International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V70I4P226

Volume 70 Issue 4, 294-302, April 2022
© 2022 Seventh Sense Research Group®

Original Article

An Efficient Android Malware Detection Framework with Stacking Ensemble Model

A. Lakshmanarao[1,2], M. Shashi[2]

[1]*Department of IT, Aditya Engineering College, Surampalem, India.*
[2]*Department of CS&SE, AU College of Engineering, Andhra University, Visakhapatnam, A.P, India.*

laxman1216@gmail.com

Received: 13 March 2022 Revised: 21 April 2022 Accepted: 22 April 2022 Published: 26 April 2022

Abstract *- Due to the increased frequency of cyber-attacks with various targeted objectives, cyber security has become a major concern for society. Android phones being the most widely used devices, they are targeted in most of the attacks with malware. So, it is vital to explore innovative ways of identifying Android Malware attacks. Machine learning and deep learning have been employed to develop classifiers to determine if an app is malware or benign. Android apps are represented by a set of attributes that can describe their behaviour. This paper proposes a stacking ensemble model for detecting Android malware. The proposed framework is designed with two variants of stacking ensemble: blending and stacking. The dex files of android apps are extracted and translated into images. Later, a stacking ensemble is applied to the image dataset. Convolutional Neural Networks are used as base learners, and a Support Vector Machine is used as a meta learner. The experimental results of modelling with blending and stacking showed 99% and 98.3% accuracy, which advocates support of the proposed framework for Android malware detection.*

Keywords - *Android malware detection, CNN, Stacking Ensemble, SVM.*

1. Introduction

The number of attacks on mobile devices appears to increase unprecedentedly. More than 14.4 million attacks on mobile phones were recorded worldwide in the second quarter of 2021 only from a single antivirus (Kaspersky reports) firm [1]. Android has a dominant position in the smartphone market. However, this success has a downside as more per cent of mobile malware targets Android phones for stealing money or personal information. Attackers could use various Android development platforms to create malicious mobile apps. Infecting users' mobile devices with malicious software might have severe implications. Despite Google Play's numerous measures to keep dangerous apps out, attackers continue to find their way onto the mobile devices and penalize unsuspecting victims. Therefore, Android malware is becoming a growing threat to businesses and individuals. Machine Learning is a field of computer science that deals with developing intelligent systems by integrating prior examples and making forecasts of future occurrences. Because of these properties are widely used in cybersecurity, such as intrusion detection and malware detection. Anti-malware solutions have focused on signature-based recognition, which requires prior knowledge of the malware in the form of a signature. Early identification of Android Malware is essential to limit the negative effects. Malware analysis techniques are classified into static Analysis and dynamic Analysis. Static Analysis is the most frequently used and preferred method by many researchers due to its low computation complexity and ease of implementation. This method analyses the application's source code without running it on an emulator or a real device. The APK archive is first unpacked to collect methods, manifests, meta-data, and media assets to perform this. The app's source code format at this point is dex bytecode, which is difficult to work with. Therefore it can be decompiled to java code/Smali code to make it more readable and process-able. After the extraction of the mobile app, several static features can be extracted. Static features include android app permission features, opcode sequences in the apk, strings, Method API features, Component features, intent features, and system command features. The extracted app does not contain all these features directly. Various tools can be used to extract all these features. In dynamic Analysis, the app is run in an isolated environment where it is feasible to obtain as much data as possible on the app's activity. In this method, additional features are extracted from the app's network traffic, sequence of events happening in the app execution, log behaviours, API monitoring etc. The authors proposed a stacking ensemble model with Convolutional Neural Networks and a Support Vector Machine for malware detection.

1.1 Ensemble Learning

In machine learning, ensemble learning is a generalized meta-approach that aims to improve prediction accuracy by combining the predictions of several models. Ensemble in machine learning can be done differently, but the three important ensemble techniques are bagging, boosting, and stacking. Although all three methods have their advantages and limitations, the suitability of a specific approach depends on model expansions and associated procedures.

1.1.1 Bagging

Bagging integrates the outcomes of numerous models of the same type, for example, decision trees, to produce a more generalized result. From the original dataset, many sample subsets are produced by replacing observations. A basic model (weak model) is generated on each of these sample subsets. The models are independent of one another and run in parallel. The final predictions are calculated by combining all of the models' predictions. Random forest is a commonly used bagging technique. In bagging, the resulting model can have a lot of bias if the proper learning procedure is not followed.

1.1.2 Boosting

Boosting is a sequential mechanism in which each consecutive model seeks to rectify the prior model's mistakes. The models that follow are reliant on the prior model in the sequence. From the original dataset, sample subsets are produced. At first, all training data points are given equal importance for getting selected in the sample. On this subset, a base model is built. This model is applied to the dataset to predict the class labels of known observations, and errors are determined. Accordingly, in the next iteration, higher weights are assigned to the wrongly predicted observations, and a new sample data is generated to build the next base model. Based on this model's (wrong) predictions, subsequent base models are built in the same way, each one fixing the deficiencies of the earlier models.

1.1.3 Stacking

Stacking ensemble builds a separate learning model, possibly with a different algorithm, on top of the base learners to combine their predictions. In other words, a stacking ensemble adapts a two-fold learning strategy that involves multiple simpler models in the first fold whose predictions would be further processed by a separate learning model at the next fold to make more accurate predictions. The stacking ensemble differs from the bagging and boosting ensemble as it applies an adaptive approach rather than a static approach for combining the predictions made by the base classifiers. Learning algorithms used to train the multiple base models in the first fold are often different from those used in the second fold. On the test set, this ensemble model is applied for making predictions. The structure of the stacking model is shown in figure-1.

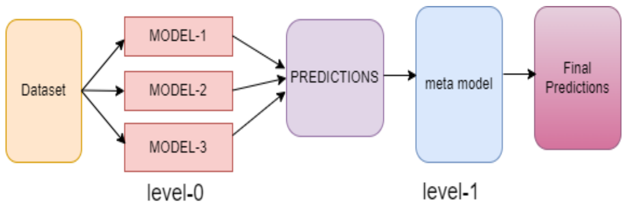

Fig. 1 Stacking Ensemble

The rest of the paper is organized in the following sections: Section 2 describes previous work done on android malware detection. Section 3 discusses the proposed methodology, Section 4 presents experimentation and results, and Section 5 concludes and provides an outlook for future work

2. Related Works

Various researchers use machine learning and deep learning methods to detect malware in android apps. In [2], a framework using static Analysis has been utilized. Each of the used permissions, susceptible APIs, observed event logs, and permission rate is used as features for the classifier. Sen Chen[3] et al. discussed various issues where malware detection techniques failed. The challenges associated with the failure of conventional ML algorithms are also discussed. Shabtai [4] et al. Information Gain was used to select permission features and code features. Using a mix of permissions and code features, an accuracy of 93% was achieved. Li et al. [5] developed a permission-based malware detection method to distinguish between malware and legitimate apps. In [5], the authors used three stages of filtering with rule-based associated mining to discover important permissions. The machine learning model with filtered permissions achieved 96% accuracy for detecting malware apk. A. Lakshmanarao [6] et al. applied CNNs for android malware classification. Android apps are converted to images in two variants. The entire apk images and dex part images of apks were created. Later, Deep Learning CNN was applied and achieved good accuracy with dex images. T Chakraborty [7] et al. utilized a method for classifying malware samples into various malware families. The model used static features and built an ensemble that combines classification and clustering techniques. In [8], the authors applied de complication to obtain all API calls in the dataset with 516 non-malware apps and 528 malware apps, then computed and prioritized the risk associated with these APIs using a mutual information model. In [9], static features are generated by extracting android manifest files, and an ensemble XGBoost was then used to detect malware apps with 95.25% accuracy. Ji Wang [10] proposed a selective ensemble learning model for android malware detection and achieved a recall of 98%.

Mahindru [11] et al. applied a Neural Network model with Principal Component Analysis for malware detection on a dataset of 1,20,000 mobile apps and achieved an accuracy of 94%. Zhu et al. [12] used a stacking ensemble to detect malware. The proposed system employs a two-way architecture, with the ensemble of the base learner using MLP and finally the output of base learners being merged using SVM. In the initial stage, the twofold disruption of features ensures the diverseness of the train subsets, & PCA is performed separately. MLP is run on each set, retaining principal components determined through PCA but changing the entire train dataset into a completely new set to ensure the base learner accuracy. The next step, called the fusion step, involves learning implicit supplemental data from base learner output to maximize classification performance. In this paper, the authors also used the stacking ensemble approach. Still, convolutional neural networks are used as base learners, and SVM was used as a meta learner for final predictions. Eslam Amer [13] et al. applied an ensemble-based machine learning model for malware detection. As an initial step, a random forest was used for feature selection. Later, an ensemble learning algorithm, "Extra Tree Classifier", was applied to selected features and achieved better results. R. S. Arslan[14] et al. proposed an ensemble ML model for malware classification. Four types of malware, namely ransomware, adware, scareware, and ransomware, are classified with an accuracy of 90.4% with an ensemble approach. Altyeb

Taha [15] et al. proposed a novel technique that utilizes permission features of android apps and aggregates the classification results of multiple classifiers, including Light GBM, Adaboost, XGBoost, Random Forest, Decision Trees, by utilizing the Choquet fuzzy integral function. Choquet's fuzzy integral is based on fuzzy measures computed using the significance level of the base classification model or a subset of classifiers. The fuzzy measure was fine-tuned with two different factors, the confidence of each classifier's classification results and the consistency of each classifier's classification results. The classification outcome with a large choquet integral was considered for final predictions. Potha et al. [16] investigated the impact of using external instances (benign or malware) of various sizes and types while using an ensemble model. The output of numerous base models, including Logistic Regression and MLP, is combined in this innovative ensemble model called ERBE (Extrinsic Random-based Ensemble). The findings showed that ensemble models with a larger and perhaps more homogeneous length of instances are far more successful than those with smaller and more diverse instances sizes. They also looked at the impact of employing either the complete feature set or a random subspace comprising instance features. They discovered that the latter helps an extrinsic ensemble model perform better. Christianah[17] et al. proposed an ensemble approach for android malware detection. Permission features from 952 non-malware and 952 malware apks are extracted. Three base learners support vector classifier-nearest neighbours, random forest is used as base learners, and a majority voting mechanism is employed for making final predictions. The final ensemble model achieved an accuracy of 98% for malware detection. Kouliaridis et al. [18] presented a simple ensemble malware detection approach. The result of numerous base models based on static or hybrid Analysis is averaged to form the ensemble model. APIs, Permissions, java classes, network traffic, and intents are used as features. The performance is measured against three different datasets using various classifiers, including Naive Bayes, Logistic Regression, Random Forest, k-NN, AdaBoost, SGD, and SVM. The final model achieved an AUC score of 97%. Rana[19] et.al proposed an ensemble technique with SBFS(substring-based feature selection)strategy.The string features of the apps are extracted, and a subset of features was selected in the final dataset. Later various classifiers, Random Forest, Decision Trees, ERT, GB, and SVM, are applied as base learners for ensemble, and an accuracy of 97.6% is achieved for malware detection. A. A.Lakshmanarao[20] et al. extracted opcode sequences from android apks, and an RNN is deep learning model was proposed on opcode sequences. As a single android app contains several opcode sequences, all the sequences are labelled with the same class label (for example, if an apk is malware, all opcode sequences from that malware apk are labelled as malware).

3. Research Methodology

The Android app is an archived file with an apk extension. Extraction of an android app produces other files, including meta-inf, manifest file, and classes.dex file, assets, res, and lib. Meta inf is a directory with app metadata. The manifest file contains the app's name, apk version, and permissions information. 'assets' is a directory of app assets. Res contains app resource information. The compiled native libraries are stored in the 'lib' directory. Classes.dex contains application code index format. The Java compiler converts Java source code into dot class files. The dx (dexter) tool, part of the Android SDK, converts the dot class files into the. Dex (dalvik executable) file. So, the code part of the android app is stored in classes.dex file. The proposed architecture is keen on the creation of images with minimal overhead. As classes. dex files are enough for differentiating malware and non-malware apps[6]; the authors extracted android apps images from classes.dex files using the image generation algorithm developed by the authors discussed in [6]. The width of images varies in the range of 32 KB-1024 KB depending on the size of apk.

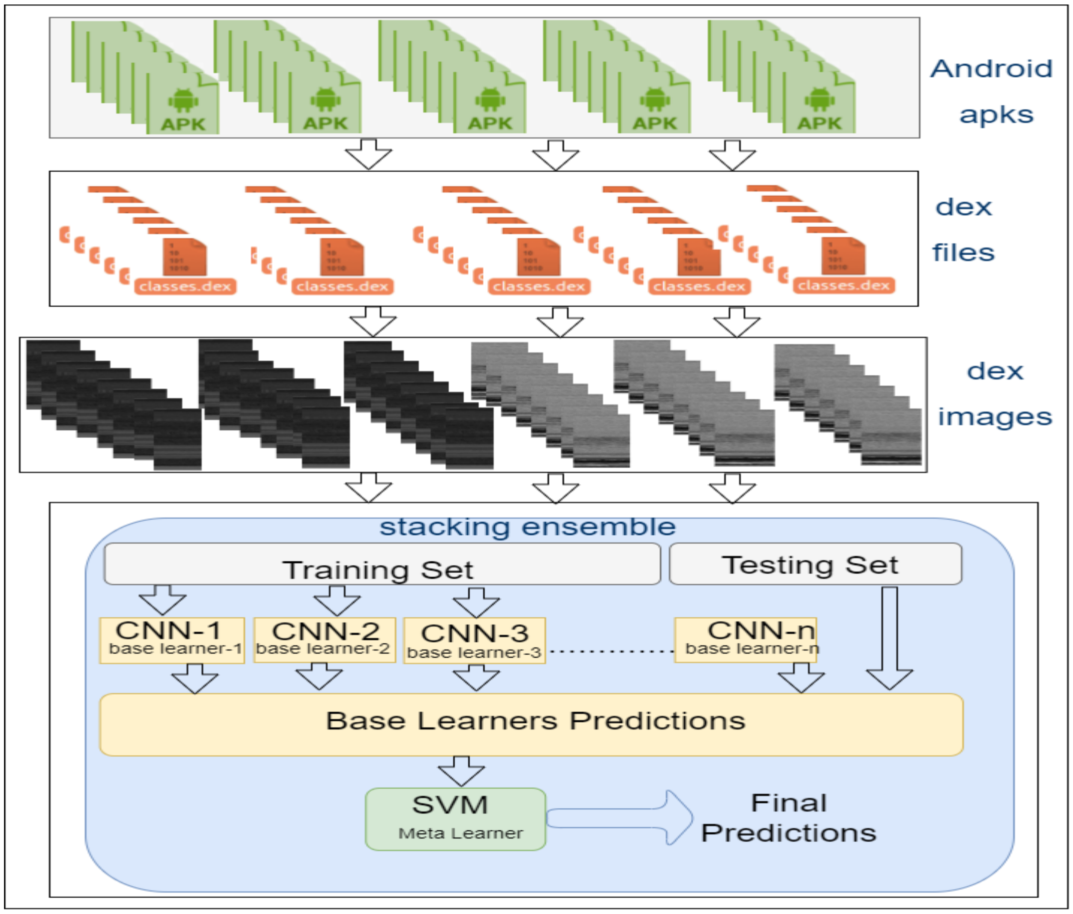

Fig. 2 Proposed Methodology

The authors proposed a stacking ensemble architecture with Convolutional Neural Networks (CNN) followed by a Support Vector Machine to process the image dataset once the Android apk dataset is converted into image form. The proposed framework is shown in Fig.2.

3.1 Stacking Ensemble

In the stacking ensemble framework, the system takes the outcomes of sub-models as inputs and tries to figure out how to combine the input predictions in the best way possible to get a better output prediction. Fig. 4 depicts the stacking ensemble with 2 levels: level-0 has base models & level-1 has the meta-model.

The steps involved in building the stacking ensemble are:

- Divide the given data set into training data and testing data.
- Initialize the appropriate hyperparameters to configure L base learners at Level-0 and build the base models using the training dataset. During training, observe the accuracy of base learners and adjust the parameters to get better accuracy.
- Train the Meta Model at the next level, Level-1, using predictions made by the base learners on a separate set of training observations.

Stacking ensembles can be trained in either blending mode or stacking mode.

3.1.1 Blending

Blending is a specific way to train the stacking ensemble that uses a holdout approach for dataset division. First, the dataset is divided into training and testing sets. Later, the training dataset is further divided into training_sub and holdout sets. Level-0 models are trained using training_sub. At Level-1, meta learner is trained using the holdout set. Later, the test set is used to make final predictions and assess the framework's generalization performance.

3.1.2 Stacking

In stacking also, train and test sets are created from the dataset. The base learners are trained on the complete set of training data. For training meta learners, a k-fold cross-validation technique is used. The training data is divided into k folds. All L base learners are applied on k-folds in the training data separately, producing (n x L) number of predictions where n is the number of samples in the training set. Later, the meta learner is trained, taking these predicted labels as input and the true label of the corresponding training samples. Finally, the testing dataset makes final predictions and assesses the framework's generalization performance.

The authors applied both stacking ensemble and blending ensemble models for malware in this paper. Convolutional neural networks are used as base learners in level-0, and Support Vector Machine is used as meta learners in level-1. In level-0, ten CNNs are trained with different hyperparameter settings regarding #layers, optimizers, number of epochs, etc.

3.2 Convolutional Neural Networks

Yann LeCun [21] introduced convolutional neural networks (also known as ConvNets) in the 1980s. In CNN, images are reduced to a form that can be processed more easily, but important features are not lost during image reduction. Convolutional layers are followed by an activation function, a pooling layer, and a fully connected layer in the CNN architecture. The model can contain any number of convolutional layers depending on the dataset's characteristics. The architecture of CNN is shown in fig. 3.

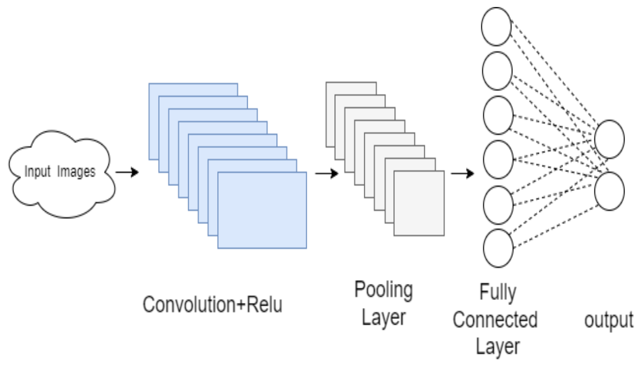

Fig. 3 Architecture of CNN

3.2.1 Convolutional Layer

This is also called as kernel layer. The Kernel/Filter is the first component in a Convolutional Layer that performs the convolution operation. The use of appropriate filters by a CNN can effectively capture spatial & temporal dependencies in an image. The output of this layer is called convolved feature. An activation function is applied at the end of the convolution layer.

3.2.2 Pooling Layer

The pooling layer takes care of reducing the size of the Convolved Feature, thereby reducing the complexity of the model. There are several variations in pooling, but max pooling is the most widely used technique. It preserves important features and reduces the size.

3.2.3 Fully Connected Layer

The output from the last pooling layer is flattened and fed through the fully connected layer. This is similar to a feed-forward network. From this stage, CNN works like an ANN only.

3.3 Support Vector Machine

Both classification & Regression problems are solved with SVM. The main goal of the algorithm is to find the best decision boundary (optimum hyperplane) for dividing data samples into different classes. Multiple separating planes may exist between the training samples of different classes, which are referred to as hyper-planes. The hyper-plane that partitions the data samples of different classes more efficiently is the one that has maximum margin between the training data points identified as support vectors as it results in minimal test error and hence is selected as the optimal hyperplane. Learning such an optimal hyper-plane often involves setting hyperparameters such as tolerance thresholds, kernel function, etc., to handle non-linearly separable data space.

4. Experimentation and Results

4.1 Details of Dataset

2511 malware apps are gathered from the site "virusshare.com" and 2508 benign apps from the play store, apkpure and CICAndMal2017[22]. Table-1 shows details of the dataset collected.

Table 1. Details of Dataset

Number of malware apks (virusshare.com)	2511
Number of Benign apks (Play store, apkpure, CICMAL2017)	2508
Total	5019

4.2 Applying the Stacking Ensemble Framework

All the collected android apps (both benign and malware) are converted into images using the algorithm developed by the authors in [6]. Later, blending and stacking are applied to the image dataset.

4.2.1 Experiments with Blending

The architecture of the proposed blending framework is shown in fig. 4. The steps in the blending model are:

- Dataset division: First, the dataset is divided into the full-training set and the testing set. The division is done with a 75%,25% ratio. The dataset contains 5019 images. So, the full-training set contains 3764 images, and the testing set contains 1255 images. The next full-training- set is further divided into the training set and holdout set with a 75% (2823 images) and 25% (941 images) ratio.

Table 2. Base Learners architecture

Base Learner	**Number of convolutional layers**	**Number of Neurons (layer-wise)**	**Optimizer**	**Number of epochs**
CNN1	4	250,200,150,100	adam	50
CNN2	3	150,70,30	sgd	60
CNN3	4	500,360,240,160	adam	50
CNN4	3	450,350,200	adam	100
CNN5	4	250,150,120,100	RMSprop	70
CNN6	4	200,180,150,70	adam	100
CNN7	3	160,120,100	adam	100
CNN8	2	160,100	sgd	120
CNN9	4	220,170,140,65	adam	120
CNN10	4	280,150,110,85	adam	125

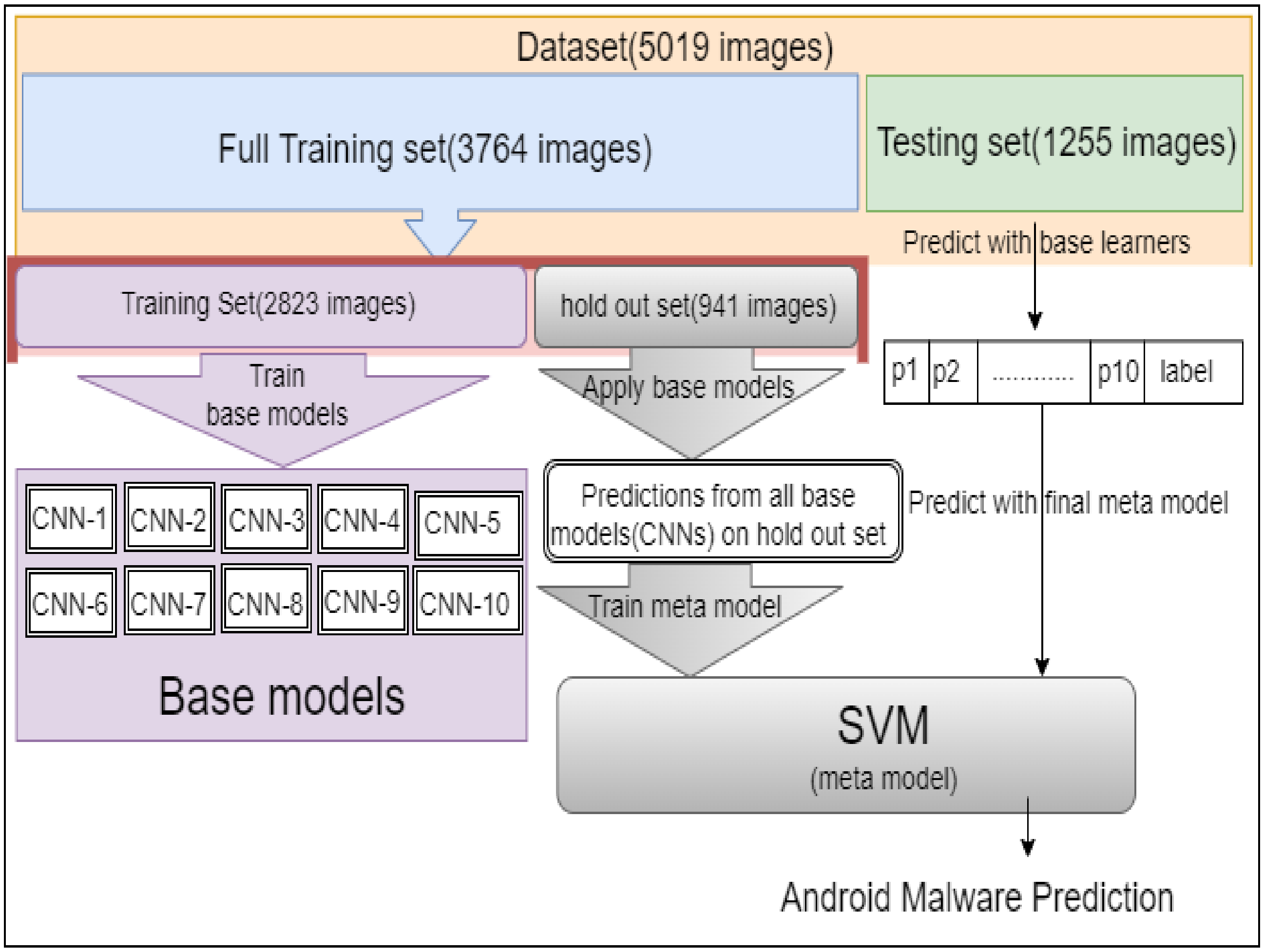

Fig. 4 Architecture of proposed blending model

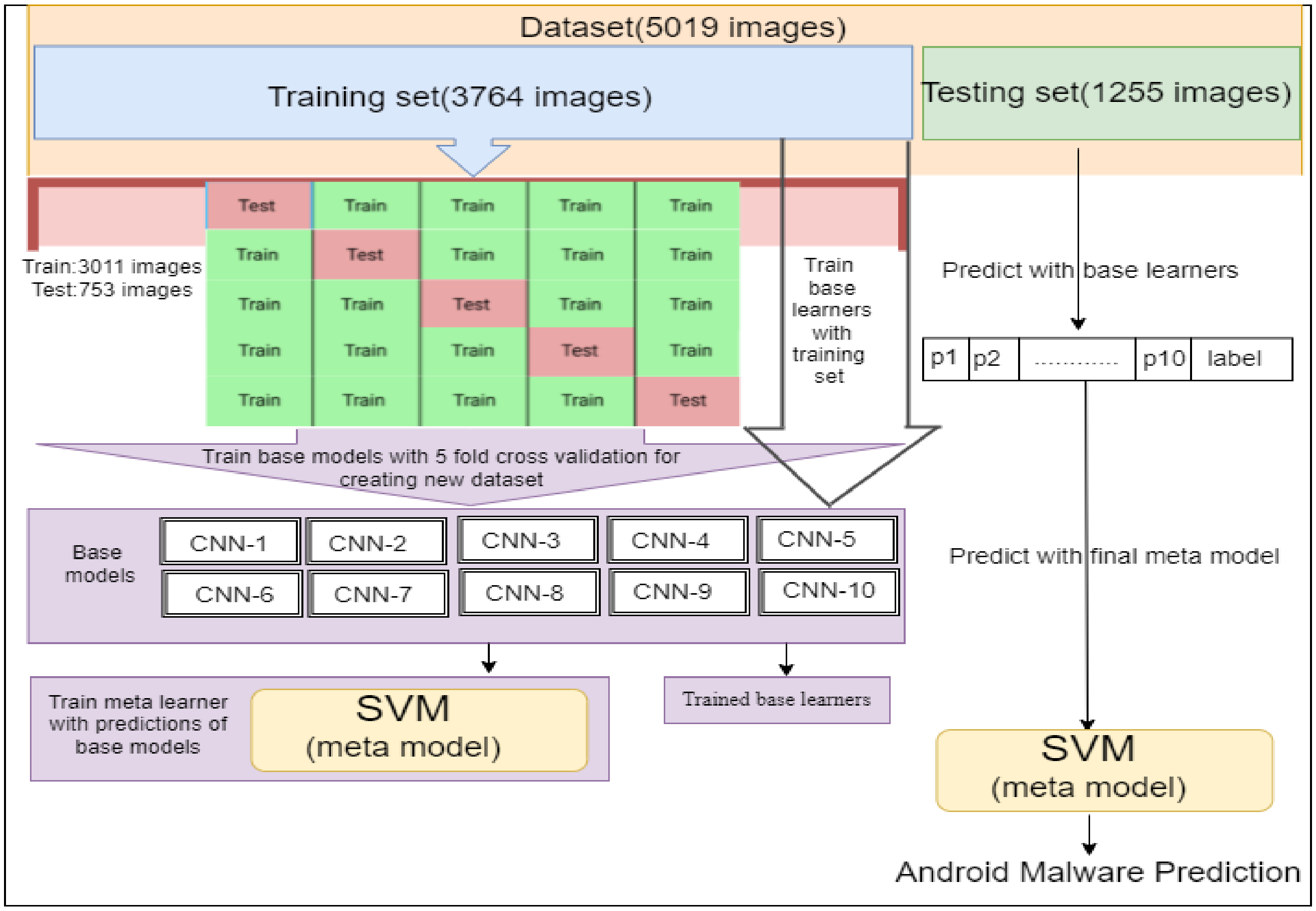
Dataset(5019 images)
Training set(3764 images)
Testing set(1255 images)
Train:3011 images
Test:753 images
Test
Train
Train base learners with training set
Predict with base learners
p1
p2
p10
label
Train base models with 5 fold cross validation for creating new dataset
Base models
CNN-1
CNN-2
CNN-3
CNN-4
CNN-5
CNN-6
CNN-7
CNN-8
CNN-9
CNN-10
Predict with final meta model
Train meta learner with predictions of base models
SVM (meta model)
Trained base learners
Android Malware Prediction

Fig. 5 Architecture of proposed stacking model

- Training base learners (CNNs): Base learners are trained on a training dataset (2823 images). Ten different CNNs are trained with different hyperparameter settings, shown in Table-2. RELU is used as an activation function in hidden layers for all base learners, the Sigmoid function is used as the output function, and 'binary_cross-entropy' is used as the cost function.
- Training meta-model (Training SVM): The meta learner used in the proposed architecture is SVM. SVM is trained on a holdout set. First, the base learner's predictions are calculated on a holdout set. The predictions and holdout set class labels to become a new dataset for training SVM. The reason for selecting a holdout set is to reduce overfitting. If both base learners and meta-learners were trained on the same dataset, there might be a possibility of overfitting.
- Testing performance of proposed model: The final model is tested on the testing set. First, the base learners apply, and predictions are stored. The predictions and the class label are fed through the proposed model to make final predictions. The proposed blending model achieved an accuracy of 99% with the testing dataset.

4.2.2 Experiments with Stacking

The architecture of the proposed stacking framework is shown in fig. 5.

The steps in the proposed staking model are:

- Dataset division: First, the dataset is divided into two parts: the training and testing sets. The division is done with a 75%,25% ratio. The dataset contains 5019 images in total. So, the full training set contains 3764 images, and the testing set contains 1255 images. Next, the training set is further divided into 5 folds.
- Training base learners (CNNs) and meta learners: Base learners are trained on 5 folds with a testing set of 753 images and a training set of 3011 images. So each base learner trained 5 times. The predictions of the training set (753 * 5 times =3765 predictions) are preserved for all base learners, and models are discarded. The architectures of base learners (ten CNNs) are shown in table-2. (Same ten CNN architectures used for blending). So, ten columns of predictions are preserved. These predictions, along with actual class labels, form a new dataset. A meta learner (SVM) has been trained

with this new data set. That means the meta learner is trained with cross-validated predictions of base models. Finally, all base learners are trained with the entire training set (3764 images).

4.3 Comparison with previous work

The performance of the proposed model was compared with previous work (Table-3). In [22], the authors applied the voting classifier and achieved 89.7% accuracy. In [8], the authors proposed ensemble learning on sensitives api's and achieved 94% accuracy. In [23], the authors proposed a hybrid deep learning technique and achieved 96.8% accuracy. The authors [12] applied a stacking ensemble with ANNs as base models and SVM as a meta-model and achieved 94.92% accuracy. This paper applied ensemble learning techniques with CNNs as base models and SVM as a meta-model. The stacking ensemble framework is applied in two different ways, namely blending and stacking and achieved an accuracy value of 99% and 98.3%, respectively(figure-6). The base models and meta-models were trained on different datasets in both techniques. So, there is no overfitting. Since the model's accuracy is very high on the new dataset (testing set), it is concluded that the proposed model is capable of efficiently classifying new samples of malware and benign.

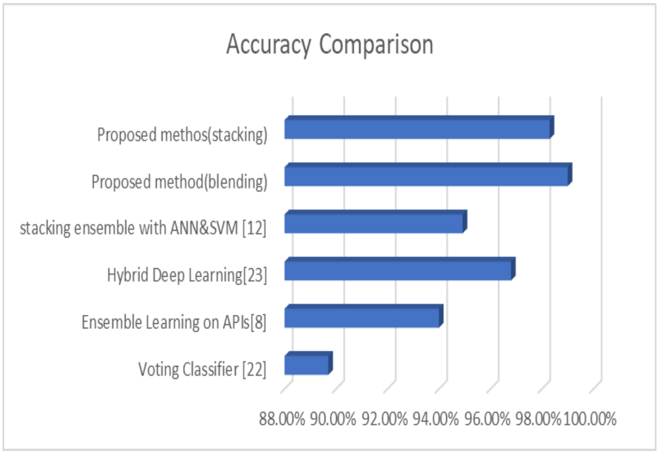

Fig. 6 Accuracy Comparison

Table 3. Details of Dataset

Method	**Accuracy**
Voting Classifier [22]	89.7%
Ensemble Learning on APIs [8]	94%
Hybrid Deep Learning [23]	96.8%
stacking ensemble with ANN&SVM [12]	94.92%
The proposed method(blending)	99%
The proposed method(stacking)	98.3%

5. Conclusion

In this paper, a stacking ensemble model was proposed for Android malware detection and accordingly, a framework was developed to investigate the efficiency of the proposed model. Malware and benign Android apps are collected and classes.dex files are extracted from apks and converted to images. The image dataset is then processed in two different ways of stacking ensemble framework: blending and stacking. The meta-model was trained with a holdout dataset, while the base models were trained with a training dataset. The proposed blending model achieved an accuracy of 99%. The meta-model was trained with a cross-validated training set, while the base models were trained with a training set. The proposed stacking model achieved an accuracy of 98.3%. CNNs are used as base models in blending and stacking, and SVM is used as the meta-model. Experimental results have shown that the proposed stacking ensemble framework outperforms conventional machine learning/deep learning models for Android malware detection.

References

[1] https://securelist.com/it-threat-evolution-q2-2021-mobile-statistics/103636/
[2] Hui-Juan Zhu, Zhu-Hong You, Ze-Xuan Zhu, Wei-Lei Shi, Xing Chen, Li Cheng., DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model, Neurocomputing, 272 (2018) 638-646. ISSN 0925-2312,https://doi.org/10.1016/j.neucom.2017.07.030.
[3] Sen Chen, MinhuiXue, Lingling Fan, Shuang Hao, Lihua Xu, Haojin Zhu, Bo Li., Automated poisoning attacks and defences in malware detection systems: An adversarial machine learning approach, Computers &Security, 73 (2018) 326-34. ISSN 0167-4048, https://doi.org/10.1016/j.cose.2017.11.007.
[4] A. Shabtai, Y. Fledel and Y. Elovici., Automated static code analysis for classifying android applications using machine learning, Int. Conf. Computational Intelligence and Security, (2010) 329–333.
[5] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an and H. Ye., Significant Permission Identification for Machine-Learning-Based Android Malware Detection, in IEEE Transactions on Industrial Informatics, 14(7) (2018) 3216-3225. doi: 10.1109/TII.2017.2789219.
[6] A. Lakshmanarao, M.Shashi., Android Malware Detection Using Convolutional Neural Networks, In Data Engineering and Intelligent Computing. Advances in Intelligent Systems and Computing, 1 (2021) 151-162. https://doi.org/10.1007/978-981-16-0171-2_15.
[7] T.Chakraborty, F. Pierazzi and V. S. Subrahmanian., EC2: Ensemble Clustering and Classification for Predicting Android Malware Families, in IEEE Transactions on Dependable and Secure Computing, 17(2) (2020) 262-277.doi: 10.1109/TDSC.2017.2739145.

[8] Yu Junhui, ZhaoChunlei, ZhengWenbai, Yunlong Li, ZhangChunxiang, Chen Chao., Android Malware Detection Using Ensemble Learning on Sensitive APIs, Springer International Professional, (2021). https://doi.org/10.1007/978-3-030-73429-9_8.

[9] D.Congyi, Guangshun S., Method for Detecting Android Malware Based on Ensemble Learning, In Proceedings of the 2020 5th International Conference on Machine Learning Technologies, (2020) 28–31. Association for Computing Machinery.

[10] Ji Wang, QiJing, Jianbo Gao., SEdroid: A Robust Android Malware Detector using Selective Ensemble Learning, (2019). arXiv:1909.03837v1.

[11] A. Mahindru and A. L. Sangal,. DeepDroid: Feature Selection approach to detect Android malware using Deep Learning, 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), (2019) 16-19. doi: 10.1109/ICSESS47205.2019.9040821.

[12] H. Zhu, Y. Li, R. Li, J. Li, Z. You and H. Song., SEDMDroid: An Enhanced Stacking Ensemble Framework for Android Malware Detection, in IEEE Transactions on Network Science and Engineering, 8(2) (2021) 984-994. doi: 10.1109/TNSE.2020.2996379.

[13] Eslam Amer, Ivan Zelinka (2019) An Ensemble-Based Malware Detection Model Using Minimum Feature Set, MENDEL. 25 (2019) 1-10. 10.13164/mendel.2.001.

[14] R. S. Arslan (2021) Identify Type of Android Malware with Machine Learning Based Ensemble Model, 2021 5th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT), (2021) 628-632, doi: 10.1109/ISMSIT52890.2021.9604661.

[15] A. Taha, O. Barukab, and S. Malebary., Fuzzy Integral-Based Multi-Classifiers Ensemble for Android Malware Classification, Mathematics, 9(22) (2021) 2880.

[16] N.Potha, V. Kouliaridis& G. Kambourakis., An extrinsic random-based ensemble approach for android malware detection, Connection Science, 33(4) (2021) 1077-1093, DOI: 10.1080/09540091.2020.1853056.

[17] Christianah, A. O., Gyunka, B. A., &Oluwatobi, A. N., Optimizing Android Malware Detection Via Ensemble Learning, International Journal of Interactive Mobile Technologies (iJIM), 14(09) (2020) 61–78. https://doi.org/10.3991/ijim.v14i09.11548.

[18] V. Kouliaridis, G. Kambourakis, D. Geneiatakis, and N. Potha ., Two Anatomists Are Better than One—Dual-Level Android Malware Detection, Symmetry, 12(7) (2020) 1128.

[19] Rana, Sung., Evaluation of advanced ensemble learning techniques for android malware detection, Vietnam Journal of Computer Science 7 (2)(2020)145–59.

[20] Lakshmanarao, A., & Shashi, M., Android Malware Detection with Deep Learning using RNN from Opcode Sequences. International Journal of Interactive Mobile Technologies (iJIM), 16(01) (2022) 145–157. https://doi.org/10.3991/ijim.v16i01.26433.

[21] LeCun, Y., Haffner, P., Bottou, L., Bengio.Y., Object Recognition with Gradient-Based Learning, In. Shape, Contour and Grouping in Computer Vision. Lecture Notes in Computer Science, 1681 (1999). Springer, Berlin, Heidelberg,https://doi.org/10.1007/3-540-46805-6_19.

[22] Martín Garcia, Alejandro & Lara-Cabrera, Raul & Camacho, David., Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset, Information Fusion. 52 (2019). 10.1016/j.inffus.2018.12.006.

[23] Lu, T., Du, Y., Ouyang, L., Chen, Q., Wang, X. (2020) Android malware detection based on a hybrid deep learning model In Secur. Commun. Netw., 8 (2020) 1–11.