Malware Classification using Multilayer Perceptron Model

Pascal Onyeka, Umar Farouk, and Yadi Zhong
Auburn University
Auburn, Alabama, USA
{pco0004,ufm0001,yadi}@auburn.edu

ABSTRACT

As technology advances, attackers try to find new ways to create and spread malware. In parallel, machine learning has been studied for years to improve classification accuracy. Implementing machine learning to understand the various types of malware can help keep pace with malware evolution. The current solutions to understanding malware can only be done on a smaller scale. However, the use of machine learning can be used to understand malware at a much larger scale to protect both small and large systems. In this project, we propose a multi-layer perceptron model to classify different malware types more accurately using the Malimg dataset. We first created a subset of Malimg (consistent with the Malimg paper [10]) to test our proposed model Multi-layer Perceptron (MLP), and then we applied the same model to a more condensed version of the original Malimg by merging the variants of the same malware type into one. For Malimg subset classification accuracy, we obtained a training accuracy of 99.6%, validation accuracy of 99.8%, and testing accuracy of 98.7% with parameters of 300 epochs and a learning rate of 0.00001. For Malimg classification accuracy, we also obtained a training accuracy of 91.6%, validation accuracy of 90.14%, and testing accuracy of 92.3% with parameters of 300 epochs and a learning rate of 0.00001. In addition, we benchmark our proposed architecture with different baseline models with SVM, CNN, and DNN. We compare our accuracy with the state-of-the-art networks and present the future directions to accelerate the classification accuracy further.

CCS CONCEPTS

• **Security and privacy → Intrusion/anomaly detection and malware mitigation**; **Malware and its mitigation**; • **Computing methodologies → Machine learning**.

KEYWORDS

Malware detection and classification, multilayer perceptron, machine learning, deep learning, transfer learning, Adam optimization.

ACM Reference Format:
Pascal Onyeka, Umar Farouk, and Yadi Zhong. 2022. Malware Classification using Multilayer Perceptron Model. In *Proceedings of Machine Learning Final Project (COMP 6630).* Auburn University, Auburn, AL, USA, 11 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As technology evolves, new malwares are being made by attackers to try and break computer systems. The continued increase of malware attacks on computer systems, mobile devices, and the Internet of Things (IoTs) emphasizes the importance and the major motivation for designing robust malware detection and classification algorithms [11]. As the threat levels of different malware vary based on their malicious functions and intents, accurate detection and classification of malware are crucial to the normal operations of devices and systems. Furthermore, resilient malware detection

COMP 6630, December 2022, Auburn, AL, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and classification helps computing systems, mobile devices, and edge devices secure from malware infections by performing both dynamic and static analysis. With machine learning, antivirus software can detect new threats without the need to rely on the conventional hash or signatures solely. Machine learning-based malware classification has been widely adopted in different antivirus software to identify potential malicious programs and code quickly. For example, (*i*) Deep Instinct D-Client relies on deep learning to detect any file before it is accessed or executed on the system [5]. The D-Client uses static file analysis and a threat prediction model to eliminate malware and other system threats, by doing so autonomously. (*ii*) Avast's CyberCapture security suite also relies on machine-learning technologies with its three main components, Malware Similarity Search, Evo-Gen, MDE (in-memory database) [3].

Our project is centered around the image-based malware classification technique using the multi-layer perceptron (MLP). Binaries of malware are converted into images for better "visualization" purposes. Once it is converted to images, well-known image processing techniques such as edge detections, local binary pattern (LBP), and discrete wavelet transformation (DWT) are applicable to pre-process the malware in a different perspective than simply examining malware information in binaries. As state-of-the-art deep neural networks (DNNs) can classify images with high prediction accuracy, our goal is to build a multi-layer perceptron with similar computations in the DNN layers to achieve the desired malware classification accuracy.

The main contribution of the report is described as follows:

- We propose a multi-layer perceptron model with 1 input layer, 1 hidden layer, and 1 output layer to perform malware classification. We have demonstrated over 90% prediction accuracy on the Malimg dataset and its subset of 8 classes.
- We analyze the backpropagation algorithm used in updating weights for both hidden and output layers. We provide in-depth derivation on the gradient calculations.
- We benchmark our proposed MLP with other baseline models in [2, 6, 8, 10]. In addition, we have created our own baseline model using AlexNet and transfer learning. Based on the experimental results, we point out the possible future directions for further improvement of the MLP model.

The rest of the report is organized as follows. We briefly introduce the background for malware detection and classification in Section 2. Our proposed multilayer perceptron model is presented in Section 3. A theoretical perspective for the gradient update in backpropagation is analyzed in Section 4. We present the model benchmarking and evaluations in Section 5. We point out the possible future directions for further improvement of the proposed MLP model in Section 6. Finally, we conclude the paper in Section 7.

2 BACKGROUND

In this section, we briefly introduce malware, the multilayer perceptron model (MLP). The background in malware images and how they can be used in MLP, our dataset selection, different baseline models, and the potential challenges are also described.

2.1 Motivation - What is Malware

Malware is any software intentionally designed to operate to the hacker's advantage and negatively affect the victim. The damage caused by malware varies, which mostly depends on the intent and aim of the attacker. In addition, malware attacks usually have the ability to bypass certain security controls, wipe information, block access, and use the resources of the victim without them knowing. There are a variety of malware types, and each one of them has a special goal and target [9]. Some common targets are industrial networks, personal computers, cell phones, and Internet of Things (IoTs) devices. A few of the main types of malware are: (*i*) virus: a type of malware that replicates itself on a host to stay on a host; (*ii*) adware: interferes with standard ads to generate 3rd party revenue; (*iii*) Trojan Horse: which gains access to a device by hiding its true intent; (*iv*) wipers: wipes files to deprive victims of data and access; (*v*) spyware: provides remote access to local data; and (*vi*) ransomware: encrypts victim's files and tries to sell the decryption key, etc.

2.2 Multi-Layer Perceptron (MLP)

Multi-layer perceptron (MLP), a machine-learning technique that is widely used in the 1980s, consists of a fully connected class of feedforward artificial neural networks (ANN). It is very effective in its ability to solve problems stochastically, allowing approximate solutions for extremely complex problems. MLP algorithm

highly resembles the working principles of a human brain. The dendrites of a neuron receive electrical signals/information from other neurons and bring it to the cell body, and axons, and send information from the cell body to other neurons once the threshold potential is met. The basic building block of an MLP is the neuron, which can implement a non-linear function of its inputs, where these inputs can be considered as the dendrites accepting the signals sent from the prior neurons. An MLP is comprised of one input layer, multiple hidden layers, and one output layer, each with a different number of neurons. Generally, each neuron computes a weighted sum of its input and passes the result through a non-linear activation function.

2.3 Malware Classification with MLP

Although malware executables are binary in nature, it is possible to transform them into non-binary files. The visualization technique allows us to map malware binaries to grayscale or RGB images. One of the popular techniques is to group 8 bits at a time to convert the bitstream to byte arrays. Then, depending on the desirable output structure, it can be mapped to a $2D$ or $3D$ array of bytes. As one byte is the foundation for a pixel, the $2D$ or $3D$ byte array is simply a grayscale or an RGB image. Due to the similar characteristics in the malware binaries of the same class, the transformed images also share many similarities within one category. Thus, it enabled us to use digital image processing techniques to pre-process the images before passing them through MLP for classification. The common feature extraction techniques used in image processing are image filters,e.g. Gabor filter for texture extraction and Canny filter for edge detections, local binary pattern (LBP), and discrete wavelet transformation (DWT). These pre-processing algorithms could possibly speed up the training process for MLP and help the model to achieve high classification accuracy due to the common features of malware exhibited in the transformed images. This also removes the need to perform runtime execution or disassembly.

2.4 Dataset Usage

Malimg [10], a publicly available dataset consisting of 25 different classes of malware in grayscale images, is selected as the malware dataset. There are a total of 9339 grayscale images with varied sizes. For machine learning with constant input size, we need to resize the images into a fixed dimension. Nataraj et al. [10] performs malware classification on the subset of Malimg with 8 classes, the entire dataset (25 classes), and by combining the variants of the same malware classes. In this project, we select the Malimg subset (8 classes) and the complete dataset by combining the variants of the same malware type as a single class.

2.5 Baseline Models

Over the years, different machine learning algorithms have been proposed to target high accuracy for malware classification [1]. The initial research on malware images, proposed by Nataraj et al., pre-process these images with GIST feature computation and then apply K-Nearest Neighbor (KNN) to predict the corresponding malware class [10]. Other models like Support Vector Machine (SVM), convolutional neural network (CNN), and different Deep Neural Network (DNN) models are used to predict malware classes [2, 6, 8, 13].

2.6 Potential Challenges

Below are some of the issues that could potentially hinder the correct operation of the model.

- Data/Image Processing: The size of each image in the dataset being used for this project could be a potential challenge during the implementation of the model. The dataset for our model contains different images with unique feature space, where the dimension varies from image to image. Finding the best dimension for resizing the dataset and extracting important features to fit our design is crucial for our classification accuracy.
- Learning rate selection: Learning rate is an important hyper-parameter in machine learning which determines how well the parameters update rule can determine an optimal neural network model with better performance. There are different techniques that could be used to select a learning rate for this project. The main challenge is that it might be difficult to choose a method that gives an optimal learning rate for our model.
- Model overfitting or underfitting: Underfitting could happen if the testing and training error is too large. Overfitting occurs when the training error is small, and the testing error is too large. There could be challenges in finding the best set of parameters that fits

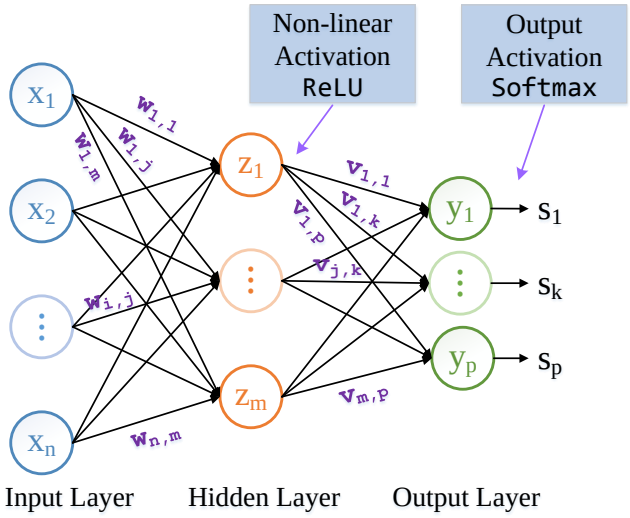

Figure 1: The proposed multi-layer perceptron model for malware classification.

the model perfectly because overfitting/underfitting could negatively impact the model.

- Diminishing gradient: The weights of the hidden layers could get minimal updates when there exist multiple hidden layers. This will have a negative effect on the performance of the model because marginal parameter update steps are being performed.

3 PROPOSED MULTI-LAYER PERCEPTRON MODEL

Figure 1 shows our proposed multi-layer perceptron (MLP) model. It consists of one input layer, one hidden layer, and one output layer, where n, m, and p denote the number of neurons for three layers, respectively. As the MLP is a fully connected architecture, one neuron is connected to all the neurons of both the previous layer and the subsequent layer. The input of MLP is denoted as x_i, and the output is s_p. The weight between the i^{th} input neuron and j^{th} hidden neuron are denoted as $w_{i,j}$. Similarly, the weight between the j^{th} neuron in the hidden layer and k^{th} neuron at the output are denoted as $v_{i,j}$. Each neuron in the hidden layer and the output layer also includes a constant bias, $w_{0,j}$ and $v_{0,k}$, respectively. We include constant 1 neurons, $x_0 = 1$ and $r_0 = 1$, for both input and hidden layers for a more compact representation of the MLP calculation with the bias $w_{0,j}$ and $v_{0,k}$, where $x_0 \times w_{0,j} = w_{0,j}$ and $r_0 \times v_{0,k} = v_{0,k}$.

To facilitate the gradient computation described in Section 4, we break down the computation of the proposed MLP into the following four steps:

- *Step-1:* Each neuron in the hidden layer computes the weighted sum of the input x_is, where the j^{th} neuron has

$$z_j = \sum_{i=1}^{n} x_i \times w_{i,j}. \tag{1}$$

  Note that this is a linear function.
- *Step-2:* The linear sum z_j passes through a non-linear function R. In this proposed MLP, we select `ReLU` as the activation function.

$$r_j = \texttt{ReLU}(z_j) = \begin{cases} z_j, & z_j \geq 0 \\ 0, & z_j < 0. \end{cases} \tag{2}$$

- *Step-3:* The neurons in the output layer perform weighted sum to all values r_js generated from the hidden layer. Specifically, the computation for the k^{th} neuron at the output layer is

$$y_k = \sum_{j=1}^{m} r_j \times v_{j,k}. \tag{3}$$

- *Step-4:* The output activation function is applied to each y_k, and the class with the highest score is recorded as the prediction label. We select the `softmax` function as it is a normalized exponential function, where the sum of all s_k is equal to 1. The final output s_k is computed as follows,

$$s_k = \texttt{softmax}(s_1, ..., s_p) = \frac{e^{y_k}}{\sum_{l=1}^{p} e^{y_l}} \tag{4}$$

4 THEORETICAL ANALYSIS ON BACKPROPAGATION

In order to learn the optimal set of weights for achieving high classification accuracy, backpropagation is used to propagate the total loss computed at the MLP output back to each weight during training. In this section, we describe how the loss gradient is computed with respect to each weight and how the weights are updated. Our weight update is based on each training epoch, where the error is accumulated from the predictions of all samples in the training dataset. We did not split the training dataset into small partitions nor use a stochastic gradient descent algorithm. We denote the learning rate for the weight update as γ, which is

a constant and is applied to all the weights in the proposed MLP. The general formulation for weight update is described as

$$w^{t+1} = w^t + (-\nabla w) \times \gamma, \tag{5}$$

where w^{t+1} represents the weight computed for the $t+1$ epoch, and the previous epoch (t^{th}) has the old weight of w^t. First, a loss function has to be computed for the misclassification between the ground truth and the prediction over all training data X_is. We select `cross-entropy` as the loss function, which is defined as

$$\begin{aligned} \mathsf{L} &= \texttt{cross-entropy} \\ &= -\frac{1}{N}\sum_{q=1}^{N}\sum_{r=1}^{p}\Big(T_{q,r}\cdot log(S_{q,r}) + \big(1-T_{q,r}\big)\cdot\big(1-log(S_{q,r})\big)\Big) \end{aligned} \tag{6}$$

where N is the number of samples in the training dataset, p is total number of prediction classes, $T_{q,r}$ is the ground-truth of that class, and $S_{q,r}$ is the MLP response for a given class. For the q^{th} training data, the MLP prediction for each label $\{s_1, s_2..., s_P\}$ are represented as $\{S_{q,1}, S_{q,2}, ..., S_{q,p}\}$. The gradient of `cross-entropy` with respect to a particular label S_k is computed as

$$\frac{\partial L}{\partial S_k} = -\frac{1}{N}\sum_{q=1}^{N}\Big(\frac{T_{q,k}}{S_{q,k}} - \frac{1-T_{q,k}}{1-S_{q,k}}\Big). \tag{7}$$

As the output probability of each class of the MLP is computed from output activation `softmax`, the gradient of `softmax` with respect to its input is

$$\begin{aligned} \frac{\partial S_k}{\partial y_k} &= \frac{\partial}{\partial y_k}\left(\frac{e^{y_k}}{\sum_{l=1}^{p} e^{y_l}}\right) = \frac{e^{y_k}}{\sum_{l=1}^{p} e^{y_l}} + \frac{-e^{y_k}\cdot e^{y_k}}{\big(\sum_{l=1}^{p} e^{y_l}\big)^2} \\ &= S_k - S_k^2 = S_k \times (1 - S_k). \end{aligned} \tag{8}$$

As the MLP prediction needs four steps to compute from the input xs, we detail the weight gradient calculation for both the output layer ($v_{j,k}$) and hidden layer ($w_{i,j}$) below.

- *Weight Gradient for Output Layer:* As y_k is a weighted sum of the input to the output layer, its derivative with respect to $v_{j,k}$ is just the corresponding input r_j itself.

$$\frac{\partial y_k}{\partial v_{j,k}} = \frac{\partial}{\partial v_{j,k}}\left(\sum_{l=1}^{m} r_l \times v_{l,k}\right) = r_j \tag{9}$$

  Thus, the negative loss gradient $-\nabla v_{j,k}$ with respect to each weight $v_{j,k}$ of the output layer is calculated using the chain rule in calculus:

$$\begin{aligned} -\nabla v_{j,k} &= \frac{\partial L}{\partial v_{j,k}} = \frac{\partial L}{\partial S_k}\cdot\frac{\partial S_k}{\partial y_k}\cdot\frac{\partial y_k}{\partial v_{j,k}} \\ &= -\frac{1}{N}\sum_{q=1}^{N}\left(\frac{T_{q,k}}{S_{q,k}} - \frac{1-T_{q,k}}{1-S_{q,k}}\right)\cdot S_k \times (1-S_k)\cdot r_j. \end{aligned} \tag{10}$$

- *Weight Gradient for Hidden Layer:* To further propagate the error loss back to the weights in the hidden layer, additional derivatives with respect to the parameters in the hidden layer need to be computed. As y_k is a function of all hidden neurons, derivative of y_k with respect to the j^{th} hidden neuron is

$$\frac{\partial y_k}{\partial r_j} = \frac{\partial}{\partial r_j}\Big(\sum_{l=1}^{m} r_l \times v_{l,k}\Big) = v_{j,k} \tag{11}$$

  Due to the non-linear property of the ReLU activation function, the derivative of its output r_j is

$$\frac{\partial r_j}{\partial z_j} = \frac{\partial}{\partial z_j}\mathsf{ReLU}(z_j) = \begin{cases} 1, & z_j \geq 0 \\ 0, & z_j < 0. \end{cases} \tag{12}$$

  The hidden layer's z_j is a weighted sum of the input neurons x_is, and its derivative with respect to each weight $w_{i,j}$ is just the corresponding primary input x_i itself.

$$\frac{\partial z_j}{\partial w_{i,j}} = \frac{\partial}{\partial w_{i,j}}\left(\sum_{l=1}^{n} x_l \times w_{l,j}\right) = x_i \tag{13}$$

  Note, there is a non-trivial difference between computing the weight gradient in the hidden layer and the output layer. Due to the output of each hidden neuron propagating to all the neurons in the output layer, the gradient calculation needs to accumulate the error loss from all neurons y_ks, not a single one only. Hence, the negative loss gradient $-\nabla w_{i,j}$ with respect to each weight $w_{i,j}$ of the hidden layer is computed as follows with the chain rule:

$$\begin{aligned} -\nabla w_{i,j} &= \frac{\partial L}{\partial w_{i,j}} = \sum_{k=1}^{p}\frac{\partial L}{\partial S_k}\cdot\frac{\partial S_k}{\partial y_k}\cdot\frac{\partial y_k}{\partial r_j}\cdot\frac{\partial r_j}{\partial z_j}\cdot\frac{\partial z_j}{\partial w_{i,j}} \\ &= \sum_{k=1}^{p}\Bigg\{-\frac{1}{N}\sum_{q=1}^{N}\left(\frac{T_{q,k}}{S_{q,k}} - \frac{1-T_{q,k}}{1-S_{q,k}}\right)\cdot S_k\cdot(1-S_k) \\ &\qquad \cdot v_{j,k}\cdot\{1/0\}\cdot x_i\Bigg\} \end{aligned} \tag{14}$$

Therefore, we have provided a comprehensive analysis of the backpropagation of `cross-entropy` error loss and gradient calculation for weight update for both output and hidden layers.

5 EXPERIMENTAL RESULT

In this section, we describe our hyperparameter selection for our proposed MLP model, the training process, and the following metric to evaluate the performance of the proposed MLP.

- The training, validation, and testing accuracy are used to determine how well the model is learning. It is a possible indicator of overfitting/underfitting issues.
- Confusion matrix is used to evaluate the performance of the model. This is done by calculating the true positive/negative (TP, TN) values and false positive/negative (FP, FN) values.
- Also, evaluation metrics such as F1 score, recall, and precision, in addition to accuracy, will be used to demonstrate the model performance further.

Note that the definition for accuracy, precision, recall, and F1 score are based on the ground truth and the prediction labels. In particular, for class i, the corresponding metric is based on the number of true positives (TP_i), true negatives (TN_i), false positives (FP_i), and false negative (FN_i), and total test samples ($Total_i = TP_i + TN_i + FP_i + FN_i$), where

$$accuracy_i = \frac{TP_i + TN_i}{Total_i} \tag{15}$$

$$precision_i = \frac{TP_i}{TP_i + FP_i} \tag{16}$$

$$recall_i = \frac{TP_i}{TP_i + FN_i} \tag{17}$$

$$F1 - score_i = \frac{2}{\frac{1}{recall_i} + \frac{1}{precision_i}}. \tag{18}$$

In addition, we compare our malware classification results with the existing machine learning and convolution neural networks-based techniques [2, 6, 8, 10]. Moreover, we use transfer learning on Alexnet to further benchmark our results with the deep neural network. The malware images are from Malming [10].

5.1 Hyperparameters Selection

The multi-layer perceptron model was designed using the Python programming language. some Python libraries were also used to process the dataset. The dataset contained images of large sizes, so to fit those images into our model, they had to be reshaped to a (64x64) image size. This was done by using the Keras and Numpy libraries in python. After that, with the use of the SciKit library, the data was split into training, testing, and validation. 80% of the data was used for training, 10% for validation, and 10% for testing. The multi-layer perceptron model consisted of an input layer of $n = 4096$ neurons, a hidden layer of $m = 256$ neurons, and an output layer of $p = 8$ neurons for our subset and $p = 19$ neurons for the full dataset. All the neurons in the hidden layer are then passed through the rectified linear unit activation function (ReLU). which is a non-linear function. It returns 0 if the input is negative, or returns the input for any positive value received. Finally, cross-entropy was used as the loss function, and to generate the output, the softmax function was used.

To achieve the best accuracy, different values for hyperparameters, such as learning rate and the number of epochs, were tried. using a learning rate of 0.0001 and the number of epochs of 100 produced a very poor accuracy for both the subset and the entire dataset. Tables 1 and 2 shows the accuracy the proposed MLP model obtained. The accuracy gotten showed that the model is not learning enough, therefore it is not classifying the data correctly. The learning rate and the number of epochs are changed to 0.00001 and 300, respectively. This produced a much more higher accuracy for both the subset and the dataset. Comparing the accuracy obtained from the subset to the entire dataset, the subset has a much higher accuracy because it contains about 4000 fewer images. To achieve a similar accuracy on the entire dataset, the number of epochs will have to be increased to about 1000. This will take a long time to process on the device currently used to execute the model. So, to save time, and not stress out the device, our optimal learning rate and the number of epochs came out to be 0.00001 and 300, respectively.

The complete source code of the python implementation of the proposed multi-layer perceptron model for malware classification can be found in GitHub [14].

5.2 Malware Classification Accuracy

Tables 1 and 2 below show the accuracy gotten for both the Malimg subset and the entire Malimg dataset. They include the accuracy obtained from trying different values for the learning rate and the number of epochs.

Table 1: Malware classification accuracy with Malimg subset.

Malimg Subset			
Parameters	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
number of epochs=100, learning rate=0.0001	56.21%	50.3%	57.7%
number of epochs=300, learning rate=0.00001	99.6%	99.8%	98.7%

Table 2: Malware classification accuracy with complete Malimg dataset.

Malimg (19 classes)			
Parameters	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
number of epochs=100, learning rate=0.0001	50.6%	49.5%	51.5%
number of epochs=300, learning rate=0.00001	91.6%	90.14%	92.3%

Based on the table, the accuracy achieved on both the subset and the entire dataset has been greatly improved when the number of epochs is increased, and the learning rate is decreased.

Figures 2 and 3 show the confusion matrix plot gotten from executing the model. Based on the plot, one can see how all the data of different classes are correctly predicted or misclassified. It helps in visualizing if a particular data was classified into the right class or not. Tables 3 and 4 shows the name of the classes, which are displayed as 0-7 on the confusion matrix.

Table 3: Class label for Malimg subset.

Label	Malware Class
0	Allaple.A
1	Fakerean
2	Instantaccess
3	Obfuscator.AD
4	Skintrim.N
5	VB.AT
6	Wintrim.BX
7	Yuner.A

Table 4: Class label for complete Malimg dataset.

Label	Malware Class	Label	Malware Class
0	Adialer.C	10	Lolyda
1	Agent.FYI	11	Malex.gen!J
2	Allaple	12	Obfuscator.AD
3	Alueron.gen!J	13	Rbot!gen
4	Autorun.K	14	Skintrim.N
5	C2LOP	15	Swizzor.gen!
6	Dialplatform.B	16	VB.AT
7	Dontovo.A	17	Wintrim.BX
8	Fakerean	18	Yuner.A
9	Instantaccess		

Table 5: Precision, recall, and F1-score for the proposed MLP model with Malimg subset.

Label	Precision	Recall	F1-score	Support
0	0.98	1.00	0.99	295
1	1.00	0.94	0.97	33
2	1.00	0.97	0.99	40
3	1.00	1.00	1.00	15
4	1.00	1.00	1.00	6
5	1.00	0.94	0.97	53
6	1.00	0.92	0.96	12
7	1.00	1.00	1.00	75

5.3 Comparison with Baseline Models

In addition, we evaluate the performance of our proposed MLP model in classifying malware images by comparing its accuracy with existing machine learning-based works [2, 6, 8, 10]. The original paper which publishes the Malimg dataset uses the K Nearest Neighbors (KNN) model [10]. It first pre-processes the images with multiple Gabor filters with different frequencies and orientations in the frequency domain. Then, the texture features are extracted using the GIST-based wavelet decomposition of images, where a total of 320 input features are obtained. It achieves a classification accuracy of 99.93% of the Malimg subset with 10-fold cross-validation. The KNN model obtains an accuracy of 99.2% for the complete Malimg dataset with malware

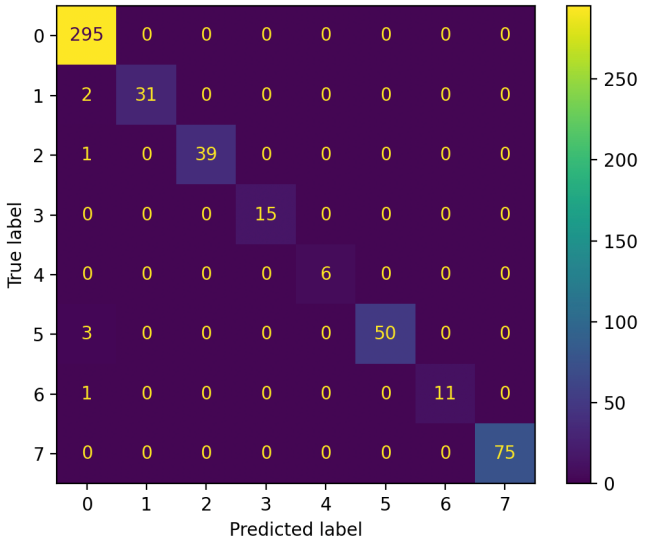

Figure 2: Confusion matrix for the proposed MLP model with Malimg subset (8 classes).

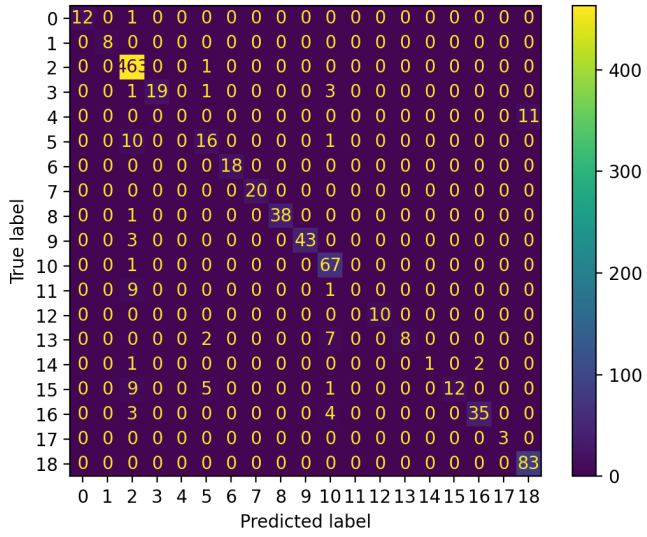

Figure 3: Confusion matrix for the proposed MLP model with Malimg (19 classes).

variants of the same class merged as one. Agarap [2] combines a Support Vector Machine (SVM) with a Gated Recurrent unit (GRU) to classify malware. In addition, different classification models have been proposed for the complete Malimg dataset (without merging variants). The GRU-SVM model uses the resized images of $[32 \times 32]$ and achieves the overall classification accuracy of 84.92%. Convolutional neural networks have also been deployed to achieve the 98.52% and 95.14% testing accuracy in [6, 8], respectively. Table7 summarize the testing accuracy of these baseline models.

Table 6: Precision, recall, and F1-score for the proposed MLP model with complete Malimg dataset.

Label	Precision	Recall	F1-score	Support
0	1.00	0.92	0.96	13
1	1.00	1.00	1.00	8
2	0.92	1.00	0.96	464
3	1.00	0.79	0.88	24
4	0.00	0.00	0.00	11
5	0.64	0.59	0.62	27
6	1.00	1.00	1.00	18
7	1.00	1.00	1.00	20
8	1.00	0.97	0.99	39
9	1.00	0.93	0.97	46
10	0.80	0.99	0.88	68
11	0.00	0.00	0.00	10
12	1.00	1.00	1.00	10
13	1.00	0.47	0.64	17
14	1.00	0.25	0.40	4
15	1.00	0.44	0.62	27
16	0.95	0.83	0.89	42
17	1.00	1.00	1.00	3
18	0.88	1.00	0.94	83

Table 7: Comparison of malware classification testing accuracy with baseline models.

	Model	Accuracy
Malimg subset	**Proposed MLP**	98.7%
	KNN [10]	99.93%
	Transfer Learning	100%
Malimg	**Proposed MLP**	92.3%
	KNN [10]	99.2%
	GRU-SVM [2]	84.92%
	CNN [6]	98.52%
	CNN [8]	95.14%
	Transfer Learning	96.61%

5.4 Benchmarking with Transfer Learning

In addition to comparing the performance of our proposed MLP with the existing works, we also trained our deep learning baseline model in MATLAB. We use the

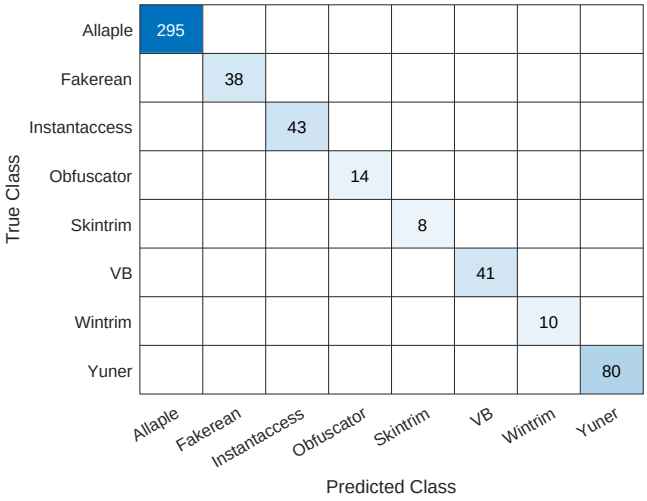

Figure 4: Confusion matrix for the testing accuracy of the Malimg subset (8 classes) dataset with AlexNet and transfer learning.

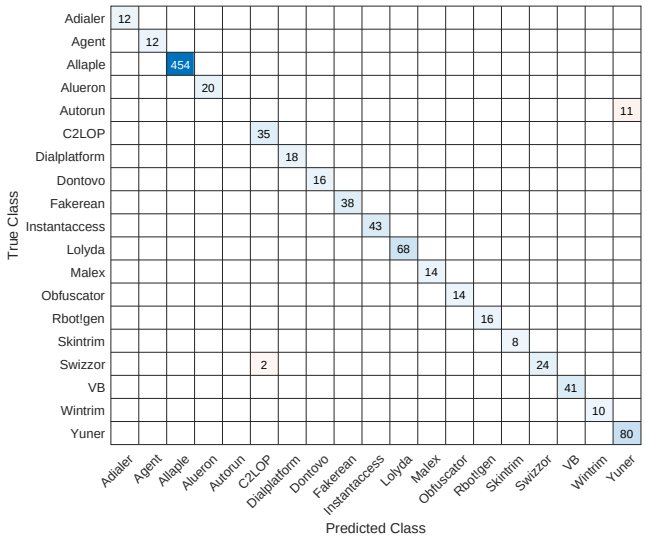

Figure 5: Confusion matrix for the testing accuracy of the complete Malimg (19 classes) dataset with AlexNet and transfer learning.

pre-trained AlexNet with transfer learning to classify malware on both the Malimg subset and the complete Malimg dataset. We reshape the images to [2272273] by replicating the 2*D* matrix 3 times to form the R, G, B channels for the color image used for the input of AlexNet. The number of output classes has been modified to 8 or 19 for the datasets we use. We train the model with 3 epochs using Adam optimizer with a learning rate of 0.001. We obtain a training, validation, and testing accuracy of 100%, 99.81&, and 100% for the Malimg subset. Also, for the Malimg dataset, we achieve the training, validation, and testing accuracy of 99.22%, 98.82%, and 96.61%. The confusion matrix for the testing accuracy for both AlexNet models trained with transfer learning is shown in Figures 4 and 5, respectively. It can be observed that the DNN model with transfer learning performs well, where there are few test samples misclassified as the wrong malware class. It can be inferred that the more hidden layers and convolution computation on images are beneficial for increasing the prediction result.

6 FUTURE DIRECTIONS - FURTHER IMPROVEMENT

Although our proposed MLP model only contains one hidden layer with 256 neurons only, our classification accuracy for both the Malimg subset (8 classes) and the complete Malimg dataset (19 classes) is comparable to the state-of-the-art models. We believe we can further improve our testing accuracy, along with precision, recall, and F1 score for malware classification. Here we list some of the potential directions for our future work to increase the classification accuracy further.

6.1 Adam Optimization

In this project, we used a fixed learning rate of 0.00001 for updating the weights for both the output layer and the hidden layer. As shown in Tables 1 and 2, a total number of 300 epochs is needed to achieve the desired malware classification accuracy. We would like to reduce the training time and the total iterations to pass the entire training dataset. Adam, Adaptive Moment Estimation, is an optimization algorithm for stochastic gradient descent [7]. We tested the proposed MLP in MATLAB with Adam optimizer and 5 epochs on the Malimg dataset. Figure 6 shows the training process with Adam optimization, where the top plot shows the accuracy for training and validation datasets, and the bottom plot visualizes the loss function for both the training and validation datasets. It can be observed that the accuracy reaches over 90% very quickly within the first epoch. It is efficient in finding the possible optimal set of parameters that achieves high classification accuracy. The training, testing, and validation accuracy is 98.44%, 99.25%, and 99.57%, respectively. Figure 7 presents the

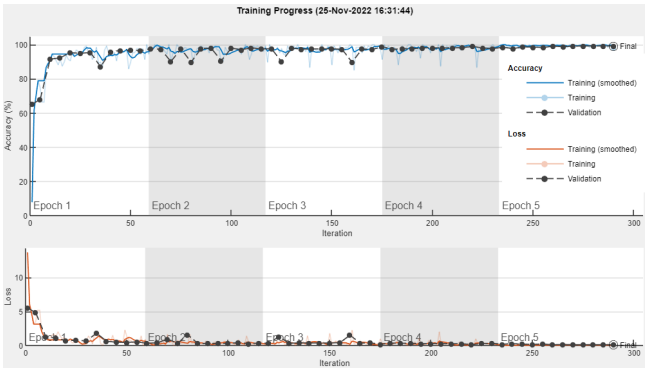

Figure 6: Training process complete Malimg (19 classes) dataset with the proposed MLP model and Adam optimizer. It includes the accuracy and loss for both the training dataset and the validation dataset.

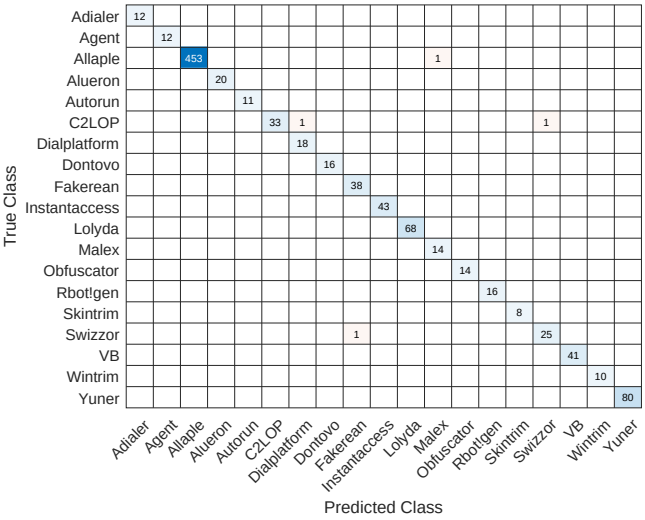

Figure 7: Confusion matrix for the testing accuracy of the complete Malimg (19 classes) dataset with the proposed MLP model and Adam optimizer.

confusion matrix of the testing dataset, where minimal misclassification can be observed. We believe applying the Adam optimizer in our model for future improvement is crucial for obtaining the desired prediction accuracy with fewer training epochs and less time duration.

6.2 Going Deeper with More Hidden Layers

The state-of-the-art DNN models have achieved significant performance boost, comparable to or even surpass human baselines. In addition to embedding 3D/4D convolutions, it is common for DNNs to have the fully-connected multiple hidden layers. Our future goal is to insert additional hidden layers in MLP and assess the accuracy improvement with each added layer. The trade-off between accuracy increase and a more complex MLP structure can be analyzed further.

6.3 Energy Efficiency Optimization

Over the year, energy-efficient machine-learning hardware accelerators have gained significant interest due to the low-latency, and low-power characteristics achieved with the customized hardware and application-specific integrated circuit (ASIC). Eyeriss is one of the forerunners in building the hardware accelerator for DNN [4, 12]. Our future direction is to optimize the proposed MLP in hardware with floating point instructions to reduce the overall energy consumption while preserving the prediction accuracy.

7 CONCLUSION

Malware attacks are on the rise, and there are increasingly more people and devices that fall victim to such attacks every day. Therefore, a machine learning model that can accurately perform malware classification will be extremely useful and beneficial to the digital society. By understanding the type of threat malware poses and how to prevent it, security against such malicious executables can be ensured, and devices can be protected. In this paper, we implemented a multi-layer perceptron (MLP) model with only 1 hidden layer. We have demonstrated that this proposed classification model can help us in achieving high prediction accuracy despite being shallow compared to other complex baselines. The chosen dataset for the project is the Malimg dataset, where malware binaries are converted into grayscale images with 2D byte matrices. We have constructed the backpropagation algorithm for weight updates and implemented the proposed MLP model in Python. We have demonstrated that our model achieves 98.7% and 92.3% testing accuracy for the Malimg subset and the complete dataset merging malware variants of the same type. Our model is further compared with the state-of-the-art CNN and DNN models in the existing literature. We also provide discussion on future directions which we believe can further increase the malware classification accuracy as well as optimize the training process and the overall energy footprint.

AUTHOR CONTRIBUTIONS

In this project,

- Pascal Onyeka worked on implementing the multi-layer perceptron model in python. Explained the different technologies/software used in implementing the model and the different challenges faced during the model implementation. Finally, He worked on the hyper-parameter tuning and the malware classification section of the paper.
- Umar Farouk worked on a python code to implement multi-layer perceptron using Jupyter Notebook. He also wrote the abstract, introduction, and background to introduce the reader to our report. He further then explained the basic definitions and some reasoning, like what malware was and why malware classification was important.
- Yadi Zhong worked on benchmarking with existing machine learning-based approaches, including adapting AlexNet with transfer learning to classify malware and incorporating the Adam optimization to compare the accuracy changes. She worked on sections of the proposed MLP model, derivations for the backpropagation algorithm, comparison with baseline models, benchmarking with transfer learning, and future work.

REFERENCES

[1] Adel Abusitta, Miles Q Li, and Benjamin CM Fung. 2021. Malware classification and composition analysis: A survey of recent developments. *Journal of Information Security and Applications* 59 (2021), 102828.

[2] Abien Fred Agarap. 2017. Towards building an intelligent anti-malware system: a deep learning approach using support vector machine (SVM) for malware classification. *arXiv preprint arXiv:1801.00318* (2017).

[3] Avast. 2016. An in-depth look at the technology behind CyberCapture, https://blog.avast.com/an-in-depth-look-at-the-technology-behind-cybercapture.

[4] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. 2016. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE journal of solid-state circuits* 52, 1 (2016), 127–138.

[5] Deep Instinct. 2022. Malware Classification in Threat Prevention, https://www.deepinstinct.com/pdf/whitepaper-malware-classification-in-threat-prevention.

[6] Mahmoud Kalash, Mrigank Rochan, Noman Mohammed, Neil DB Bruce, Yang Wang, and Farkhund Iqbal. 2018. Malware classification with deep convolutional neural networks. In *2018 9th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 1–5.

[7] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[8] Hugo Mallet. 2020. Malware Classification using Convolutional Neural Networks — Step by Step Tutorial, https://towardsdatascience.com/malware-classification-using-convolutional-neural-networks-step-by-step-tutorial-a3e8d97122f.

[9] Microsoft. 2022. What is malware? Microsoft, https://www.microsoft.com/en-us/security/business/security-101/what-is-malware.

[10] Lakshmanan Nataraj, Sreejith Karthikeyan, Gregoire Jacob, and Bangalore S Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security*. 1–7.

[11] Ikram Ben Abdel Ouahab, Lotfi Elaachak, and Mohammed Bouhorma. 2021. Image-Based Malware Classification Using Multi-layer Perceptron.. In *NISS (Springer)*. 453–464.

[12] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. 2017. Efficient processing of deep neural networks: A tutorial and survey. *Proc. IEEE* 105, 12 (2017), 2295–2329.

[13] Danish Vasan, Mamoun Alazab, Sobia Wassan, Hamad Naeem, Babak Safaei, and Qin Zheng. 2020. IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture. *Computer Networks* 171 (2020), 107138.

[14] COMP-6630-Final-Project-Malware Classification with Multi-Layer Perceptron. 2022. https://github.com/paschal27/Final-Project---COMP-6630.