as variables c_{t-1} and h_{t-1}. The symbol $\circ$ denotes an element wise multiplication.

The result of the forget, input, and output gates is represented by f_t, i_t, and o_t. In each of the three gates, W is the set of weights to be applied to the input x, while U is the set of weights to be applied on the previous time step output, h_{t-1}. The dimensions of x and h_{t-1} are respectively the chosen dimensions for input and output, while the dimension for W and U are such that the output has the same dimensions of h_{t-1}. Symbol b denotes the bias vector, having the same dimensions of both h and h_{t-1}. Symbol σ_g denotes the activation function of the gate. Each gate is mathematically equivalent to a fully connected layer where the inputs are the concatenation of the x and h_{t-1} vectors and the output has the same dimensions of h.

The activation function usually chosen for the gates is the sigmoid, which gives values in the $[0, 1]$ range. If the forget gate f_t results zero for a given component, the corresponding component in the c_{t-1} vector will have no influence on c_t or h_t. If all the resulting components of the forget gate are zero, which corresponds to the zero vector, then the previous cell's state c_{t-1} is completely ignored. Similarly, if the input gate i_t results in a zero component, the corresponding component in the c_t vector will not be influenced by x and h_{t-1}.

If all components of f_t and i_t are equal to 1, then $c_t = c_{t-1} + \sigma_c(x_t W_c + h_{t-1} U_c + b_c)$, which is the case where c_t is fully influenced by c_{t-1}, x and h_{t-1}. The output gate o_t follows a similar pattern, but it regulates how h_t will be influenced by c_t.

$$f_t = \sigma_g.(x_t.W_f + h_{t-1}.U_f + b_f) \tag{2.2}$$
$$i_t = \sigma_g.(x_t.W_i + h_{t-1}.U_i + b_i) \tag{2.3}$$
$$o_t = \sigma_g.(x_t.W_o + h_{t-1}.U_o + b_o) \tag{2.4}$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c.(x_t.W_c + h_{t-1}.U_c + b_c) \tag{2.5}$$
$$h_t = o_t \circ \sigma_h(c_t) \tag{2.6}$$

2.4 Data carving

Ali *et al.* [AMJK18a] divide the data carving process into three steps: identification, which classifies the file type of individual chunks of data; validation, which includes a list of requirements of a file that are needed for its recovery to be considered successful; and reassembling, which attempts to reconstruct the original file.

Nadeem [Nad13] groups the carving techniques into three generations, each extending the previous one. The first generation is header-footer based carving. It uses file signatures like magic-bytes, headers, and footers to identify the beginning and end of a file. The second generation is structure-based carving, also called “semantic carving” or “deep carving”. It reduces the number of false positives by using file structure knowledge to perform validation. The third generation advances reassembling with methods to deal with fragmentation. It tries to infer relationships and the order between chunks of data based on content and statistical analysis to reassemble the original file.

For file type detection, which could be mapped to the identification step in the process steps of Ali *et al.* [AMJK18a], Amirami *et al.* [ATB08] cite three categories: extension-based identification, magic bytes-based identification, and content-based identification. In extension-based identification, the content of the file is ignored and only its filename extension is used. Magic bytes-based identification uses signatures, generally a fixed string, usually at the beginning of a file. It is a common strategy that uses header/footer, but not all files adopt it. Content-based identification identifies the file using some statistical modeling of its content.

Beebe *et al.* [BMLS13] identify three content-based approaches to classify file and data types, also referring only to the identification step of the Ali *et al.* [AMJK18a] data carving process division: semantic parsing, nonsemantic parsing, and machine learning. Semantic parsing relies on the file structure to identify its type. Nonsemantic parsing searches for strings that are commonly found in specific files. Machine learning uses supervised and unsupervised algorithms, like Support Vector Machine (SVM), k-Nearest Neighbors (kNN), and neural networks.

A summary of the categorization schemes is depicted in Table 2.1. As Nadeem [Nad13] does not specifically mention machine learning, it is left out of generation classification.

Table 2.1 – Summary of data carving categorization schemes according to other authors

Steps	Techniques		Generation
Identification	extension-based		1st
	magic bytes-based		
	content-based	semantic	2nd
		non-semantic	
		machine learning	—
Validation			2nd
Reassembling			3rd

2.5 Conclusion

In this study, the types of layers described, the fully-connected layer of the Multilayer Perceptron, the convolutional and maxpooling layers of Convolutional Neural Networks, and the layers of the Long Short-Term Memory architecture, will be combined in different ways to create models to classify file fragments. Of the three data carving steps mentioned, identification, validation, and reassembling, this study focuses only in the identification one. It also focuses on file fragments, in contrast with the easier task of whole file classification, which can rely on headers and footers.

3. RELATED WORK

Several authors reviewed available data carving tools [AMJK18a] [QZG+14] [Nad13] [Rou08], but the tool listing from Ali *et al.* [AMJK18a] was found to be the most comprehensive one. Among the listed tools, only Foremost [KKM19], Scalpel [RIR05], and PhotoRec [Gre19] support a wide range of file formats. For example, Photorec supports more than 300 file types.

The available data carving tools generally do not take advantage of the latest techniques that research on the field offers, often still relying on header/footer identification and providing limited reassembling capabilities. According to Ali *et al.* [AMJK18a], artificial intelligence techniques are found to be not fully utilized in this field.

Comparing the accuracy between research papers and available software is difficult because, as they rely on header and footer identification, their performance would be more properly compared to whole file classification instead of file fragment classification, the latter being a harder problem.

The early studies on data carving used private datasets [KS06a] [Vee07] [EM07] [ME08] [CC08] [LOST10] [CBS+10] [KGLPO10], making it difficult to compare the results of different studies. In 2009, Garfinkel *et al.* [GFRD09] published the Govdocs1 dataset with 1 million files (a small amount of those files was removed later, for legal reasons). While some of the later works on the field still use private datasets, the Govdocs1 dataset has become the most used choice in studies related to data carving. Therefore, this related work summary focuses on studies on file fragment classification that used the Govdocs1 dataset.

Depending on the objective, the focus of each study may either be whole file classification or file fragment classification. The whole file classification is generally an easier task because most file types, even those with high entropy and low compression rates, have well-structured headers, with easily recognizable patterns.

Most of the early work in this field used some form of dimensionality reduction in the input data before performing classification, as byte frequency distribution [KS06b] [Har07] [ATB08] [ALSH10a] [ALSH10b] [SZ12] [ATM13] [QZG+14] [MSR14] [AMJK18b], compression rate [Axe10] [PMB13], and various statistical techniques [Vee07] [EM07] [ME08] [CC08] [LOST10] [KGLPO10] [GYSC11]. The usage of raw bytes [1] as input is recently becoming more popular [Hie18] [CLJ+18] [WQW+18] [WSS18] [VIP+19], especially in studies applying machine learning techniques, which seems to be a rising trend in the last ten years.

Each of the following studies chose a subset of file types from the Govdocs1 dataset, splitting this subset further into training and validation parts. The training dataset was used to train the machine learning solution to classify file fragments, while the validation

[1] raw bytes: the original bytes of the file, without conversion or dimensionality reduction

dataset was used to verify how accurately the model could predict the type of the file from which the fragment was extracted from.

Axelsson [Axe10] used the normalized compression distance as an input feature to a kNN algorithm, picking pairs of 512-byte fragments, compressing them with the bzip2 algorithm and comparing the length of the results. He got an average accuracy of around 34% using 28 file types.

Fitzgerald *et al.* [FMMZ12] used an SVM classifier with various statistical measures as input features, including unigram and bigram[2] histogram counts, Shannon entropy, and compression length. Omitting the first and the last fragments of each file, they got an average accuracy of 49.1% using 24 file types.

Beebe *et al.* [BMLS13] compared various measures of complexity and byte frequency distributions of 512-byte fragments as input features to an SVM algorithm. They used 8 data types, txt, base64, base85, urlencoded, fat fs data, ntfs fs data, ext fs data, aes256, and 30 file types from the Govdocs1 dataset and some other sources (19 are at least partially from the Govdocs1 dataset). They got an average accuracy of 73.4% combining unigram and bigrams as input features.

Chen *et al.* [CLJ+18] used a Convolutional Neural Network (CNN) with five convolutional layers and 3 fully connected layers to classify fragments of 4096 bytes, converting each fragment into a 64x64 grayscale picture. They got an average accuracy of 70.9% using 16 file types.

Hiester [Hie18] apparently was the first to use an LSTM network to perform file fragment classification. He compared the results of three types of neural networks: feedforward, convolutional, and LSTM. He used four file types from the Govdocs1 dataset. Each bit of a 512-byte fragment was translated into two features, resulting in 8192 features per block (512*8*2). The first and last sectors of each file in the training set were discarded. In the experiments, the datasets were limited to one gigabyte to fit in memory. He got an average accuracy of 98% using an LSTM network, 73% using CNN, and 76% using MLP.

Wang *et al.* [WQW+18] used sparse dictionaries for n-grams to extract features of 512-byte fragments, using this as input to an SVM classifier. They got an average accuracy of 61.3% using 18 file types.

Wang *et al.* [WSS18] compared CNN, SVM, kNN, and XGboost, with fragments of different sizes, converting each byte to a 256-length vector (one-hot encoding). The CNN combined three parallel convolutional layers of widths 4, 8 and 16. They used 20 file types from the Govdocs1 dataset. Using CNN, they got an average accuracy of around 68% with 512-byte fragments, and around 78% with 4096-byte fragments.

[2] In this context, unigram is a single byte, while bigrams are sequences of two bytes.

Vulinović *et al.* [VIP+19] compared a CNN with MLP, using 512 raw bytes as the CNN input, and byte histograms as input to the MLP. Using 18 file types, they got a macro average F1-score of 61% using CNN and 81% using MLP.

Table 3.1 summarizes the results of file fragment classification studies using Govdocs1 dataset and Table 3.2 lists the file types used in each study.

Table 3.1 – File fragment classification studies using the Govdocs1 dataset

Study	SVM	kNN	NN	Dataset	Raw bytes	Fragment size	Number of types	Accuracy	F1-score
Axelsson [Axe10]		x		Govdocs1	No	512	28	34%	
Fitzgerald *et al.* [FMMZ12]	x			Govdocs1	No	512	24	49%	48%
Beebe *et al.* [BMLS13]	x			Govdocs1 + other sources	No	512	38	73%	
Hiester [Hie18]			MLP CNN LSTM	Govdocs1	Yes	512	4	77% 73% 98%	
Chen [CLJ+18]			CNN	Govdocs1	Yes	4096	16	71%	
Wang [WQW+18]	x			Govdocs1	Yes	512	18	61%	61%
Wang [WSS18]			CNN	Govdocs1	Yes	512 4096	20	68% 78%	
Vulinovic [VIP+19]			MLP CNN	Govdocs1	Yes	512	18		81% 61%

In the early stages of the study described in this dissertation, the search for a better model was assumed to be the best approach to improve file fragment classification results. This assumption seems to be also present in the many of the reviewed works, as their approaches suggest. This study found indications that this assumption may be false, and that further improvement may require a review of the labeling system commonly used, that uses the extension of the file as the label for all its fragments.

Table 3.2 – The Govdocs1 dataset and file types used in each study

Extension	Number of files	Number of blocks	this study	Axelsson [Axe10]	Fitzgerald *et al.* [FMMZ12]	Beebe *et al.* [BMLS13]	Hiester [Hie18]	Chen [CLJ$^+$18]	Wang [WQW$^+$18]	Wang [WSS18]	Vulinovic [VIP$^+$19]
pdf	231232	268071071	x	x	x	x		x	x	x	x
html	214568	25710908	x	x	x	x		x	x	x	x
jpg	109233	73242253	x	x	x	x	x	x	x	x	x
txt	78286	99435540	x		x	x			x	x	x
doc	76616	60654930	x	x	x	x		x	x	x	x
xls	62635	58718224	x	x	x	x		x	x	x	x
ppt	49702	251210471	x	x	x	x		x	x	x	x
gif	36302	5962516	x	x	x	x	x	x	x	x	x
xml	33458	16954875	x	x	x	x	x	x	x	x	x
ps	22015	56547464	x	x	x	x			x	x	x
csv	18360	6843009	x	x	x	x	x	x	x	x	x
gz	13725	17748905	x	x	x	x		x	x	x	x
log	9976	8467819	x			x		x		x	
eps	5191	5756138	x	x							
unk	5186	2983922								x	
png	4125	2207489	x	x	x	x		x	x	x	x
swf	3476	3798321	x	x	x				x	x	x
dbase3	2601	38972	x							x	
pps	1619	7432480	x	x	x						
rtf	1125	958239	x		x			x	x	x	x
kml	993	309422	x								
kmz	943	549462	x								
text	839	1527118		x				x			
hlp	659	8692	x								
f	602	94543	x							x	
sql	462	244634	x	x	x						
wp	364	87643	x								
dwf	299	85500	x								
java	292	14530	x	x	x	x		x		x	
pptx	215	1151796	x	x	x	x			x		x
tex	163	10520			x						
docx	163	65969		x	x	x		x	x		x
bmp	72	62686		x	x	x					
pub	55	1421		x							
xlsx	37	12910		x	x	x			x		x
zip	10	31525		x	x						
ttf	10	1540		x							
xbm	8	578		x							
js	2	36		x							

4. EXPLORATORY RESEARCH OF MODELS BASED ON ACCURACY

In this chapter, to address the first research question “**How do different neural network models compare to each other in terms of training performance and quality of results for file fragment classification?**”, different neural networks are trained and validated at the file fragment identification task. Their accuracy is then compared, to identify which models should be considered or disregarded.

4.1 Method

This study uses the Govdocs1 dataset [GFRD09], which was fully downloaded and then organized by file extension. This dataset has files with 63 different extensions. The 33 extensions with less than 200 files were discarded, to ensure the training and validation dataset would have at least 100 files of each file type. The “text” and “unk” extensions were discarded because examination showed that files with these extensions use multiple formats and they do not correspond to a single file type. The remaining 28 extensions are listed in Table 3.2.

To select a sample instance from the Govdocs1 dataset, first, a random file is selected among those available, without replacement. Then, a block from this file is randomly selected. After all files have participated in the process, the process may be repeated as long as necessary. This way, all files are considered and the classes are easily balanced. This contrasts with the sampling technique applied in other works, where all the sectors are grouped before sampling, which may lead to an imbalance between classes.

The input features of the neural network for each instance are a 512x256 matrix representing only one block of 512 bytes of a random file of the dataset. Each of the 512 bytes is one-hot encoded, meaning that its value is converted into a vector with 256 elements, with only one of them set to 1, corresponding to the value of the byte, while the others are set to zero.

The output of the neural network for a given instance is a vector with a size equal to the number of classes, subject to a softmax function, which applies the exponential function on the vector and then normalizes it. Each value will represent the predicted probability that the instance belongs to a specific class.

The neural networks under consideration used different combinations of convolutional, max pooling, LSTM, and fully connected layers, applying categorical cross-entropy as the loss function. Binary cross-entropy and mean squared error were considered during initial tests, but categorical cross-entropy gave faster training times.

Fourteen neural network models participated in this evaluation. Their names indicate the types of layers that compose their architecture, using the letter “C” to indicate a convolutional layer, “D” to indicate a dense layer, also called fully-connected, “L” to indicate an LSTM layer, and “M” to indicate a max pooling layer. The convolutional layers do not use padding. Table 4.1 lists the parameters of the models, which can be analyzed in more detail in the code repository [1]. One of then, “D” is a simple single-layer perceptron. Two of them, “LD” and “LL”, use LSTM layers without convolutional layers, while three of them, “CD”, “CM”, and “CCM”, use convolutional layers without LSTM layers. Eight models, “CL”, “CML”, “CLL”, “CLD”, “CCL”, “CCLL”, “CMCML”, and “CMCMLL”, combine convolutional layers and LSTM layers.

Table 4.1 – Details of 14 models trained on 28 classes

Model	Convolutional layers			Max pooling			LSTM		Dense
	output size	window size	stride length	pool size	stride length	channel first	output size	return sequences	output size
D									28
LD							32	no	28
LL							32 28	yes no	
CD	64	64	8						28
CM	28	32	1	481	1	no			
CCM	28 28	32 2	1 2	 240	 1	 no			
CL	28	32	32				28	no	
CML	32	32	32	2	2	yes	28	no	
CLL	32	32	32				64 28	yes no	
CLD	256	16	16				128	no	28
CCL	128 64	8 8	8 8				 28	 no	
CCLL	128 64	8 8	8 8				64 28	yes no	
CMCML	128 64	8 8	8 8	2 2	2 2	yes yes	 28	 no	
CMCMLL	128 64	8 8	8 8	2 2	2 2	yes yes	64 28	yes no	

All the training sessions used the Adam [KB14] optimization algorithm to guide backpropagation, which was selected because it showed good learning speed in the preliminary tests without requiring fine-tuning of parameters.

The models were trained until no further improvement was observed in the last 10 epochs, using categorical cross-entropy loss.

After the 14 models were compared, one of the models with best accuracy results and faster training time, “CLD”, was chosen to have its results compared with other works, using a longer training session than those used to assess the 14 models.

[1] http://github.com/atilaromero/carving-experiments

In multiclass classification problems, accuracy is calculated as the number of correctly classificated instances divided by the total number of instances.

Using this extended training parameters, the “CLD” model was trained with the 28 file types used in the first stage. Then, a different model was built from scratch using the same file types of each of the most recent works listed in Table 3.1: Hiester [Hie18], Chen *et al.* [CLJ+18], Wang *et al.* [WQW+18], Wang *et al.* [WSS18], and Vulinović *et al.* [VIP+19].

This comparison has three important restrictions. Chen used 4096 bytes for the block size, while this work used 512 bytes only. Vulinović got an F1-score[2] of 81% for the Multilayer Perceptron (MLP) fed with byte histograms, a better result than the one obtained with a Convolutional Neural Network (CNN) fed with raw bytes, 61%. But the comparison with the “CLD” model of this study was based on Vulinović results with the CNN, not the MLP, because it is a more similar solution, one that uses raw bytes instead of byte histograms. Additionally, his results use F1-score, while this work measured accuracy.

4.2 Results

The experiments did not take advantage of GPU acceleration and were conducted on a single computer with 256GB of RAM and with 2 Intel®Xeon®E5-2630 v2 processors, with 6 cores each, with 2 hyper-threads per core, or 24 hyper-threads in total.

The main software and frameworks used to build the experiments were Python 3.6 [Ros19], Jupyter notebook [Pé19], Tensorflow 1.14.0 [Bra19], Keras 2.2.4-tf [Cho19], and Fedora Linux 27. The source code for the experiments is available at http://github.com/atilaromero/carving-experiments.

To prepare the Govdocs1 dataset [GFRD09], 200 files from each one of the 28 chosen file types were randomly selected, 100 to use in the training dataset and 100 to use in the validation dataset. A batch size of 100 was used, with 28 steps per epoch.

The two models that used LSTM layers without convolutional layers presented slower learning in comparison with the others, as can be seen in Figure 4.1. The final validation accuracy values can be consulted in Figure 4.2, in which the bar plot shows the validation accuracy of the considered models. Their training time, in minutes, and the total epochs processed is also indicated. The training was interrupted when no further improvement was observed for 10 consecutive epochs.

The validation accuracy values of the remaining models were in the 35%-54% range. To check if the two slower models could give better results if executed for a longer period, they were trained for 10 hours. The model identified as “LD”, which uses an LSTM

[2] In binary classification problems, F1-score is the harmonic mean between precision and recal. In this context, precision is the number of true positives divided by the sum between true positives and false positives, while recall is the number of true positives divided by the sum of true positives and false negatives.

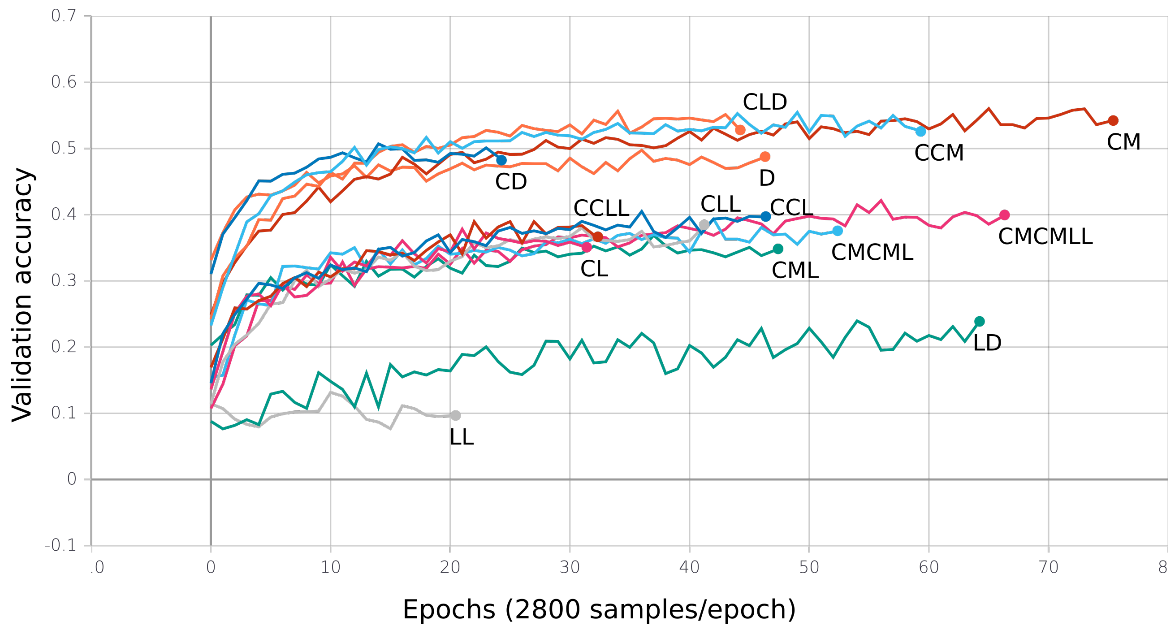

Figure 4.1 – Learning curves of some models (validation accuracy vs. time)

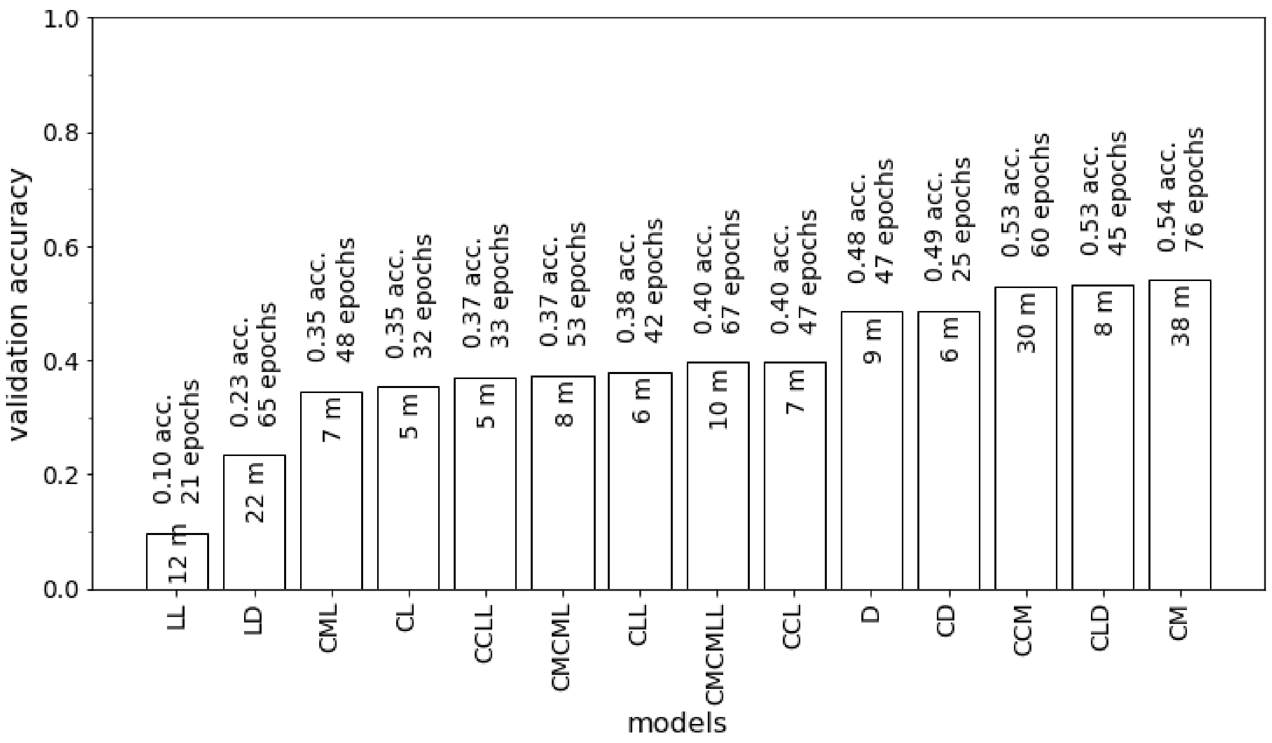

Figure 4.2 – The bar plot shows the validation accuracy of the considered models. Their training time, in minutes, and the total epochs processed is also indicated. The training was interrupted when no further improvement was observed for 10 consecutive epochs.

layer followed by a fully connected layer, was able to achieve an accuracy of 49.4%, while the other, “LL”, achieved only 15.3%.

The model “CLD” was chosen to be compared with other works because it was among the three models with the highest accuracy, but achieved its results in less time and showed high resilience to changes in parameters during the tests, while the others did not.

In a second stage, a longer training session was performed, using a batch size of 300 samples instead of 100, and increasing the improvement threshold of the early stopping condition (maximum number of epochs to wait for improvements) from 10 to 20. These parameters were chosen by trial and error. For the initial 28 file types, the “CLD” model training time increased from eight minutes to three hours using these parameters.

Using longer training sessions, the accuracy for the 28 file types used in the first stage increased from 54% to 63%. Using the same file types of other works, the “CLD” model achieved an accuracy of 67% for the file types from Chen *et al.* [CLJ+18], 91% for Hiester [Hie18], 59% for Wang *et al.* [WQW+18], 65% for Wang *et al.* [WSS18], and 61% for Vulinović *et al.* [VIP+19]. The results can be seen in Figure 4.3, in which the bar plot shows the comparison between the validation accuracy obtained with “CLD” and those achieved in other studies.

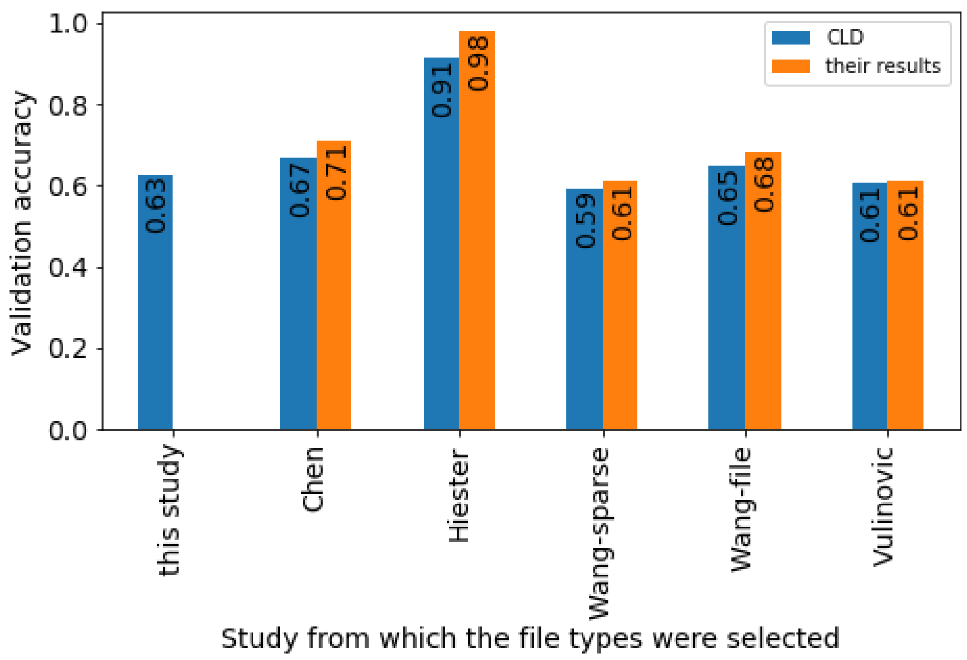

Figure 4.3 – The “CLD” model was trained from scratch using the same file types used in other works. The bar plot shows the comparison between the validation accuracy obtained with “CLD” and those achieved in other studies.

4.3 Discussion

This chapter presented some alternative models in the file fragment classification task. The idea was to identify the most promising models for improvement. But an apparent limit was found on how far these models could be improved.

Answering the first research question "**How do different neural network models compare to each other in terms of training performance and quality of results in file fragment classification?**", the results may be grouped into three sets: i) the models that used LSTM without a convolutional layer performed poorly; ii) the models with convolutional layers that used LSTM as the last layer had intermediary results; iii) the remaining models, which achieved better results, were the single-layer perceptron (single fully-connected layer model, identified as "D") and the models that used convolutional models and used a fully-connected layer or max pooling as the last layer.

The three models with best accuracy results are identified as "CCM", "CM", and "CLD", but the latter had faster training time and also showed good resilience to changes: during preliminary tests and later when test conditions were altered to try to improve results, this model and its variations were always among the best models.

In this stage of the research, the models were not trained to exhaustion. This was initially done to identify which models would be the most promising for future testing, but further attempts to tune layer types, quantity and parameters resulted in accuracy values still close to 60%. Most models required short training times and few examples to approach their limits. This suggests that some patterns were very easy to find but they were insufficient to achieve higher accuracy. This raises the question of whether there are harder patterns that could be found by a better model not yet tried or if this a more fundamental issue that would not be solved by a trial and error approach of tweaking of parameters and layer modifications.

However, the 14 models used in this study represent a very small sample of all possible model architectures using the chosen layer types. Moreover, there is no certainty that in different circumstances the best performing models will continue to outperform the others. For those reasons, the search for an alternative model could also be a valid research direction. Since the proposed models use a small number of layers and considering that deep networks have shown good results in other applications, a study that increases the number of layers of the models presented here may improve results.

A comparison was made between a chosen model, "CLD", and recent works in the field. While "CLD" achieved slight lower accuracy values than the other studies, it can be noticed that the values are similar: while the result values of these other works range from 61% to 98%, a difference of 37%, the differences between the results from this study and theirs do not exceed 7%. This is an indication that the high variability that these studies show between each other is caused by the difference in the file types they choose. This supports the claim that advances in this area should come from error analysis.

The Govdocs1 dataset brought an important basis for comparing different carving solutions. But to achieve easily reproducible results, the models must also be publicly available, a condition not all revised studies fulfill. Being available as Jupyter notebooks at https://github.com/atilaromero/carving-experiments, the results described here should require little effort to be reproduced. Thus the source code can be used as a basis of comparison in future researches, generating models with the same architecture of those presented here.

5. DESCRIPTIVE RESEARCH ON INFLUENCE OF NUMBER OF CLASSES

In Chapter 4, it was observed that achieving accuracy values above 60% on file fragment classification using 28 classes was a difficult task to the considered models. In other studies results [Hie18] [SZ12] [ATM13] [MSR14] and on initial tests, the achieved accuracy for a small number of classes was considerably higher. Hiester [Hie18], for instance, achieves an accuracy of 98% using four classes.

To address the second research question "**How does the accuracy of neural network models changes relative to the number of classes in file fragment classification?**", this chapter describes several neural network training sessions using different sets of file types and the attempt to understand how the accuracy of the resulting models are affected by composition of file types in the set.

5.1 Method

The model architecture used in this research stage, "CLD", is described in the previous chapter and has three layers. The first is a convolutional layer with 256 output units, a window of size 16, and a stride of 16. The second is an LSTM layer of 128 units. The last is a fully-connected layer where the number of output units matches the number of classes of the input dataset. The same dataset handling, sampling, output, and input encoding described in Chapter 4 were used here.

All the training sessions used the Adam [KB14] optimization algorithm to guide backpropagation, which was selected because it performed well in the preliminary tests without requiring fine-tuning of parameters.

The models were trained until no further improvement was observed in the last 10 epochs, using categorical cross-entropy loss.

5.1.1 Accuracy vs. number of classes

Several independent models with the same general architecture, but different weights, were trained with 2 to 28 extensions from the Govdocs1 dataset. For each number in the range 2 to 28, a random subset of extensions was selected to compose the dataset. Each extension has 200 file samples, half placed in the training dataset and half in the validation dataset. Each dataset was used to train a new model, that should distinguish between the selected subsets of extensions in that filtered dataset. This process was repeated five times.

5.1.2 Accuracy of pairs of classes

In addition to this sampling of extensions using different quantities of classes, all 378 combinations of pairs of extensions were compared. Each pair was used to compose a dataset and to train a new model for each one, using the same process described in the previous section.

Given the number of different models trained (135 for the N classes phase and 378 for the pairs), it was important to use a model architecture that had a fast training time. Models trained with the chosen architecture took about 10 minutes to train. Without this short training period, this research part would not be feasible.

5.1.3 Principal Component Analysis

The accuracy obtained after the training of models with pairs of classes was used to build a 28x28 matrix. Here the accuracy of the model trained for that particular pair of extensions was used as a distance measure of how similar those file types are. If a model was able to achieve 100% accuracy, the file types would be considered different, if the accuracy is 50%, they would be considered indistinguishable, as 50% is the average accuracy of a random guess classifier for two classes.

Following this concept, the diagonal entries of the matrix were filled with the value 0.5, as each class should be indistinguishable from itself. Then, a Principal Component Analysis (PCA) [ATB08] was used to treat this matrix. The 28x28 matrix can be interpreted as 28 vectors of 28 dimensions. For each class, the vector indicates how similar that class is to each of the others. This could be plotted in a space of 28 dimensions if such representation could be built. What PCA does is to reduce the dimensionality of this space. Here the dimensions were reduced to two, to be shown in a 2D graph. To do that, it chooses as first axis the one that maximizes the variance in the data and then does the same for the second dimension. That produces a projection in which the points are as farther away as possible.

While it may not be obvious how to attribute meaning to the resulting coordinate system, this technique is useful to visually group classes with similar characteristics.

5.2 Results

The experiments did not take advantage of GPU acceleration and were conducted on a single computer with 256GB of RAM and with 2 Intel®Xeon®E5-2630 v2 processors, with 6 cores each, with 2 hyper-threads per core, or 24 hyper-threads in total.

The main software and frameworks used to build the experiments were Python 3.6 [Ros19], Jupyter notebook [Pé19], Tensorflow 1.14.0 [Bra19], Keras 2.2.4-tf [Cho19], and Fedora Linux 27. The source code for the experiments is available at http://github.com/atilaromero/carving-experiments.

5.2.1 Accuracy vs. number of classes

Figure 5.1 shows the graph of accuracy versus number of classes. Each of those points is a model trained with the indicated number of classes. The class for a block sample is the file extension of the original file. The classes/extensions used to train each model were taken at random. The bottom line indicates for comparison how accurate a random guess classifier would be. The points for two classes use transparency to improve visualization, as many of them are too close. For 2 classes, the accuracy values are between 50% and 100%, with a greater concentration near 100%. For 28 classes, the accuracy values are between 55% and 58%.

The lines “hard file types first” and “easy file types first” were created by selecting the file types that would compose the dataset. The selection was based on the results described in the next subsection, “Accuracy of pairs of classes”, ordering the file types by their minimum accuracy values and following this order to choose file types. For example, the “hard file types first” point for three classes was obtained by training a model with the file types “pps”, “ppt”, and “gz”, the three classes with lower minima in Figure 5.2.

5.2.2 Accuracy of pairs of classes

Figure 5.2 shows the graph of the accuracy of each class when compared individually with each one of the others, resulting in 378 models, one for each possible pair of classes. File types where all the points are close to 100% are hardly mistaken for other types, while file types presenting accuracy values close to 50% can be mistaken for other file types by the model.

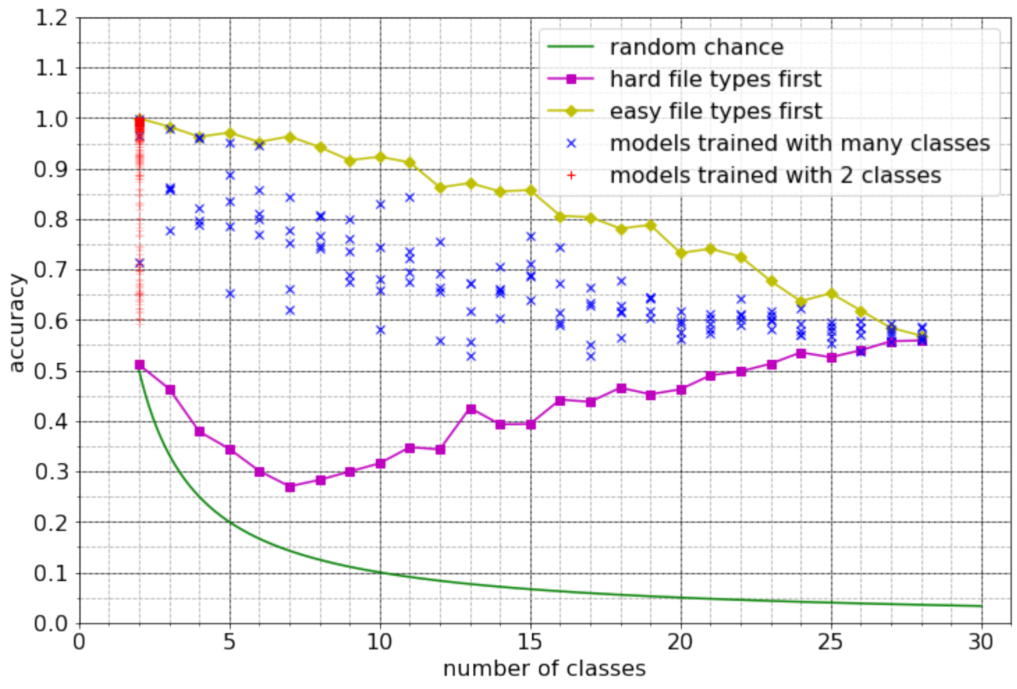

Figure 5.1 – Validation accuracy by number of classes

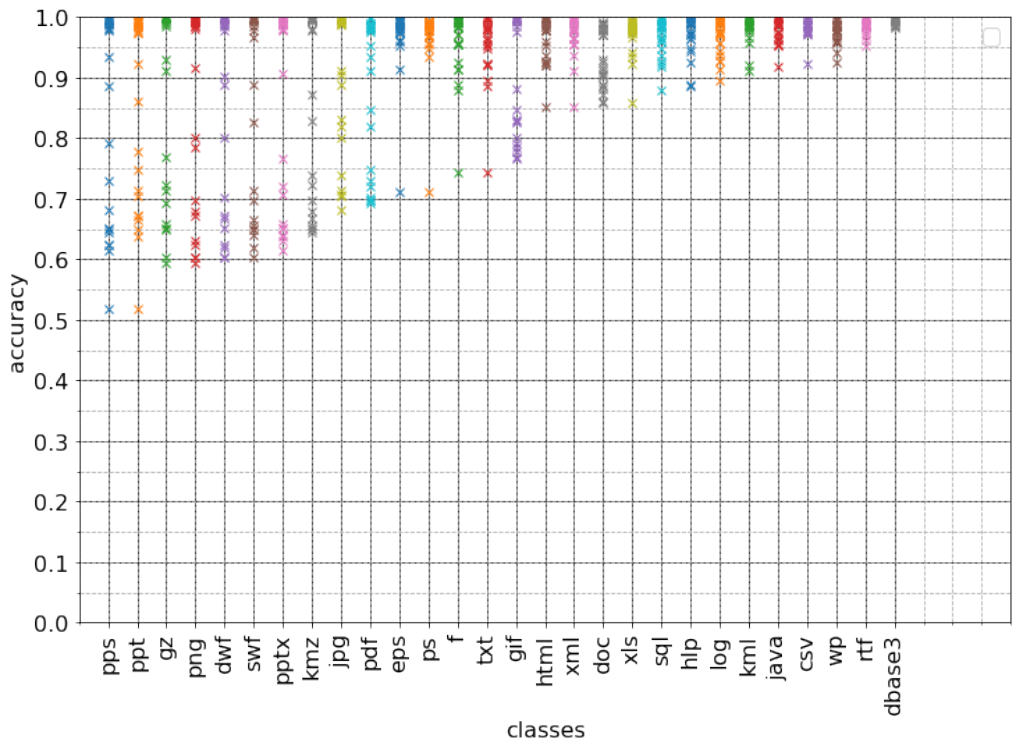

Figure 5.2 – Validation accuracy of models trained with pair of classes

5.2.3 Principal Component Analysis

A 28x28 matrix was made using the accuracy of the models built for each pair of classes as a distance measure. A PCA dimensionality reduction technique was used to plot this data in a 2D graph, grouping similar file types. The result is shown in figures 5.3 and 5.4. Similar file types are next to each other, while those that can be easily distinguished by the models are represented by points that are distant from each other.

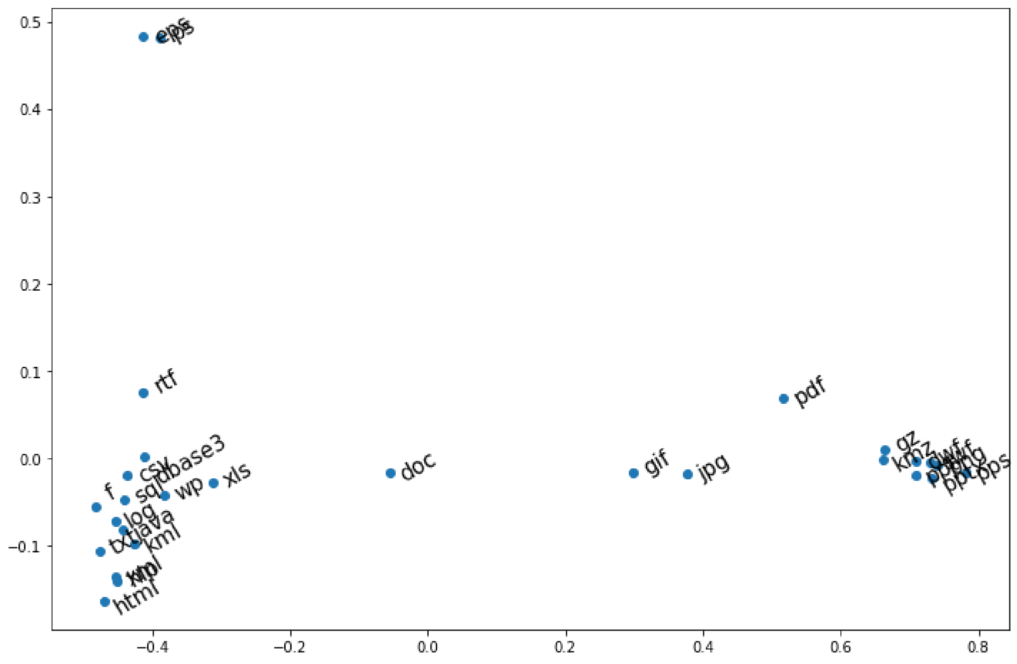

Figure 5.3 – PCA of accuracy of models trained with pair of classes

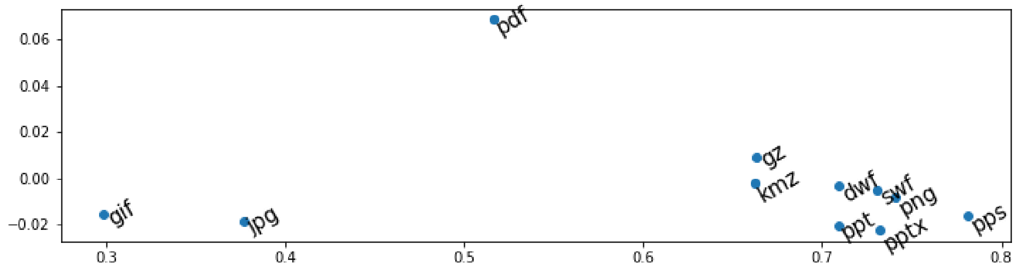

Figure 5.4 – PCA of accuracy of models trained with pair of classes - lower right detail

The data fed to the PCA algorithm is shown in Table 5.1.

5.3 Discussion

The results of this Chapter highlight the importance of the file types chosen to compose the dataset. File types that hold compressed data or images are harder do classify. This suggests that their entropy is a source of classification errors. This also may explain why previous works in the field have achieved different results but the “CLD” model was able to almost match their results, as shown in Figure 4.3.

5.3.1 Accuracy vs. number of classes

In Figure 5.1, a decreasing trend was observed. An increase in the number of classes appears to be correlated with a decrease in accuracy. Another relevant aspect of the graph is that the range of the results seems to be smaller when more classes are used.

This pattern is understandable: as the number of classes grows, the harder the classification problem is, leading to a decrease in accuracy. Meanwhile, the individual contributions of each class to the overall result diminish, leading to a decrease in variation between the different results for a given number of classes.

This behavior is an important aspect to consider during the evaluation of file fragments studies. This observation is in agreement with Beebe *et al.* [BMLS13] observation, that the studies that select fewer classes tend to yield higher results.

Still, with 42% [1] of samples being misclassified when the number of classes is 28, the question of what are the error sources and how they can be addressed requires attention.

The number of possible combinations of file types to compose the datasets depends on the number of classes being considered. For 28 classes, there is only one possible combination, while for two classes there are 378. For intermediary values, the numbers are much higher, which is the number of possible combinations disregarding the order of the elements: $\frac{28!}{(28-n)!n!}$. For 14 file types, there are 40,116,600 combinations. For this reason, the significance of the 5 samples diminishes for intermediary values.

5.3.2 Accuracy of pairs of classes

The accuracy of models trained with pairs of classes, shown in Figure 5.2, suggests a reverse correlation between entropy and accuracy. Generally, file types with higher entropy

[1]In the extended training session described in Chapter 4, a higher accuracy was obtained and 38% of samples were misclassified, instead of 42%.

tend to have lower minima, with the GIF file type being a notable exception. Most of these files use some form of compression, as image files for example.

It was demonstrated that the accuracy of a new model may be manipulated by the selection of file types that will compose the dataset. The lines “hard file types first” and “easy file types first” of Figure 5.1 were created using the order shown in Figure 5.2, resulting in lines that seem to be close to the minimum and maximum of the possible accuracy values.

5.3.3 Principal Component Analysis

The usage of PCA on the 28x28 distance matrix produced a 2D projection where, as shown in figures 5.3 and 5.4, a group of file types that use compression or contains images are grouped near each other: “gif”, “jpg”, “pdf”, “gz”, “kmz”, “dwf”, “ppt”, “swf”, “png”, “pptx”, and “pps”.

5.3.4 Final considerations

Answering the second research question “**How does the accuracy of neural network models changes relative to the number of classes in file fragment classification?**”, it was observed that an increase in the number of extensions selected to compose the training tends to decrease accuracy and to decrease variation in results. But the number of classes alone is not as important as the type of extension selected: some file types when included in the experiment have a much higher negative impact than others. This observation was demonstrated in the “hard file types first” and “easy file types first” of Figure 5.1, where the file types selected to compose were intentionally chosen, once to degrade results and once to improve them.

File types that contain images or that use compression were identified as those that have the highest negative effect on results, which suggests that their entropy may contribute to the error.

The number of samples taken was small when compared to the number of all possible file types combinations. This imposes a limit on the conclusions that can be reached, and this limitation is hard to overcome.

The group that emerged as file types that most degrade results are files that use compression or contain images. While they are known for their high entropy, no measure of entropy was used to reinforce this claim. This aspect is addressed in the next chapter.

Table 5.1 – Data fed to the PCA algorithm

	csv	dbase3	doc	dwf	eps	f	gif	gz	hlp	html	java	jpg	kml	kmz	log	pdf	png	pps	ppt	pptx	ps	rtf	sql	swf	txt	wp	xls	xml
csv	0.50	1.00	0.99	1.00	0.98	0.97	1.00	1.00	0.98	0.98	0.97	1.00	0.98	1.00	0.97	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.98	1.00	0.92	0.99	0.99	0.98
dbase3	1.00	0.50	0.99	1.00	1.00	1.00	1.00	1.00	0.99	0.99	0.99	1.00	0.99	1.00	1.00	1.00	1.00	1.00	0.99	1.00	0.99	1.00	0.98	1.00	1.00	1.00	0.99	1.00
doc	0.99	0.99	0.50	0.90	0.98	0.97	0.88	0.93	0.97	0.98	0.99	0.90	0.98	0.87	0.99	0.91	0.92	0.89	0.86	0.91	0.97	0.99	0.98	0.89	0.98	0.93	0.86	0.98
dwf	1.00	1.00	0.90	0.50	1.00	0.99	0.80	0.60	1.00	1.00	1.00	0.89	1.00	0.67	0.99	0.70	0.60	0.62	0.67	0.65	0.98	1.00	0.99	0.62	1.00	0.99	0.98	1.00
eps	0.98	1.00	0.98	1.00	0.50	0.91	1.00	0.99	0.99	0.98	0.99	0.99	1.00	0.99	0.98	0.95	0.99	0.98	0.97	0.99	0.71	0.96	0.96	0.99	0.98	0.99	0.98	0.98
f	0.97	1.00	0.97	0.99	0.91	0.50	1.00	1.00	0.89	0.93	0.95	0.99	0.99	0.99	0.91	0.99	0.99	1.00	0.99	0.99	0.96	0.99	0.88	1.00	0.74	0.96	0.98	0.95
gif	1.00	1.00	0.88	0.80	1.00	1.00	0.50	0.77	1.00	1.00	1.00	0.83	1.00	0.83	0.99	0.85	0.78	0.79	0.78	0.77	0.99	1.00	0.99	0.83	1.00	0.98	0.97	1.00
gz	1.00	1.00	0.93	0.60	0.99	1.00	0.77	0.50	1.00	1.00	1.00	0.91	1.00	0.72	1.00	0.69	0.59	0.65	0.71	0.66	0.98	0.99	1.00	0.65	1.00	0.99	0.99	1.00
hlp	0.98	0.99	0.97	1.00	0.99	0.89	1.00	1.00	0.50	0.93	0.96	0.99	0.99	1.00	0.94	1.00	0.99	0.99	0.99	0.99	0.99	0.99	0.95	1.00	0.89	0.97	0.97	0.96
html	0.98	0.99	0.98	1.00	0.98	0.93	1.00	1.00	0.93	0.50	0.95	1.00	0.92	1.00	0.93	0.99	0.99	0.99	0.99	1.00	0.98	0.98	0.94	1.00	0.92	0.96	0.98	0.85
java	0.97	0.99	0.99	1.00	0.99	0.95	1.00	1.00	0.96	0.95	0.50	1.00	0.98	1.00	0.97	0.99	1.00	1.00	0.99	1.00	0.99	0.99	0.92	1.00	0.96	0.98	1.00	0.96
jpg	1.00	1.00	0.90	0.89	0.99	0.99	0.83	0.91	0.99	1.00	1.00	0.50	1.00	0.74	0.99	0.82	0.80	0.68	0.70	0.71	0.99	1.00	1.00	0.71	1.00	0.99	0.99	1.00
kml	0.98	0.99	0.98	1.00	1.00	0.99	1.00	1.00	0.99	0.92	0.98	1.00	0.50	1.00	0.96	0.98	0.99	0.99	0.99	1.00	0.99	0.99	0.99	1.00	0.97	1.00	0.99	0.91
kmz	1.00	1.00	0.87	0.67	0.99	0.99	0.83	0.72	1.00	1.00	1.00	0.74	1.00	0.50	0.99	0.70	0.68	0.64	0.65	0.65	0.98	1.00	1.00	0.66	1.00	0.99	0.98	1.00
log	0.97	1.00	0.99	0.99	0.98	0.91	0.99	1.00	0.94	0.93	0.97	0.99	0.96	0.99	0.50	0.98	1.00	0.99	1.00	0.99	0.96	0.97	0.93	1.00	0.89	0.98	0.97	0.94
pdf	1.00	1.00	0.91	0.70	0.95	0.99	0.85	0.69	1.00	0.99	0.99	0.82	0.98	0.70	0.98	0.50	0.70	0.73	0.75	0.72	0.93	0.99	0.98	0.70	0.98	0.98	0.98	0.98
png	1.00	1.00	0.92	0.60	0.99	0.99	0.78	0.59	0.99	0.99	1.00	0.80	0.99	0.68	1.00	0.70	0.50	0.62	0.67	0.63	0.98	1.00	0.99	0.60	0.99	0.98	0.98	1.00
pps	1.00	1.00	0.89	0.62	0.98	1.00	0.79	0.65	0.99	0.99	1.00	0.68	0.99	0.64	0.99	0.73	0.62	0.50	0.52	0.61	0.98	0.99	0.99	0.65	1.00	0.98	0.93	1.00
ppt	1.00	0.99	0.86	0.67	0.97	0.99	0.78	0.71	0.99	0.99	0.99	0.70	0.99	0.65	1.00	0.75	0.67	0.52	0.50	0.64	0.99	0.98	0.97	0.66	0.98	0.97	0.92	0.99
pptx	1.00	1.00	0.91	0.65	0.99	0.99	0.77	0.66	0.99	1.00	1.00	0.71	1.00	0.65	0.99	0.72	0.63	0.61	0.64	0.50	0.99	1.00	0.99	0.64	0.98	0.98	0.98	0.99
ps	0.99	0.99	0.97	0.98	0.71	0.96	0.99	0.98	0.99	0.98	0.99	0.99	0.99	0.98	0.96	0.93	0.98	0.98	0.99	0.99	0.50	0.97	0.97	0.98	0.95	0.98	0.99	0.99
rtf	0.99	1.00	0.99	1.00	0.96	0.99	1.00	0.99	0.99	0.98	0.99	1.00	0.99	1.00	0.97	0.99	1.00	0.99	0.98	1.00	0.97	0.50	0.98	1.00	0.95	0.98	0.99	0.98
sql	0.98	0.98	0.98	0.99	0.96	0.88	0.99	1.00	0.95	0.94	0.92	1.00	0.99	1.00	0.93	0.98	0.99	0.99	0.97	0.99	0.97	0.98	0.50	1.00	0.92	0.96	0.99	0.96
swf	1.00	1.00	0.89	0.62	0.99	1.00	0.83	0.65	1.00	1.00	1.00	0.71	1.00	0.66	1.00	0.70	0.60	0.65	0.66	0.64	0.98	1.00	1.00	0.50	0.99	0.99	0.97	1.00
txt	0.92	1.00	0.98	1.00	0.98	0.74	1.00	1.00	0.89	0.92	0.96	1.00	0.97	1.00	0.89	0.98	0.99	1.00	0.98	0.98	0.95	0.95	0.92	0.99	0.50	0.96	0.98	0.96
wp	0.99	1.00	0.93	0.99	0.99	0.96	0.98	0.99	0.97	0.96	0.98	0.99	1.00	0.99	0.98	0.98	0.98	0.98	0.97	0.98	0.98	0.98	0.96	0.99	0.96	0.50	0.94	0.97
xls	0.99	0.99	0.86	0.98	0.98	0.98	0.97	0.99	0.97	0.98	1.00	0.99	0.99	0.98	0.97	0.98	0.98	0.93	0.92	0.98	0.99	0.99	0.99	0.97	0.98	0.94	0.50	0.98
xml	0.98	1.00	0.98	1.00	0.98	0.95	1.00	1.00	0.96	0.85	0.96	1.00	0.91	1.00	0.94	0.98	1.00	1.00	0.99	0.99	0.99	0.98	0.96	1.00	0.96	0.97	0.98	0.50

6. EXPERIMENT ON RANDOM DATA DETECTION

In the previous chapter, it was observed that file types that use compression or contain images have a higher negative impact on accuracy than other file types do. This raises the question of whether high entropy may be related to the observed errors.

In this work, the entropy of a data block can be defined as the amount of information required to describe that data. This is closely related to how compressible the data is: it can be said that a block of data that can be compressed into a small size has low entropy, as it is being described with fewer data. The presence of structures in the data makes it easier to describe and to compress, thus reducing its entropy. Conversely, a block of random data is not easy to describe and hardly compressible, so it is considered to have high entropy.

Before conducting experiments related to the cause of errors, a list of conceivable error sources (E1 to E5) was elaborated:

E1. For some data structure, the model cannot distinguish it from random data. This can happen if the pattern in the data is too complex, beyond the capabilities of the model. It may be the case that in practice no model can perform this distinction or, instead, this may be a limitation of this particular model only. This situation may occur in files that use compression or cryptography, or more generally, any file type with high entropy.

E2. Different file types using different data structures, but the model cannot distinguish between them. In this case, the model can perceive that there are structures in the data, but it is not able to differentiate them.

E3. Different file types using the same data structure. It is common for a given file type to employ different types of data structures. If two or more file types make use of the same data structure, the existence of that structure will then not be sufficient to differentiate those file types, as it may belong to any of them. The reverse, a file type that uses multiple kinds of data structures, does not constitute a problem as it simply results in extra work during the model training.

E4. Same file type with multiple extensions. If the same file type appears in the dataset with multiple extensions, “JPG” and “JPEG” for example, the model will not be able to predict which label is used in the validation dataset for a given instance, since this distinction exists in the labeling, but not in the content.

E5. Files that contain other files. Some file types need to embed other files. If the inner file is categorized, even if the model finds an unmistakable pattern, it will not match the label, which will be the extension of the outer file. A “PDF” file, for example, can embed a “JPG” file. If the model predicts “JPG” as the class of a portion of this file, it will not match the instance label on the dataset, which would be “PDF”.

The first error source, E1, is the focus of this chapter. In the file fragment classification task, it is expected that some portion of the classification errors of a given model be caused by its inability to recognize patterns in the data if the data have high entropy. In these situations, even humans may be unable to tell if a sample is valid or if it is random data. A search for an alternative model may lead to better results, but the chances are low, as this type of error is hard to mitigate.

In the second error source, E2, it is conceivable that avoiding misinterpretation of one structure as another is possible since the model has successfully detected structures. This error source is not explored in this study. Mitigation of this type of error probably would be achieved through more training or by a search for a better model.

The three last error sources listed above, E3 to E5, are not explored in this study. They are similar in nature, as the last two may be viewed as special cases of E3. This type of error is better mitigated through a revision of the labeling system.

The goal of the experiment of this chapter is to test the implicit hypothesis of the third research question, *i.e.*, **that part of the file fragment classification errors can be explained by the inability of the models to distinguish high entropy data from random data**. Not all file types have this kind of data, and for those that do, they should compose only a portion of the file.

6.1 Method

For this experiment, a new model is created for each considered file type. The input of the model is a one-hot encoded block than may come from a random generator or from the Govdocs1 dataset. Samples from the Govdocs1 dataset are labeled “structured”, while samples from the random generator are labeled “random”. The model should predict from which source the sample came.

The 28 file types from the Govdocs1 dataset [GFRD09] used in chapters 4 and 5 were used in this experiment, placing 100 files in the training dataset and 100 in the validation dataset.

The model architecture used in this research stage, “CLD”, is described in Chapter 4 and has three layers. The first is a convolutional layer with 256 output units, a window of size 16, and a stride of 16. The second is an LSTM layer of 128 units. The last is a fully-connected layer where the number of output units matches the number of classes of the input dataset.

All the training sessions used the Adam [KB14] optimization algorithm to guide backpropagation. The stopping condition used is 10 epochs without improvement in validation accuracy, using categorical cross-entropy loss.

6.1.1 From data source to measure of entropy

There is a difference in the nature of the predetermined labels and the predicted ones. The predetermined labels reflect the data source from which the samples were extracted, while the predicted labels are based on the content of the data. For the random generator, this is not a problem, as this data source only has one kind of data. But the samples from the Govdocs1 dataset may have two kinds of data. For some of the samples, it is expected that the true classification should be “random” instead of “structured”. These are the samples that are too complex to this kind of model, the ones where the model cannot find any pattern. Unfortunately, there is no direct way to know the perfect labels in advance, hence all samples of the Govdocs1 dataset are labeled “structured”. Thus, it is known that the samples from Govdocs1 dataset have an unknown portion of conditions positives and negatives.

Here, the “structured” samples are considered positive and the “random” samples are considered negative. Thus the samples that are classified as “random” by the model are the sum of the true negatives and the false negatives. Similarly, the samples classified as “structured” are the sum of the true positives and the false positives.

The random generator can be used to predict the proportion between false positives and true negatives of the chosen file type from the Govdocs1 dataset. Since the condition negative (which is the sum of false positives and true negatives) is those samples that ideally should be marked “random”, the classifier should treat them the same way it treats the random generator samples.

In other words, the Govdocs1 dataset for a particular file type is assumed to have a subset of samples that should be labeled “random” because the model cannot find patterns in them. Some of those samples may be incorrectly labeled “structured” by the model. These are the false positives. But this should happen in the same proportion that the model classifies samples from the random generator as “structured”.

Using this method, the quantity that would best represent the entropy of the dataset would be $(1 - \mathit{Prevalence})$, where $\mathit{Prevalence} = \frac{\sum \mathit{ConditionPositive}}{\sum \mathit{TotalPopulation}}$. Unfortunately, without knowing the false negative rate, the best that can be achieved is a range where the maximum entropy value occurs when the false negative rate is zero. Assuming the false negative rate to be zero, the prevalence will be equal to the true positives over the entire population. Thus, the entropy range can be calculated using only the accuracy values of the trained model obtained separately on the Govdocs1 dataset and on the random dataset, as shown in equations 6.1 to 6.4.

$$1 - \frac{Entropy}{measure} = Govdocs\ True\ Positive \tag{6.1}$$

$$= Govdocs\ Predicted\ Positive - Govdocs\ False\ Positive \tag{6.2}$$

$$= GovdocsPredPOS - \left(GovdocsPredNEG * \frac{randomPredPOS}{randomPredNEG}\right) \tag{6.3}$$

$$= GovdocsKerasACC - \frac{(1 - GovdocsKerasACC) * (1 - randomKerasACC)}{randomKerasACC} \tag{6.4}$$

In equation 6.3, the (*GovdocsPredNEG*) quantity is the portion of samples marked as “random” by the model when classifying samples from the Govdocs1 dataset. Normally, *Predicted Negative* = *True Negative* + *False Negative*, but the no false negative assumption is being used. The *GovdocsKerasACC* is the accuracy calculated by Keras during validation of the Govdocs1 dataset. It is not the real accuracy since this validation considers all samples marked “random” as errors. Thus *GovdocsKerasACC* = *GovdocsPredPOS*. The accuracy calculated by Keras during validation of the random generator consider as correct the samples marked as “random”, thus *randomKerasACC* = *randomPredNEG*.

Multiplying (1 − *GovdocsKerasACC*) value by $\frac{(1-randomKerasACC)}{randomKerasACC}$ should give the portion of samples that are erroneously marked as “structured” by the model, the false positives. Here it is assumed that $\frac{False\ positive}{True\ negative}$ on the Govdocs dataset is equal to $\frac{randomPredPOS}{randomPredNEG}$ on the random generator.

After this entropy measure is calculated for all the file types, it can be used to estimate the amount of error that can be attributed to the model’s inability to recognize structures, testing the proposed hypothesis. For a dataset with a balanced file type distribution, this will be the mean of the entropy measures.

6.2 Results

The experiments did not take advantage of GPU acceleration and were conducted on a single computer with 256GB of RAM and with 2 Intel®Xeon®E5-2630 v2 processors, with 6 cores each, with 2 hyper-threads per core, or 24 hyper-threads in total.

The main software and frameworks used to build the experiments were Python 3.6 [Ros19], Jupyter notebook [Pé19], Tensorflow 1.14.0 [Bra19], Keras 2.2.4-tf [Cho19], and Fedora Linux 27. The source code for the experiments is available at http://github.com/atilaromero/carving-experiments.

Table 6.1 – Complement of entropy measure.

Category	GovdocsACC	RandomACC	Time	Epochs	GovdocsTP	GovdocsPrec
dwf	0.528	0.914	16m16s	37	0.483589	0.915888
gz	0.613	0.906	16m59s	37	0.572848	0.934499
swf	0.592	0.970	19m58s	43	0.579381	0.978685
kmz	0.589	0.983	16m21s	36	0.581892	0.987932
png	0.659	0.960	19m40s	46	0.644792	0.978440
pdf	0.674	0.977	29m59s	70	0.666325	0.988613
pps	0.748	0.962	26m12s	57	0.738046	0.986692
pptx	0.827	0.828	12m59s	30	0.791063	0.956545
ppt	0.827	0.977	28m18s	63	0.822927	0.995075
gif	0.848	0.991	16m40s	39	0.846620	0.998372
jpg	0.865	0.990	25m31s	59	0.863636	0.998424
doc	0.885	0.990	14m13s	31	0.883838	0.998687
eps	0.989	1.000	8m00s	19	0.989000	1.000000
ps	0.990	1.000	4m55s	11	0.990000	1.000000
xls	0.991	1.000	9m51s	23	0.991000	1.000000
sql	0.995	1.000	5m19s	11	0.995000	1.000000
kml	0.995	1.000	7m43s	16	0.995000	1.000000
log	0.998	1.000	5m19s	12	0.998000	1.000000
wp	0.998	1.000	5m19s	11	0.998000	1.000000
hlp	1.000	1.000	4m49s	11	1.000000	1.000000
rtf	1.000	1.000	5m17s	12	1.000000	1.000000
html	1.000	1.000	5m01s	11	1.000000	1.000000
f	1.000	1.000	4m50s	11	1.000000	1.000000
java	1.000	1.000	5m18s	11	1.000000	1.000000
txt	1.000	1.000	4m42s	11	1.000000	1.000000
csv	1.000	1.000	5m19s	12	1.000000	1.000000
xml	1.000	1.000	5m14s	11	1.000000	1.000000
dbase3	1.000	1.000	4m42s	11	1.000000	1.000000

The results obtained are shown in Table 6.1. The entropy measure is the complement of the GovdocsTP column. The GovdocsACC columns contains the accuracy validations values as calculated by Keras on the Govdocs1 dataset, thus considering all samples labeled as "random" as errors. The RandomACC column lists the portion of samples from the random generator classified as "random" by the model for that file type. The Time and Epoch columns refer to the training session of each model. The GovdocsTP column uses the equation 6.4, giving the complement of the entropy. GovdocsPrec is simply $GovdocsTP/GovdocsACC$.

Figure 6.1 shows, for each file type, the percent of 512-byte fragments identified as "structured", from a sample of 1000 fragments for each file type. In the figure, the 87% value for JPG, for example, means that up to 87% of the blocks of the JPG dataset have recognizable patterns. Since the assumption of no false negatives was used, the real value, which is unknown, can be lower. The entropy measure proposed is the complement of this value, for example, 13% for JPG.

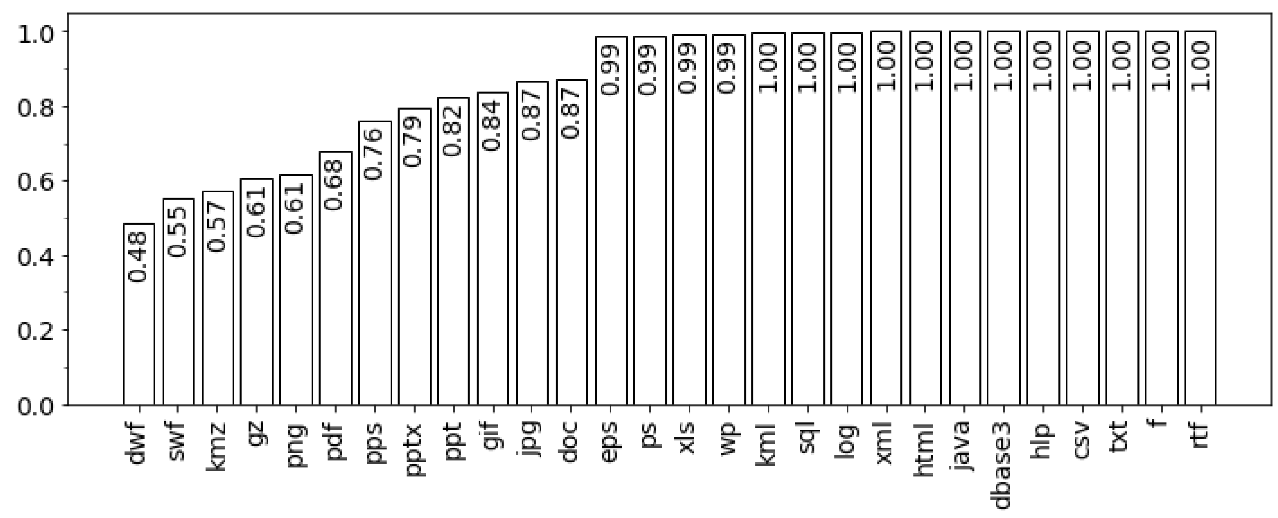

Figure 6.1 – Estimate of the percentage of samples with recognizable structures, using models trained to distinguish each file type from random data.

6.3 Discussion

Having a minimum number of recognizable structures for each file type, it is now possible to calculate the maximum number of the classification errors that can be attributed to error cause E1, which happens when the complexity of the data is beyond the model's capability to recognize patterns.

The mean true positive estimate for all file types, using data listed in Figure 6.1, is 87.3%. Assuming a dataset with balanced classes, if all the blocks considered random data were misclassified, then the maximum amount of error due to data complexity would be 12.7%. If all blocks considered random were classified as DWG, which is the class with less recognizable patterns, then the maximum amount of error would be 10.9%.

Since the higher validation accuracy obtained in Chapter 4 for these 28 file types using the "CLD" models was 63%, that model has an error rate of 37%. But only 12.7% of those errors can be attributed to the error cause E1, which happens when the model is unable to detect structures in the data.

Answering the third research question, "**Can the inability of the models to distinguish high entropy data from random data explain part of the file fragment classification errors? If so, to what extent?**", the hypothesis proposed in this chapter is only partially confirmed: some of the errors can be attributed to error cause E1, but only to some extent. About 2/3 of the errors cannot be explained by error cause E1. Since the no false negative assumption was used, 2/3 may be an underestimation.

The no false negative assumption considers that no sample with recognizable structures is erroneously classified as “random”. Thus, it is expected that the exact valid fragment count, which is unknown in practice, will be higher than the number obtained. Taking the JPG file type as an example, in which 87% of the blocks were identified as “structured”, this means that the true entropy value would be some unknown value between zero and 13%.

Also, it is important to notice that the use of different network architectures may result in different measures, as their ability to recognize patterns in the data may be different.

7. CONCLUSION

File fragment classification aims to identify the original type of file from which a given block of data was extracted. Machine learning techniques, and neural networks in particular, have the potential to improve this field, because the support of a new file type using traditional methods is laborious and not automatic. While studies on neural networks applied for file fragment classification have shown good results, to apply those contributions on real scenarios a better understanding is required on how those methods respond to an increase in number of supported file types and what are the main sources of errors of this approach.

The research described in this study was divided into three parts. In Chapter 4, during the search for the best model to classify file fragments, an apparent limit on how far these models could be improved was found. This limitation can be also observed in other works, since training the model "CLD" with the same file types used in those works resulted in accuracy values similar to theirs.

In Chapter 5, the influence of the number of classes on accuracy was explored. It was observed that an increase in the number of classes tends to decrease the accuracy and to decrease variation in results. But the number of classes alone was found to be less important than the type of extension selected: some file types when included in the experiment have a much higher negative impact than others, especially those that use compression or contain images.

In Chapter 6, a method to measure entropy was proposed. One advantage of this method is that it can be customized to a particular neural network architecture. It was used to measure, for each file type, what portion of the data did not have structures detectable by the chosen model, "CLD".

This measure was used to verify the hypothesis that part of the errors observed in Chapter 4 were caused by high entropy on some file types, as the results of Chapter 5 suggested.

The portion of data that could be identified as containing structures was higher than expected. While some of the errors could be explained by the inability of the models to distinguish high entropy data from random data, this could only explain about 1/3 of the observed errors (12.7% out of 37%).

This raises the question of whether the remaining 2/3 of errors could be explained by different file types using the same data structures. This error source comes from the practice of using the extension of the file as the class to each of its parts. An analogy with speech recognition would be to label each syllable of a spoken word using the word as its label and then try to predict the whole word using only a syllable. In this case, spoken words with same syllable would be misclassified with a high probability.

This is an important contribution because using the extension of the file as the fragment label is common practice in studies in this field. The inefficiency of this labeling system may have been overlooked because the accuracy errors were likely attributed to the existence of high entropy data. Since high entropy data can only explain a part of the misclassification errors, as this study showed in Chapter 6, the assumption that the search for a better model would be the best approach to improve file fragment classification results may be wrong because the labeling system should be reviewed first. It is possible that existing models already reached the limit of what can be done without a revision in the labeling system, as was shown in Chapter 4, where the “CLD” model achieved results similar to other studies.

Unfortunately, in the file fragmentation task the potential labels for smaller parts may be less obvious than it is in speech recognition. The best-case scenario would be if the neural network itself could choose the labeling. But evaluating those predicted labels using labels of its own choosing would introduce bias, as it could prefer to create only easy labels.

An additional contribution of this work is the availability of the source code used to generate the results, which is a feature that not all studies provide. It can be used as a basis of comparison in future researches, generating models with the same architecture of those presented here.

7.1 Future work

For future research on file fragment classification, the hypothesis that the remaining 2/3 of the observed errors are caused by similar data structures used by multiple files should be explored.

Also, the search for a procedure to automatically label inner data structures of files may be a promising strategy. The automatic identification of inner data structures is specially interesting because it could help to interpret the data.

For future research on reassembling, an LSTM that uses the file fragment classification scores as input is an option. Another potentially helpful approach to reassembling would be a neural network to predict, for a specific file type, which bytes belong to the file and which do not. That could be used to detect the start and the end of the file.

REFERENCES

[ALSH10a] Ahmed, I.; Lhee, K.; Shin, H.; Hong, M. “Content-based file-type identification using cosine similarity and a divide-and-conquer approach”, *Institution of Electronics and Telecommunication Engineers Technical Review, India*, vol. 27–6, 2010, pp. 465–477.

[ALSH10b] Ahmed, I.; Lhee, K.-s.; Shin, H.; Hong, M. “Fast file-type identification”. In: ACM Symposium on Applied Computing, 2010, pp. 1601–1602.

[AMJK18a] Ali, R.; Mohamad, K.; Jamel, S.; Khalid, S. “A review of digital forensics methods for JPEG file carving”, *Journal of Theoretical and Applied Information Technology*, vol. 96–17, 2018, pp. 5841–5856.

[AMJK18b] Ali, R. R.; Mohamad, K. M.; Jamel, S.; Khalid, S. K. A. “Classification of JPEG Files by Using Extreme Learning Machine”. In: Recent Advances on Soft Computing and Data Mining, 2018, pp. 33–42.

[AS91] Asimov, I.; Silverberg, R. “Nightfall”. New York, NY, USA: Bantam books, 1991, 352p.

[ATB08] Amirani, M. C.; Toorani, M.; Beheshti, A. “A new approach to content-based file type detection”. In: IEEE Symposium on Computers and Communications, 2008, pp. 1103–1108.

[ATM13] Amirani, M. C.; Toorani, M.; Mihandoost, S. “Feature-based Type Identification of File Fragments”, *Security and Communication Networks*, vol. 6–1, 2013, pp. 115–128.

[Axe10] Axelsson, S. “The Normalised Compression Distance as a file fragment classifier”, *Digital Investigation*, vol. 7, 2010, pp. S24–S31.

[BMLS13] Beebe, N. L.; Maddox, L. A.; Liu, L.; Sun, M. “Sceadan: Using Concatenated N-Gram Vectors for Improved File and Data Type Classification”, *IEEE Transactions on Information Forensics and Security*, vol. 8–9, Sep 2013, pp. 1519–1530.

[Bra19] Brain, G. “TensorFlow 1.14.0”. Retrieved from: https://www.tensorflow.org/, Jan 2019.

[CBS+10] Conti, G.; Bratus, S.; Shubina, A.; Sangster, B.; Ragsdale, R.; Supan, M.; Lichtenberg, A.; Perez-Alemany, R. “Automated mapping of large binary objects using primitive fragment type classification”, *Digital Investigation*, vol. 7, 2010, pp. S3–S12.

[CC08] Calhoun, W. C.; Coles, D. “Predicting the types of file fragments”, *Digital Investigation*, vol. 5, 2008, pp. S14–S20.

[Cho19] Chollet, F. “Keras 2.2.4-tf”. Retrieved from: https://keras.io/, Jan 2019.

[CLJ+18] Chen, Q.; Liao, Q.; Jiang, Z. L.; Fang, J.; Yiu, S.; Xi, G.; Li, R.; Yi, Z.; Wang, X.; Hui, L. C. “File Fragment Classification Using Grayscale Image Conversion and Deep Learning in Digital Forensics”. In: IEEE Security and Privacy Workshops, 2018, pp. 140–147.

[EM07] Erbacher, R. F.; Mulholland, J. “Identification and localization of data types within large-scale file systems”. In: Systematic Approaches to Digital Forensic Engineering, 2007, pp. 55–70.

[FMMZ12] Fitzgerald, S.; Mathews, G.; Morris, C.; Zhulyn, O. “Using NLP techniques for file fragment classification”, *Digital Investigation*, vol. 9, 2012, pp. S44–S49.

[Gar07] Garfinkel, S. “Carving contiguous and fragmented files with fast object validation”. In: Digital Forensic Research Workshop Annual Conference, 2007, pp. S2–S12.

[GFRD09] Garfinkel, S.; Farrell, P.; Roussev, V.; Dinolt, G. “Bringing science to digital forensics with standardized forensic corpora”, *Digital Investigation*, vol. 6, 2009, pp. S2–S11.

[GMH13] Graves, A.; Mohamed, A.-R.; Hinton, G. “Speech recognition with deep recurrent neural networks”. In: IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 6645–6649.

[Gre19] Grenier, C. “Photorec”. Retrieved from: https://www.cgsecurity.org/wiki/PhotoRec, Jan 2019.

[GSC00] Gers, F.; Schmidhuber, J.; Cummins, F. “Learning to forget: continual prediction with LSTM.”, *Neural Computation*, vol. 12–10, 2000, pp. 2451.

[GYSC11] Gopal, S.; Yang, Y.; Salomatin, K.; Carbonell, J. “Statistical learning for file-type identification”. In: 10th International Conference on Machine Learning and Applications and Workshops, 2011, pp. 68–73.

[Har07] Harris, R. M. “Using artificial neural networks for forensic file type identification”, Master’s Thesis, Purdue University, 2007, 56p. Retrieved from: https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/bibtex_archive/2007-19.ps, Dec 2019.

[Hie18] Hiester, L. “File Fragment Classification Using Neural Networks with Lossless Representations”, Undergraduate Honors Theses, East Tennessee State University, 2018, 36p. Retrieved from: https://dc.etsu.edu/cgi/viewcontent.cgi?article=1523&context=honors, Dec 2019.

[Hin87] Hinton, G. E. “Learning translation invariant recognition in a massively parallel networks”. In: International Conference on Parallel Architectures and Languages Europe, 1987, pp. 1–13.

[HS97] Hochreiter, S.; Schmidhuber, J. “Long short-term memory”, *Neural Computation*, vol. 9–8, 1997, pp. 1735–1780.

[KB14] Kingma, D. P.; Ba, J. “Adam: A method for stochastic optimization”, *arXiv preprint arXiv:1412.6980*, 2014.

[KGLPO10] Kattan, A.; Galván-López, E.; Poli, R.; O’Neill, M. “GP-fileprints: file types detection using genetic programming”. In: European Conference on Genetic Programming, 2010, pp. 134–145.

[KKM19] Kendall, K.; Kornblum, J.; Mikus, N. “Foremost”. Retrieved from: http://foremost.sourceforge.net/, Jan 2019.

[KS06a] Karresand, M.; Shahmehri, N. “File type identification of data fragments by their binary structure”. In: IEEE Information Assurance Workshop, 2006, pp. 140–147.

[KS06b] Karresand, M.; Shahmehri, N. “Oscar—file type identification of binary data in disk clusters and ram pages”. In: IFIP International Information Security Conference, 2006, pp. 413–424.

[LB95] LeCun, Y.; Bengio, Y. “Convolutional networks for images, speech, and time series”, *The handbook of brain theory and neural networks*, vol. 3361–10, 1995, pp. 15.

[LBD+89] LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W.; Jackel, L. D. “Backpropagation applied to handwritten zip code recognition”, *Neural Computation*, vol. 1–4, 1989, pp. 541–551.

[LOST10] Li, Q.; Ong, A.; Suganthan, P.; Thing, V. “A novel support vector machine approach to high entropy data fragment classification”. In: South African Information Security Multi-Conference, 2010, pp. 236–247.

[MBB+92] Matan, O.; Baird, H. S.; Bromley, J. M.; Burges, C. J.; Denker, J. S.; Jackel, L. D.; Le Cun, Y.; Pednault, E. P.; Satterfield, W. D.; Stenard, C. E. “Reading handwritten digits: A zip code recognition system", *Computer*, vol. 25–7, 1992, pp. 59–63.

[ME08] Moody, S. J.; Erbacher, R. F. "Sádi-statistical analysis for data type identification". In: Systematic Approaches to Digital Forensic Engineering, 2008, pp. 41–54.

[MH03] McDaniel, M.; Heydari, M. "Content based file type detection algorithms". In: 36th Annual Hawaii International Conference on System Sciences, 2003, 10p.

[MSR14] Maslim, A.; Sitompul, O. S.; Rahmat, R. F. "Distributed autonomous Neuro-Gen Learning Engine for content-based document file type identification". In: International Conference on Cyber and IT Service Management, 2014, pp. 63–68.

[Nad13] Nadeem Ashraf, M. "Forensic Multimedia File Carving", Master's Thesis, KTH Royal Institute of Technology, 2013, 56p. Retrieved from: http://www.diva-portal.org/smash/get/diva2:613183/FULLTEXT01.pdf, Dec 2019.

[PM09] Pal, A.; Memon, N. "The evolution of file carving", *IEEE signal processing magazine*, vol. 26–2, 2009, pp. 59–71.

[PMB13] Penrose, P.; Macfarlane, R.; Buchanan, W. J. "Approaches to the classification of high entropy file fragments", *Digital Investigation*, vol. 10–4, 2013, pp. 372–384.

[Pé19] Pérez, F. "Jupyter 4.4.0". Retrieved from: https://jupyter.org/, Nov 2019.

[QZG+14] Qiu, W.; Zhu, R.; Guo, J.; Tang, X.; Liu, B.; Huang, Z. "A new approach to multimedia files carving". In: IEEE 14th International Conference on Bioinformatics and Bioengineering, 2014, pp. 105–110.

[RHM86] Rumelhart, D. E.; Hinton, G. E.; McClelland, J. L. "A general framework for parallel distributed processing", *Parallel distributed processing: Explorations in the microstructure of cognition*, vol. 1, 1986, pp. 45–76.

[RIR05] Richard III, G.; Roussev, V. "Scalpel: A frugal, high performance file carver". In: Digital Forensic Research Workshop, 2005, 11p.

[Ros58] Rosenblatt, F. "The perceptron: a probabilistic model for information storage and organization in the brain.", *Psychological review*, vol. 65–6, 1958, pp. 386.

[Ros19] Rossum, G. "Python 3.6". Retrieved from: https://www.python.org/, Jan 2019.

[Rou08] Roux, B. “Reconstructing Textual File Fragments Using Unsupervised Machine Learning Techniques”, Master’s Thesis, University of New Orleans, 2008, 52p. Retrieved from: https://scholarworks.uno.edu/cgi/viewcontent.cgi?article=1861&context=td, Dec 2019.

[SZ12] Sportiello, L.; Zanero, S. “Context-based file block classification”. In: IFIP International Conference on Digital Forensics, 2012, pp. 67–82.

[Vee07] Veenman, C. J. “Statistical disk cluster classification for file carving”. In: 3rd International Symposium on Information Assurance and Security, 2007, pp. 393–398.

[VIP+19] Vulinović, K.; Ivković, L.; Petrović, J.; Skračić, K.; Pale, P. “Neural Networks for File Fragment Classification”. In: 42nd International Convention for Information and Communication Technology, Electronics and Microelectronics, 2019, pp. 1395–1399.

[WQW+18] Wang, F.; Quach, T.-T.; Wheeler, J.; Aimone, J. B.; James, C. D. “Sparse coding for n-gram feature extraction and training for file fragment classification”, *IEEE Transactions on Information Forensics and Security*, vol. 13–10, 2018, pp. 2553–2562.

[WSS18] Wang, Y.; Su, Z.; Song, D. “File Fragment Type Identification with Convolutional Neural Networks”. In: International Conference on Machine Learning Technologies, 2018, pp. 41–47.