




# Düzce University Journal of Science & Technology

Research Article

## Android Malware Analysis and Benchmarking with Deep Learning<sup>1</sup>

 Taylan Kural <sup>a,\*</sup>,  Yusuf SÖNMEZ <sup>b</sup>,  Murat DENER <sup>a</sup>

<sup>a</sup> Information Security Engineering, Graduate School of Natural and Applied Sciences, Gazi University, Ankara, TURKEY

<sup>b</sup> Faculty of Information and Telecommunication Technologies, Azerbaijan Technical University, Baku, AZERBAIJAN

\* Corresponding author's e-mail address: [taylan.kural@gazi.edu.tr](mailto:taylan.kural@gazi.edu.tr)

DOI: [10.29130/dubited.1015654](https://doi.org/10.29130/dubited.1015654)

### ABSTRACT

Android operating system has been widely used in mobile phones, televisions, smart watches, cars and other Internet of Things applications with its open source structure and wide application market. This widespread use and open-source nature make this operating system and its devices easy and lucrative targets for cyber attackers. One of the most used methods often preferred by attackers is to install malware applications on user devices. As the number of malware programs is increasing, the traditional methods can be insufficient in detecting. Machine learning-based and deep learning-based methods have achieved promising results in malware detection and classification. Deep learning-based methods have an increasing use in malware detection, thanks to the low need for domain expertise and their feature extracting capabilities. Convolutional neural networks (CNN) are popular deep learning methods that are widely used in visual analysis of malware by transforming them to images. In this study, a batch fine-tune transfer learning method was proposed and used on popular CNN models, Xception, ResNet, VGG, Inception, MobileNet, DenseNet, NasNet, EfficientNet. According to the results, the models were analyzed and compared with metrics like accuracy, specificity, recall, precision, F1-score.

**Keywords:** Deep learning, Android malware analysis, Image analysis

## Derin Öğrenmeyle Android Kötücül Yazılım Analizi ve Kıyaslanması

### ÖZ

Android işletim sistemi, açık kaynak olan yapısı, geniş uygulama marketiyle telefonlarda, televizyonlarda, saatlerde, arabalarda ve diğer nesnelerin interneti uygulamalarında yaygın olarak kullanılmaktadır. Bu yaygın kullanım ve açık kaynak yapısı, kötücül niyet barındıran saldırganlar için bu işletim sistemini ve sahip olduğu cihazları kolay ve kazançlı hedefler haline getirmektedir. Saldırganlar tarafından sıklıkla tercih edilen bir yöntem de kötücül yazılım uygulamalarının kullanıcı cihazlarına yüklenmesidir. Bu yazılımların sayısı gün geçtikçe artmakta, kötücül yazılımları tespitinde geleneksel yöntemler yetersiz kalabilmektedir. Kötücül yazılım tespitinde makine öğrenmesi ve derin öğrenme tabanlı yöntemler umut veren sonuçlar elde etmişlerdir. Özellikle derin öğrenme tabanlı yöntemler, alan uzmanlık bilgisi gereksiniminin azlığı ve kendi kendine özellik çıkarabilen yapıları sayesinde, kötücül yazılım tespitinde artan bir kullanıma sahiptirler. Kötücül yazılımların görsel imajlara dönüştürülerek bu imajlar üzerinde CNN tabanlı derin öğrenme modelleriyle görsel kötücül yazılım analizleri gerçekleştirilmektedir. Çalışmada, popüler CNN modelleri olan Xception, ResNet, VGG, Inception, MobileNet, DenseNet, NasNet, EfficientNet sunulan toplu ince ayar öğrenim aktarma yöntemiyle eğitilmiş ve elde edilen sonuçlara göre modeller doğruluk, kesinlik, geri çağırma, hassaslık, F1 skoru metriklerine göre kıyaslanmıştır.

<sup>1</sup> The part of this study was presented as an oral presentation in ICAIAME 2021.  
Received: 27/10/2021, Revised: 13/12/2021, Accepted: 18/12/2021

## **I. INTRODUCTION**

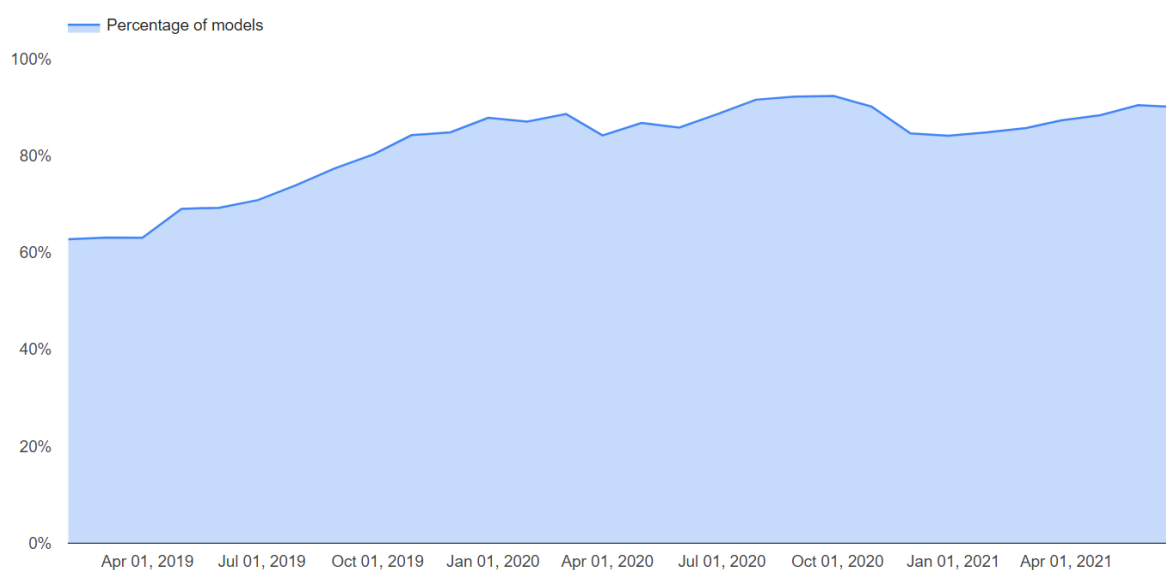
Android mobile operating system since 2008 its debut with the HTC Dream, has been in development and change. This change has been also changing the World with the advent of smartphones. The fact that smartphones can be more useful by installing applications, the emergence of new applications targeting mobile phones, especially communication (like chatting, social media etc.). The creation of billion-dollar economies by these applications makes smartphones an integral part of daily life, both at the level of personal and corporate users. As the devices become more connected to the network, the users become more connected to their devices. All these developments have caused newly established enterprises and old companies to want to take part in the mobile application markets, thus wanting to be in constant contact with their users, and for this reason, they have changed their digital representations, which will serve their customers, from a web-first approach to a mobile-first approach. These may be indicators of a paradigm shift to a mobile-centric life. The Android operating system, which started its journey with smartphones, continues with cars, televisions, watches, tablet computers and other Internet of Things applications, and today it is called the Android ecosystem.

This change should also be followed and evaluated in terms of cybersecurity and information security. While the increasing use of mobile services in private and public institutions makes user lives easier, a large amount of personal and sensitive data is generated during the process. While banking applications were very few in the first years of smartphones, the term “mobile banking” is being highly used today. Banks are shifting their services to mobile. Especially with the ongoing Covid-19 pandemic that started in 2019, it becomes less preferable to receive and provide services from physical branches. As the emergence of new technologies brings new concepts such as blockchain and cryptocurrency into our lives, mobile wallets are the places where these new commodities are stored. Similarly, concepts such as in-app purchases, mobile payments, and near-field contactless payments have taken a place in our lives. These developments lead mobile devices to become the center of economic activities, and along with the communication capabilities of the devices, they can also become the center of our lives depending on the situation. This situation makes mobile environments a lucrative and attractive target for cyber attackers.

Although it took 2 years for the first known malware, FakePlayer [3], to appear in the Android operating system, after made its debut in 2008, today it takes its place in hundreds of new malware systems every year. In a recent report published by Malwarebytes in 2021 [4], stated that over the years, the detection of malware is become more difficult and silent, and trickier. In the same report, the amount of detection of the adware named “HiddenAds” in 2019 is increased nearly 2.5 times, from 288,704 to 704,418. The report shows that adware programs are still quite easy and lucrative “business” areas for cyber attackers. Another point that draws attention in the report is the huge increase in fake banking applications. While the amount of BankBot malware detected is 5,025 in 2019, it has increased to 198,031 in 2020. In Covid19 pandemic, it’s clear that new business opportunities are emerging for cyber attackers. It would not be wrong to think that with the banking services changing to mobile, more people will be affected by these attacks. The threat of Risk software, a new category that has emerged in recent years, is getting worse and worse, as stated in the same report. Threats that remain on the device despite the factory reset are indicated in the report. Also, it is stated that there is cooperation among the malware programs. If a malware has infected a device, it installs other malware programs to the device over time. The report states that the panic and fear caused by the Covid-19 pandemic is used by attackers for social engineering attacks. This shows that attackers develop new methods and techniques according to new events and

trends and strengthens the claim that the mobile ecosystem is attractive to attackers and worth investing in.

Android is unrivaled on its own. It's an open-source operating system which is used by many manufacturers all around the world. There are more than 3 billion Android devices worldwide by 2021[5]. But this freedom creates another security problem, because the devices in the ecosystem cannot get security updates from the manufacturer. The Android operating system is being updated frequently, announcing a new version almost every year. As of the date of this article, the latest version is Android 11 operating system, Android 12 will be released soon, but only a small of number of devices are lucky enough to get this new update. With the Google support policy to suspend "old" versions of Android operating systems for nearly three years periods, billions of devices will be insecure and easy targets for attackers. The Google Transparency Report [6] confirms this. According to this report published by Google, which reports the status of current roaming devices, the rate of devices that came out and received updates in the last 90 days in the last two years was announced as 62% in January 2019, and 90% in June 2021 [Figure 1]. Although there has been an increase in the rates, it should be noted that these rates only cover the devices released in the last two years.



**Figure 1.** Percentage of models with 90-day security updates [6].

According to the data obtained from Statista [7], the operating system distributions of the current devices are Android 11 17.73%, Android 10 36.47%. Considering that the oldest version released in the last two years is Android 10, almost half of the 3 billion devices will not get security updates from Google in the future. Even if supported versions get security updates, it's up to the manufacturers to deliver updates to devices. As the number of unsupported devices increase, these insecure devices will continue to exist in the Android ecosystem and remain as potential danger in the cyber world with their security vulnerabilities. When all these issues are evaluated, mobile devices that do not receive updates threaten cyberspace, just as garbage thrown into space or old satellites threaten real space.

With the increase in threats, many solutions have been proposed to solve these problems, but the most promising are machine learning and deep learning applications. Especially, deep learning models as most of them have self-feature-extracting capabilities. Convolutional neural networks (CNN) are popular deep learning methods that have been widely used in visual analysis. By transforming malware to images, it's possible to use CNNs for detection and classification of malware applications.

The main purpose of this paper is comparing the performance of the CNN models in Android malware analysis. For this reason, a new method is proposed. As researchers present new CNN models, the number of publicly available pre-trained models are increasing. While using a fine-tune transfer learning approach manually on a small number of CNN models may not be challenging, as the number of the models increase, using the manual approach could be a challenge. The proposed method offers a

solution to this problem by training models in a batch instead of manual approach. The method also gives insight into models' fine-tune transfer learning performance in the problem domain, which could be useful for researchers and developers for selecting the right CNN pre-trained model for the problem. We couldn't find any batch fine-tune transfer learning approach in the literature. To the best of our knowledge, the proposed method is one of the first, if not the first. In this paper, the proposed method is used for benchmarking popular CNN models' fine-tune transfer learning performance of Android malware detection and classification. The results show the best possible models for different problem settings, which could be useful for domain researchers and developers.

The contributions of this study are as follows:

- DroidMalImg visual dataset was created from the CICMalDroid2020 [1] dataset and made available for open access. [2]
- Popular convolutional neural network models were trained with our proposed batch fine-tune transfer learning method and their performances were compared.

## **II. RELATED WORKS**

Malware analysis is the study of detecting and classifying malware. Detection studies include a binary classification as malicious, benign, while classification studies are studies of various classes, such as categorical, familial. Malware analysis is performed manually or with the help of various tools by people with domain knowledge and expertise. This slows down the analysis work considerably and delays the rapid and on-site response to increasing threats. In recent years, with technological developments, especially with the increasing processing capacity of computers and the development of calculation methods based on graphics processing units, machine learning and deep learning studies have gained momentum and become popular in many fields. With AlexNet[8] a convolutional neural network which surpassed its machine learning rivals in the ImageNet[9] a competition on image classification, deep learning applications have gained more popularity in many fields and applications. These new methods and approaches also find their place in Android malware analysis studies.

Convolutional neural networks (CNN), which is a deep learning approach also used in static analysis malware analysis studies, started with LeNeT[10], which was first developed by Yann LeCun et al. In a simple ESA architecture, the image to be introduced is transformed into a matrix, which passes through the convolution layers where the convolution operation is applied. In convolution layers, the features of the image are extracted automatically. The extracted features pass through a s down sampling or up sampling layers. These processes are continued throughout the convolutional layers and differ from architecture to architecture. The resulting feature maps are vectorized in the flatten layer and generate the input values of the fully connected layer. The output layer is a neuron layer with proper activation function to the problem. The visual classification success of CNN algorithms has enabled them to be used not only in visual classification, but also in malware analysis.

In Deep4MalDroid[11], authors extracted smali formatted files from the application files and applied the CNN model on the opcode sequences. In the study, 3 different data sets, 1,260 malicious, 863 benign and 2,475 malicious, 3,627 benign and 9,902 malicious, 9,268 benign were used, and the model obtained 98%, 80%, 87% accuracy, respectively. In MalDozer[12], API call sequences obtained from the ".dex" file were used. Word2vec algorithm was used in the creation of the sequences, and the CNN model was trained with the obtained vector. The model achieved an F1 score of 96.29% on a dataset which contains 37,066 malicious, 37,627 benign samples.

People, who are visual creatures, have transformed the signals they obtained from nature into visual stimuli with various tools. When we think of sound waves, we often think of visual waves that appear on the screen of a music player. Similarly, in oscilloscopes, it provides visualization of electrical signals. Visualization of signals and data in a way that the human eye can perceive with various ways and methods and applying analysis processes on these called visual analysis. Visual analysis studies

are carried out on Android applications by creating images from the whole or part of the application file. CNN models are preferred mostly in the analyzes made on these images because of their proven performance on the image classification and detection. In the study named R2-D2[13] carried out in 2018, classes.dex files extracted from applications which were collected between January 2017 and August 2017. Approximately, 2 million malicious and benign samples were converted into RGB channeled, color images. For coloring the images, the authors used a mapping method that assigns color based on pixel value. In the study, the authors tried the popular CNN models at that time, AlexNet, VGG, GoogleNet, Inception-V3, and achieved the best result with Inception-V3, with 93% accuracy.

Transfer learning approach, which is based on the use of success in one field in another related field, shortens the time spent in the development of new methods and models, and increases the success achieved. In transfer learning, it is aimed to transfer the experience and learning (model weights) gained by the model in a similar problem to a new problem with a little adaptation. With the increase in the number of pre-trained models and the ease of access, transfer learning methods are becoming more preferred in today's studies. In the study conducted in 2020[14], the authors applied the transfer learning method with DenseNet121, DenseNet169, EfficientNetB7, InceptionV3, MobileNet, MobileNetV2, ResNet50, VGG16, VGG19, Xception and their proposed CNN model by converting all application files to grayscale images. The authors, who prepared two types of datasets, used the CICMalDroid2020 dataset to create the dataset. In the data set, which they called balanced which consisted of 10,878 benign and 10,878 malware samples, the CNN model they proposed obtained the highest accuracy with 74%. Although the study used the most CNN models, it was not specified whether models were fine-tuned or not. Model performances can be further increased by fine tuning.

Fine-tuning is a method that is applied at the point where the learning performance of the model does not increase any more. After the model has trained with transfer learning, fine-tuning can help the model to get even better results. In a study called IMCFN[15] conducted in 2020, the authors trained proposed CNN on the ImageNet dataset and then applied transfer learning with fine-tuning method on the datasets consisting of malware images. Both the MalImg dataset, which consists of images of Windows malware, and the Android-IOT dataset, which consists of Android malware, were used for training and testing and samples were classified with 98.82% and 97.35% accuracy, respectively.

### **III. MATERIALS AND PROPOSED METHOD**

#### **A. LAB ENVIRONMENT**

All experiments were performed in the Google Colab environment. Google Colab is a Google service that allows interactive software development with a tool like Jupyter Notebooks. Although Colab distributes hardware resources according to availability, experiments were mostly carried out in an environment which has an Nvidia V100-SXM2 graphics processing unit with 16GB VRAM, 51GB system memory and 8 core virtual processor. Models were created with the Tensorflow/Keras v2.6.0 framework. On Tensorflow version 2 and above, the Keras library is now default for creating models. AndroGuard v3.5.5 was used for reverse engineering of Android applications. While preparing the dataset, NumPy v1.19.5 was used for data processing, and Python Image Library (PIL) was used for creating and sizing the images. Seaborn and Matplotlib were used to visualize the test results and draw plots. In order to compare the test outputs and epoch times, the benchmarking software called AI-Benchmark[16] was run in the experimental environment, and a total score of 34,196 was obtained, this value is 35,086 for the same graphics processing unit in the tool's own rankings. For this reason, we estimate that the epoch times will be close to the physical environment.

## B. DATASET

In this study, we used the CICMalDroid2020 dataset to create our own image dataset. The dataset is one of the most up to date datasets in the time of this study. In the dataset's paper, authors claimed that all the applications have been collected between 2017 to 2018. It has raw application files open to the public which make it better for preprocessing. In the dataset, there are a total of 17,247 applications, including 1,515 Adware, 2,506 Banking, 4,362 Riskware, 4,822 SMS, 4,042 Benign. CICMalDroid2020 dataset is a categorical dataset and categorizes can be summarized as following:

- **Riskware** programs are malicious applications and will not show their malicious intentions when they are first installed. Through remote commands from their developers, they can install Trojan, ransomware, or use social engineering to force device users to behave as expected. This unknown and dangerous nature makes them difficult to detect.
- **Banking** programs are fake versions of real banking applications which try to empty out users' accounts. By imitating banking applications, they try to gain access to victims' bank accounts. As stated in the dataset paper, most of them are based on Trojan horses that can steal sensitive information and send it to the attacker's command and control server.
- **SMS** malware uses mobile phone SMS services to read, write, or intercept SMS traffic. They can send messages from the victims' device, so can subscribe to premium services without victims' knowledge or consent. Since two-factor authentication or online payment methods mostly use SMS as the second factor of verification, these malwares are one of the most troublesome ones.
- **Adware** programs are malicious software that forces user devices to display unwanted advertisements around the interface. Adware can also be part of a legitimate freemium application. At first, they may seem less malicious when compared to other malware programs, but as recent security reports show that they can download and install other types of malware to users' mobile phones.
- **Benign** applications are the legitimate applications that can be downloaded from the official Android market or third-party stores. As stated in the dataset paper, all benign applications were scanned with VirusTotal by authors.

To use in CNN models, the selected dataset must be transformed to image based one. The steps for transforming dataset to image based one can be summarized as following:

1. First, classes.dex files of applications were extracted from raw APK files with Androguard reverse engineering tool. Dex files are dalvik executable files that have the main logic and codes of the applications.
2. With Numpy, classes.dex files were read byte by byte, as each byte becomes a pixel. Square root of the size of the .dex file was used for the one dimension of the square shaped image matrices. If the square root of file size is smaller than 224, then the minimum dimension is set to 224. Empty cells in the matrices filled by zeros.
3. By using the Python Image Library tool, raw byte matrices were saved as three-channels (RGB) .jpeg files. In this step, all the applications in the dataset were transformed to grayscale jpeg images. The main reason for images to be grayscale is that this method copies one channel (for example red) to others (green, blue).
4. Lastly, because dex files sizes differ from each other, images are in different sizes too. To be used in CNN models, Lanczos algorithm was used to down sample images to 224x224 image sizes.

Examples of final images of the dataset have shown in Figure 2. It's hard for the human eye to recognize the patterns. But benign application patterns seem to be clearer to others. There were 464 corrupted application files in the CICMalDroid2020 dataset which cannot be extracted with the Androguard tool. After preprocessing, 16,783 grayscale images have been created. We named this image dataset as DroidMallmg and made it public[2]. Dataset has been divided into three parts: train, validation, and test. We have randomly sampled 80% of the images for each category and created a

train set, 20% for the validation set and 20% of the validation set sampled for the test set. The distribution of the data set is summarized in Table 1.

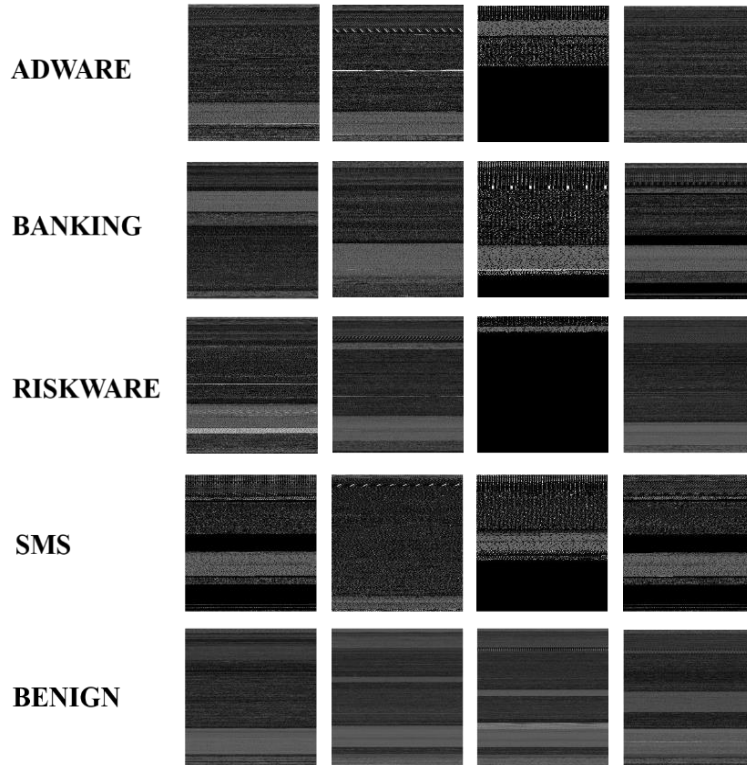


Figure 2. Example images of dataset.

Table 1. DroidMailImg dataset distribution.

Category	Train Set	Validation Set	Test Set	Total
Adware	1,211	242	61	1,514
Banking	2,003	400	101	2,504
Riskware	3,124	624	157	3,905
SMS	3,856	772	193	4,821
Benign	3,231	646	162	4,039
<b>Total</b>	<b>13,425</b>	<b>2,684</b>	<b>674</b>	<b>16,783</b>

### C. PROPOSED METHOD: BATCH FINE-TUNE TRANSFER LEARNING

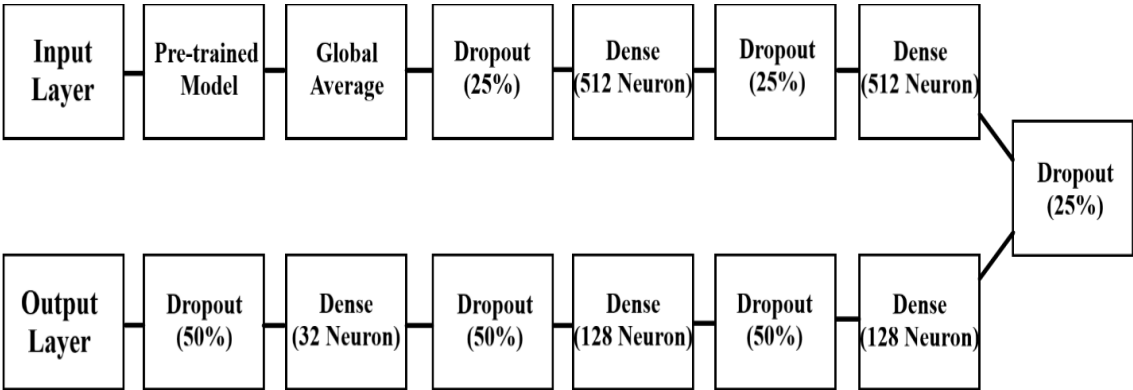
CNN models are one of the popular research areas in deep learning applications. Many of them are publicly available, pre-trained and their numbers are increasing as new models have been proposed. At the time of this writing, there are 25 models publicly available in Tensorflow/Keras library. As the numbers increase it could be useful in some situations to apply the same fine-tune transfer learning method on them, instead of one model. One example for this is comparison, benchmarking of fine-tune transfer learning capabilities of models in the research area. As the number of models increases, it will be better to see which model can do better before investing more time. Another usage could be

ensemble of models, fine-tuned models can be fused together to get even better performance. For these reasons, a method for applying fine-tune transfer learning on more than one model was proposed. The method has two stages, transfer learning and fine-tuning.

**C.1. Transfer Learning**

Transfer learning is widely used in deep learning applications today. It depends on transferring one domain knowledge to other similar domains. Also transforming one domain to another is a useful method which we use in this paper. By transforming applications to images, malware classification problem turns into an image classification problem.

To use our proposed method, first, we randomly selected a model from the model list which is Xception. We do basic hyper-parameter tuning and customization in this model. Important thing is that tuning and customization should not overfit the model randomly selected. We loaded weights which were pre-trained in ImageNet to the model. Pre-trained base model’s weights were frozen which means training can’t change them. We remove the top layer (output) of the model then replace it with our customized layers (Global average layer, dense layers and dropouts) as shown in Figure 3. ReLU activation function used for fully connected layer neurons and softmax activation function used for output layer. Nadam, which is Adam with Nesterov momentum, was used as model optimizer. Learning rate was set to 0.001. Between neuron layers, a dropout layer was used to avoid overfitting. All image inputs have been preprocessed according to the model needs using Tensorflow. This basic customization and hyper-parameter values applied to all the models in the list. In this stage, only custom added layers are trainable. All the models in the list were trained for 999 epoch with a batch size of 32. Validation accuracy was monitored for early stopping and if not increased in 20 epochs, training was stopped.



*Figure 3. Customized model architecture.*

**C.2. Fine-Tuning**

In the fine-tuning stage, some of the frozen layers of pre-trained models unfreeze. This makes layers trainable again. Main problem in batch fine-tuning is how to unfreeze layers in all models without doing it by hand. If we set a rule to unfreeze the last 5 layers of a pre-trained base model, small models may have a big impact, but other large models may not. To overcome this issue, we proposed a new hyper-parameter value specific to this method which is unfreeze rate. The layers to be unfreeze are determined by multiplying the number of layers of the model by the unfreeze rate. In this paper, the unfreeze rate was set to 15% which means If a model has 100 layers, the last 15 layers will be unfreeze. If the number isn’t an integer, we ceil it to an integer. After unfreezing layers of the models, we trained all the models with the same settings as stage one, 999 epoch, 20 epoch early stopping and batch size is 32. Same optimizer (Nadam) in stage one was used but the learning rate was lowered 0.0001. The number of layers and parameters that are trained in this stage is shown in Table 2.



*Table 2. Base models and fine tune information.*

<b>Base Model</b>	<b>Total Layers of Base Model</b>	<b>Trainable Layers of Base Model</b>	<b>Total Parameters of Base Model</b>	<b>Total Parameters (with custom layers)</b>	<b>Fine-tuned total parameters (with custom layers)</b>
<b>Xception[17]</b>	132	20	20,861,480	22,259,693	8,724,589
<b>VGG16[18]</b>	19	3	14,714,688	15,326,469	5,331,397
<b>VGG19</b>	22	4	20,024,384	20,636,165	7,691,205
<b>ResNet50V2[19]</b>	190	29	23,564,800	24,963,013	13,482,437
<b>InceptionV3[20]</b>	311	47	21,802,784	23,200,997	7,719,557
<b>InceptionResNetV2[21]</b>	780	117	54,336,736	55,472,805	19,508,325
<b>MobileNet[22]</b>	86	13	3,228,864	4,102,789	2,467,781
<b>DenseNet121[23]</b>	427	65	7,037,504	7,911,429	2,220,933
<b>NASNetMobile[24]</b>	769	116	4,269,716	5,160,025	2,486,453
<b>EfficientNetB1[25]</b>	339	51	6,575,239	7,580,236	4,507,509

## **IV. MATERIALS AND PROPOSED METHOD**

In this section, models have been evaluated by two different approaches. The first approach is to evaluate the categorical classification performance of the models. Since the basis of the study was categorical classification, the sparse categorical accuracy metric was used as the evaluation parameter. The difference from the categorical accuracy metric is that its calculations are not one-hot encoded but use integer output values. The second approach we follow is the adaptation of categorical classification to binary classification which is malware detection. For malware detection, the main issue is whether the analyzed application software is malicious or benign. Riskware, banking, SMS, Adware applications are collected in one class as malware and benign ones in another class. In this way, detection performances of categorical classification models can also be understood. It should be considered that these detection performances are transformations and different results can be obtained if the model is trained for detection from the beginning. Also models fine-tune performance with the proposed method has been evaluated in this section.

### **A. FINE-TUNE PERFORMANCE OF MODELS**

On Figure 4, the point where the fine-tuning training starts is shown with a green line. As seen, the effect of this method on each model is different from each other. The two most positively affected models were InceptionResNetV2 and DenseNet121, with increases of 5.9% and 5% on validation dataset accuracy. On the other hand, the result of the method on MobileNet, VGG16 models was destructive and negatively affected the performance of the models. To avoid this situation, the best model values obtained so far were saved as model weights, and these saved weights were not changed

unless the model reached a better point. In this way, the models were not affected by the destructive effect of the method, so they can only benefit from the positive effect in the fine-tune stage. Another way could be tuning the unfreeze rate of the method. As a result of the training, the models are shown in Table 3 in order, according to the categorical classification accuracy on the validation data set. In the same table, test dataset accuracies are also shown. If the test set wasn't used the best model seems to be Xception with the validation accuracy of 93.62%, but the best model on the test dataset is DenseNet121 with 94.51% accuracy. For better decision, malware detection performance of the models could be checked. The detection performances of the models are shown in Table 4,5. In malware detection, DenseNet121 has clearly shown better performance on both test and validation datasets. As the problem changes, the best model for the problem seems to be changed.

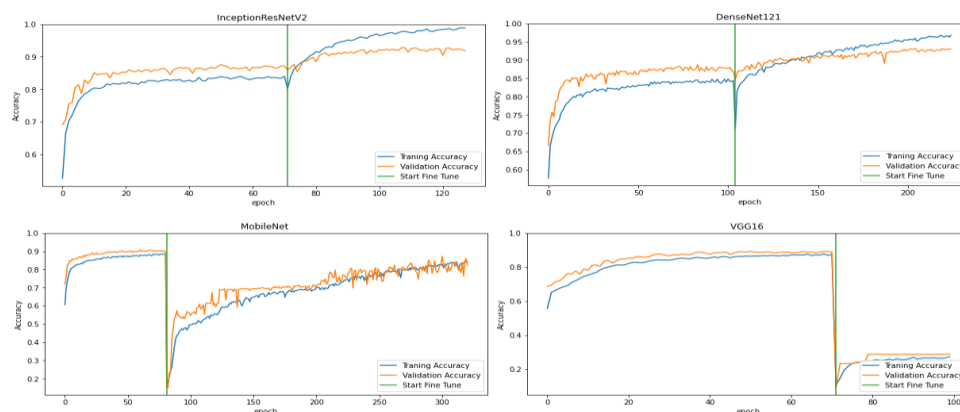


Figure 4. Fine-tune effect on performance of models.

Table 3. Categorical classification performance of models on validation and test dataset.

Model	Transfer Learning Validation Accuracy (%)	Fine-Tune Validation Accuracy (%)	Fine-Tune Test Accuracy (%)	Transfer Learning Total Epochs	Transfer Learning Epoch Time (sec)	Fine Tune Epochs	Fine Tune Epoch Time (sec)
<b>Xception</b>	92.06	<b>93.62</b>	93.91	120	25	81	30
<b>EfficientNetB1</b>	89.41	93.36	93.91	127	19	91	24
<b>VGG19</b>	89.49	93.25	93.17	89	25	162	28
<b>DenseNet121</b>	88.74	93.18	<b>94.51</b>	104	21	121	27
<b>ResNet50V2</b>	91.35	93.10	93.91	94	19	94	24
<b>InceptionResNetV2</b>	87.59	92.80	92.13	71	39	57	48
<b>InceptionV3</b>	88.52	92.43	91.24	94	18	62	21
<b>NASNetMobile</b>	90.64	92.25	92.58	129	20	62	29
<b>MobileNet</b>	90.64	90.64	90.94	81	10	240	11
<b>VGG16</b>	89.23	89.23	90.65	71	21	29	23

*Table 4. Malware detection performance on validation dataset.*

Model	F1 score	Accuracy	Precision	Recall	True Negative Rate	False Negative Rate	False Positive Rate
	(%)	(%)	(%)	(%)	(%)	(%)	(%)
Xception	98.50	97.72	98.47	98.52	95.20	1.47	4.79
EfficientNetB1	98.45	97.65	98.38	98.52	94.89	1.47	5.10
VGG19	98.52	97.76	98.76	98.28	96.13	1.71	3.86
DenseNet121	<b>98.57</b>	<b>97.83</b>	<b>98.86</b>	98.28	<b>96.43</b>	1.71	<b>3.56</b>
ResNet50V2	98.30	97.42	98.28	98.33	94.58	1.66	5.41
InceptionResNetV2	98.38	97.54	98.38	98.38	94.89	1.61	5.10
InceptionV3	98.41	97.57	98.09	<b>98.72</b>	93.96	<b>1.27</b>	6.03
NASNetMobile	98.00	96.98	98.32	97.69	94.73	2.30	5.26
MobileNet	97.94	96.90	98.65	97.25	95.82	2.74	4.17
VGG16	97.81	96.68	97.69	97.93	92.72	2.06	7.27

*Table 5. Malware detection performance on test dataset.*

Model	F1 score	Accuracy	Precision	Recall	True Negative Rate	False Negative Rate	False Positive Rate
	(%)	(%)	(%)	(%)	(%)	(%)	(%)
Xception	97.85	96.73	97.66	98.04	92.59	1.95	7.40
EfficientNetB1	97.76	96.58	97.10	98.43	90.74	1.56	9.25
VGG19	98.05	97.03	97.49	<b>98.63</b>	91.97	<b>1.36</b>	8.02
DenseNet121	98.05	97.03	<b>97.85</b>	98.24	<b>93.20</b>	1.75	<b>6.79</b>
ResNet50V2	97.57	96.29	97.09	98.04	90.74	1.95	9.25
InceptionResNetV2	97.46	96.14	97.27	97.65	91.35	2.34	8.64
InceptionV3	97.66	96.43	97.28	98.04	91.35	1.95	8.64
NASNetMobile	97.56	96.29	97.46	97.65	91.97	2.34	8.02
MobileNet	96.65	94.95	97.42	95.89	91.97	4.10	8.02
VGG16	97.17	95.69	96.89	97.46	90.12	2.53	9.87

## **V. CONCLUSION**

The number of devices using the Android operating system is increasing day by day, the security update support of the devices lasts for about 3 years for each Android version, but even if the relevant version receives a security update, these updates are not offered by the device manufacturers or are offered late. All these devices are seen as profitable targets for attackers. As stated in the study, these vulnerable devices pose a potential threat in cyberspace, just as old satellites threaten space. Against these threats, studies using machine learning and deep learning are becoming widespread. Deep learning applications, with their ability to extract features, may be more suitable solutions.

As public and pre-trained deep learning models increase, selecting a model for a problem becomes harder. To overcome this issue, we proposed a batch fine-tune transfer learning method which can be used to compare models for their fine-tune transfer learning capabilities in that domain. This method can give researchers, developers insight about overall performance of the models in the problem domain. Before investing more time to model, seeing overall performance could be a good start. In our domain as Android malware analysis, with the proposed method, we trained and compared popular pre-trained CNN models. Both Xception and DenseNet121 got high accuracy results in categorical classification and detection on validation and test datasets.

## **VI. REFERENCES**

- [1] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi and A. A. Ghorbani, "Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning," *2020 IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 2020, pp. 515-522.
- [2] T. Kural, Y. Sönmez and M. Dener. (2021, September 5). *DroidMalImg Dataset* [Online]. Available:[https://drive.google.com/drive/folders/1b70zhVMEnlfv2UC\\_56Q9PzwXUhSXw58u?usp=sharing](https://drive.google.com/drive/folders/1b70zhVMEnlfv2UC_56Q9PzwXUhSXw58u?usp=sharing)
- [3] Kaspersky. (2010, August 9). *First SMS trojan detected for smartphones running Android* [Online]. Available: [https://www.kaspersky.com/about/press-releases/2010\\_first-sms-trojan-detected-for-smartphones-running-android](https://www.kaspersky.com/about/press-releases/2010_first-sms-trojan-detected-for-smartphones-running-android).
- [4] Malwarebytes. (2021). *State of Malware* [Online]. Available: [https://go.malwarebytes.com/rs/805-USG-300/images/MWB\\_StateOfMalwareReport2021.pdf](https://go.malwarebytes.com/rs/805-USG-300/images/MWB_StateOfMalwareReport2021.pdf).
- [5] A. Cranz. (2021, May 18). *There are over 3 billion active Android devices* [Online]. Available: <https://www.theverge.com/2021/5/18/22440813/android-devices-active-number-smartphones-google-2021>.
- [6] Google. (2021). *Google transparency report* [Online]. Available: <https://transparencyreport.google.com/>.
- [7] S. O'Dea. (2021, Jun 30). *Mobile Android version share Worldwide 2018-2021* [Online]. Available: <https://www.statista.com/statistics/921152/mobile-android-version-share-worldwide/>.
- [8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NIPS)*, Lake Tahoe, NV, United States, 2012, pp. 1097-1105.

- [9] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, United States, 2009, pp. 248-255.
- [10] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 1998.
- [11] S. Hou, A. Saas, L. Chen and Y. Ye, "Deep4MalDroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs," *2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW)*, Omaha, NE, United States, 2016, pp. 104-111.
- [12] E. M. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digital Investigation*, vol. 24, pp. S48-59, 2018.
- [13] T. H. Huang and H. Kao, "R2-D2: ColoR-inspired Convolutional NeuRal Network (CNN)-based Android Malware Detections," *2018 IEEE International Conference on Big Data (Big Data)*, Seattle, WA, United States, 2018, pp. 2633-2642.
- [14] M. Al-Fawa'reh, A. Saif, M. T. Jafar and A. Elhassan, "Malware Detection by Eating a Whole APK," *2020 15th International Conference for Internet Technology and Secured Transactions (ICITST)*, London, United Kingdom, 2020, pp. 1-7.
- [15] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, "IMCFN: Image-based Malware classification using fine-tuned convolutional neural network architecture," *Computer Networks*, vol. 171, p. 107138, 2020.
- [16] A. Ignatov, R. Timofte, A. Kulik, S. Yang, K. Wang, F. Baum, M. Wu, L. Xu, and L. Van Gool, "AI Benchmark: All About Deep Learning on Smartphones in 2019," *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Seoul, Korea, 2019, pp. 3617-3635.
- [17] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, United States, 2017, pp. 1800-1807.
- [18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *The 3rd International Conference on Learning Representations (ICLR2015)*, San Diego, CA, United States, 2015, pp. 1-14.
- [19] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," *European conference on computer vision (ECCV)*, Amsterdam, The Netherlands, 2016, pp. 630-645.
- [20] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception architecture for computer vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, NV, United States, 2016, pp. 2818-2826.
- [21] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," *In Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, CA, United States, 2017, pp. 4278-4284.
- [22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

- [23] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, United States, 2017, pp. 4700-4708.
- [24] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Salt Lake City, UT, United States, 2018, pp. 8697-8710.
- [25] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," *International Conference on Machine Learning (PMLR)*, Long Beach, CA, United States, 2019, pp. 6105-6144.