Multimedia Tools and Applications (2022) 81:30997–31015
https://doi.org/10.1007/s11042-022-12615-7

# S-DCNN: stacked deep convolutional neural networks for malware classification

**Anil Singh Parihar**[1] · **Shashank Kumar**[1] · **Savya Khosla**[1]

Received: 24 June 2021 / Revised: 5 November 2021 / Accepted: 9 February 2022 /
Published online: 8 April 2022


**Abstract**
Malware classification continues to be exceedingly difficult due to the exponential growth in the number and variants of malicious files. It is crucial to classify malicious files based on their intent, activity, and threat to have a robust malware protection and post-attack recovery system in place. This paper proposes a novel deep learning-based model, S-DCNN, to classify malware binary files into their respective malware families efficiently. S-DCNN uses the image-based representation of the malware binaries and leverages the concepts of transfer learning and ensemble learning. The model incorporates three deep convolutional neural networks, namely ResNet50, Xception, and EfficientNet-B4. The ensemble technique is used to combine these component models' predictions and a multilayered perceptron is used as a meta classifier. The ensemble technique fuses the diverse knowledge of the component models, resulting in high generalizability and low variance of the S-DCNN. Further, it eliminates the use of feature engineering, reverse engineering, disassembly, and other domain-specific techniques earlier used for malware classification. To establish S-DCNN's robustness and generalizability, the performance of proposed model is evaluated on the Malimg dataset, a dataset collected from VirusShare, and packed malware dataset counterparts of both Malimg and VirusShare datasets. The proposed method achieves a state-of-the-art 10-fold accuracy of 99.43% on the Malimg dataset and an accuracy of 99.65% on the VirusShare dataset.

**Keywords** Deep convolutional neural networks · Ensemble model · Malware classification · Pattern recognition · Security · Transfer learning

## 1 Introduction

The software that aims to harm a system without the user's knowledge or consent is malicious software (abbreviated as malware). Malware typically performs harmful actions that violate the system's confidentiality, integrity, and availability. Such harmful actions make

✉ Anil Singh Parihar
parihar.anil@gmail.com

[1] Machine Learning Research Lab, Department of Computer Science & Engineering, Delhi Technological University, Delhi, India

the system prone to unauthorized access and modification of sensitive information. The ongoing research in this field revealed the staggering growth rate of malware (i.e., malicious) attacks and the pressing need to alleviate them. For instance, McAfee Labs reported [2] that in the first quarter of 2019, Ransomware attacks increased by 118%, and new classes of the same are also identified. Similarly, Kaspersky Security Network (KSN) published a report [16] showing that malware attacked more than 70% of the users, which is devised to collect users' data without their knowledge. With the increasing dependency of individuals, institutions, and organizations on computers and databases for their several services, such as the ability to store important and, perhaps, sensitive information, there is an evident need to restrain the ever-increasing rate of malware attacks.

In literature, Signature-based methods and Heuristic-based methods are two conventional techniques for malware classification. The signature-based malware classification method compares the suspected file with the database (a blocklist) containing the signatures of already known malicious software. It detects and classifies the dubious file if a minimum of 8-bit code pattern matches the signatures database. However, this method poses a critical drawback. Since the signature database is extracted from already known malicious software, all the signatures are well-known to the malware authors. Thus, malware authors can easily elude this technique by using basic obfuscation methods. Another drawback lies in its inability to classify malware files that are variants of already known malware. Hence, this anti-malware approach proves futile and limiting when dealing with new and updated malware programs. Heuristic-based methods are dependent on comparing suspected malware's code with already known malware present in the heuristic database. The system marks the suspected file as a potential threat if the comparison is similar to the database. However, the size of the required database is increasing exponentially with the increasing number of malicious software.

As the dependency on technology is increasing, thus malware analysis becomes essential. Malware analysis involves learning the source, operation, intent, and likely outcomes of malicious software. Static analysis and dynamic analysis are two techniques for malware analysis. The static analysis is a signature-based method that analyses the suspicious file by dismantling it without any execution. The static analysis method performs reverse-engineering on an executable version of the malware to gather the human interpretable code. The static analysis method carefully investigates the reverse-engineered code to analyze the program's intent. The investigation in static analysis is carried out through several techniques involving fingerprints, debugging, memory dumping, and many others. On the other hand, dynamic analysis is a behavior-based method that executes the malware in a safe, isolated and controlled setting. The dynamic analysis executes the suspicious file and observes the execution for any suspicious activity. Since this risks harming the system, execution occurs in an isolated virtual environment, such as a sandbox. A debugger is used to ensure a proper understanding of the purpose and function of the suspicious file. However, malware authors can elude these techniques with simple obfuscation techniques.

In the literature, researchers proposed various algorithms for the detection and classification of malware. A set of algorithms [27, 29] depend on extensive feature engineering to build a database of malware features. However, as the adversaries generate new malware every day, updating the database containing the features with the samples of new malware files becomes tough. Further, traditional machine learning methods perform well in a small domain [21, 22] where the dataset is small, and the target classes are relatively smaller. However, as the complexity increases and the number of target classes grows, machine learning (ML) methods show their limited performance. The traditional ML methods require more time to be trained on large datasets and lack the ability to learn the highly

multidimensional features required for accurate classification. Hence, the traditional methods become cost-ineffective. Therefore, it is not practical to use traditional approaches to malware classification in the real-world setup. Recently, researchers [8, 15, 32] proposed deep learning-based techniques for malware classification. These techniques have shown considerable improvement in malware detection and classification tasks.

In this paper, a deep learning-based method (S-DCNN) is proposed to perform the task of malware classification. The proposed S-DCNN architecture uses the image-based representation of malicious files to train a classifier for malware binaries as shown in Fig. 1. It can be noted from Fig. 1 that the proposed S-DCNN extracts image based features from the malware files. The proposed S-DCNN model uses the paradigms of transfer learning to leverage the capabilities of state-of-the-art computer vision models. By stacking sub-models having significantly different learning mechanisms, the ensemble yields a robust classifier. Further, due to the texture-based analysis of malware images, the proposed model is resilient to obfuscation techniques like section encryption and packing [30]. The main contributions of this work are as follows:

– A novel ensemble of deep convolutional neural networks is proposed for performing the task of malware classification. The proposed model is simple, robust, computationally efficient, and thus, ideal for real-world applications. Further, it does not require feature engineering and domain expertise.
– The performance of models trained on non-obfuscated malware is evaluated on obfuscated malware exhibiting various levels of obfuscation. To our knowledge CNN models are being evaluated in this setting for the first time.

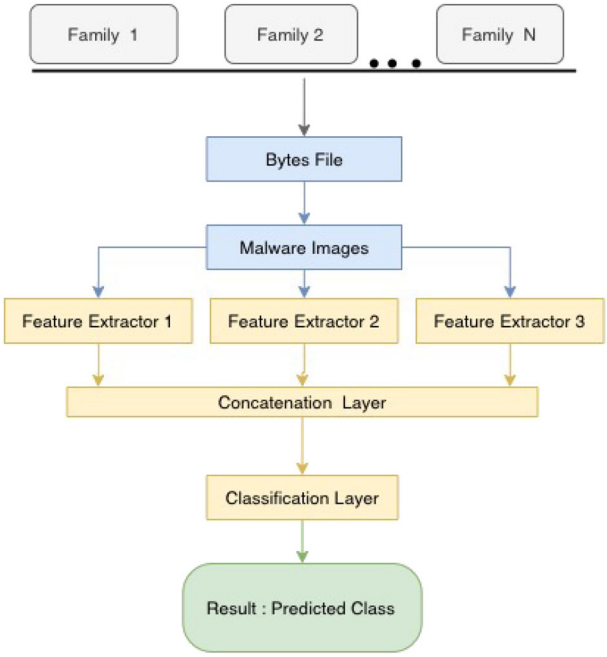


**Fig. 1** Flow-graph of proposed S-DCNN for malware classification

– A comprehensive evaluation of the proposed model is performed using several evaluation metrics and carry out extensive experimentation for model design to ensure the adequacy of the model's architecture. Based on a comparison with recent malware classification techniques, it is concluded that the proposed model exhibits superior performance.

The organization of the rest of this article is as follows. Section 2 presents a review of the recent techniques for malware classification. A detailed discussion of the proposed S-DCNN model is shown in Section 3. Section 4 demonstrates the results and experiments to establish the competence of the proposed model. Finally, the conclusion of the proposed model is shown in Section 5.

## 2 Related work

Malware detection is the task of identifying whether a given executable file is malicious or not. Once a file is identified as malicious, malware classification is performed to identify its malware family. Nowadays, machine learning approaches solve various problems like event forecasting [20], collision avoidance [37], patient outcome prediction [6] and others. Researchers [25, 29, 38, 45] proposed malware detection and classification based on machine learning techniques, namely SVM, AdaBoost, Decision Trees, and others. Schultz et al. [38] proposed an algorithm for static malware analysis based on features extracted from program executables, strings, and byte n-grams. The algorithm uses the Multinomial Naïve Bayes to classify malware files and achieves an accuracy of 97.11%. Schultz et al. used the number of unique calls to the system used within DLLs, a list of printable characters, byte sequence, and others as features for classification. Zhang et al. [45] proposed an algorithm for ransomware classification. The algorithm uses n-grams of opcodes as input for static malware analysis. Moskovitch et al. [25] proposed the use of opcode-based n-grams as features and experimented with various machine learning methods like Decision Trees, Naïve Bayes, Artificial Neural Networks, and others. Narayanan et al. [29] used PCA to extract global level features, which are used as input to the nearest neighbor classifier to classify the input files into one of 8 malware families. Angelo et al. [9] proposed association rules based malware classification. It uses subsequences of API calls and markov chain to perform malware classification. These techniques uses handcraft features for malware classification which restricts the performance of the algorithms. The proposed S-DCNN uses automatic feature extractor which helps in improving the performance.

Deep Learning (DL) has garnered significant interest in various fields [4, 17, 18, 24] over the past few years. DL methods provide solutions for most of the challenges better than the traditional methods. Jain et al. [14] analysed the performance of malware classification based on CNN and extreme learning machines. Saxe et al. [36] proposed a four-layered Neural Network for Binary Malware Classification. The method trains four fully connected layers on features extracted from the processed malware binaries. Pascanu et al. [34] developed an algorithm that uses Recurrent Neural Networks to capture the relationships across time in the event streams of API calls encoded into 114 high-level events. Alsulami et al. [1] proposed to train Convolutional Recurrent Neural Network on the Microsoft Windows prefetch files. It classifies samples that belong to rare malware families. Further, the model is capable of tuning on new malware samples. Kolosnjaji et al. [19] proposed an architecture consisting of CNN and LSTM as components. The CNN component captured the local relationships, while the LSTM layers captured the long-term dependencies. The input data

consisted of sequences of API calls made to the system. Yuan et al. [44] proposed a byte-level malware classifier using the markov images. The algorithm generates markov images from the binary files and then performs the classification of the sample. These algorithms uses the processed binary files as the features that limits the capabilities of classifiers and sometimes require larger amount of time.

Nataraj et al. [30] proposed a new type of feature extraction method. Nataraj et al. proposed transforming binary samples into grayscale images, which helps in classification using any general image classification pipeline. In another work, Nataraj et al. [31] demonstrated the superiority of binary texture analysis over other feature extraction methods for malware classification. GIST feature [33] is used for extracting a texture-based feature from the generated malware images. Nataraj et al. performed k-Nearest Neighbors-based classification on the extracted features. Naeem et al. [27] proposed an efficient framework named Malware Image Classification System for IoT infrastructure. The system first converts the incoming malware files into grayscale images and then extracts the global and local features for classification from the grayscale image. Gibert et al. [11, 12] developed a file-agnostic method for malware classification using a CNN-SVM architecture. The output features of the CNN are given as input to an SVM to classify the malware files into their respective families. Cui et al. [8] proposed the use of bat algorithm for data balancing and employed CNN-based architecture to achieve an accuracy of 94.50% on Malimg Dataset[30]. Nisa et al. [32] proposed a distinguished feature extraction method from an input image. The feature extractor [32] combines the Scale Feature Texture Analyzer (SFTA) and features from deep learning architectures (like InceptionV3 and AlexNet, pre-trained on the ImageNet dataset). The combined features are used for training of different variants of Decision Trees, SVM, KNN, and other classifiers. These methods uses machine learning based classifiers which limits the performance of the algorithm by sometimes avoiding the relationship among the features.

Vasan et al. [40] proposed IMCFN, a CNN-based model for malware classification. The method converts malware binaries into colored images and uses data augmentation techniques to fine-tune a deep convolutional neural network previously trained on the ImageNet dataset. In another work, Vasan et al. [41] proposed an ensemble of deep learning models for classifying malware files. The architecture comprises Resnet50 and VGG16 fine-tuned on the malware images. The three one-vs-all SVMs trained on the features extracted from the models. The k-fold cross-validation for the model is not given to support the generalizability. Saadat et al. [35] proposed a CNN-XGBoost model that uses CNN for feature extraction and XGBoost for classification. Saadat et al. demonstrated that CNN-XGBoost architecture performs better than CNN-SVM, CNN-Softmax, and other configurations. Venkatraman et al. [42] proposed a unified hybrid deep learning and visualization technique for malware detection and classification. For classification purposes, the application of hybrid image-based approaches with deep learning architectures is investigated. Verma et al. [43] proposed a first and second-order texture statistics for multiclass malware classification. The algorithm analyses the texture of the file to classify the malware. Transfer learning enables the methods to solve the problems of other domains [3]. Gao et al. [10] use transfer learning based semi-supervised method for malware classification. Gao et al. proposed an RNN-based byte classifier and an asm classifier. Çayır et al. [5] introduced an ensemble of Capsule Networks (CapsNet) for malware classification. Treating the CapsNet as weak classifiers, an ensemble using the bootstrap aggregation is developed. The CNNs extract a 128-dimensional feature vector from the input malware images, then fed the features to the capsule network.

Most of the existing works concentrate on extracting features from the image-based representations of the malware files and then classifying them using different machine learning techniques. Many researchers used CNN-based architectures for feature extraction from malware images. The techniques like Capsule Networks, CNN-SVM, and CNN-XGBoost give modest results only. As per observation, the methods that perform very well use a combination of multiple feature extractors to generate feature representations for malware files and then use convoluted machine learning models to perform classification from these features. However, the lack of explanation behind the choice of models used for the feature extraction and the rationale behind the particular combination leaves a knowledge gap. A diverse and comprehensive set of feature representations from the input images with even a simple classifier can achieve very high accuracy. To this end, This paper presents S-DCNN, an ensemble of carefully chosen CNN-based models that generates a robust feature representation of the input so that even a simple ANN classifier can achieve state-of-the-art results using these features.

## 3 Methodology

This section contains a detailed discussion of the proposed method and architecture. Three deep convolutional networks, namely ResNet50, Xception, and EfficientNet-B4, are trained for the concerned task. Their concatenated outputs are used as features for a multilayered perceptron, which gives the final output. The proposed method is capable of tackling standard obfuscation techniques such as section encryption and packing due to the use of texture-based analysis of the malware. Moreover, since the proposed model is an ensemble of models of considerably different learning mechanisms, it forms a good representation of the malware images. Consequently, it is robust and generalizable (the ablation experiments in Section 4 further substantiate this).

The upcoming subsection explains the dataset, the sub-models, the fine-tuning process and the training of the final model.

### 3.1 Dataset

The proposed method in this paper uses the Malimg Dataset [30]. The dataset contains 9339 data points corresponding to 25 classes. These data points are image-based representations of malware binaries, and the classes are their respective malware families.

Further, the 343 samples corresponding to 5 malware families from VirusShare.com are scrapped to test the generalizability of the proposed model. Table 1 gives the collected data's statistics. To test the robustness of the proposed model against obfuscation, a packed

**Table 1** Dataset downloaded from VirusShare

| Family | Number of samples |
|---|---|
| Lolyda.BF | 100 |
| VB | 116 |
| Prepscram | 39 |
| Eksor.A | 40 |
| Wintrim.BF | 48 |

malware dataset is generated from the VirusShare dataset for testing purposes. The images are generated for the packed and unpacked dataset by combining the bits of malware binaries into 8-bit vectors and organizing these vectors into a two-dimensional array. The 2D array is then rendered as a grayscale image.

Figure 2 shows a sample of image-based representation for each malware family, and Fig. 3 shows the data distribution. The resized images of uniform shape 224 × 224 are considered for the use case.

### 3.2 Transfer learning

Transfer Learning is a learning paradigm in which a model trained for one task is used as an initialization point for a different task. Transfer learning is extremely useful as it saves training time, leads to better-performing models, and solves the issue of data deficiency. The pre-trained models on the ImageNet dataset as the starting point are used for leveraging the concept of transfer learning. Then, the fine-tuning of these models for the task of malware classification is performed. Once fine-tuned, the models act as feature extractors for the malware images. The sub-models that form a part of the ensemble architecture are ResNet50, Xception, and EfficientNet-B4.

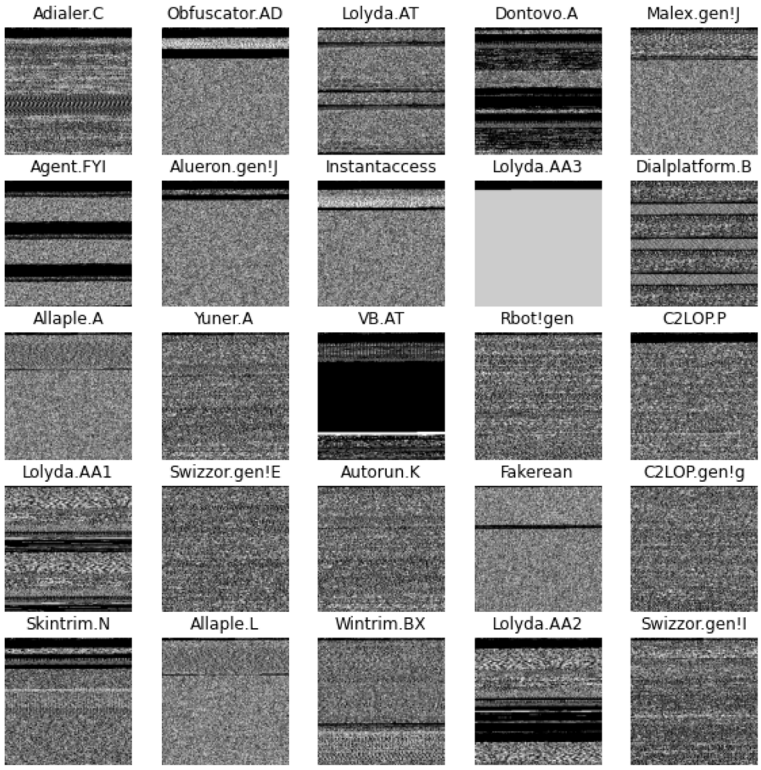


**Fig. 2** Image-based representation of malware binaries

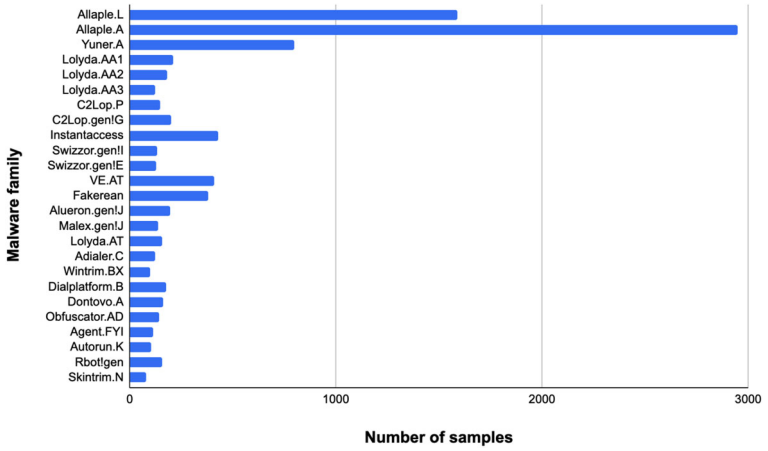


**Fig. 3** Data distribution

### 3.2.1 ResNet50

Residual Network [13], or ResNet for short, is a deep convolutional neural network that deploys a series of residual blocks containing skip connections. These skip connections address the issue of diminishing accuracy with an increasing number of layers in the model. Thus, it facilitates the formulation of profound neural networks. ResNet50 is a Residual Network having 50 layers. This model has led to a series of breakthroughs in the domain of image classification. Consequently, it is a prevalent choice for building transfer-learning-based models.

### 3.2.2 Xception

Xception [7] is a type of convolutional neural network that uses the concept of modified depthwise separable convolutions. In general, depthwise separable convolution uses depthwise convolution (channel-wise spatial convolution) followed by pointwise convolution ($1 \times 1$ convolution). The modified depthwise separable convolutions used in Xception perform these operations in reverse order; i.e., pointwise convolution is followed by depthwise convolution. These modified depthwise separable convolutions are used throughout the architecture as inception modules. Further, Xception deploys residual connections like ResNet to give an accuracy boost to the deep learning architecture.

### 3.2.3 EfficientNet-B4

Tan and Le proposed EfficientNet [39] that puts forward a systematic way of scaling up convolutional neural networks. It explores scaling across three dimensions - width (the number of channels), depth (the number of layers), and resolution (resolution of the input image). It proposes compound scaling wherein these three dimensions are systematically scaled to ensure optimal performance (accuracy) and efficiency (FLOPS). EfficientNet establishes its efficacy in the domain of transfer learning as it achieves state-of-the-art accuracy in 5 out of the eight standard transfer learning datasets. EfficientNet-B4 is used in the proposed ensemble model.

### 3.2.4 Fine-tuning

Each of the models mentioned earlier is fine-tuned to perform malware classification on the Malimg dataset. These models are initialized with the pre-trained ImageNet weights, and the top-most layer of 1000 neurons is removed. While fine-tuning, the batch normalization layers are kept non-trainable, and a dropout is applied on the final feature map of the sub-models. Finally, a fully connected layer containing 25 classes is appended to generate the final output. The customized models are then trained using the categorical cross-entropy loss given by (1).

$$L_{cce} = \sum_{i} y_i \cdot log(p_i) \tag{1}$$

Here, $L_{cce}$ represents the categorical cross-entropy loss, $y_i$ represents the true class label of $i^{th}$ data sample, and $p_i$ denotes the model's prediction for $i^{th}$ sample.

### 3.3 Ensemble learning

Figure 4 shows the complete model architecture of S-DCNN. Three models, viz. ResNet50, Xception, and EfficientNet-B4 are stacked together to form the ensemble. The feature vectors derived from these models are concatenated to form a single input feature vector. The concatenated feature vector is fed as input to a multilayered perceptron (MLP). While training the final ensemble model, the three fine-tuned sub-models act as feature extractors and are not further trained. In contrast, the MLP is trained using the categorical cross-entropy loss as given by (1).

## 4 Results and experiments

This section shows a discussion about the implementation details, results and ablation experiments.

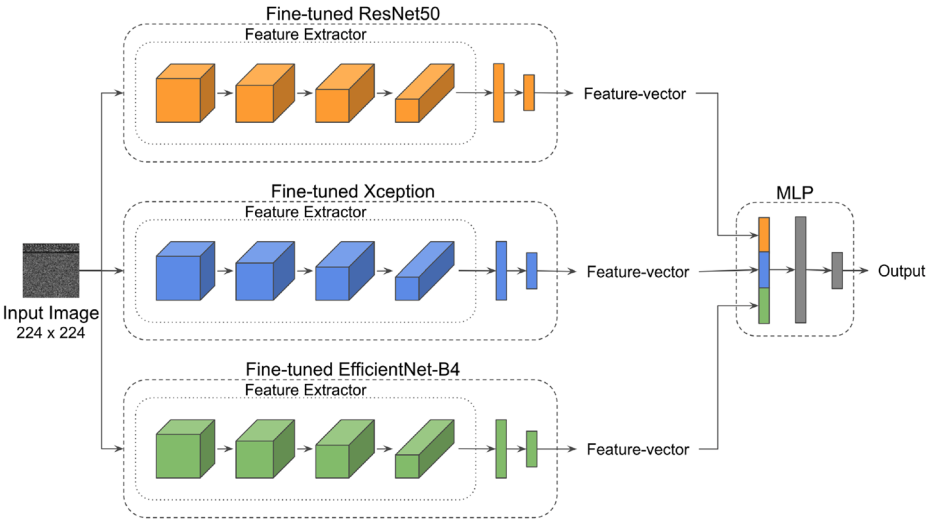


**Fig. 4** S-DCNN model architecture

### 4.1 Implementation details

#### 4.1.1 Model design and parameters

After trying out several different deep convolutional models, It is observed that the ensemble of ResNet50, Xception, and EfficientNet-B4 performed the best for our use case. The input size of the grayscale image is fixed to $224 \times 224$ for all models.

For the models mentioned above, the model uses the ImageNet weights as the starting point. The topmost fully connected layer containing 1000 neurons is removed and the rest of the model is used as a feature extractor (as highlighted in Fig. 4). After removing the top layer, the dropout is applied on the extracted output having a dropout rate of 0.1. The method flattens the feature map and appends a fully connected layer containing 25 neurons to generate the output. These models are trained using Adam optimizer. Further, the training uses a learning rate scheduler with a polynomial decay factor of 0.01 and an early stopping mechanism. The initial learning rate is 5e-5, and the batch size is 32.

The 25-dimensional outputs from the three models are concatenated for the ensemble model to form a 75-dimensional feature vector. This feature vector is used as the input to a multilayered perceptron (MLP). The hidden layer of MLP is a fully connected layer of 75 neurons and a dropout with a rate of 0.5. The output layer is a fully connected layer of 25 neurons, which generates the final output. The model uses ReLU activation on the hidden layer and SoftMax classifier on the output layer. The ensemble is trained using Adam optimizer and a batch size of 32. Again, the learning rate scheduler with a polynomial decay factor as 0.01 and an early stopping mechanism to train the MLP is used. The initial learning rate is 1e-3. A single Nvidia Tesla T4 GPU is used for all the experiments.

#### 4.1.2 Evaluation metrics

Since the size of the dataset is small and there is no pre-defined train-test split, it would be inappropriate to randomly divide the dataset into a fixed training and testing sets and then evaluate the model's performance. Thus, the stratified 10-fold cross-validation is performed to evaluate the performance of S-DCNN. The 10-fold cross-validation ensures that the proposed model is robust and has a good generalization ability.

The statistical measures used to evaluate the performance of proposed model are accuracy, precision, recall, and F1-score. Taking $TP$, $TN$, $FP$, and $FN$ as the number of true-positive, true-negative, false-positive, and false-negative predictions respectively. Equations (2), (3), (4), and (5) are the formulae of the concerned evaluation metrics:

$$Accuracy\ (A) = \frac{TP + TN}{TP + TN + FP + FN} \tag{2}$$

$$Precision\ (P) = \frac{TP}{TP + FP} \tag{3}$$

$$Recall\ (R) = \frac{TP}{TP + FN} \tag{4}$$

$$F1 - score\ (F) = \frac{2 \times P \times R}{P + R} \tag{5}$$

For 10-fold cross-validation scores, the dataset is randomly shuffled and divided into ten groups. Then the model was trained and evaluated ten times such that each group got to be the testing set once while the other nine combined to form the training set. Equation (6) gives the generic form of 10-fold evaluation metrics used to evaluate the model's performance.

$$E = \frac{1}{10} \times \sum_{i=1}^{10} E_i \tag{6}$$

Here, $E$ represents the evaluation metrics i.e., $E \in \{A, P, R, F\}$, and $E_i$ is the value of the evaluation metric for $i^{th}$ fold.

### 4.2 Results

Table 2 shows the 10-fold cross-validation results of the proposed model and the sub-models. One may observe that the ensemble of ResNet50, Xception, and EfficientNet-B4 performs better than the three models individually. For example, the ensemble performs 0.52 percentage points better in 10-fold accuracy and 0.53 percentage points better in 10-fold F1-score from the best performing sub-model. Highlighted values show the results of the proposed method.

Further, the box plot in Fig. 5 compares the distribution of accuracies achieved by S-DCNN and the constituent models in different folds. It can be observed that the ensemble model achieves very stable and high accuracy across folds while achieving a fold-wise maximum accuracy of 99.78%. The mean accuracy achieved by the S-DCNN (99.43%) is far above the maximum accuracy achieved by any base model in any fold. This observation establishes a clear advantage of the S-DCNN over Resnet-50, EfficientNet-B4, and Xception.

In literature, 10-fold accuracy is widely used as the evaluation metric for the malware classification task on the Malimg dataset. Table 3 compares the performance of S-DCNN with other recent models. One may note from the table that S-DCNN gives the best accuracy values. Further, S-DCNN demonstrates excellent efficiency with the prediction time per sample of 0.062 seconds on the NVIDIA Tesla T4 GPU. Highlighted values indicate the results of the proposed method.

Figure 6 shows the performance of S-DCNN using a confusion matrix. Due to the 10-fold cross-validation, ten confusion matrices are generated (one for every fold) and then added to give the final matrix. As can be seen, the model got a little confused between swizzor.gen!i and swizzor.gen!i. The confusion is because of a high level of similarity between samples of these classes. Further, it misclassified some samples from swizzor.gen!i class as c2lop.p and some samples of c2lop.gen!g class as c2lop.p. Other than these, the model performed well for all other classes. These results also demonstrate that S-DCNN is resilient to obfuscation. For example, despite the multilayered encryption of the code section in for allaple.l and allaple.a, the model is able to classify all samples of these

**Table 2** Classification performance of S-DCNN and the sub-models

| Model | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| ResNet50 | 98.80% | 98.87% | 98.75% | 98.81% |
| Xception | 98.91% | 98.95% | 98.85% | 98.90% |
| EfficientNet-B4 | 98.80% | 98.90% | 98.72% | 98.81% |
| **S-DCNN** | **99.43%** | **99.44%** | **99.43%** | **99.43%** |

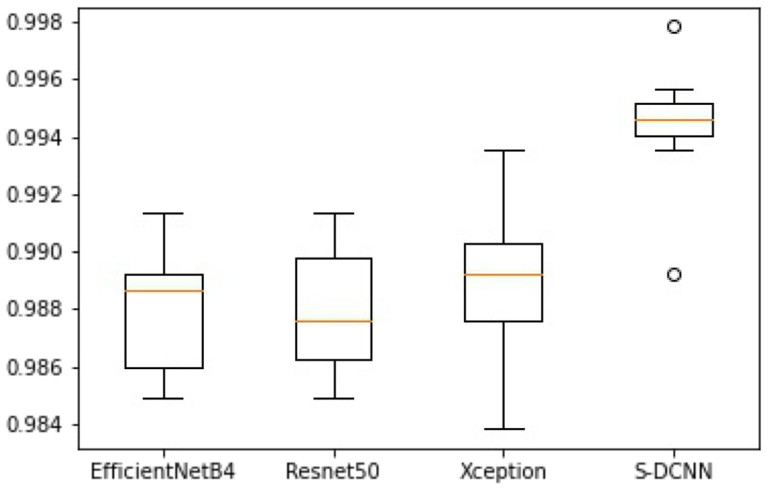


**Fig. 5** Box Plot of the sub-models and the ensemble model

classes correctly, thereby demonstrating that it can handle sectional encryption in malware files. Similarly, S-DCNN is able to classify all samples of the obfuscator.ad class correctly.

S-DCNN achieves a 3-fold cross-validation accuracy of 99.65% on the malware executables collected from VirusShare.com. Further, to demonstrate the robustness of the proposed ensemble model against obfuscation, The testing is performed on packed malware files after training them on unpacked malware files. UPX packer is used for packing the executable files collected from VirusShare.com. Table 4 shows the accuracies achieved by S-DCNN and its component models. It can be observed that S-DCNN outperforms the component models, which establishes a clear advantage. The results of the proposed method are shown in bold.

**Table 3** Comparative study of different malware classification models

| Model | Accuracy |
|---|---|
| Nataraj et al. [30] | 97.18% |
| Liu et al. [23] | 98.90% |
| Kalash et al. [15] | 98.50% |
| Naeem et al. [28] | 98.00% |
| Cui et al. [8] | 94.50% |
| Gibert et al. [11] | 98.48% |
| Naeem et al. [26] | 98.18% |
| Nisa et al. [32] | 99.30% |
| Yuan et al. [44] | 97.30% |
| Verma et al. [43] | 98.97% |
| **S-DCNN** | **99.43%** |

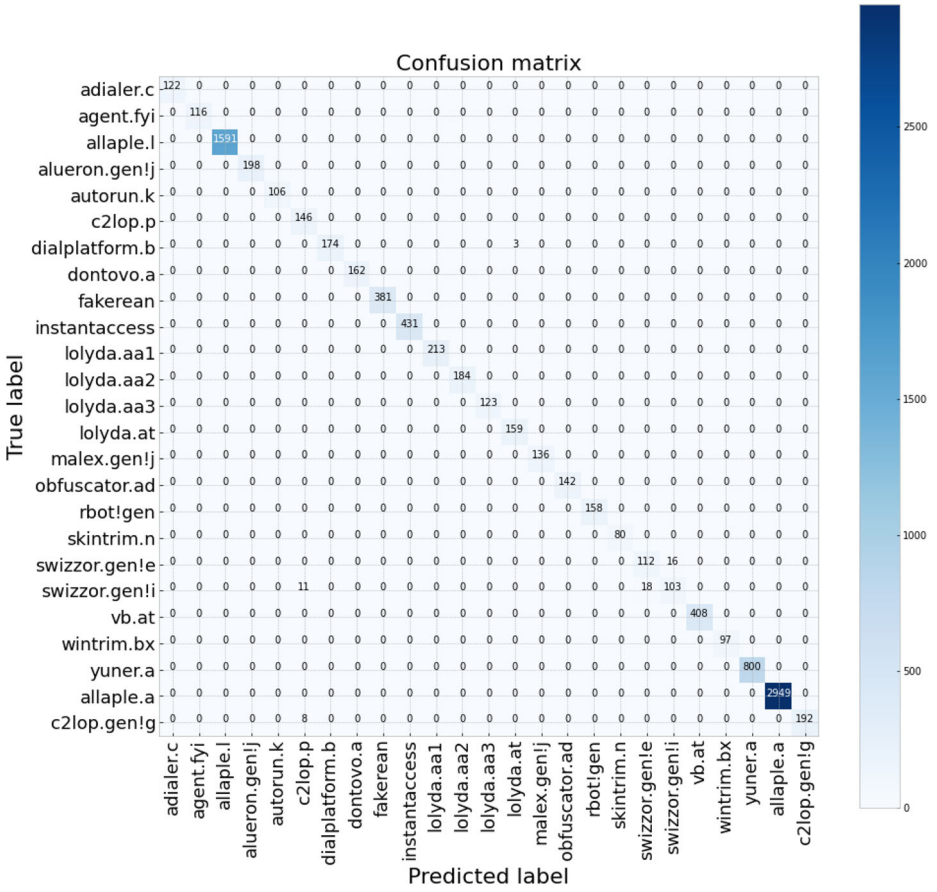


**Fig. 6** Confusion Matrix for S-DCNN

## 4.3 Ablation experiments

### 4.3.1 Model selection

In computer vision, some of the most recent advances are made by leveraging the ideas of residual connections, inception modules, and developing efficiently scaled model architectures. It is hypothesized that an ensemble comprising of all the techniques mentioned above is capable of learning robust and diverse features from any given input. Thus, the

**Table 4** Classification performance of S-DCNN on packed malware executables

| Model | Accuracy |
|---|---|
| ResNet50 | 85.40 |
| Xception | 86.30 |
| EfficientNet-B4 | 86.40 |
| **S-DCNN** | **89.70%** |

**Table 5** Performance comparison of different deep convolution networks

| Model | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| VGG16 | 98.17% | 98.17% | 98.06% | 98.11% |
| ResNet50 | 98.80% | 98.87% | 98.75% | 98.81% |
| InceptionV3 | 97.63% | 97.69% | 97.56% | 97.63% |
| Xception | 98.91% | 98.95% | 98.85% | 98.90% |
| EfficientNet-B3 | 98.74% | 98.80% | 98.69% | 98.74% |
| EfficientNet-B4 | 98.80% | 98.90% | 98.72% | 98.81% |

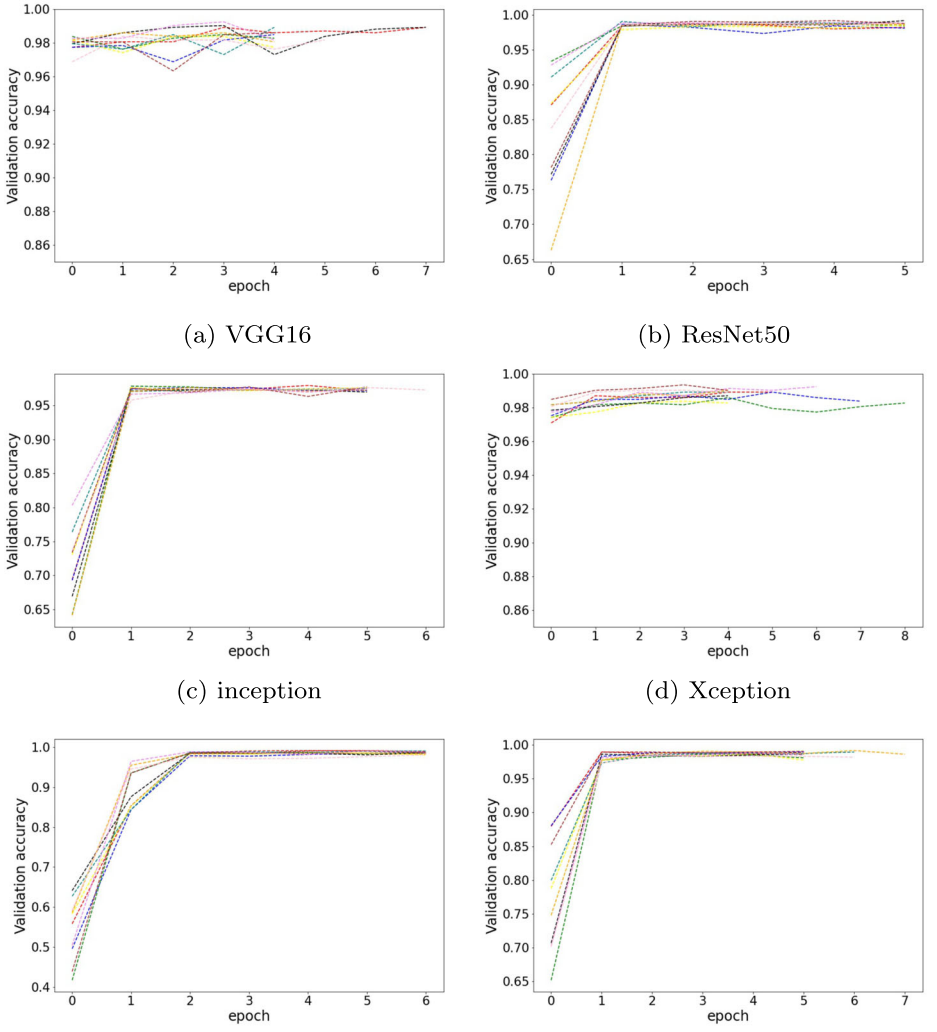


**Fig. 7** Validation accuracy vs Epochs for the 10 folds of the models

**Table 6** Performance comparison of different ensemble models

| Ensemble | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| V + R + X | 99.11% | 99.38% | 98.38% | 98.87% |
| V + R + E4 | 99.19% | 99.33% | 97.70% | 98.48% |
| V + X + E4 | 98.99% | 99.01% | 98.91% | 98.95% |
| **R + X + E4** | **99.43%** | **99.44%** | **99.43%** | **99.43%** |
| V + X + R + E4 | 99.26% | 99.42% | 98.69% | 99.05% |

experimentation with six different deep convolutional networks viz., VGG16, ResNet50, InceptionV3, Xception, EfficientNet-B3, and EfficientNet-B4 is performed to ensure that the ensemble model is built using the appropriate combination of sub-models. These particular models are considered from their respective families to ensure an appropriate trade-off between the model's performance and its inference time. Further, these models are well-understood and well-established in the field of transfer learning. Table 5 encapsulates the results for these models. As can be seen from the table, on average, Xception, ResNet50, and EfficientNet-B4 give the best results (10-fold cross-validation scores). Figure 7 shows the graph for validation accuracy vs. epochs for the 10-folds of models. The results of the proposed method are shown in bold.

Further experimentations reveal that even though the individual models achieve great results, the models individually are not capable of state-of-the-art performance. Thus, The different combinations of ensemble architectures are formulated for evaluating the performance of the possible sub-models. Since InceptionV3 and Xception are quite similar and the former's results are considerably lower than the latter model, InceptionV3 is excluded from the choice of sub-model. Similarly, EfficientNet-B3 and EfficientNet-B4 essentially belong to the same family. Thus, based on their stand-alone performance, the model selection for the ensemble is restricted to only EfficientNet-B4. Different possible combinations of ensemble architectures from VGG16, ResNet50, Xception, and EfficientNet-B4 are evaluated. These models employ very different learning techniques. Consequently, the models learn different representations of malware images, which help the final ensemble model make sound decisions. Table 6 summarizes the results of different combinations of ensemble architectures. This paper uses V, R, X, and E4 as placeholders for VGG16, ResNet50, Xception, and EfficientNet-B4, respectively. As shown in the table, the combination of ResNet50, Xception, and EfficientNet-B4 performs significantly better than the other combinations. The model configuration used in the proposed method is highlighted in bold

#### 4.3.2 Ensemble technique

Using ResNet50, Xception, and EfficientNet-B4 as sub-models, The experimentation is performed with three ensemble techniques - hard-voting ensemble, soft-voting ensemble, and stacking generalization ensemble. Table 7 encapsulates the results obtained with each of these techniques. The results of the proposed method are indicated in bold.

In a hard-voting ensemble, predictions are made using each sub-model. For each data point, each prediction is considered as a vote for the predicted class, and the class that received the maximum sum of votes is reported as the final prediction. In a soft-voting ensemble, each sub-model gives the class probability. These probabilities are added, and the class that received the maximum sum of probabilities is reported as the final prediction. A drawback of this technique is that each model contributes equally to the prediction

**Table 7** Comparative study of different ensemble techniques

| Ensemble technique | Accuracy | Precision | Recall | F1-score |
|---|---|---|---|---|
| Hard-voting | 98.92% | 97.39% | 97.13% | 97.26% |
| Soft-voting | 99.03% | 97.53% | 97.38% | 97.45% |
| **Stacked** | **99.43%** | **99.44%** | **99.43%** | **99.43%** |

irrespective of its performance. Thus, the results obtained with this technique are a little sub-optimal.

Further, a stacked generalization ensemble is used for the experiments. In this technique, a learning algorithm is used to combine the predictions of the sub-models. In particular, a multilayered perceptron (MLP) is used as a meta classifier that took the concatenated output of the sub-models as input and learned the best possible way to combine the input predictions to form the output predictions. One may notice from Table 7 that the stacked generalization ensemble yields significantly better results as compared to the hard-voting ensemble and soft-voting ensemble.

The major findings of the experimentation are as follows:

1. ResNet50, Xception, and EfficientNet-B4 as sub-models performs better feature extraction than the other deep-learning based models.
2. The multilayered perceptron based classifier shows that the light classifiers can perform the classification effectively with a robust feature extractor.
3. The performance of the sub-models without ensemble is limited. Further, the ensemble of sub-models helps in achieving the best performance.

### 4.4 Computational complexity

The analysis of computational cost for an algorithm is important. This section presents the average running time analysis to show the relative computational time required per algorithm. The analysis is performed on a system with Tesla T4 GPU. The malware file from the Malimg and VirusShare datasets are used to analyse the average running time of the proposed algorithms. The parameters in the code of various algorithms are used as given in

**Table 8** Average running time per sample (in seconds)

| Model | Run time |
|---|---|
| Nataraj et al. [30] | 0.118 |
| Liu et al. [23] | 0.097 |
| Kalash et al. [15] | 0.091 |
| Naeem et al. [28] | 0.081 |
| Cui et al. [8] | 0.078 |
| Gibert et al. [11] | 0.102 |
| Naeem et al. [26] | 0.195 |
| Nisa et al. [32] | 0.099 |
| Yuan et al. [44] | 0.103 |
| Verma et al. [43] | 0.071 |
| **S-DCNN** | **0.062** |

the papers. Table 8 shows the average computation time to process a sample malware for the algorithms. Most of the algorithms uses either CNN-based deep classifiers or complex machine learning classifiers which requires comparably larger time. The proposed S-DCNN uses a light Multilayered perceptron based classifier which improves the speed of the proposed method. It can be noticed from Table 8 that the proposed S-DCNN outperforms the other contemporary algorithms. The results of the proposed method are shown in bold.

## 5 Conclusion

This paper proposes a stacked deep convolutional model to address malware classification problem. The proposed method uses the concept of transfer learning and ensemble learning. It formulates a robust and efficient architecture for malware classification by ensemble of the sub-models. The ensemble model extracts the features from the malware file to perform the multilayered perceptron based classification. The effective ensemble enables a simple classifier to classify the samples effectively. It is demonstrated that the proposed model is able to tackle the challenges posed by the unbalanced nature of the real-world datasets and is resilient to standard obfuscation techniques. Further, it is established that the proposed model is effective with the help of a comprehensive evaluation using suitable statistical metrics, extensive result comparison with existing models, and diverse ablation experiments.

### Declarations

**Conflict of Interests** The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## References


1. Alsulami B, Mancoridis S (2018) Behavioral malware classification using convolutional recurrent neural networks. In: 2018 13th international conference on malicious and unwanted software (MALWARE), pp 103–111. https://doi.org/10.1109/MALWARE.2018.8659358
2. Beek C, Dunton T, Fokker J, Grobman S, Hux T, Polzer T, Lopez MR, Roccia T, Saavedra-Morales J, Samani R, Sherstobitof R (2019) Accessed on December 27, 2020 Mcafee labs threats report. https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-aug-2019.pdf
3. Bhowmik A, Kumar S, Bhat N (2019) Eye disease prediction from optical coherence tomography images with transfer learning. In: International conference on engineering applications of neural networks. Springer, pp 104–114
4. Bhowmik A, Kumar S, Bhat N (2021) Evolution of automatic visual description techniques-a methodological survey. Multimed Tools Appl, 1–45
5. Çayır A, Ünal U, Dağ H (2021) Random capsnet forest model for imbalanced malware type classification task. Comput Secur 102:102133. https://doi.org/10.1016/j.cose.2020.102133
6. Chaudhary P, Gupta DK, Singh S (2021) Outcome prediction of patients for different stages of sepsis using machine learning models. In: Advances in communication and computational technology. Springer, Singapore, pp 1085–1098
7. Chollet F (2017) Xception: deep learning with depthwise separable convolutions. In: 2017 IEEE Conference on computer vision and pattern recognition (CVPR), pp 1800–1807. https://doi.org/10.1109/CVPR.2017.195
8. Cui Z, Xue F, Cai X, Cao Y, ge Wang G, Chen J (2018) Detection of malicious code variants based on deep learning. IEEE Trans Industr Inform 14(7):3187–3196. https://doi.org/10.1109/tii.2018.2822680
9. D'Angelo G, Ficco M, Palmieri F (2021) Association rule-based malware classification using common subsequences of api calls. Appl Soft Comput 105:107234. https://doi.org/10.1016/j.asoc.2021.107234

10. Gao X, Hu C, Shan C, Liu B, Niu Z, Xie H (2020) Malware classification for the cloud via semi-supervised transfer learning. J Inform Secur Applic 55:102661. https://doi.org/10.1016/j.jisa.2020.102661
11. Gibert D, Mateu C, Planes J, Vicens R (2018) Using convolutional neural networks for classification of malware represented as images. J Comput Virol Hack Techniques 15(1):15–28. https://doi.org/10.1007/s11416-018-0323-0
12. Gibert D, Mateu C, Planes J (2020) Hydra: a multimodal deep learning framework for malware classification. Comput Secur 95:101873. https://doi.org/10.1016/j.cose.2020.101873
13. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on computer vision and pattern recognition (CVPR), pp 770–778. https://doi.org/10.1109/CVPR.2016.90
14. Jain M, Andreopoulos W, Stamp M (2020) Convolutional neural networks and extreme learning machines for malware classification. J Comput Virol Hack Techniques 16(3):229–244
15. Kalash M, Rochan M, Mohammed N, Bruce NDB, Wang Y, Iqbal F (2018) Malware classification with deep convolutional neural networks. In: 2018 9th IFIP international conference on new technologies, mobility and security (NTMS), pp 1–5. https://doi.org/10.1109/NTMS.2018.8328749
16. Kaspersky (2020) Accessed on December 27, 2020 Protecting your personal data online at every point. https://media.kasperskydaily.com/wp-content/uploads/sites/92/2020/01/27103216/International-Privacy-Day-2020-Kaspersky-report.pdf
17. Katyal S, Kumar S, Sakhuja R, Gupta S (2018) Object detection in foggy conditions by fusion of saliency map and yolo. In: In 2018 12th international conference on sensing technology (ICST), pp 154–159. https://doi.org/10.1109/ICSensT.2018.8603632
18. Kaur G, Singh S, Rani R, Kumar R, Malik A (2021) High-quality reversible data hiding scheme using sorting and enhanced pairwise pee. IET Image Processing. https://doi.org/10.1049/ipr2.12212
19. Kolosnjaji B, Zarras A, Webster G, Eckert C (2016) Deep learning for classification of malware system call sequences. In: AI 2016: advances in artificial intelligence. Springer International Publishing, pp 137–149. https://doi.org/10.1007/978-3-319-50127-7_11
20. Kumar M, Gupta DK, Singh S (2021) Extreme event forecasting using machine learning models. In: Advances in communication and computational technology. Springer, Singapore, pp 1503–1514
21. Kumar N, Kumar R, Malik A, Singh S (2021) Low bandwidth data hiding for multimedia systems based on bit redundancy. Multimed Tools Appl, 1–19
22. Kumar R, Chand S, Singh S (2019) An optimal high capacity reversible data hiding scheme using move to front coding for lzw codes. Multimed Tools Appl 78(16):22977–23001
23. Liu L, Wang B, Yu B, Zhong Q (2017) Automatic malware classification and new malware detection using machine learning. Front Inform Technol Electron Eng 18:1336–1347
24. Malik A, Kumar R, Singh S (2018) A new image steganography technique based on pixel intensity and similarity in secret message. In: International Conference on Advances in Computing, Communication Control and Networking (ICACCCN). IEEE, pp 828–831
25. Moskovitch R, Feher C, Tzachar N, Berger E, Gitelman M, Dolev S, Elovici Y (2008) Unknown malcode detection using OPCODE representation. In: Intelligence and security informatics. Springer, Berlin, pp 204–215. https://doi.org/10.1007/978-3-540-89900-6_21
26. Naeem H (2019) Detection of malicious activities in internet of things environment based on binary visualization and machine intelligence. Wirel Pers Commun 108(4):2609–2629. https://doi.org/10.1007/s11277-019-06540-6
27. Naeem H, Guo B, Naeem MR (2018) A light-weight malware static visual analysis for iot infrastructure. In: 2018 International conference on artificial intelligence and big data (ICAIBD), pp 240–244. https://doi.org/10.1109/ICAIBD.2018.8396202
28. Naeem H, Guo B, Naeem MR, Ullah F, Aldabbas H, Javed MS (2019) Identification of malicious code variants based on image visualization. Comput Electr Eng 76:225–237. https://doi.org/10.1016/j.compeleceng.2019.03.015
29. Narayanan BN, Djaneye-Boundjou O, Kebede TM (2016) Performance analysis of machine learning and pattern recognition algorithms for malware classification. In: 2016 IEEE national aerospace and electronics conference (NAECON) and ohio innovation summit (OIS), pp 338–342. https://doi.org/10.1109/NAECON.2016.7856826
30. Nataraj L, Karthikeyan S, Jacob G, Manjunath BS (2011) Malware images: visualization and automatic classification. In: the 8th International symposium on visualization for cyber security. Association for Computing Machinery, New York, NY, USA, VizSec '11. https://doi.org/10.1145/2016904.2016908
31. Nataraj L, Yegneswaran V, Porras P, Zhang J (2011) A comparative assessment of malware classification using binary texture analysis and dynamic analysis. In: Proceedings of the 4th ACM workshop on

security and artificial intelligence. Association for Computing Machinery, New York, NY, USA, AISec '11, pp 21–30. https://doi.org/10.1145/2046684.2046689
32. Nisa M, Shah JH, Kanwal S, Raza M, Khan MA, Damaševičius R, Blažauskas T (2020) Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features. Appl Sci 10(14):4966. https://doi.org/10.3390/app10144966
33. Oliva A, Torralba A (2001) Modeling the shape of the scene: a holistic representation of the spatial envelope. Int J Comput Vis 42(3):145–175
34. Pascanu R, Stokes JW, Sanossian H, Marinescu M, Thomas A (2015) Malware classification with recurrent networks. In: IEEE International conference on acoustics, speech and signal processing (ICASSP), pp 1916–1920. https://doi.org/10.1109/ICASSP.2015.7178304
35. Saadat S, Raymond VJ (2020) Malware classification using CNN-XGBoost model. In: Artificial intelligence techniques for advanced computing applications. Springer, Singapore, pp 191–202. https://doi.org/10.1007/978-981-15-5329-5_19
36. Saxe J, Berlin K (2015) Deep neural network based malware detection using two dimensional binary program features. In: 2015 10th international conference on malicious and unwanted software (MALWARE), pp 11–20. https://doi.org/10.1109/MALWARE.2015.7413680
37. Saxena A, Gupta DK, Singh S (2021) An animal detection and collision avoidance system using deep learning. In: Advances in communication and computational technology. Springer, Singapore, pp 1069–1084
38. Schultz MG, Eskin E, Zadok F, Stolfo SJ (2001) Data mining methods for detection of new malicious executables. In: Proceedings 200 IEEE symposium on security and privacy S P 2001, pp 38–49. https://doi.org/10.1109/SECPRI.2001.924286
39. Tan M, Le Q (2019) EfficientNet: rethinking model scaling for convolutional neural networks. In: Proceedings of the 36th international conference on machine learning, PMLR, proceedings of machine learning research, vol 97, pp 6105–6114
40. Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, Zheng Q (2020) Imcfn: image-based malware classification using fine-tuned convolutional neural network architecture. Comput Netw 171:107138. https://doi.org/10.1016/j.comnet.2020.107138
41. Vasan D, Alazab M, Wassan S, Safaei B, Zheng Q (2020) Image-based malware classification using ensemble of cnn architectures (imcec). Comput Secur 92:101748. https://doi.org/10.1016/j.cose.2020.101748
42. Venkatraman S, Alazab M, Vinayakumar R (2019) A hybrid deep learning image-based analysis for effective malware detection. J Inform Secur Applic 47:377–389. https://doi.org/10.1016/j.jisa.2019.06.006
43. Verma V, Muttoo SK, Singh VB (2020) Multiclass malware classification via first- and second-order texture statistics. Comput Secur 97:101895. https://doi.org/10.1016/j.cose.2020.101895
44. Yuan B, Wang J, Liu D, Guo W, Wu P, Bao X (2020) Byte-level malware classification based on markov images and deep learning. Comput Secur 92:101740. https://doi.org/10.1016/j.cose.2020.101740
45. Zhang H, Xiao X, Mercaldo F, Ni S, Martinelli F, Sangaiah AK (2019) Classification of ransomware families with machine learning based on n-gram of opcodes. Futur Gener Comput Syst 90:211–221. https://doi.org/10.1016/j.future.2018.07.052