5-2018

File Fragment Classification Using Neural Networks with Lossless Representations

Luke Hiester

File Fragment Classification Using Neural Networks with Lossless Representations

Luke Hiester

May 2018

Abstract

This study explores the use of neural networks as universal models for classifying file fragments. This approach differs from previous work in its lossless feature representation, with fragments' bits as direct input, and its use of feedforward, recurrent, and convolutional networks as classifiers, whereas previous work has only tested feedforward networks. Due to the study's exploratory nature, the models were not directly evaluated in a practical setting; rather, easily reproducible experiments were performed to attempt to answer the intial question of whether this approach is worthwhile to pursue further, especially due to its high computational cost. The experiments tested classification of fragments of homogeneous file types as an idealized case, rather than using a realistic set of types, because the types of interest are highly application-dependent. The recurrent networks achieved 98 percent accuracy in distinguishing 4 file types, suggesting that this approach may be capable of yielding models with sufficient performance for practical applications. The potential applications depend mainly on the model performance gains achievable by future work but include binary mapping, deep packet inspection, and file carving.

Acknowledgements

I would like to thank Dr. Brian Bennett and Dr. Jeff Knisley for their feedback and attention to detail; the Honors College and the University Honors Scholars program for their support of undergraduate research; and Dr. Jay Jarman for providing continued guidance since before the inception of this particular study through its completion, and for inspiring my interest in machine learning.

Contents

1 Introduction

This study investigates a novel approach for applying neural networks to file fragment classification. File fragments are small, isolated segments of files, and file fragment classification is the task of identifying the type or format of a fragment, which may be a primitive encoding such as UTF-8 text, a file type such as JPEG, or some other type, based on the application. This problem initially gained interest due to its potential applications in digital forensics.

The approach employed in this study has two core components. First, the fragments' individual bits are used directly as model input—no previous work was found using this or any other lossless feature representation. Second, multiple neural network architectures are explored—previous work has only tested feedforward networks, but this study also tests recurrent and convolutional networks. One of the fundamental differences of this approach from existing work is that practically all of the previous models were constrained to consider only specific patterns, but these models are constructed to be capable of recognizing arbitrary patterns.

One major school of thought in file fragment classification research is that classifiers *should* be specialized and should consider the most specific patterns possible in order to maximize performance in terms of both accuracy and speed. This is based largely on the needs of specific applications where the performance of generalized models constructed in the past has been inadequte. However, none of the past models were truly universally generalizable, which may have negatively impacted their performance and also represents an unmet need for other applications where generalized models are necessary. This study seeks to address both of these issues.

This paper is organized as follows. Section 2 establishes some context, surveying prior research in file fragment classification and highlighting influencing works for this study. Section 3 discusses the resources and tools used, and section 4 outlines the experimental procedure. Section 5 presents the results of the experiments. Section 6 explores the results and discusses potential applications of this work. Section 7 concludes.

2 Background

The research area of file fragment classification originated as an extension of content-based file type identification. In this context, "content-based" refers to using the patterns in the body of a file in order to assign it a type, rather than internal metadata such as headers or external metadata such as file extensions. These normal signatures may be unreliable for several reasons—file extensions may easily be changed [25], and file signatures located in headers or other fixed locations may be missing for several reasons [22]. Neither of these signatures can be used in file fragment classification, where both the file name/extension and the location of the fragment within the file are unknown.

2.1 Early Work

In his 2001 master's thesis, McDaniel presented the first formal examination of content-based techniques for file type identification [24]. Shortly afterwards, McDaniel and Heydari's seminal 2003 paper [25] presented the same techniques in a more accessible medium, quickly inspiring work by other researchers.

McDaniel and Heydari presented three approaches for classifying files based on their content. All three are distance-based models, with a file classified by computing various measures for it and assigning it the type having the closest average values of the same measures precalculated on a set of reference files, with the comparison weighted inversely by the observed variance of each measure. In byte frequency analysis (BFA), the features used to represent a file are the frequencies of all 256 possible byte values within it. In byte frequency cross-correlation (BFC), the features are the pairwise differences between the byte frequencies. In file header/trailer (FHT), the features are the frequencies of each value for each byte within a given distance from the beginning or end of the file.

Li, Wang, Stolfo, and Herzog presented a similar file type identification model in 2005, which they termed the "fileprint" model [22]. Although sometimes described as an extension of McDaniel and Heydari's BFA technique, it was actually a repurposed version of the model Wang and Stolfo had previously applied to network packet analysis [35]. The fileprint model is similar to the BFA technique in that it uses comparisons of byte frequencies weighted by variance—although in a slightly different form—but it differs by using multiple byte frequency distributions per file type, with the byte frequencies of a set of reference files for each type clustered using the k-means algorithm and then averaged together to form one centroid of byte frequencies per cluster.

Li et al.'s paper was the first to address the case of classifying arbitrary fragments of files, making a comment about classification techniques needing to be generalizable to fragments of files contained in individual network packets and thus not being able to rely on header information or signatures at specific locations. However, they did not test this; in their experiments they only classified full files or the first n bytes of files for various values of n.

Karresand and Shahmehri's studies in 2006 were the first to test classification of file fragments [19, 20]. Their first model [19] was similar to Li et al.'s fileprint model using one byte frequency distribution per file type, and they used it to classify 4,096-byte fragments with a focus on identifying fragments of JPEG files. Their second model [20] introduced the distribution of the difference between consecutive bytes, or the "rate of change."

These early works inspired many other researchers to explore file fragment classification, and the approaches presented within them were emulated with slight adjustments in many ways. In particular, models using byte frequency distribution and/or distance-based classification techniques have remained highly popular since.

2.2 Survey of Techniques

A large number of features have been tried in file fragment classification and file type identification models.[1] The most frequently used feature has been the byte frequency distribution (BFD) or unigram frequencies, i.e. the frequency of occurrence of each possible byte value [1–5, 7, 8, 11, 12, 15, 16, 19, 20, 22–26, 31–35]. Some studies have also utilized bigram frequencies [7, 8, 12, 15, 31–33], and one study used trigram frequencies [15]. Another popular family of statistics have been byte value mean [7–10, 12, 26, 31, 32], mean absolute deviation [7], standard deviation [7, 9, 26], skewness [7], and kurtosis [7, 26]. Several features related to complexity and information content have been popular, including Shannon entropy [7–10, 12, 17, 31, 32, 34], compressibility [7, 8, 12, 17, 27, 34], and Lempel-Ziv complexity [31, 32]. Other features appearing in multiple studies include Hamming weight [7, 8, 10, 12], difference between consecutive byte values (or "rate of change") [7, 8, 12, 20, 31, 32], maximum length of repeated byte values [7, 8, 12], frequency of occurrence of bytes in ASCII range [7, 9], and various measures computed in a sliding window through the file or fragment [17, 26]. Less common features include pairwise differences between frequencies of byte values [25], distribution of the exclusive-OR of consecutive byte values [11], chi-square statistics [10], the NIST randomness metrics suite [27], and GIST image processing features [36]. Multiple unique features appear in [9], including least common substring and subsequence measures, the sum of the four highest byte frequencies, the correlation of values of consecutive bytes, and the correlation of frequencies of consecutive bytes.

Several studies have also experimented with feature extraction and dimensionality reduction techniques. Principal component analysis [1,4,5] and autoencoder networks [4, 5] have been used to extract features with unigram counts as input. In [21], a genetic algorithm was used to evolve a feature extractor from the raw contents of files using class separability in the final feature space as the fitness metric.

For a while, the most popular classification models in this area were distance-based or nearest-neighbors models [2, 3, 6, 10, 15, 17–22, 24, 25, 33, 35, 36]. However since their first usage in this area in 2010 [23], SVM models have become more popular [3, 5, 7, 12, 15, 31, 32, 36]. Feedforward neural networks have been used with moderate frequency throughout the history of the field [1, 3–5, 11, 16, 27]. Linear discriminants [2, 3, 9, 34] and decisions trees [3, 36] have also been used in multiple studies. Some studies used specially designed models for recognizing specific types [19, 20, 29]. One study used a naïve Bayes classifier [36].

The previous works using neural networks are of particular interest here, since these models are the focus of this study. In 2008, Amirani et al. used a feedforward network to classify whole files using unigram frequencies processed by principal component analysis and an autoencoder network [4]. In 2011, Ahmed et al. investigated classifying whole files by unigram frequencies with several different models, including feedforward networks [3]. In 2013, Amirani

[1]As no further mention is made of most of these techniques in the rest of this paper, a thorough explanation of them is not provided here. See the cited works for more information.

et al. extended their previous work to file fragments, classifying fragments by unigram frequencies and the same feature extraction pipeline as their 2008 study, this time trying both feedforward networks and SVM models as classifiers [5]. Also in 2013, Penrose et al. used feedforward networks to differentiate fragments of compressed and encrypted files using the NIST randomness feature suite [27]. In 2014, Aaron et al. used feedforward networks tuned by a genetic algorithm to classify file fragments by unigram frequencies processed by principal component analysis [1].

Two previous studies used file contents as direct input to neural network models. Of the two, the 2005 study by Dunham et al. [11] is related less closely to the approach in this study. They used feedforward networks to classify delta files of encrypted data[2] to identify whether they were encryped with the same key, and to attempt to identify the file types of the original, unencrypted data. The main features used were based on unigram frequencies within the delta files, but the numeric values of the first 32 bytes of the delta files[3] were used as features also. This feature representation differs from the one used in this study in that only a small portion of the contents were provided directly as input and that, due to the nature of the objects being classified, these 32 bytes represented a rather complicated transformation from raw bytes of actual files. However, their study appears to be the first instance both of providing byte content directly to a classification model and of using neural network models in the areas of file type identification and file fragment classification.

The most closely related previous work is Harris's study in 2007 [16]. Harris used feedforward networks to classify file fragments, alternatively using unigram frequency or actual byte values as input. Interestingly, this approach did not yield good results, with all of the networks tested having less than 50 percent accuracy on classifying five types of files, and networks using byte values as direct input performing worse than those using unigram frequency. Perhaps these results discouraged other researchers from exploring this approach further, but there were several issues that likely impacted the results negatively. First, the file types chosen were problematic for several reasons, as discussed in section 2.3, and may have actually made the problem theoretically intractable (an issue shared by many existing works in file fragment classification). Second, the networks may have sufferred from slow training due to the use of mean squared error as the training objective in a network with hyperbolic tangent units [14]. Third, and related to the previous point, the networks were not allowed to train long enough to converge; Harris noted that they were all stopped after 20,000 epochs and none of them had yet reached convergence. Fourth, the *numeric* values of the bytes were used as input—although in theory the raw contents could be losslessly extracted from this representation, practically it is a complicated aggregation from which the true raw contents (the bits) are difficult to extract for any model except one specifically designed to do so, even further complicated by the fact that the byte values were rescaled in this case.

[2] In this context, a "delta file" is the sequence of corresponding bytes of two encrypted files, combined by a bitwise exclusive-OR operation.

[3] I.e. the exclusive-OR of the corresponding first 32 bytes in the two encrypted files.

No existing work was found that shared the primary techniques of interest in this study. The first core technique in this study is the use of multiple neural network architectures; all existing work with neural networks in this area used only feedforward networks, perhaps due to recurrent and convolutional networks being relatively recent. The second core technique is the use of bits as direct input to the classification models; only two studies were found using raw file data as input, and both used the numeric values of bytes, perhaps due to the notational convention of representing byte values as numbers—however, this convention is purely for convenience and is only a true representation of the bytes' values when they are explicitly being used to store one-byte numbers; in most cases, the numeric value is not an appropriate way to interpret the contents.

2.3 Roussev and Garfinkel's Case for Specialized Approaches

In 2009, Roussev and Garfinkel critically examined the existing work in the area of file fragment classification [28]. They identified multiple problematic assumptions that were common in prior studies (and have continued to be common through the time of writing of this paper), issued several challenges for the performance and quality of file fragment classifiers, made suggestions for how to reformulate the problem in a conceptually sound manner, and gave practical recommendations for how to construct useful tools.

One common problem with previous work was a poor definition of "file type." Roussev and Garfinkel noted that most authors implicitly equated file type with file extension, working with the assumption that files with distinct extensions have well-defined "types," without investigating further. In actuality, file extensions are arbitrary naming conventions that acquire meaning solely through common usage. File extensions do not impose any constraints on the contents of the files they name. Some file types are standardized and have well-defined formats including standard extensions that should be used to name them, and through implicit social contract, it is often reasonable to assume that files with those extensions are actually files of the corresponding types. However, this is not *safe* to assume because a file may have an intentionally deceptive name. Additionally, standardized formats are themselves rarely unique, typically using primitive formats and data encodings shared with many other file types. Thus, in the general case a set of file extensions may not correspond to a meaningful set of "types," and even when they do, files of these types may not have distinctive formats for their contents.

Another common issue was assuming, without specific knowledge of the formats under investigation, that distinctive signatures would naturally arise from statistics collected from files or fragments. As noted by Roussev and Garfinkel, files are synthetic objects and contain only the patterns they are designed to contain. In particular, statistics based on byte frequency can only reliably differentiate several broad classes of files, certainly at a coarser granularity than that at which previous studies have attempted to apply them.

The main effect of these flawed assumptions was the use of sets of file types which are inherently impossible to classify correctly by any means other than pure chance. Compressed file types are one such set. Many compression formats yield byte distributions that are very close to uniformly random[4], which makes statistics computed from byte frequencies unable to differentiate many sets of compressed types. An even worse recurring issue noted by Roussev and Garfinkel was the inclusion of multiple file types that use the Deflate compression format to encode their contents, such as PNG images and ZIP archives—when fragments are extracted from these file types, they are impossile to distinguish because their contents are formatted identically.

An even more probematic set of file types when classifying fragments are compound formats; not only do many file types have highly heterogeneous contents, but also a significant number allow arbitrary other files or blocks of data to be embedded within them. A fragment extracted from a file of compound type may be another valid type and may be impossible to identify as originating from a file of the compound type by any means. Roussev and Garfinkel specifically examined PDF files as an example of such a compound type. In a sample of over 100,000 PDF files, they found that an average of only 9.7 percent of the contents were characteristic of the PDF format, with the other 90.3 percent being embedded blocks of data encoded with non-unique formats. In particular, they found that 49.1 percent of the contents were Deflate-encoded, meaning that a significant number of fragments extracted from PDF files are completely indistinguishable from fragments of other Deflate-encoded types.

Roussev and Garfinkel also made several challenges based on the inadequacy of previous classification models for the purposes of file carving, a frequently cited application and the main one which interested them. File carving attempts to recover the files on a storage device that may be unreadable for several reasons, such as being deleted. Most storage devices, such as magnetic hard disks, are broken into discrete, equally sized segments, each of which can only belong to one file. File fragment classification is applied by considering all segments of the storage device as fragments and attempting to classify them individually.

One of their challenges was providing error estimates based on large, publicly available collections of files, in order to give realistic estimates of the models' generalizability in a reproducible manner. Soon afterwards, they released such a collection, as described in section 3.1.

Another of their challenges was high accuracy, arguing that file fragment classifiers would need higher than 99 percent accuracy to be useful for file carving. This was based on the fact that that modern storage devices may contain billions of fragments, making thorough analysis of them impossible without an enormous labor budget or file fragment classifiers capable of mapping out the majority of the device contents automatically. Although 99 percent was a fairly high threshold, it was not a very radical challenge because the central goal for classification models is to make them as accurate as possible anyway.

A slightly more radical challenge was perfect precision for classifiers. Even if

[4]One notable exception is the JPEG format.

unable to identify all fragments, Roussev and Garfinkel argued that a classifier should not make any false predictions in order to be useful for file carving. The primary use of file fragment classification in file carving is to partially alleviate the combinatorial explosion of possible file reconstructions by eliminating large numbers of fragments from consideration; in order for this to work, the fragments must be eliminated conclusively. To aid in this, they suggested that classifiers should be allowed to give a response of *unknown* if unable to make a prediction with high confidence.

Their final challenge was that file fragment classifiers should be fast enough to process fragments at the same rate at which they can be read from storage devices. This was yet another practical consideration for making classifiers useful for file carving; achieving this speed would be necessary to allow scaling up to keep pace with the growth of storage device capacity.

In addition to formulating a set of practical requirements for classifiers, Roussev and Garfinkel also presented a new formulation of the file fragment classification problem based on their observations of the issues in previous work. They stated that a classifier should identify both the *primitive* data type of the fragment and any compound file types inside of which it contains evidence of being embedded.

They also argued that the only way to achieve all of the practical requirements was for classifiers to be *specialized*—hand-designed to recognize one type each based on the most specific knowledge of the format possible. This contrasted heavily with the previously prominent goal of constructing classifiers to recognize as many types as possible using automatically extracted patterns. Through several case studies as well as implementations of specialized classifiers achieving impressive results and meeting all of their challenges, they provided fairly convincing evidence for this argument.

2.4 Conti et al.'s Case for Generalized Approaches

In 2010, Conti et al. presented a different use case with a different set of practical requirements for file fragment classification models [10]. Rather than file carving, they mainly considered the application of binary mapping, in which the goal is to automatically produce a human-readable map or index of a large binary object—such as a very large file, a memory dump, or a storage device—consisting of the location, size, and type of each region, to serve as a navigational aid for manual analysis. File fragment classification is applied to binary mapping by using a fragment classifier either after or as part of segmenting the binary object into distinct regions, providing the types that are ultimately incorporated into the map.

Binary objects tend to be broken into regions of similarly formatted data. This can easily be observed through a visual that Conti et al. called a *byteplot*, an example of which is shown in Figure 1. These visuals and other high-level overviews of the structure of binary objects are highly useful aids for manual analysis, especially due to the inherent limitations of screen space on the amount of raw data which can be shown at once, such as when viewing the raw data in a hex editor. Figure 2 shows an example of manual analysis using a hex editor. As the objects under investigation grow in size and the relative portion of the objects visible at once shrinks, they become progressively more difficult to analyze manually using tools such as hex editors. For example, most consumer-grade hard drives sold at the time of writing contain more than one billion times the amount of data visible in Figure 2. Having high-level summaries for binary objects of this scale is critical for finding information of interest in a timely manner.

Conti et al.'s work is especially notable for its theoretical soundness. Like Roussev and Garfinkel, Conti et al. noted the issues with composite file types in previous file fragment classification research. Perhaps inspired by Roussev and Garfinkel's comments on the poorly defined "file types" of past work, and likely also due to the specific needs of binary mapping, they limited the types under consideration to *primitive types* and explicitly defined such types as "families of homogeneous data with closely related binary structure." They applied a distance-based file fragment classification model in a sliding window in order to produce a coarse binary map with these types.

Conti et al. envisioned a binary mapping model being embedded into a graphical tool to produce maps tightly integrated with other components including a raw data view and several visuals such as byteplots. Accelerating manual analysis in this way represents a different approach from that of Roussev and Garfinkel, who mainly aimed to produce high-performance automated file carvers—tools solving a related but different task. As such, the requirements differ between the two.

In contrast with Roussev and Garfinkel's strong advocacy for specialized classifiers, Conti et al. argued that generalized classification models would be necessary for the application of binary mapping. One reason was to allow recognition of undocumented types, for which specific information is not available, to attempt to rapidly extract relevant patterns automatically and avoid the overhead of manual reverse-engineering. Another reason was to easily allow recognition of newly encountered types; ideally, the hypothetical binary exploration program would allow a user to provide examples of an arbitrary type not included in the initial classification model and would then attempt to identify regions of that type in the map—specialized approaches would be impossible in this case because the types of interest would not be known at the tool's time of creation.

2.5 Rationale of This Study's Approach

One of the ideas behind this study is that generalized models constructed in the past may have been unable to meet Roussev and Garfinkel's accuracy challenge due to their own limitations—they simply may not have been powerful or flexible enough. Although Roussev and Garfinkel argue convincingly that specialized approaches are superior for file carving and similar applications, the space of possible generalized models is far from exhausted, so they cannot be conclusively said to be incapable of comparable performance.

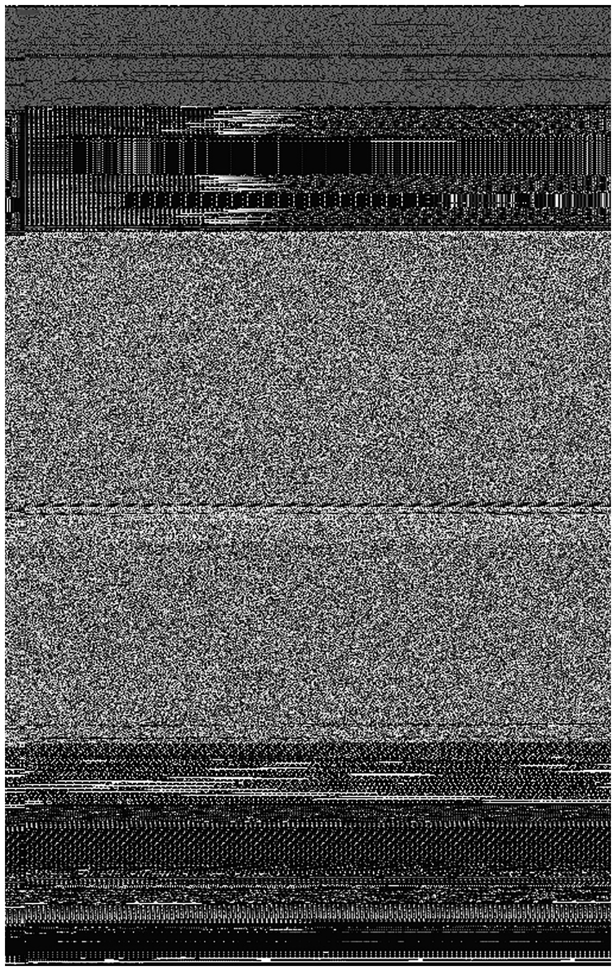
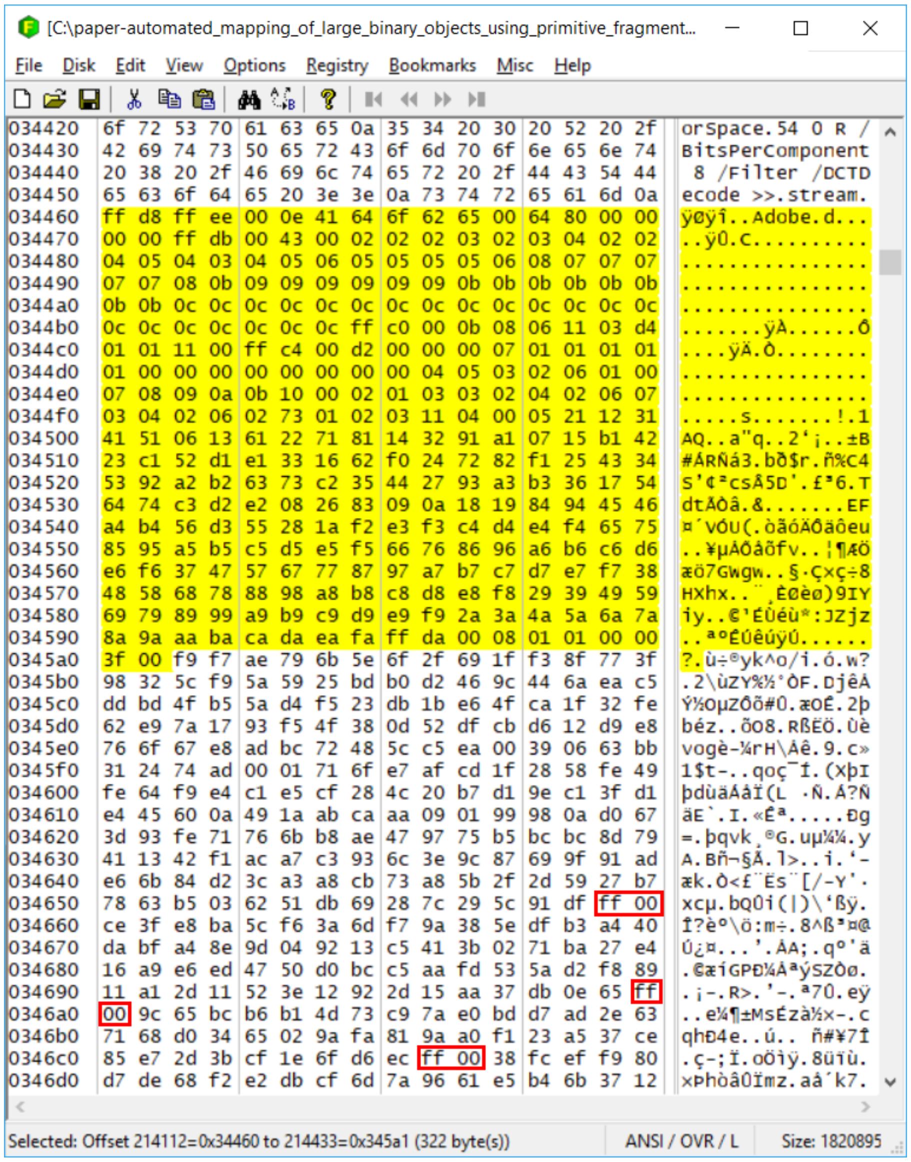


Figure 1: One of the byteplots constructed by Conti et al. appearing in [10], showing the contents of a Microsoft Word 2003 document and the dissimilarity of the patterns in different regions. Reproduced with permission.


[C:\paper-automated_mapping_of_large_binary_objects_using_primitive_fragment...
File Disk Edit View Options Registry Bookmarks Misc Help

```
034420  6f 72 53 70 61 63 65 0a 35 34 20 30 20 52 20 2f  orSpace.54 0 R /
034430  42 69 74 73 50 65 72 43 6f 6d 70 6f 6e 65 6e 74  BitsPerComponent
034440  20 38 20 2f 46 69 6c 74 65 72 20 2f 44 43 54 44   8 /Filter /DCTD
034450  65 63 6f 64 65 20 3e 3e 0a 73 74 72 65 61 6d 0a  ecode >>.stream.
034460  ff d8 ff ee 00 0e 41 64 6f 62 65 00 64 80 00 00  ÿØÿî..Adobe.d...
034470  00 00 ff db 00 43 00 02 02 02 03 02 03 04 02 02  ..ÿÛ.C..........
034480  04 05 04 03 04 05 06 05 05 05 05 06 08 07 07 07  ................
034490  07 07 08 0b 09 09 09 09 09 09 0b 0b 0b 0b 0b 0b  ................
0344a0  0b 0b 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c 0c  ................
0344b0  0c 0c 0c 0c 0c 0c 0c ff c0 00 0b 08 06 11 03 d4  .......ÿÀ......Ô
0344c0  01 01 11 00 ff c4 00 d2 00 00 00 07 01 01 01 01  ....ÿÄ.Ò........
0344d0  01 00 00 00 00 00 00 00 00 04 05 03 02 06 01 00  ................
0344e0  07 08 09 0a 0b 10 00 02 01 03 03 02 04 02 06 07  ................
0344f0  03 04 02 06 02 73 01 02 03 11 04 00 05 21 12 31  .....s.......!.1
034500  41 51 06 13 61 22 71 81 14 32 91 a1 07 15 b1 42  AQ..a"q..2'¡..±B
034510  23 c1 52 d1 e1 33 16 62 f0 24 72 82 f1 25 43 34  #ÁRÑá3.bð$r.ñ%C4
034520  53 92 a2 b2 63 73 c2 35 44 27 93 a3 b3 36 17 54  S'¢²csÂ5D'.£³6.T
034530  64 74 c3 d2 e2 08 26 83 09 0a 18 19 84 94 45 46  dtÃÒâ.&.......EF
034540  a4 b4 56 d3 55 28 1a f2 e3 f3 c4 d4 e4 f4 65 75  ¤´VÓU(.òãóÄÔäôeu
034550  85 95 a5 b5 c5 d5 e5 f5 66 76 86 96 a6 b6 c6 d6  ..¥µÅÕåõfv..¦¶ÆÖ
034560  e6 f6 37 47 57 67 77 87 97 a7 b7 c7 d7 e7 f7 38  æö7GWgw..§·Ç×ç÷8
034570  48 58 68 78 88 98 a8 b8 c8 d8 e8 f8 29 39 49 59  HXhx..¨¸ÈØèø)9IY
034580  69 79 89 99 a9 b9 c9 d9 e9 f9 2a 3a 4a 5a 6a 7a  iy..©¹ÉÙéù*:JZjz
034590  8a 9a aa ba ca da ea fa ff da 00 08 01 01 00 00  ..ªºÊÚêúÿÚ......
0345a0  3f 00 f9 f7 ae 79 6b 5e 6f 2f 69 1f f3 8f 77 3f  ?.ù÷®yk^o/i.ó.w?
0345b0  98 32 5c f9 5a 59 25 bd b0 d2 46 9c 44 6a ea c5  .2\ùZY%½°ÒF.DjêÅ
0345c0  dd bd 4f b5 5a d4 f5 23 db 1b e6 4f ca 1f 32 fe  Ý½OµZÔõ#Û.æOÊ.2þ
0345d0  62 e9 7a 17 93 f5 4f 38 0d 52 df cb d6 12 d9 e8  béz..õO8.RßËÖ.Ùè
0345e0  76 6f 67 e8 ad bc 72 48 5c c5 ea 00 39 06 63 bb  vogè-¼rH\Åê.9.c»
0345f0  31 24 74 ad 00 01 71 6f e7 af cd 1f 28 58 fe 49  1$t-..qoç¯Í.(XþI
034600  fe 64 f9 e4 c1 e5 cf 28 4c 20 b7 d1 9e c1 3f d1  þdùäÁåÏ(L ·Ñ.Á?Ñ
034610  e4 45 60 0a 49 1a ab ca aa 09 01 99 98 0a d0 67  äE`.I.«Êª.....Ðg
034620  3d 93 fe 71 76 6b b8 ae 47 97 75 b5 bc bc 8d 79  =.þqvk¸®G.uµ¼¼.y
034630  41 13 42 f1 ac a7 c3 93 6c 3e 9c 87 69 9f 91 ad  A.Bñ¬§Ã.l>..i.'-
034640  e6 6b 84 d2 3c a3 a8 cb 73 a8 5b 2f 2d 59 27 b7  æk.Ò<£¨Ës¨[/-Y'·
034650  78 63 b5 03 62 51 db 69 28 7c 29 5c 91 df ff 00  xcµ.bQÛi(|)\'ßÿ.
034660  ce 3f e8 ba 5c f6 3a 6d f7 9a 38 5e df b3 a4 40  Î?èº\ö:m÷.8^ß³¤@
034670  da bf a4 8e 9d 04 92 13 c5 41 3b 02 71 ba 27 e4  Ú¿¤...'.ÅA;.qº'ä
034680  16 a9 e6 ed 47 50 d0 bc c5 aa fd 53 5a d2 f8 89  .©æíGPÐ¼ÅªýSZÒø.
034690  11 a1 2d 11 52 3e 12 92 2d 15 aa 37 db 0e 65 ff  .¡-.R>.'-.ª7Û.eÿ
0346a0  00 9c 65 bc b6 b1 4d 73 c9 7a e0 bd d7 ad 2e 63  ..e¼¶±MsÉzà½×-.c
0346b0  71 68 d0 34 65 02 9a fa 81 9a a0 f1 23 a5 37 ce  qhÐ4e..ú.. ñ#¥7Î
0346c0  85 e7 2d 3b cf 1e 6f d6 ec ff 00 38 fc ef f9 80  .ç-;Ï.oÖìÿ.8üïù.
0346d0  d7 de 68 f2 e2 db cf 6d 7a 96 61 e5 b4 6b 37 12  ×Þhòâ ÛÏmz.aå´k7.
```

Selected: Offset 214112=0x34460 to 214433=0x345a1 (322 byte(s)) ANSI / OVR / L Size: 1820895

Figure 2: A simple example of manual analysis using a hex editor. The visible portion shows a JPEG image embedded in a PDF file, and the goal was to extract the image. Noteworthy patterns revealing the nature of content include the header of the image (highlighted), the PDF metadata for the embedded object visible above the image header in the right pane, and the recurring `FF 00` byte pattern characteristic of the JPEG format visible in the image body in the left pane. The program shown in this screenshot is Frhed, a popular free hex editor. The PDF file shown is a digital copy of [10], and the embedded JPEG image is the original copy of the image shown in Figure 1.

Another motivation for further exploration of generalized approaches is that they are sometimes necessary. As noted by Conti et al., in some applications such as binary mapping, specialized approaches are not always possible.

The goal of this study is to produce a universal file fragment classification model, inspired primarily by the observation that, besides flawed assumptions leading to attempts to perform impossible tasks, all file fragment classification models in the past have been inherently limited in their pattern recognition capacity and their ability to generalize to new types.

All generalized models for file fragment classification and file type identification in the past have used lossy representations for their input—none have been capable of using all of the information contained in the objects to be classified. In order to be universally generalizable, the representation must be lossless—any loss of information shrinks the space of recognizable patterns—and the individual bits are practically the only lossless representation. Additionally, the past models have often imposed additional inherent limitations on the patterns that may be recognized; to avoid these limitations, a model capable of recognizing arbitrary patterns must be used. As universal function approximators, neural networks meet this criterion.

Neural networks trained with bits as input should theoretically be able to function as universal file fragment classifiers. Whether they are able to achieve adequate performace in practice is the primary research question of this study.

3 Resources and Tools

This section describes the primary third-party resources and tools utilized in this study's experiments. The corpus from which fragments were extracted is publicly and freely available, and all of the software used is open-source.

3.1 Govdocs1

File fragment datasets used in this study were extracted from a file corpus created by Garfinkel et al. as part of an effort to encourage "a culture of rigor and insistence on the reproducibility of results" within the field of digital forensics [13]. In alignment with this effort, this study utilized their publicly available Govdocs1 corpus, and care has been taken to explicitly and unambiguously describe the dataset creation procedure (see section 4.1).

The Govdocs1 corpus is comprised of nearly one million files collected from public `.gov` domain websites. The files in the corpus are summarized in Table 1. A total of 63 file extensions appear in the corpus. PDF files are the most numerous, comprising more than one-fifth of the corpus by both size and number of files.

Since its creation, the Govdocs1 corpus has been used for model evaluation in many works in file fragment classification and file type identification [1, 6–8, 12, 27, 36].

Extension	**Count**	**Total Size (GB)**
pdf	231232	127.881
html	214567	12.311
jpg	109233	34.951
txt	78285	47.433
doc	76616	28.923
xls	62634	27.999
ppt	49702	119.787
gif	36302	2.852
xml	33458	8.093
ps	22015	26.969
csv	18360	3.267
gz	13725	8.467
log	9976	4.040
eps	5191	2.746
unk	5186	1.424
png	4125	1.054
swf	3476	1.812
dbase3	2601	0.019
pps	1619	3.544
rtf	1125	0.457

Table 1: Summary of files in the Govdocs1 corpus with the 20 most frequently occurring extensions. The files with the 43 remaining extensions account for only 0.7 percent of the corpus by number and 0.5 percent by size.

3.2 Python

All of the necessary code for the experiments was written in Python. Python is highly popular and has an increasing market share across a wide variety of disciplines. According to polls conducted on the website KDnuggets, the percentage of respondents who used Python as their main analytics, data science, and/or machine learning platform rose from 34 percent in 2016 to 41 percent in 2017; this placed Python as the most popular platform in 2017, followed by R at 36 percent [37].

Python was chosen especially due to the active and growing community of machine learning researchers and practitioners who use Python as their primary software platform, as well as the wide range of open-source tools developed and supported by this community. For example, the popular open-source `scikit-learn` library for Python, which provides a broad set of general-purpose machine learning utilities and numerous implementations of machine learning algorithms, had a lifetime total of 1,031 contributors as of the time of writing [38].

3.3 Keras

Neural network models were built and trained using Keras, an open-source deep learning library for Python [39]. At the time of writing, Keras was the most popular open-source Python library for neural networks[5], based on the number of "stars" on GitHub, with Keras having been bookmarked by 26,447 distinct users. Figure 3 compares the star counts of the most popular neural network libraries for Python.

Using Keras greatly decreased the iteration time of experiments. In comparison with implementations created from scratch, the premade, pre-tested implementations of neural network components and training algorithms greatly reduced the coding overhead and thus allowed a much larger portion of time to be devoted to experiments themselves.

3.4 Theano

Theano was used in conjunction with Keras, as its backend library for constructing models. The direct usage of Theano was limited to basic configuration.

Theano was chosen over the other two backend libraries supported by Keras—TensorFlow and CNTK—based on simple benchmarking tests. All three were tested, and Theano was chosen for superior performance (i.e. training speed) with the particular combination of models, datasets, and hardware used in this study. Extensive benchmarking was outside the scope of this study, and would have compromised the motivating objective of completing the experiments in a timely manner. If these experiments were to be replicated, performing simple tests to select the highest-performing backend would be recommended.

[5]Excluding lower-level libraries such as TensorFlow and Theano which are more general-purpose and provide utilities to support *implementing* neural networks.

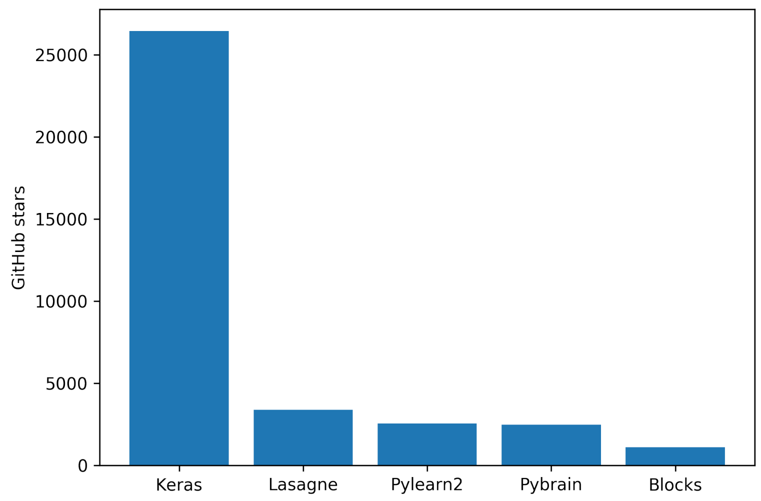


Figure 3: Number of "stars" on GitHub for several of the most popular Python libraries for neural networks, as of 6 March 2018.

An additional future consideration for selecting a backend is that maintenance of Theano will be discontinued shortly after the time of writing [40]. For any isolated, short-term experiments conducted fairly soon afterwards, this may not have a large impact, but the discontinuation of support could greatly affect Theano's long-term stability. Thus, choosing one of the other supported backends, TensorFlow or CNTK, is recommended.

4 Methods

Using these resources, two experiments were conducted. The commonalities of their procedures are discussed in sections 4.1 through 4.4, and their differences are discussed in section 4.5.

4.1 Dataset Extraction

File fragment datasets were sampled from the Govdocs1 corpus using the following procedure. First, the selection pool of files was formed by filtering out files of insufficient size–a 512 byte fragment was extracted from each selected file while excluding the first 512 bytes, so 1,024 bytes was the minimum file size. From this pool, an equal number of files were randomly selected for each chosen type. Each selected file was partitioned into fragments at 512-byte boundaries, and a random fragment was selected from each file, excluding the first, and excluding the last if its length was not a full 512 bytes.

The datasets were uniformly stratified by file type to avoid bias. Files and fragments were selected as described in order to give each file in the corpus equal likelihood of representation. To give each *fragment* equal likelihood of selection would decrease the expected number of represented files due to the greater chance of selection from larger files, contrary to the goal of capturing as much diversity as possible.[6]

The first fragment was excluded from each file to prevent the models from being skewed by header data, as performed in many works in file fragment classification [7–10, 12, 23, 27]. The last fragment was excluded if shorter than 512 bytes in order to maintain a fixed fragment size, because variable-length fragments are outside the scope of this study.

Fragments of 512 bytes were chosen because this corresponds to the smallest common size of hard drive sectors, which are the smallest unit which may need to be processed in isolation in file carving. The most popular fragment sizes among previous work have been 512 bytes [6–9, 12, 16, 31, 32] and 4,096 bytes [19, 20, 23, 27, 31, 34]. Penrose et al. argued that 4,096 bytes was a safe choice because all hard drive manufacturers have used this as their sector size since 2011 [27]. However, as noted by Axelsson, 512 bytes is a conservative choice [6], so this size was adopted.

Due to the exploratory nature of this study, a significantly smaller subset of the total quantity of data available in the corpus was selected for the experimental datasets to aid reproducibility. The size of the datasets was limited such that the total size of all feature vectors would not exceed what could fit into memory on an average personal computer or single-user workstation at the time of writing. This was done for the sake of computation time; samples could be stored in memory in a much more compact form, but repeatedly generating the feature vectors on demand would add a significant amount of time to training. Even greater overhead would result from only loading partial chunks of the dataset into memory at once.

In the absence of actual data on the distribution of available memory, a maximum dataset size of 1 GB was adopted. Assuming all features (8,192 per fragment) to be 32-bit floating point numbers and that features are stored with negligible overhead (both of which were true in these experiments), this translates to a maximum of 32,768 fragments in a single dataset.

4.2 Feature Representation

The feature representation for file fragments is a central component of this study's approach. The fragments are represented to the models by their bits, i.e. each individual bit was used as a feature. Using the bits as features is a lossless representation, unlike practically all of the representations used in prior work.

The bits are treated as categorical features, and are decomposed using the standard technique for encoding categorical data as real-valued inputs: two dummy variables are created for each bit, with one of them set to 1 and the

[6] This assumes that the variance of fragment contents in all files of a specific type is higher than the variance of the contents of fragments within a single file.

other set to 0, depending on the value of the bit. This is done to give the model equal and symmetrical capacity to recognize patterns based on bits of either value. Although neural network models of arbitrarily large size may theoretically be able to extract the same patterns regardless of whether the bits were split into two features, a model of fixed size would be artificially limited by singular features. Without being decomposed, bits with a value of 1 would have a much stronger influence on the model than bits with a value of 0. As an example, for a unit connected to the network input, e.g. in the first hidden layer for a feedforward network, inputs with nonzero value all have their own parameters (weights) for influencing the unit's activation, whereas all inputs with a value of 0 collectively have only a single parameter (the bias) because their weights have no effect.

With 512 bytes per fragment, 8 bits per byte, and 2 features per bit, each fragment is represented by a total of 8,192 features. This high dimensionality has varying impacts on networks of different architectures, as discussed in the following section.

4.3 Models

Feedforward, recurrent, and convolutional neworks were used in order to compare the performance of different network architectures. Recurrent networks were hypothesized to have both higher accuracy and higher computational cost than networks of the other two architectures; all three were tried to test this. Identical feature representations are used for the input for all models, but the input has a different shape for each architecture.

Input is provided to the feedforward networks in a flat format, i.e. as a single vector with 8,192 entries. This network architecture does not possess any qualities which mitigate the high dimensionality of the input data and was expected to be impacted the most by it.

The recurrent networks process one byte of input per time step: 16 features at a time for 512 steps. By only processing 16 features at a time, the recurrent networks have strong built-in mitigation for the high dimensionality of the input.

The convolutional networks are provided with input in the form of a 512×16 matrix, with one byte per row. The convolution is applied in one dimesion, with filters of size $n \times 16$, each filter mapping each sequence of n consecutive bytes (including overlapping sequences, i.e. with a stride length of 1) to a scalar processed by subsequent layers of the network. Convolutional networks seemed to be a natural fit for this task because they are designed both for handling high-dimensional input and recognizing shift-invariant patterns. Both of these qualities are highly desirable in this context.

4.4 Model Tuning

Each of the three network architectures also have many hyperparameters[7]. In these experiments, model selection was performed through a process of trial-and-

[7] I.e. parameters manually selected rather than learned.

error by iteratively evaluating a model with a specific set of hyperparameters and then manually selecting a new set by attempting to make an educated guess based on the cumulative observations. This process consumed more time than any other activity in this study.

Each hyperparameter set was evaluated using 8-fold cross validation. The number 8 was chosen as the closest number to the most popular choice of $k = 10$ for k-fold cross validation which would evenly divide all of the datasets used in these experiments. Although not strictly necessary, having even partitions slightly decreased the complexity of the cross-validation procedure.

Although the maximum dataset size of 32,768 samples was used for the final performance evaluation (as described in section 4.1), separate datasets of 2,048 samples were used for the model tuning process in each experiment. Using a smaller set greatly reduced the iteration time for hyperparameter tuning and thus allowed a larger number of configurations to be tested. A competing objective was that the datasets needed to be of sufficient size to obtain reasonably reliable performance estimates. The tuning set size was chosen to attempt to balance the speed of tuning with the variance in performance.

An additional benefit of using a separate dataset for tuning was that holding out a subset of the data for the final performance estimate was unnecessary. The tuning and evaluation sets were sampled independently, and although they were not ensured to be completely disjoint, the probability of them having a significant overlap is negligible. Thus, no further action was needed to avoid the performance metrics being artificially inflated by the tuning process due to selecting the minimum generalization error observed on a single set; these optimistically high generalization measures for the best model on the tuning set were simply discarded.

The final, reported performance evaluations were obtained by using the best hyperparameters found though the tuning procedure and retraining the models on the previously unseen full-sized evaluation set, performing one final round of 8-fold cross validation.

4.5 Experiments

In both experiments, sets of fragment types were selected with distinctive formats relative to each other, and datasets were constructed with equal numbers of each type. The two experiments differed mainly in the set of fragment types used for each.

In experiment A, fragments of CSV and XML files were used. This was intended as a minimal experiment where achieving perfect accuracy would clearly be possible in theory. A procedural rule-based classifier capable of perfect accuracy in distinguishing these types likely could be constructed without great difficulty.

Distinguishing CSV and XML fragments is a simple, conceptually clear task. They are both file types with distinctive, homogeneous formats, unlike most. These are among the few file types for which the central conceptual difficulty of file fragment classification is avoided—a careful consideration of their data encodings and how to construct meaningful groupings of their fragments into *types* is unnecessary.

Experiment B used fragments of CSV, XML, JPEG, and GIF files. This selection made experiment B a slightly more complex problem than experiment A, but it was still relatively simple in comparison with the general case in that fragments of these file types have mutually distinctive formats.

This set of types posessed several desirable qualities. Including CSV and XML in this set served as a test of the model's ability to retain its accuracy in making easy distinctions in the presence of more difficult distinctions. Adding JPEG and GIF to these two tested the model's ability to distinguish high and low entropy fragments.[8] Including both JPEG and GIF tested whether the model could separate high entropy fragments with distinctive patterns (JPEG) from those without obvious patterns (GIF).

Although the practical complexity of experiment B was roughly equal to that of experiment A, distinguishing these four types is a more conceptually complex problem. All four types have mutually distinctive formts relative to each other, but unlike CSV and XML files, JPEG and GIF files do not have homogeneous formats. Files of both of these types contain header sections which have very dissimilar formats from their bodies. Additionally, although GIF fragments are distinctive relative to the other three types, the body of a GIF file does not have a distinctive format at all in the general case—the body is encoded using the LZW compression scheme, which is shared by other file types in addition to being difficult to recognize due to the lack of obvious patterns. Although the body of a JPEG file is also compressed, the compression is rather unique due to the frequent occurrence of the byte sequence `FF 00` throughout the body, as discussed by Roussev and Garfinkel [28] and shown in Figure 2.

5 Results

This section presents the final results of the models selected for each network type in each experiment on the evaluation datasets. Several wall clock times are listed in the results, which are of course highly environment-dependent. The hardware used for these experiments was a laptop with 8 GB of memory, an Intel Core i7-6500U processor, and a NVIDIA GeForce 940MX GPU with 4 GB of dedicated memory. The experiments were run on Ubuntu 16.04 with Python 3.5.2 in a Jupyter 4.3.0 notebook using an IPython 6.1.0 kernel. For the models, Keras 2.0.8 and Theano 0.9.0 were used, with Theano configured to run on the GPU.

5.1 Experiment A

Table 2 shows the accuracy and training time of the best model for each network type in experiment A. Tables 3, 4, and 5 show the confusion matrices for the

[8] "Entropy" here refers to Shannon entropy; the higher the entropy of a fragment, the closer its contents are to uniformly random.

predictions of the feedforward, recurrent, and convolutional networks, respectively. The recurrent network was much slower, but the other two networks did not achieve comparable accuracy.

These performance evaluations represent the combined predictions of 8-fold cross validation on the large dataset using the hyperparameters chosen through the model selection procedure on the small dataset. The feedforward network had a single hidden layer of 16 hyperbolic tangent units and used an L_2 regularization coefficient of 0.07. The recurrent network had a single hidden layer of 32 Long Short Term Memory (LSTM) units and was trained with no regularization and a momentum of 0.9. The convolutional network had a convolutional layer of 2 filters with a width of 1 byte followed by a standard densely connected hidden layer with 16 hyperbolic tangent units, and it used an L_2 regularization coefficient of 0.05. All three networks were trained using stochastic gradient descent with a batch size of 256 and a learning rate of 0.01. The feedforward and convolutional networks were trained until convergence, but the loss of the recurrent network exhibited oscillating behavior during training, so it was stopped after 256 epochs of no observed improvement, and the model state that had yielded the lowest observed loss so far was restored.

All networks in both experiments had a number of sigmoid output units equal to the number of types they classified—2 output units for experiment A. When producing a prediction, each output unit independently gave a probability estimate in the open interval $(0, 1)$ for the fragment belonging to the corresponding type.[9] Thus, each network may be considered as a group of binary classifiers with shared intermediate representations. Predictions were obtained by selecting the type of the output unit with the maximum probability estimate. This was essentially a simple voting scheme among the binary classifiers, taking into account the types' mutual exclusivity.

All networks in this study used categorical cross-entropy loss, which may have negatively impacted the results. As discussed in section 6.2, a better choice would have been binary cross-entropy.

As suggested by Roussev and Garfinkel [28], the possibility of classifying a fragment as *unknown* was explored. This was done by imposing a minimum confidence required to give a prediction, giving a prediction of *unknown* if the maximum probability estimate fell below this threshold. This introduced a tradeoff between precision and recall, shown for the recurrent network in Figure 4. This tradeoff is based directly on the distribution of confidence in correct and incorrect predictions, which is shown for the recurrent network in Figure 5.

Most of the incorrect predictions were made with fairly low confidence, with a few outliers. Using a confidence threshold of 50 percent yielded perfect precision but a recall of only 52 percent; the confusion matrix for these predictions is shown in Table 6. A low recall was obtained because the confidence in correct

[9] This contrasts with the common practice in multiclass neural network classifiers of using a softmax output layer that produces a single probability distribution over the types. The rationale was to use a network architecture that could potentially support sets of types that are not mutually exclusive, e.g. having a single network for predicting primitive types as well as evidence of container types.

predictions was distributed very widely. Additionally, error bars on the required threshold for perfect precision are not known because this was only a single sample.

Network	**Accuracy**	**Training Time**
Feedforward	0.901947	0.95 hours
Recurrent	0.995789	78.75 hours
Convolutional	0.898163	7.30 hours

Table 2: Results summary for experiment A. The training time is the sum of the individual wall clock times across all 8 folds.

	CSV	**XML**
CSV	14546	1838
XML	1375	15009

Table 3: Confusion matrix for the feedforward network in experiment A. The rows correspond to the true type, and the columns correspond to the predicted type.

	CSV	**XML**
CSV	16290	94
XML	44	16340

Table 4: Confusion matrix for the recurrent network in experiment A.

	CSV	**XML**
CSV	14326	2058
XML	1279	15105

Table 5: Confusion matrix for the convolutional network in experiment A.

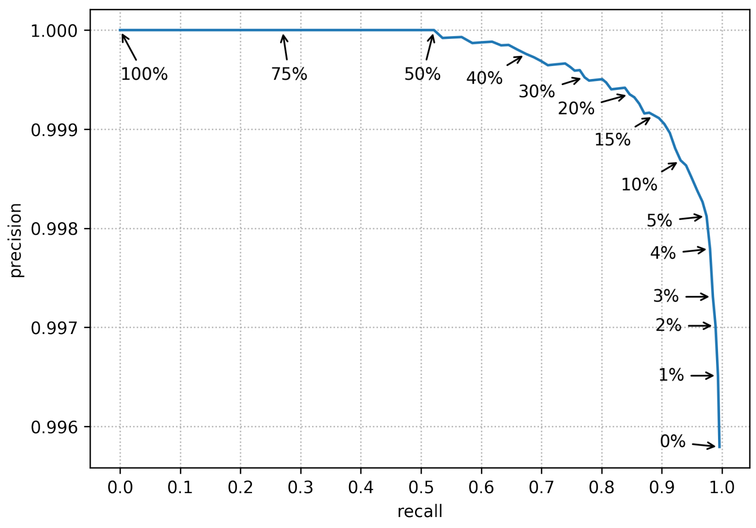


Figure 4: Precision-recall curve for the recurrent network in experiment A, obtained by varying the minimum confidence required to give a prediction. The labelled points indicate the corresponding confidence thresholds.

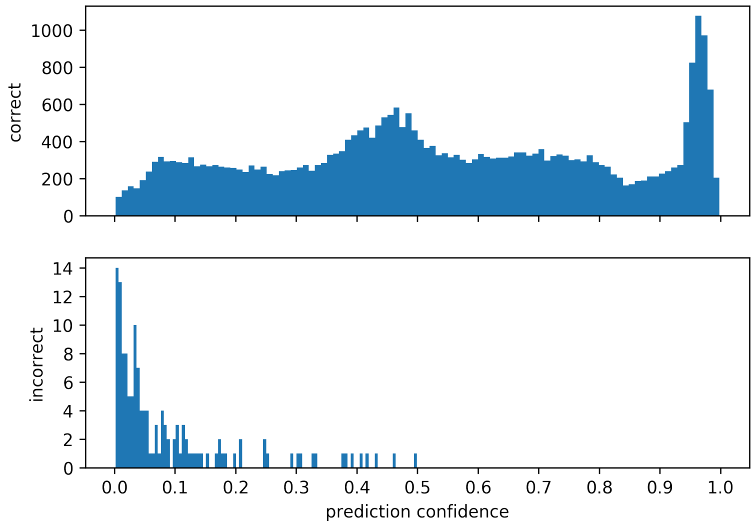


Figure 5: Distribution of confidence in the recurrent network's predictions in experiment A.

	CSV	**XML**	**?**
CSV	5894	0	10490
XML	0	11195	5189

Table 6: Confusion matrix for the recurrent network in experiment A with a confidence threshold of 50 percent.

5.2 Experiment B

Table 7 shows the accuracy and training time of the best model for each network type in experiment B. Tables 8, 9, and 10 show the confusion matrices for the feedforward, recurrent, and convolutional networks, respectively. As in experiment A, the recurrent network required much more training time, but it outperformed the others by an even larger margin in this experiment.

The feedforward network had a single hidden layer with 256 hyperbolic tangent units and was trained with an L_2 regularization coefficient of 0.11 and a momentum of 0.9. The recurrent network had a single hidden layer with 32 LSTM units and was trained with a momentum of 0.9 and no regularization. The convolutional network had a convolutional layer of 32 filters with a width of 1 byte followed by a layer of 32 hyperbolic tangent units, and it was trained with an L_2 regularization coefficient of 0.05 and a momentum of 0.9. Like experiment A, all three were trained with stochastic gradient descent with a batch size of 256 and a learning rate of 0.01, except for the convolutional network which used a learning rate of 0.03. The feedforward and convolutional networks were trained until convergence, and the recurrent network's training was stopped after 512 epochs of no observed improvement.

Like in experiment A, a varying minimum confidence threshold was applied to obtain the recurrent network's precision-recall curve, shown in Figure 6. Unlike the recurrent network in experiment A, this model was not able to achieve perfect precision while maintaining any significant portion of its recall. Compared with experiment A, the confidence in its incorrect predictions was slightly more widely distributed, and the confidence in its correct predictions was significantly lower, as shown in Figure 7. An example of the model's predictions with a confidence threshold of 5 percent is shown in Table 11; with this threshold, the model had a precision of 99.5 percent and a recall of 70.6 percent.

Network	**Accuracy**	**Training Time**
Feedforward	0.768494	2.15 hours
Recurrent	0.980438	134.11 hours
Convolutional	0.733856	7.37 hours

Table 7: Results summary for experiment B.

	CSV	**XML**	**JPG**	**GIF**
CSV	6659	1526	1	6
XML	697	7491	2	2
JPG	59	28	6717	1388
GIF	7	5	3865	4315

Table 8: Confusion matrix for the feedforward network in experiment B.

	CSV	**XML**	**JPG**	**GIF**
CSV	8115	73	4	0
XML	31	8151	10	0
JPG	3	32	7906	251
GIF	0	3	234	7955

Table 9: Confusion matrix for the recurrent network in experiment B.

	CSV	**XML**	**JPG**	**GIF**
CSV	6887	1298	1	6
XML	1082	7106	3	1
JPG	68	27	4474	3623
GIF	9	7	2596	5580

Table 10: Confusion matrix for the convolutional network in experiment B.

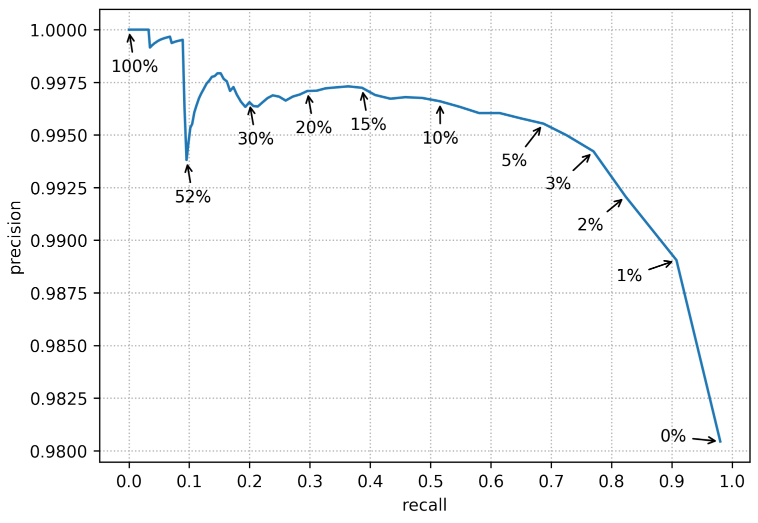


Figure 6: Precision-recall curve for the recurrent network in experiment B.

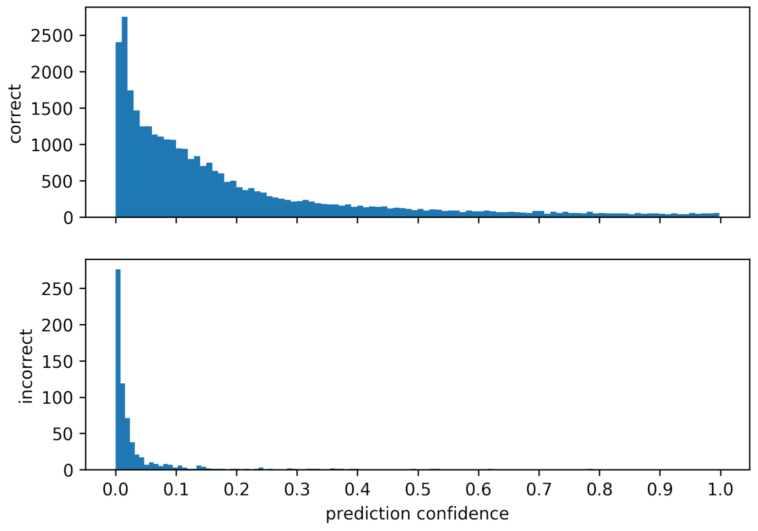


Figure 7: Distribution of confidence in the recurrent network's predictions in experiment B.

	CSV	**XML**	**JPG**	**GIF**	**?**
CSV	6778	8	0	0	1406
XML	2	4764	1	0	3425
JPG	2	16	5788	38	2348
GIF	0	0	28	5191	2973

Table 11: Confusion matrix for the recurrent network in experiment B with a confidence threshold of 5 percent.

6 Discussion

The results of these simple experiments are fairly promising, and they warrant further investigation of this approach.

6.1 Relative Performance of Network Architectures

Although the two experiments in this study constitute a fairly small sample, recurrent networks outperformed feedforward and convolutional networks by a sufficiently large margin to suggest that recurrent architectures may be the best suited for this task. Further work would be needed to determine this conclusively, but recurrent networks appear to be the most promising.

Perhaps one factor contributing to their superior performance is the lesser impact of the high dimensionality of the input on them. The recurrent networks only have 16 inputs for the one byte provided per time step, whereas the feedforward and convolutional networks have 8,192 inputs. This may lead these two types to require significantly larger training sets to achieve the same performance. Convolutional networks possess some built-in mitigation for high dimensionality through their shared parameters, but they appear to consistently perform slightly worse than feedforward networks in this area; this result was unexpected, and the reason for it is unclear.

Feedforward networks may benefit from dimensionality reduction techniques, but dimensionality reduction is impossible to perform losslessly, countering one of the core principles of this approach. Additionally, dimensonality reduction techniques would be practically rather difficult to apply in this context for two main reasons. First, fast linear techniques such as principal component analysis would be of limited use because most of the features of interest are nonlinear with respect to the bit representation. Second, nonlinear techniques such as autoencoder networks would be highly (possibly prohibitively) expensive to construct due to the very large number of samples required to capture most of the true distribution of fragments.

6.2 Improving Model Performance

The accuracy obtained so far is promising, and several methods seem likely to yield even better model performance. The simplest is using larger datasets, which are readily available, e.g. by extracting larger sets of fragments from Govdocs1. The upper bound on dataset size in these experiments was adopted mainly to avoid the overhead of streaming partial datasets to and from storage devices; experiments utilizing larger training sets would likely need to implement partial loading of datasets or would need to be executed on machines with significantly more memory than the hardware used in these experiments.

Another method for improving performance would be trying different training techniques. These experiments only used the standard stochastic gradient descent algorithm, but neural network research has produced many other adjusted training algorithms and techniques which may be valuable in this task. In particular, gradient clipping may be useful to counteract the chaotic training behavior and unpredictable dramatic increases in loss for the recurrent networks [14].

Automated model selection techniques would potentially be the most expensive method for improving performance but would likely be successful in finding better models than manual tuning alone. However, simple techniques such as grid searches should be avoided due to the very large number of hyeperparameters; a grid search procedure could easily be constructed for these models with a duration exceeding a human lifetime (without the inclusion of trivially small differences in parameter values). More sophisticated techniques should be employed to make better use of computing resources. In particular, neural network models selected through automated techniques based on Bayesian optimization have been demonstrated to be capable of surpassing the performance of models tuned manually by human experts [30].

One of the first methods of improving performance that should be tried is replacing the categorical cross-entropy loss with binary cross-entropy loss. The combination of sigmoid output units and categorical cross-entropy loss that was used in these experiments is problematic because, besides not being the true cross-entropy, incorrect predictions are not directly penalized during training. Given each probability estimate p_i for a single training example and corresponding target $y_i \in \{0, 1\}$ where $y_i = 1$ if the training example belongs to class i, the categorical cross-entropy is defined as

$$-\sum_i \big[y_i \ln p_i\big]$$

and the binary cross-entropy is defined as

$$-\sum_i \big[y_i \ln p_i + (1 - y_i) \ln (1 - p_i)\big].$$

Under minimization, the categorical cross-entropy drives p_i towards 1 if $y_i = 1$, but it lacks the second term of the binary cross-entropy and thus does not drive p_i towards 0 if $y_i = 0$. Part of the reason for this is that categorical cross-entropy is typically used with softmax output layers representing a single probability distribution over mutually exclusive classes, where increasing one output decreases all of the others [14]. However, since sigmoid output units were used, having no such interdependency, the probability estimates for the incorrect classes were simply ignored during training, likely contributing to the wide distribution in prediction confidence. This oversight was noticed after the experiments were completed.

6.3 Applicability

The current best models do not meet all of Roussev and Garfinkel's requirements for practical file fragment classifiers for file carving. The challenge of reliable error estimates was met by using a large dataset extracted from the Govdocs1 corpus, and the challenge of speed could be met through the use of parallelism if models were found with adequate performance. The challenges of perfect precision and 99 percent accuracy were not quite met; whether this approach would be applicable to file carving depends mainly on whether models could be found that are capable of achieving perfect precision while maintaining high recall. The techniques discussed in the previous section should be used to attempt to determine whether recurrent networks are capable of this.

Another potential application area for this approach is deep packet inspection. Classifying network packet payloads is a closely related problem, differing only in the fact that the target types are a superset of those in file fragment classification. Network intrusion detection would be the main potential application in this area, for which researchers have used similar models to those used in file fragment classification since before the emergence of any literature on the latter [35]. In particular, this approach may be capable of providing powerful heuristics for detecting malware in network traffic.

This approach does appear to be highly applicable to binary mapping, due to its need for generalized classifiers and less strict requirements on precision than file carving. The ability to construct universal classifiers is necessary to fulfill Conti et al.'s vision of creating a tool that allows users to identify arbitrary new types to be included in the mapping. Recurrent networks in particular also have a desirable quality: they could potentially be used to classify variable-length fragments, if a separate segmenting model were used in order to identify exact boundaries between regions.

6.4 Practical Considerations for a Binary Mapping Tool

Due to the rather high computational cost of training recurrent networks using this approach, several challenges would be faced in attempting to embed them into a binary mapping tool. These challenges would be mostly mitigated by embedding pretrained models into the tool designed to identify common types of interest, but this would not be possible for models allowing the user to specify new types to be included.

Initially, when the user has not yet requested any new types to be included in the map, the tool could use pretrained models. Then, when the user identifies a new type, the tool could begin constructing a new model in the background. The model construction process could potentially be extensively parametrized to give the user fine-grained control, or attempt to automatically select an appropriate set of options.

Rather than only attempting to train a recurrent network for identifying new types, the tool could try simpler models first and use them if they are able to achieve adequate performance. This would allow substantially reducing the cost in some cases and in other cases would provide some evidence to defend the high costs if recurrent networks were the only option with sufficient accuracy.

Recurrent networks could be tuned in the background using an automated procedure such as Bayesian optimization. The tuning could easily be parallelized, including across multiple machines; this could be offloaded to a high performance cluster in order to speed up model construction and to ease the burden on the user's workstation. Training of individual models could also potentially be parallelized across multiple machines by sharding the training data across nodes in a cluster, computing gradients locally and sending them to a controller, and broadcasting parameter updates from the controller to the gradient computation nodes.

Note that offloading computations to a model training cluster would not necessarily require any dedicated hardware. Due to the growth of cloud computing, a model training cluster could potentially be allocated, rescaled, and deallocated dynamically and automatically in the cloud, introducing a simple tradeoff between speed and cost. The tool could then offer the user straightforward cost management options such as only allocating model training resources when explicitly requested and user-specified hourly price limits.

7 Conclusion

These experiments aimed to serve as a preliminary test for the ability of neural networks using bits as input to function as generalized file fragment classifiers. The main contributions of this study are the proposal of a novel lossless feature representation for file fragments and the introduction of neural network architectures previously unused in this area and better suited for this task.

Future work should perhaps continue to perform simple benchmarks to compare the performance of different network architectures but should probably focus on recurrent networks. These outperformed feedforward and convolutional networks by a rather large margin.

Specific topics for future research include the following:

- Training models with binary cross-entropy to see if performance improves as expected.
- Training models with larger datasets, including ones too large to fit into memory.

- Trying adjustments to the training algorithm such as gradient clipping.
- Applying automated model tuning techniques such as Bayesian optimization.
- Training recurrent networks for classifying variable-length fragments.
- Preparing training sets with embedded fragments including container metadata and adding the container formats to the models' target types.
- Training recurrent networks for recognizing other types of fragments that may be encountered in binary mapping, such as memory pages of target executables.
- Training recurrent networks for identifying malware in network packets.

Promising results have been obtained for recurrent networks. Whether they will be able to achieve sufficient performance to function as universal file fragment classification models in a practical setting is an open question, but the answer to this question seems likely to be *yes*.

References

[1] Aaron et al., "Distributed Autonomous Neuro-Gen Learning Engine for Content-Based Document File Type Identification," in *Int. Conf. Cyber and IT Service Management*, South Tangerang, 2014, pp. 63-68.

[2] I. Ahmed et al., "On Improving the Accuracy and Performance of Content-Based File Type Identification," in *Australasian Conf. Information Security and Privacy, ACISP 2009: Information Security and Privacy*, Brisbane, 2009, pp. 44-59.

[3] I. Ahmed et al., "Fast Content-Based File-Type Identification," in *IFIP Int. Conf. Digital Forensics, DigitalForensics 2011: Advances in Digital Forensics VII*, Orlando, pp. 65-75.

[4] M.C. Amirani et al., "A New Approach to Content-Based File Type Detection," in *IEEE Symp. Computers and Communications*, Marrakech, 2008, pp. 1103-1108.

[5] M.C. Amirani et al., "Feature-Based Type Identification of File Fragments," *Security and Commun. Networks*, vol. 6, no. 1, pp. 115-128, Jan. 2013.

[6] S. Axelsson, "The Normalised Compression Distance as a File Fragment Classifier," in *Digital Forensic Research Conf., DFRWS 2010, Digital Investigation 7*, Portland, pp. S24-S31.

[7] N.L. Beebe et al., "Sceadan: Using Concatenated N-Gram Vectors for Improved File and Data Type Classification," *IEEE Trans. Inf. Forens. Security*, vol. 8, no. 9, pp. 1519-1530, Sep. 2013.

[8] N. Beebe et al., "Data Type Classification: Hierarchical Class-to-Type Modelling," in *IFIP Int. Conf. Digital Forensics, DigitalForensics 2016: Advances in Digital Forensics XII*, New Delhi, pp. 325-343.

[9] W.C. Calhoun and D. Coles, "Predicting the Types of File Fragments," in *Digital Forensic Research Conf., DFRWS 2008, Digital Investigation 5*, Baltimore, pp. S14-S20.

[10] G. Conti et al., "Automated Mapping of Large Binary Objects Using Primitive Fragment Type Classification," in *Digital Forensic Research Conf., DFRWS 2010, Digital Investigation 7*, Portland, pp. S3-S12.

[11] J.G. Dunham et al., "Classifying File Type of Stream Ciphers in Depth Using Neural Networks," in *3rd ACS/IEEE Int. Conf. Computer Systems and Applications*, Cairo, 2005.

[12] S. Fitzgerald et al., "Using NLP Techniques for File Fragment Classification," in *Digital Forensic Research Conf., DFRWS 2012, Digital Investigation 9*, Washington, DC, pp. S44-S49.

[13] S. Garfinkel et al., "Bringing Science to Digital Forensics with Standardized Forensic Corpora," in *Digital Forensic Research Conf., DFRWS 2009, Digital Investigation 6*, Montreal, pp. S2-S11.

[14] I. Goodfellow et al., *Deep Learning*, Cambridge, MA: MIT Press, 2016.

[15] S. Gopal et al., "Statistical Learning for File-Type Identification," in *10th Int. Conf. Machine Learning and Applications and Workshops*, Honolulu, 2011, pp. 68-73.

[16] R.M. Harris, "Using Artificial Neural Networks for Forensic File Type Identification," M.S. thesis, Perdue Univ., West Lafeyette, IN, 2007.

[17] G.A. Hall and W.P. Davis, "Sliding Window Measurement for File Type Identification," ManTech Security & Mission Assurance, 2006.

[18] X. Jin and J. Kim, "A Practical Video Fragment Identification System," *Int. J. Multimedia and Ubiquitous Eng.*, vol. 10, no. 6, pp. 165-176, June 2015.

[19] M. Karresand and N. Shahmehri, "Oscar—File Type Identification of Binary Data in Disk Clusters and RAM Pages," in *IFIP Int. Information Security Conf., SEC 2006: Security and Privacy in Dynamic Environments*, Karlstad, 2006, pp. 413-424.

[20] M. Karresand and N. Shahmehri, "File Type Identification of Data Fragments by Their Binary Structure," in *IEEE Information Assurance Workshop*, West Point, 2006, pp. 140-147.

[21] A. Kattan et al., "GP-Fileprints: File Types Detection Using Genetic Programming," in *European Conf. Genetic Programming, EuroGP 2010: Genetic Programming*, Istanbul, 2010, pp. 134-145.

[22] W.J. Li et al., "Fileprints: Identifying File Types by N-Gram Analysis," in *Proc. Sixth Annu. IEEE SMC Information Assurance Workshop*, West Point, 2005, pp. 64-71.

[23] Q. Li et al., "A Novel Support Vector Machine Approach to High Entropy Data Fragment Classification," in *Proc. South African Information Security Multi-Conf*, Port Elizabeth, 2010, pp. 236-247.

[24] M. McDaniel, "Automatic File Type Detection Algorithm," M.S. thesis, Dept. Comp. Sci., James Madison Univ., Harrisonburg, VA, 2001.

[25] M. McDaniel and M.H. Heydari, "Content Based File Type Detection Algorithms," in *Proc. 36th Annu. Hawaii Int. Conf. System Sciences*, Big Island, 2003.

[26] S.J. Moody and R.F. Erbacher, "SÁDI – Statistical Analysis for Data Type Identification," in *Third Int. Workshop Systematic Approches to Digital Forensic Eng.*, Oakland, 2008, pp. 41-54.

[27] P. Penrose et al., "Approaches to the Classification of High Entropy File Fragments," *Digital Investigation*, vol. 10, no. 4, pp. 372-384, Dec. 2013.

[28] V. Roussev and S.L. Garfinkel, "File Fragment Classification—The Case for Specialized Approaches," in *Fourth Int. IEEE Workshop Systematic Approaches to Digital Forensic Eng.*, Berkeley, 2009, pp. 3-14.

[29] V. Roussev and C. Quates, "File Fragment Encoding Classification—An Empirical Approach," in *Digital Forensic Research Conf., DFRWS 2013, Digital Investigation 10*, Monterey, pp. S69-S77.

[30] J. Snoek et al., "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems*, Lake Tahoe, 2012, pp. 2951-2959.

[31] L. Sportiello and S. Zanero, "File Block Classification by Support Vector Machine," in *Sixth Int. Conf. Availability, Reliability and Security*, Vienna, 2011, pp. 307-312.

[32] L. Sportiello and S. Zanero, "Context-Based File Block Classification," in *IFIP Int. Conf. Digital Forensics, DigitalForensics 2012: Advances in Digital Forensics VIII*, Pretoria, pp. 67-82.

[33] S.J. Stolfo et al., "Fileprint Analysis for Malware Detection," in *Proc. 2005 ACM Workshop Rapid Malcode*, Alexandria.

[34] C.J. Veenman, “Statistical Disk Cluster Classification for File Carving,” in *Third Int. Symp. Information Assurance and Security*, Manchester, 2007, pp. 393-398.

[35] K. Wang and S.J. Stolfo, “Anomalous Payload-Based Network Intrusion Detection,” in *Int. Workshop Recent Advances in Intrusion Detection, RAID 2004: Recent Advances in Intrusion Detection*, Sophia Antipolis, 2004, pp. 203-222.

[36] T. Xu et al. (2014), “A File Fragment Classification Method Based on Grayscale Image,” *J. Computers*, vol. 9, no. 8, pp. 1863-1870, Aug. 2014.

[37] G. Piatetsky, KDnuggets, “Python overtakes R, becomes the leader in Data Science, Machine Learning platforms,” 2017 [Online]. Available: http://www.kdnuggets.com/2017/08/python-overtakes-r-leader-analytics-data-science.html. Accessed 6 Mar. 2018.

[38] scikit-learn, https://github.com/scikit-learn/scikit-learn. Accessed 6 Mar. 2018.

[39] F. Chollet et al., Keras, https://github.com/keras-team/keras.

[40] P. Lamblin, “MILA and the future of Theano,” 28 Sep. 2017 [Online]. Available: https://groups.google.com/forum/#!msg/theano-users/7Poq8BZutbY/rNCIfvAEAwAJ