Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A DEEP LEARNING BASED TECHNIQUE FOR THE CLASSIFICATION OF MALWARE IMAGES

MD. HARIS UDDIN SHARIF[1], NASMIN JIWANI[2], KETAN GUPTA[3], MEHMOOD ALI MOHAMMED[4], DR.MERAJ FARHEEN ANSARI[5]

[1]School of Computer & Information Sciences, University of the Cumberlands, KY, 40769, USA
[2]School of Computer & Information Sciences, University of the Cumberlands, KY, 40769, USA
[3]School of Computer & Information Sciences, University of the Cumberlands, KY, 40769, USA
[4]School of Computer & Information Sciences, University of the Cumberlands, KY, 40769, USA
[5]School of Computer & Information Sciences, University of the Cumberlands, KY, 40769, USA

E-mail : [1]haris.uddin.sharif@gmail.com, [2]Nasminjiwani@gmail.com, [3]ketan1722@gmail.com, [4]mehmood.db9@gmail.com, [5] merajfarheenansari25@gmail.com

ABSTRACT

Because of the fast expansion of the internet and technology, a slew of developing malware and attack techniques has evolved. As a result, researchers concentrated their efforts on machine learning and deep learning techniques to detect malware. Many organizations have been developing new algorithms and products to secure people from these scams. On the other hand, Malware kinds have been expanding substantially in recent years. The anti-virus companies have been discovering millions of new malware variants every year. Therefore, new intelligent malware detection methods must be solved as soon as possible to halt this rise. Malware is becoming more prevalent, more diverse, and more sophisticated. Deep learning in malware detection through images has recently been demonstrated to be highly effective. We also employed an Image-based Malware dataset [Malimg] and used the different deep learning algorithms, CNN, Caps-Net, VGG16, ResNet, and InceptionV3, for malware detection. The dataset images were transported through the pre-processing pipeline and into the deep learning pipeline, where they were used to train deep learning models in the right way. As part of the model training process, all images were resized to be the same size and proportions. A factor of 1/255 was then applied to the images, resulting in a conversion from RGB value to grayscale, which restored the original RGB values to their correct positions. Later, the dataset was segmented into two groups, train, and test. The VGG16, ResNet50, and InceptionV3 models detected the malware images. A combination of the Adam optimizer and the cross-entropy loss function was used to train all of the models. The models were trained for 50 epochs using early stopping criteria. Finally, the model composition method was used to classify malware images where the previously trained models were combined. The custom CNN model, the VGG16, ResNet50, and InceptionV3 models were combined to predict a single outcome for the experimental condition. The proposed technique provided very promising results.

Keywords: *Malware Prediction, VGG16, ResNet50, Caps-Net, Image-Based Malware Prediction, Cyber Analysis, Deep Learning, Cyber Security*

Proposed Acronyms

EC	= Ensemble Classifier	IoT	= Internet of Things
ANN	= Artificial Neural Network	VGG16	=Visual Geometry Group
Caps-Net	= Capsule Network	KDD	= Knowledge Discovery Databases
CNN	= Convolutional Neural Networks	KNN	= K Nearest Neighbor
CVA	= Cross Validation Accuracy	KNNA	= K-Nearest Neighboring Algorithm
DDoS	= Distributed Denial of Service	LASSO	= Least Absolute Shrinkage and Selection
DL	= Deep Learning	Operator	
DoS	= Denial of Service	LSTM	= Long Short-Term Memory
DPA	= Deep Learning Algorithm	ML	= Machine Learning

DT	= Decision Tree	MLA	= Machine Learning Algorithms
EC	= Ensemble Classifier	MLP	= Multi-Layer Perceptron
ECA	= Ensemble Classification Algorithme	NCF	= network connection features
FAR	= False Alarm Rate	NFS	= Network Features Selection
FN	= False Negative	NN	= Neural Network
FP	= False Positive	PCA	= Principal Component Analysis
FSS	= Feature Selection System	Probe	= Probing Attack
GA	= Genetic Algorithm	PSO	= Particle Swarm Optimization
HELAD	= A new incompatibility detection model called HELAD	RFE	= Recursive Feature Elimination
		R2L	= Remote to Local
IDS	= Intrusion Detection System	RC	= Random Classifier
SVM	= Support Vector Machine	RF	= Random Forest
TML	= Traditional Machine Learning	SNN	= Standard Neural Networks
TN	= True Negative	TP	= True Positive
TNS	= Traditional neural networks	U2R	= User to Root

1. INTRODUCTION

In Attacks on the internet have increased exponentially, and malware has emerged as one of the most severe threats to network security. According to a recent study, millions of sensors regularly capture millions of harmful threat events per second [1]. In parallel with the rise in popularity of mobile devices and IoT, malware has also increased prevalence. Globally, according to the most recent threat reports, the number of users who faced Android malware increased by more than 1.7 million. Viruses and malware are among the most significant security dangers facing internet users. Malware is defined as any type of harmful code that can compromise a digital system's integrity, confidentiality, and operation [2]. Malware is divided into several categories by its functions, including Trojans, worms, and backdoors. These classes are further subdivided into families based on the sort of variations that are present in them. When creating variations of an existing malware family, malware authors employ various obfuscation techniques, including code transposition, subroutine reordering, and code insertion, to ensure that the infections remain undetected [3]. Discovering malware variations is the most challenging aspect of internet security. Numerous malware versions, such as Nuwar, Storm, and Kekihos, have characteristics, implying that the same malware developer generated them.

According to Symantec estimates, millions of malwares have been discovered, and the number is growing continuously. Criminals have also begun to conduct crimes online rather than in a person. Criminals typically employ malicious software to initiate cyberattacks against victim computers. Antimalware systems from the past are frequently inadequate in coping with today's diversity and amount of malware. Malware analysis is a rapidly increasing discipline that requires considerable attention due to technological advancement in social networks, mobile environment, cloud computing, the Internet of things (IoT), and the industrial Internet of Things (IIoT). It was created for simple goals in the early stages of malware development, making it easier to detect. This type of malware is referred to as conventional malware. However, malware that can function in kernel mode and is more damaging and difficult to detect than typical malware might be classified as next-generation malware these days. This type of malware is extremely adept at bypassing security software that runs in kernel modes, such as firewalls and antivirus software. Generally, classic malware is composed of a single process and does not employ sophisticated strategies for concealment. On the other hand, new generation malware runs numerous existing or new processes concurrently and employs various obfuscation techniques to conceal itself and establish a lasting presence in the system. This new-generation malware can launch more devastating operations such as targeted and persistent attacks that have never been seen before, and the attacks employ many types of malwares. The frequency, sophistication, and cost of malware attacks on the global economy have been steadily growing in recent years. According to scientific and industry reports, over 1 million malware files are developed daily, and cybercrime is expected to cost the global economy approximately $6 trillion annually by 2021 [4].

The recent research also indicated that mobile malware is increasing in popularity. According to McAfee's mobile threat report, backdoors and banking Trojans targeting mobile devices have increased significantly [5]. Additionally, malware assaults targeting social media platforms, healthcare, cloud computing, and Cryptocurrency are growing. The malware must be discovered to protect genuine users and businesses from it. Malware detection is

the process of identifying whether or not a certain application is harmful. Most modern malware detection systems rely significantly on the antivirus software containing signature databases to detect dangerous patterns, which is a feature of most antivirus software.

On the other hand, such software is incapable of detecting new or undiscovered malware. Furthermore, these technologies have their own set of limitations, such as the inability to identify malware that has been packaged or encrypted. Even the simple reusability of code with some packers can create a new form of malware capable of evading signature-based detection systems. Because of these constraints, signature-based approaches cannot detect the majority of packed malware, which allows it to remain undiscovered for an extended period. Static and dynamic analysis approaches are the two most often utilized techniques for malware identification and prevention.

Additionally, a hybrid technique for malware detection that incorporates both static and dynamic analysis is being developed and tested. And have also been investigated [5]. Static code analysis is time-consuming and depends significantly on reverse-engineering malware to complete successfully. Code obfuscation is a significant difficulty with static analysis, and one of the disadvantages of dynamic analysis is that it depends on the execution environment to expose its whole behavior. Over time, researchers offered novel ways to detect, including behavioral, heuristic, and model checking-based approaches.

Due to these approaches, data mining and machine learning methods are also being increasingly employed in malware detection. Recently, novel ways to detect have been proposed, including those based on deep learning, cloud computing, mobile devices, and the Internet of things. Heuristic detection approaches also effectively detect known malware and certain undiscovered malware. Behaviours, model checking, and cloud-based techniques, on the other hand, outperform traditional approaches when dealing with unknown and sophisticated malware. Deep learning, mobile devices, and Internet of Things (IoT)-based techniques are also being developed to identify a fraction of known and undiscovered malware. However, it has not been demonstrated conclusively that one detection method is more successful than the others. This is because each approach has its own set of benefits and weaknesses, and under particular situations, one method may detect more accurately than another.

1.1 Importance of The Study

Feature engineering, feature selection, and representation approaches are used to develop machine learning algorithms. The set of characteristics associated with a matching class is used to train a model, which is then utilized to generate a dividing plane between benign and malicious objects. This dividing plane aids in detecting malware and the classification of malware into its associated malware family. Both feature engineering and feature selection methods necessitate a thorough understanding of the domain. Static and dynamic analysis may both be used to determine the various characteristics. Static analysis is a technique for capturing information from a binary program without running it on the computer. Dynamic analysis is the practice of observing malware activity in real-time while running in a controlled environment. Dynamic analysis has the potential to be an effective long-term solution for malware detection systems. In real-time malware detection, dynamic analysis cannot be used since it requires a significant amount of time to evaluate the activity of the infection. A harmful payload might be delivered during this time, making it ineffective.

Compared to statically gathered data, malware detection approaches based on dynamic analysis are more resistant to obfuscation methods than statically obtained data. In most cases, commercial anti-malware solutions employ a combination of static and dynamic analysis methodologies to detect and remove the malware. Traditional machine learning-based malware detection systems have a significant drawback. They are heavily reliant on approaches such as feature engineering and learning and feature representation methodologies, which need deep domain expertise. Furthermore, if an attacker becomes familiar with the characteristics, the malware detection may readily have circumvented [6].

Machine Learning algorithms require data with a range of malware patterns to be successful. Because of security and privacy issues, there is extremely little publicly available benchmark data for malware analysis research.

Even though just a few datasets are available, each comes with its own set of scratching comments, as most of them are out of date. Many of the results of machine learning-based malware analysis that have been published have been based on the author's datasets. Even though there are publically available sources for crawling malware datasets, building a quality dataset for the study is time-consuming and difficult. Because of these challenges, establishing a general machine learning-based malware analysis system that can be deployed in real-time has been

hampered in recent years. More importantly, the participants explored the compelling concerns associated with using data science approaches [7]. Deep learning, which is the more advanced model of neural networks, has recently outperformed traditional machine learning algorithms in many tasks in the fields of natural language processing, computer vision, and many others [8].

In recent years, a novel technique for malware detection based on mage visualization [9]- [10] has been investigated by several researchers to discriminate between malicious programs. Visualization-based techniques decrease the requirement for domain specialists and eliminate the need for manual feature engineering, resulting in time savings. A Convolutional Neural Network (CNN) architecture is a type of neural network that can extract information from an input image without human intervention. Furthermore, well-defined CNN architecture like ResNet 50, Inception V3, and AlexNet may be used as feature extractors and classifiers [11]- [12]. These networks are trained on huge datasets of images. Without further training, they may be utilized as a classifier for comparable classification issues in the target domain, such as malware image classification. The transfer learning-based malware classification approach has been examined in several existing works of literature. Pre-trained networks with well-established topologies may be used either as feature extractors in conjunction with the machine learning methods for classification on the target domain [11]- [12] or fine-tuning a classification model for the target domain [13]. However, only a few researchers have addressed the issue of overfitting when working with the smaller or unbalanced datasets to date. Competitive advantage is provided to the algorithm by the early stopping regularization approach, which allows it to halt the model's training process based on validation data results. It is also computationally challenging to execute a whole deep CNN model on a short dataset because of many variables. Because of an early stopping approach, the model converges quickly and efficiently without imposing additional burden on the training process, resulting in a computationally efficient model. Retraining the pre-trained model with several convolutional layers is time-consuming and impractical for use.

1.2 Contribution of The Study

The researchers developed and assessed various machine learning and deep learning algorithms to increase their productivity, which was often used in conjunction with the information reduction technique. On the other hand, these algorithms have shown favorable outcomes when a set of assessment measures has been used. On the other hand, such models are worthless when identifying malware in real-world networks. In this field, there was a tendency to focus on exceeding specular results for a particular dataset rather than delving further into machine learning-based virus identification models. Several studies have been conducted in a real-world context due to this reaction. Though, these approaches are troublesome since they are often assessed using just one dataset with such a consistent list of qualities, which may not be practical to collect or maintain in an actual network communication stream, they are beneficial in theory. The further point is that, due to machine learning and deep learning, there is occasionally room for improvements. in hyperparameters when various datasets are allocated to the same model. In this research study

- We proposed a self-sufficient model in terms of a wide range of advantages and trained it using a Malimg –An image-based dataset.
- We performed a malware image-based classification using custom CNN, and their pre-trained networks.
- We first investigated Capsule Network's performance (Caps-Net) for malware image classification.

2. RELATED WORKS

To protect computer systems from malware, we must first detect malware before it can cause damage to the computer systems. When it comes to detecting malware, three classic ways have been used: behavior-based, heuristic-based, and signature-based detection. There are lots of pros and cons to using these strategies. Signature-based detection effectively identifies known malware by pattern matching. Still, it is ineffective at identifying unknown malware because malware can alter its properties, resulting in a new signing this method cannot detect. While this method can identify both well-known and unknown malware, it has the potential to produce high error rates for both false-positive and false-negative results. The behavior of suspicious files is observed by approaches based on behavior-based malware detection. Resources and time are required for this approach to be implemented and monitored to be effective. In this part, we highlighted the advantages and disadvantages of some of the prominent classification models used for malware detection, which typically relied on static and dynamic analysis and their variants in more recent years. However,

when dealing with a large amount of data, it is even more critical to consider image processing techniques to improve data visualization and make more informed decisions.

2.1 Malware Classification Using Statistical Analysis

Some security researchers have utilized domain-level understanding of portable executables to identify static malware in their work. Analysis of byte-n grams and strings are now the two most often used approaches for static malware detection that do not require domain-level expertise to be effective. The n-gram technique, on the other hand, is computationally costly, and the performance is well below average. When developing a machine learning model to discriminate between malware and benign file, it is frequently challenging to apply domain-level knowledge to extract the essential features. Similarly, with the continually changing specifications from time to time, the malware detection system will need to be updated to fulfill further security needs. In a study [14], the authors have attempted to address this issue by combining machine learning algorithms with features derived from the parsed information in the PE file. They used formatting of agnostic characteristics such as byte entropy histogram, raw byte histogram, and string extraction. They also have made a dataset containing features, raw files, and related code available to the public since deep learning models require more examination and investigation. Similarly, wholly linked classical networks and recurrent neural networks were also used to detect malware using 300 bytes of information [15].

2.2 Malware Classification Using Dynamic Analysis

The dynamic malware analysis method is more resistant to obfuscation techniques than static malware analysis approaches, which are more used in the industry. In a research study [16], features from API calls were extracted and fed to CNN for classification purposes using dynamic analysis. They employed around 170 samples and acquired a quality measure of 0.96- AUC as a result. In another study [17], the authors said that they had gotten a shallow feed-forward network feature set of API requests from a large number of benign and malicious samples that had been gathered privately. It outperforms the previous technique in terms of performance, but it does not include research on execution speed, which is critical for real-time deployments of software. In [18], a study of the echo state networks (ESN) and recurrent neural networks (RNNs) were carried out to understand the language of malware. Compared to RNNs, the ESNs outperformed them in most of the studies. The study [19] was carried out to establish when to terminate the virus execution about the network connectivity being used. The overall time required by this procedure was 67 percent shorter than the time required by traditional methods. An RNN and its variants, long short-term memory (LSTM), and CNN were used for malware classification in [20], employing API call long sequence as features while CNN was used for classification. The most significant disadvantage of the approach was that they required more time to evaluate the system's behavior as it was being executed. This combination reported very promising results. It was also discovered through dynamic analysis that these system calls were made, and their technique was shown to outperform previously utilized algorithms such as HMM and Support vector machines. However, the most significant shortcoming was the lack of considering the significance of execution time in the context of malware detection in real-time. Multiple studies have been conducted to examine the effectiveness of malware detection strategies based on static, dynamic, and hybrid analysis methodologies. There was comparison research on detection rates and the usage of HMM on both static and dynamic feature sets in [21], which included a large number of malware families and included both static and dynamic analysis of feature sets. Their finds revealed that dynamic analysis often provided the highest detection rates.

2.3 Malware Images

It is possible to describe malware executables as a matrix of binary or hexadecimal strings, which may be translated into a form that can be thought of as an image. Malware developers typically add to or update the code in existing malware to produce a new variant. As a result, when the file structure is displayed as an image, it is much easier to see minute addition or modifications to various areas of the file structure. Initially suggested by [22], this approach for converting malware into graphics involves converting raw bytecode PE files to grayscale image data, where each pixel is represented by one or more bits. Similarly, in [23], the authors accomplished image-based malware classification utilizing an ensemble CNN architecture to identify packed and unpacked malware files. In another work [24], malware binary in IoT contexts was transformed into an image, and CNN's was utilized to classify the malware families. The proposed technique reported 94% accuracy for

goodware and DDoS malware and 81.8% accuracy for goodware malware. In [25], the researchers presented the MCSC model, i.e., Malware Classification Using SimHash and CNN. They hashed decompiled malware code and transformed it into grayscale images before training CNNs to classify malware. The proposed technique was validated on malware image samples and reported 98.86% classification accuracy.

2.4 Malware Classification Using Image Processing Techniques

Malicious software assaults are on the rise, and in recent years, new malware may be simply created by modifying existing malware from a well-known malware family in a straightforward manner. To address this challenge, it is necessary to become familiar with the features of malware that are similar to one another and may be used to group malware into families. Several research studies [26]- [28] exploited the fact that most malware variants are similar in structure, employing digital signal and image processing techniques to classify malware. They converted malware codes to grayscale images and discovered that malware belonging to the same malware family appears to be pretty similar structure and texture. Because Image processing techniques do not involve disassembly or code execution, they are significantly quicker than static and dynamic analysis. The primary advantage of this strategy is that it can handle compressed malware and can work with a wide variety of malware regardless of the operating system. Experimental results indicate a classification accuracy of 98% when applied to an extensive malware database and is also resistant to typical obfuscation techniques, such as encryption. They also proposed an Image-based dataset Malimg for malware classification. In these studies, the researchers also demonstrated Search and Retrieval of Malware, an online search, and retrieval system that analyzed binary executables using similarity metrics.

They also demonstrated signal, a signal processing-based system for detecting malware similarities. It can handle both packed and unpacked samples, bypassing the resource-intensive unpacking step. Recently, the Malimg dataset has been utilized to compare the efficacy of advanced machine learning algorithms to that of traditional machine learning algorithms. Rather than relying on various signal and image processing approaches, the application of deep learning algorithms is translated into malware classification using the Maligmg dataset [29]- [30]. Similarly, SVM combined with deep learning architecture such as CNN and RNN variants were also explored in [29] and reported very promising results.

2.5 Malware Classification Based on Deep Learning

Deep learning is used to learn the properties of malware and benign files by analyzing large datasets. Deep learning has been employed in various domains, including speech recognition and image recognition, as an effective artificial intelligence [30]. For instance, [31] developed MCSC, a malware classification approach that combines visualization and deep learning techniques. They extracted the Opcode commands from the malware executable and then encrypted them using SimHash. They transformed SimHash values into grayscale images by converting them into pixels. Finally, the Convolutional Neural Network was employed to train the images, and malware families were identified. The proposed solution produced a high degree of classification accuracy in small-scale application settings but could not detect malware more quickly in a large-scale application environment.

In [32], the author's presented a deep learning approach without relying on reverse engineering. Their approach obtained a classification accuracy of 98.2 percent using just 10860 samples from nine malware families. Similarly, in [33], a combination of CNN and LSTM was employed to automatically learn the characteristics from infected files. It significantly decreased the cost of developing artificial features. The proposed technique achieved a classification accuracy of 99.36% using only 10,860 samples from nine malware families. Another research work [34] suggested an architecture based on CNN's for classifying malware samples. They conducted research on the most complex malware dataset known as Mailing. While analyzing 9339 samples from 25 malware families, their design achieved 98.52% accuracy. They tested 10% of the samples within a family at random. [35] introduced a deep learning architecture for malware detection based on CNNs. They conducted experiments on the Malimg dataset and considered 25 malware families. The proposed model achieved a 98% accuracy rate when applied to 9339 samples. The experiment randomly picked 10% of the family's samples. The concept of deep learning was also employed in another research study [36]. They used various deep learning techniques, including CNN architecture, to identify intrusion in both network-based instruction detection systems and host-based intrusion detection systems, with a claimed accuracy of over 98%. The suggested

approach does not provide sufficient information on the malware's structure and properties and does not account for overhead time.

In [37], the author discussed using the image-based technique for identifying suspicious system activity and advocated using hybrid image-based approaches in conjunction with CNN-based deep learning architecture for successful malware classifications. They presented two- CNN-based models, Unidirectional GRU and Bidirectional GRU, and then assessed and compared their performance to other current CNN architectures such as Unidirectional LSTM and Bidirectional LSTM. They conducted an experiment using two publicly available datasets: Microsoft Malware Classification challenge and Mailing. The proposed architecture reported an average accuracy of 96% but did not account for overhead time. Several researchers suggested using data balancing approaches to minimize the possibility of malware detection misclassification. [38], developed a weighted Softmax loss algorithm to balance the imbalance malware dataset.

Similarly, [39] suggested a CNN-based malware variant identification technique. Additionally, they resolved the data imbalance issue by utilizing a BAT method for data equilibrium. While analyzing 9339 samples from 25 malware families, their approach achieved a 94.5% classification accuracy. A cost-effective solution was utilized to address the unbalanced multiclass malware family issue. Recently, security researchers have begun forecasting the image classification problem like a malware classification [36]. The models of CNN, such as VGG -16, ResNet -50, and Inception V2, have been implemented for the intrusion detection system. The focus of this study is an Image-based malware classification using these models.

3. PROPOSED METHODOLOGY

The All of the dataset images were transferred through the pre-processing pipeline and into the deep learning pipeline for appropriate training of deep learning models. To train the model, all images were shrunk to the same proportions as each other. Following that, the images were rescaled from RGB value to grayscale by a factor of 1/255 to bring the RGB values back into balance. They are considered too high for good model performance if they fall within the range of 0 – 255. Finally, the dataset was subdivided into two groups: the training and test sets. Malware images were distributed in two sets: a training set and a test set with a 70 percent training set and a 30% test set. Many deep learning models for malware classification were trained when a malware images dataset was prepared and made available to the researchers. The Adam optimizer and the cross-entropy loss function were employed for the proposed custom CNN architecture. The default value for the remaining hyperparameters was utilized for the rest of the parameters. The classifier's performance was improved by reducing the learning rate on the plateau, which was enabled by default. A further step was taken to train three distinct deep learning models to classify Malware Images. These models were labeled as VGG-16, ResNet 50, and Inception V3. The Adam optimizer and the cross-entropy loss function were combined to train all models. Using early halting criteria, the models were trained for 50 epochs. With the help of the evaluation measures, the results of all trained models were compared. Finally, the classification of malware images the model composition method was employed. The composite model was created by combining several previously trained models. The custom CNN models and the VGG 16, ResNet 50, and InceptionV3 models were merged to predict a single outcome for the experiment. An overview of the proposed methodology is presented in figure 1. The detailed employed dataset and proposed architecture are presented in the subsequent section.

Table 1. Algorithm

Algorithm Composite Model Algorithm for The Classification	
Input: Sample image of Malimg dataset	
Output: Class or Malware type of image	
Step 1:	Sample-image = load ("path of the testing set image)
Step 2:	Result1 = custom-CNN. Predict (Sample-image)
Step 3:	Result2 = VGG16. Predict (Sample-image)
Step 4:	Result3 = ResNet50. Predict (Sample-image)
Step 5:	Result4 = InceptionV3. Predict (Sample-image)
Step 6:	Final-Result = max* (Result1, Result2, Result3, Result4)
Step 7:	Return Final-Result

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

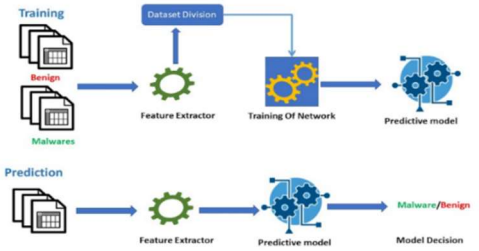

Fig. 1 An overview of Proposed Methodology

3.1 Dataset

It is necessary to construct a large dataset with various samples to evaluate the effectiveness of traditional machine learning and deep learning architecture. Because of the privacy-preserving practices of individuals and organizations, there are extremely few public datasets for possible cyber security studies for malware detection purposes. As malware has grown sophisticated, finding a single source containing all the different malware families has become increasingly difficult. Many researchers have attempted various efforts to build the dataset and collaborated on their findings; however, there is currently no one dataset that has been published where all of the necessary samples can be found. In this study, the Malimg dataset is used to classify Malwares. The Malimg dataset was acquired from the Kaggle repository, which comprises mostly 9458 malware samples that have been classified into 25 different classes. The most notable characteristic of this dataset is not supplying only malware samples once but are also providing images of malware samples as they appear on the disc.

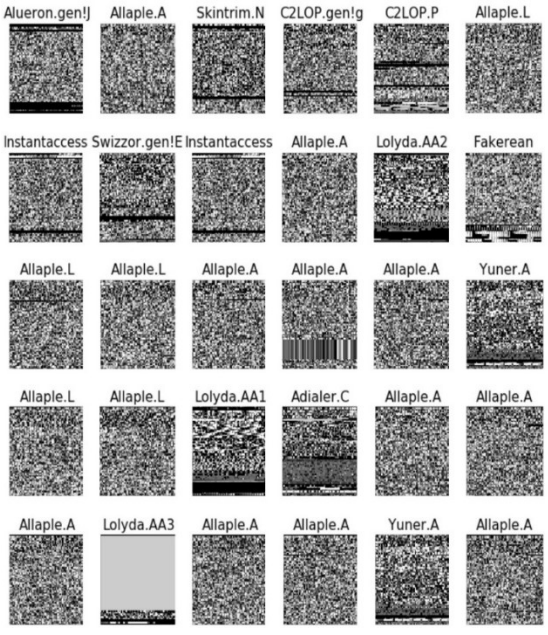

Fig. 2 A sample Images of Malimg Dataset

The work in bytes of executable le files is analogous to the work in that floats are allocated inconsequentially to values that would be later be revealed as pixel values of the grayscale image. The malware classes in the dataset are unequally distributed, with the bulk of samples belonging to the class 'Allaple. A' comprises 2949 samples, and the least number of samples belonging to the lowest class, which contains just 80 samples. The random samples of the malware image dataset have been presented in Figure 1. The images in each category have distinct styles that allow distinguishing between the samples of a family, regardless of whether or not they are examples of another family in the same class. After the collection of the samples, we performed image preprocessing. It is an important stage in the development of a classification system. Removing any unnecessary information from images during preprocessing is necessary to improve the classification rate. The preprocessing is carried out by the types of image that has been received. Image processing steps such as noise removal, skew correction, and binarization. After the preprocessing steps, we prepared the training data of malware images and fed them to the deep neural networks for feature extraction and classification.

3.2 VGG- 16

VGG-16 is a Convolutional Neural Networks (CNN) architecture that is simple and widely used in visual object classification and detection research. Initially, It was used for ImageNet, a big database project utilized in object recognition software research. Karen Simonyan and Andrew Zisserman from the University of Oxford developed and introduced the VGG 16 architecture. The term 'VGG' stands for Visual Geometry Group, a group of researchers at the University of Oxford that worked together to build this architecture. The number 16 indicates that this architecture comprises 16 layers of information.

The VGG-16 model achieved 92.7% classification accuracy in the ImageNet dataset, which contained 14 million images belonging to the 1000 different classes. One of the most well-known models submitted to the ImageNet Large Scale Visual Recognition Challenge 2014. It improved the AlexNet design by substituting large kernel filters with three-three kernel-sized filters one after another in the first and second convolutional layers, respectively. The VGG 16 has been employed in different deep learning classification problems due to its simplicity of implementation because a very small 3 * 3 filter size was used throughout the whole

network, with a stride of 1 pixel being used throughout the network. In the previous network, such as AlexNet, this was 11 * 11 with stride 4, and the same field in ZFNet was 7* 7 with stride 2. The concept of using 3 *3 filters uniformly distinguishes the VGG. Two consecutive 3 * 3 filters produce an effective receptive field of 5 *5 due to the combination of the three filters.

Similarly, three 3 * 3 filters can create a receptive field 7 * 7 by combining them. It is possible to substitute for a large receptive area by combining numerous 3 * 3filters in this manner. The advantage of this arrangement is, as an alternative to the one non-linear activation layer that would present id 7*7, there are three non-linear activation layers in addition to the three convolutional layers. As a result, the decision-making functions become more discriminative. It provides the network with the potential to converge at a faster rate. Second, it has the additional benefit of greatly reducing the number of weight parameters which lessen the likelihood that the network will become overfitting during the training session. According to the architecture of VGG- 16, It was assumed that the input to the network was a fixed size image with three channels, RGB with a resolution of 224 X 224 pixels. The only pre-processing that has been done is to normalize the RGB values for each individual pixel.

This was accomplished by removing the mean value from each pixel in the image. A 3 * 3 receptive size image was sent through the first stack of two convolutional layers with a receptive size of 3* 3, after which Relu activation functions were performed. Each of these two levels has a total of 64 filters in it. The convolution stride and padding were fixed at one pixel, and the padding was fixed at one pixel. This arrangement keeps the spatial resolution of the image intact, and the output activation map size was the same as the dimension of the input image. The activation maps were then run via spatial max-pooling layer over a 2 * 2-pixel window with a stride of 2 pixels. After this, the activations were then routed through a second stack similar to the first stack but with 128 filters instead of 64 filters in the first stack. There were three Convolutional layers and a max-pooling layer in this stack, followed by the fourth stack.

Fig. 3 VGG -16 Architecture for Malware Image Classification

Similarly, the 256 filters were used in the fourth stack, and so on. In the end, the stacks of convolutional layers followed the three fully connected layers with a flatting layer in between. The employed VGG-16 architecture in this study for malware image classification is presented in Figure 3.

Compared to the AlexNet, the VGG-16 architecture was an upgrade since it replaced the large kernel size filter with various 3 * 3 kernel sized filters that applied into stack manners. When working with the pre-trained Networks, two approaches can be used, features extract action and fine-tuning. We used VGG16 for feature extraction as well as for classification purposes. It is distinctive in that, rather than having many hyper-parameters, it contains convolutional layers of 3 * 3filter with a stride 1 and always utilizes the same padding and max pool layer. We initialized the model checkpoints during training and prepared the malware image size according to the network input.

In the end, adjust the value of neurons at the Fully connected layer as, by default, it corresponded from 1000 classes of ImageNet.

3.3 Resnet 50

AlexNet earned the first prize in the LSVRC2012 classification challenge in 2012, and since then, ResNet has been the most intriguing thing to happen in the world of computer vision and deep learning. Because of the foundation that ResNet provides, it became possible to train extremely deep neural networks, which means that a network can have thousands of layers while still achieving excellent performance, a previously impossible feat. The ResNet was initially applied to the image recognition problems, but as reported in the literature, this framework was extended to various tasks other than computer vision to obtain higher accuracy and greater precision. As we know, the deep Convolutional Neural networks are extremely good at identifying low, mild, and high-level features in images, and stacking more layers generally results in better accuracy. The question arises as to whether increasing the number of layers will improve model performance? With this question comes the problem of vanishing gradients, which has been addressed in various ways and has enabled networks with tens of layers to converge. However, when the deep neural networks converge, another problem arises: the accuracy becomes saturated and degrades rapidly. This was not caused by the overfitting, as one might expect, and adding more layers to a suitable deep model only increased the training error. To address this further, the researchers employed a shallower model and deeper model, both of which were constructed with layers from the shallow model and identified layers added on the top of them. As a result, the deeper model should not have produced any training error because the added layers were only the identity layers. To overcome this issue, a deep residual learning architecture was introduced where the author proposed a shortcut connection that merely performed identity mappings. Because there were no additional parameters introduced to the model due to this shortcut identity mapping, it was possible to keep the computing time under control. As mentioned earlier, the Residual Network design was selected as the winner of the ILSVRC competition, and jamming was the one that invented ResNet. To do this, he set out to create ultra-deep networks that were not affected by the vanishing gradient problem that has plagued previous generations of the networks.

ResNet employs a variety of layer counts, including 34, 50, 101, 152, and even 120 layers in some instances. ResNet-50, a convolutional neural network with 50 layers, is one of the versions of ResNet. A total of 48 convolution layers are included and 1 Max pooling and 1 Average pooling layer. The ResNet 50 is a deep residual learning framework built on a neural network. It can resolve the vanishing gradient problem even when working with incredibly dense neural networks. ResNet 50, despite the fact it contains 50 layers, has around 23 million trainable parameters, which are significantly less than the trainable parameters of previous architectures. Even if the explanations behind its performance are still up for debate, the most straightforward method to comprehend it is to describe residual blocks and how they function, as presented in figure 4.

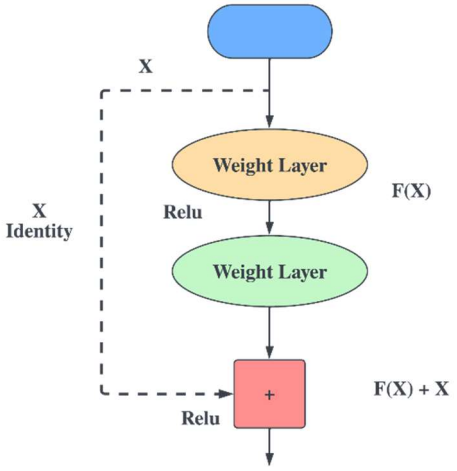

Fig. 4 Residual Learning Block

Suppose, a residual block has y as an input and wants to learn the true distribution H(y). The difference between input and true learning can be write as

$$R(y) = Output - Input$$
$$R(y) = H(y) - y$$

After the rearranging this equation we will have.

$$H(y) = R(y) + y$$

The residual block is attempted to figure out what the genuine output is H(y); as the residual block has an identity link arising as a result of they, the layers are learning the residual, which is represented by the letter R(y). When using a standard network, the layers are responsible for learning the true output H(y), but the layers of the residual network are responsible for learning the residual R(y).

Fig. 5 The Architecture Of The Resnet 50 Is Illustrated

Furthermore, it has been shown that learning the residual of the output and input, rather than just the inputs, is more straightforward. Thus because they have been bypassed and do not add any complexity to the design, the residual identity model allows for the reuse of activation functions from earlier levels [40]. The architecture of the ResNet 50 is illustrated in figure 5.

In this research, we used ResNet 50 architecture for the features extraction and classification of Malware images with different experimental settings and tuned on a various number of epochs to achieve better results.

3.4 Inception V3

Since Krizhevesky et al. [41] won the 2012 image completion, their network "AlexNet" has been successfully applied to a broader range of computer vision tasks, including object detection and video classification. In the wake of these breakthroughs, researchers began a new line of investigations into improving the performance of convolutional neural networks. With the implementation of deeper and larger networks beginning in 2014, the quality of network topologies has improved dramatically. In the 2014 ILSVRC challenge, the VGGNet [42] and Google Net [43] networks achieved a similar level of performance. It was shown that the improvements in classification tend to translate into considerable quality improvements across a wide range of application areas, which was the fascinating findings. This indicated that architectural advances in deep learning could be used to increase the performance of a wide range of other computer vision applications that are becoming increasingly reliant on high-quality, learned visual features, such as object detection and tracking. Improvements in network quality have also led to the development of new application areas for convolutional networks in different circumstances where the AlexNet feature could not compete with hand-engineered features.

Even though VGGNet offered the appealing virtue of architectural simplicity, this appears at a hefty cost: assessing the network needs a significant amount of computational power, which is not always available. On the other hand, Google Net [43] was built with the inception architecture in mind, and it was meant to perform well even when faced with stringent memory and computation power. Compared to AlexNet, which used 60 million parameters, Google Net used only 5 parameters, representing a 12 percent reduction in parameter usage. Furthermore, VGGNet used approximately three times as many parameters as AlexNet. In addition, the computational cost of Inception is far cheaper than that of VGG Net or its more powerful descendants. This has made it possible to use Inception networks in computer vision and big data tasks, where a large amount of data needs to be processed at a reasonable cost, or scenarios where memory or processing capability is fundamentally constrained. In this study, we employed the 3rd version of Inception named Inception V3 for malware image classification.

Through the modifications to the previous Inception architectures, InceptionV3 strives to consume fewer processing resources. Inception V3 has shown to be more computationally efficient than VGGNet, both in terms of the number of parameters generated by the network and the cost incurred. Care must be taken while making changes to an Inception network not to lose computational gains.

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Consequently, it turns out to be challenging to adapt Inception networks to diverse use cases because of the ambiguity around the new network's efficiency. Many strategies have been proposed for improving the network in an Inception v3 model to make the model more adaptable. Parallel calculations, regularization, dimension reduction, and factorized convolutions are just a few of them. According to the architecture, the Inception v3 contains 48 layers consisting of different parameters such as factorized convolutions, smaller convolutions, Asymmetric convolutions, Grid Size reduction, and auxiliary classifier.

The factorized convolutions are used in the architecture to reduce the number of parameters in a network. This helps to improve computational efficiency. It is also used to monitor the network's efficiency. Similarly, the large convolutions are replaced with smaller convolutions, which results in faster training. The grid size reduction is also employed in Inception v3, where pooling techniques are commonly used to reduce the grid size. However, a more effective strategy is given to overcome the computational cost bottleneck. A small CNN is placed between layers during training as an auxiliary classifier, and the loss it incurs is added to the net loss. This classifier works as a regularizer in Inception v3 [44].

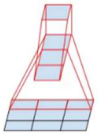

*Fig. 6 3*3 smaller convolutions*

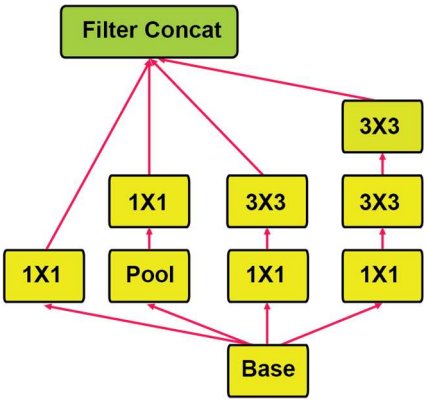

Fig. 7 Inception Module where two 3 3 convolutions replace each 5 * 5 convolution.*

Fig. 8 Inception V3 architecture for Malware Image classification.

3.5 Convolutional Neural Network (CNN)

Even though Convolutional Neural Networks were introduced for the first time in the early 1990s [45], they did not gain much attention from the research and academic community due to the scarcity of large datasets, the complexity of the algorithms, and the length of time required for training. Large datasets such as ImageNet [46] and the introduction of astonishingly effective GPUs have combined significantly improved the performance on a wide range of learning tasks while simultaneously reducing the training time. As a result of Krizhevesky et al. [47]'s usage of CNNs in the Image Net competition, CNNs have received widespread acceptance. The CNN-based system

outperformed those based on traditional methodologies, but it also had a much lower effort rate. The use of neural networks for recognition tasks has exploded since then, and they have been applied to a wide range of pattern classification tasks. Traditional networks require input in a single vector; however, the fully connected design results in a considerable number of weights per neuron due to the fully linked topology. Overfitting occurs as a result of the fact that such networks do not scale well for image-based data. On the other hand, CNN's are optimized for image recognition and classification since they are predicated on the assumption that the input is mostly an image. Because each neuron is connected to a specific part of the image, the number of weights associated with a neuron is significantly reduced compared to a fully connected architecture.

In CNN's, the neurons are arranged according to the three dimensions of height, width, and depth. A CNN is composed mainly of convolutional and pooling layers placed on top of one another, followed by the fully connected layer. In CNN's, the convolutional layers are the central component that extracts a distinct feature from the image to create a more accurate representation. The output of the convolution process is an activation map, which is then passed on to the next layer of the algorithm structure. When the early convolutional layers detect low-level features, the following convolutional layers combine these features to discover the high-level features. Each filter in a convolutional layer creates an individual activation map or features to calculate output volume. Most of the time, a non-linearity is introduced into the network by applying an activation function to the output of the convolutional layers. In the activation function, the ReLu is the most usually employed. The vanishing gradient problem with the standard sigmoid function is effectively avoided by ReLu, despite its simplicity. The ReLu can be defined as

$$R(x) = max\ (0,\ x)$$

When the convolutional layers are added repeatedly, the pooling layer is introduced to down sample the feature maps. It is important to note that the down sampling procedure not only helps to lower the number of parameters in the network, but it also helps to prevent overfitting. The max-pooling operation is used in the convolutional network. It picks the maximum filter response from among all of the filter responses examined in a particular region of the input volume among the various pooling process utilized. Following a sequence of pooling layers, the feature maps produced by the fully connected layer that serves as the classifier are fed into the fully connected layer. In the FC layer, all neurons are connected to all of the neurons in the preceding layers in the same way.

In addition to the pre-trained network, we employed the CNN and trained it from scratch. The architecture of a CNN is a function of many factors and variables. In this study, we developed and compared various CNN architectures by altering their kernel sizes, their number of layers, and the number of filters within each convolutional layer. The proposed architecture has convolutional 144 * 144 and 71 * 71. After the convolutional layers, the max pooling operation was performed where 71 * 71 and 32 * 32 dimensions with 64 filters were used for the pooling operation. After the pooling layer, the flatten layer was used to convert the data into a feature vector for further classification processing. After the flattening layer, the two dense layers of 128 and 25 neurons were used. As the output generated by the dense layer is an m dimensional vector, thus the dense layers were used to change the vector's dimension. Normally, in the architecture, it is a layer firmly related to its preceding layer and works to change the dimension of the output by performing matrix-vector multiplication. The detail of the proposed architecture of CNN is illustrated in Table II.

Table 2. CNN's Proposed Architecture of Malware Image Classification

Model: "sequential"

Layer (type)	Output Shape	Param #
Conv2d (Conv2D)	(None, 144, 144, 128)	18944
max_pooling2d (MaxPooling2D)	(None, 71, 71, 128)	0
conv2d_1 (Conv2D)	(None, 67, 67 64)	204864
max_pooling2d_1 (Maxpooling2D)	(None, 32, 32, 64)	0
flatten (Flatten)	(None, 65536)	0
dense (Dense)	(None, 128)	8388736
dense_1 (Dense)	(None, 25)	3225

Total params: 8,615,769
Trainable params: 8,615,769
Non-trainable params: 0

3.6 Capsule Neural Network (CAPS – Net)

An example of an artificial neural network is a Caps Net. Because CNN does not have many restrictions for computer vision tasks, The Capsule Network was developed as an alternative to CNN. Convolutional neural networks, or CNNs, were

initially designed to categorize images by merging convolutional and pooling layers. These networks are sometimes referred to as the "cornerstone" in some circles. Even though the CNNs could produce accurate results, there was a noticeable decline in their performance. This resulted from a reduction in the data dimension, which brought about a loss of information. As an alternative to convolutional neural networks, which Geoffrey Hinton invented, he came up with a novel architecture called the capsule neural network [48]. Capsules are used in capsule networks rather than neurons, used in traditional networks. All of the essential information present in an image to generate a vector was contained within the capsules.

In contrast to neurons, which can only output a scalar quantity, tablets can keep tabs on the future path while being monitored. Consequently, if we start moving the features about in their respective positions, the value of the vector will remain the same, but the vector's direction will change to represent the movement of the features. An encoder and a decoder are the two parts that make up a Caps Net, as specified by the design.

When used together, the encoder and decoder make up six levels in the system. To be more specific, encoders transform the input image into a vector with 16 dimensions by using that image as a source. The input image is converted into a 16-dimensional vector by the first three layers of the network's architecture. The convolutional neural network, the primary caps network, and the digital caps network are the three layers that contribute to the construction of the encoder that Caps Net uses. The first layer is responsible for isolating the most fundamental characteristics of the images. The second layer is responsible for taking these essential characteristics and identifying more intricate relationships between them; the capsule sizes available in this layer can vary depending on the dataset being used. The variation in the number of capsules present in the third layer follows the pattern established in the second layer. When figuring out which capsules from the primary caps will be moved to the Digit caps, it is vital to compare and contrast the weight of the lower-level capsules with that of the higher-level capsules. We employed two convolutional and one fully connected layer in the encoder section. The convolutional layer used the ReLu activation function, which uses 256 convolutional kernels of 9x9 size, and the stride is 1. This layer is in charge of converting the intensities of the pixels to the activities of the local feature detector, and the results are subsequently supplied to the Primary Caps layer.

The Primary Caps layer is a convolutional layer that consists of 32 channels of convolutional 8-D capsules. There were 8 convolutional units within each capsule, with a 9 x 9 kernel and a stride of 2. Inverse graphics were created by primary capsules, which means that the process of really creating an image is reverse-engineered by these capsules. A 6*6*8 output tensor was produced due to the capsules applying eight 9*9*256 kernels to the 20*20*256 input volume. In light of the fact that there were 32 8- D capsules, the output would have the dimensions of 6 * 6 * 8 * 32. Each class in the Digit Caps layer is comprised of 16- D capsules, and each capsule gets input from the corresponding low-level capsules. The weight matrix was employed for affine transformation against each 8- D capsule. In the end, an instantiation parameter encoding was accomplished through the utilization of reconstruction loss. When calculating, The loss of each training sample was compared against all of the output classes. The overall loss was calculated by adding up the individual losses of each digit capsule. An overview of the proposed architecture is presented in Figure 9 and Figure 10, respectively.

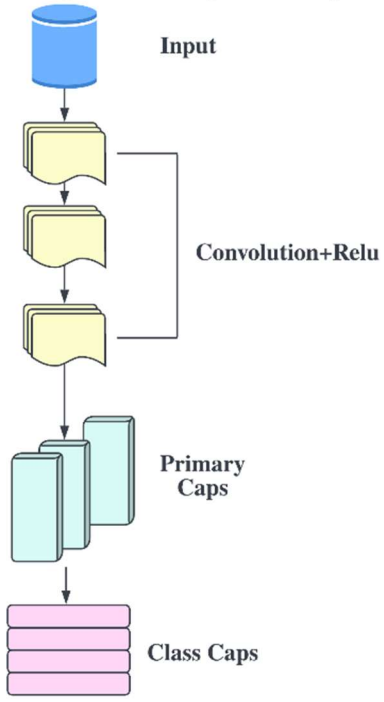

Fig. 9 Architecture of Caps-Net.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 64, 64, 3)	0	
conv_2d_ (Conv2D)	(None, 64, 64, 32)	24608	input_1[0][0]
batch-normalization_1 (BatchNormalization)	(None, 64, 64, 32)	128	conv2D_1[0][0]
activation_1 (Activation)	(None, 64, 64, 32)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 28, 28, 72)	186696	activation_1[0][0]
reshape_1 (Reshape)	(None, 6272, 9)	0	conv2D_2[0][0]
lambda_1 (Lambda)	(None, 6272, 9)	0	reshape_1[0][0]
routing_layer_1 (CapsuleLayer)	(None, 25, 18)	25558400	lambda_1[0][0]
input_2 (InputLayer)	(None, 25)	0	
mask_1 (Mask)	(None, 18)	0	routing_layer_1[0][0] input_2[0][0]
dense_1 (Dense)	(None, 16)	304	mask_1[0][0]
dense_2 (Dense)	(None, 32)	544	dense_1[0][0]
dense_3 (Dense)	(None, 12288)	405504	dense_2[0][0]
output (Length)	(None, 25)	0	routing_layer_1[0][0]
output_recon (Reshape)	(None, 64, 64, 3)	0	dense_3[0][0]

Total params: 26,176,184
Trainable params: 26,019,320
Non-trainable params: 156,864

Fig. 10 Detailed-Architecture of Caps-Net

4. EXPERIMENTAL SETUP

The study was carried out on a hardware computer equipped with 8GB of RAM, a 1TB hard drive, and a GPU with 11G capability. Python version 2.6.10 was installed and configured to design and test the models. The list of used libraries and their purpose are listed below.

4.1 Numpy

NumPy is a numerical python; numerous-dimensional and one-dimensional array items can be computed and manipulated using NumPy. We used this library for data pre-processing.

4.2 Keras

Deep learning API Keras is built on Tensor Flow, a machine learning platform. Keras may be used to train deep neural networks in python. Initially, it was designed to allow for quick testing to occur. It is being created to deliver results as quickly as possible to conduct high-quality research. We used this library in this research to train our proposed models.

4.3 Tensor Flow

In machine learning and artificial intelligence, Tensor Flow is a runtime environment that is entirely open-source and available for free to anybody to use. It can also be put to use in a variety of other situations. Although it focuses on deep neural network-based training and validation, it is not without its limitations.

4.5 Pillow

Pillow is referred to as a Python Imaging Library (PIL), and it allows to view, alter, and save images in the Python programming language. The most recent edition can recognize and handle various file types. Writing assistance is restricted to some of the most extensively used exchanging and presentation formats to achieve this. In this research, we used this library for image preparation and pre-processing.

4.6 Scikit-Learn

Scikit-learn (Sklearn) is a machine learning library written in Python that is widely used and extremely powerful. This library takes advantage of Python consistency API to provide a set of rapid tools for machine learning and statistical modeling, similar classification and prediction, clustering, and data preprocessing, among other things.

4.8 Pandas

Pandas is a Python programming language's data processing and analysis software package. The data formats and methods for processing numeric records and time-series data that are included are particularly relevant. We also used this library for data processing.

4.9 Seaborn

Seaborn is, in fact, a Python module that allows creating statistical visualization. It is made on top of matplotlib and close associates to panda's dataset models. Seborn is a tool that aids in exploring and analyzing information. Its charting units work with data frames and matrices that include the entire set of data, doing the necessary semantic mapping and statistical aggregation within them to create usable graphs and charts.

4.10 Matplotlib

Matplotlib is, in fact, a Python and its extension NumPy-based cross-platform data visualization and graphical charting program that may be used on a variety of systems. As a result, it is a suitable open-source replacement for the MATLAB programming language. Matplotlib APIs can be used to integrate graphs into the graphical user interface and other applications. We used this library for the generation of charts and graphs.

5. EVALUATION MEASURES

The Different evaluation methods were employed to compare the performance of trained CNN models to compare their results. These evaluation measures aim to generate a numeric value for the model in terms of model performance by employing various mathematical formulas and techniques. The accuracy, precision, recall, and F1-score are the evaluation measures that have been chosen.

5.1 Accuracy

The most straightforward and clear performance metric is accuracy, just the proportion of correctly predicted observations to all observations in a dataset. Assuming it is accurate, our model would lead one to believe that it is the best. Yes, accuracy is a relevant statistic, but only when the datasets are symmetric and the number of samples for each class is almost evenly distributed across the datasets. Even though the Malimg dataset does not have a balanced distribution of types, the accuracy of all trained models was assessed for the purpose of a fair comparison.

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

5.2 Precision

In statistics, precision is defined as the proportion of precisely anticipated positive samples over the total number of accurately predicted positive samples.

$$Precision = \frac{TP}{TP+FP}$$

5.3 Recall

The recall rate is the proportion of really predicted positive samples in a class over the total number of positive samples in the class.

$$Recall = \frac{TP}{TP+FN}$$

5.4 F1- Score

The F1 Score is the weighted mean of the true positive rate and the false positive rate, and it takes into consideration both.

$$F1\text{- }Score = \frac{2*(Recall+Precision)}{Recall+Precision}$$

6. RESULTS AND DISCUSSION

This research presented a deep learning approach for malware classification. The Kaggle malware dataset, based on Malware images, was used for this purpose. With a 70 – 30 split, the dataset was divided into training and testing. The Malimg dataset was used for training 70 % of the time, with the remaining 30% used for testing. The Malimg dataset trained many built-in CNN models, including a custom convolutional neural network. Each model's performance was assessed using assessment metrics such as accuracy, precision, recall, and F1-Score. Initially, a custom CNN model based on convolutional layers, Max pooling layers, and dense layers were trained. With default hyperparameter settings, the Adam optimizer and Cross-Entropy were employed. The CNN model revealed a 90% classification accuracy rate. The accuracy, loss and confusion matrix of CNN model is presented below.

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

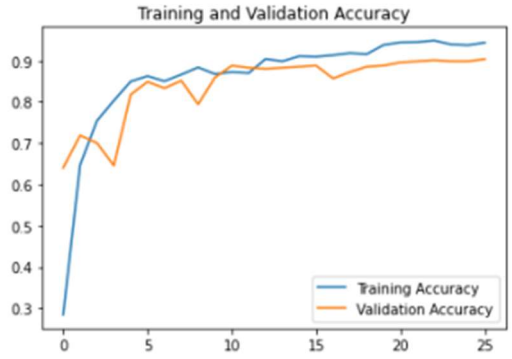

Fig. 10 Accuracy of CNN model during Training.

Fig. 11 Loss of CNN model during Training

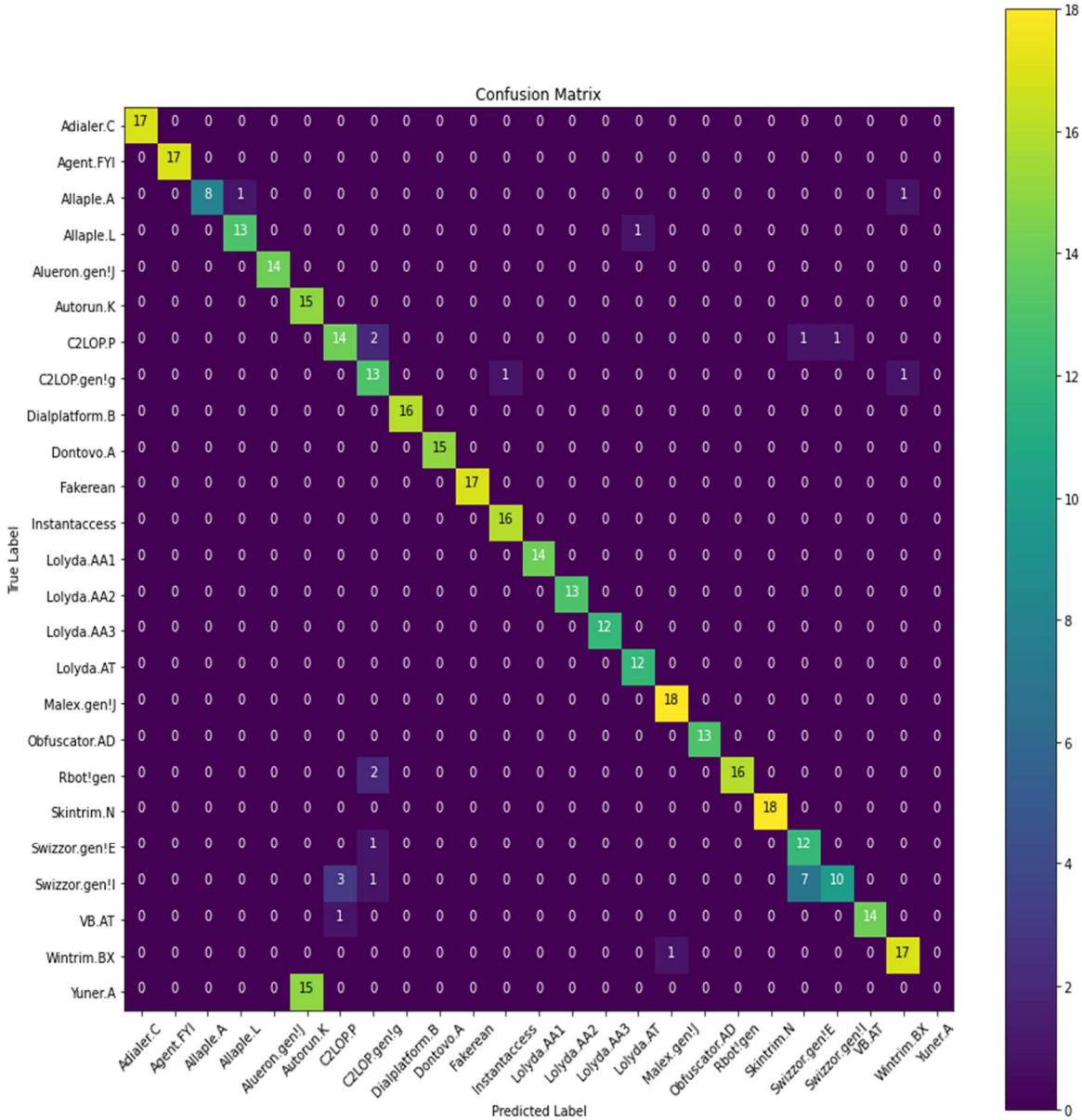

Fig. 12 Confusion Matrix of CNN model

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

One of the most common types of deep learning architecture, the Convolutional neural networks are increasingly being used in various computer vision-related applications. CNNs have defined the outcomes that are considered state of the art in various fields, including image classification, object detection, and segmentation. These networks each have their own set of challenges and obstacles when dealing with particular kinds of images. The CNNs are doomed to failure of they are continually fed images of varying dimensions and orientations. To overcome these problems, the Caps Net was proposed. Instead of doing computations on their inputs as regular neurons do, capsules "encapsulated" the results of those computations into a small vector of highly informative outputs. These sets capsules apart from conventional neurons. When compared to an artificial neuron, a capsule's focus is on a vector, whereas an artificial neuron is concerned with a scaler. A capsule can be thought of as a replacement or alternative for artificial neurons. On the complete set of data, Caps Net stopped training at epoch 30. The training loss in the last epoch reached the value of 0.0436, while the validation loss reached the value of 0.0455. The Caps Net achieved 90% classification accuracy. The training and validation loss for each epoch is presented in Figure 13 and Figure 14.

Fig. 13 Caps- Net Model Training and Validation Loss

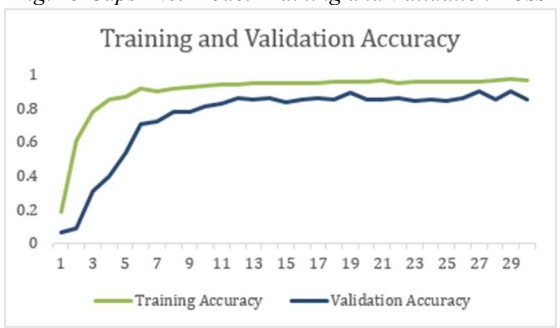

Fig. 14 Caps- Net Model Training and Validation Loss

The malware was then classified using three separate built-in deep learning models (VGG16, ResNet50, and Inception V3). The Malimg dataset was used to train the VGG16 model for malware classification. VGG16 is a convolutional neural network (CNN) design awarded first place in the 2014 ILSVR competition. It is widely recognized as one of the most advanced vision model architectures yet devised. The convolutional and max pool layers were placed in the same way throughout the architecture. For the model's training, the transfer learning technique was applied. Following the model's training and model was evaluated using the testing set. For the test set, the model demonstrated an accuracy of 80%. The accuracy, loss, and confusion matrix are illustrated below.

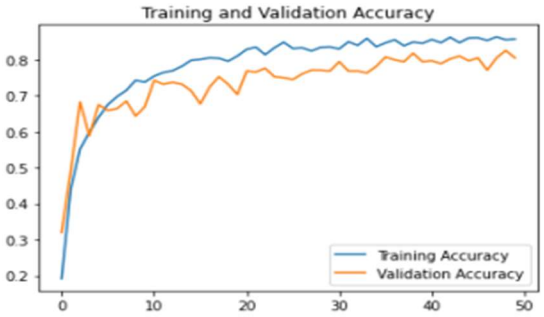

Fig. 15 Accuracy of VGG 16 Model for training and validation set

Fig. 16 Loss of VGG 16 Model for training and validation set

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

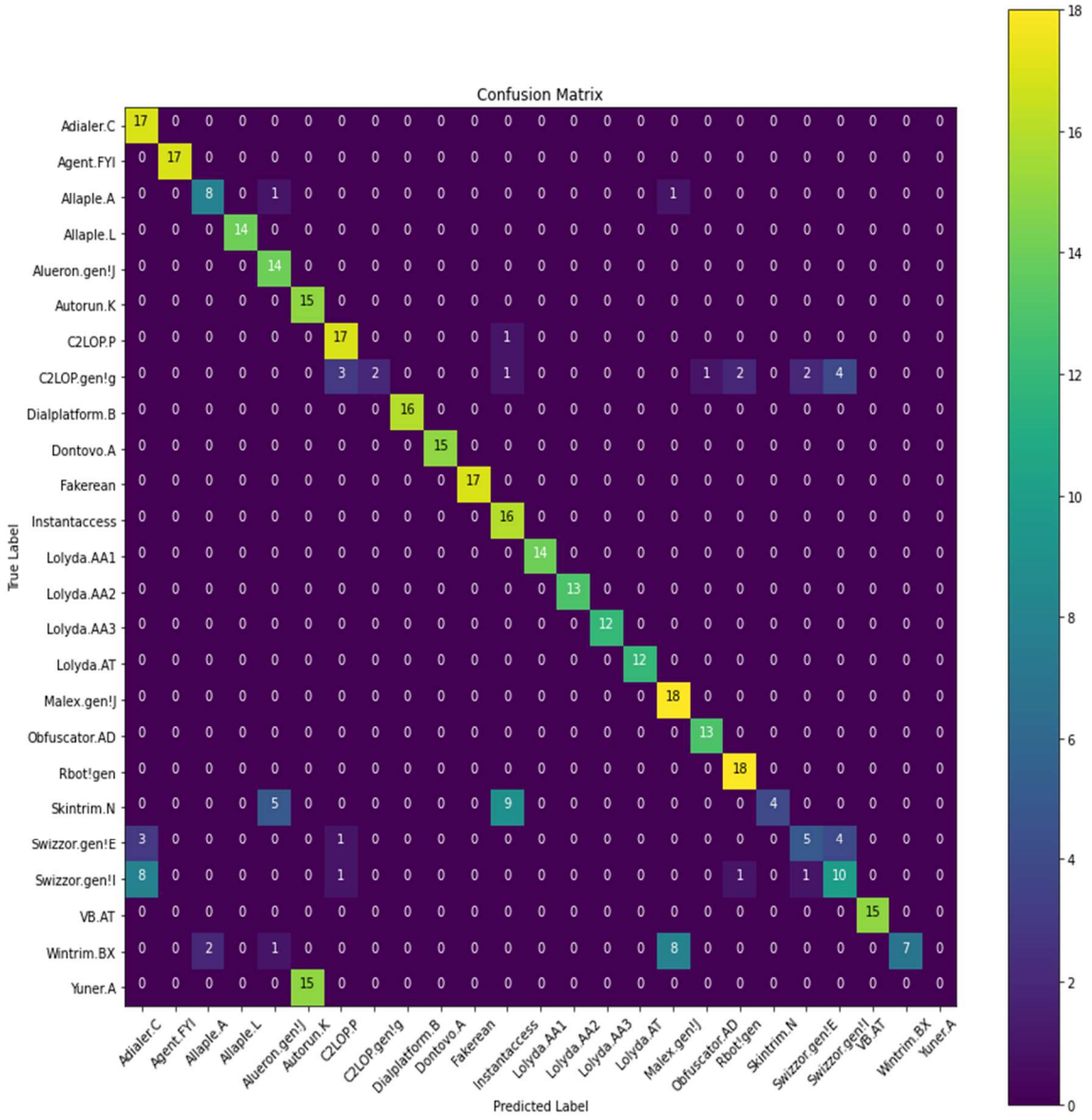

Fig. 17 Confusion Matrix of VGG 16 model for the testing set

After this, the ResNet 50 was then trained with the same hyperparameters for classification. ResNet-50 is a deep CNN model with 50 layers trained on the ImageNet database of 1000 different objects. The ResNet 50 exhibited an accuracy of 81% classification accuracy. The accuracy, loss, and confusion matrix of ResNet 50 is presented below.

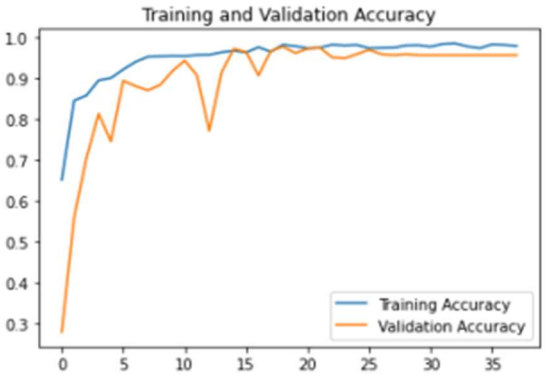

Fig. 18 Accuracy of ResNet 50 model for training and validation set

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

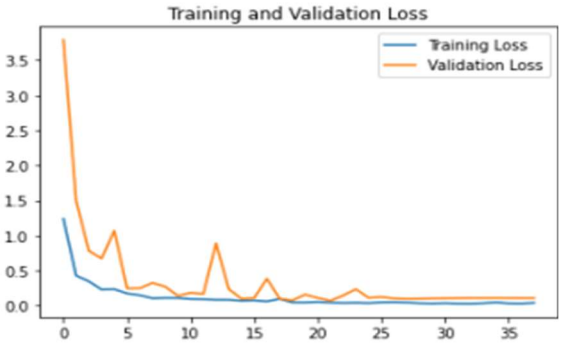

Fig. 19 Loss of ResNet 50 model for training and validation set

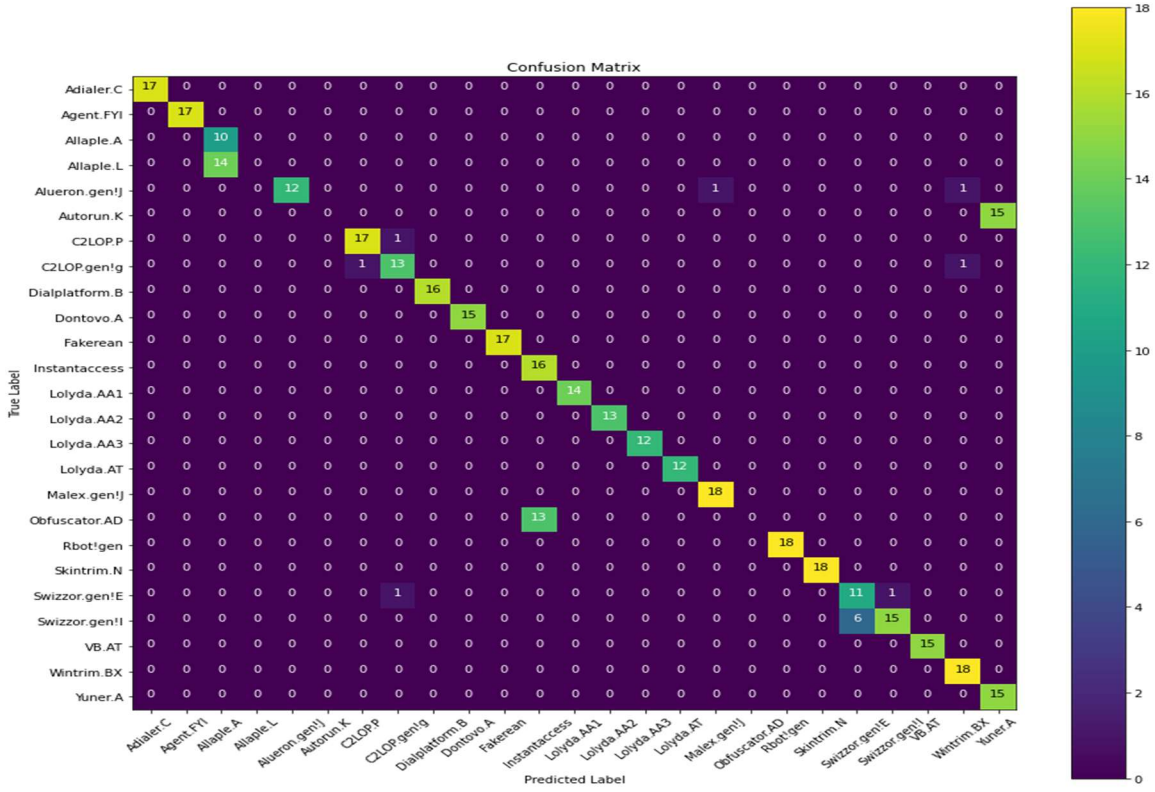

Fig. 20 Confusion Matrix of ResNet 50 model for test set

Inception V3 was the third built-in malware classification model used in this study. Inception V3 is a CNN model that belongs to the Inception family and includes Label Smoothing, factorized 7 * 7 Convolutions, and an extra classification algorithm to transport labeled data deeper down the structure, among their improvements. The Inception V3 computed 87% classification accuracy of malware images. The training, validation accuracy, loss, and confusion matrix are presented in a given section.

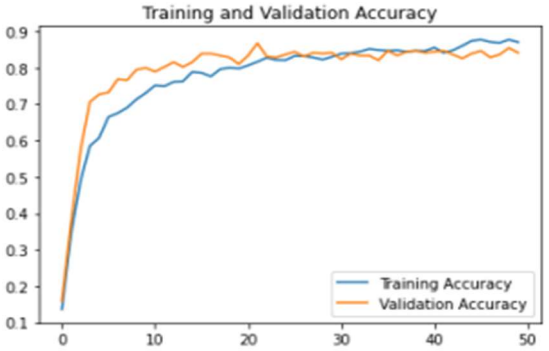

Fig. 21 Accuracy of Inception V3 for training and validation set

Journal of Theoretical and Applied Information Technology
15th January 2023. Vol.101. No 1

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

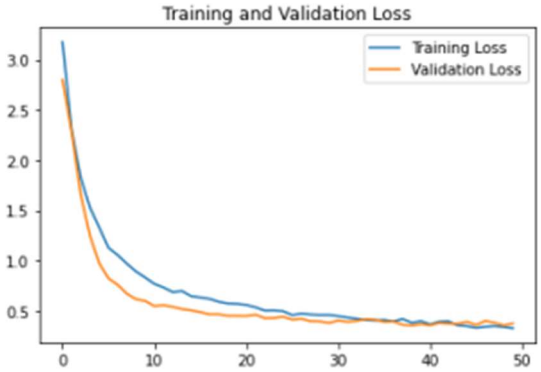

Fig. 22 Loss of Inception V3 model for training and validation set

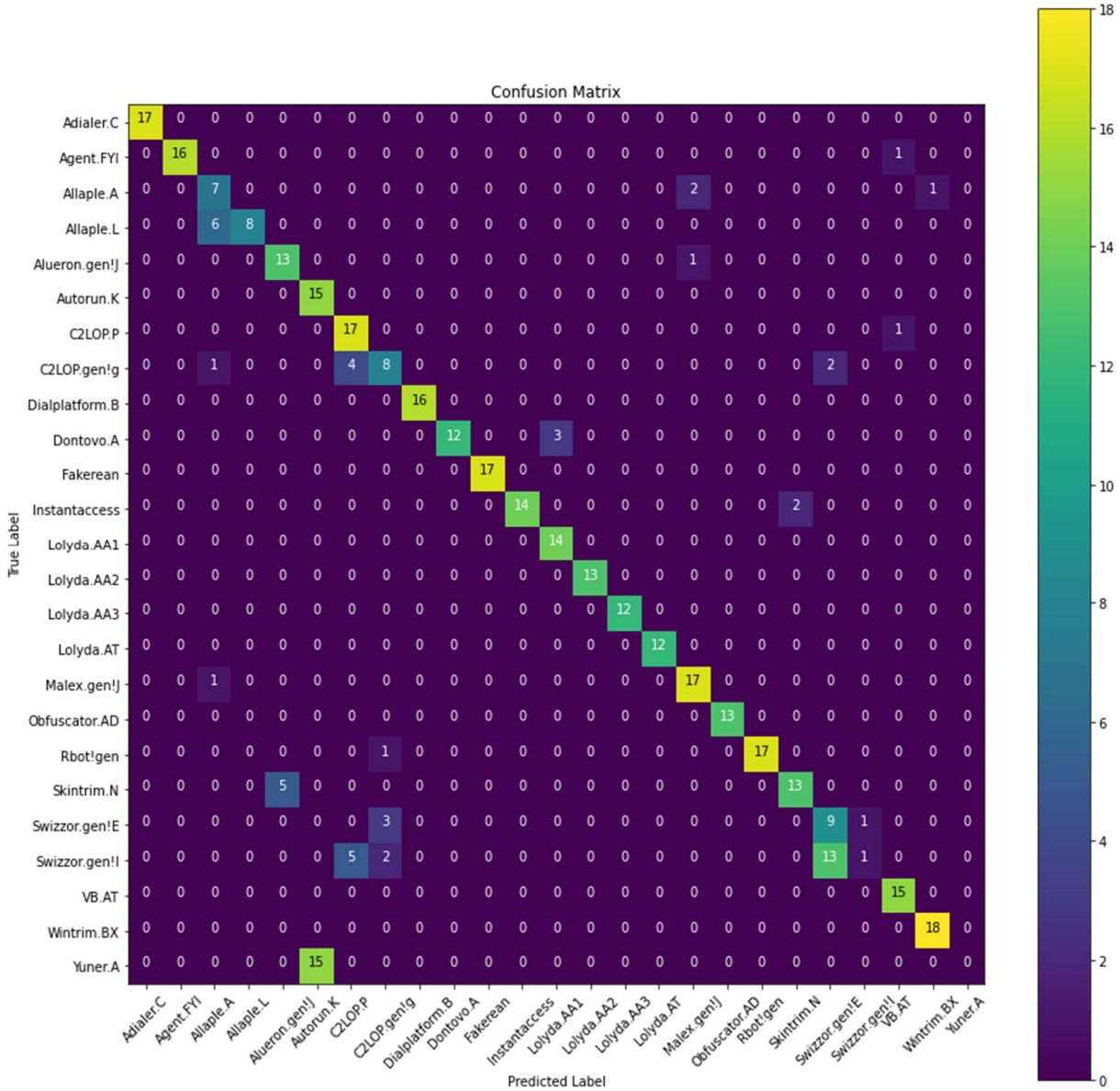

Fig. 23 Confusion Matrix of Inception V3 model for test set

Finally, we created a composite model by combining all trained models. The findings were produced using all of the model's predictions, and the final decision was decided using the majority rule. The accuracy of the composite model was 92 %. The confusion matrix of the test for the composite model is presented in figure 19.

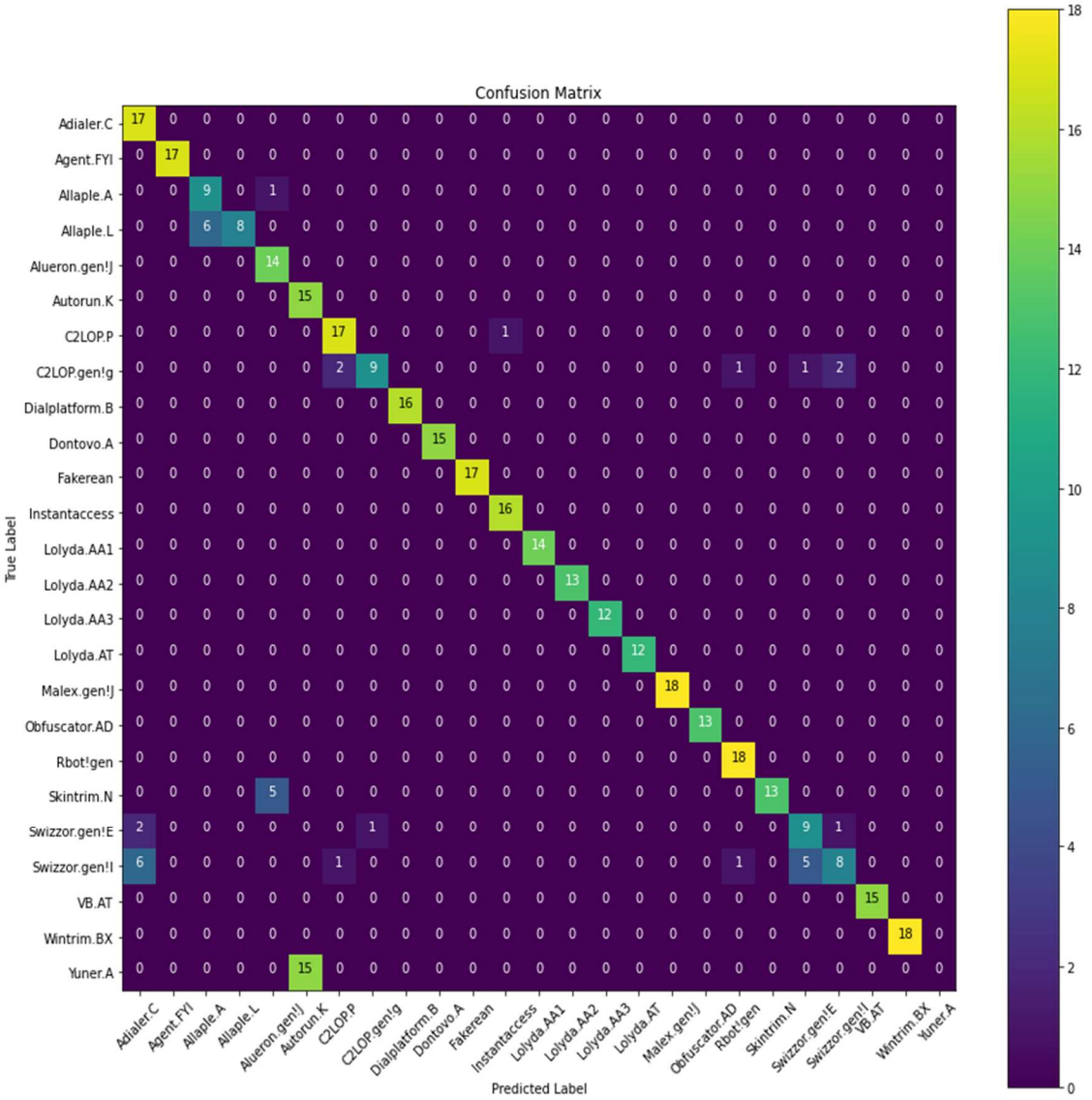

Fig. 24 Confusion Matrix of Composite Model for test set

After all of the models had been trained, the assessment measure was used to compare the outcomes. The testing set featured an almost equal number of malware images for each class, despite the fact that the original dataset was unbalanced. As a result, all of the models were compared to the accuracy score in order to choose the best Malware classification, model. CNN displayed a malware classification accuracy score of 90%. The confusion matrix revealed that the vast majority of classes are correctly labeled 100 percent of the time. However, there are eight classifications in the confusion matrix that have an accuracy score of 90 to 95 percent. Yuner A Malware is a well-known sort of malware that reduces the accuracy of a custom CNN model. All Yuner A malware samples were incorrectly classed as Autorn K malware samples, according to the confusion matrix. As a result, the Yuner A malware type's accuracy was 0%. This Yuner A class behavior presents several probabilistic difficulties and confirms that custom CNN was unable to train on Yuner A class data. The Yuner A class pattern is probably too complicated for model training, or there is no pattern for learning at all. On

the other hand, the model produced a substantial result for the classification of malware types other than Yuner A.

Next, the trained weights were then utilized for training the VGG16 model, which exhibited an accuracy of 80% for the test set. The VGG 16 model accurately learned certain malware classes, and it showed 100 percent accuracy for these classes. There were seven more classes that scored in the 80 to 90 percent range. For the VGG16 model, the confusion matrix demonstrated that Autorn K and Skintrim N are the most misclassified malware type. All of the Autorun K malware samples in the test set were incorrectly identified as Yuner A malware. The five Skintrim N samples are classed as Alueron, while the 10 samples are labeled as instant access. Out of 18 samples, 15 samples of Autorun K have been misclassified. The remaining malware is classified with an average accuracy of 90%. The Autorun K and Skintrim class VGG16 models, on the other hand, showed an accuracy of 80%.

Similarly, the Malimg dataset was also used to train the ResNet 50 model for malware image classification. The ResNet 50 model was trained using ImageNet weights that had been pre-trained. For the test set, the model exhibited an accuracy of 81%. The ResNet 50 confusion matrix revealed that most malware types are correctly learned and classified with a 100 percent accuracy score. Only 5 samples are misclassified in ResNet 50 confusion Matrix, and they belong to three separate classes. Few classes are completely misclassified, indicating that the model could not learn the pattern of these images for classification. Yuner A and Autorun K have been mislabeled as Fakerean, the third malware variant. The malware type Obfuscator was completely misclassified as Instant access malware. Except for these three classes, the remaining classes are classified with an accuracy rate of around 99%. Inception V3 is also trained to classify the different types of malware images. The result of Inception V3 was similar to that of custom CNN. It also had 90 to 95 % accuracy for the bulk of classes, and the Yuner A class was fully misclassified as Autorn K malware type, which was comparable to custom CNN. However, the Inception V3 total accuracy score was lower than the custom CNN accuracy.

Finally, the malware types were classified using a composite model. The composite model combines all of the previously described models and makes decisions based on a majority vote. It does not predict itself and will not decide based on a list of predictions. The composite model returned the class with the most occurrences in the given list. For testing data, the composite model exhibited a 92% accuracy. The confusion matrix of the composite model revealed that the majority of malware variants were classified 100 percent of the time. However, there existed another Yuner A class that had been misclassified. All 15 Yuner A malware samples were re-classified as Autorun K class using a composite model. However, 9 classes have an accuracy score of greater than 95%, and the composite models' average accuracy score for all malware types except Yuner A was about 98%. As a result of the outcomes of all models, some malware types were too difficult to learn for all of them.

The Yuner A and Autoren K malware types appear to be comparable malware types; as different models misclassify these classes interchangeably. All Yuner A samples were classified as Autorun K malware by Custom CNN and Inception V3; however, the VGG16 fully misclassified all Autorun K malware samples as Yuner A malware samples. ResNet 50, on the other hand, incorrectly classified all samples from both classes as the third malware type (Instant access). The three classes (Autorun K, Instant access, and Yuner A) collectively downscaled the performance of trained models. By accurately classifying the Autorun K and Instant access classes. The composite model increases performance. The Yuner A, on the other hand, remains a challenging class for models as the composite model misclassified it as well. Because the composite model makes decisions based on the majority of cases, the Yuner A class is too tough for the model to grasp. The details of results are mentioned in given table 3.

Table 3. Details of Results

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Custom CNN	90.07	90.01	89.95	90.00
Caps-Net	90%	83.93	84.87	81.87
VGG-16	80.16	80.00	79.90	80.10
ResNet-50	81.20	81.01	80.90	80.16
Inception-V3	87.10	86.90	87.04	87.09
Ensemble Model	92.30	92.05	92.15	92.25

6. CONCLUSION

In this study, we used the Malimg dataset to classify malwares. In addition, multiple built-in deep learning models were trained for a fair comparison of the models, also, first time the performance of Capsule Neural Network (Caps-Net) is explored for

malware image classification. Finally, the trained models were combined into a composite model. A few related classes such as Yuner A, Instant access, and Autorun K degrades the model's performance. Because all built-in models consistently misclassify samples belonging to these three classes, it is considered that these classes are the most similar and that models cannot differentiate them. However, while the composite model corrected the predictions for two classes (Instant access and Autorun K), the composite model misclassified the Yuner A class. The accurate prediction of the Yuner A class of malwares may necessitate a complicated deep learning model architecture or a few Image preprocessing procedures.

Furthermore, the proposed composite model correctly diagnoses malware 92 % of the time. The significant accuracy score indicated that the model is strong enough to classify malwares using image-based techniques. However, future studies will need to adjust the training scheme or model architecture to accurately classify complex malware types.

REFERENCES:

[1] Fossi, M., Egan, G., Haley, K., Johnson, E., Mack, T., Adams, T., ... & Wood, P., "Symantec internet security threat report trends for 2010," Volume XVI, 2011.

[2] Su, Jiawei, Danilo Vargas Vasconcellos, Sanjiva Prasad, Daniele Sgandurra, Yaokai Feng, and Kouichi Sakurai. "Lightweight classification of IoT malware based on image recognition." *In 2018 IEEE 42Nd annual computer software and applications conference (COMPSAC),* vol. 2, pp. 664-669. IEEE, 2018.

[3] Shabtai, A., Moskovitch, R., Elovici, Y., & Glezer, C., "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *information security technical report,* vol. 14, no. 1, pp. 16-29, 2009.

[4] S. Morgan, "2019 cybersecurity almanac: 100 facts, figures, predictions and statistics," *Cisco and Cybersecurity Ventures.* Accessed: Nov. 10, 2019. [Online]. Available: https://cybersecurityventures.com/ cybersecurity-almanac-2019

[5] Abdullah, T. A., Ali, W., & Abdulghafor, R., "Empirical study on intelligent Android malware detection based on supervised machine learning," *Int. J. Adv. Comput. Sci. Appl. (IJACSA),* vol. 11, no. 4, 2020.

[6] H. S. Anderson, A. Kharkar, B. Filar, and P. Roth, "Evading Machine Learning Malware Detection," *New York, NY, USA: Black Hat,* 2017

[7] R. Verma, ''Security analytics: Adapting data science for security challenges,'' *in Proc. 4th ACM Int. Workshop Secur. Privacy Anal. New York, NY, USA: ACM*, pp. 40–41, 2018.

[8] Y. LeCun, Y. Bengio, and G. Hinton, ''Deep learning,'' *Nature,* vol. 521, no. 7553, pp. 436–444, 2015.

[9] Davuluru, V. S. P., Narayanan, B. N., & Balster, E. J., "Convolutional neural networks as classification tools and feature extractors for distinguishing malware programs," *In 2019 IEEE National Aerospace and Electronics Conference (NAECON),* pp. 273-278, IEEE, 2019.

[10] Narayanan, B. N., & Davuluru, V. S. P., "Ensemble malware classification system using deep neural networks," *Electronics,* vol. 9, no. 5, 721, 2020.

[11] Davuluru, V. S. P., Narayanan, B. N., & Balster, E. J., "Convolutional neural networks as classification tools and feature extractors for distinguishing malware programs," *In 2019 IEEE National Aerospace and Electronics Conference (NAECON),* pp. 273-278, IEEE, 2019

[12] Narayanan, B. N., & Davuluru, V. S. P., "Ensemble malware classification system using deep neural networks," *Electronics,* Vol. 9, no. 5, 721, 2020.

[13] Vasan, D., Alazab, M., Wassan, S., Naeem, H., Safaei, B., & Zheng, Q., "IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture," *Computer Networks,* 171, 107138, 2020.

[14] H. S. Anderson and P. Roth, ''EMBER: An open dataset for training static PE malware machine learning models,'' 2018. [Online] https://arxiv.org/ abs/1804.04637

[15] M. Krcál, O. Švec, M. Bálek, and O. Jašek, "Deep Convolutional Malware Classifiers Can Learn from Raw Executables and Labels Only," 2018. [Online]. Available: https://openreview.net/forum?id=HkHrmM1PM

[16] S. Tobiyama, Y. Yamaguchi, H. Shimada, T. Ikuse, and T. Yagi, ''Malware detection with deep neural network using process behavior,'' *in Proc. IEEE 40th Annu. Comput. Softw. Appl. Conf. (COMPSAC),* vol. 2, pp. 577–582, 2016

[17] W. Huang, J. W. Stokes, ''Mtnet: A multi-task neural network for dynamic malware classification,'' *in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment, Cham, Switzerland: Springer,* pp. 399–418, 2016.

[18] R. Pascanu, J. W. Stokes, H. Sanossian, M. Marinescu, and A. Thomas, ''Malware classification with recurrent networks,'' *in Proc. IEEE Int. Conf. Acoust., Speech Signal Process (ICASSP),* pp. 1916–1920, 2015.

[19] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba, and T. Yada, ''Efficient dynamic malware analysis based on network behavior using deep learning,'' *in Proc. IEEE Global Commun. Conf. (GLOBECOM),* pp. 1–7, 2016

[20] B. Kolosnjaji, A. Zarras, G. Webster, and C. Eckert, ''Deep learning for classification of malware system call sequences,'' *in Proc. Australas. Joint Conf. Artif. Intell. Cham, Switzerland: Springer,* pp. 137–149, 2016.

[21] A. Damodaran, F. Di Troia, C. A. Visaggio, T. H. Austin, and M. Stamp, ''A comparison of static, dynamic, and hybrid analysis for malware detection,'' *J. Comput. Virology Hacking Techn.,* vol. 13, no. 1, pp. 1–12, 2017.

[22] Nataraj, L., Karthikeyan, S., Jacob, G., & Manjunath, B. S., "Malware images: visualization and automatic classification," *In Proceedings of the 8th international symposium on visualization for cyber security,* pp. 1-7, 2011.

[23] Vasan, D., Alazab, M., Wassan, S., Safaei, B., & Zheng, Q., "Image-Based malware classification using ensemble of CNN architectures (IMCEC)," *Computers & Security,* 92, 101748, 2020.

[24] Su, J., Vasconcellos, D. V., Prasad, S., Sgandurra, D., Feng, Y., & Sakurai, K., "Lightweight classification of IoT malware based on image recognition," *In 2018 IEEE 42Nd annual computer software and applications conference (COMPSAC),* 2, pp. 664-669, IEEE, 2018.

[25] Ni, S., Qian, Q., & Zhang, R., "Malware identification using visualization images and deep learning," *Computers & Security,* 77, pp. 871-885, 2018.

[26] L. Nataraj, "A Signal Processing Approach To Malware Analysis," *Santa Barbara, CA, USA: Univ.* California, 2015.

[27] L. Nataraj, V. Yegneswaran, P. Porras, and J. Zhang, ''A comparative assessment of malware classification using binary texture analysis and dynamic analysis,'' *in Proc. 4th ACM Workshop Secur. Artif. Intell. New York, NY, USA: ACM,* pp. 21–30

[28] M. Farrokhmanesh and A. Hamzeh, ''A novel method for malware detection using audio signal processing techniques,'' *in Proc. Artif. Intell. Robot. (IRANOPEN),* pp. 85–91, Apr. 2016.

[29] A. F. Agarap and F. J. H. Pepito. (2017). ''Towards building an intelligent anti-malware system: A deep learning approach using support vector machine (SVM) for malware classification.'' [Online]. Available: https://arxiv.org/abs/1801.00318

[30] E. Rezende, G. Ruppert, T. Carvalho, A. Theophilo, F. Ramos, and P. de Geus, ''Malicious software classification using VGG16 deep neural network's bottleneck features,'' *in Information Technology-New Generations. Cham, Switzerland: Springer,* 2018, pp. 51–59

[31] Yu, H., Wang, J., Bai, Y., Yang, W., & Xia, G. S., "Analysis of large-scale UAV images using a multi-scale hierarchical representation," *Geo-spatial information science,* 21(1), 33-44, 2018.

[32] Ni, S., Qian, Q., & Zhang, R., "Malware identification using visualization images and deep learning," *Computers & Security,* 77, pp. 871-885, 2018.

[33] Le, Q., Boydell, O., Mac Namee, B., & Scanlon, M., "Deep learning at the shallow end: Malware classification for non-domain experts," *Digital Investigation,* 26, S118-S126, 2018.

[34] Yan, J., Qi, Y., & Rao, Q., "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks,* 2018.

[35] Masum, M., Shahriar, H., Haddad, H., Faruk, M. J. H., Valero, M., Khan, M. A., ... & Wu, F., "Bayesian Hyperparameter Optimization for Deep Neural Network-Based Network Intrusion Detection," *In 2021 IEEE International Conference on Big Data (Big Data),* pp. 5413-5419, IEEE, 2021

[36] Kumar, R., Xiaosong, Z., Khan, R. U., Ahad, I., & Kumar, J., "Malicious code detection based on image processing using deep learning," *In Proceedings of the 2018 International Conference on Computing and Artificial Intelligence,* pp. 81-85, 2018.

[37] Namanya, A. P., Awan, I. U., Disso, J. P., & Younas, M., "Similarity hash based scoring of portable executable files for efficient malware

detection in IoT," *Future Generation Computer Systems,* 110, pp. 824-832, 2020

[38] Venkatraman, S., Alazab, M., & Vinayakumar, R., "A hybrid deep learning image-based analysis for effective malware detection," *Journal of Information Security and Applications,* 47, pp. 377-389, 2019.

[39] Yue, S., "Imbalanced malware images classification: a CNN based approach," arXiv preprint arXiv:1708.08042, 2017.

[40] Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G. G., & Chen, J., "Detection of malicious code variants based on deep learning," *IEEE Transactions on Industrial Informatics,* 14(7), 3187-3196, 2018.

[41] Mandal, B., Okeukwu, A., & Theis, Y., "Masked face recognition using resnet-50," arXiv preprint arXiv:2104.08997, 2021

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., "Imagenet large scale visual recognition challenge," 2014

[43] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* pp. 1–9, 2015.

[45] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z., "Rethinking the inception architecture for computer vision," *In Proceedings of the IEEE conference on computer vision and pattern recognition,* pp. 2818-2826, 2016.

[46] Yann LeCun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel, "Handwritten digit recognition with a backpropagation network," *In Proc. of Advances in neural information processing systems,* pp. 396–404, 1990

[47] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision,* 115(3):211–252, 2015.

[48] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *In Proc. of Advances in neural information processing systems,* pages 1097–1105, 2012

[49] Vijayakumar, T., "Comparative study of capsule neural network in various applications," *Journal of Artificial Intelligence,* 1(01), 19-27, 2019.