The ISC Int'l Journal of Information Security

July 2021, Volume 13, Number 2 (pp. 101–116)

http://www.isecure-journal.org

File Fragment Type Classification by Bag-Of-Visual-Words

Mina Erfan [1] and Saeed Jalili [1,*]

[1] *Computer Engineering Department, Tarbiat Modares University, Tehran, Iran.*

ARTICLE INFO.

Article history:
Received: August 15, 2020
Revised: September 15, 2020
Accepted: April 8, 2021
Published Online: May 8, 2021

Keywords:
Bag-Of-Visual-Words (BOVW), Digital Forensics, File Type Classification

Type: Research Article

doi: 10.22042/isecure.2021.243876.570

dor: 20.1001.1.20082045.2021.13.2.2.7

ABSTRACT

File fragment's type classification in the absence of header and file system information, is a major building block in various solutions devoted to file carving, memory analysis and network forensics. Over the past decades, a substantial amount of effort has been put into developing methods to classify file fragments. Meanwhile, there has been little innovation on the basics of approaches given into file and fragment type classification. In this research, by mapping each fragment as an 8-bit grayscale image, a method of texture analysis has been used in place of a classifier. Essentially, we show how to construct a vocabulary of visual words with the Bag-of-Visual-Words method. Using the n-gram technique, the feature vector is comprised of visual words occurrence. On the classification of 31 file types over 31000 fragments, our approach reached a maximum overall accuracy of 74.9% in classifying 512 byte fragments and 87.3% in classifying 4096 byte fragments.

1 Introduction

The classification of file fragments from data storage devices when criminals deleted file system information, is an important issue. Digital forensics analysts often have to reconstruct trace, log and evidence files which their indexes (e.g., i-nodes in unix and FAT32, NTFS in windows) are deleted by criminals. Whenever the storage device/filesystem remains intact, reconstruction of files can be easily performed using the metadata on filesystem. But in real scenario crimes, it is more likely that the metadata is also defected or even deleted entirely. During a process called file carving, reconstruction of such files must be conducted using the only remaining option for data source, the content of fragments themselves. Hence, determining the type of these fragments is considered as an essential step in file carving. Fragment type detection is also useful in other use cases. For instance, in network security domain, packets' payload data type examination could boost the early identification of malicious executable codes which would facilitate malware detection [1]. Moreover, it would also help with identifying damaged/spoofed headers in order to avoid true type detection by signature-based methods [2]. Due to the extremely large search space of determining whether or not a fragment belongs to a specific file type, it is crucial to develop an automated method for this purpose. Almost all commercial and off-the-shelf tools, such as TrID [3], are reliant on file signature and magic numbers. This makes them ineffective and numb wherever such data is missing, corrupted, or its location on the file sys-

* Corresponding author.
Email addresses: erfan.mina@gmail.com, sjalili@modares.ac.ir
ISSN: 2008-2045

tem is unknown. There has been little innovation on the basis and inner mechanism of most previous approaches to the file type classification; A review of past researches shows that most of them are just limited to use a number of statistical properties and/or byte frequency analyses. The research carried out by Pullaperuma *et al.* [4] made us think of this conjecture that texture analysis could also keep up with byte frequency and statistical properties analysis in fragment classification domain as an effective method. However, unlike Pullaperuma, we are looking for a method that is neither limited to few number of data types nor largely-sized fragments. To this goal, in our proposed approach, a new method for analyzing the texture of file fragments is provided that can differentiate more file types, even for the 512-byte fragments. The underlying idea is to find the specific visual words associated with each file type. A visual word represents a region of pixels with similar properties. By using a vocabulary of visual words, we can use the *Bag-Of-Visual-Words* (BOVW) technique. This idea has been used in image processing [5] but its capability and success in detecting file types is primarily investigated in this research for the first time. In our proposed method, the fragments are first handedly mapped onto images and visual words are extracted. Then, the number of occurrences of each visual word is counted as an n-gram in each fragment of the file and a feature vector is created for each fragment. By classifying these vectors, a model for predicting the type of file is fitted. Results on 31000 fragments have shown the effectiveness of the proposed method.

This paper is organized as follows. Section 2, gives a brief review of related work in this area. Section 3, characterizes our approach, which includes how to extract the vocabulary of visual words using BOVW, as well as detailed description on construction of feature vectors and classification model. Through Section 4 and Section 5, we bring along two extensions on top of our proposed method which tend to reduce false positives and expand applicability by segregation of outlier fragments and re-granulation of fragment sizes, respectively. In Section 6, parameters of the proposed method are evaluated. Results according to experiments are also represented and discussed. Finally, we conclude and suggest future work in Section 7 and Section 8.

2 Related Work

In the last two decades, a plethora of researches have been dedicated to file and fragment type classification. Particularly, recent works in this field explore the application of statistical analysis combined with machine learning techniques like artificial neural networks. Some of the more prominent solutions are outlined here.

2.1 Byte Frequency Distribution Approach

McDaniel and Heydari considered the identification of file type based on *byte frequency distribution* (BFD), *byte frequency correlation* (BFC) and header/footer analysis [6]. While the first two yielded unconvincing 27.5% and 45.8% accuracies respectively, the third one achieved 95.8% accuracy. However, it is not well-equipped for use cases with damaged files or missing header/footer information. Li *et al.* [2] conducted BFD analysis over the first 20, 200, 500 and 1000 bytes as well as the whole files to classify 8 file types. They exploited *K-means* clustering to create centroid for each file type. Best result is reported on 20 bytes blocks. Evidently, as the block sizes increased, accuracy decreased. This is due to the fact that with just the first 20 bytes in consideration, the method is limited to use magic numbers. However, increasing the number of bytes would decrease the influence of those factors. Ahmad *et al.* [7] developed the method used by Li *et al.* [2] to classify file types. They clustered files with similar BFDs, regardless of their type. Cosine similarity metric was used to assign an unknown fragment to a cluster. In each cluster, *Linear Discriminant Analysis* (LDA) method was used to make the distinction between file fragments. This method led to an overall classification accuracy of 77% for 10 file types. Kattan *et al.* [8] extracted features from BFD using *Principle Component Analysis* (PCA) and gave feature vectors to a multilayer *Artificial Neural network* (ANN) to produce fileprints. On classification of 6 file types, 98% average true positive was reported. Since experiments have been performed on complete files having header, PCA may have extracted files header information as the most outstanding component [9]. Karresand and Shahmehri [10] introduced new method named *Oscar* which used BFD vectors to classify fragment types. Oscar was later leveraged to utilize *Rate of Change* (ROC) for consecutive bytes [11], which increased the accuracy of 4KB JPG fragments detection up to 99.2%. Unfortunately, ROC could not help with enhancing the classification of other file formats. Karampidis *et al.* [12] proposed a methodology for file type classification. A three stage process including feature extraction using BFD vectors, feature selection utilizing genetic algorithm and

classification by examining multiple machine learning algorithms was conducted over 3 image file types. They achieved their best results by applying neural networks which reportedly led to accuracy of 100%, 98.81% and 100% for JPG, PNG and GIF file types respectively.

2.2 Other Statistical Approach

Veenman [13] proposed statistical method which combined *BFD*, *Shannon entropy* and *Kolmogorov complexity* as the set of features. In order to classify file types of fragments, 4KB each, Fisher linear discriminant was applied. An overall accuracy of 45% was reported for 11 file types. Erbacher and Mulholland [14] used statistical features calculated from BFD vectors over *sliding windows* of sizes 64, 256, 1024, 4096 and 16384 bytes. They showed window sizes of 256 and 1024 bytes are most effective. They claimed that only 5 features were sufficient to classify 7 file types. Moody and Erbacher [15] elaborated on that idea. They used a sliding window of 256 bytes in size. They extracted a variety of statistical features (such as average, kurtosis and standard deviation) from byte values in each window in order to identify fragments of 8 file types. They reported an average classification accuracy of 72%. Calhon and Coles [16] attempted to classify 4KB fragments of 4 file types. They extended Veenman's work [13] by using additional features (such as Longest Common Subsequence) and applying *Fisher Linear Discriminant*. The main distinction to their work is that Veenman fit one linear discriminant function for all files in the sample, while Calhon and Coles clustered files with similar BFD and trained the discriminant for each cluster. They have also chosen pairwise classification over multi-type classification. Hence, so that there is a lesser chance of misclassification. Axelsson [17] made use of *K-Nearest-neighbor* (KNN) to classify 512-byte fragments from 28 file types. He applied *Nearest Compression distance* (NCD) as a measure of similarity. Average classification accuracy reported is about 35%. Li *et al.* [18] used a *Support Vector Machine* (SVM) to classify 4KB fragments of 4 file types using BFD vectors. An average classification accuracy of 81.5% was achieved in their experiments. Conti *et al.* [19] used *Shannon entropy*, *mean byte-value*, *chi square goodness of fit* and *Hamming Weight* to provide the statistical coordinate system for a KNN classifier with Euclidean distance measure in order to classify 1KB binary fragments. They arrived at 98.55% accuracy for Random/Compressed/Encrypted fragments, 100% for both Base64 encoded and plain fragments, 96.7% for machine code, 98.7% for Text and 82.5% for bitmap fragments. Penrose *et al.* [9] focused on classifying fragments of high entropy file types, specially encrypted and compressed files. They proposed two methods to detect randomness along with a second phase classifier as an Artificial Neural Network (ANN). *NIST statistical tests* and *usage of compressibility* as two separate measures of randomness lead to 91%, 82%, 76% and 70% accuracy for encrypted and compressed 4KB fragments, respectively. Zheng *et al.* [20] proposed a method to classify fragments using data types instead of file types. This way, fragments of PNG and GZ files are labeled as deflate data type instead of their corresponding file types. They used an SVM supported by BFD and Shannon entropy as input parameters to classify 512-byte data fragments. The result depicts an 88.58% accuracy over 12 data types which indicates a 21.2% bump regarding file type classification accuracy.

2.3 N-gram Approach

Cao *et al.* [21] presented a file type classification algorithm which distinguishes four file types using 1-gram and 2-gram analysis. They also proposed a feature selection evaluation function to select grams with strongest discrimination power. This approach seems promising when 256 2-gram features are used, which reportedly reaches an average F-value of 85%. Gopal *et al.* [22] evaluated several commercial off-the-shelf softwares against some classification methods such as SVM and KNN using cosine similarity distance metrics. However, the results are only reported on the performance of SVM. The proclaimed accuracy, using the *macro-averaged F1 measure*, for classification of 512 byte file fragments is about 33%. Fitzgerald *et al.* [23] extended on Li *et al.* [18] work. They utilized 1-gram and 2-gram as well as some statistical measures (such as Shannon entropy, Hamming weight and Kolmogorov complexity) to identify 512 byte fragments of 24 different file types using an SVM discriminator. An average classification accuracy of 42.5% is reported. Beebe et al. [24] made research on 38 file and data type classification and developed a study called Sceadan. They gathered several byte frequency based measures (such as Hamming weight, low ASCII frequency, medium ASCII frequency, high ASCII frequency) as well as 1-gram and 2-gram features and planned for comparative examination of

different input vectors, SVM kernels, and SVM parameters in their experiments. 73.4% classification accuracy is reported using a linear function since the SVM kernel concatenated 1-gram and 2-gram features as the input vector. Divakaran *et al.* [25] carried out an investigation to evaluate different set of features (such as n-gram frequencies, entropy, mean, standard deviation and kurtosis) as well as stream size for classification of 10 file types in network traffic. Multiple scenarios were performed and 89% accuracy was attained for random streams size of 6000 byte, although the classification algorithm application and specifications are left unclear and asks for more clarification. Vulinović *et al.* [26] applied *Feed-forward Neural Networks* (FNNs) which were trained with 1-grams and 2-grams to classify 512-byte fragments of 18 file types. The classifier achieved the classifier macro-averaged F1 score of 88%. Bhatt *et al.* [27] explored a hierarchical classification approach to identify 512-byte file fragments of 14 file types. They evaluate their model applying SVM as base classifier. Their experiment resulted in an average accuracy of 66% and also F1 measure of 66% using 1-grams and 2-grams as well as some other statistical features (such as mean byte-value, Hamming weight and Contiguity). Skračić *et al.* [28] explored classification of 18 file types using 1-grams as discriminating feature. They used a dataset of 512-byte blocks to evaluate their approach. They developed an idea of merging older MS-Office file formats (doc, ppt, and xls), and new Office formats (docx, pptx, and xlsx) into two separate higher-level classes to simply develop a hierarchical classification approach. Using an ensemble of classifiers such as FNNs, SVMs, *Random Forests* (RFs) and *Vector Logic Regression* (VLR), an overall accuracy of 72.66% was obtained.

2.4 Novel Approach

Conti *et al.* [29] argued that multi-type files can be distinguished when they are depicted as grayscale images, thus facilitating various manual forensic analysis tasks. Due to the embedding of multiple data types within a single file, multi-type files have different regions with categorical and structural distinctions to each other. Pullaperuma *et al.* [4], inspired by this idea, presented a novel approach to file fragment classification. They considered a file fragment as a grayscale image from which, features would be extracted using *Gray Level Co-occurrence Matrix* (GLCM). KNN is then applied to classify fragments of 7 data types which resulted in 86.86% accuracy for 64×64 sized fragments but degraded for smaller fragments. There has been little innovation on the basis and inner mechanism of most previous approaches to the file type classification; A review of past researches shows that most of them are just limited to use a number of statistical properties and/or byte frequency analyses. The research carried out by Pullaperuma *et al.* [4] made us think of this conjecture that texture analysis could also keep up with byte frequency and statistical properties analysis in fragment classification domain as an effective method. However, unlike Pullaperuma, we are looking for a method that is neither limited to few number of data types nor largely-sized fragments. To this goal, in our proposed approach, a new method for analyzing the texture of file fragments is provided that can differentiate more file types, even for the 512-byte fragments. The underlying idea is to find the specific visual words associated with each file type. A visual word represents a region of pixels with similar properties. By using a vocabulary of visual words, we can use the BOVW technique. This idea has been used in image processing [5] but its capability and success in detecting file types is primarily investigated in this research for the first time. In our proposed method, the fragments (of all types) are first mapped onto images and visual words are extracted. Then, the number of occurrences of each visual word is counted as an n-gram in each fragment of the file and a feature vector is created for each fragment. By learning a classifier using these feature vectors, a model for predicting the type of file is fitted.

3 Proposed Method

The architecture of the proposed method presented in Figure 1, consists of two sections of training and testing. In the training phase, we try to learn a model for predicting the file type of fragments. In this case, after extracting file fragments, they are mapped to grayscale images (by considering each byte as a pixel), and then, for each pixel in each image a feature vector is extracted. By clustering these vectors, the center of each cluster, or in other words, a region of pixels with similar properties are selected as visual words. Next, a vector of visual words occurrences is created in n-gram terms for each image (in other words, each fragment). Finally, these vectors are used to learn a model for predicting the file type of fragments. In the testing phase, an unknown file fragment is first mapped into a grayscale image and vectors of its pixels properties are extracted. By comparing these vectors with the

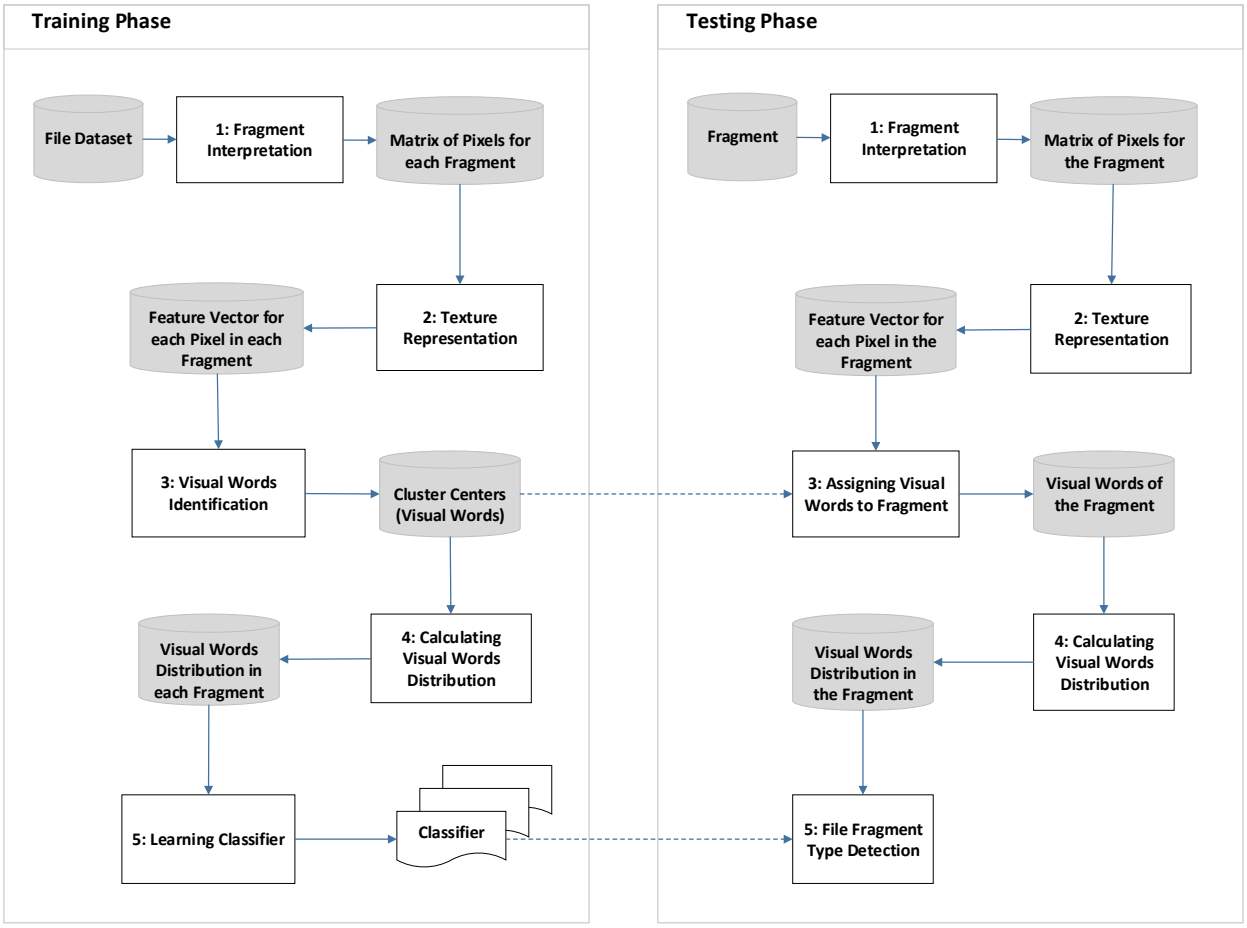

Figure 1. Proposed method architecture

cluster centers (visual words) obtained during the learning phase, each vector is assigned to a visual word. By calculating visual words occurrences as n-gram, a vector of visual words distributions is created, which is given to the learned model to identify the type of the fragment.

3.1 Training Phase

3.1.1 Fragment Interpretation

Each file fragment is 512-bytes long. Given that each byte out of a fragment has a value between 0 to 255, it can be interpreted as an image pixel with a fixation on the spectrum of 256 possible shades of gray. Hence, fragments could be considered as an 8-bit grayscale images. To achieve this, every fragment is organized into a 2D matrix whose dimensions corresponds to the final dimensions of the grayscale image and each byte of fragment represents one matrix element. These dimensions are one of the key parameters which have been studied in this research. Considering the length of the fragments (512 bytes), every feasible dimension was brute-forced. To make it clear assume a 9-byte fragment whose bytes contain values 1 to 9 respectively. Table 1 shows how it could be fit into a 3×3 matrix.

Table 1. A 9-byte fragment as pixel matrix: each byte of the fragment is an element in the matrix

1	2	3
4	5	6
7	8	9

3.1.2 Texture Representation

The first step is to create a search space of visual words in an image $\mathbf{I}$ with $M \times N = 512$ pixels and gray values between 0 and 255. It has been pointed out that a visual word represents a region of pixels with similar properties. To determine these regions a radius r is intended and all possible regions with at most radius r get extracted. Evidently, pixels within a single region

are considered neighbors and a connection between them is expected. Let $x = 1, \ldots, M$ and $y = 1, \ldots, N$ be the cartesian coordinates of the pixel (x, y). Two pixels $p_1(x, y)$ and $p_2(x', y')$ are connected if their Euclidean distance is smaller than a radius r.

$$E = \left\{ e = (p_1, p_2) \in \mathbf{I} \times \mathbf{I} \,\middle|\, \sqrt{(x - x')^2 + (y - y')^2} \leq r \right\} \quad (1)$$

For each $e \in E$ relation, a weight $w(e)$ is assigned. As indicated by the Equation 2, this weight is determined by the square of the Euclidean distance between two connected pixels and the difference in gray intensity of the two pixels, normalized by the square of the radius r [30]. $\mathcal{L}_{p_i} \in [0 - 255]$ indicates the intensity of light in pixel p_i, where its maximum value in the image is L.

$$w(e) = (x - x')^2 + (y - y')^2 + r^2 \frac{|\mathcal{L}_{p_1} - \mathcal{L}_{p_2}|}{L}, \quad \forall e \in E. \quad (2)$$

Since the weight function $w(e)$ might contain a wide range of values, it is convenient to be normalized into the [0,1] interval. This is delivered by using the maximum conceivable value of the geometric distance between the two neighboring pixels, r^2, and the normalized maximum feasible difference in shades of gray, r^2. Hence, Equation 3 [30] manifests the final derivation for calculating the weight of a pixel relation.

$$w(e) = \frac{(x - x')^2 + (y - y')^2 + r^2 \frac{|\mathcal{L}_{p_1} - \mathcal{L}_{p_2}|}{L}}{r^2 + r^2}. \quad (3)$$

After computing weights on pixel relations, each pixel can be characterized by a union of its intensity and a vector of its weights to all its related neighbors (Equation 4).

$$\psi(p_i) = \left\{ w_{e_{i,j}} \,\middle|\, \forall e_{i,j} \in E \right\} \bigcup \mathcal{L}_{p_i}. \quad (4)$$

Algorithm 1 simply indicates the process of extracting pixel feature vectors which is described above. Let X be a set of file fragments. Each fragment $f \in X$ is converted to a graysclae image. The algorithm iterates over all pixels of all images. For each pixel in each Image, a region with radius of r, i.e., a $(2r+1) \times (2r+1)$ window, around the pixel is visited to calculate the weight of the pixel association to its neighbor pixels based on Equation 3. After extracting these weights, each pixel can be represented by a union of its intensity and a vector of its weights to all its neighbors. Considering an image I having N pixels, a $(2r+1) \times (2r+1)$ window around each pixel is visited to calculate the weight of the pixels associations to their neighbors. Therefore, the complexity of this algorithm is $(2r+1) \times (2r+1) \times N$. Since the parameter r is small in comparison with N, so it can be stated that computational complexity of the algorithm is N.

Algorithm 1: Extracting pixels eeature vectors

Input: $X = f_1, f_2, , f_m$, a dataset of fragments
r : radius
E : set of connections of pixels
Output: $D = \psi(p_1), , \psi(p_{512 \times m})$: Set of feature vectors

```
Function FeatureVectors(X, radius, E):
    foreach f ∈ X do
        Convert f to a 2D matrix I(grayscale image I)
        foreach pixel p_i in I do
            foreach pixel p_j inside a (2r+1) × (2r+1) window do
                compute w based on equation (3)
            end
            ψ(p_i) = { w_{e_{i,j}} | ∀e_{i,j} ∈ E } ⋃ L_{p_i}
        end
    end
End Function
```

The length of vector ψ depends on the r value which determines the maximum number of neighboring pixels. Considering that the boundary pixels have fewer neighbors in compare to middle pixels, zeros must be added to the boundary pixels feature vectors until all feature vectors have equal length.

3.1.3 Visual Words Identification

Bag of words (BOW) technique is commonly used in the context of text classification. Its main idea is based on creating a histogram of words occurrences in the text. *BOVW* is in part a derivation of the BOW along the same general lines. That said, its only contrast is the adoption of a selective feature set for the range of pixels in place of text words in BOW. Therefore, a visual word serves as a range of pixels with similar characteristics [5]. To identify these visual words, for all pixels p_i in all fragments in training set, pixel properties are conceptualized and extracted as the pixel feature vector (512 vector for each fragment) $\psi(p_i)$ from Equation 4. A clustering method is applied on pixel properties which returns $C = \{c_1, \ldots, c_k\}$ centroids each corresponding to a visual word. Given the centroids, it is trivial to assign a visual word to each vector $\psi(p_i)$ itself, according to its Euclidean distance to the associated centroid. In this study, we choose *mini-batch K-means* clustering algorithm [31] which is a breed of general K-means often used to reduce the computational time. This algorithm selects and processes a subset of the input-data, *mini-batches*, in each iteration. Mini-batches recede the amount of computing needed whenever

convergence to a localized solution is considered to be sufficient.

3.1.4 Calculating Visual Words Distribution

The N-gram analysis has been used in various fields. N-gram is a subsequence of N data items from a given sequence. It is a type of probabilistic model for estimating the probability that the next items replicate the preceding ones [21]. The single item tokens are called unigrams, but tokens consisting of any fixed number of item can also be considered. Two consecutive item tokens are called bigrams, and bigram counts capture more information about the structure of the data being classified than do unigram counts alone. In this research, visual words are contemplated as data items or grams. Therefore, the number of 1-gram and 2-gram occurrences would be accounted and enumerated. Given that, for each fragment in the training set, a vector of visual words occurrences is calculated.

3.1.5 Learning Classifiers

Given that the purpose of this study is to classify 31 file types, we face a multiclass problem. In this research, *one vs. all* strategy is used to solve the problem, in which for each file type a classifier is learned using the vectors of the visual words occurrences for fragments of that file type as positive-label data and the vectors of the visual words occurrences of other file types fragments as negative data. As a result, for each file type, a binary classifier separates it from the others. Thus the multiclass problem is solved. In this research, support vector machines (SVMs) machine learning algorithm is used for classification. SVM is a supervised machine learning algorithm widely used for the purpose of classification as well as regresseion. It is based on statistical learning theory and was developed by Vapnik [32]. SVM creates a discriminant between classes C_1 and C_2 that divides the input space in two. The decision regions R_1 for C_1 and R_2 for C_2. Such discriminant, known as hyperplane is implemented such that it is as distant as possible to the closest instances from each of the classes. These closest instances are termed support vectors. Given a labeled training set $X = \{x^t, r^t\}$ where $r^t = +1$ if $x^t \in C_1$ and $r^t = -1$ if $x^t \in C_2$. A hyperplane could be defined as

$$wx^T + w_0 = 0, \tag{5}$$

where w^T is the weight vector and w_0 is the threshold. w_0 determines the location of the hyperplane and w^T determines its orientation. w^T and w_0 would be found such that

$$r^t(wx^T + w_0) \geq 1. \tag{6}$$

The optimal hyperplane not only separates the instances but also maximizes the total margin $\frac{2}{||w||^2}$ or minimizes $\frac{1}{2}||w||^2$ (see Figure 2). Therefore, the task can be transformed to an optimization problem defined as

$$\min \frac{1}{2}||w||^2 \quad subject \quad to \quad r^t(wx^T + w_0) \geq 1, \quad \forall t. \tag{7}$$

In addition to performing linear classification, SVM can efficiently perform a non-linear classification using a kernel function, implicitly mapping data inputs into high-dimensional feature spaces. The most popular, general purpose kernel functions are

- linear:
$$k(x_i, x_j) = x_i x_j \tag{8}$$
- polynomial:
$$k(x_i, x_j) = (x_i x_j + 1)^q, \text{ for } q > 1 \tag{9}$$
- RBF:
$$k(x_i, x_j) = \exp\left[-\gamma(||x_i - x_j||^2)\right], \text{ for } \gamma > 0 \tag{10}$$
- sigmoidal:
$$k(x_i, x_j) = \tanh(2x_i x_j + 1) \tag{11}$$

In this study, different SVM's kernels have been investigated and their results are provided in the next section.

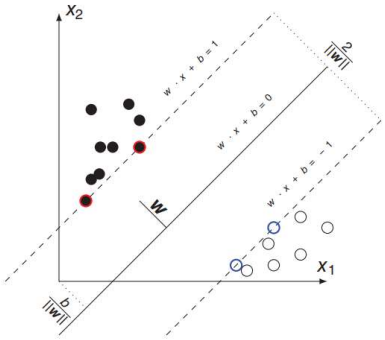

Figure 2. Maximum-margin hyperplane and margins for a SVM trained with samples from two classes [33]

3.2 Testing Phase

3.2.1 Fragment Interpretation

The fragment must be organized into a 2D matrix with dimensions corresponding to those learned in the training phase. This will consider each byte of the fragment as a pixel indicating the gray intensity.

3.2.2 Texture Representation

The connection between each pixel and its neighboring pixels in radius r, is weighted by the Equation 3. The properties of the region around each pixel can be represented by a vector containing the weight of the pixel connections. This vector was accounted by the Equation 4.

3.2.3 Assigning Visual Words to Fragment

After extracting the pixels feature vectors, the similarity of these vectors with the visual words learned in the training phase (centers of learned clusters) is measured by the Euclidean distance between the feature vectors and each center, which results in the designation of its corresponding visual word.

3.2.4 Calculating Visual Words Distribution

As provided in Section 3.1.4, n-gram is a contiguous subsequence of n items from a larger sequence. An n-gram of one item and two items is termed as a "1-gram" and "2-gram" respectively. The number of 1-gram and 2-gram of visual words occurrence are obtained for the test sample. This vector indicates the frequency distribution of visual words in terms of 1-grams and 2-grams. 1-grams demonstrate the number of times each visual word appeared in an image (fragment) and 2-grams, count all different subsequences of 2 visual words in the image.

3.2.5 File Fragment Type Detection

Finally, the vector of visual words occurrences of the fragment is given to the model learned in the training phase. As a result of *one vs. all* strategy, 31 classifiers are learned in the model. The output of each classifier is the probability of belonging to the file type the learner has learned. Hence, the fragment will be labeled as the class with higher probability.

4 Extending the Proposed Method to Reduce False Positive Rates

The false positive rate is an important criterion in determining the type of file fragments. Since the fragment type detection is mainly done for the purpose of file carving, in the event of type misclassification, the file reconstruction process would lead to the creation of a file whose content might be partly corrupted. Hence, in order to reduce the false positive rate, a threshold is considered as *minimum classification assurance* for each file type. For each class C_i with false positive rate of more than one percent, a threshold $\mathcal{T}_i$ is defined for the probability of assigning a sample to that class. If a test sample, get labeled to a class C_i with a probability lower than the class threshold $\mathcal{T}_i$, then the sample is reassigned to *unknown* class. The threshold for the class C_i is defined using equation Equation 12. In this formula, $mean_i$ is equal to average probability that a sample from class C_i emerge as false positive; dev_i is equal to the average standard deviation that a sample from class C_i turn out as false positive. $mean_i$ and dev_i are in part given by Equation 13 and Equation 14, respectively. In these equations, $\|FP_i\|$, is the number of all false positive samples in the class C_i which is simply calculated using confusion matrix. This matrix is automatically generated during classification in testing phase. In the case of $K \geq 2$ classes, the class confusion matrix is a $k \times k$ matrix whose entry (j, i) contains the number of instances that belong to C_j but are assigned to C_i. $prob\,[FP_n]$ is the probability of the nth false positive sample from class C_i which is obtained directly using SVM classifier's probability estimation outputs for predicted labels.

$$\mathcal{T}_i = mean_i - \beta \times dev_i, \tag{12}$$

$$mean_i = \frac{1}{\|FP_i\|} \sum_n prob\,[FP_n], \tag{13}$$

$$dev_i = \frac{1}{\|FP_i\|} \sum_n (mean_i - prob\,[FP_n])^2. \tag{14}$$

5 Extending the Proposed Method for 4096 Byte Fragments

Given that the new hard disks are 4096 bytes long in sector size, the proposed method is carried over and evaluated for 4096-byte fragments as well and its results is presented in the next section.

6 Experiments and Results

In this section, we assess each parameter during the process of tuning to achieve the best final results. It should be noted that we use a machine with 16 cores, 2.3 GHz each, 64 GB of RAM and 64-bit Windows 7 operating system to run our experiments. To create visual words, the implementation of *mini-batch K-means* algorithm in python's *Anaconda* package [34] is used. To create SVM classifier, the *regularized L2 loss function* is applied in linear kernel experiments using *LIBLINEAR* [35], while *LIBSVM* [36] is used for RBF and Polynomial kernels experiments.

6.1 Building The Dataset

We used 31 file types for our experiments as shown in Table 2. We collected our data from Govdocs1 corpus [37] and also Filetypes1 [38] which is a supplement for Govdocs1. These collections have been created for the purpose of applications in computer forensics education research. To create the dataset, files from Govdocs1 and Filetypes1 are randomly selected and used without replacement, while their extensions are considered as the class labels. TrID [3] was also used to validate the correspondence between file extensions and signatures. This tool is designed to identify the type of file using its signature. After completing file selection and validation steps, each file is disintegrated and broke down into 512 byte fragments and saved as pairs $(frag_i, fileType_j)$. The first fragment of each file is removed to prevent the possibility of file signatures information involvement in identification process. Besides, the last fragment of each file is omitted to ensure an overall fixed block size of 512-bytes long. Finally, for everyone of the 31 file types, from many initially fragmented files, 1000 fragments are randomly selected to set up a dataset of 31000 fragments total. Table 2 shows information of sample files.

Table 2. Data information

Type	Total number of fragments	Size(MB)
txt	59312	23.4
xml	21269	10.4
xls	26307	12.8
wmv	28764	14
tif	13092	6.4
ps	27168	13.2
ppt	48262	23.5
mp4	21460	10.4
mp3	86211	42.1
m4a	28711	14
log	36120	17.6
json	18026	8.81
js	12729	6.26
jpg	22003	10.8
java	13005	6.57
html	18340	9.15
gif	24123	11.8
flv	41393	20.2
doc	33129	16.2
csv	13794	6.83
css	9906	4.93
bz2	20148	9.85
bmp	58640	28.6
avi	27934	13.6
pdf	173325	84.8
docx	37404	18.3
gz	55180	26.9
png	77300	37.8
pptx	50438	24.6
xlsx	26856	13.1
zip	28074	13.7

6.2 Parameter Tuning

Set aside the obvious learning parameters, kernel and the penalty (c) factor in SVM learner, the proposed method effectiveness also depends on the values of a few other parameters. Namely, dimensions of the image (M and N), the asserted neighborhood radius around each pixel (r), and the number of clusters (aka. visual words) (k). As previously mentioned, given that fragments are 512 bytes long, every feasible dimension is surveyed. We bound the search space for neighborhood radius to the set $R = \{1, 2\sqrt{2}, 3\sqrt{2}, 4\sqrt{2}\}$ which covers square shaped sections of different sizes in the pixel plane. To avoid exploding the search space and colluding the impact of each individual parameter, just one factor is varied at a time while the others are kept fixed. To conduct the experiment, following steps are taken:

(1) A specific image dimension ($M \times N$) is fixed for this iteration.
(2) A neighborhood radius r is also chosen for this iteration.
(3) For each pixel p_i of each image $\mathbf{I}$ in the training set, a pixel feature vector $\psi(p_i)$ is created using r value.
(4) Resulting vectors $\Psi_{\mathbf{P}}$ are clustered in order to introduce the visual words.
(5) Assign a visual word to each pixel feature vector $\psi(p_i)$ using the Euclidean distance to clusters' centroids.
(6) Take account of the frequency distribution of 1-gram and 2-gram of visual words $\mathcal{F}^{\texttt{BOVW}_\mathbf{P}}$ in each image $\mathbf{I}$ which is obtained by counting every subsequence of one visual word and two visual words occurrence in each image.
(7) Learn a SVM classifier with linear kernel to classify fragments based on resulted frequency distributions $\mathcal{F}^{\texttt{BOVW}_\mathbf{P}}$.

Table 3 compares the accuracy of fragment type detection for different image dimensions. For evaluating the impact of various dimension configurations, constant values of $r = 1$ and $k = 100$ are considered. As it can be seen the best outcome is obtained for a 1×512 pixel image. Particularly, the best accuracy is achieved by the linear sequencing of bytes. This

could be understood due to the innate nature of a binary/text files. Unlike images, actual adjacent pixels in such files are the sequenced ones and vertical vicinity impacts are almost always accidental. However, the results for other dimension settings are also significantly close to the best. In the next steps, the best value of the dimension parameters ($M \times N$), is used to tune the other parameters. Table 4 shows the

Table 3. Accuracy rate comparison for k=100, c(SVM penalty factor)=0.001, SVM linear kernel and r=1 according to different dimension

dimension	Accuracy	dimension	Accuracy
512×1	74.9	1×512	74.9
256×2	72.03	2×256	72.38
128×4	71.29	4×128	72.13
64×8	71.26	8×64	71.86
32×16	71.48	16×32	71.5

accuracy for different values of radius parameter r. The outcomes are pretty close. According to Table 4, $r = 1$ is the best radius. With $M \times N = 1 \times 512$ as the dimensions and $r = 1$ as the radius, each pixel will have a maximum of 2 neighbors, which means that ψ vectors have 3 values. Another parameter to be

Table 4. Accuracy rate comparison for k = 100 and $M \times N = 1 \times 512$, c(SVM penalty factor)=0. and SVM linear kernel according to different radius

radius	Accuracy
1	74.9
$2\sqrt{2}$	72.19
$3\sqrt{2}$	70.3
$4\sqrt{2}$	69.03

tuned is the number of clusters (k) in the mini-batch K-means clustering algorithm. To this goal, all possible values of k in the interval $[100, 600]$ are tested. As shown in the Figure 3, the best result is accomplished by $k = 400$. For $k = 400$ we repeated the dimension and radius parameter tuning steps and it is confirmed that dimensions $M \times N = 1 \times 512$ and $r = 1$ are optimum values. By considering $k = 400$, 400 and 160000 features are acquired for 1-gram and 2-gram, which indicate the number of repetitions for single and double visual words, respectively. A number of features, with always zero values in the dataset samples were eliminated so that 118424 2-gram features are remained. Therefore, a total of 118824 features forge our feature vector. We used 10-fold cross-validation in our experiments, which means the original sample is randomly partitioned into 10 equal sized subsamples. Of the 10 subsamples, a single subsample is retained as

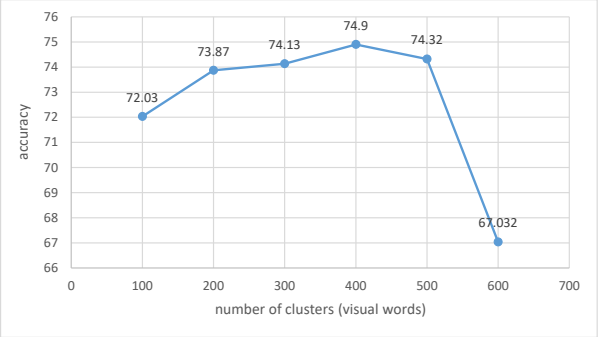

Figure 3. Accuracy rate comparison for $M \times N = 1 \times 512$, c(SVM penalty factor)=0.001, SVM linear kernel and r=1 according to different number of clusters

the testing set, and the remaining 9 subsamples are used as training set. The cross-validation process is then repeated 10 times. The training set samples, are given to a SVM classifier to learn 31 classes on the *one vs. all* strategy. It is expected that a SVM classifier operates more efficiently in the face of sparse structures, which happens to be the case since the N-gram technique usually leads to sparse vectors. The SVM classification method has been tested with different kernels. Figure 4 shows results for each of linear, RBF, sigmoidal and polynomial kernels with their optimum c parameter (gamma is presumed to be $1/\#features$). As it can be seen, liner kernel is superior concerning the accuracy rate. Figure 5 shows a comparison between different values of c parameter for this kernel. It should be noted that the SVM results are reported for SVM's primal form with linear kernel and the SVM's dual form with sigmoid, polynomial and RBF kernels. Table 5 shows the re-

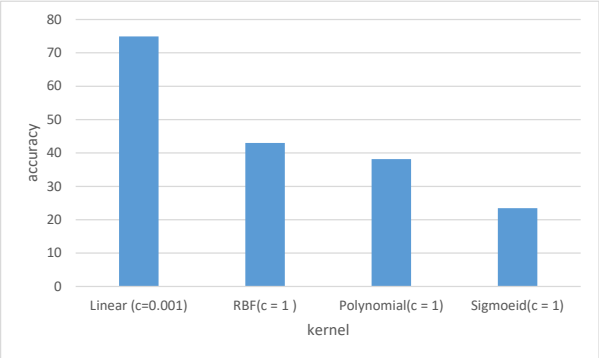

Figure 4. Accuracy rate comparison for $M \times N = 1 \times 512$, $gamma = \frac{1}{\#features}$, r=1 and k=400 according to different svm kernels

sults for examinations concerning the use of either 1-gram or 2-gram, or both. According to Table 5, the

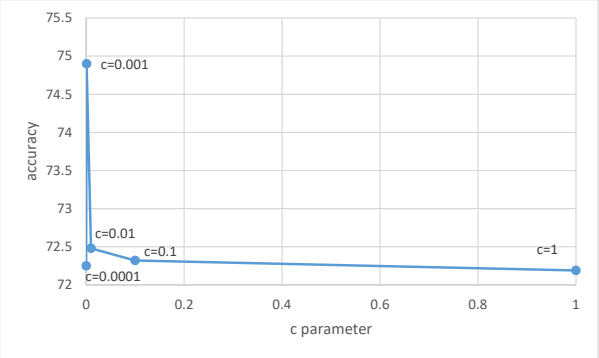

Figure 5. Accuracy rate comparison for $M \times N = 1 \times 512$, r=1 and k=400 according to different values of c for linear kernel

best accuracy indicates the use of 1-gram and 2-gram together.

Table 5. Accuracy rate comparison using 1-gram, 2gram and both for $M \times N = 1 \times 512$, c (SVM penalty factor)=0.001, SVM linear kernel, r=1 and k=400

n-gram	Accuracy
1-gram	69
1-gram	72.3
1-gram + 2-gram	74.9

6.3 Comparison

In order to evaluate the proposed method, it is compared to an earlier study named Sceadan [24] which was reviewed in Section 2 and its implementation is available at [39]. From all the related work reviewed previously, considering the number of file types taken into account, data set availability and remarkable success rate, only Sceadan is conveniently suited to compare our results upon. Sceadan developers collected most of their sample files (66%) from the Govdocs1 dataset [37] and the remaining from other sources such as the Internet or personal files. They segmented the sample files into 512 byte fragments which resulted in 999,147 total fragments of size 512 byte, across 30 file types and 8 data types. To implement our proposed method, using the dataset presented in [24], it is necessary to have raw fragments which the dataset has been built up from. However, none of those extra nuggets are available for now, leaving out only the Sceadan final dataset which because of its innate differences from ours, makes the implementation of our method on Sceadan dataset impossible. In order to make comparison possible, Sceadan method is also tested using this study dataset. Regarding the evaluation criteria in past research, true positive rate, false positive rate and accuracy rate are also examined here. They are calculated according to Equation 15 - Equation 17. tp and tn are the number of positive and negative samples respectively which are classified correctly. fp is the number of negative samples classified as positive. p and n are the number of positive and negative samples respectively.

$$tp\_rate = \frac{tp}{p}, \tag{15}$$

$$fp\_rate = \frac{fp}{n}, \tag{16}$$

$$accuracy = \frac{tp + tn}{p + n}. \tag{17}$$

The comparison of the two methods in Table 6 shows that the proposed method has outperformed Sceadan in terms of the accuracy rate. Also, the proposed method has improved the false positive rate as well as true positive rate of the tif, ps, ppt, m4a, json, js, gif, flv, doc and xlsx file types. For other file types other than log and jpg the proposed method is superior in either false positive rate according to Figure 6 or true positive rate according to Figure 7. Sceadan method has also used 1-gram and 2-gram features to learn file types. Nonetheless, it contemplates the gram concept as merely the byte value of pixel. Considering this is a pretty common tradition among a lot of previous research, Sceadan method innovation is arguably limited to a good choice of parameters for the SVM classifier. However, in this study, visual words or regions of fragments with similar properties are considered as grams. As shown in Figure 6 a number of file types have relatively high false positive rates though. Such high misclassification between some file types can be explained by invoking two general arguments: first, using the same compression algorithms (such as Deflate) in various file types produces the same patterns that classifier could not tell them apart. Second, whenever multi-type files with several embedded data types get involved, the classifier is incapable of determining the correct baseline file type.

6.4 Evaluating the Extended Proposed Method to Reduce False Positive Rates

To evaluate this extension, a validation set of 3100 samples (100 samples per each file type) has been created and then is given to the learned classifier. Table 7 presents results, assuming $\beta = 1$ for both our method and Sceadan. 16% of validation set samples were assigned to *unknown* labeled class where, some of them could be classified correctly if the probability threshold was not applied. This explains the decrement of accuracy rate in both Sceadan and proposed

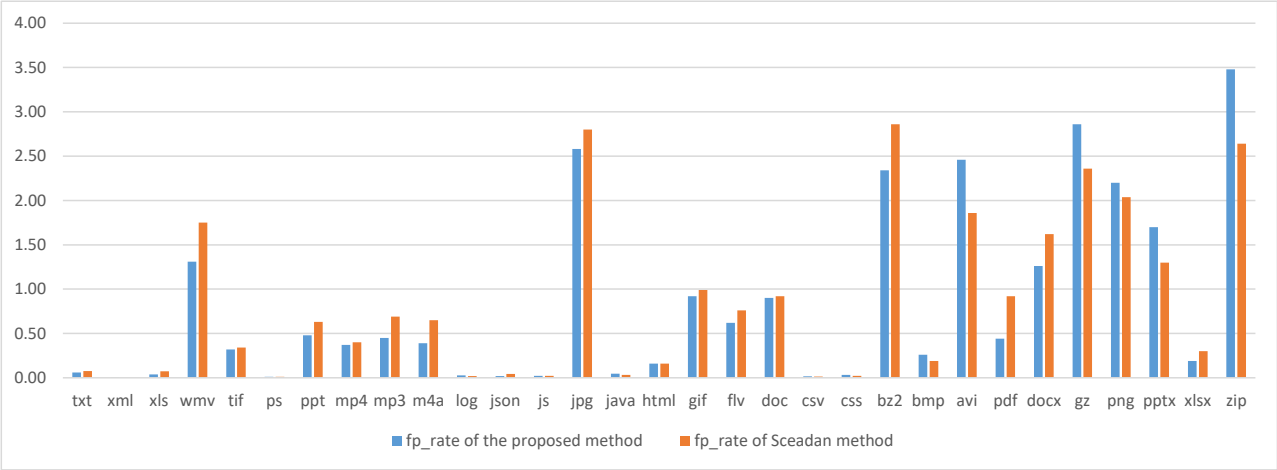

Figure 6. Comparison of fp_rate between the proposed method and Sceadan method

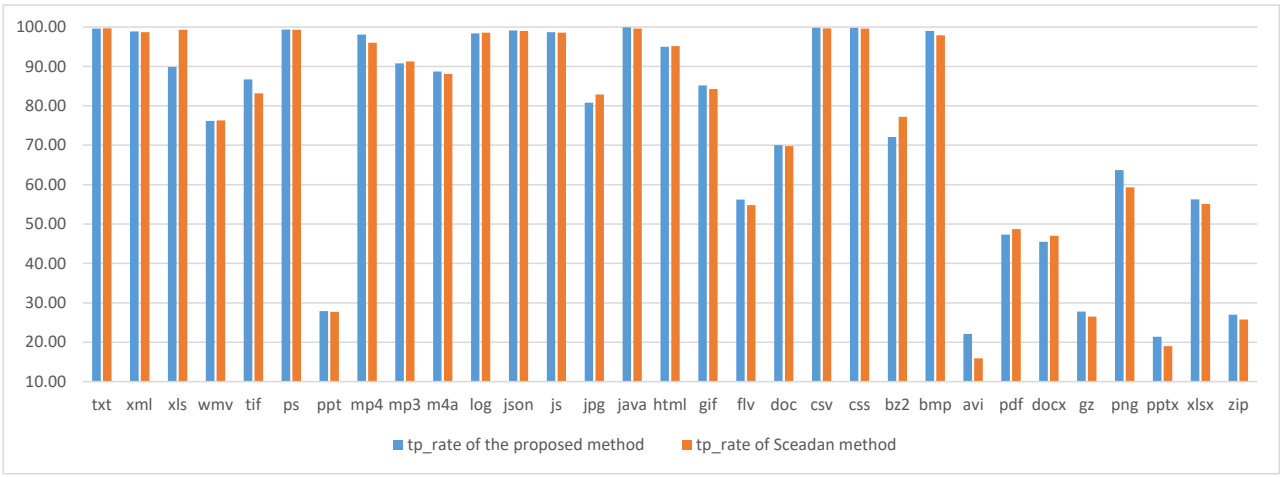

Figure 7. Comparison of tp_rate between the proposed method and Sceadan method

method. As the results demonstrates, considering an *unknown* labeled class has dominated the extended proposed method over Sceadan in terms of accuracy. The extended proposed method is better in false positive rate as well as true positive rate than Sceadan for the txt, xls, tif, mp4, mp3, flv, pdf, png, xlsx and zip file types. Also, for 7 file types it is superior in false positive rate according to Figure 8, for 6 file types in true positive rate according to Figure 9 and for 4 file types has equal results with Sceadan method.

6.5 Evaluating the Extended Proposed Method for 4096 Byte Fragments

The proposed method is carried over 4096-byte fragments as well. Table 8 indicates the results. The accuracy of 87.3% indicates that the proposed method outshines its own efficiency for even larger fragments. For 4096-byte fragments, the difference between the accuracy of the proposed method and Sceadan method is improved from 0.6% to 3.72%. Table 8 depicts the false positive and the true positive rates for each file type. By comparing Table 6 and Table 8, it could be summarized that the larger the file size, the easier it would get to identify common patterns between fragments of a file type.

7 Conclusion

In this study we explored a new approach to file fragment classification using the file types texture analysis. The main innovation of this study is to utilize a texture-derived dictionary of visual words with *BOVW* in order to train an abundance measure for the likelihood of fragment-file-type association. In this technique, the occurrences rate of every visual word is accounted as n-grams for all fragments, which yields a bunch of distribution vectors of vi-

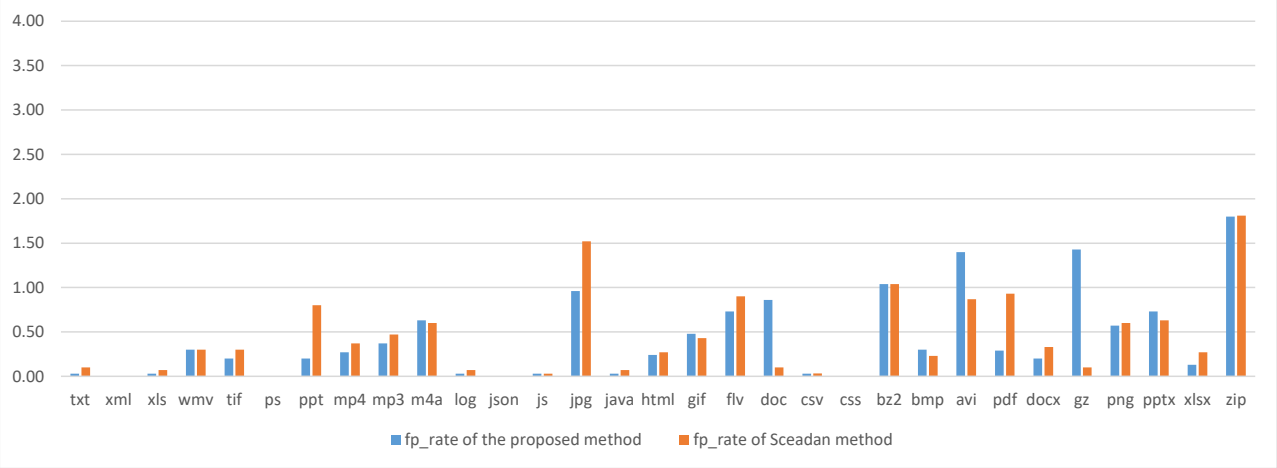

Figure 8. Comparison of fp_rate between the proposed method and Sceadan method by considering a class labeled *unknown*

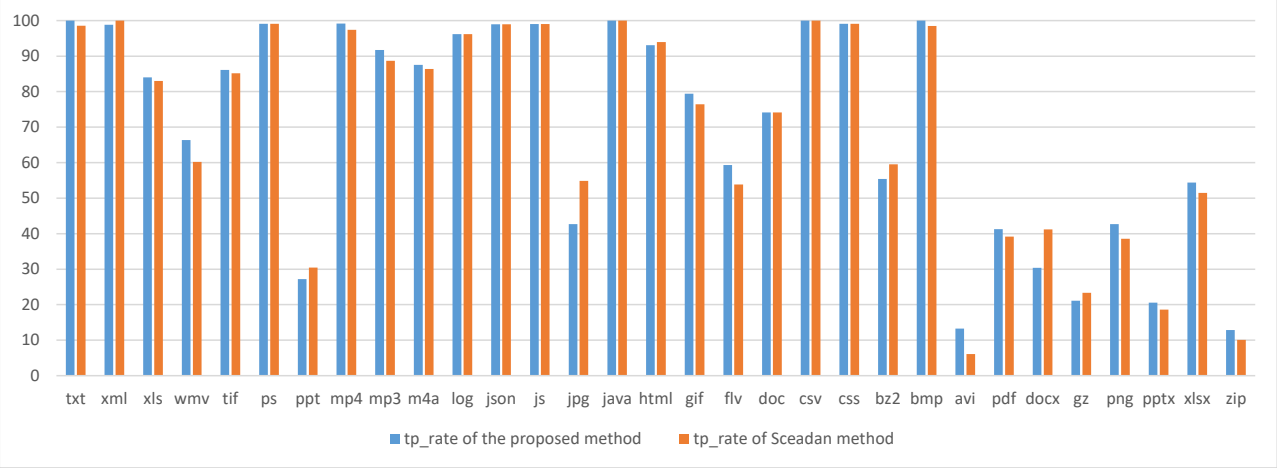

Figure 9. Comparison of tp_rate between the proposed method and Sceadan method by considering a class labeled *unknown*

sual words for that texture. Classifying these vectors helps with file type discernment. For 512-byte long fragments, the proposed method records a 74.9% average accuracy rate which significantly trumps the random labeling method with an accuracy rate of 3.2% (1/31). Furthermore, about 67.5% of inspected file types have an average accuracy rate of at least 70%. Experiments iterated for 4KB long fragments, shows that the proposed method enjoys an average boost of at least 10% in accuracy rates over all file types so that 77.4% of them have an average accuracy rate of at least 70%. This exhibits an efficiency advantage, leaning toward modern hard drives which 4KB sectors are considered a de-facto specification. Overall, our method's type identification rate has improved on previous works. While most of related studies with better benchmarks have used intact whole files [2][21], fragments with meta and header information [2][7, 8][10, 11, 13, 14][22], or smaller set of file types [2][4][6–11, 13–16][21, 22], our approach enhances on all these three fronts.

8 Future Work

In this research, texture analysis is considered to segregate file fragments. Many studies have been conducted to extract textural properties in image processing. We can mention the use of complex networks theory among them. As future work, we intend to model file fragments using complex networks and extracting their structural elements. It can provide much information about the content of fragments, which can be utilized in order to identify file fragments type.

Table 6. Comparison of the proposed method and Sceadan method

file type	proposed method		Sceadan method[24]	
	tp	fp	tp	fp
txt	99.60	**0.06**	**99.7**	0.077
xml	**98.90**	0.003	98.7	**0**
xls	89.90	**0.037**	**99.3**	0.073
wmv	76.20	**1.31**	**76.3**	1.75
tif	**86.70**	**0.32**	83.2	0.34
ps	**99.4**	0.01	99.3	0.01
ppt	**27.90**	**0.48**	27.7	0.63
mp4	**98.10**	**0.37**	96	0.40
mp3	90.80	**0.45**	**91.3**	0.69
m4a	**88.70**	**0.39**	88.1	0.65
log	98.40	0.027	**98.6**	**0.02**
json	**99.10**	**0.02**	99	0.043
js	**98.70**	0.023	98.6	0.023
jpg	80.80	**2.58**	**82.9**	2.80
java	**99.90**	0.047	99.6	**0.033**
html	95	0.16	**95.2**	0.16
gif	**85.20**	**0.92**	84.3	0.99
flv	**56.20**	**0.62**	54.8	0.76
doc	**70**	**0.90**	69.8	0.92
csv	**99.80**	0.017	99.7	**0.013**
css	**99.80**	0.033	99.6	**0.023**
bz2	72.10	**2.34**	**77.2**	2.86
bmp	**99**	0.26	97.9	**0.19**
avi	**22.10**	2.46	15.9	**1.86**
pdf	47.30	**0.44**	**48.7**	0.92
docx	45.50	**1.26**	**47**	1.62
gz	**27.80**	2.86	26.5	**2.36**
png	**63.70**	2.20	59.3	**2.037**
pptx	**21.40**	1.70	19	**1.30**
xlsx	**56.30**	**0.19**	55.1	0.30
zip	**27**	3.48	25.8	**2.64**
accuracy	74.9		74.3	

Table 7. Comparison of the proposed method and Sceadan method by considering a class labeled *unknown*

file type	proposed method		Sceadan method[24]	
	tp	fp	tp	fp
txt	**100**	**0.03**	98.56	0.10
xml	98.82	0	**100**	0
xls	**84**	**0.03**	83	0.07
wmv	**66.33**	0.3	60.20	0.3
tif	**86.14**	**0.2**	85.15	0.30
ps	99.11	0	99.11	0
ppt	27.17	**0.2**	**30.43**	0.80
mp4	**99.14**	**0.27**	97.41	0.37
mp3	**91.75**	**0.37**	88.66	0.47
m4a	**87.5**	0.63	86.37	**0.60**
log	96.19	**0.03**	96.19	0.07
json	98.95	0	98.95	0
js	99.06	0.03	99.06	0.03
jpg	42.69	**0.96**	**54.88**	1.52
java	100	**0.03**	100	0.07
html	93.11	**0.24**	**93.97**	0.27
gif	**79.41**	0.48	76.47	**0.43**
flv	**59.34**	**0.73**	53.85	0.90
doc	74.16	0.86	74.16	**0.1**
csv	100	**0.03**	100	0.033
css	99.12	0	99.12	0
bz2	55.37	1.04	**59.55**	1.04
bmp	**100**	0.3	98.52	**0.23**
avi	**13.27**	1.4	6.12	**0.87**
pdf	**41.24**	**0.29**	39.17	0.93
docx	30.39	**0.2**	**41.18**	0.33
gz	21.11	1.43	**23.33**	**0.1**
png	**42.71**	**0.57**	38.54	0.60
pptx	**20.59**	0.73	18.63	**0.63**
xlsx	**54.37**	**0.13**	51.46	0.27
zip	**12.84**	**1.8**	10.09	1.81
accuracy	70.13		69.7	

Table 8. Comparison of the proposed method and Sceadan method for 4096 byte fragments

file type	proposed method		Sceadan method[24]	
	tp	fp	tp	fp
txt	100	0.03	100	**0.01**
xml	**99.90**	0	99.20	0
xls	**99.60**	**0**	94	0.03
wmv	99.70	**0.10**	97.70	0.937
tif	**97.50**	**0.13**	95.90	0.18
ps	99.90	**0.003**	99.90	0.007
ppt	**34.50**	0.36	26.70	**0.28**
mp4	**99.60**	**0.02**	99	0.057
mp3	**98.60**	**0.01**	97.50	0.17
m4a	**97.70**	**0.0017**	94.70	0.25
log	**99.50**	**0.003**	99.30	0.013
json	100	**0**	100	0.003
js	**99.70**	**0**	99.60	0.003
jpg	**95.50**	**1.45**	93.70	1.90
java	100	**0.003**	100	1.003
html	**99.40**	**0.017**	98.50	0.043
gif	**95.10**	**0.34**	94.60	0.447
flv	**92.50**	**0.03**	91.90	0.347
doc	**75.20**	**0.19**	70.40	0.363
csv	99.80	0.003	99.80	**0**
css	99.90	**0.003**	**100**	0.013
bz2	**99.70**	**0.27**	96	1.053
bmp	**98.90**	0.06	98	**0.047**
avi	**57.40**	2.12	35.90	**1.31**
pdf	**68.70**	**0.11**	65.60	0.30
docx	**64.50**	**0.51**	53.20	0.82
gz	**62.70**	**2.21**	50.40	2.47
png	**93.10**	**1.01**	84.70	1.47
pptx	**43.30**	**0.67**	32.70	1.04
xlsx	**70.40**	**0.017**	60.50	0.13
zip	**65.30**	3.42	61.60	**3.27**
accuracy	87.3		83.58	

References

[1] Like Zhang and G.B. White. An approach to detect executable content for anomaly based network intrusion detection. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing*, pages 1–8, 2007.

[2] Wei-Jen Li, Ke Wang, Salvatore J Stolfo, and Benjamin Herzog. Fileprints: Identifying file types by n-gram analysis. In *Proceedings of the Sixth Annual IEEE Workshop on Information Assurance*, pages 64–71, 2005.

[3] Marco Pontello. Trid–file identifier, 2013. http://mark0.net/soft-trid-e.html.

[4] P.P. Pullaperuma and A.T. Dharmaratne. Taxonomy of file fragments using gray-level co-occurrence matrices. In *Proceedings of 2013 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–7, 2013.

[5] Leonardo FS Scabini, Wesley N Gonçalves, and Amaury A Castro. Texture analysis by bag-of-visual-words of complex networks. In *Proceedings of Iberoamerican Congress on Pattern Recognition*, pages 485–492, 2015.

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

[7] Irfan Ahmed, Kyung-suk Lhee, Hyunjung Shin, and ManPyo Hong. On improving the accuracy and performance of content-based file type identification. In *Proceedings of the 14th Australasian Conference on Information Security and Privacy*, pages 44–59, 2009.

[8] Ahmed Kattan, Edgar Galván-López, Riccardo Poli, and Michael ONeill. Gp-fileprints: file types detection using genetic programming. In *Proceedings of the European Conference on Genetic Programming*, pages 134–145, 2010.

[9] Philip Penrose, Richard Macfarlane, and William J Buchanan. Approaches to the classification of high entropy file fragments. *Digital Investigation*, 10(4):372–384, 2013.

[10] Martin Karresand and Nahid Shahmehri. Oscarfile type identification of binary data in disk clusters and ram pages. In *Proceedings of the IFIP International Information Security Conference*, pages 413–424, 2006.

[11] Martin Karresand and Nahid Shahmehri. File type identification of data fragments by their binary structure. In *Proceedings of the IEEE Information Assurance Workshop*, pages 140–147, 2006.

[12] Konstantinos Karampidis, Ergina Kavallieratou, and Giorgos Papadourakis. Comparison of classification algorithms for file type detection a digital forensics perspective. *Polytech. Open Libr. Int. Bull. Inf. Technol. Sci.*, 56:15–20, 2017.

[13] Cor J Veenman. Statistical disk cluster classification for file carving. In *Proceedings of the Third International Symposium on Information Assurance and Security*, pages 393–398, 2007.

[14] Robert F Erbacher and John Mulholland. Identification and localization of data types within large-scale file systems. In *Proceedings of the Second International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 55–70, 2007.

[15] Sarah J Moody and Robert F Erbacher. Sádi-statistical analysis for data type identification. In *Proceedings of the Third International Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 41–54, 2008.

[16] William C Calhoun and Drue Coles. Predicting the types of file fragments. *digital investigation*, 5:S14–S20, 2008.

[17] Stefan Axelsson. The normalised compression distance as a file fragment classifier. *digital investigation*, 7:S24–S31, 2010.

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

[19] Gregory Conti, Sergey Bratus, Anna Shubina, Benjamin Sangster, Roy Ragsdale, Matthew Supan, Andrew Lichtenberg, and Robert Perez-Alemany. Automated mapping of large binary objects using primitive fragment type classification. *digital investigation*, 7:S3–S12, 2010.

[20] Ning Zheng, Jinlong Wang, Ting Wu, and Ming Xu. A fragment classification method depending on data type. In *Proceedings of 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*, pages 1948–1953, 2015.

[21] Ding Cao, Junyong Luo, Meijuan Yin, and Huijie Yang. Feature selection based file type identification algorithm. In *Proceedings of 2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, volume 3, pages 58–62, 2010.

[22] Siddharth Gopal, Yiming Yang, Konstantin Salomatin, and Jaime Carbonell. Statistical learning for file-type identification. In *Proceedings of the 10th International Conference on Machine Learning and Applications(ICMLA)*, volume 1, pages 68–73, 2011.

[23] Simran Fitzgerald, George Mathews, Colin Morris, and Oles Zhulyn. Using nlp techniques for file fragment classification. *Digital Investigation*, 9:S44–S49, 2012.

[24] Nicole L Beebe, Laurence A Maddox, Lishu Liu, and Minghe Sun. Sceadan: using concatenated n-gram vectors for improved file and data type classification. *IEEE Transactions on Information Forensics and Security*, 8(9):1519–1530, 2013.

[25] Dinil Mon Divakaran, Yung Siang Liau, and Vrizlynn LL Thing. Accurate in-network file-type classification. In *SG-CRC*, pages 139–146, 2016.

[26] Kristijan Vulinović, Lucija Ivković, Juraj Petrović, Kristian Skračić, and Predrag Pale. Neural networks for file fragment classification. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1194–1198, 2019.

[27] Manish Bhatt, Avdesh Mishra, Md Wasi Ul Kabir, SE Blake-Gatto, Rishav Rajendra, Md Tamjidul Hoque, and Irfan Ahmed. Hierarchy-based file fragment classification. *Machine Learning and Knowledge Extraction*, 2(3):216–232, 2020.

[28] K Skračić, F Rukavina, K Miličić, J Petrović, and P Pale. File fragment classification with focus on ole and ooxml classes. In *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, pages 1250–1253. IEEE.

[29] Gregory Conti, Sergey Bratus, Anna Shubina, Andrew Lichtenberg, Roy Ragsdale, Robert Perez-Alemany, Benjamin Sangster, and Matthew Supan. A visual study of primitive binary fragment types. *White Paper, Black Hat USA*, 2010.

[30] André Ricardo Backes, Dalcimar Casanova, and Odemir Martinez Bruno. Texture analysis and classification: A complex network-based approach. *Information Sciences*, 219:168–180, 2013.

[31] David Sculley. Web-scale k-means clustering. In *Proceedings of the 19th international conference on World wide web*, pages 1177–1178, 2010.

[32] N Vapnik Vladimir and V Vapnik. Statistical learning theory. *Xu JH and Zhang XG. translation. Beijing: Publishing House of Electronics Industry, 2O04*, 1998.

[33] Alberto Cano. A survey on graphic processing unit computing for large-scale data mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(1):e1232, 2018.

[34] Continuum Analytics. Anaconda python distribution, 2015. https://www.anaconda.com/.

[35] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of machine learning research*, 9:1871–1874, 2008.

[36] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2:27, 2011.

[37] Simson Garfinkel, Paul Farrell, Vassil Roussev, and George Dinolt. Bringing science to digital forensics with standardized forensic corpora. *digital investigation*, 6:S2–S11, 2009.

[38] Nicole L Beebe. Utsa filetypes1 dataset, 2016. http://digitalcorpora.org/corp/files/filetypes1/.

[39] N. L Beebe. Sceadan. https://github.com/UTSA-cyber/sceadan.

Mina Erfan received the M.Sc. degree in information technology engineering from Tarbiat Modares University, in 2017. Her main research interests include digital forensics, data science and machine learning.

Saeed Jalili received the Ph.D. degree from Bradford University in 1991 and the M.Sc. degree in computer science from Sharif University of Technology in 1985. He is currently an associate professor at Tarbiate Modares University and his main research interests are machine learning, computer network security, software protection, software runtime verification and self-adaptive systems.