Light-Weight File Fragments Classification using Depthwise Separable Convolutions

Kunwar Muhammed Saaim[1], Muhamad Felemban[2,3], Saleh Alsaleh[2,3], and Ahmad Almulhem[2,3]

[1] Aligrah Muslim University, Aligarh, Uttar Pradesh, India
kmsaaim@zhcet.ac.in
[2] Computer Engineering Department, KFUPM, Dhahran, Saudi Arabia
[3] Interdiscipinary Research Center for Intelligent Secure Systems, KFUPM
{mfelemban,salehs,ahmadsm}@kfupm.edu.sa

Abstract. In digital forensics, classification of file fragments is an important step to complete the file carving process. There exist several approaches to identify the type of file fragments without relying on meta-data. Examples of such approaches are using features like header/footer and N-gram to identify the fragment type. Recently, deep learning models have been successfully used to build classification models to achieve this task. In this paper, we propose a light-weight file fragment classification using depthwise separable convolutional neural network model. We show that our proposed model does not only yield faster inference time, but also provide higher accuracy as compared to the state-of-art convolutional neural network based models. In particular, our model achieves an accuracy of 78.45% on the FFT-75 dataset with 100K parameters and 167M FLOPs, which is 24x faster and 4-5x smaller than the state-of-the-art classifier in the literature.

Keywords: Digital Forensics, File Carving, File Fragments Classification, Deep Learning, Depthwise Separable Convolution

1 Introduction

Digital forensics is the science of collecting digital evidence to investigate cybercrimes and cyberattack scenarios. The process of digital forensics include preservation, identification, and extraction of data and computer files. There are several digital forensics tools and techniques that facilitate this process [27, 32]. Often, attackers attempt to wipe out any evidence that incriminates them, for example, by formatting the hard disk [6]. In such scenarios, traditional recovery methods based on file system meta-data are deemed to be ineffective.

To overcome this challenge, digital investigators use file carving to reconstruct files based on their contents [15, 24, 26]. In other words, file carving process recovers files from blocks of binary data without using any information available in the file system structure. Researchers have attempted to reconstruct files entirely or partially using a variety of techniques including header/footer matching [33],

probabilistic measures [3], and n-gram analysis [41]. With the absence of metadata, it is challenging to identify the type of a carved file. The problem is even more challenging when file carving is carried out on fragmented files. File fragmentation is used when there is not enough contiguous space to write a file on the hard disk. In general, two tasks can be recognized in fragmented file carving: selecting a candidate sequence of file blocks and classifying the type of the fragment [26].

There exist several approaches for file fragments classification in the literature [1, 4, 5, 8, 12, 16, 25, 29, 33, 39, 41–43]. For example, *Sceadan* tool determines the type of bulk data based file content [5]. Recently, deep learning has been used to identify the type of file fragments. For example, Gray-scale [8] and FiFTy [29] tools illustrate how Convolutional Neural Networks (CNNs) can be efficiently used for file fragments classification. Although CNN is the most used deep learning model for classification, we argue that existing solutions can be further improved in terms of performance and accuracy by modifying the architecture of the neural network.

To illustrate, the convolution layer in CNNs can be either 2D convolution layer [8] or 1D convolution layer [29]. One drawback of CNNs is the exponential increase in number of parameters as the number of layers increase, i.e., the model depth increases. Such increase in the number of parameters increases the complexity of the model and, consequently, increases the training and inference time. On the other hand, deep CNN models provides better accuracy. Therefore, it is utterly important to find a balance between the speed and accuracy in terms of the number of layers. To over come this challenge, researchers proposed a computationally cheaper model called depthwise separable convolution [21, 36, 37]. Depthwise separable convolution is cheap because it dramatically reduces the required number of parameters in the model.

In this paper, we propose a light-weight file fragment classification based on depthwise separable convolutional neural network. The rationale behind designing a light-weight classification model is to provide the ability to process huge volume of data without requiring significant computational resources. Therefore, the process of file fragment classification can be feasible without specialized hardware, e.g., GPUs and TPUs. To evaluate the performance of our model, we compared our results with FiFTy [29] and a baseline Recurrent Neural Networks (RNN) [19]. We have trained 12 different models using various file types and different fragment sizes using FFT-75 dataset [28]. The results show that our model achieves an accuracy similar to FiFTy using 6.3x less floating-point operations (FLOPs) on 4096 bytes fragment and 85x less FLOPs on 512 bytes fragment as compared to FiFTy. Furthermore, our model achieves 78.45% accuracy with 100K parameters and 167M FLOPs, while FiFTy achieves 77.04% accuracy with 449K parameters and 1047M FLOPs. Finally, our model outperforms the previous best classifier by being 24x faster on GPU than FiFTy and 650x faster than baseline RNN.

The rest of this paper is organized as follows. In Section 2, we provide necessary background for this work. In Section 3, we discuss the state-of-the-art of

file fragment classification. In Section 4, we present our model. In Section 5, we discuss the experiments and evaluation results. Finally, conclusions and future work are presented in Section 6.

2 Preliminaries

2.1 Convolution Neural Network

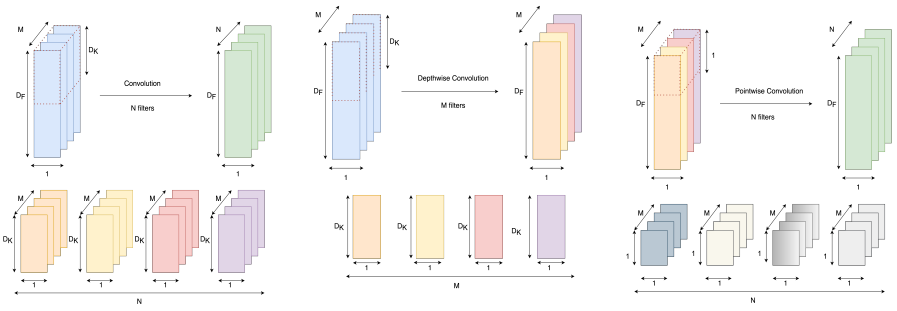
CNN is a type of artificial neural network that is widely used for image and object recognition and classification. CNN contains one or many convolution layers that can extract features from the inputs and predict an output based on the extracted features [13]. A convolution layer can be considered as a 2D matrix (kernel) that performs the convolution operation. Mathematically, convolution operation is the dot product between the kernel and the input of the layer. Different kernels can be used to give importance to the input data points whose output is stacked in depth. The kernel filters are smaller than the layer's input so that kernel weights can be shared across input dimensions.

In the context of file fragment classification, the input to CNN consists of a vector of 4096 or 512 dimensions, which corresponds to 4KB or 512 bytes fragments, respectively. Then, the input is fed to an embedding layer that transforms the input vector into a continuous (4096,32) or (512,32) dimensional vector, based on the size of the fragments. The embedding layer can be thought of as a preprocessing step to prepare input vectors for the subsequent CNN layers. The output of the embedding layer is fed into a convolution layer with kernels of any size. Several convolution layers can be stacked [29]. As a result of stacking standard convolution layers when developing a deep neural network (DNN), the model can have a large number of parameters.

2.2 Depthwise Separable Convolution

The separable convolution model was first introduced by Sifre and Mallat [34]. Since then, several models have been proposed based on the idea of separable convolution, including Xception-net [9] and Mobilenets [21]. The automatic feature extraction behaviour of standard convolution can be achieved by smaller and faster depthwise separable convolution. Depthwise separable convolution is a form of factorized convolutions in which the normal convolution kernel is split into two parts: one depthwise convolution and a 1×1 convolution, called pointwise convolution. The splitting of convolution into two layers has a drastic effect on reducing the number of parameters and computation time required for training and inference. The depthwise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs of the depthwise convolution [21].

Figure 1a depicts a standard one-dimensional convolution operation that takes $D_{\mathrm{F}} \times 1 \times M$ feature maps as input and produces $D_{\mathrm{F}} \times 1 \times N$ feature maps, where D_F is length of the input feature, M is the number of input channels,



(a) Standard Convolution (b) Depthwise Convolution (c) Pointwise Convolution

Fig. 1: Standard convolution in (a) is factorized into two layers: Depthwise Convolution in (b) and pointwise in (c).

and N is the number of output channels. In convolutional network, the number of parameters with kernel size K is $D_{\mathrm{K}} \times 1 \times N \times M$. Subsequently, the total computation cost is

$$D_{\mathrm{F}} \cdot N \cdot M \cdot D_{\mathrm{K}} \tag{1}$$

Unlike CNNs, depthwise separable convolution factorizes the convolution layer into two layers: depthwise convolution depicted in figure 1b and pointwise convolution shown in figure 1c. The depthwise convolution layer with one filter per input channel has $D_{\mathrm{K}} \times 1 \times M$ parameters with K sized kernel and M number of channels. Therefore, depthwise convolution has a computation cost of:

$$D_{\mathrm{F}} \cdot M \cdot D_{\mathrm{K}} \tag{2}$$

On the other hand, pointwise convolution uses a linear combination of filtered feature maps with the help of 1×1 convolution. The cost of 1×1 convolution is the same as standard convolution having a kernel size of 1, i.e., $D_{\mathrm{F}} \times N \times M$. Consequently, the total computation cost of depthwise separable convolution is:

$$D_F \cdot M \cdot D_K + D_F \cdot N \cdot M \tag{3}$$

As a result, the total reduction in the computation cost when using depthwise separable convolution is:

$$\frac{D_F \cdot M \cdot D_K + D_F \cdot N \cdot M}{D_F \cdot N \cdot M \cdot D_K} \tag{4}$$

$$= \frac{1}{N} + \frac{1}{D_K} \tag{5}$$

The value of N ranges from 32 to 1024 and $\mathrm{D_K}$ for one-dimensional convolution can be between 9 and 27. Therefore, the reduction in computation cost is between 85% and 95%.

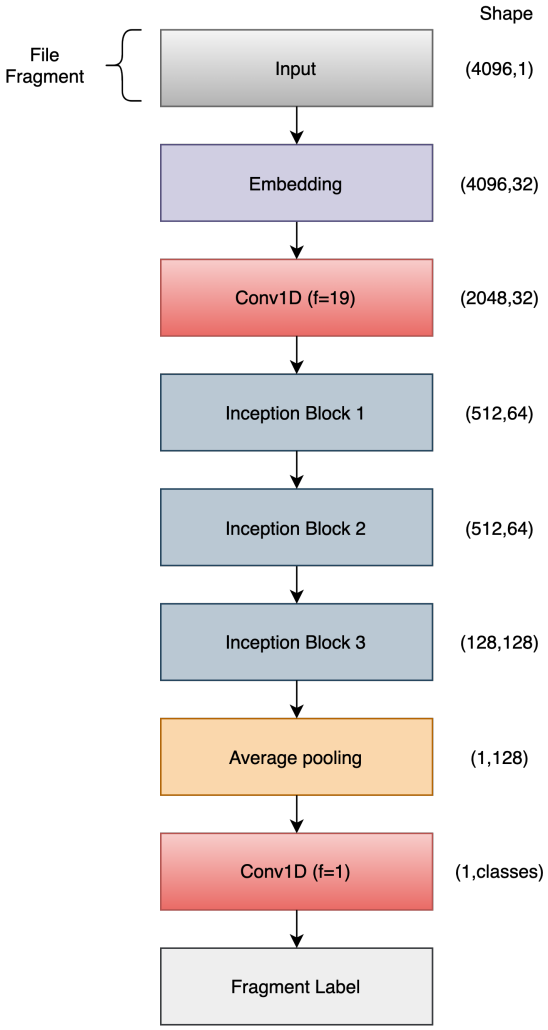


Fig. 2: Network Architecture

3 Depthwise Separable Convolutional Model for File Fragments Classification

In this section, we describe our depthwise separable convolutional architecture for file fragment classification. The objective is to reduce the inference time of file fragment classification while maintaining high accuracy. Figure 2 shows the overall architecture of the model. It consists of an embedding layer, a standard convolution block, and depthwise separable inception blocks. A file fragment byte ranging from 0 to 255 is transformed into a dense continuous vector of 32

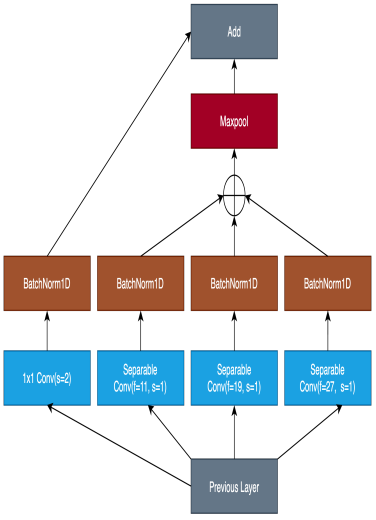


Fig. 3: Depthwise Separable Convolution Inception Block

dimensions by a learnable embedding layer. In order to extract essential features automatically for classification, the dense vector representation is passed through a standard convolution block and multiple depthwise separable inception blocks. To get the final features for classification, the output of the last depthwise separable convolution block is averaged along the spatial dimension. The class probabilities are determined from feature vectors using 1×1 convolution followed by *softmax* activation function [11]. A Hardswish function [20] is used to activate all the blocks in the network.

$$Hardswish(x) = \begin{cases} 0 & if x \leq -3, \\ x & if x \geq +3, \\ \frac{x \cdot (x+3)}{6} & otherwise \end{cases} \tag{6}$$

In general, the embedding layer maps every file fragment byte value to a 32-dimensional continuous vector learned at training time. It is used to compress the input feature dimension into a smaller and more realistic space. Each single byte value would be represented by a 256-dimensional sparse vector in the absence of the embedding layer. With multiple layers of standard and depthwise separable convolutions, features are automatically derived by non-linearly transforming inputs.

3.1 Inception Block

Figure 3 depicts the architecture of the inception block. The previous layer's output is passed to three parallel depthwise separable convolutions and one

Grouping	Files
Archive	apk, jar, msi, dmg, 7z, bz2, deb, gz, pkg, rar, rpm, xz, zip
Audio	aiff, flac, m4a, mp3, ogg, wav, wma
Bitmap	jpg, tiff, heic, bmp, gif, png
Executable	exe, mach-o, elf, dll
Human-readable	md, rtf, txt, tex, json, html, xml, log, csv
Office	doc, docx, key, ppt, pptx, xls, xlsx
Published	djvu, epub, mobi, pdf
Raw	arw, cr2, dng, gpr, nef, nrw, orf, pef, raf, rw2, 3fr
Vector	ai, eps, psd
Video	mov, mp4, 3gp, avi, mkv, ogv, webm
Miscellaneous	pcap, ttf, dwg, sqlite

Table 1: Grouping of different file types

1×1 convolution. Typically, the kernel size can have any value. However, we use kernels of sizes of 11, 19, and 27, with strides of 1, 1, 1, and 4, respectively. Kernel sizes were chosen because they performed well in the previous state-of-the-art network [29] and were not large enough to significantly increase that model parameters. A batch normalization layer follows the convolution layers. All the depthwise separable convolution layer output are added and max-pooled with a pool size of 4. The 1×1 convolution branch is added to the max-pooled output. If the input and output have same number of channel, then the 1x1 convolution layer is replaced by the direct addition of the previous layer output to the max-pooled output as a skip connection.

4 Performance Evaluation

In this section, we present the results of our experimentation. We first describe the dataset used in our experimentation. Then, we give a brief description of a baseline model based on RNN. Finally, we provide an in-depth comparison of our proposed architecture with the baseline model, and FiFTy [29].

4.1 Dataset

To evaluate the performance of our model, we used the dataset provided by Mittal et al. in [28], which contains a balanced number of files per class. Other datasets, e.g., [14], are highly imbalanced with 20 files-types comprising 99.3% of the dataset and remaining 0.7% belonging to 43 file types. The dataset used is composed of 75 types of files that are organized into 6 different scenarios. The Scenarios are described as follows in [29]:

Model	Neural Network	Block Size	# Params	Accuracy	Speed[ms/block]†	Speed [min/GB]†
Our model	Depthwise Separable CNN	4096	**103,083**	**78.45**	**2.65**	**0.055**
		512	**103,083**	**65.89**	**2.78**	**0.382**
FiFTy	1-D CNN	4096	449,867	77.04	38.189	1.366
		512	289,995	65.66	38.67	3.052
Baseline RNN	LSTM	4096	717,643	70.51	268.58	36.375
		512	379,851	**67.5**	126.54	33.431

† Computed on Nvidia Titan X

Table 2: Comparison of results of three models on all 75 file types

1. (All; 75 classes): All file types are separate classes; this is the most generic case and can be aggregated into more specialized use-cases.
2. (Use-specific; 11 classes): File types are grouped into 11 classes according to their use; this information may be useful for elaborate, hierarchical classification or for determining the primary use of an unknown device.
3. (Media Carver - Photos & Videos; 25 classes): Every file type tagged as a bitmap (6), RAW photo (11) or video (7) is considered as a separate class; all remaining types are grouped into one other class.
4. (Coarse Photo Carver; 5 classes): Separate classes for different photographic types: JPEG, 11 RAW images, 7 videos, 5 remaining bitmaps are grouped into one separate class per category; all remaining types are grouped into one other class.
5. (Specialized JPEG Carver; 2 classes): JPEG is a separate class, and the remaining 74 file types are grouped into one other class; scenario intended for analyzing disk images from generic devices.
6. (Camera-Specialized JPEG Carver; 2 classes): JPEG is a separate class, and the remaining photographic/video types (11 RAW images, 3GP, MOV, MKV, TIFF and HEIC) are grouped into one other class; scenario intended for analyzing SD cards from digital cameras.

4.2 Baseline Model

As a baseline model for performance comparison, we implemented a Long Short Term Memory (LSTM) based model. Prior research using LSTMs to classify file fragments had been published [16]. Nevertheless, the low number of learnable parameters hindered their effectiveness in classifying 75 types of files. Further, the model fed raw bytes directly to the LSTM layers in a one-hot representation. We implemented LSTM models with an embedding layer. The model specifications are as follows. File fragment byte sequences were fed into a 32-dimensional embedding layer, followed by bidirectional and unidirectional LSTM layers. The LSTM layer has 128 neurons in each of its two layers for 512-byte fragments and 128 and 256 neurons in its bidirectional and unidirectional layers for 4 KB fragments, respectively. A softmax dense layer produced the output labels.

4.3 Experimental setup

We used automated hyper-parameter tuning for learning rate, optimizer choice, and activation function using TPE [7] implemented through Optuna [2]. We

Scenario	Fragment Size	#param (Ours)	#param (FiFTy)	Inf. Time (ours) (CPU)[ms/block]	Inf. Time (FiFTy) (CPU)[ms/block]	Inf. Time (ours) (GPU)[ms/block]	Inf. Time (FiFTy) (GPU)[ms/block]
1	4096	**103,083**	449,867	**56.502**	121.476	**2.656**	38.189
	512	**103,083**	289,995	**35.691**	75.551	**2.782**	38.673
2	4096	**94,827**	597,259	**50.067**	92.324	**2.720**	29.826
	512	**94,827**	269,323	**38.888**	89.344	**2.795**	29.809
3	4096	**96,633**	453,529	**48.24**	102.808	**2.646**	37.286
	512	**96,633**	690,073	**34.654**	99.710	**2.903**	35.361
4	4096	**94,053**	684,485	**49.341**	100.117	**2.775**	40.176
	512	**94,053**	474,885	**34.121**	79.346	**2.751**	35.965
5	4096	**93,666**	138,386	**49.124**	99.855	**2.878**	39.262
	512	**93,666**	336,770	**35.236**	79.831	**2.725**	42.413
6	4096	**93,666**	666,242	**49.35**	98.536	**2.689**	34.931
	512	**93,666**	242,114	**34.403**	89.104	**2.807**	38.982

Table 3: Comparison between fifty and our model in terms of inference time and model Parameters

found out that Adam optimizer [23] and Hardswish [20] are the best optimizer and activation function, respectively. We did not perform tuning of convolution kernel size; we used the kernel sizes from previous work [29] that proved to be effective. The kernel sizes for different branches of the inception block were taken as 11,19, and 27.

The accuracy for different networks was calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

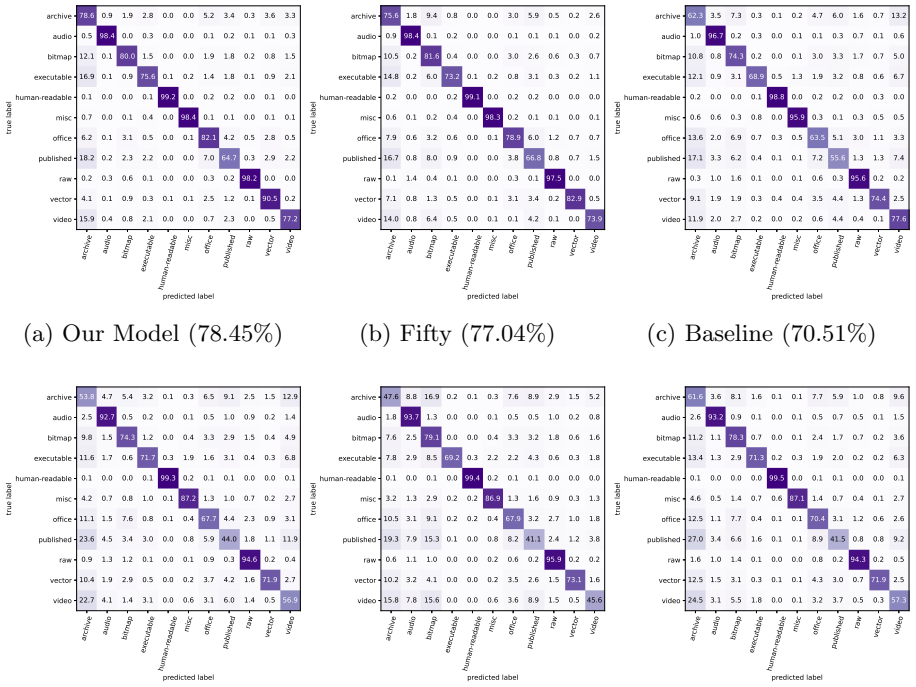
where TP is True Positive, TN is True Negative, FP is False Positive, and FN is False Negative.

To achieve higher accuracy from smaller models, we pretrained our network on corresponding fragment size dataset. We tried to leverage the performance of transfer learning in gaining higher accuracy [31]. In particular, to develop a 4096-byte fragment, the model was pretrained on 512-byte fragment dataset and vice-versa. We found that 6-8% accuracy was increased when pretraining was done using 512-byte fragment data for developing the 4096-byte fragment model. However, similar performance was not achieved when pretraining was done 4096-byte fragment data for the 512-byte fragment model.

All of our experiments were run on a machine with dual Intel Xeon CPU E5-2620 CPUs at 2.40 GHz (12 physical cores, 24 logical cores), 192 GB RAM, and a single Nvidia Titan X GPU, running on Ubuntu 20.04 Operating System. Pytorch 1.5.0 was used for neural network design.

4.4 Results

We observe that our models, in general, perform better than FiFTy models in inference time with no significant loss of accuracy. The results are summarised in TABLE 2. Moreover, we observe our model is superior to both FiFTy and RNN in terms of accuracy and inference time for scenario 1 (75 file types). In other words, when run on 1 GB data of 4096-byte fragments, FiFTy and RNN model are 24x and 650x slower than our proposed model, respectively. Using



(a) Our Model (78.45%) (b) Fifty (77.04%) (c) Baseline (70.51%)

(d) Our Model (65.89%) (e) Fifty (65.66%) (f) Baseline (67.5%)

Fig. 4: Confusion matrices for Our model, FiFTy [29] and Baseline RNN model on 4 KB fragments in (a)-(c) and 512 bytes in (d)-(f). Due to large number of classes(75), classes belonging to same super class were clustered into one.

512-byte fragments, FiFTy is 8x slower, and RNNs is 86x slower than our model when run on 1 GB of data. RNN models perform well with 512-byte fragments due to their ability to handle short sequences. In contrast, when the fragment size increases to 4096-bytes, the performance deteriorates due to the vanishing gradient problem of RNNs [18]. While RNNs has higher accuracy on the 512-byte fragments, the model is not practical because it takes a long time for inference.

In view of a large number of file types, plotting a confusion matrix of all file types was not feasible, so we plotted the confusion matrix of file types grouped by use-cases in Figure 4. The grouping is shown in Table 1. Most of the misclassification is happening in Archive file group due to other file types embedded in them.

The TABLE 3 summarizes the number of neural network parameters in our proposed model and the FiFTy model. For all six scenarios and both fragment sizes, our models have far fewer parameters than FiFTy. Additionally, the table compares the inference times for each of the six scenarios on both the GPU and CPU. Our model outperforms FiFTy by a wide margin on GPU and CPU inference time for all six scenarios. There is a 50% reduction in time for 4096-

Scenario	Fragment Size	FLOPs(ours)	FLOPs(FiFTy)
1	4096	**167.83**	1047.59
	512	**21.00**	1801.71
2	4096	**167.82**	1327.90
	512	**20.99**	918.06
3	4096	**167.82**	647.78
	512	**20.99**	3579.57
4	4096	**167.81**	2378.51
	512	**20.98**	1576.71
5	4096	**167.81**	488.37
	512	**20.98**	2330.48
6	4096	**167.81**	1126.00
	512	**20.98**	611.30

Table 4: Comparison between fifty and our model for floating point operations (Mega FLOPs)

byte file fragments and a 58% reduction for 512-byte files fragments compared to FiFTy on CPU. For GPU, the time reduction is around 92% for both fragment types.

To provide hardware-independent metrics, we calculated the floating-point operations (FLOPs) in our models and FiFTy. TABLE 4 provides the FLOPs comparison for FiFTy and our models. Comparatively to FiFTy, our models have 81% fewer floating-point operations for 4096-byte fragments and 97% fewer for 512-byte fragments.

Our proposed models and FiFTy's accuracy is compared in TABLE 5. The accuracy of our models is comparable with that of FiFTy, although our models have a faster inference time. Using the main scenario of 75 files, our model achieves a higher accuracy with a significantly shorter inference time. Other scenarios have comparable accuracy to corresponding FiFTy models with a much faster run time. Overall, we achieve far better inference time while maintaining similar accuracy.

4.5 Discussion

Our models lag in the same area as previous work in the literature, e.g., [8,29]. In particular, we observe that high entropy files are difficult to classify because there is no statistical trace that the convolutional kernel can extract. Moreover, many files are container types that have another type of object embedded in them, e.g., *pdf* files that can contain embedded *jpg* images. As a result, classifiers behave erratically. Moreover, our model does not perform well for similar file types with different format, e.g., *ppt*, *pptx*, and *key*. However, other file types, e.g., *doc* and *docx*. are not affected by this as *docx* are zip file containing all the associated *XML* files whereas *doc* is stored as binary. Finally, the feasibility of using our model to classify encrypted file fragments will be investigated in the future.

Scenario	#of files	Fragment Size	Acc (Ours)	Acc (FiFTy)
1	75	4096	**78.45**	77.04
		512	**65.89**	65.66
2	11	4096	85.7	**89.91**
		512	75.84	**78.97**
3	25	4096	93.06	**94.64**
		512	80.79	**87.97**
4	5	4096	**94.17**	94.03
		512	87.14	**90.30**
5	2	4096	**99.28**	99.12
		512	98.94	**99.07**
6	2	4096	**99.59**	99.59
		512	98.76	**99.23**

Table 5: Comparison between fifty and our model in terms of accuracy

5 Related Work

Several techniques have been proposed for file fragments identification and classification. The techniques range from basic methods like using magic numbers, and file headers [10] to more advanced methods using machine learning and deep learning [5,29]. Generally, file carving methods can be classified into 1) statistical methods, 2) machine learning methods, and 3) deep learning methods.

Statistical methods Karresand et al. [22] proposed *Oscar*, for determining the probable file type of binary data fragment. *Oscar* uses Byte Frequency Distribution of 4096-byte-sized fragments of different file types for the construction of vector models based on mean and standard deviation, called centroids. A vector distance between the mean and standard deviation of the segment to be classified and the centroids is calculated; if it is lower than a certain threshold value, the segment is classified as a modeling file. Binary classification was done by combining 49 file types into one class and JPEG into another. JPEG achieved a detection rate of 97.90, which was mainly due to byte-stuffing [40].

Machine learning methods In [25], Li et al. proposed a Support Vector Machine (SVM) model that is trained using the histogram of the data bytes as a feature vector. The SVM is used for binary classification with one class fixed as JPEG and the other class varying as DLL, EXE, PDF, and MP3. Similarly, Fitzgerald et al. [12] proposed an SVM model for the classification of file fragments. The file fragments were presented as a bag of words (bag of bytes) with feature vector consisting of uni-gram and bi-gram counts and other statistical measures such as entropy. Zheng et al. [43] proposed a similar approach of using byte frequency distribution (BFD) or histogram and entropy as a feature vector for SVM.

An prominant tool in file carving is *Sceadan*, which is an open-source tool developed by Beebe et al. [5] used various statistical features such as uni-gram, bi-gram, entropy, mean byte value, longest streak, etc. Ten separate input vectors

were prepared out of the statistical features; the vectors were grouped into four sets: uni-grams, bi-grams, all global features except n-grams, a subset of global features. These sets were fed to SVMs of linear and radial basis function (RBM) for classification into 38 diverse file types. They reported an accuracy of 73.4%. Increasing the number of features(in terms of the type, not number) that are input to the SVM harmed the accuracy. The best accuracy was achieved by concatenation of uni-gram and bi-gram.

Deep learning methods Wang et al. [42] used one layer convolution with multiple kernel sizes. The raw byte values were converted to their corresponding binary representation and fed to the embedding layer to convert the binary value to a continuous dense vector. They studied 20 file types from the *GovDocs1* dataset. Chen et al. [8] took a different approach and converted the 4096-byte file fragment to a 64x63 gray-scale image. Their intuition was that data fragments from different files would have different texture features, reflected in the gray-scale image. The gray-scale images were fed to a deep CNN network like VGG [35] with many convolutions and max-pool layers followed by dense classification. Hiester [16] compared feed-forward, recurrent and convolutional neural networks with input fragments in the form of binary representation (fragment bits). The studied file types were JPEG, GIF, XML and CSV. On these easily distinguishable file types, only recurrent networks gave satisfactory results. They emphasized lossless representation for achieving high accuracy, but this binary representation increases the input's dimensionality. Mittal et al. [29] developed *FiFTy*, an open-source convolutional neural network for file fragment type identification. An open-source dataset was also developed by them, which is reported to be the largest open source dataset for file fragment classification. A compact neural network was developed that used trainable embedding space and convolutional neural networks.

When compared with neural network-based classifier [8, 29, 42] our model achieves better accuracy on largest number of file types. We have only compared our models with deep learning based methods as they can be accelerated using GPUs. Our models far exceed recurrent neural network based classifier [16] by being 650x faster. Compared to machine learning based classifier [5], which takes 9 min/GB, our model takes 0.05 min/GB.

6 Conclusion

File fragments classification is an essential task in digital forensics. In this paper, we proposed an efficient file fragments classification model. Our primary focus is to develop a faster model than the current best model without compromising on accuracy. We designed a classification model for different scenarios with all having parameters less than 100K. Our model is based on depthwise separable convolution layers that are connected in inception-like fashion. The evaluation results show that depthwise separable convolutions perform fast file fragments classification with reasonable accuracy.

Several improvements can be made to increase the classification accuracy for classes with high misclassification rate. Without hardware constraints, neural architecture search [30, 44] is the best method to design an architecture for data-specific models. Moreover, for simple scenarios like classifying JPEGs against other file types, redundant connections in the nueral network can be removed using in-network distillation [17, 38].

7 Acknowledgement

The authors would like to acknowledge the Interdisciplinary Research Center for Intelligent Secure Systems at KFUPM.

References

1. Ahmed, I., Lhee, K.S., Shin, H.J., Hong, M.P.: Fast content-based file type identification. Advances in Digital Forensics VII IFIP Advances in Information and Communication Technology p. 65–75 (2011). https://doi.org/10.1007978-3-642-24212-0_5
2. Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M.: Optuna: A next-generation hyperparameter optimization framework. In: Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2019)
3. Alghafli, K., Yeun, C.Y., Damiani, E.: Techniques for measuring the probability of adjacency between carved video fragments: The vidcarve approach. IEEE Transactions on Sustainable Computing (2019)
4. Amirani, M.C., Toorani, M., Mihandoost, S.: Feature-based type identification of file fragments. Security and Communication Networks **6**, 115–128 (2013)
5. Beebe, N.L., Maddox, L.A., Liu, L., Sun, M.: Sceadan: using concatenated n-gram vectors for improved file and data type classification. IEEE Transactions on Information Forensics and Security **8**(9), 1519–1530 (2013)
6. Bennett, D.: The challenges facing computer forensics investigators in obtaining information from mobile devices for use in criminal investigations. Information Security Journal: A Global Perspective **21**(3), 159–168 (2012)
7. Bergstra, J.S., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 24, pp. 2546–2554. Curran Associates, Inc. (2011), http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf
8. Chen, Q., Hui, L.C., Liu, D., Zhang, E., Liao, Q., Jiang, Z.L., Fang, J., Yiu, S., Xi, G., Li, R.e.a.: File fragment classification using grayscale image conversion and deep learning in digital forensics. 2018 IEEE Security and Privacy Workshops (SPW) (2018). https://doi.org/10.1109/spw.2018.00029
9. Chollet, F.: Xception: Deep learning with depthwise separable convolutions. pp. 1800–1807 (07 2017). https://doi.org/10.1109/CVPR.2017.195
10. Darwin, I.F.: Libmagic (2008), ftp://ftp.astron.com/pub/file
11. Dunne, R.A., Campbell, N.A.: On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In: Proc. 8th Aust. Conf. on the Neural Networks, Melbourne. vol. 181, p. 185. Citeseer (1997)

12. Fitzgerald, S., Mathews, G., Morris, C., Zhulyn, O.: Using nlp techniques for file fragment classification. Digital Investigation **9** (2012). https://doi.org/10.1016/j.diin.2012.05.008
13. Fukushima, K.: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol. Cybern. **36**, 193–202 (1980)
14. Garfinkel, S., Farrell, P., Roussev, V., Dinolt, G.: Bringing science to digital forensics with standardized forensic corpora. Digital Investigation **6** (2009). https://doi.org/10.1016/j.diin.2009.06.016
15. Garfinkel, S.L.: Carving contiguous and fragmented files with fast object validation. digital investigation **4**, 2–12 (2007)
16. Hiester, L.: File fragment classification using neural networks with lossless representations (2018)
17. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531 (2015)
18. Hochreiter, S.: The vanishing gradient problem during learning recurrent neural nets and problem solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **6**(02), 107–116 (1998)
19. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
20. Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE International Conference on Computer Vision. pp. 1314–1324 (2019)
21. Howard, A., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: Efficient convolutional neural networks for mobile vision applications (04 2017)
22. Karresand, M., Shahmehri, N.: Oscar - file type identification of binary data in disk clusters and ram pages. In: SEC (2006)
23. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
24. Lei, Z.: Forensic analysis of unallocated space. Ph.D. thesis, UOIT (2011)
25. Li, Q., Ong, A., Suganthan, P., Thing, V.: A novel support vector machine approach to high entropy data fragment classification (01 2010)
26. Lin, X.: File carving. In: Introductory Computer Forensics, pp. 211–233. Springer (2018)
27. Marziale, L., Richard III, G.G., Roussev, V.: Massive threading: Using gpus to increase the performance of digital forensics tools. digital investigation **4**, 73–81 (2007)
28. Memon, G.M.P.K.N.: File fragment type (fft) - 75 dataset (2019). https://doi.org/10.21227/kfxw-8084, http://dx.doi.org/10.21227/kfxw-8084
29. Mittal, G., Korus, P., Memon, N.: Fifty: large-scale file fragment type identification using convolutional neural networks. IEEE Transactions on Information Forensics and Security **16**, 28–41 (2020)
30. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. arXiv preprint arXiv:1802.03268 (2018)
31. Pratt, L.Y.: Discriminability-based transfer between neural networks. In: Advances in neural information processing systems. pp. 204–211 (1993)
32. Rafique, M., Khan, M.: Exploring static and live digital forensics: Methods, practices and tools. International Journal of Scientific & Engineering Research **4**(10), 1048–1056 (2013)
33. Richard III, G.G., Roussev, V.: Scalpel: A frugal, high performance file carver. In: DFRWS. Citeseer (2005)

34. SIfre, L., Mallat, S.: Rigid-motion scattering for texture classification (03 2014)
35. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR **abs/1409.1556** (2014), http://arxiv.org/abs/1409.1556
36. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Computer Vision and Pattern Recognition (CVPR) (2015), http://arxiv.org/abs/1409.4842
37. Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. arXiv preprint arXiv:1905.11946 (2019)
38. Tian, Y., Krishnan, D., Isola, P.: Contrastive representation distillation. arXiv preprint arXiv:1910.10699 (2019)
39. Vulinović, K., Ivković, L., Petrović, J., Skračić, K., Pale, P.: Neural networks for file fragment classification. In: 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp. 1194–1198. IEEE (2019)
40. Wallace, G.K.: The jpeg still picture compression standard. Communications of the ACM **34**(4), 30–44 (1991). https://doi.org/10.1145/103085.103089
41. Wang, F., Quach, T.T., Wheeler, J., Aimone, J.B., James, C.D.: Sparse coding for n-gram feature extraction and training for file fragment classification. IEEE Transactions on Information Forensics and Security **13**(10), 2553–2562 (2018)
42. Wang, Y., Su, Z., Song, D.: File fragment type identification with convolutional neural networks. Proceedings of the 2018 International Conference on Machine Learning Technologies - ICMLT 18 (2018). https://doi.org/10.1145/3231884.3231889
43. Zheng, N., Wang, J., Wu, T., Xu, M.: A fragment classification method depending on data type. 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing pp. 1948–1953 (2015)
44. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8697–8710 (2018)