FiFTy: Large-scale File Fragment Type Identification using Neural Networks

Govind Mittal, Paweł Korus and Nasir Memon, *Fellow, IEEE*

***Abstract*—We present *FiFTy*, a modern file type identification tool for memory forensics and data carving. In contrast to previous approaches based on hand-crafted features, we design a compact neural network architecture, which uses a trainable embedding space, akin to successful natural language processing models. Our approach dispenses with explicit feature extraction which is a bottleneck in legacy systems. We evaluate the proposed method on a novel dataset with 75 file types - the most diverse and balanced dataset reported to date. *FiFTy* consistently outperforms all baselines in terms of speed, accuracy and individual misclassification rates. We achieved an average accuracy of 77.5% with processing speed of ≈38 sec/GB, which is better and more than an order of magnitude faster than the previous state-of-the-art tool - Sceadan (69% at 9 min/GB). Our tool and the corresponding dataset are available publicly online.**

***Index Terms*—file type classification, memory forensics, carving, machine learning, convolutional neural network.**

I. Introduction

Memory forensics and data carving heavily rely on data/file-type identification from small chunks of memory (Fig. 1). Many existing techniques are based on simple heuristics (like common byte markers) or hand-engineered features (unigrams [1–3], bigrams [1, 2], Shannon entropy [1–4] or Koglomorov complexity [1–4], etc. - see Section II for more details). Such an approach suffers from multiple scalability issues, ranging from a limited number of supported file types, deteriorating classification accuracy, up to slow processing speed. Feature extraction alone is responsible for a bulk of the processing time [1]. (Our evaluation corroborates this; see Tab. IV.) Some studies consider boosting, and compute individual features only when necessary [5].

To address these limitations, we explore modern neural networks to dispense with explicit feature computation. We seek compact architectures delivering good balance between accuracy and speed. Our models are 1-D convolutional neural networks (CNNs) that take blocks of raw bytes as input, and embed them in a trainable latent space - akin to successful techniques from natural language processing [6]. The resulting classifier consistently outperforms state-of-the-art models reported in the literature and can be an order of magnitude faster. We base our results on a novel dataset with 75 diverse and representative file types (nearly twice as much as the largest prior work [1]). Our model achieves an average accuracy of 77.5% (using blocks of 4,096 bytes) with processing speed of ≈38 sec/GB. The previous state-of-the art system, Sceadan [1], achieved 69% accuracy and required 9 min/GB in the same conditions (detailed breakdown in Tab. I).

While several recent studies started to explore deep learning models for the problem at hand, their solutions have fallen short. Hiester [7] considered fully connected, convolutional and recurrent networks, but studied only 4 very different file types (JPEG, GIF, XML and CSV) which can be easily distinguished using simple techniques. The proposed models convert input byte blocks to 1-hot encoding, which leads to unnecessary dimensionality explosion. Chen et al. [8] took a different approach, and reshaped 512-byte blocks as 64×64 grayscale images. Despite using modern CNNs and a small 16-type dataset, their approached delivered sub-par performance with accuracy of only 71%.

Our study is significantly broader in scope. In addition to the largest reported dataset (Tab. I), we also study several smaller scenarios with application-specific selection of file types. We focus on graphic and photographic data, which occur in photo carving applications. Our selection reflects the most popular files, that can be found on SD cards from modern cameras, smartphones and camera-enabled IoT devices. We also include multiple previously unstudied types like HEIC/HEIF (high efficiency image file format [9]) and numerous RAW photo varieties (including GPR from GoPro cameras). To accommodate variations of cluster size in different file systems, we consider both 512 and 4,096-byte inputs.

To ensure fair comparison, we evaluate our approach against three strong baselines: 1) a state-of-the-art file type classifier (*Sceadan* [1]); 2) a fully connected neural network trained on commonly used hand-crafted global block features (denoted as *NN-GF*); 3) a convolutional neural network trained on byte co-occurrences (*NN-CO*). The comparison was done on the same datasets, and using the same computational resources (using GPU acceleration, when possible). The proposed approach consistently outperformed all baselines with respect to speed, overall accuracy, and individual misclassification rates.

The contributions of our work include:

- A compact neural network architecture for file type classification with state-of-the-art performance and runtime; we also explore the hyperparameter space using the Tree-structured Parzen Estimator (TPE) [10].
- Detailed evaluation of the proposed approach against 3 strong baselines, including a state-of-the-art tool based on support vector machines [1], and modern neural networks trained on hand-crafted features.
- Detailed analysis and discussion of several classification scenarios relevant to photo carving applications.
- A ready-to-use, open-source implementation (Python 3) available at https://github.com/mittalgovind/fifty and via

G. Mittal and N. Memon are with the Tandon School of Engineering, New York University, USA (e-mail: {mittal,memon}@nyu.edu).

P. Korus is with the Tandon School of Engineering, New York University, USA, and also with the Department of Telecommunications, AGH University of Science and Technology, Poland (e-mail: pkorus@nyu.edu).

Supplementary materials, inlcuding source code, all data sets, and additional results are freely available at https://github.com/mittalgovind/fifty `pip3 install fifty`.

- A novel data-set (75 popular file types) publicly available at http://dx.doi.org/10.21227/kfxw-8084.

The paper is organized as follows. Section II, reviews related work. In Section III, we describe the proposed model, and discuss key design decisions and hyper-parameter selection. We report the results of experimental evaluation in Section IV and conclude in Section V.

II. Related Work

Most of the previous tools relied on hand-crafted heuristic features, and were limited to very small datasets. Only recently, researchers began exploring deep learning, but the reported solutions were still limited in both scope and performance. In Tab. I, we summarize the approaches, evaluation scope and the performance of previous studies. The table is divided into two sections. The top 4 rows summarize the results of our evaluation of *FiFTy* and three baseline methods (*Sceadan, NN-GF, NN-CO*) in the same conditions (all 75 file types, 4,096-byte blocks; same hardware). The remaining rows summarize the results reported in original publications.

A. Hand-crafted Features

Karresand et al. [11] developed *Oscar*, a classifier based on two features: rate of change (RoC) of consecutive bytes, and binary frequency distribution (BFD). They defined a 1-norm metric as the distance of these features from precomputed centroids. This approach worked well for JPEG photos due to *byte-stuffing* [12], a technique involving insertion of `0x00` after `0xFF`. The RoC distribution captures this feature easily and thus leads to near-perfect detection of JPEG files (99.2%) but with high false positive rates for other high-entropy filetypes, e.g., Windows executables and ZIP.

Veenman [3] trained a supervised classifier on a dataset described by 1-grams, entropy, and Kolmogorov complexity. They employed a two-stage strategy with multi-class and binary classification (one-against-all) using the Fischer linear discriminant (FLD) aiming to maximize inter-class distances. While this method yielded high accuracy for JPEG files (98%), it averaged to only 45% on a dataset with 11 filetypes. Calhoun et al. [13] also used the FLD, but included additional features, e.g., longest common strings and subsequences. While they report good results, and consider various feature combinations, the study was limited to 4 filetypes (JPG, PDF, GIF and BMP).

Fitzgerald et al. [2] used unigrams and bigrams along with entropy and complexity measures to train an SVM classifier. They used the bag-of-words model (here, bag-of-bytes) to classify text documents (file fragment). This method worked well for text formats but failed for all high entropy filetypes.

Beebe et al. [1] developed *Sceadan*, a tool based on various statistics features, including unigrams, bigrams, entropy, longest streak, etc. The features were originally extracted from 512-byte blocks, and organized into four sets: unigram, bigram, all other global features and a subset of global features. These feature sets were then used for classification using SVMs with linear and radial basis function (RBF) kernels. The authors concluded that increasing the number of types of features had a negative effect on the classification accuracy, and that unigrams & bigrams worked best as they are. Until now, *Sceadan* remains the state-of-the-art open source tool and the corresponding paper is the most comprehensive study to date - it covered 38 diverse filetypes and reported accuracy of 73.8%. We used *Sceadan* as a strong baseline in our experiments. In Tab. I, we collect both the results reported by the authors, as well as the results from our evaluation (with both 512 and 4096-byte blocks). More information about the experiment is available in Section IV-B.

Xu et al. [14] converted 512-byte blocks into 32×32 px grayscale images and extracted a 512-D GIST image descriptors based on Gabor filters. Finally, they experimented with various classification algorithms, (e.g., Naive-Bayes, SVMs, KNN, SMO, J48) and concluded that KNN performed best both in terms of speed and accuracy.

Zheng et al. [15] also relied on SVMs but employed only two features (binary frequency distribution and Shannon entropy). They explicitly differentiate between *file-types* (that could contain other embedded files, e.g., PDF and DOC) and *data-types* (that are often embedded within other files, e.g., JPEG photos), but ultimately deal only with filetypes without any embedded foreign objects.

Beebe et al. [16] experimented with clustering of 52 filetypes, and proposed a 2-layer hierarchical structure where filetypes are first grouped into 6 classes and classifiers are trained separately for each class. The authors used the same features as all other studies, including unigrams, bigrams and global statistical features (mean, entropy, complexity, etc.). The paper focuses on exploratory analysis based on K-means clustering, and reports file-level accuracy of 74.08%. The utilized classification strategy is not reported.

B. Automatic Feature Extraction

Wang et al. [4] used sparse coding for automatic feature extraction. They compute higher-order N-grams (N = 4, 8, 16, 32, 64) and store them in dictionaries. The dictionaries are subsequently averaged and concatenated with bigram and unigram frequencies. Similarly to other studies, feature extraction takes most of the processing time and makes the approach infeasible for high-volume applications. The reported accuracy was also sub-par, reaching on average 61.3% for 18 filetypes.

Hiester [7] considered fully connected, convolutional and recurrent neural networks with raw bytes fed directly as model inputs (1-hot encoded). The study included only 4 very different filetypes (JPG, GIF, XML and CSV) which are easy to distinguish with existing techniques. The adopted 1-hot encoding contributes to excessive dimensionality and slow training. Only the recurrent network achieved satisfactory accuracy, and did so at excessive computational effort. The presented approach is not scalable to real-world applications.

Chen et al. [8] took an approach similar to Xu et al. [14] by converting 512-byte fragments into 64×64 images. Despite using a CNN architecture, the model delivers sub-par performance with accuracy of only 71% among 16 filetypes. Moreover, the network fails to classify even very distinct types like HTML or plain text files.

TABLE I

QUANTITATIVE REVIEW OF RELATED WORK IN REVERSE CHRONOLOGICAL ORDER; REPORTED METRICS CORRESPOND TO: OUR EXPERIMENTS ON THE SAME DATASET (75 FILE TYPES; 512 AND 4,096-BYTE INPUTS) AND HARDWARE (TOP 4 ROWS); AND RESULTS REPORTED IN ORIGINAL PUBLICATIONS (ALL REMAINING ROWS)

Year	**Classifier**	**Features**	**Feature dim.**	**#File types**	**Block size**	**Accuracy**	**JPEG Acc.**	**Speed [ms/block]**†	**Speed [min/GB]**†	**Training Time**‡	**Remarks**	**Ref.**
2019	CNN	raw bytes	n/a	75^1	512	65.6	83.5	0.04	1.50	7.51 hours	*FiFTy*	Self
			n/a	75^1	4096	77.5	86.3	0.15	0.64	12.53 hours		
2019	CNN	co-occurence matrix	(128, 128)	75^1	512	64.4	77.0	0.84	29.36	8.67 hours	*NN-CO*	Self
				75^1	4096	75.3	92.5	1.05	4.59	9.04 hours		
2019	NN	global stat. features	14	75^1	512	45.4	53.2	0.61	21.32	2.88 hours	*NN-GF*	Self
				75^1	4096	46.8	48.6	2.67	11.67	10.53 hours		
2019	SVM	statistical: unigram, bi-gram, entropy, etc	65792	75^1	512	57.3	81.5	1.48	51.7	44 hours	*Sceadan* [1]; our evaluation	Self
				75^1	4096	69.0	91.8	2.06	9.0	10 days*		
2018	CNN	1-hot enc. bytes	8192	4^2	512	73.4	96.5	n/a	n/a	7 hours	Few & very different file types	[7]
	RNN			4^2	512	98.04	96.5	n/a	n/a	135 hours		
2018	CNN	image pixels	(64, 64)	16^2	4096	70.9	15	n/a	n/a	n/a		[8]
2018	SVM	dictionary of N-Grams	1024 + 1296 + 1444	18^2	512	61.3	65.0	n/a	n/a	235 ms/block	Feature extraction is the bottle-neck	[4]
2016	K-means	unigrams, bigrams, and global statistics (mean, entropy, complex-ity, etc.)	n/a	$52^{2,3}$ (6 classes)	512	74.0	n/a	n/a	n/a	n/a	Focus on exploratory heirarchical clustering	[16]
2015	SVM	BFD, entropy	n/a	12^2	512	67.4	69.2	n/a	n/a	n/a	Rudimentary analysis	[15]
2014	KNN	image pixels	512 (32×32)	29^2	1024	39.7	22	n/a	n/a	0.001 ms/block	Includes various classifiers & grid search	[14]
2013	SVM	statistical: unigram, bi-gram, entropy, etc	65792	$38^{2,3}$	512	73.8	81	0.30	10.6	n/a	*Sceadan*; as reported by authors	[1]
2012	SVM	unigram, bigram, entropy, Hamming weight, complex-ity, longest cont. streak	$256 + 256^2 + 6$	24^2	512	49.1	17.4	n/a	n/a	n/a	Poor results for all high-entropy file types (incl. JPG)	[2]
2008	FLD	longest common substrings, other statistical measures	21	4^3	512	86	83.6	n/a	n/a	n/a	Pair-wise analysis	[13]
			-	4^3	896	88.3	92	n/a	n/a	n/a		
2007	FLD	1-gram, entropy, Kolmogorov complexity	n/a	11^3	4096	45	98	n/a	n/a	n/a		[3]
2006	VQ	rate of change and binary freq. distribution	n/a	4^3	512	n/a	99.2	0.02	0.6	n/a	JPEG-specific method (byte mark-ers); high FPR for other types	[11]

Dataset: ^{1} - the proposed FFT-75 dataset[17] ; ^{2} - a public dataset GovDocs1 [18] ; ^{3} - Private dataset; * ref. section IV-C; † - Computed on Tesla V100 (Top 4 rows) ; ‡ - Computed on Tesla P40 (Top 4)

III. Proposed Method

This section discusses our motivations for the approach, use-cases, description of the general architecture of models included in *FiFTy* and the dataset used.

A. Motivations and Assumptions

The key motivation for developing *FiFTy* was to facilitate development of an advanced photo carving tool. Many previous carvers rely on simple heuristics and entropy thresholding to find JPEG blocks in a raw disk image [19]. Adoption of previous file type classifiers is limited due to lack of specialization and scalability issues. Most studies classified JPEG images against completely different formats (CSV, HTML, DOC, XLS) which are unlikely to occur in memory images from cameras and IoT devices. The complexity of the problem leads to trade-offs which can compromise the speed [4] or accuracy [1, 14]. While allowing for rapid triage, reliance on simple heuristics leads to excessive false positive matches, which need to be considered and discarded at later stages. This bottleneck affects even the most recent carvers [19–22].

In this work, we focus on three aspects: *speed, accuracy* and *generality*. We rely on modern neural networks, which can automatically learn effective feature representations from raw bytes and dispense with explicit feature computation - a major bottleneck in previous solutions [1] (Tab. IV). Moreover, NN models can take advantage of advanced accelerators (GPUs, TPUs, VPUs) and can be easily deployed in desktop, server, mobile and edge environments [23]. We used *Keras* which abstracts model definition and allows to exchange computational backends [24]. In our experiments, we used Python 3.6 and the Tensorflow (1.5.0) backend. All experiments were run on a server with 4 Intel Xeon CPUs (E5-2680 v4 @ 2.40GHz) and a single Tesla V100 GPU. Our models achieve nearly identical runtime regardless of the number of output classes (Tab. III).

To facilitate representation learning, we collected a novel dataset with 75 filetypes including: 6 bitmap photo formats, 11 RAW photo formats, 7 video formats, 7 audio formats, and many others. Hence, not only is it the largest reported dataset, but also the only one uniquely suited for data carving applications. However, to make our discussion as general as possible, we consider 6 application scenarios with different levels of granularity (see Section IV-A).

B. General Model Architecture

We feed raw byte blocks as the input to our models. Instead of 1-hot encoding, we use a trainable embedding layer in a manner similar to natural language processing [6]. This steps embeds long blocks of bytes within a real-valued space optimized for filetype classification, and better suited for rapid processing. Depending on the scenario at hand, this typically represents each byte with 16-64 real numbers - instead of a fixed sparse binary representation with over a million dimensions (e.g., $4096 \times 256 = 1,048,576$). In the following steps, we gradually decrease the dimensionality and reach a compact representation used for final classification.

Our architecture is 1-dimensional and relies on convolutions, followed by max-pooling to successively reduce the

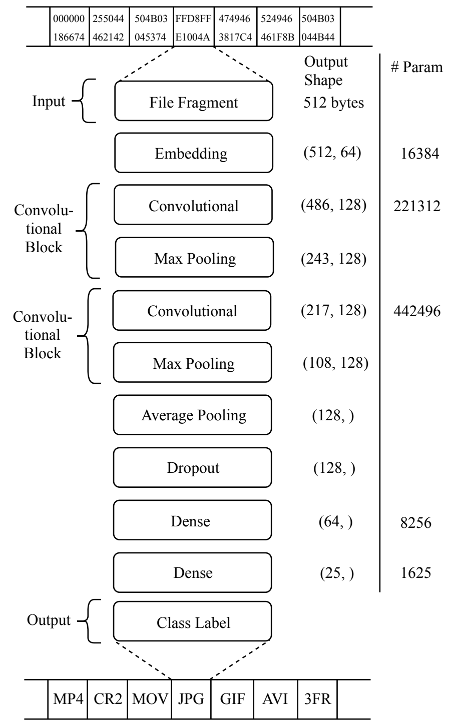


Fig. 1. Illustration of the proposed network architecture; the presented model corresponds to hyper-parameters optimized for scenario #3 w. 512-byte inputs; the reported numbers indicate how dimensionality changes with successive processing steps, and how parametrization is allocated throughout the model.

size of the problem. We refer to these units as *convolutional blocks* and vary their number as a tunable hyper-parameter. Our final vectorized representation is obtained by average pooling with dropout. In the last step, two fully connected layers map the obtained feature vectors into class probabilities. All hidden layers are activated by *Leaky Rectified Linear Units (LeakyReLU)* with $\alpha = 0.3$.

The overall architecture is shown in Fig. 1. We used the same general architecture for all tested scenarios (Section IV-A; Tab. III) and only vary hyper-parameters (Section III-C). The model shown in the figure corresponds to hyper-parameters optimized for scenario #3 with block size of 512 bytes. The reported numbers indicate how dimensionality changes with successive processing steps, and how parametrization is allocated throughout the model.

We also experimented with recurrent architectures using GRU units placed after the last convolutional layer. While initially this seemed to be more accurate, careful hyper-parameter tuning eventually reduced performance gap to 1%, which no longer justified nearly $40\times$ increase in processing time. While probably better performance could be obtained, longer processing time makes it difficult to explore the hyper-parameter space. As a result, we currently believe that carefully tuned 1-D convolutional models are the best choice with respect to the accuracy-speed trade-off.

C. Design Space and Hyper-parameter Optimization

We use the same general network architecture described in Section III-B for all our experiments, and only tune a few

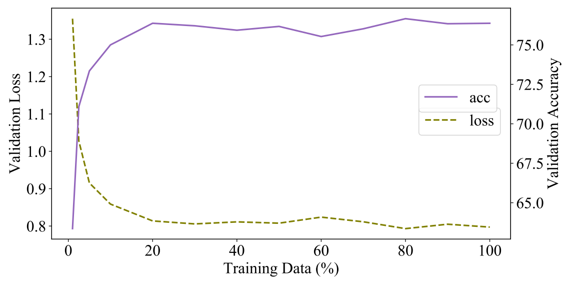
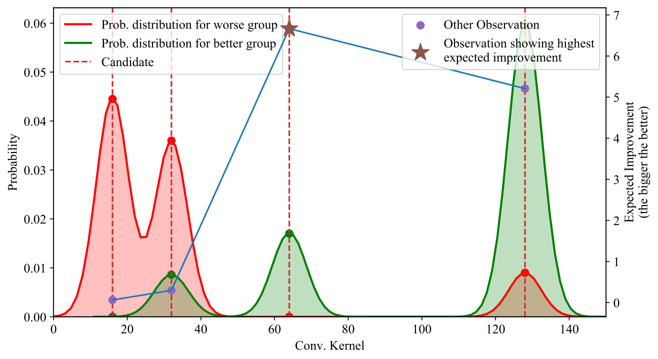
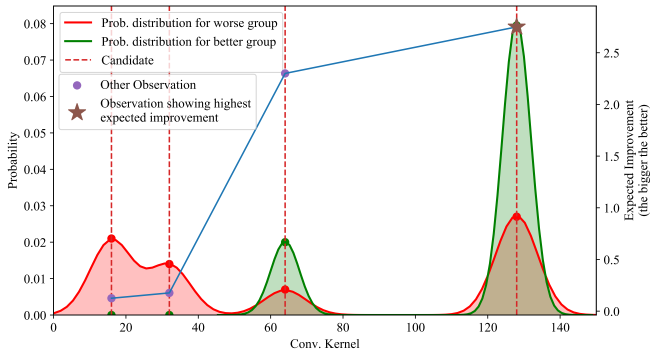


Fig. 2. Impact of training data size on validation accuracy and loss (scenario #1 with 4,096-byte blocks).



(a) After initial 20 random samples TPE selects 64 as the next value.



(b) After 200 samples, 128 seems to be the optimal value.

Fig. 3. Hyper-parameter value selection (here conv. kernel size) using TPE: probability distributions for top performers (red) and others (green) and the *expected improvement* (blue line).

hyper-parameters to adjust the model to the scenario at hand. The scenarios differ in the number of output classes and reflect various application needs, ranging from general-purpose to increasingly specialized photo caring (Section IV-A).

1) Hyper-parameter Definition: After hand tuning the general architecture, we defined 6 hyper-parameters which will be optimized automatically using the Tree-structured Parzen Estimator (TPE) [10]. For each dimension, we chose a few possible values spanning a wide range of reasonable choices.

- *# dense units:* the number of units in the hidden dense layer; $\{16, 32, 64, 128, 256\}$;
- *embedding size:* dimensionality of the trainable real-valued embedding space; $\{16, 32, 48, 64\}$;
- *conv. kernel:* the size of the convolution kernel (same for all layers); $\{16, 32, 64, 128\}$;
- *conv. stride:* convolution stride (same for all layers): $\{3, 11, 18, 27, 35\}$;
- *# conv. blocks:* the number of convolutional blocks (conv + max-pooling); $\{1, 2, 3\}$;
- *max-pooling size:* max pooling window size; $\{2, 4, 6, 8\}$.

In a separate, prior experiment, we established that adoption of *dropout* is beneficial before the hidden dense layer (with dropping probability of 0.1).

We sampled network configurations using the TPE (we used the implementation available in the *hyperopt* package [25]). In total, we collected 225 models for each of the 12 scenarios (6 base scenarios with 512 and 4,096-byte inputs, respectively). The first 20 configurations are fully random, and the remaining are sampled to maximize model performance. To speed up exploration, we used 10% of the training data and 40% of the validation data (see Fig. 2). Most scenarios completed within a day, except for the largest two (scenario #1 with 75-classes for both 512 and 4096-byte inputs) that took up to 36 hours. Finally, we retrained the best candidate models on the entire dataset, and used a separate hold-out test set for the final evaluation.

2) Tree-structured Parzen Estimator: TPE is a sequential model-based optimization (SMBO) algorithm [26] proposed by Bergstra et al. [10], which models the probability that given a scoring metric L, a hyperparameter value θ leads to a better performance, denoted as $p(\theta|L)$. The algorithm samples subsequent values for evaluation based on historical performances.

The algorithm can be summarized as follows:

1) Initialize the model with N_α random samples from the hyper-parameter space S.
2) Divide the sampled models into two groups based on a threshold on the performance quantile.
3) Estimate the probability density functions for top performers (p_1) and the rest (p_2).
4) The next sample $\hat{\theta}$ is determined using the *expected improvement*:

$$\hat{\theta} = \underset{\theta\in\Theta}{\text{argmax}}\ EI(\theta) = \underset{\theta\in\Theta}{\text{argmax}}\ \frac{p_1(\theta)}{p_2(\theta)}, \tag{1}$$

where $p_1(\theta)$ and $p_1(\theta)$ are the group membership probabilities defined above.

5) Evaluate the network defined by $\hat{\theta}$ and repeat steps 2 to 5 until N samples are collected.

Fig. 3 shows the modeled probabilities and the expected improvement (EI) for an example hyper-parameter from our design space (conv. kernel). Fig. 3(a) illustrates the initialization step of TPE. It collected 20 random samples and divided it into two equal groups using a quantile (here, 0.5). The EI is calculated and the value of 64 maximizes it. Fig. 3(b) illustrates the distributions after 200 samples have been collected, showing that the worse sample distribution is moving towards the better sample distribution (here, towards higher kernel size).

3) Final Models and Key Observations: In Fig. 4, we show the impact of the hyper-parameters on the validation accuracy for 4,096-byte inputs (see supplementary materials for the

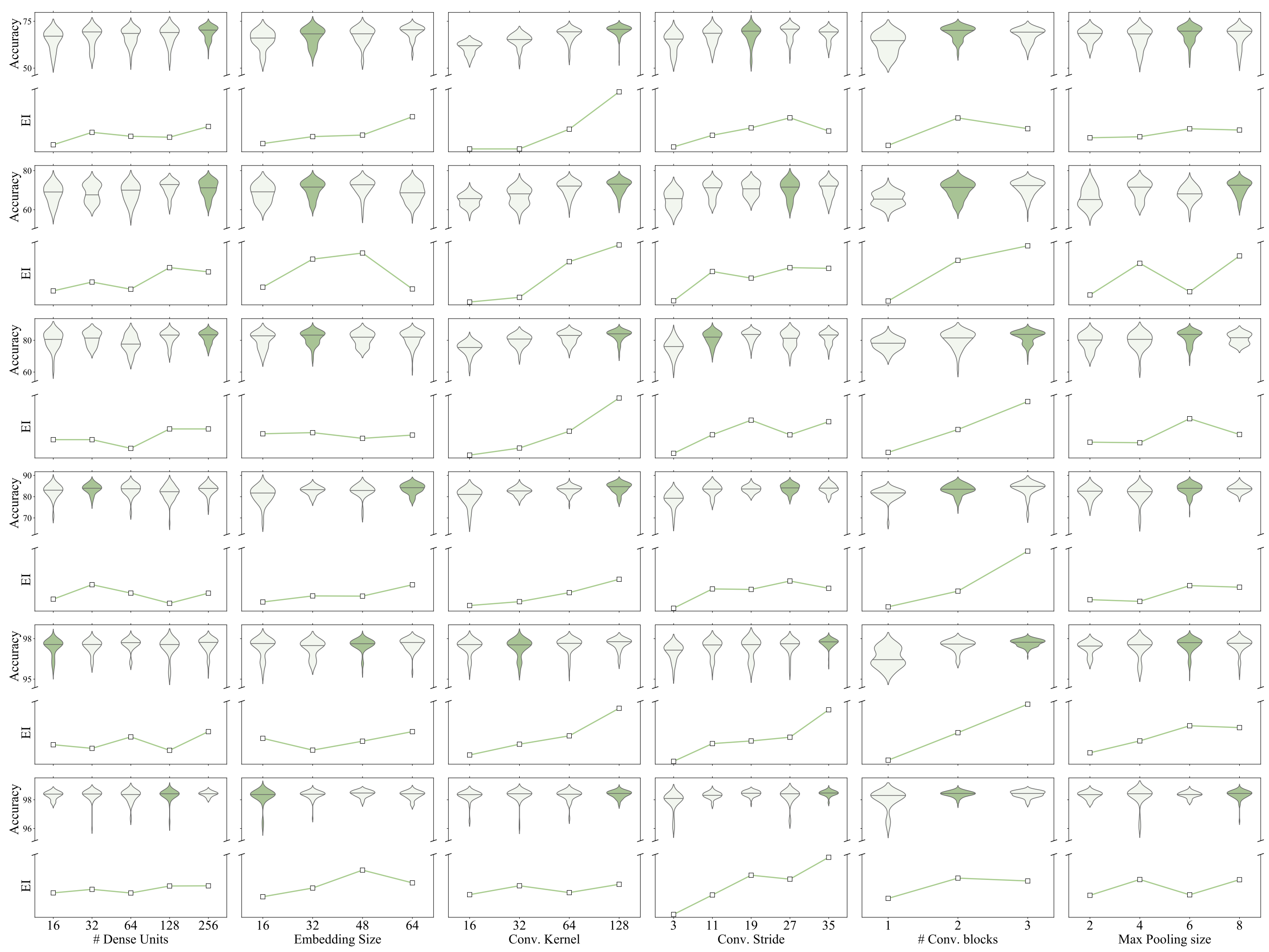


Fig. 4. Impact of hyper-parameter variation on the validation accuracy for all scenarios with 4096-byte inputs: (top) violin plots showing distribution of accuracy for sampled networks - dark color corrosponds to the TPE chosen value; (bottom) *expected improvement* estimates from TPE.

512-byte version). Each plot shows a violin plot with the distributions of validation accuracies (top), and the *expected improvement* from the TPE. The best hyper-parameters are marked in dark green. Note that these values correspond to the best sampled network, and due to non-trivial interactions and stochastic nature of the process, may not necessarily correspond to seemingly best values suggested by TPE. The key observations are as follows:

1) The convolution kernel size and stride seem to have the stringest and most consistent effect. Larger values lead to better performance regardless of the scenario.
2) A single convolutional layer is typically not enough, two or three blocks are needed for best performance.
3) In most scenarios, larger embedding sizes seem to be preferred, but good network with good performance can also be obtained for smaller embeddings - especially for smaller number of classes.
4) For 4,096-byte blocks, stronger max-pooling seems to lead to better performance.

Tab. II shows the final architectures obtained for all scenarios for both 512 and 4096-byte inputs.

IV. Experimental Evaluation

In this section, we present the results of our experimental evaluation. First, we introduce a novel dataset and distinguish several evaluation scenarios. Then, we describe 3 reference models, including a state-of-the-art tool *Sceadan* and two alternative NN-based baselines. Finally, we report our results discuss the key limitations and causes of errors.

A. Scenarios and Dataset

Previous works relied mostly on small-scale private datasets or on *GovDocs* [18] - an unbalanced corpus with 20 filetypes comprising 99.3% and 43 filetypes comprising the remaining 0.7% of the dataset. *FiFTy* comes with a novel, large and balanced corpus, named FFT-75, comprising of 75 different filetypes with emphasis on multimedia types which dominate disk images extracted from contemporary devices.

We prepared the dataset by downloading public files from various public Internet sources and from our own collection. Instead of sharing the actual files, we sampled 102,400 small blocks for each filetype to obtain a balanced dataset well-suited for classification. Large and complex files cannot be recovered

TABLE II
MODEL ARCHITECTURES OBTAINED WITH TPE FOR EACH SCENARIO AND BLOCK SIZE.

Scenario	Model Description	# Params
	***FiFTy* models for 512-byte blocks**	
#1	E (64) - C1D (128, 27) - MP (4) - AP - D (0.1) - F (256) - F (75)	289,995
#2	E (48) - C1D (128, 11) - MP (4) - C1D (128, 11) - MP (4) - AP - D (0.1) - F (64) - F (11)	269,323
#3	E (64) - C1D (128, 27) - MP (2) - C1D (128, 27) - MP (2) - AP - D (0.1) - F (64) - F (25)	690,073
#4	E (48) - C1D (128, 19) - MP (4) - C1D (128, 19) - MP (4) - AP - D (0.1) - F (256) - F (5)	474,885
#5	E (64) - C1D (128, 35) - MP (8) - AP - D (0.1) - F (256) - F (2)	336,770
#6	E (32) - C1D (128, 11) - MP (6) - C1D (128, 11) - MP (6) - AP - D (0.1) - F (64) - F (2)	242,114
	***FiFTy* models for 4,096-byte blocks**	
#1	E (32) - C1D (128, 19) - MP (4) - C1D (128, 19) - MP (4) - AP - D (0.1) - F (256) - F (75)	449,867
#2	E (32) - C1D (128, 27) - MP (8) - C1D (128, 27) - MP (8) - AP - D (0.1) - F (256) - F (11)	597,259
#3	E (32) - C1D (128, 11) - MP (6) - C1D (128, 11) - MP (6) - C1D (128, 11) - MP (6) - AP - D (0.1) - F (256) - F (25)	453,529
#4	E (64) - C1D (128, 27) - MP (6) - C1D (128, 27) - MP (6) - AP - D (0.1) - F (32) - F (5)	684,485
#5	E (48) - C1D (32, 35) - MP (6) - C1D (32, 35) - MP (6) - C1D (32, 35) - MP (6) - AP - D (0.1) - F (16) - F (2)	138,386
#6	E (16) - C1D (128, 35)- MP (8) - C1D (128, 35) - MP (8) - AP - D (0.1) - F (128) - F (2)	666,242
Baseline	**Baseline neural network models (scenario #1 for 512 and 4,096-byte blocks)**	
NN-CO	C2D (48, 3) - C2D (48, 3) - C2D (48, 3) - C2D (48, 3) - V - F (64) - F(75)	44,304,571
NN-GF	F (256) - F (256) - F (256) - F (75)	157,771

Layers: [E] embedding (⟨embedding vector length⟩); [C1D/C2D] convolution (⟨filter size, stride⟩); MP] 1-D max-pooling (⟨size⟩) ; [AP] average pooling; [F] fully connected (⟨# units⟩); [D] dropout (⟨dropout value⟩); [V] flatten.

TABLE III
DETAILED RESULTS FOR ALL 6 SCENARIOS FOR 512 AND 4096-BYTE BLOCKS.

Scen.	#Classes	Classes	Composition[1]	Accuracy		JPEG Acc.		Train. time [h]		Inf. time [$\frac{\text{ms}}{\text{block}}$]		#Samples
				512	4096	512	4096	512	4096	512	4096	
#1	75	All	75	65.6	77.5	83.5	86.3	7.51	12.53	0.043	0.146	7500k
#2	11	Bitmaps + RAW + Vector + Video + Archives + Exe + Office + Published + Human Readable + Audio + Other	(6) + (11) + (3) + (7) + (13) + (4) + (7) + (4) + (9) + (7) + (4)	78.9	89.8	-	-	1.98	7.55	0.036	0.294	1935k
#3	25	Bitmaps + RAW + Video + other	6 + 11 + 7 + (51)	87.9	94.6	93.3	98.9	4.26	10.16	0.063	0.227	2300k
#4	5	JPEG + RAW + Video + 5_Bitmaps + Other	(1) + (11) + (7) + (5) + (51)	90.2	94.1	98.6	99.1	0.50	4.95	0.043	0.400	1054k
#5	2	JPEG + non-jpegs	(1) + (74)	99.0	99.2	99.3	99.2	0.32	1.89	0.045	0.134	1036k
#6	2	JPEG + non-jpegs	(1) + (11 + 3 + 2)	99.3	99.6	99.5	99.7	0.41	4.34	0.047	0.259	1000k

[1] Values in paranthesis represent the number of file types grouped into one class; values without paranthesis represent separate classes for each file type.

from the obtained blocks. For tiny text files potentially smaller than block size, we made sure to rely on open-source materials that explicitly permit sharing. To accommodate various file-system cluster sizes, we consider both 512 and 4,096-byte blocks. The blocks are shuffled to evenly distribute the file-types within the training (80%), validation (10%) and hold-out testing (10%) subsets.

The included file-types are grouped by use-cases that include bitmaps (6 formats), RAW photographs (11), vector graphics (3), videos (7), archives (13), executable files (4), office documents (7), rich publications (4), human-readable text files (9), audios (7) and 4 others. More information, including detailed accuracy and misclassification rates are collected in Tab. VI. To the best of our knowledge, the obtained corpus is the largest and most comprehensive to date.

We distinguish *6 scenarios* that correspond to various use cases and have different granularity (Tab. III). We focus on photo carving applications, where scenarios #3 to #6 are the most relevant:

- *#1 (All; 75 classes)*: All filetypes are separate classes; this is the most generic case and can be aggregated into more specialized use-cases.
- *#2 (Use-specific; 11)*: Filetypes are grouped into 11 classes according to their use (see tags in 2nd column of Tab. VI); this information may be useful for more-detailed, hierarchical classification or for determining the primary use of an unknown device.
- *#3 (Media Carver - Photos & Videos; 25)*: Every filetype 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)*: 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)*: JPEG is a separate class and the remaining 74 filetypes are grouped into one

TABLE IV
COMPUTATION TIME FOR GLOBAL STATISTICAL & BIGRAM FEATURES.

Feature	Description	Runtime [ms/block] 4096	512
Kolgomorov Complexity	reduction of block length after bzip2/gzip compression [27, 28]	0.856	0.238
Arithmetic Mean	self explanatory	0.003	0.001
Geometric Mean	self explanatory	0.063	0.008
Harmonic Mean	self explanatory	0.048	0.007
Standard Deviation	self explanatory	0.046	0.006
Mean absolute deviation	standard deviation using abs. value of difference instead of squares	0.077	0.009
Hamming Weight	average number of set bits in the block	0.015	0.002
Kurtosis of Byte Value	measure of peakedness in the byte value distribution graph	0.110	0.015
Skewness	Measure of asymmetry of the byte value distribution graph	0.113	0.017
Longest Byte streak	length of longest streak of repeating bytes in the block.	0.981	0.156
Low ASCII Range Freq	frequency of bytes in range: `0x00 - 0x1F`	0.006	0.001
Med ASCII Range Freq	frequency of bytes in range: `0x20 - 0x7F`	0.013	0.002
High ASCII Range Freq	frequency of bytes in range: `0x80 - 0xFF`	0.018	0.002
Shannon Entropy	$\Sigma \frac{1}{p} log_2 \frac{1}{p}$	0.289	0.115
Total feature extraction time for NN-GF		**2.636**	**0.579**
Bigram matrix	byte co-occurrence matrix (NN-CO)	**0.567**	**0.370**

other class; scenario intended for analyzing disk images from generic devices.

- *#6 (Camera-Specialized JPEG Carver; 2)*: 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.

Tab. III shows the class composition of each scenario, overall accuracy, JPEG accuracy, training and inference time along with number of samples in the particular dataset for all considered scenarios. The table shows that in general the accuracy improves as the number of classes decreases - except for scenario #2 because it is more difficult than scenario #3. Scenario #2 groups filetypes by use, and not by internal structure, which may exhibit completely different statistics, e.g., the *bitmap* class consists of TIFF, HEIC, BMP, GIF, PNG and JPG which include images with no compression, as well as lossless or lossy compression. These distinct characteristics can be easily treated as separate classes in scenario #3.

B. Baseline Methods

To validate our approach, we compare *FiFTy* against three baseline methods: the state-of-the-art tool *Sceadan*, and two neural networks trained on global statistical features (*NN-GF*) and co-occurrence features (*NN-CO*). We test all methods on the largest scenario (#1) for both 512 and 4,096-byte blocks. For *Sceadan*, we used the open-source implementation and the provided training scripts [29]. For the remaining two baselines, we used the same TPE-based procedure (Section III-C) to optimize the models and their hyper-parameters. The final network architectures are collected at the bottom of Tab. II.

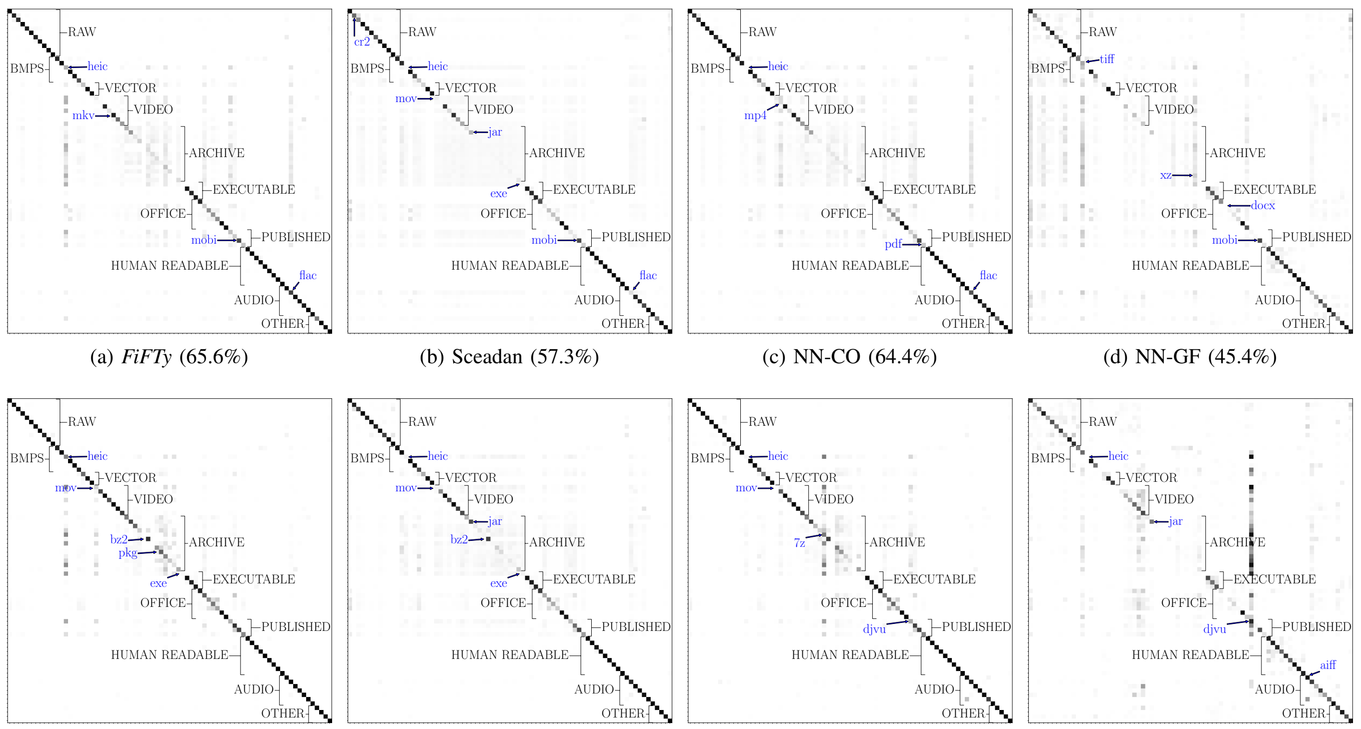
1) NN-CO: The co-occurrence or bigram features were commonly used for file type classification [1, 2, 4, 11]. The feature is constructed from a matrix of byte co-occurrences, i.e., a matrix of shape (256, 256) with each element a_{ij} corresponding to the frequency of occurrence of byte value j after byte value i. Bigram features lead to good classification accuracy, but tends to be very sparse - especially for 512-byte blocks - and relatively slow to compute (Tab. IV). As a result, they tend to be a performance bottleneck [1]. Due to the sparsity and excessive memory requirements, we down-sampled the matrices to (128, 128) by average pooling. The *NN-CO* model comprises four 2-dimensional convolutional layers, and two dense layers operating on a flattened 1-dim representation. We keep the same kernel size and stride parameters for all layers, and treat them as hyper-parameters. Additional hyper-parameters include number of convolutional layers, presence of max pooling and the number of units in the hidden dense layer.

2) NN-GF: We use 14 global statistical features (Tab. IV), commonly used in previous studies, as the input to the final baseline. Each feature maps an input block into a single scalar value. The features vary widely with respect to their computational effort, and also lead to performance bottlenecks in high-volume environments. The numbers reported in Tab. IV correspond to feature extraction in Python 3.6 performed by calling low-level routines from popular external libraries (we relied on scipy 1.3.0 and numpy 1.16.4). This makes the runtime comparable to low-level C/C++ code. The classifier consists of four dense layers, and the number of units in all hidden layers was used as separate hyper-parameters.

C. Evaluation Results

Tab. I summarizes the overall accuracy and processing speed for *FiFTy* and all of the baselines on scenario #1 with 75 file types. We report both the average accuracy and the JPEG identification accuracy. *FiFTy* clearly outperforms all of the other methods - both with respect to accuracy and runtime. (Due to clearly inferior performance, we omit *NN-GF* in the following discussion.) On 4,096-byte blocks, *FiFTy* achieved average accuracy of 77.5%, compared to 75.3% and 69.0% for the runner-ups (*NN-CO* and *Sceadan*) which were 7$\times$ and 14$\times$ slower, respectively. On the smaller 512-byte blocks, the accuracy deteriorates to 65.6% vs. 64.4% and 57.3% with an even steeper speed-up gains - 20$\times$ and 34$\times$.

In terms of training time, *FiFTy* was comparable to *NN-CO* and significantly faster than *Sceadan*, which took 44 hours for 512-byte inputs and over 10 days for 4096-byte inputs. Our model required $\approx$ 8 and 12 hours to train the final model on the entire dataset (on the slower P40 GPU), and between 24 - 36 hours for hyper-parameter search (depending on the



(a) *FiFTy* (65.6%) (b) Sceadan (57.3%) (c) NN-CO (64.4%) (d) NN-GF (45.4%)

(e) *FiFTy* (77.5%) (f) Sceadan (69.6%) (g) NN-CO (75.3%) (h) NN-GF (46.8%)

Fig. 5. Heat maps of confusion matrices for scenario #1 of *FiFTy*, Sceadan [1], NN-CO and NN-GF with block sizes of 512 bytes in subfigures **(a)-(d)** and 4096 bytes in subfigures **(e)-(h)**. Darker color means higher value.

scenario at hand). If needed, faster training times are possible - a sub-set of the training set and a slightly simpler model can cut down the training time to around 1-2 h with only minor degradation in accuracy (Fig. 2).

Fig. 5 compares confusion matrices for all of the models. Due to a large number of classes, the included figure depicts groups of file types and marks selected individual file types of interest. For detailed file-type-level results, please refer to supplementary materials or Table VI which summarizes the most common mistakes. *FiFTy* delivered the best accuracy for 43 out of 75 individual file types. The *NN-CO* was the winner for 27 and *Sceadan* for 4 (for 4,096 blocks). Whenever *FiFTy* lost, the difference was marginal.

1) Ensemble of FiFTy *and* NN *baselines:* We consider ensemble learning to assess whether inclusion of hand-engineered features could improve the accuracy. By averaging the probabilities prodicted by *FiFTy*, *NN-CO*, and *NN-GF* we can achieve 78.7% on 4,096-byte inputs - a marginal improvement not worth the enormous computational effort. We also tried training a simple combination NN, but did not observe any further improvements.

2) Additional Scenarios: Tab. III summarizes *FiFTy* evaluation results for all test scenarios. As the number of classes decreases, we obtain increasingly more reliable results, reaching 94.6 % accuracy for 25 multimedia file types and 99.7% accuracy for the most constrained JPEG-vs-others scenario. To the best of our knowledge, these are the best such results to date. The best previously reported JPEG identification accuracy was 99.2% on a dataset with only 4 very different file types [11]. Our classifier considered 16 other photographic and video types in the alternative class, which makes it suitable for photo carving applications. Even for the detailed multimedia scenario #3 (25 file types), we achieved JPEG classification accuracy of 98.9%.

TABLE V
INFERENCE TIME (MS/BLOCK) OF *FiFTy* FOR VARIOUS SCENARIOS TESTED ON A TESLA V100, P40, K80 GPU AND 4 E5-2690 CPUS.

Scenario	**512-byte blocks**				**4,096-byte blocks**			
GPUs	**V100**	**P40**	**K80**	**CPU**	**V100**	**P40**	**K80**	**CPU**
#1	0.043	0.058	0.181	3.102	0.146	0.256	0.947	16.836
#2	0.036	0.047	0.126	1.907	0.163	0.294	1.014	20.519
#3	0.063	0.086	0.353	5.694	0.119	0.227	0.628	11.526
#4	0.043	0.057	0.191	2.859	0.245	0.400	1.705	33.900
#5	0.045	0.057	0.232	3.893	0.097	0.134	0.541	12.569
#6	0.047	0.059	0.105	1.330	0.137	0.259	0.901	17.519

3) Runtime Evaluation: The classifier comprising the heart of *FiFTy* is a neural network implemented in a high-level framework Keras. The model can take advantage of GPUs to speed up computations. In this experiment, we assess the inference speed on several popular server GPUs (nNidia Tesla V100, P40 and K80). The results are collected in Tab. V. Even with parallel processing, general-purpose CPUs are not competitive for high-volume applications. Adoption of GPUs significantly speeds-up the computations. On the fastest considered GPU and with 4,096-byte blocks, we achieved the speed of 214 Mbps for the largest scenario with 75 classes, and up to 322 Mbps for the simplest 2-class problem.

D. Misclassification Analysis

The last two columns in Tab. VI show a misclassification breakdown for all of the considered file types. For brevity, we show only top-2 types most confused with each given file type (for scenario #1/4096). There are several reasons for such errors. We list the most interesting observations below:

- *HEIC & MOV:* HEIF is a new image format used in the recent iPhones [30]. It is essentially a container for still photos encoded using the intra-frame mode of the HEVC video codec [31]. This leads to misclassification errors, since HEIC photographs are essentially small movies and tend to be confused with actual MOV videos. Moreover, due to high entropy, both formats are sometimes confused with archives - especially the XZ and RAR formats.
- *Archives:* Data compression (whether lossy or lossless) reduces redundancy and stores the same information using fewer bits. Archive file types are characterized by high entropies, thus minimizing statistical artifacts which could be exploited to identify them. Almost all included types suffer from this problem. Although the proposed *FiFTy* improves upon previous methods, the accuracy still averages around only 39%. The only format with reasonable accuracy was BZ2, which obtained 80%.
- *Compound filetypes:* Many filetypes, e.g., PDF, PPT, KEY, DOC, MOBI are containers and may have embedded objects of other types, e.g., JPEG or other bitmaps, audio, video, etc. This confuses all classifiers, but seems to be mostly a data labeling problem. In practice, such segments should probably be carved using dedicated tools for the embedded objects. However, the container file should most likely be carved as well. More complex logic will be needed both for carving such data as well as properly labeling the training data for file type classifiers.
- *Presentation File Types:* We observed a cluster of mutual misclassifications for common presentation file types (PPT, PPTX and KEY). Interestingly, we didn't observe simialar relationships between document and spreadsheet types. To some extend the problem may stem from file type conversion (see below).
- *Errors stemming from type conversion:* Some files in our dataset were generated by converting from other files of a similar type (to reach the required number of samples). We observed elevated mutual misclassification rates for some of the pairs (WAV & AIFF, KEY & PPTX, PPT & PPTX and MOBI & EPUB). Other pairs (RTF & TXT, HTML & MD or TEX & MD) remain unaffected. More information about the conversion can be found in the documentation of the dataset.

E. Limitations and Future Work

The presented neural network architecture reflects both the effects of hands-on experimentation and automatic hyper-parameter tuning. Out of necessity, we explored only a small fraction of the design space. It is certainly possible to expect that better architectures could be found using modern architecture search [32] or even random sampling methods [33]. These methods still remain an area of active research, and its application for the problem at hand is left for future work.

Our current hyper-parameter search was driven mainly by validation accuracy. This sometimes leads to surprisingly high model complexity for seemingly simpler problems (Tab. III). In the future work, we're planning to investigate the relationship between model complexity (e.g., parameter count), computational effort and accuracy. The problem requires a separate treatment since it heavily depends on the target runtime environment, and both hardware platforms and tensor libraries can result in widely different behavior [34].

Additional improvements in classification accuracy can be expected by adopting more complex logic to deal with embedded filetypes as well as with the dependencies between labeling decisions among neighboring blocks. For example, random field models could be used to reach an agreement regarding a larger chunk of data based on weak predictions of individual blocks [35]. This information could further be combined with the locations of known headers/footers.

Finally, we emphasize that in real applications additional care may be needed for certain filetypes to accommodate the expected content characteristics. For example, statistical properties of video files may depend not only on the container, but also on the audio/video codecs. In our study, we relied on material that we captured using common camera-enabled devices (e.g., GoPro, dash cameras). We used default codec settings which should be most applicable in common media carving applications. Our dataset included mainly MPEG4, H.264, H.265, VP8, VP9 and OGV codecs with different distribution across containers. We leave a more detailed study of this phenomenon for future work.

V. Conclusions

Our study addressed the problem of large-scale file fragment type identification. We explored neural network models, which dispense with explicit feature engineering - a bottleneck from both development and computation perspectives. Our architecture is based on 1-D convolutions. We accept raw blocks of bytes as input, and embed them in a trainable real-valued latent space. By changing a few hyper-parameters, our architecture can be easily adapted to various applications. We used the Tree-structured Parzen Estimator [10] to analyze the impact of hyper-parameters and find their optimal values. The resulting models are included in *FiFTy*, an open source tool for file type identification.

Our evaluation is based on a novel dataset with 75 popular file types (available publicly at [17]). We focused on multimedia and photographic types which commonly occur in data carving applications. Based on the obtained results, we can conclude that neural networks are an extremely effective and flexible tool for file type identification. The trained classifiers consistently outperformed all baseline methods. Moreover, thanks to modern software frameworks and hardware accelerators, classification can run an order of magnitude faster.

References

[1] N. L Beebe, L. A Maddox, L Liu, and M Sun, "Sceadan: Using Concatenated N-Gram Vectors for Improved File and Data Type Classification,"

TABLE VI
FILE-TYPE BREAKDOWN OF CLASSIFICATION ACCURACY FOR ALL MODELS; MISCLASSIFICATION REPORTED FOR *FiFTy* ON 4,096-BYTES INPUTS.

		512 bytes				**4096 bytes**				**Misclassification**	
Filetype	**Tag**	**NN-CO**	**NN-GF**	**Sceadan**	**FiFTy**	**NN-CO**	**NN-GF**	**Sceadan**	**FiFTy**	**Top 1**	**Top 2**
ARW	Raw	97.8	88.6	97.3	**97.8**	98.8	84.7	97.5	**98.9**	JPG (0.4)	MOV (0.1)
CR2	Raw	82.4	43.1	58.2	**86.7**	**95.3**	30.0	88.9	94.8	NEF (1.1)	DNG (0.6)
DNG	Raw	81.6	18.1	47.7	**82.7**	**96.7**	54.6	75.3	96.1	CR2 (2.0)	NEF (0.4)
GPR	Raw	98.7	89.8	98.1	**99.2**	99.5	87.2	99.3	**99.9**	CR2 (<0.1)	PEF (<0.1)
NEF	Raw	**89.3**	59.6	82.5	87.7	89.0	36.1	88.3	**95.3**	NRW (0.9)	JPG (0.9)
NRW	Raw	96.1	89.4	93.9	**96.9**	96.7	68.7	92.2	**97.7**	RAF (0.9)	3FR (0.3)
ORF	Raw	84.0	37.7	63.7	**86.3**	**97.6**	51.9	87.3	96.3	FLAC (1.4)	NEF (0.7)
PEF	Raw	**96.1**	84.3	88.8	95.1	98.9	77.5	96.8	**99.1**	NEF (<0.1)	ORF (<0.1)
RAF	Raw	97.9	95.4	96.6	**98.3**	**91.3**	34.0	73.6	87.1	FLAC (4.6)	3FR (4.1)
RW2	Raw	**96.6**	92.4	95.5	96.5	97.9	89.3	95.9	**97.9**	JPG (1.0)	PPT (0.2)
3FR	Raw	99.4	96.0	96.6	**99.6**	92.6	81.0	97.2	**99.5**	DNG (0.2)	TIFF (0.2)
JPG	Bitmap	77.5	53.2	83.8	**83.5**	**92.5**	48.6	91.8	86.3	PPT (3.9)	PPTX (2.3)
TIFF	Bitmap	90.0	34.5	83.8	**96.1**	98.6	57.9	97.7	**99.0**	APK (0.2)	PPT (0.2)
HEIC	Bitmap	2.0	1.8	2.9	**31.7**	4.0	0.0	5.2	**49.5**	MOV (15.3)	XZ (11.6)
BMP	Bitmap	97.5	90.1	96.1	**98.0**	**99.4**	91.3	98.7	98.4	3FR (0.5)	DLL (0.3)
GIF	Bitmap	84.5	42.3	72.3	**93.5**	98.5	54.3	96.2	**99.1**	MSI (<0.1)	OGV (<0.1)
PNG	Bitmap	47.6	1.5	21.7	**67.2**	72.9	0.4	69.5	**88.0**	PKG (1.7)	RAR (1.2)
AI	Vector	**25.5**	12.6	17.4	23.3	50.9	10.6	31.7	**57.7**	EPUB (6.9)	PKG (5.8)
EPS	Vector	**98.8**	95.2	97.8	98.7	**98.0**	88.2	97.4	95.8	AI (3.4)	PDF (0.4)
PSD	Vector	93.7	86.2	91.2	**95.3**	95.6	75.3	94.4	**95.9**	AI (2.1)	JPG (0.4)
MOV	Video	**8.7**	3.6	6.1	6.1	0.0	0.0	4.8	**18.5**	HEIC (43.1)	RAR (13.0)
MP4	Video	**14.8**	0.0	2.8	1.6	62.2	7.0	45.5	**71.8**	HEIC (11.2)	XZ (2.9)
3GP	Video	**91.6**	23.1	79.3	85.6	**98.8**	32.1	98.7	98.1	AVI (0.6)	DMG (0.2)
AVI	Video	13.8	1.7	**14.7**	10.9	64.9	32.6	66.4	**67.1**	HEIC (13.5)	MOV (6.2)
MKV	Video	**90.5**	23.5	82.3	88.0	**99.1**	57.5	98.9	98.4	OGV (0.4)	DMG (0.2)
OGV	Video	**67.4**	4.6	37.3	57.3	94.5	59.0	76.0	**94.9**	OGG (1.3)	MSI (0.8)
WEBM	Video	39.6	21.4	33.3	**39.8**	71.5	71.6	71.2	**78.1**	HEIC (10.0)	MOV (3.2)
APK	Archive	**32.5**	1.5	10.0	29.9	48.3	9.0	29.0	**49.7**	PKG (11.5)	PPT (3.5)
JAR	Archive	36.3	20.4	28.9	**41.2**	**74.8**	53.4	70.2	73.8	GZ (6.1)	RPM (4.1)
MSI	Archive	5.1	0.3	3.3	**10.1**	26.6	1.5	7.6	**60.1**	PKG (5.2)	EPUB (3.0)
DMG	Archive	7.0	1.1	3.4	**18.2**	17.4	0.4	9.1	**19.4**	HEIC (17.4)	PKG (14.6)
7Z	Archive	0.9	1.4	**2.8**	0.0	**53.2**	0.0	4.6	0.1	HEIC (47.8)	MOV (15.4)
BZ2	Archive	8.6	1.2	2.5	**13.9**	75.8	4.3	74.8	**80.6**	DMG (8.5)	HEIC (3.2)
DEB	Archive	9.9	7.0	2.9	**13.8**	1.5	0.0	**3.9**	0.8	HEIC (38.8)	RAR (10.6)
GZ	Archive	11.3	0.0	3.4	**13.2**	**53.0**	4.9	19.1	42.1	PKG (17.9)	RPM (11.6)
PKG	Archive	**5.6**	1.9	4.5	4.5	41.0	0.8	14.8	**62.7**	GZ (6.5)	APK (5.7)
RAR	Archive	13.0	4.2	3.0	**24.0**	18.0	1.7	3.9	**46.6**	ZIP (6.7)	HEIC (6.3)
RPM	Archive	7.9	2.6	5.4	**13.6**	10.7	0.5	8.9	**21.9**	PKG (12.8)	GZ (10.1)
XZ	Archive	0.0	**19.4**	2.9	0.0	0.0	1.3	5.0	**12.1**	HEIC (45.6)	MOV (15.4)
ZIP	Archive	**16.2**	1.8	9.5	13.8	**36.6**	9.7	22.2	34.2	RAR (21.6)	M4A (5.5)
EXE	Executable	**9.2**	0.8	2.4	0.3	4.3	0.3	**5.4**	2.7	HEIC (39.7)	MOV (13.0)
MACH-O	Executable	**93.6**	75.0	90.0	92.6	**95.7**	71.8	93.7	95.0	DLL (2.1)	TXT (0.6)
ELF	Executable	**87.6**	65.4	83.7	85.9	**92.2**	54.9	89.3	86.4	DLL (4.6)	MACH-O(2.1)
DLL	Executable	**91.7**	67.8	86.2	91.1	**94.4**	65.7	86.4	91.7	TXT (2.8)	ELF (0.6)
DOC	Office	**91.4**	44.8	70.5	87.2	**91.6**	12.4	77.6	88.9	PPT (2.2)	APK (1.2)
DOCX	Office	14.7	0.7	6.3	**19.4**	49.9	0.1	31.7	**60.6**	EPUB (20.7)	RAR (8.7)
KEY	Office	35.8	4.5	23.6	**44.1**	**56.7**	10.3	44.3	43.1	PPTX (19.9)	PPT (13.2)
PPT	Office	**44.2**	10.9	23.6	41.3	**58.4**	15.7	32.5	53.5	PPTX (6.2)	PKG (3.8)
PPTX	Office	38.3	14.4	24.0	**44.4**	32.7	2.6	20.2	**36.0**	PPT (14.6)	KEY (11.5)
XLS	Office	**99.4**	98.7	99.1	99.3	99.2	96.7	99.0	**99.3**	DLL (0.3)	MSI (0.1)
XLSX	Office	**96.8**	65.6	92.7	95.2	**97.4**	20.7	91.3	97.2	PKG (0.6)	AI (0.5)
DJVU	Published	18.0	3.5	10.5	**24.6**	34.8	**84.2**	24.2	30.4	HEIC (33.7)	MOV (12.5)
EPUB	Published	**29.3**	2.0	9.2	20.7	74.3	3.3	56.0	**79.7**	DOCX (10.5)	PKG (2.0)
MOBI	Published	72.5	**72.8**	71.5	72.4	**74.4**	69.6	72.9	73.7	EPUB (19.2)	DOCX (2.4)
PDF	Published	21.9	11.5	20.0	**23.1**	**48.3**	11.6	38.1	45.8	HEIC (13.1)	MOV (4.7)
MD	Human-readable	**98.0**	77.9	93.9	97.1	**99.9**	86.6	98.9	97.4	HTML (1.7)	TEX (0.5)
RTF	Human-readable	99.5	90.2	99.7	**99.7**	100.0	87.7	**100.0**	99.8	TXT (<0.1)	DOC (<0.1)
TXT	Human-readable	95.0	85.9	**95.4**	93.7	92.9	84.1	92.5	**93.0**	DOCX (2.1)	EPUB (2.0)
TEX	Human-readable	**97.8**	66.3	93.0	97.6	**98.8**	58.1	97.2	98.3	TXT (1.6)	RTF (<0.1)
JSON	Human-readable	**99.8**	84.3	99.5	99.5	**100.0**	81.5	99.8	99.8	TXT (<0.1)	HTML (<0.1)
HTML	Human-readable	**98.4**	74.2	94.9	97.5	99.5	59.1	98.9	**99.6**	TXT (0.2)	MD (0.1)
XML	Human-readable	100.0	99.3	99.9	**100.0**	100.0	99.9	100.0	**100.0**	-	-
LOG	Human-readable	**100.0**	99.3	99.9	99.9	99.9	91.2	99.9	**99.9**	TXT (<0.1)	TXT (<0.1)
CSV	Human-readable	**99.9**	92.1	99.6	99.6	99.0	87.3	98.4	**99.7**	TXT (<0.1)	TXT (<0.1)
AIFF	Audio	95.7	96.2	94.9	**99.4**	94.5	95.0	98.4	**98.8**	TXT (0.8)	TXT (0.8)
FLAC	Audio	66.9	18.5	13.2	**74.2**	87.8	76.2	85.3	**97.9**	ORF (0.4)	ORF (0.4)
M4A	Audio	92.6	34.6	86.7	**94.3**	96.5	29.6	**98.4**	98.2	HEIC (0.3)	MOV (0.3)
MP3	Audio	94.0	70.8	85.1	**94.4**	98.9	77.6	98.8	**98.9**	APK (0.2)	APK (0.2)
OGG	Audio	84.2	55.3	72.3	**90.4**	95.4	37.3	93.3	**95.5**	FLAC (2.3)	FLAC (2.3)
WAV	Audio	99.9	99.9	99.8	**100.0**	74.0	58.2	97.6	**98.7**	TXT (0.6)	APK (0.4)
WMA	Audio	**99.2**	73.8	98.7	98.2	99.9	95.7	99.9	**99.9**	3GP (<0.1)	APK (<0.1)
PCAP	Misc	**55.4**	32.9	48.6	51.0	**95.6**	69.6	94.7	95.4	HEIC (0.9)	MOV (0.4)
TTF	Misc	98.1	86.8	96.3	**98.1**	99.2	90.2	98.4	**99.3**	DLL (0.3)	DLL (0.3)
DWG	Misc	**96.5**	90.1	93.0	96.2	**98.8**	67.5	97.0	96.9	BMP (0.4)	BMP (0.4)
SQLITE	Misc	**98.9**	87.2	97.1	98.6	99.5	91.3	98.6	**99.8**	DLL (<0.1)	DLL (<0.1)
-	**Total Wins**	**32**	**2**	**3**	***38***	**27**	**1**	**4**	***43***	-	-

Note: Ties were broken depending upon runtime speed.

IEEE Tran. Inf. Forensics and Security, vol. 8, no. 9, pp. 1519–1530, 2013.

[2] S Fitzgerald, G Mathews, C Morris, and O Zhulyn, "Using NLP techniques for file fragment classification," *Digital Investigation*, vol. 9, pp. S44–S49, aug 2012.

[3] C. J Veenman, "Statistical Disk Cluster Classification for File Carving," in *IEEE Int. Symposium on Information Assurance and Security*, 2007.

[4] F Wang, T.-T Quach, J Wheeler, J. B Aimone, and C. D James, "Sparse Coding for N-Gram Feature Extraction and Training for File Fragment Classification," *IEEE Tran. Inf. Forensics and Security*, vol. 13, no. 10, pp. 2553–2562, oct 2018.

[5] L Li, U Topkara, B Coskun, and N Memon, "Cocost: a computational cost efficient classifier," in *2009 Ninth IEEE International Conference on Data Mining*. IEEE, 2009, pp. 268–277.

[6] T Mikolov, K Chen, G Corrado, and J Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[7] L Hiester, "File Fragment Classification Using Neural Networks with Lossless Representations," 2018.

[8] Q Chen, Q Liao, Z. L Jiang, J Fang, S Yiu, G Xi, R Li, Z Yi, X Wang, L. C Hui, D Liu, and E Zhang, "File Fragment Classification Using Grayscale Image Conversion and Deep Learning in Digital Forensics," in *IEEE Security and Privacy Workshops*, 2018.

[9] M Hannuksela, E Aksu, J Lainema, and V. M Vadakital, "Overview of the high efficiency image file format," in *JCTVC-V0072*. 2015.

[10] J. S Bergstra, R Bardenet, Y Bengio, and B Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, 2011, pp. 2546–2554.

[11] M Karresand and N Shahmehri, "File Type Identification of Data Fragments by Their Binary Structure," in *2006 IEEE Information Assurance Workshop*.

[12] G. K Wallace, "The jpeg still picture compression standard," *IEEE Tran. consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[13] W. C Calhoun and D Coles, "Predicting the types of file fragments," *Digital Investigation*, vol. 5, pp. S14–S20, sep 2008.

[14] T Xu, M Xu, Y Ren, J Xu, H Zhang, and N Zheng, "A File Fragment Classification Method Based on Grayscale Image," *Journal of Computers*, vol. 9, no. 8, aug 2014.

[15] N Zheng, J Wang, T Wu, and M Xu, "A Fragment Classification Method Depending on Data Type," in *IEEE Int. Conf. on Computer and Information Technology Ubiquitous Computing and Communications Dependable Autonomic and Secure Computing Pervasive Intelligence and Computing*, oct 2015.

[16] N Beebe, L Liu, and M Sun, "Data Type Classification: Hierarchical Class-to-Type Modeling," in *IFIP Advances in Information and Communication Technology*, pp. 325–343. Springer Int. Publishing, 2016.

[17] G Mittal, P Korus, and N Memon, "File fragment type (fft) - 75 dataset," http://dx.doi.org/10.21227/kfxw-8084, 2019.

[18] "GovDocs Dataset," https://digitalcorpora.org/corpora/files, Accessed Jun 20, 2019.

[19] J De Bock and P De Smet, "Jpgcarve: An advanced tool for automated recovery of fragmented jpeg files," *IEEE Tran. Inf. Forensics and Security*, vol. 11, no. 1, pp. 19–34, Jan 2016.

[20] "JPEG Recovery Lab," https://www.disktuna.com/photo-recovery-easeus-data-recovery-vs-jpeg-recovery-lab/, Accessed Jun 20, 2019.

[21] E Uzun and H. T Sencar, "Carving orphaned jpeg file fragments," *IEEE Tran. Inf. Forensics and Security*, vol. 10, no. 8, pp. 1549–1563, 2015.

[22] E Durmus, P Korus, and N Memon, "Every shred helps: Assembling evidence from orphaned jpeg fragments," *IEEE Tran. Inf. Forensics and Security*, vol. 14, no. 9, pp. 2372–2386, 2019.

[23] "Intel Movidius Myriad X VPU," https://www.movidius.com/myriadx, Accessed Jun 20, 2019.

[24] F Chollet et al., "Keras," https://keras.io, 2015.

[25] J Bergstra, D Yamins, and D. D Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," 2013.

[26] F Hutter, H. H Hoos, and K Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Int. Conf. on Learning and Intelligent Optimization*, Berlin, Heidelberg, 2011, LION'05, pp. 507–523, Springer-Verlag.

[27] "Bz2 package," https://docs.python.org/3/library/bz2.html, Accessed Jun 20, 2019.

[28] "Zlib package," https://docs.python.org/3/library/zlib.html, Accessed Jun 20, 2019.

[29] "Sceadan - open source implementation," https://github.com/UTSA-cyber/sceadan.

[30] "Everything you need to know about the JPEG-killing HEIF format Apple is adopting," https://www.cultofmac.com/487808/heif-vs-jpeg-image-files/, Accessed Jun 20, 2019.

[31] G. J Sullivan, J.-R Ohm, W.-J Han, and T Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Tran. Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, 2012.

[32] C Liu, B Zoph, M Neumann, J Shlens, W Hua, L.-J Li, L Fei-Fei, A Yuille, J Huang, and K Murphy, "Progressive neural architecture search," in *Proc. European Conf. Computer Vision*, 2018, pp. 19–34.

[33] S Xie, A Kirillov, R Girshick, and K He, "Exploring randomly wired neural networks for image recognition," *arXiv preprint arXiv:1904.01569*, 2019.

[34] S Shi, Q Wang, P Xu, and X Chu, "Benchmarking state-of-the-art deep learning software tools," in *Int. Conf. Cloud Computing and Big Data (CCBD)*. IEEE, 2016, pp. 99–104.

[35] J Lafferty, A McCallum, and F. C Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

Supplementary Materials for FiFTy: Large-scale File Fragment Type Identification using Neural Networks

Govind Mittal[1], Paweł Korus[1,2], and Nasir Memon[1]
New York University[1], AGH University of Science and Technology[2]

I. Source Code

Our file fragment classifier toolbox is available at https://github.com/mittalgovind/fifty and the supporting FFT-75 dataset is available at http://dx.doi.org/10.21227/kfxw-8084

II. Contents

- Effect of hyper-parameter variation on various scenarios with block size of 512 and 4096 bytes.
- Full confusion matrices for block sizes of 512 and 4096 bytes of :
  - FiFTy
  - Sceadan
  - NN-GF
  - NN-CO
  - Scenario #2 - #6
- List of sources for each filetype in the corpus FFT-75.

Fig. 1. Impact of hyper-parameter variation on the validation accuracy for all scenarios with 512-byte inputs: (top) violin plots showing distribution of accuracy for sampled networks - dark color corrosponds to the TPE chosen value; (bottom) *expected improvement* estimates from TPE.

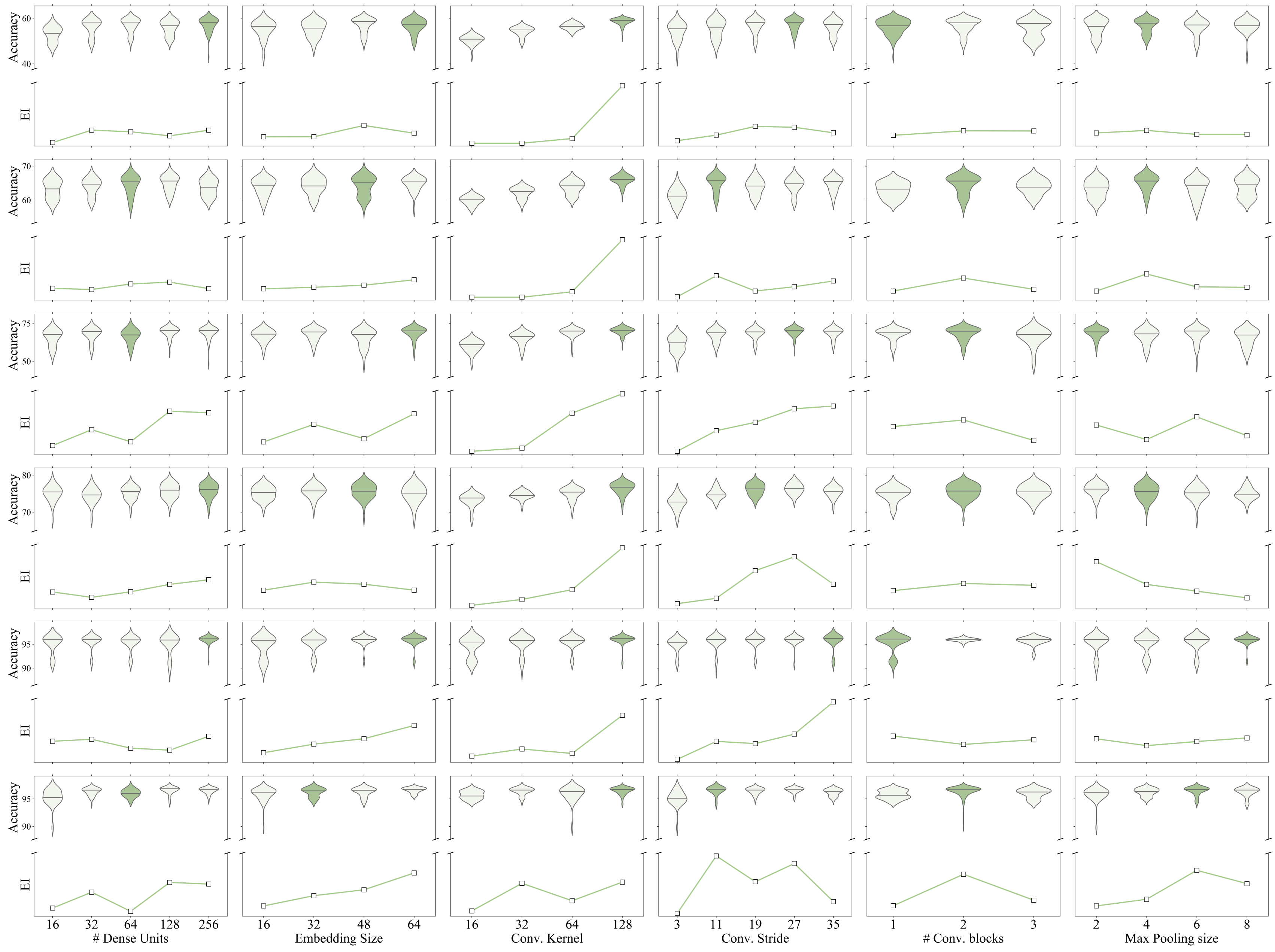
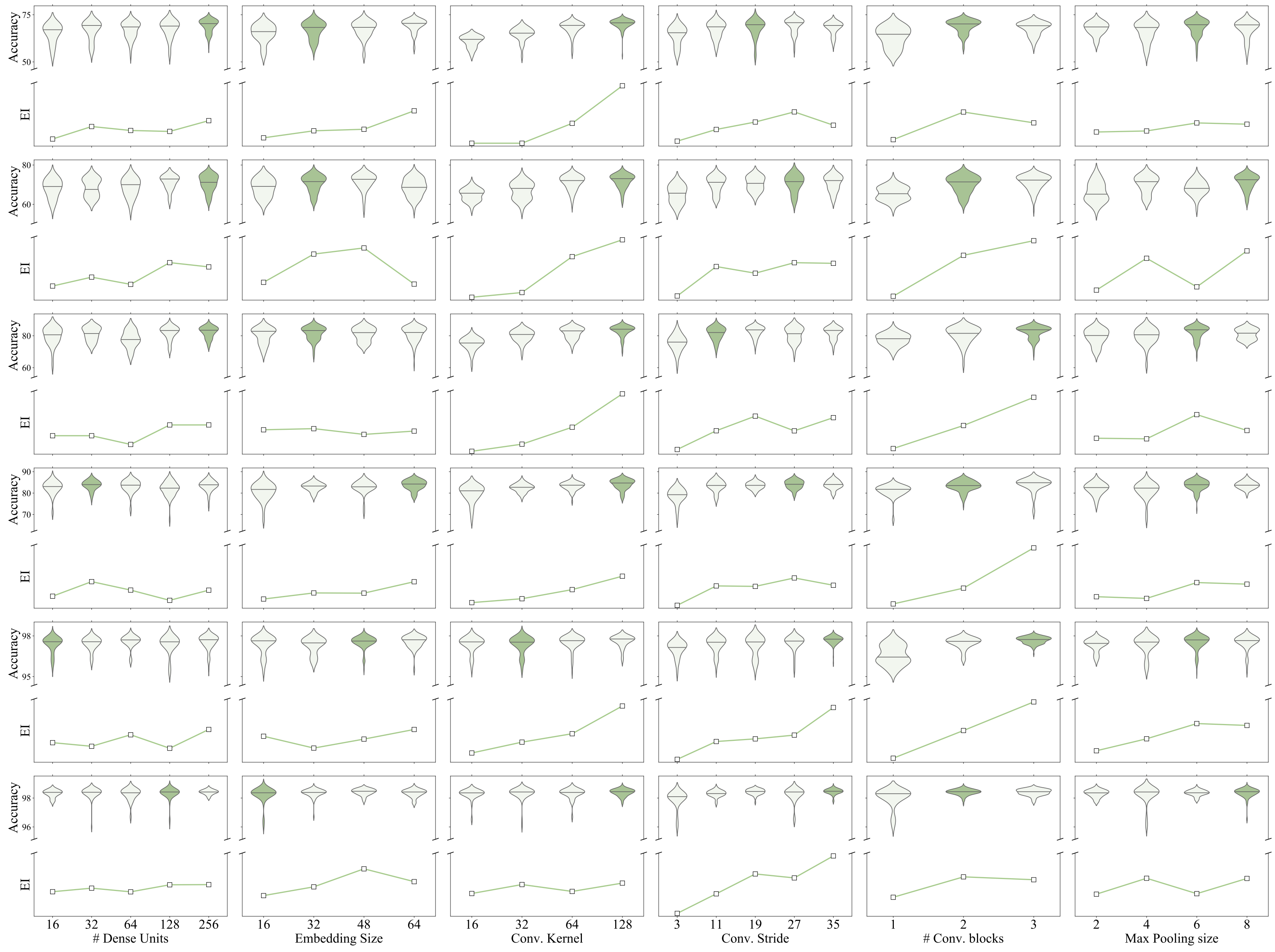


Fig. 2.   Impact of hyper-parameter variation on the validation accuracy for all scenarios with 4096-byte inputs: (top) violin plots showing distribution of accuracy for sampled networks - dark color corrosponds to the TPE chosen value; (bottom) *expected improvement* estimates from TPE.



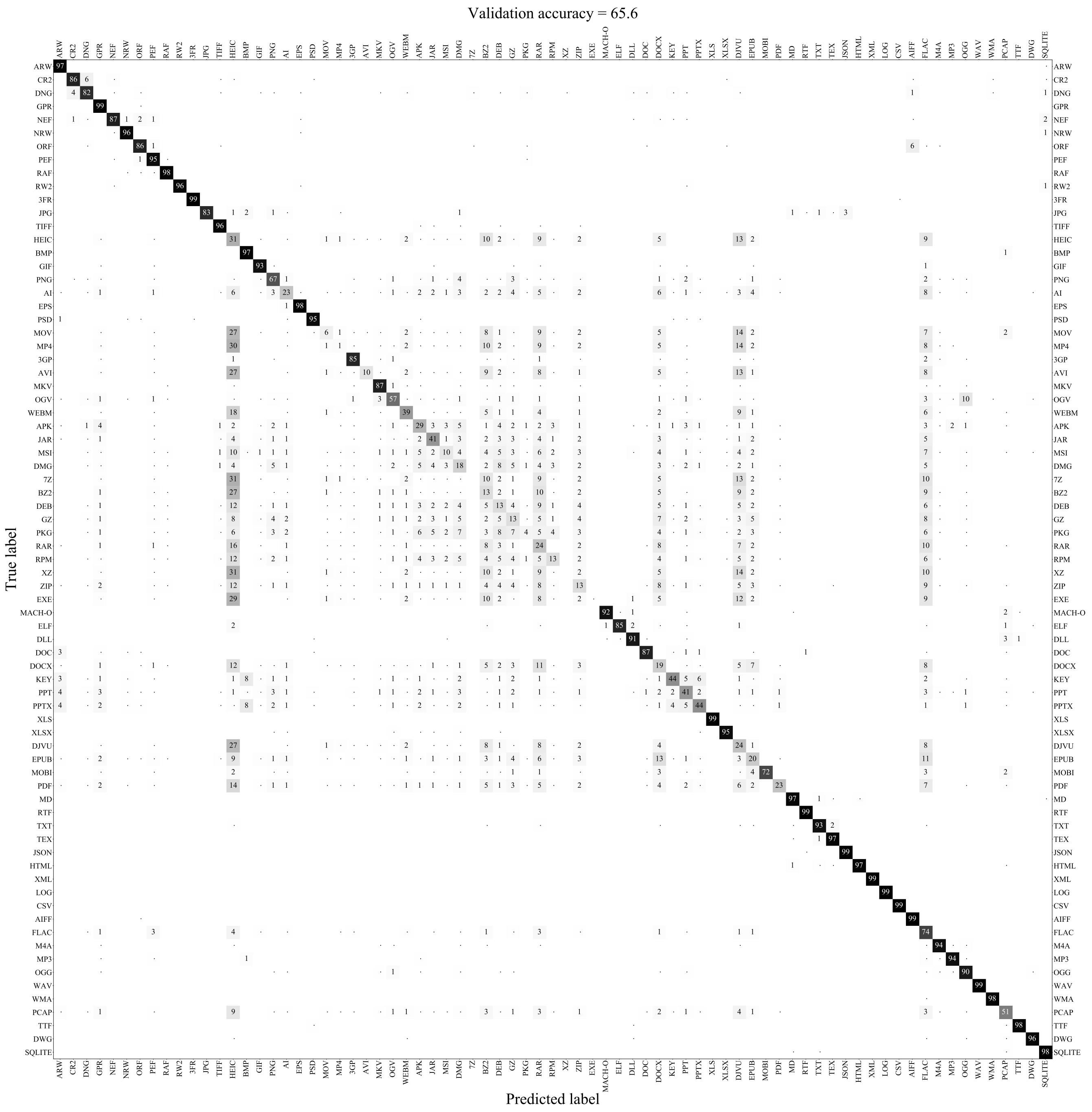


Fig. 3. Confusion matrix for *FiFTy* with block size of 512 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

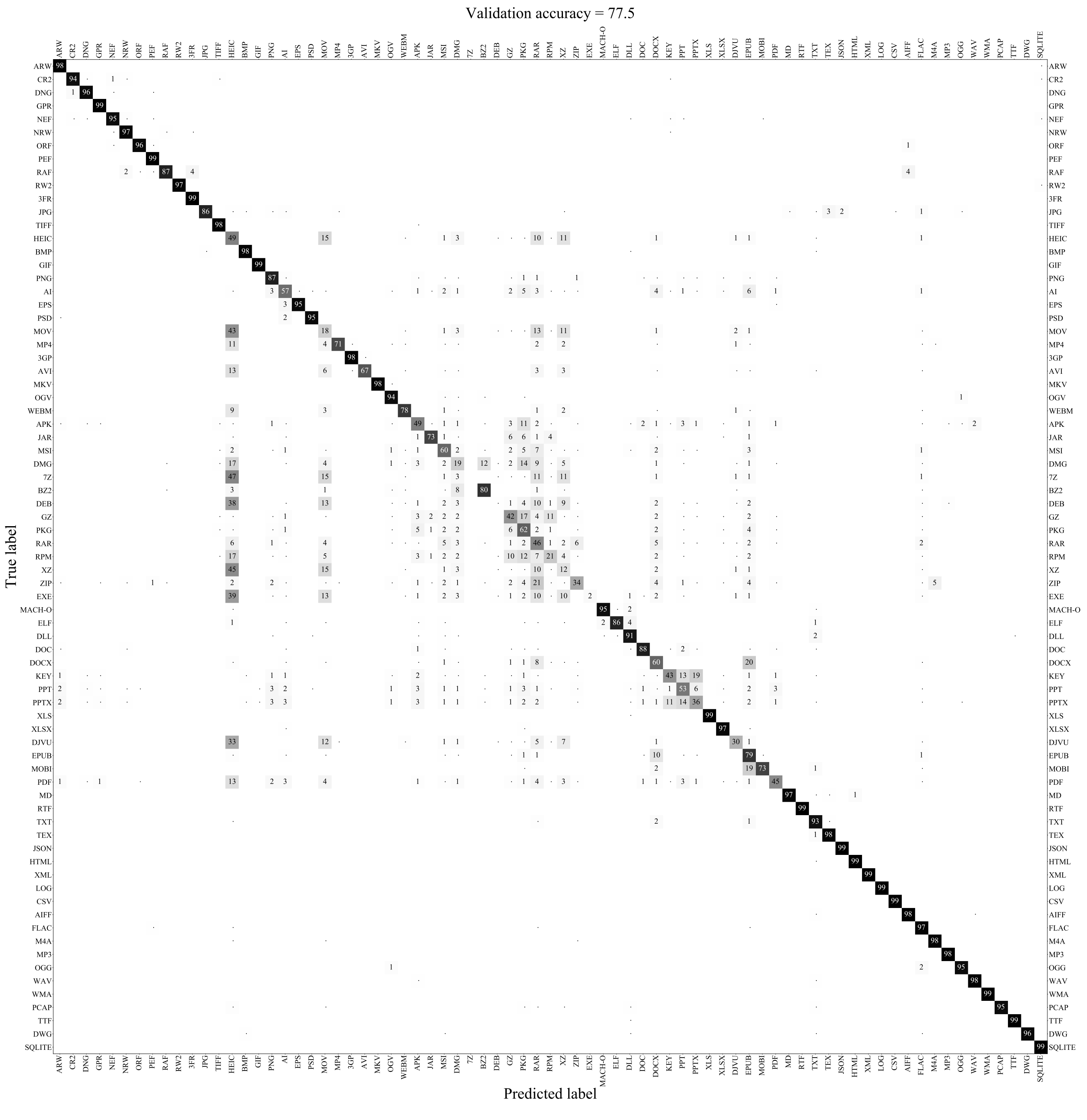


Fig. 4. Confusion matrix for *FiFTy* with block size of 4096 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

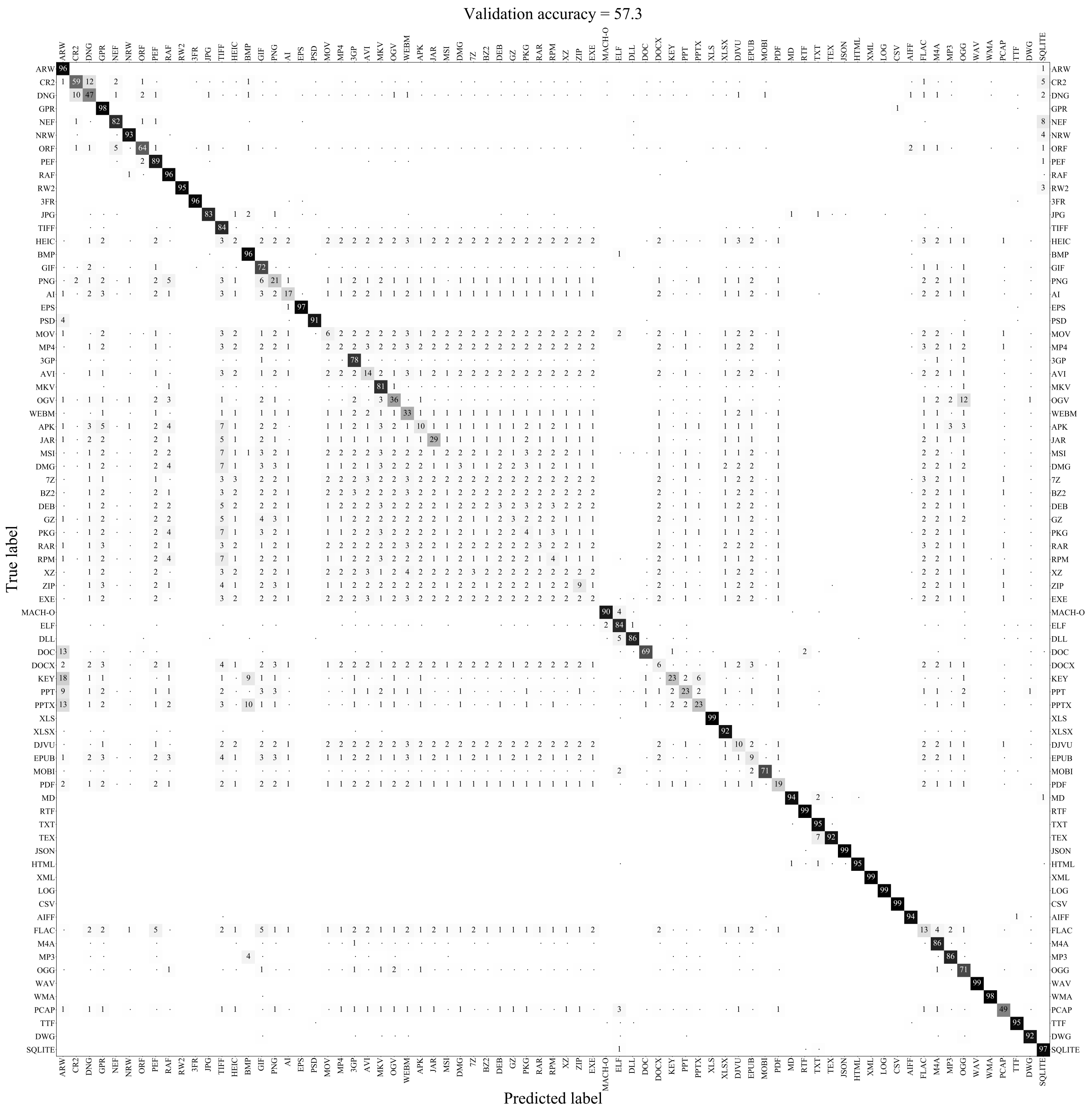


Fig. 5. Confusion matrix for Sceadan with block size of 512 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

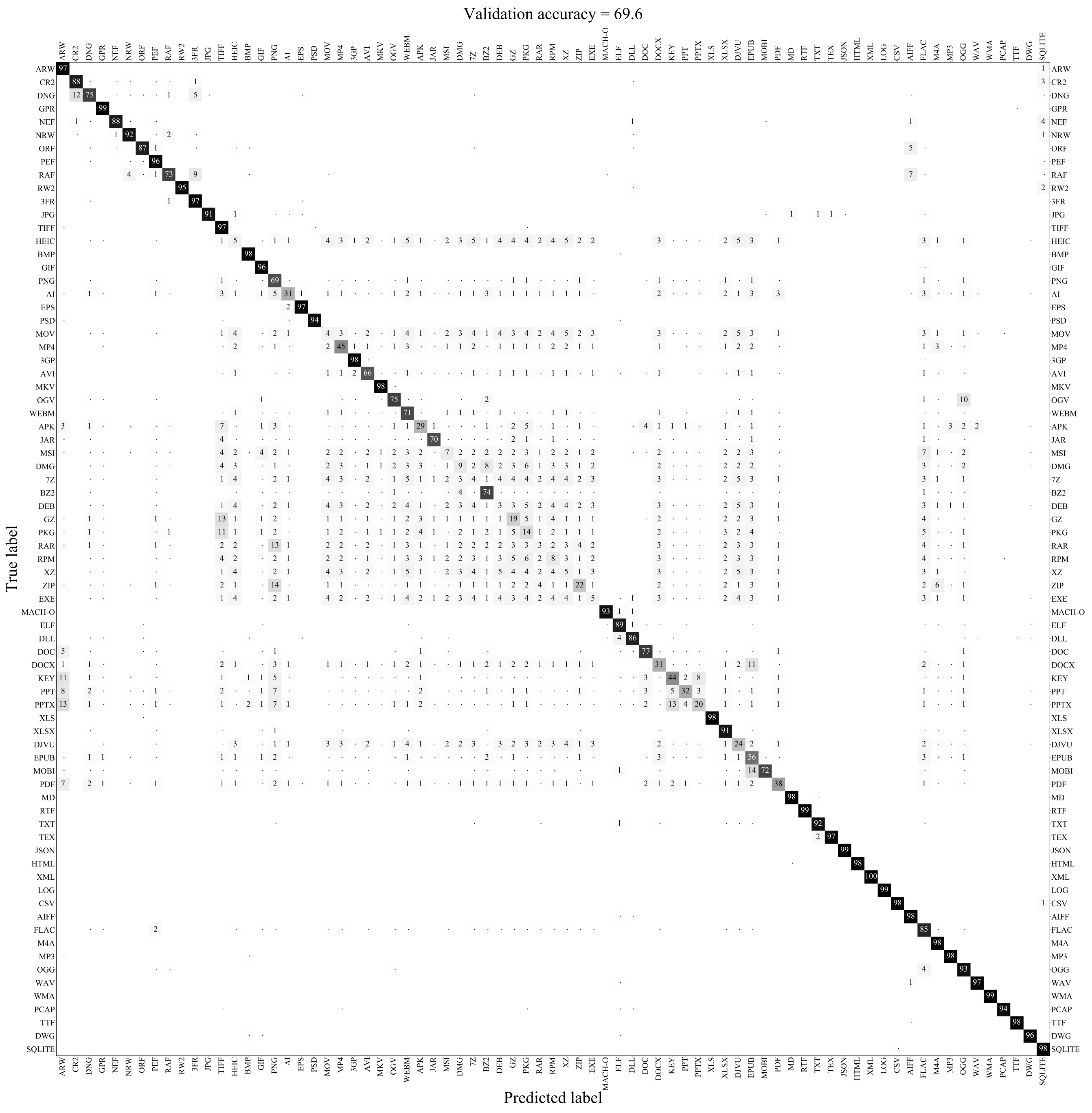


Fig. 6. Confusion matrix for Sceadan with block size of 4096 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

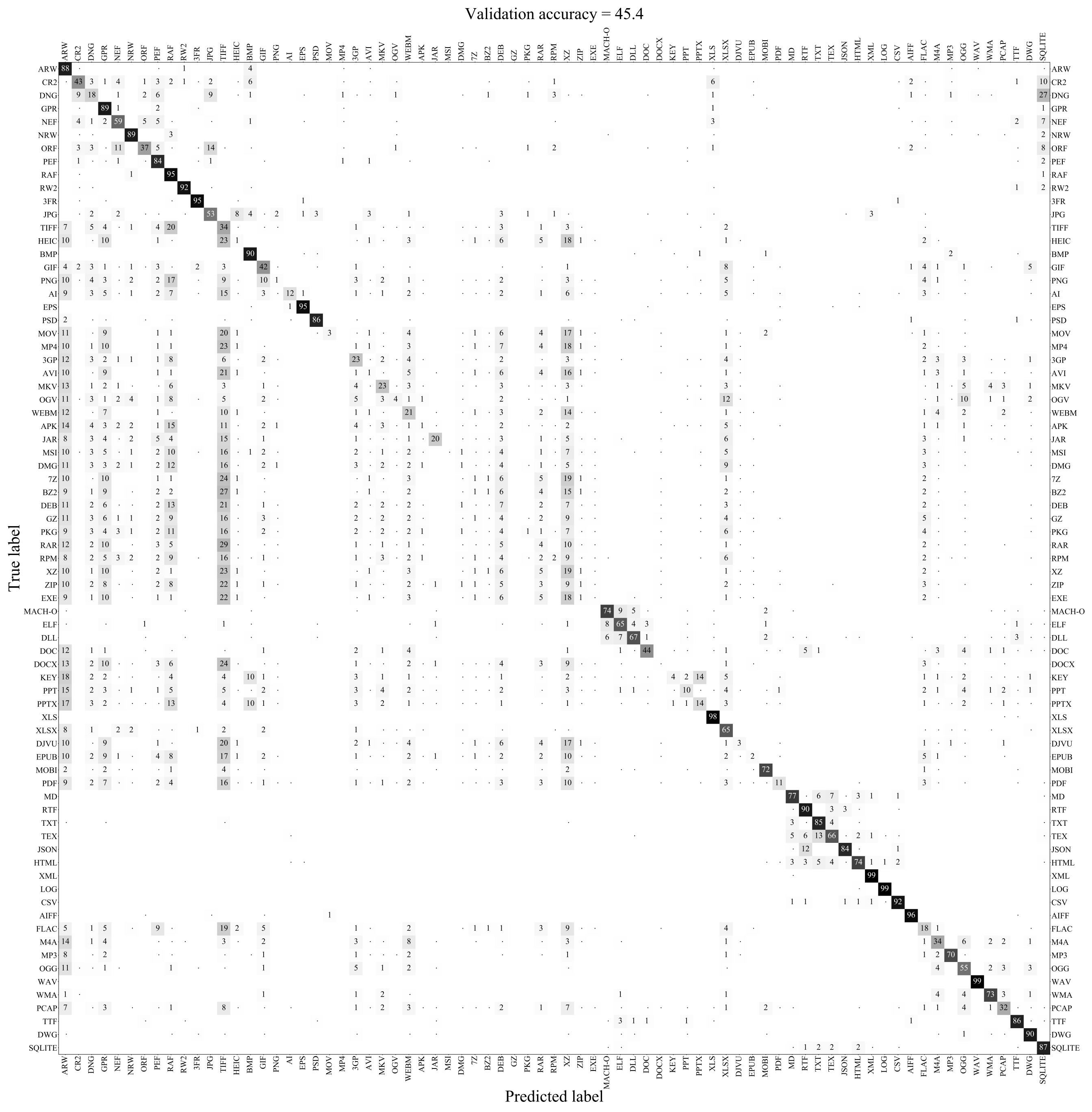


Fig. 7. Confusion matrix for baseline NN-GF with block size of 512 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

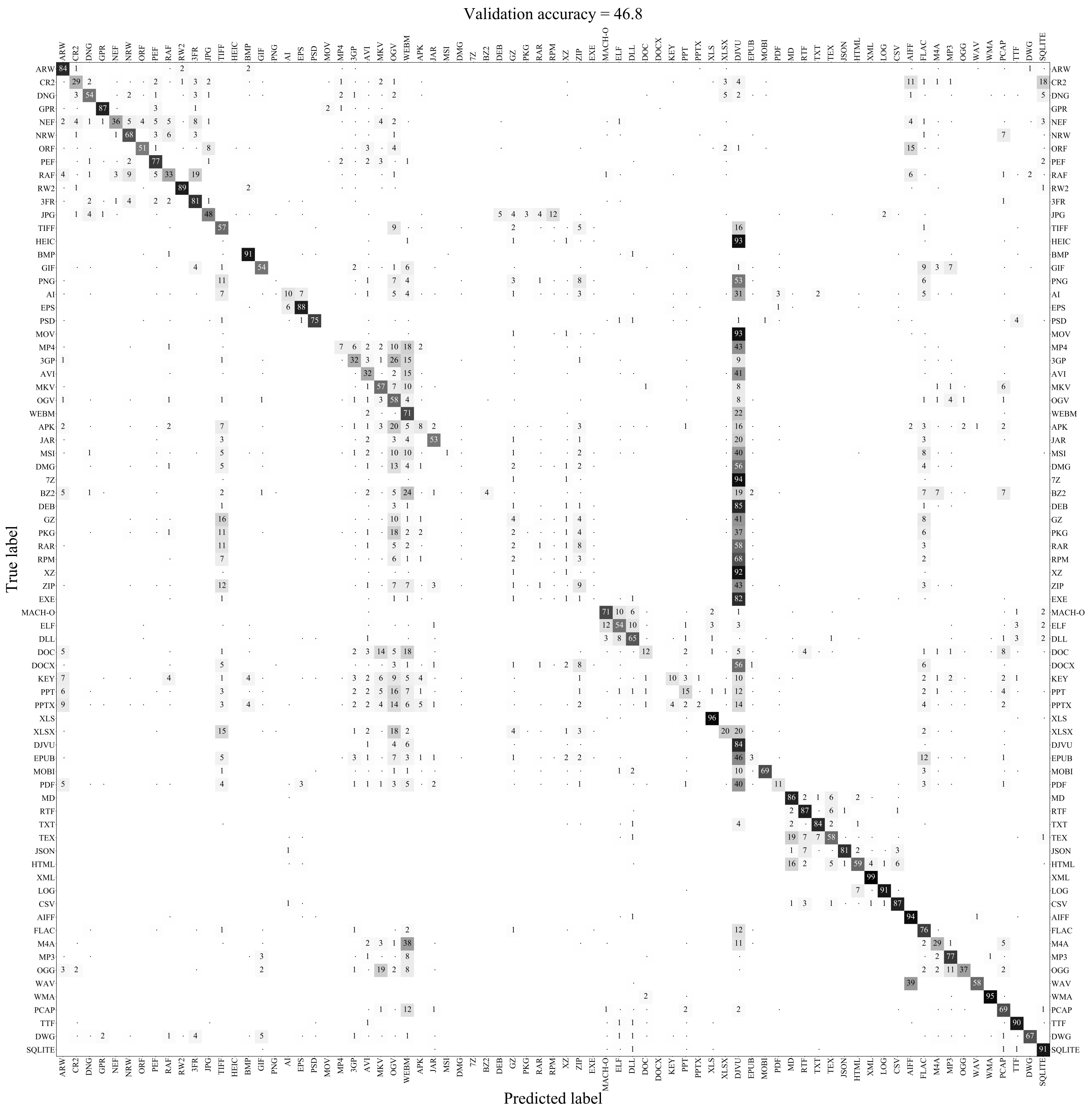


Fig. 8. Confusion matrix for baseline NN-GF with block size of 4096 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

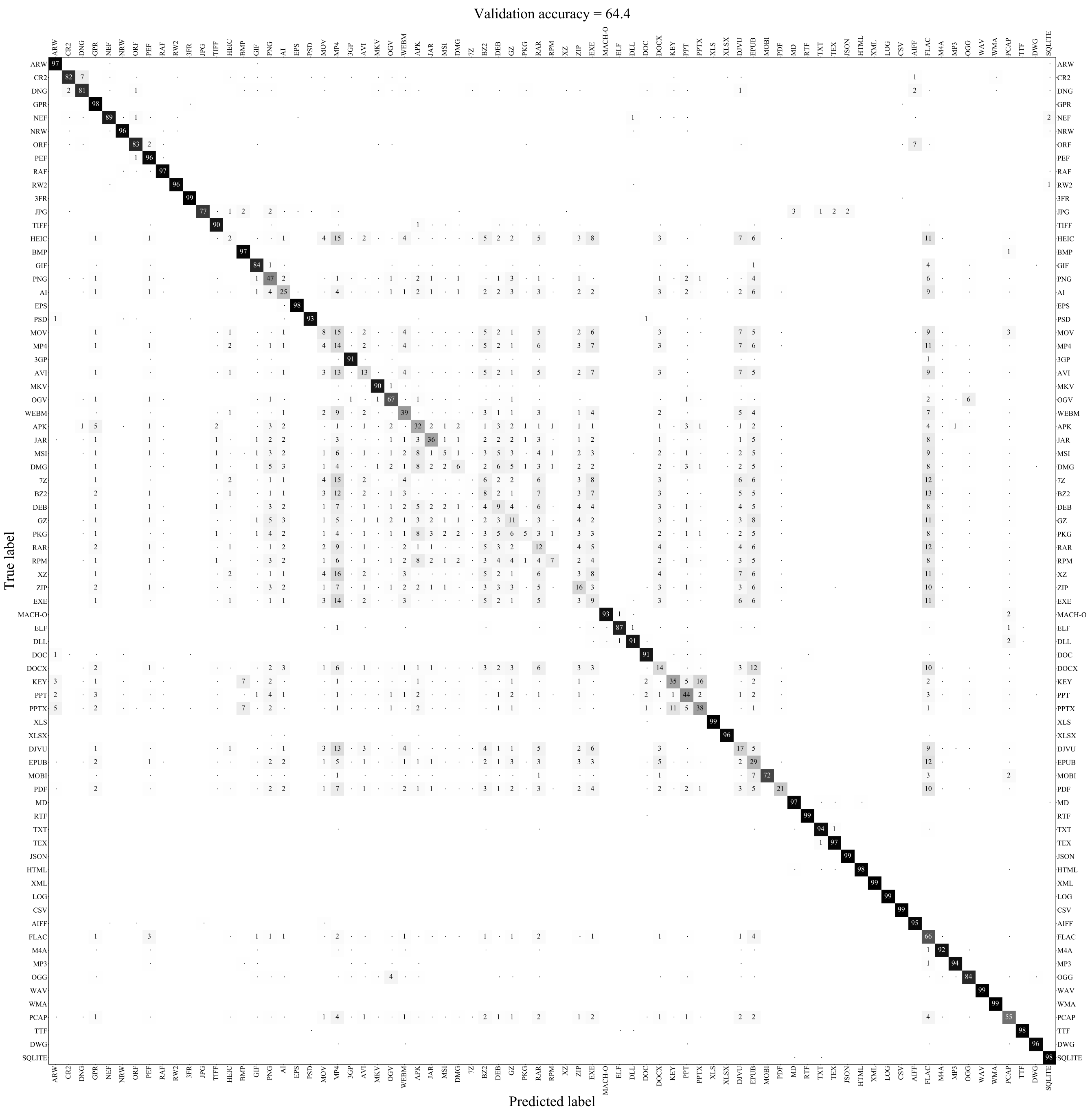


Fig. 9. Confusion matrix for baseline NN-CO with block size of 512 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.

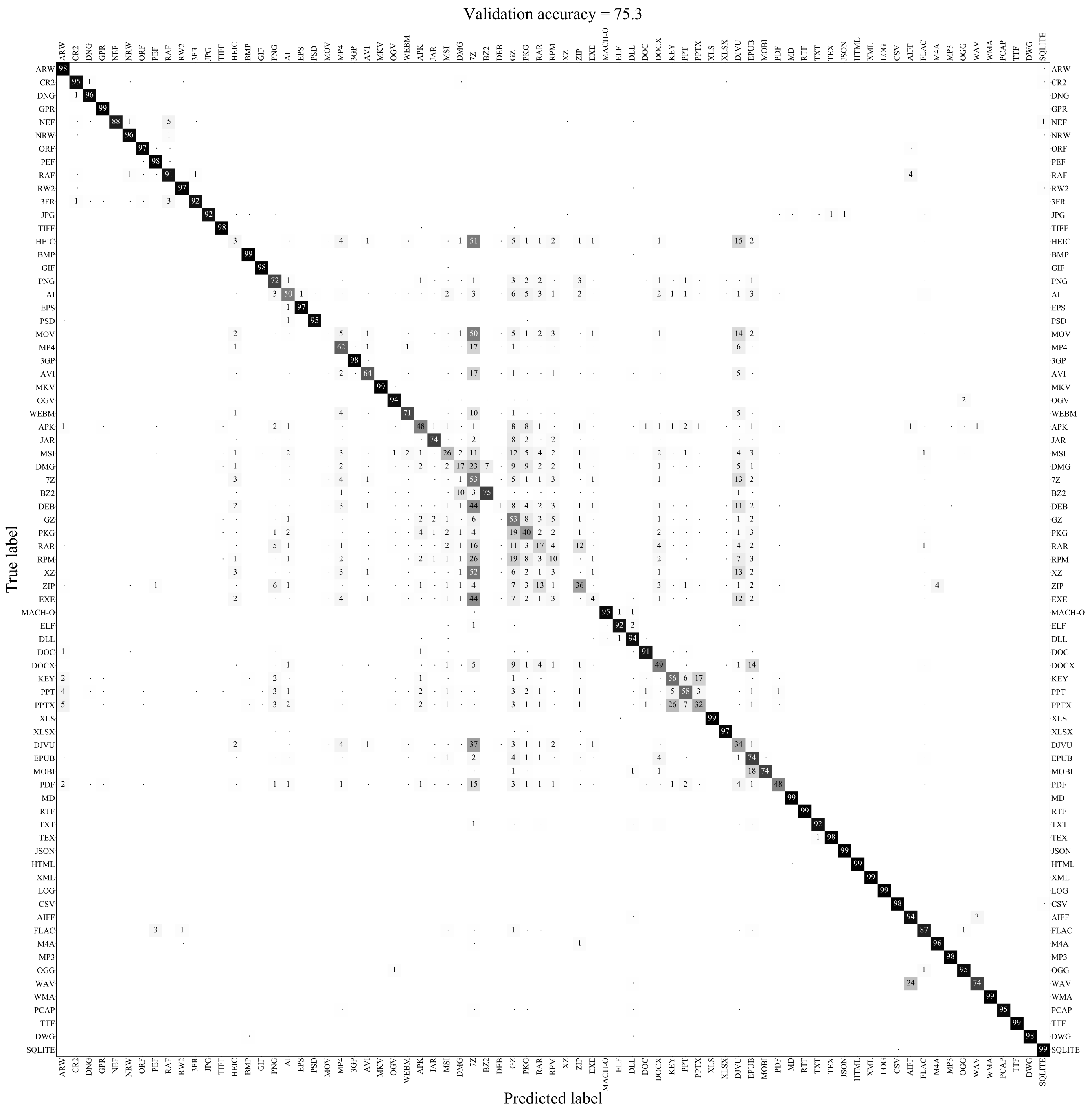
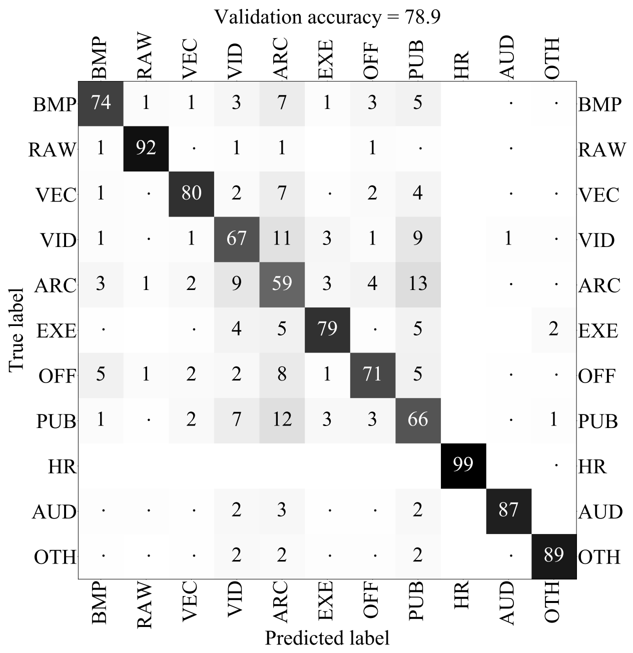
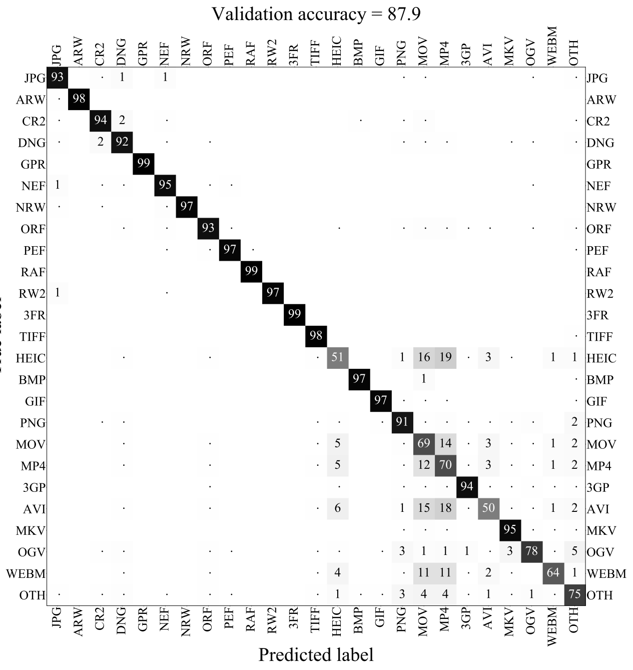


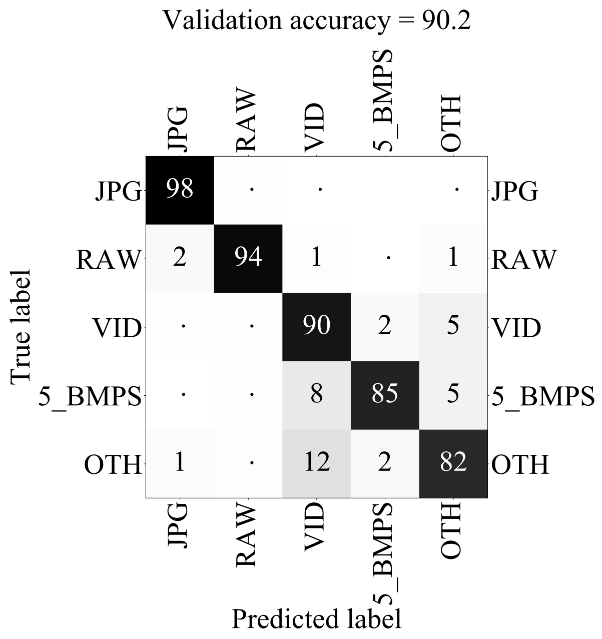
Fig. 10. Confusion matrix for baseline NN-CO with block size of 4096 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.



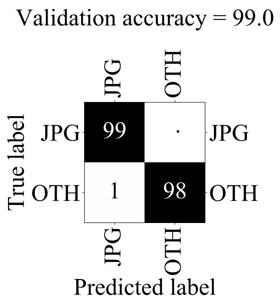
(a) Scenario #2



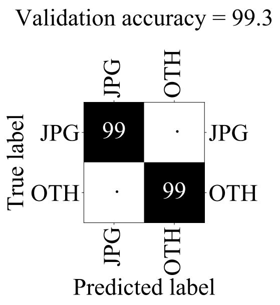
(b) Scenario #3



(c) Scenario #4

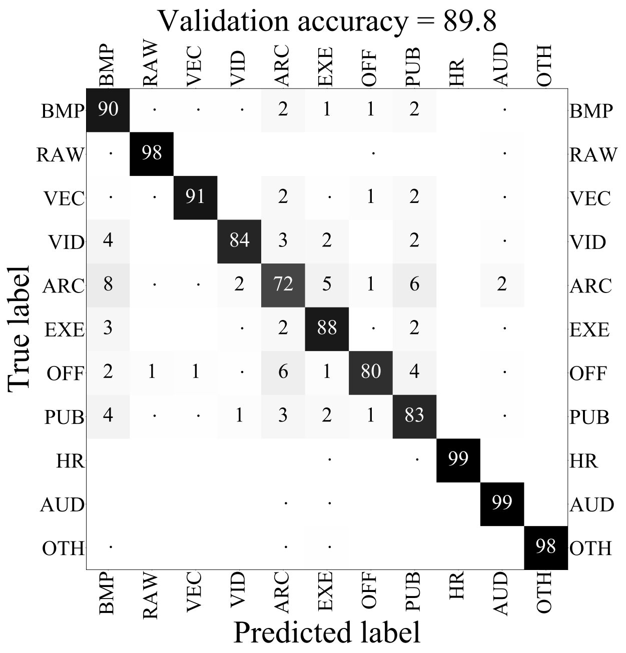


(d) Scenario #5

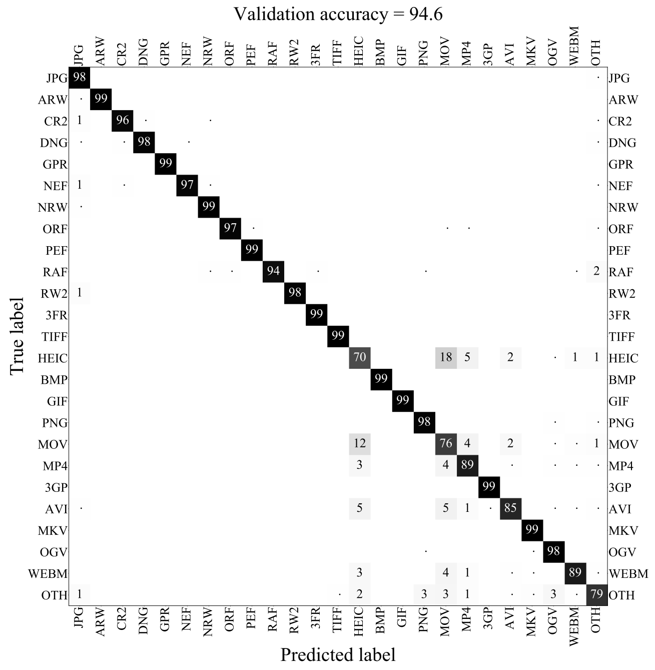


(e) Scenario #6

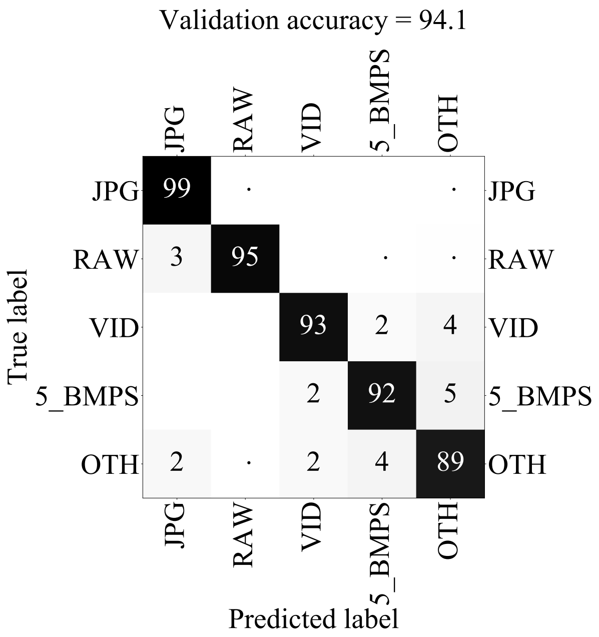
Fig. 11. Confusion matrices of the remaining five scenarios with block size of 512 bytes. Darker color means higher value and dot (.) means a non-zero value greater than 0.2%.



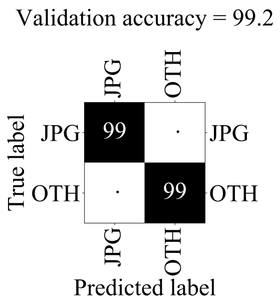
(a) Scenario #2



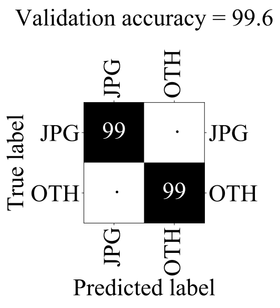
(b) Scenario #3



(c) Scenario #4



(d) Scenario #5



(e) Scenario #6

Fig. 12. Confusion matrices of the remaining five scenarios with block size of 4096 bytes. Darker color means higher value dot (.) means a non-zero value greater than 0.2%.

TABLE I
File Types Studied along with their description and sources[1].

Filetype	Description	Source(s)
JPG	Joint Photographers Experts Group (JPEG)	dpreview.com, self-shot
ARW	Raw Sony camera images	wesaturate.com
CR2	Raw Canon camera images	wesaturate.com
DNG	Raw Adobe camera images	wesaturate.com
GPR	Raw GoPro camera images	self-shot with GoPro Hero 6 Black
NEF	Raw Nikon camera images	loki.disi.unitn.it/raise
NRW	Raw Nikon camera images	photographyblog.com
ORF	Raw Olympus camera images	rawsamples.ch/index.php
PEF	Raw Pentax camera images	dpreview.com
RAF	Raw Fuji camera images	rawsamples.ch/index.php
RW2	Raw Panasonic camera images	rawsamples.ch/index.php
3FR	Raw Hasselblad camera images	hasselblad.com
TIFF	Tagged Image File Format	rawsamples.ch/index.php
HEIC	High Efficiency Image Format based on video frames	Encoded from JPG images from Nokiatech
BMP	Bitmap images	3axis.co/free-bmp-files
GIF	Graphic Interchange Format	commons.wikimedia.org
PNG	Portable Network Graphics	commons.wikimedia.org, archive.org
AI	Adobe Illustrator vector image	3axis.co/free-ai-files
EPS	Encapsulated PostScript vector	free-vectors.com
PSD	Photoshop vector file	livven.me/psds/
MOV	QuickTime File Format	self-shot98
MP4	MPEG-4 Part 14	archive.org, self-shot
3GP	Multimedia container videos format	archive.org
AVI	Audio Video Interleave container format	archive.org
MKV	Matroska Multimedia Container	archive.org
OGV	Ogg Vorbis video encoding format	commons.wikimedia.org
WEBM	Web videos	commons.wikimedia.org
APK	Android application package	archive.org
EXE	Windows executable	installers taken from open source developers' website
JAR	Java class package (compiled)	jar-download.com
MSI	Windows Installer	installers taken from open source developers' website
DMG	macOS application package	installers taken from open source developers' website
7Z	7-zip archive	rawsamples.ch
BZ2	Burrows–Wheeler archive	installers taken from open source developers' website
DEB	Linux/Unix application package	installers taken from open source developers' website
GZ	GNU Gzip	installers taken from open source developers' website
PKG	macOS compressed installer	installers taken from open source developers' website
TTF	True-type font	dafont.com

[1]All the links in the table have been accessed on or before 31st March, 2019.

Filetype	Description	Source(s)
RAR	Roshal Archive by Microsoft	archive.org
RPM	RPM package manager (Red Hat)	installers taken from open source developers' website
XZ	XZ (GNU LGPL/GPL)	installers taken from open source developers' website
ZIP	ZIP archive	self-compressed
DJVU	Digital Document Format by Yann LeCun	commons.wikimedia.org
EPUB	Electronic Publication for iBooks	gutenberg.org
MOBI	Kindle E-book	gutenberg.org
DOC	Microsoft Office (2007) Word	converted from .epub files
DOCX	Microsoft Office (2013) Word	gutenberg.org
MD	Markdown	converted from HTML, github repositories
RTF	Rich text format	converted from .txt files
TXT	Text file	gutenberg.org
PDF	Portable Document Format	arXiv.org, gutenberg.org, commons.wikimedia.org
KEY	macOS keynote presentation	converted from .pptx files
PPT	Microsoft Office (2007) Powerpoint	kmworld.com
PPTX	Microsoft Office (2013) Powerpoint	pptx-templates.com, kmworld.com
LOG	Log files	self syslogs
JSON	JavaScript Object Notation for database	kaggle.com, github.com
DWG	CAD drawing	dwgmodels.com
SQLITE	SQL database	kaggle.com
CSV	Comma-separated values	kaggle.com, who.int
XLS	Microsoft Office (2007) Excel	converted from .csv files
XLSX	Microsoft Office (2013) Excel	converted from .csv files
AIFF	Audio Interchange File Format	Converted from .flac files
FLAC	Free Lossless Audio Codec	commons.wikimedia.org, converted from .wav files
M4A	Audio-only MPEG-4	freemusicarchive.org
MP3	MPEG-1/2 Audio Layer III	freemusicarchive.org
OGG	Audio container format developed by Xiph.Org	commons.wikimedia.org
WAV	Waveform Audio File format	commons.wikimedia.org
WMA	Windows Media Audio developed by Microsoft	converted from .wav files
HTML	HyperText Markup Language	waybackmachine.org
XML	Extensible Markup Language	gutenberg.org
PCAP	Wireshark captured network packets	Capture-the-flag events from github.com
DLL	Dynamic Link Library (Windows Executable)	dlldump.com
ELF	Linux executable	Capture-the-flag events from github.com
MACH-O	macOS executable	Anaconda virtual env.
TEX	$\LaTeX$	arXiv.org