File Fragment Classification using Content Based Analysis

Anirudh Bhat[1,*], *Aryan* Likhite[1,**], *Swaraj* Chavan[1,***], and *Leena* Ragha[1,****]

[1]Ramrao Adik Institute Of Technology, 400706 Nerul, Navi Mumbai India

Abstract. One of the major components in Digital Forensics is the extraction of files from a criminal's hard drives. To achieve this, several techniques are used. One of these techniques is using file carvers. File carvers are used when the system metadata or the file table is damaged but the contents of the hard drive are still intact. File carvers work on the raw fragments in the hard disk and reconstruct files by classifying the fragments and then reassembling them to form the complete file. Hence the classification of file fragments has been an important problem in the field of digital forensics. The work on this problem has mainly relied on finding the specific byte sequences in the file header and footer. However, classification based on header and footer is not reliable as they may be modified or missing. In this project, the goal is to present a machine learning-based approach for content-based analysis to recognize the file types of file fragments. It does so by training a Feed-Forward Neural Network with a 2-byte sequence histogram feature vector which is calculated for each file. These files are obtained from a publicly available file corpus named Govdocs1. The results show that content-based analysis is more reliable than relying on the header and footer data of files.

1 Introduction

There are several ways to restore files. If the file system index is available, standard methods use it to restore the blocks of the deleted file. In computer forensics, file carving is an act of extracting data from a disc drive or other storage device without using the file system that created the file with originality. It is a tool that recovers files in an unallocated space without any file information and is used to retrieve information and performing a digital forensic investigation. It's commonly referred to as "carving," which is a term for the process of organizing raw data using format-specific features present in the format. As a forensics technique that recuperates records dependent on file structure and content and with no matching file system meta-data, file carving is regularly used to recover files from the unallocated space in a drive. Unallocated space alludes to the zone of the drive which no longer holds any document data as demonstrated by the record framework structures like the document table. On account of harmed or missing document framework structures, this may include the entire drive. In straightforward words, numerous file systems don't zero-out the information when they erase it. All things being equal, they essentially eliminate the information on where it is. File carving is the way toward reproducing documents by checking the crude bytes of the circle and reassembling them. This is typically done by inspecting the header (the initial not many bytes) and footer (the last couple of bytes) of a record.

*e-mail: anirudh.b.123@gmail.com
**e-mail: aryanlikhite@gmail.com
***e-mail: swarajchavan540@gmail.com
****e-mail: leena.ragha@rait.ac.in

2 Motivation

File carvers that use file metadata are inherently faulty. Data are fragmented on modern hard drives which imply that to identify the block, the header and footer data can not always be present. File carvers that use the principle of graphs for file fragment recognition are limited to dealing with only bi-fragments. If the header data is tampered with then the 'file' command available in Linux-based file systems fails to recognize the file.

As there are statistical variations between different file formats, the different file format fragment has to be distinguished by using the knowledge that the byte sequence and byte frequency are not the same in each file format. Hence using these features it is feasible to classify different file types using algorithms that would provide high accuracy.

This study can then facilitate a means to overcome the primary limitation of modern file carvers which is the inability to identify the fragments in the hard drive which are not the header or footer fragment and hence can help improve them.

3 Related Work

3.1 Survey of Existing Systems

Many file carvers use "magic numbers" to recover files. A popular tool "Scalpel", available for both Linux and Windows OS, uses such a technique.[5]. The Unix file command is perhaps the most widely used file identification command in history. The file command compares particular regions of a file (usually the file's header and footer) to a database and returns the first match. When dealing with full data, this command is fairly effective. When a

file fragment is presented with a file fragment, it means that it is a "data" file. Some complex file carvers use the graph theory to achieve file carving capabilities.[4] In the process of the Format Validation phase, several techniques are studied as well.

Non-header and non-footer based approach was first used in [10]. They used historgrams of ASIC codes which was given to a clustering algorithm. The approach yield an accuracy of 27.7% for byte frequency analysis and 46% for byte frequency cross-relation

In [9] various classifiers were used such as SVM, k-NN, k-means, decision trees and neural networks. Classification speed was the main goal. Also, instead of using hand-engineered features, they used sampled file blocks. They also used the first block of every file to boost their accuracy as the first block contains the magic numbers

Fitzgerald[3] used NLP Techniques for the identification of file fragments. They selected 24 file fragments for their classification. The dataset they created was obtained from the publicly available dataset. They extracted several features which include 1-gram, Shannon Entropy, and Hamming weights. For classification, they went with the route of using a Support Vector Machine combined with a bag-of-words model which is used in NLP. Their approach yielded an accuracy of around 47%

F. Wang[6] used Sparse Coding for N-Gram Feature Extraction. For their dataset, they used the publicly available dataset of Govdocs1. They used the approach of creating a sparse dictionary for N-Grams of higher-order such as N = 4,8,16,32,64 etc. They then used these dictionaries to create N-Gram frequencies of the given file fragment. Support Vector Machine (SVM) was used for the classification and generation of the model. They decided to include 18 file formats in their studies and achieved an identification accuracy of around 61.31%.

Q. Chen et al.[2] took the approach of classification of file fragments by converting fragments to gray scale images and applying deep learning techniques. They took fragments of 512 bytes. These fragments of 512 bytes were then converted into 64x64 greyscale images. These greyscale images were then used to train a convolution neural network (CNN) and train a model. The generated model was trained and tested on 16 file formats from the public dataset of Govdocs1. The accuracy they achieved was around 71%.

In another research paper by F. Wang [7], they used the approach of classification using Convolution Neural Networks. They formalized the task as a multi-class classification problem. The goal was to locate specific byte patterns that belong to an identifiable file type, similar to magic numbers. To achieve this, they used convolution neural networks. The dataset they used was from Govdocs1. They performed tests on 20 different file formats. They performed experiments with several different fragment sizes such as 64 bytes, 126 bytes, 256 bytes, etc. Their results showed that as we increase the size of the fragments used for classification, the accuracy is increased. Also, if the fragment belongs to the start or the end of the file, the classification accuracy is higher as compared to a fragment taken from the middle of the file

Bhatt, M[1] used a different classification technique than most others. Their approach was a mixture of clustering and classification. They separated the file types into complex and simple file types. The simple file types were further differentiated into text files and image files. They extracted a total of 10 features. They used information gain which is a feature ranking metric. They then used Support Vector Machine for the classification of the file fragments. The accuracy they measured was around 67.78%.

3.2 Limitations of Existing Systems

File carvers that use file metadata are inherently flawed as files in modern hard drives are fragmented. This means that the header and footer data may not always be present to classify the block.

File carvers that use the graph theory for the identification of file fragments are limited to working with only bi-fragments[4]. This means that they are incapable of detecting more than 2 fragments. Also, the existing 'file' command available in Linux-based file systems, fails to identify the file if the header data is tampered with. This is because it uses header and footer content for the identification of the file.

Fitzgerald[3] which uses the NLP techniques manages to get an accuracy of around 47.5%, which is less than desirable for use in real-world data. In a real scenario, if a file carver was able to correctly identify less than half of the files, there would be a huge time loss for the forensic expert.

F. Wang[6] used Sparse Coding. In this approach, they created a sparse dictionary of N-Grams. While this method is suitable when the value of N is kept low, the time for calculating the feature vector increases exponentially as we increase the value of N. Their results show that higher values of N yield far better accuracy than lower values of N. So, to achieve acceptable accuracy's, we need to increase the value of N, but increasing the value of N makes the processes of feature calculation more time consuming

Q. Chen et al.[2] converted the file fragments into greyscale images and achieved an accuracy of around 70.4%. While the accuracy is high compared to most other studies, converting fragments into greyscale images and passing them through a CNN, is not feasible in real-time applications. Real-time applications need to be fast and should be able to work on a relatively large database.

Bhatt. M[1] used the Hierarchy-Based File Fragment Classification method. They classified simple file correctly but for complex file, often it failed to classify the specific type of complex file accurately. Moreover, the time required for the feature extraction for one file was very high, and this time can add up when working with a large sum of data in real-world situations.

Besides all this, none of the studies have provided a codebase to test their results or use them as a base for further research in this domain. The experiments are done on different datasets and parameters, so the results and not comparable.

4 Proposed Approach

4.1 Proposed Work

The first step was to gather a bunch of files for training and testing purposes from the Govdocs1 dataset. *Govdocs1* is a publicly available corpus of files. Govdocs1 contains approximately 900,000 files with 213 different file types. But the distribution of these file types is not uniform. Some file types appear more than 150,000 times while some file types have less than 10 files. Hence there is a huge disbalance between the different file types. For an accurate and fair test, it is ideal that every file type in the training and testing data-set should have an equal count of files. This is required so that the models do not become biased to one file type due to an excessive amount of training on particular file types. To overcome this problem, it was decided to select 13 extensions based on their frequency in the Govdocs1 data-set and their relative popularity.

The Govdocs1 corpus provides pre-packaged "threads". These threads contain 1000 files randomly sampled from the main corpora specifically meant to be used for research purposes. The process began by downloading multiple such threads and collecting all the files in a single location. Then, out of the 13 selected file formats, 300 random files were sampled for each of the 13 file types, 200 for training, and 100 for testing. This formed the main data-set for training and testing the models. The 13 file formats used were as follows

PDF	HTML	BMP	JPG
RTF	PNG	DOC	TXT
XLS	GIF	XML	PS
CSV			

For any machine learning algorithm, feature extraction is the most important step. In [1] a variety of features were tested and ranked using a feature ranking metric called information gain. The results showed that Unigram Count Distribution or Byte Histogram, was the most dominating feature with an information gain score of 3.807 while other features had scores in the low 1's and 0's. A byte histogram of a file is the distribution of occurrence of each byte in the file. Since there are 8 bits in a byte, there is 256 such byte that every file can contain.

Figure 1, visualizes the Byte histogram of several file types by converting the histogram into 16x16 grey-scale images. From the image, one can observe that the distribution of bytes in different file formats varies vastly. This visualization shows us that using a byte histogram as a feature vector is possible and can yield good results.

Although using single bytes as a feature vector has proven to be useful[6], a lot of information is lost when considering only a single byte. There are certain sequences of bytes that occur more in a specific file type. For example, in JPG files, before the byte sequence of "FF" the byte sequence of "00" is stuffed as it is used as an escape character. This makes the frequency of byte sequence "00" much higher as compared to other file types. Another example that can be given is that of HTML files. HTML

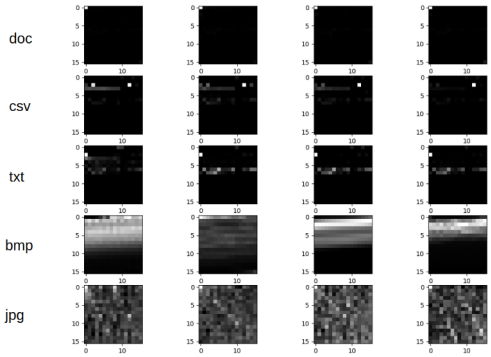

Figure 1. Byte Histogram of several file types converted to Greyscale Images

files contain a large number of "</" character sequences, so subsequently, the byte sequence of "</" is higher in HTML files than most other files. Hence instead of considering single bytes, it is better to consider a sequence of 2 bytes to gain more information and possibly increase the accuracy. To test this, we ran multiple tests with SVM as well as MLP with uni-gram and bi-gram as data input. The results are discussed in Section VI. Based on our results, instead of using a single byte as a feature vector, 2-byte sequence histograms are used as the feature vector. As there is a 256 possible byte, considering a sequence of 2 byte makes the possible feature vector of size 256*256 that is 65536.

4.2 Proposed Methodology

As discussed in the above section, a 65536 sized array is used as the feature vector. A data set of 13 file types with 200 files for training and 100 files for testing for each of the 13 file types is created. This brings us to a total of 1300 files. For each of these 1300 files, a 2-byte sequence histogram is generated. The actual process of generation of this histogram is explained in detail in Section 5.1. After generation of the histogram, they are saved in a CSV file separated by training and testing histograms respectively. This data is then tested on a Feed-Forward Neural network as well as a Convolution Neural network to test which neural network performs better.

Feed-Forward Neural Networks is a kind of Artificial Neural Networks. They are made of the basic building block of computational units called neurons. A feed-forward network is generated by using multiple neurons in multiple layers. They are arranged in a way that the output of each neuron of one layer is fed as input to every neuron in the next layer. Each of these neurons also has an activation function, which ensures a degree of non-linearity in the model. Each layer can have a different number of neurons and there can be different number of layers depending on the capacity of the learning needed. However, the number of neurons in the first input layer and the last input layer are fixed. The number of neurons in the first input layer must match the number of features and the number

of output neurons must match the number of possible outcomes. Furthermore, the choice of activation function can make a big difference. The network was tested with several hidden layers and in the end, it was decided to settle with 4 hidden layers based on performance speed and accuracy. As for the activation function, after testing with multiple ones, it was found that "RELU" appeared to be the best, and the least time-consuming choice.

For testing our data with a Convolution Neural Network, the data was first reshaped into a 256x256 matrix. This data was then passed through 2 layers of a convolutional network followed by a feed-forward neural network.

After a model is generated using the training data, the model is then saved and used to power a command-line utility. A command-line utility will provide ease of access to the end-users and will act as a replacement for the standard "file" command used by Digital Forensic Experts.

5 System Design

The design of the system is mainly divided into 4 major components, namely Data Collection, Data Processing, Training and Testing, and the Command Line Tool. Each of these phases is explained in detail below

5.1 Data Collection

The first step in data collection is downloading the threads from the Govdocs1 dataset. These threads are a collection of 1000 files. 9 such threads were downloaded and then finally collected in a single folder. The next thing to decide is how many file formats to consider. Based on the number of files present for each file format, 13 of the most commonly occurring file formats are picked. These 13 file formats should have an equal number of files for each format. Hence randomly, 300 files for these 13 formats are sampled from the set. From these 300 files, 200 files are sampled for the training set, and the remaining 100 files are reserved for the testing set. With this, an equal number of files for each file format is present so that the training data is not dominated by only 1 single file.

5.2 Data Processing

In the data processing phase, these files are needed to be converted into their respective 2 pair byte histogram. To achieve this, first, start with a file and load it into memory. Now a random starting point is selected from the file such that the starting point is at least 4096 bytes behind the end of the file. After picking the starting point, then read the next 4096 bytes. This represents the fragment. 4096 bytes are selected because that is the most common fragment size in modern hard drives. For every sampled fragment, the raw hex data is analyzed and the 2 pair Byte Histogram is generated. This forms the feature vector and is saved in a file separated by training and testing data. This process is repeated for every such file.

5.3 Training and Testing

The train.csv file is first loaded into the memory. Then a Feed-Forward Neural network represented as FNN in the diagram is created based on the architecture discussed above. All the data from the train.csv file is used to train the created Feed Forward Neural Network. After successful training, load the test.csv file into memory which has the testing data. This data is then fed into the trained FNN, which classifies the file. This classification is then compared with the real file format and hence the accuracy is measured. The generated model is saved for powering the command-line tool.

5.4 Command Line Tool

The command-line tool is used as the final product which users can use to classify the fragments. When the command-line tool is used, first, the saved model is loaded into memory. The user specifies a file for which they wish to know the extension. This file is then read and then a 2 pair byte histogram is generated for the file. This generated Byte Histogram is fed into the FNN which classifies the file into one of the 13 selected formats. This classification result is then shown to the user

6 Results

Figure 3 shows the results of the tests of unigram vs bigram. In total, 4 test runs were conducted with different datasets to make the results more authentic. The two classifiers used were SVM and MLP and the features were chosen were Unigram and Bi-Gram. From the bar graph, it can be seen that MLP with Bi-Gram features was the most successful. This proves that the theory for going with the Bi-Gram feature as it retains more information was correct because it provided us with more accuracy as compared to going with Unigram features.

Architecture	Accuracy
65536x1024x512x13	90.32
65536x1024x256x13	89.92
65536x512x256x13	89.43
65536x256x128x13	89.00
65536x512x13	88.7

Table 1. List of architecture and their accuracy

Table 1 represents the result of our tests conducted with different architectures with MLP. It can be observed that the accuracy is more the less similar even after using different architectures. This shows that the architecture of the Neural Network doesn't affect our results significantly. The best performing architecture was the one with 2 hidden layers of 1024 and 512 neurons respectively

The tests were also conducted on a Convolution Neural Network. The result of Convolution Neural Networks was found be around 75% - 80%. This is because histograms do not posses any spatial data. CNN's are better used for

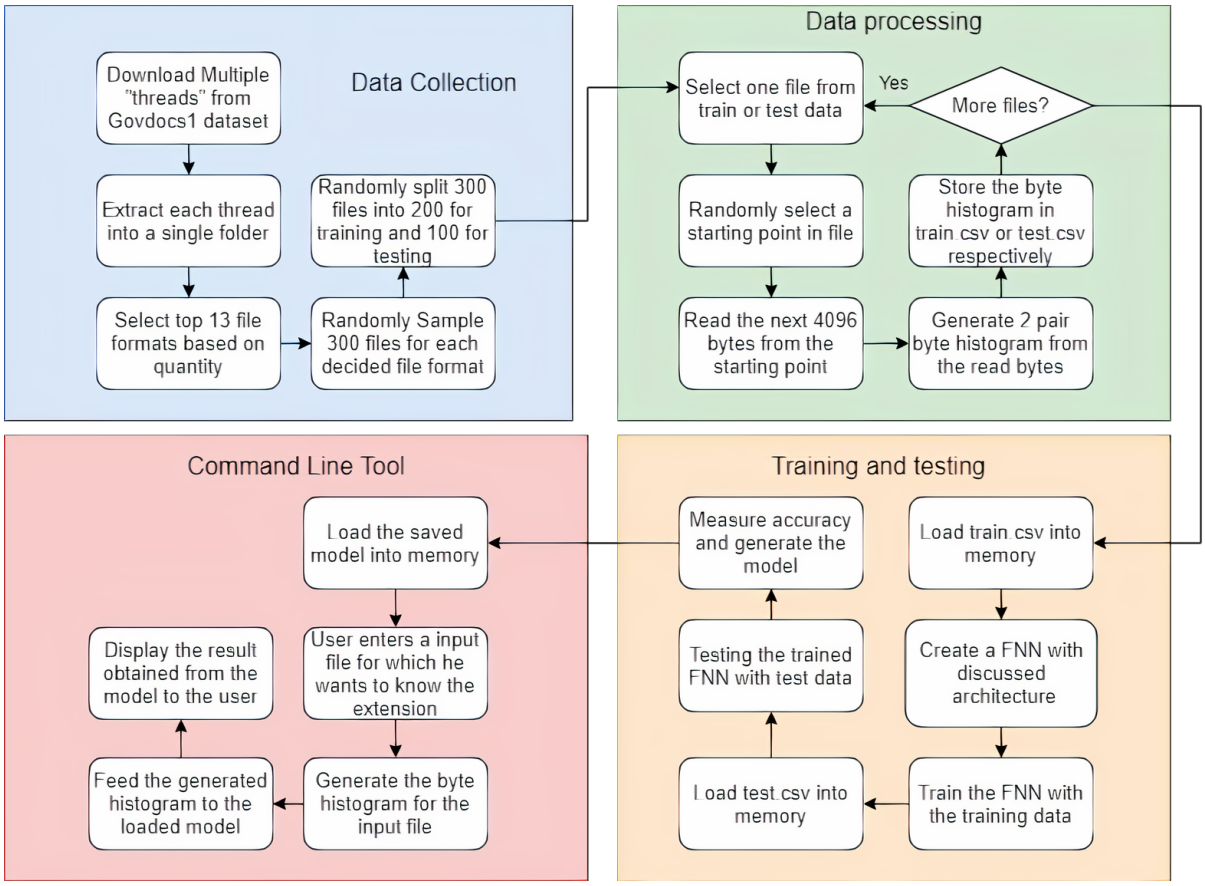

Figure 2. System Design which shows the workflow of the entire system

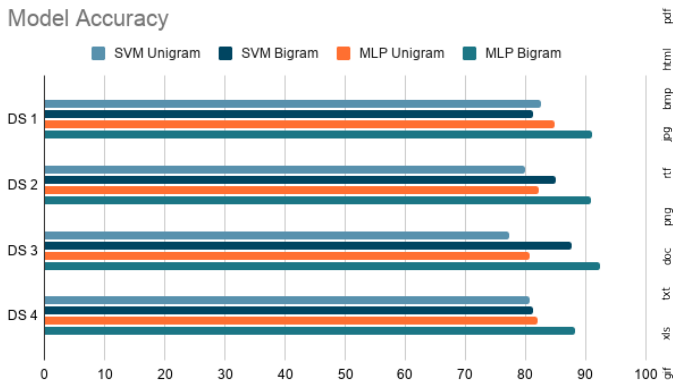

Figure 3. Accuracy of models used on different data sets

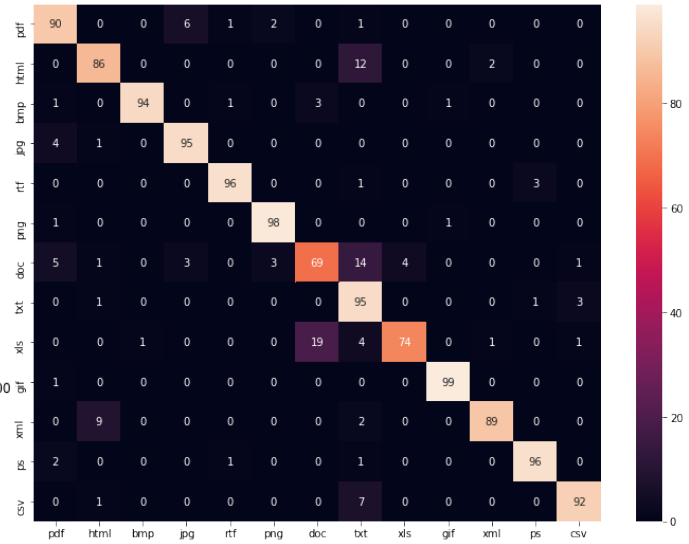

Figure 4. Accuracy of models used on different data sets

image datasets. Hence in comparision with CNN, MLP performs much better.

Figure 4 shows the confusion matrix of our results for the MLP Classifier. It is evident that for the majority of the file formats, the MLP Classifier works well and classifies the files correctly. We can see that for HTML files, some of the HTML files were classified as TXT files. The predictions are made on a 4096-byte fragment. Due to the small nature of the fragment, it is quite possible that the particular fragment selected from the HTML files contained only pure text part and hence it classified them as TXT files. We also observe that many of the XML files were classified as HTML files. These two file formats have a lot in common, such as both of them use "tags" and contain a lot of text. The similarity between these file formats is the reason that XML files are getting classified as HTML files.

Upon examination of PDF files, it was observed that a large portion of PDF files contained images. The images in PDF files are stored as a separate object that contains the raw binary data of the image. Similar to the HTML and TXT confusion, due to the small size of the fragment, it is quite possible that the fragment extracted from the PDF files contained only the pure image raw bytes and hence some PDFs are classified as jpg and png

The worst performing file types were doc and xls. Both these file types can be considered "complex" file types. A file type can be considered complex when it combines elements from different file formats like text and images. Looking at the results of XLS files, we can see that a lot of XLS files were classified as doc files. The results may seem confusing at first as at a surface level, we do not find any similarities between these two file formats. After opening these two file formats in a hex editor and taking a look at their hex data, it was observed that the byte sequence between these two file formats is very similar. In both the file formats, doc as well as xls, there is a big portion where the bytes "FF" or "00" are repeated for long stretches. Naturally, the bigram frequency of the bytes "FF" and "00" are high in both these file formats as compared to any other file format. Due to such similar nature of raw binary data between these two file formats, many of xls files are classified as doc files. The worst performing file format was DOC. The majority of doc files were classified as txt files. This behavior is expected as the majority of doc files are filled with long stretches of texts, so the fragment extracted from these doc files during the training and testing period might only contain text. Some doc files are classified as xls files. The justification for this is the same as to why xls files are classified as doc files. Some doc files are also classified as png and jpg files. Doc files often contain images as well along with the text. Hence the reason for some doc files getting classified as jpg or png is that the particular fragment tested might only contain raw image bytes.

Overall, the confusion matrix shows us that the current classification method works well with simple file formats which only contain one specific type of data and works a little worse for complex file formats which usually contain multiple types of data.

7 Conclusion and Future Work

After several trials, it was found that using single bytes can cause a lot of information to be lost, making the accuracy a bit low. Using 2 bytes helps to retain much more information hence improving the accuracy. It can also be concluded that Feed Forward Neural network works much better than SVM for this particular scenario

The main objective of this paper was to utilize all the research work that has been done in the field of file fragment classification and create a tool that is available for everyone to use on their local machine. In these experiments, only the top 13 of the most frequently used file formats were considered, due to hardware and resource constraints. In an ideal world, the training should be done on a large dataset with many varieties of file formats. This is where the open-source part of the project comes to use. Anyone with more resources or a server with better hardware can use this codebase to train more datasets, and possibly find more features to extract and improve on the original codebase.

References

[1] Bhatt, Manish, Avdesh Mishra, MdWasi Ul Kavbir, S. E. Blake-Gatto, Rishav Rajendra, Md Tamjidul Hoqueand Irfan Ahmed. "Hierarchy-Based File Fragment Classification." *Machine Learning and Knowledge Extraction* 2, no. 3 (2020): 216-232.

[2] Chen, Qn, Qing Liaoe L. Jiang, Jun Fang, Siuming Yiu, Guikai Xi, Rong Li et al. "File fragment classification using grayscale image conversion and deep learning n digital forensics." In *2018 IEEE Security and Privacy Workshops (SPW)*, pp. 140-147. IEEE, 2018.

[3] Fitzgerald, Simran, George Mathews, Colin Morris, and Oles Zhulvyn. "Using NLP techniques for file fragment classification." *Digital Investigation 9* (2012): S44-S49.

[4] Pal, Anavndabrata,v and Nasir Memon. "The evolution of file carving." *IEEE sigvnal processing magazine* 26, no. 2 (2009): 59-71.

[5] Poisel, Rainer, Simon Tjoa, and Paul Tavolato. "Advancevd file carving approaches for multivvmedia files." J. *Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 2, no. 4 (2011): 42-58.

[6] Wang, Felix, Tu-Thach Quach, Jvason Wheeler, James B. Aimone, andv Conrad D. vJames. "Sparse coding for n-gram featurve vextravction and training for file fragmenvt cvlassifvicativon." v*IEEE Transactions ovn Information Forensics and Security 13*, no. 10 (2018): 2553-2562.

[7] Wang, Yanchao, Zhongqian Su, and Dayi Song. "File fragment type identification wvith convolutional neural networks." In Proceedings of the 2018 International Conference on Machine Learning Technologies, pp. 41-47. 2018.

[8] Lee, Seokjun, and Taeshik Shon. "Improved deleted file recovery technique for Ext2/3 filesystem." The Journal of Supercomputing 70, no. 1 (2014): 20-30.

[9] Ahmed, Irfan, Kyung-Suk Lhee, Hyun-Jung Shin, and Man-Pyo Hong. "Fast content-based file type identification." In IFIP International Conference on Digital Forensics, pp. 65-75. Springer, Berlin, Heidelberg, 2011.

[10] McDaniel, Mason, and Mohammad Hossain Heydari. "Content based file type detection algorithms." In 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the, pp. 10-pp. IEEE, 2003.