Malware Classification Using Customized Convolutional Neural Networks by Leaky RELU Activated Function

Rupali Komatwar, Manesh Kokare

Abstract: **In most recent years, the quantity of malware has increased enormously, raising a significant security risk to financial organizations, businesses and individuals. In order to analyze and identify such malware effectively, we need to be able to generalize them into groups and define their corresponding families based on their behavior. In this paper, a novel Customized Convolutional Neural Network (CCNN)is efficiently proposed. This modifies the elemental form of structurally based design of the convolutional network in such a manner to recognize and classifies the malware images into different family. A Leaky Rectified Linear Unit (Leaky RELU)is incorporated in our architecture, which has the aspect of no zero slopes in negative direction. Therefore, it develops the pictures of productivity of 64x64x50. Along with this Kernel Percolate is introduced, it detect an image pixels with a value of 1 along the diagonals of the image. The experiment is conducted over 9339 samples of the 25 families to classify malware pictures. The outcome of the experiment shows that, contrary to other CNN malware classification models, CCNN has been more than 98.81% accurate. Therefore, the suggested CCNN model reflects in conceptual designs the most comprehensive of malware classification systems.**

Keywords: **Malware classification, CNN, CCNN, Malware visualization, Deep Learning.**

I. INTRODUCTION

The detection and analysis of malicious software (malware) is becoming increasingly complicated by a growing amount of malware samples freshly identified. There has therefore been an exponential increase of new malware versions recognized by safety firms over the years. This is an overwhelming amount of malware analyst data because they must extract relevant data from these very large data sets. The statistics page of Virus Total demonstrates that it is possible to achieve or even exceed one million files per day by freshly presented samples for analyses [1]. This volume of samples is a difficult job for cryptography. While attempts are being made to automate reverse engineering and malware analysis, detection and assessment processes for manual or heuristic purposes remain extremely prominent. The results not only improve in pure numbers, but also in diverse samples, generating distinct versions of a common software malware that uses polymorphic and metamorphic algorithms. This includes signature schemes for the correct detection, classification and analysis of malware. In addition, cryptography procedures are encumbered to measure up to millions of samples.

Many machine learning systems were designed to solve issues of handling a big amount of malware samples. Previous works have actually tackled the issue of malware modeling [2].In addition to behavioral information, static code characteristics for statistical analysis were also used as information sources [3]. In addition, work is being done to combine static and dynamic methods [4].Nevertheless these methods are not widely used in the analytical malware, despite the growing popularity of neural network in machine learning apps that have led to performance improvements in many fields. However, attempts are being made in the region of implementation to implement new neural networks. Feed forward networks were

recently used to analyze malware code, for example [5].

Observations of different malware families conclude that malware belongs to one family have the same visual features and malware belongs to two different families have different visual features. Motivated by this observation we are introducing a novel approach for analyzing and classifying malware using image processing. In a broader perspective, a malware enact able can be characterized in the form of a binary string of zeros and ones. Image stretch provides more information about the assembly of malware. Furthermore, the various sections of the binary can be easily perceived when malware is viewed as an image where various subsections have a different texture.

In our proposed methodology, there is no need for disassembly or execution since the method entirely works on raw bytes, thereby making it faster than both static analysis and dynamic analysis. And the structure of packed malware variants do not change after packing since the proposed method is able to find similarity among packed malware variants where static analysis approaches like control flow graph analysis fail. The Malware authors use similar obfuscation techniques when creating malware variants of non-Windows based Operating Systems Linux, Android and OS X. Therefore, the proposed method need not be re-developed for a particular Operating System while traditional static and dynamic analysis based techniques need to be re-developed.

In CCNN is asserted for the automatic extension of the visualizing features from the executing files and thereby accomplishes in the malware classification. Although the number of CNN methods is accomplished using the appropriate form of intended balanced form of data. While, the dataset of the malware image [18] is highly imbalanced in nature. While some malware families have variants, however others only have less number of variants. Consequently, CNN models previously well-known and trained might poorly accomplish in this existing scenario. Thereby, the current paper inspired by the above challenges, uses CCNN for unbalancing malware families. By using the standard evaluation metrics, extensive evaluation is rendered and accomplished on the dataset available publically, the Ma1IMG [18] comprising of 9339 samples of the 25 families. For Malware classification comparative analysis based on the image based machine learning technique. From the experimental results, it was demonstrated that the proposed CCNN is highest at 98.81% thereby highlighting its feasibility as an accurate form of classified system.

The rest of the paper is organized as follows: the relevant research is presented in Section II. Additionally, the proposed CCNN model is conferred in Section III. While, learning algorithm of CCNN is explored and highlighted in Section IV. Experimental results and comparative assessment is given in Section V. Section VI concludes with future work.

2. Literature Survey:

Hu et al [6] Design, implementation, and evaluation of a malware database management system known as SMIT, which is capable of effectively assessing malware based on function call diagrams, which is known to be less sensitive to the instructional obscurations frequently used to avoid detection of AV software by malware authors. Since every malware program is displayed as a graph, an issue in a graphic database is caused by the search for the comparable Malware in a specified Malware Sample.

Jang et al [7] presented Current Bit Shred, a system to analyze and cluster large-scale malware resemblance and to automatically detect semantinal and intrafamily interactions within clusters. The main concept behind Bit Shred is to use feature hacking to decrease the large function spaces prevalent in malware analytics dramatically.

Kolbitsch et al [8] Offer a new, effective and efficient malware detection strategy, and can therefore be used to substitute or supplement

traditional anti-virus host software. In a controlled setting our strategy first analyzes the malware program in order to create a behavioral model. These models describe the data flows between calls that are crucial to the malware's work and therefore simple obscure or polymorphic techniques cannot be evaded readily.

Bayer et al [9] propose a scalable clustering strategy for identifying and grouping samples of malware with comparable behaviors. In order to get traces of malware programs, we first conduct dynamic analytical work. The tracks are then generalized into conducting profiles that more abstractly characterize the activity of a program.

Santos et al [10] Propose to be used as a file signature for the detection of unknown Malware with a fake favorable proportion (ngrams, each substring with a bigger strings with set length n). We demonstrate that n-grams can detect unknown malware effectively.

Roberto et al [11] suggest a quick statistical malware detection tool to enhance the scalability of current malware collection and analysis methods. Given a vast set of binaries that may include both previously unrecognized malware and benign executables, McBoost decreases the time of evaluation by classifying, filtering and transferring only suspect binaries to a comprehensive binary analysis procedure for extracting signatures.

Espoir et al [12] suggest a malware picture classification Convolutional Neural Network model that achieves 98% precision. The results of Microsoft's 2015 malwaredness classification competition winner who also used a convolutionary neural network strategy to obtain 99 percent precision by using three types of characteristics obtained from nearly half of a terabyte of malware samples did not, however, overcome using picture characteristics alone.

Jiawei et al [13] Offers a new lightweight strategy in IoT environment for detecting malware DDos. First, we remove gray pictures converted from binaries on one channel and then use a lightweight convolutionary nervous network to classify families of IoT malware.

Daniel et al [14] suggested is to obtain from disassembled binaries a single channel gray-scale picture sequence transformed from the malware. Then a two-bit configuration network (TBN) is used to detect IoT families of malware, which can encode weights of the two-bit network top.

In [6], [7] described common static code based analysis method is a control flow graph (CFG) analysis. It does not work on packed/obfuscated malware because the control flow of packed malware discloses only the unpacking predictable and not the actual flow. And [8], [9] the malicious code conduct is highlighted by running the suspicious code which are executable of the malware in a virtual sandboxed environment for several proceedings. It is time-consuming since the malware has to be observed for several minutes. In [10], [11] presented Statistical and content analysis based techniques are based on a variety of techniques: n-grams, n-perms, hash-based techniques, and file structure based techniques. Though its valuation was accomplished on the clustering of an unpacked malware dataset, no malware samples were used for testing the accuracy of the system. For [12], [13], [14] described the utility of the techniques based on machine learning for addressing the issues pertaining to the detection of malware and augmentation of the classification. Hence these methods complex in nature for obfuscation malicious code that modifies the whole binary structure and the selection of scaling parameter is acute to the network training. So to overcome all above mentioned issues in existing system, there is a necessity to proposed new methodology to tackle those issues.

3. Customized Convolutional Neural Network (CCNN):

Convolutionary Neural Networks (CNN) in the present erais the most effectively based profound learning models for image classification. Biology science is the inspiration of the multi-story architectures of CNN. High accuracy and rapid growth in the research of CNN motivated us to use deep learning approach to classify the malware images into different families. Proposed Customized Convolutionary Neural Network (CCNN) modifies the basic form of structurally-based design of the convolutionary network in such a way that CCNN acknowledges and classifies malware pictures into distinct families and produces much improved results than prior methodologies for malware classification. CCNN Model optimizes the number of layers instead of placing any layer restrictions. In addition, a variety of filter sizes were used for intermediate convolutionary CCNN layers.

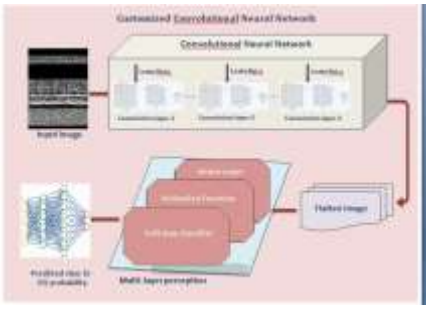

Figure 1: Schematic Block diagram of proposed Classification of Malware Image System

Figure 1. shows the schematic block diagram of the suggested scheme for malware classification using CCNN. An input malware picture is a size 64x64x1 gray scale image used for CCNN pre-processing. Here, the size of the convolution layer is further improved in addition to the activation functioning. In order to decrease the size of the output picture generated from the convolution layers, a stomp of two is made in max pooling. The picture will be flattened in the next step and transferred to the Multi-Layer Perceptron and a fully linked layer. The last stage is to use the soft max classifier to return the picture class probability for each class from 0 to 25.

Multi-layered perceptron stimulated CNNs. A subclass of neural systems is determined between certain layers as the convents with compulsory accessibility models. Neural networks do not very well scale for complete picture without the input of anyone else. Typical neural systems usually lead to the problem of over fitting. In contrast, customized convolutional neural network abuses the locally-based features of the given malware images, such as treating malware image pixels that are discerned nearby and inaccessible in an unexpected manner. The suggested technique is therefore used to classify the multiple malware pictures based on the characteristics of the picture. Figure 2, which achieved enhanced outcomes throughout the experimental assessment, shows the suggested CCNN architecture. Here, CCNN's layer perceptive perception is also displayed for malware image classification.

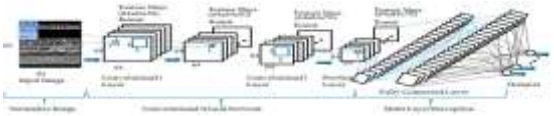

Figure 2: Architecture of CCNN

A) Convolution Layer

As shown in Figure 1, the input of the CCNN is a malware gray scale image $x^{w,h,d}$, wherein the width is determined by w, d is determined by d (d=1) and height with h of themalware image. Through this layer, invariant features of malware images are cultured hierarchically and automatically. Firstly convolution layer recognizes low-level features of an image to reserve spatial association between image pixels by exhausting minor patches of the image. It recognizes the local image characteristics from one layer and subsequently maps them to the featuring maps. Additionally, a 2 Dimensional

presentation of convolutionally layers is discerned in Figure 3, wherein the input malware image is a 64x64x1 matrix (width x height x depth) and filter of size 3x3x1 is mapped on it to convert given image to feature map.

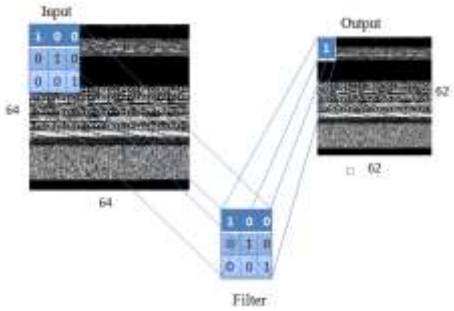

Figure 3: Feature map mirroring

In figure 3, the first layer of convolution, 50 filters of size 5x5x1 have been utilized. The CCNN model takes an input malware image of size 64×64×1 with a zero padding p=2. Filter solicitation is determined as the dot product calculation along with the input malware image. Here, the output volume is of size 64×64×50 where w= h=64= ((64-5) +2*2)/1+1 and depth d=50. Total numbers of neurons in this layer are 64×64×50=204800 neurons. Moreover, every 204800 neuron is associated to a locally based region of the kernel. On the next two Convolutional Layers, we applied 70 convolution filters of 3x3x1 and striding it amongst an input image with a strider 1. Initially, these filters are assigned with random values and then apply these filters on training data-set. The next step is activation function.

B) Leaky ReLU:

Traditional CNN architecture uses Rectified Leaner Unit (ReLU) activation function for bringing non-linearity into the weights present in intermediate layers of CNN. While the architecture proposed by us uses the concept of Leaky ReLU which solves the dying ReLU problem, which states that when there is negative gradient involved, the ReLU activation function completely reduces it to zero, while the leaky ReLU does not do it, because it does not have zero slopes present in negative direction, as shown by the following equation (2). This function is utilized to mark a distinct identification of probable features. Hence, Leaky ReLU yields 64x64x50 productivity images. This productivity images needs to be sampled to decompress into its original size. Thus, the image is given to the pooling layer for sampling.

C) Pooling Layer:

After the convolution layer the input malware image is converted as an image stack. Next step is the implementation of Pooling Layer, this is how we shrink the image stack and this is pretty straight forward. To reduce the input's spatial size, pooling is used. Pooling controls an over-fitting problem by reducing an image feature dimensions and calculations. Pooling builds the network invariant to trivial biases and alterations. Importantly, it assures to acquire a scale-invariant illustration of the malware image. The different types of pooling include the Max, Min and Average Pooling. In a proposed CCNN model, the max pooling is utilized. We start with the window size of 2x2 pixels. Pick a stride of 2 and walk your window in stride across a filtered image which is the output of earlier convolutional layer and for each window take maximum value. Thus, pooling layer generates an output image of size 32x32x50. A visualization of the pooling procedure is shown in Figure 4.

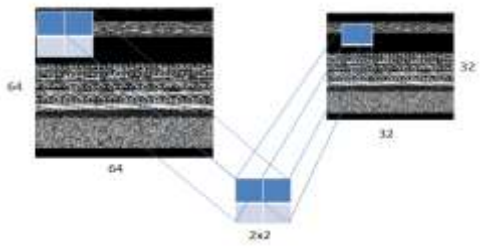

Figure 4: Visualization of Max Pooling

Output images of Pooling layer goes out in the form of input image towards second convolutionally based layer. Into the second layer, 70 kernels have been applied of size 3x3x1. All remaining parameters are the same as the first convolutional layer. The output of the second layer after applying the Leaky ReLU activation function is 64x64x70 and then max pooling for resampling the malware image. Repeat the same procedure for the third convolutional layer with 70 kernels of size 3x3x1 and generated output volume of size 64x64x70 images. Initially, all kernels are assigned a randomized value. In each epoch, kernel's value is changed for extracting the malware image features. After sampling the image, it is passed to fully connected layer for reducing the neuron overlapping and error minimization.

D) Fully Connected Layer:

In this layer, neurons are associated with all activation of an earlier layer. Here, along with a bias offset simple matrix multiplication is accomplished. Moreover, neurons are whispered when two or more related neurons discerned same malware features frequently for ensuring established codependency or co-adaption on all neurons that introduces the problem of over fitting. In our work, a dropout function is utilized in this kind of situation to eradicate such type of complexity by overlooking neurons random sets through the training phase. Drop Out function shuts down nodes randomly so that learning or pattern based updates can be prevented. Next 256 dense layers along with soft max classifiers are used in proposed CCNN for classifying the malware images.

E) Soft max classifiers:

Soft max classifiers offer you the likelihood of each class label while loss of the hinge provides you the margin. As humans, it is much easier for us to interpret probabilities rather than marginal scores (such as loss of hinge and loss of squared hinge).In addition, for datasets such as Image Net, we often look at Convolutional Neural Networks ' rank-5 precision (where we check to see if the ground-truth label for a specified input image is in the top-5 expected labels returned by a network).For the classification of the malware images, the soft max classifier is opted into its corresponding families.

Additionally, because of the primary randomness in weights, there must be change in the weights systematically and subsequently CCNN model is utilized. Remember that each k output neurons, which connects to all the neurons of the earlier layer indirectly relates to malware family. The number of output neurons must be noticed and must be equivalent to the number of malware families.

F) CCNN Learning Algorithm:

Step I: The MalIng dataset had malware images with distinct width and height. For proposed CCNN, we required a malware image of size 64x64x1. So, pre-process malware image to get the required size of an input image. An input of the model is a grayscale image that is an executable represented as $x^{w,h,d}$ wherein h is determined as the height,w is denoted as width and d as the depth(d=1). Image initializing is also carried out

before feeding the input to CCNN system for smoothing and noise removal of an input image.

Step II: The convolutional Layer

Provided the input image asI(w × h), a (f1 × f2) kernel (filter) F and constant biasb, the convolution layer output is as described below:

$$(I * F)_{ij} = \sum_{m=0}^{f_1-1} \sum_{n=0}^{f_2-1} F_{m,n}. I_{i+m,j+n} + b \quad (1)$$

Where $0 \le i \le w - f_1$ and $0 \le j \le h - f_2$

Step III: Leaky Rely activation function is given by equation (2) and used on the convolution maps to map particular output to a particular input.

$$f(x) = \{0.01x \text{ for } x < 0 \quad x \text{ for } x => 0 \quad (2)$$

x – The actual value of a pixel.

Step IV: Pooling Layer

Pick a stride usually 2 or 3 and walk your window in stride across the filtered image and for each window take maximum value, called as down sampling operation. Pool Layer produces a volume [w1×h1×d1] wherein the w1, h1, d1 are given by equation (3), (4), and (5):

$$w1 = \frac{w-f}{s} + 1 \quad (3)$$

$$h1 = \frac{h-f}{s} + 1 \quad (4)$$

$$d1 = d \quad (5)$$

Step V: Fully connected layer

All the features in the end are concatenated into a single vector that is extracted from previous layers.

Consider: The input to unit a_i^l (i^{th} unit in layerl) Q_i^l: The output of unit a_i^l.

f(x): Activation function

w_{ij}^l: weight from some unit

a_i^l's output to some other unit $a_j^{l=1}$

L: output layer, l: the hidden layer

E: the squared error function

1) Forward Propagation:

1. output of step I is used and thereby the inputs are calculated for the next layer:

$$P_i^l = \sum_j w_{ji}^{l-1} Q_j^{l-1} \quad (6)$$

2. Calculate activations for the layer which has known input:

$$Q_i^l = f(x) + Q_i^l \quad (7)$$

$where\ f(x)$is calculated by equation 2.

3. Repeat step 1 and 2 of forward propagations for the productivity layer.

2) Backward Propagation:

1. **Calculation of errors** at the output layer L:

$$\frac{\partial E}{\partial Q_i^L} = \frac{dE}{dQ_i^L} Q^L \quad (8)$$

2. Update bias by calculating the partial derivative of the error pertaining to image input of neuron at 1^{st} layer l for which errors are known

$$\frac{\partial E}{\partial P_j^l} = f(P_j^l)\left(\frac{dE}{dQ_j^l}\right) \quad (9)$$

$$b_j = b_j - b_{j-1}\frac{\partial E}{\partial P_j^l} \quad (10)$$

3. Errors computation at the earlier layer by employing a partial derivative of error intended in step I:

$$\frac{\partial E}{\partial Q_j^l} = \sum w_{ij}^l \left(\frac{dE}{dP_j^{l+1}}\right) \quad (11)$$

4. Steps 2 and 3 are repeated of back propagation till partial derivatives of errors are recognized at entire layers accepting an input layer.

5. Calculation of the gradient of the error:

$$\frac{\partial E}{\partial w_{ij}^l} = Q_i^l \left(\frac{\partial E}{\partial P_j^{l+1}}\right) \quad (12)$$

Step VI: Repeat steps II to V till error become less than or equal to minimum error (Chosen by the programmer).

Step VII: The trained model is ready after step VI. Use this model for the classification of the testing malware images.

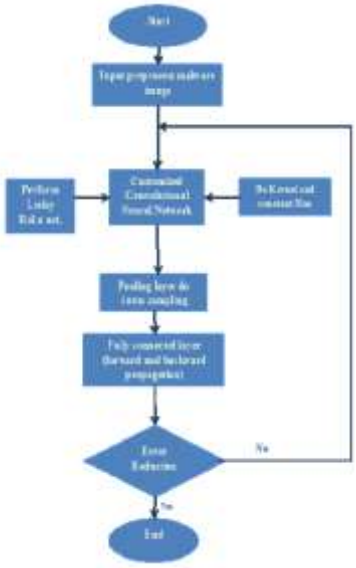

Figure 5: process flow of our proposed methodology

Once the kernel for the certain features is known, convolution operation extracts those certain features. In advance, neither the nature of an individual nor the total number of features is known to the CCNN. Henceforth, the convolution layer is not able to extract any specific feature. The convolution layer predicted the features in forward propagation and then back propagation attempts to precise those predictions sequentially. After completion of training, many filters are alleviated over error minimization and finally convolution extracts features which cannot prior determine. In the proposed CCNN model, anonymous features have been extracted from 9342 malware over 3 convolution layers, first layer with 50 filters and remaining two with 70 filters each. Furthermore, the precise nature of every feature can be discerned once the training ends. The following is presented in Figure 5.

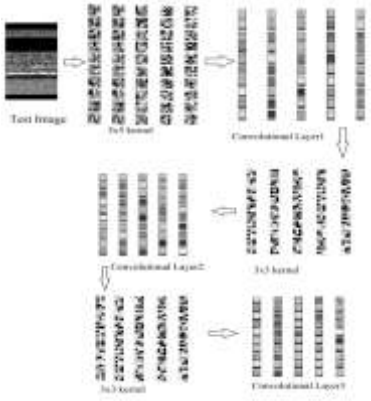

Figure 6: Visualization of Convolutional Layer during Testing

4. Result And Discussion:

A) Dataset

Natrajan[20] provided the MalIMg dataset[18], which contains 9339 grayscale images in total, divided into 25 different classes which are highly imbalanced. It contains some packed malware, packed with UPX packer such as VB. AT, Yuner. A, Malex.gen!J, Rbot!gen and Autorun. K. Among this dataset 80% data is used by CCNN for training, 10% for validating and remaining 10% for testing.

B) simulations Result

One of the primary things you want to prevent would be over fitting if you train a machine learning model. This is when your model fits well with the training data, but it can't generalize and make precise data predictions that it hasn't seen before. Data researchers use a method called cross-validation to find out if their model is over fitting, where they divide their information into two components-the training set and the validation set. To train the model, the training set is used, while the validation set is only used to assess the output of the model. To certify training properly, for each epoch, the validation accuracy is discerned.

The training accuracy is different from validation accuracy is shown in Figure 7 and how variation loss was optimally utilized is understood with the help of the graph as presented in Figure 8.

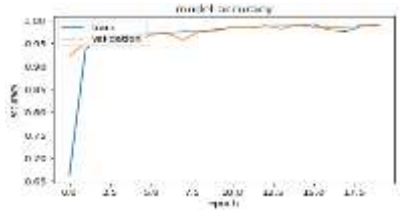

Figure 7: Training Accuracy vs. Validation Accuracy

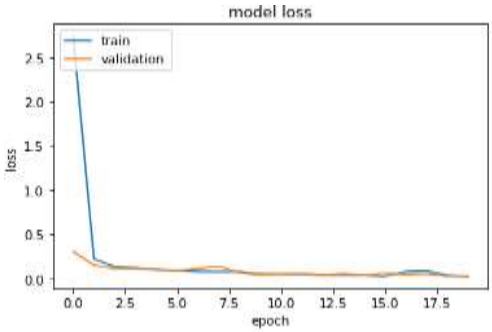

Figure 8: Training Loss vs. Validation Loss

C) Performance analysis:

Sometimes accuracy can be misleading measures, thus, to select the finest model, so excessive evolutionary metrics have been approached by us: specificity, sensitivity, negative predictive value, precision, false positive and negative rate, recall, false discovery rate, and F1 score.

Tab. 1: Result of Evaluation Metrics

Customized Convolutional Neural Networks	
Sensitivity	0.950
Specificity	0.998
Precision	0.947
Negative Predictive Value	0.99
F1 Score	0.941
Recall	0.951
False Positive	0.005

rate	
False Negative rate	0.049
False Discovery rate	0.052

The result of all these matrices on our testing dataset is shown in Table 1. High Recall asserts that the classes of testing malware data are recognized correctly. Moreover, precision indicates that high accuracy of CCNN model in detecting a class to have the corresponding malware images. Table 1 concludes that average of metrics sensitivity, specificity, precision, negative predictive values, recall and F1 score is greater than 0.9 thereby indicating that the proposed CCNN's results are more promising. On the other hand, the average of metrics false negative rate, false discovery rate, recall is less than 0.05, which shows the misclassification rate of CCNN model is very less. Figure 10 shows the performance analysis of our proposed methodology.

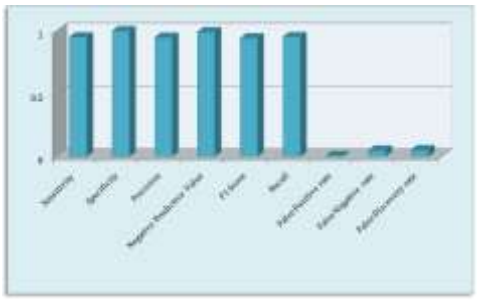

Figure 9: performance analysis of our proposed method

1. Sensitivity:

Sensitivity (S) is the number determining the true positive (TP) over the number of true positive plus the number of false negative (FN).

$$S = \frac{TP}{TP + FN} \tag{13}$$

2. Specificity:

Specificity (SF) is determined as the true negative (TN) over the number of true negative (TN) plus the number of false positive (FP).

$$\mathrm{SF} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} \tag{14}$$

3. Precision:

Precision (P) is determined as the number of true positive (TP) over the number of true positive (TP) plus the number of false positive (FP).

$$P = \frac{TP}{TP + FP} \tag{15}$$

4. Negative predicted value:

Negative predicted value (NP) is determined as the number of true negative (TN) over the number of true negative (TN) plus the number of false negative (FN).

$$NP = \frac{TN}{TN + FN} \tag{16}$$

5. False positive rate:

False positive rate (FPR) is d determined as the number of false positive (FP) over the number of true positive (TP) plus the number of false negative (FN).

$$FPR = \frac{FP}{TP + FN} \tag{17}$$

6. False discovery rate:

False discovery rate (FDR) is determined as the number of false positive (FP) over the number of true positive (TN) plus the number of false positive (FP).

$$FDR = \frac{FP}{TP + FP} \tag{18}$$

7. Recall:

Recall (R) is determined as the number of true positive (TP) over the number of true positive (TP) plus the number of false negative(FN).

$$R = \frac{TP}{TP + FN} \quad (19)$$

8. F1 Score:

F1 score (F1) is determined as the number of true negative (TN) over the number of true negative (TN) plus the number of false positive(FP).

$$SF = \frac{TN}{TN + FP} \quad (20)$$

The FI score is determined as the weighted average of precision, defined as the following:

$$F1 = 2\frac{P.R}{P + R} \quad (21)$$

At the same time, to illustrate the performance of malware classification we use confusion matrix.

D) Comparison of proposed system with existing techniques:

Experimental results are compared with CNN-FADL [14], GIST-knn[15], CNN-SVM [16], GRU-SVM[16], MLP-SVM[16], CNN-IDC [17] and CNNC[18] and wherein all these methods are used on the same dataset, MalIMG Dataset. The comparative results are shown in Table 3. From the table, it can be concluded that CCNN have the highest accuracy for malware classification. Figure 10 shows the accuracy comparison for proposed with existing methodology.

Tab. 2: Comparing experimental Result on MalIMG Dataset

GIST-knn	**CNN-SVM**	**GRU-SVM**	**MLP-SVM**	**CNN-FADL**	**CNN-IDC**	**CNNC**	**Proposed methodology CCNN**
98%	77.22%	84.92%	80.46%	98%	95.80%	96%	**98.81%**

Confusion matrix is a form of tabularized representation for pronouncing the performance of proposed classifiers, which is tested on the testing dataset for which the consistent true values are already determined.

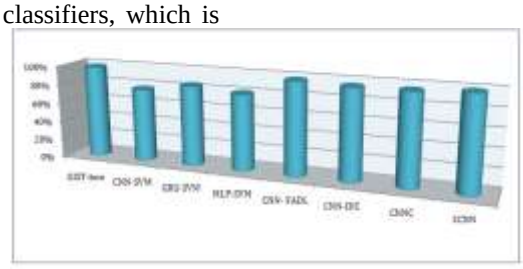

Fig. 10: Comparing accuracy on MalIMG Dataset

There are two main advantages of CCNN

methodology against the methodology of Natrajan [19]. Firstly, the proposed time for the classification is not disciplined depending on the training dataset size however it "memorizes" the training dataset. In significance, when a new malware is acknowledged CCNN goes through all training illustrations. Secondly, if a hacker knows that GIST mine features are grounded on the global structure of malware image then just with the help of rearranging various sections of the malware code, the detection method could be broken. In contrast, in our approach, by just reallocation technique for changing the malware code might not yield such undesired effects as customized convolutional networks are able to acquire features invariant to transformation. For the proposed model, the confusion matrix is shown in Figure 11 and Table 3. As observed in Table 3, only one major source of misclassification is Swizzor. gen!l is there, otherwise all malware are classified accurately.

Table 3: Confusion Matrix

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	**.08**	0	**.75**	.08	0	0	0	0	.09	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	.32	0	**.075**	0.12	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**0.94**	0	0	0	0	.06	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

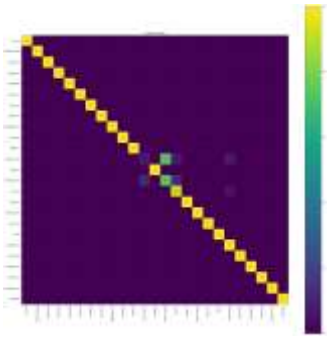

Figure 11: Confusion Matrix of CCNN malware classification model

5.Conclusion And Future Work:

This paper presents a novel Customized Convolutional Neural Networks for classification of malware based on its visualization and recognition of gray-scale image. As far as we realize, finding patterns from the pixel material of malware described as image is the first approach to applying customized neural networks. And also, the malware filter is used based on convolutionary neural networks as well as it turned more efficaciously by using different convolution layers method. An experimental outcome demonstrates that the precision is 98.81 percent based on our strategy. Moreover, CCCN is fully automatic, and researchers can use it directly to tackle their own issues of image classification, whether or not they have CNN knowledge.

Even the malware programs belonging to the same family is having identical patterns in visualization as an image, but for the encrypted or compressed image, the visualization may be completely different. In such case, only CCNN is not reporting the exact classification result. The combination of the CNN with knn or SVM, may result in more promising results.

REFERENCES

[1] Virus Total, (2015). File Statistics. *https://www.virustotal.com/en/statistics/, Nov.*

[2] Attaluri, S., McGhee, S and Stamp, M. (2009). Profile Hidden Markov Models and Metamorphic Virus Detection. *Journal in computer virology*. 5(2), 151–169.

[3] Schultz, M.G., Eskin, E., Zadok, E., and Stolfo, S.J. (2001) Data Mining Methods for Detection of New Malicious Executables. *in IEEE Symposium on Security and Privacy*.

[4] Kolosnjaji, B., Zarras, A., Lengyel, T., Webster, G., and Eckert, C. (2016). Adaptive semantics-aware malware classification in Detection of Intrusions and Malware, and Vulnerability Assessment. *Springer*. 419– 439.

[5] Saxe, Jand K. Berlin, (2015). Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. *arXiv preprint arXiv:1508.03096.*

[6] Hu, X., Chiueh, T and Shin, K.G. (2009). Large-scale malware indexing using function call graphs. *in Proceedings of the 16th ACM conference on Computer and communications security ACM*. 611–620.

[7] Jang, J., Brumley, D and Venkataraman, S. (2011). Bitshred: feature hashing malware for scalable triage and semantic analysis. *in Proceedings of the 18th ACM conference on Computer and communications security*. 309-320.

[8] Kolbitsch, C., Comparetti, P.M., Kruegel, C., Kirda, E., Zhou, X and Wang, X. (2009). Effective and efficient malware detection at the end host. *in Proceedings of Usenix Security symposium*. 351-398.

[9] Bayer, U., MilaniComparetti, P., Hlauschek, C., Kruegel, C and Kirda, E. (2009). Scalable, behavior-based malware clustering. *in Proceedings of NDSS'09.*

[10] Santos, Penya, Y., Devesa, J and Bringas, P. (2009). N-grams-based file signatures for malware detection. *In Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS*. 317-320.

[11] Roberto Perdisci and Andrea Lanzi. (2008). McBoost Boosting scalability in malware collection and analysis using statistical classification of executables. *Computer Security Applications*. 301-310.

[12] Espoir K. Kabanga & Chang Hoon Kim. (2018). Malware Images Classification Using Convolutional Neural Network. *in Journal of computer and communications.* 153-158.

[13] Jiawei Su, DaniloVasconcellos Vargas, Sanjiva Prasad, Daniele Sgandurra, YaokaiFeng, Kouichi Sakurai. (2018). Lightweight Classification of IoT Malware Basedon Image Recognition. *IEEE International Conference on Computer Software & Applications*. 664-669.

[14] Daniel Gibert, CarlesMateu, Jordi Planes, Ramon Vicens. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques, Springer*. 15-28.

[15] Nataraj, L., Karthikeyan, S., Jacob, G., and Manjunath, B.S. (2011). Malware images: visualization and automatic classification. *In Proc. of the 8th International Symposium on Visualization for Cyber Security, Viz Sec '1, USA, ACM.* 4(7), 1-4

[16] Abien Fred M. Agarap. (2019). Towards Building an Intelligent Anti-Malware System: A Deep Learning Approach using Support Vector Machine (SVM) for Malware Classification.

[17] Songqing Yue. (2017). Imbalanced Malware Images Classification: a CNN based Approach.

[18] http://old.vision.ece.ucsb.edu/spam/malimg.shtml

[19] Claudio Guarnieri. (2010) Cuckoo Sandbox..

[20] Retrieved fromhttp://www.cuckoosandbox.org/.