ISSN: 3026-0442

Proceedings of the 2nd International Conference on Education, Science Technology and Health (2nd I CONESTH 20 24 Bina Bangsa Getsempena University , Sept 10-12, 2 0 24, Banda Aceh, Indonesia

Impact of Hyperparameter Optimizer for Image Malware Detection

Iik Muhammad Malik Matin[1] , Ayu Rosyida Zain[1] , Aaz Muhammad Hafidz Azis[2]

[1] State Polytechnic of Jakarta, Indonesia
[2] Telkom University, Indonesia

***** Corresponding email: *iik.muhamad.malik.matin@tik.pnj.ac.id*

ABSTRACT

Image-based malware detection has become an area of further research in dealing with image-based malware attacks. Various deep learning models have been used to improve detection accuracy. One popular architecture is VGG16, a convolutional network widely used in image classification. In this study, we explore the impact of hyperparameter tuning on the optimization of the VGG16 model for image-based malware detection. The hyperparameter experiments conducted in this study are optimizer, and the number of epochs. Through 6 experiments with parameter variations, we evaluate the performance of the VGG16 model using several SGD, and Adam optimizers and the number of epochs consisting of 100, 250 and 500 epochs. The experimental results show that the selection and tuning of the optimizer can affect the performance of the model in terms of accuracy and training efficiency. The optimized Adam optimizer gives the best results, with higher detection accuracy than the SGD optimizer. The results show that the Adam optimizer has the highest accuracy reaching 85%.

Keywords: *VGG16, Adam, SGD, Optimizer.*

INTRODUCTION

In recent years, cybersecurity threats have increased rapidly. One of the most common attacks is malware. Malware can cause significant losses to individuals and organizations, including data theft, system failures, and service disruptions. (Rathore et al., 2018) . Therefore, developing effective methods to

detect and prevent malware is very important. Machine learning-based malware detection has shown more effective results compared to other methods (Firdausi et al., 2010) . especially with the use of image processing techniques to analyze suspicious files and identify behavioral patterns.

One of the important aspects in developing a machine learning model is hyperparameters. Hyperparameters are parameters that are set before the model training process and can affect the model's performance. In image-based malware detection, selecting the right optimizer and tuning hyperparameters can improve the accuracy and efficiency of detection. Various optimizers, such as Adam, and SGD have different characteristics in the parameter update process that can affect the final result of malware detection.

Currently, research related to image-based malware detection in machine learning has been widely conducted, but studies on image-based malware detection are still limited. Previous studies have been conducted including Image-based malware detection systems can use Convolutional Neural Network (CNN) based model algorithms (Gibert et al., 2018) . Several studies have been conducted by Gibert (Gibert et al., 2018) using the Convolutional Neural Network model with a bytes dataset. In the following year, research was conducted using the same model but using the gray bytes dataset (Gibert et al., 2019) . Kalash et al. used CNN with 25 epochs (Kalash et al., 2018) . While (Le et al., 2018) used CNN with 100 epochs. Transfer learning has been used by Salas et al. (Salas et al., 2023) with achievements on the MobileNet architecture.

Hyperparameter optimization can improve model performance including image recognition. Therefore, this paper aims to conduct hyperparameter optimizer experiments on model performance in detecting malware from file images.

METHODS

Data Collection

In this study, the dataset used is the Maldeb dataset. This dataset consists of 1905 samples divided into 2 labels, namely 1033 malware and 872 benign. Maldeb is an image-based dataset that has been converted from binary samples to images as depicted in Figure 1.

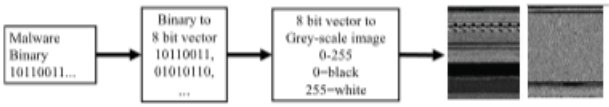

Figure 1. Extraction Method

Extracting binary samples into images produces malware and benign samples as shown in Figure 2.

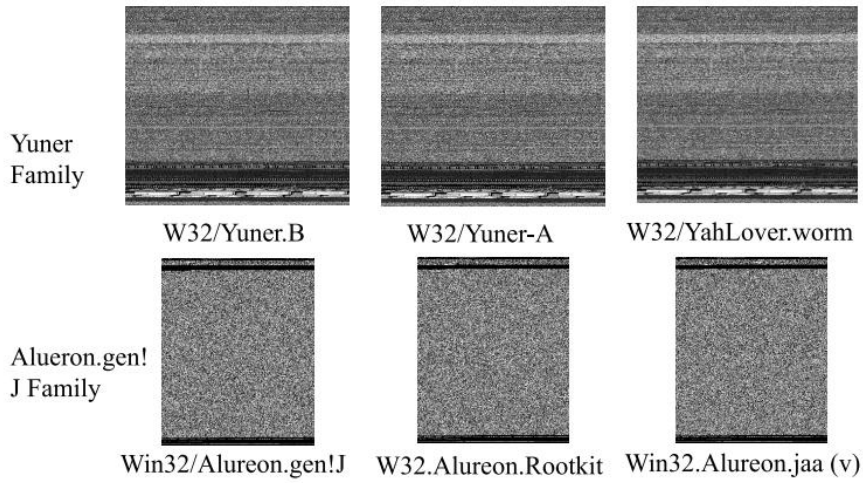

Figure 2. Image Malware Dataset (July & Ismail, 2024)

Design

The model architecture used in this study uses the VGG16 base model that has been pre-trained using the ImageNet dataset. This architecture is built on a series of convolutional layers with 3x3 kernels, which repeatedly capture local patterns from the input image. One of the reasons for choosing the VGG16 architecture is the simplicity of its design which only uses 3x3 filters in each convolutional layer, with a 2x2 pooling layer to reduce spatial dimensions, without sacrificing classification accuracy.

Input Layer

The classification process starts from the input layer, which receives images in a 224x224x3 format, according to the VGG16 architecture standard. In detecting malware, the input data is an image representation generated from a malware file that is converted into a visual image. This visual representation allows the network to capture patterns that are difficult to capture by manual feature-based techniques.

VGG16 as Feature Extractor

The VGG16 model used in this architecture acts as the primary feature extractor. The convolution layers are responsible for capturing the important features of the input image through 13 consecutive convolution layers. Each convolution layer uses a 3x3 kernel, with increasing filter depths from 64, 128, 256, to 512 in each block. After each convolution block, a 2x2 Max Pooling layer is used to reduce the spatial dimension, which helps in reducing the computational complexity while retaining the important features.

Flattening and Dense Layer
After the output of the last layer of VGG16, which is a three-dimensional tensor, this layer is then flattened into a one-dimensional vector. This process is necessary so that the data can be passed to the following Dense layer. The Dense layer added to this architecture consists of 512 neurons and uses the ReLU activation function. This layer is responsible for capturing the non-linear relationship between the features extracted by VGG16 and the malware class label.

Dropout for Regularization
We add a Dropout layer after the Dense layer. This Dropout layer sets the dropout rate to 0.5, which means that half of the neurons in this layer will be randomly deactivated on each training iteration. This way, the model does not rely too much on a small number of neurons, but instead learns to distribute learning across the network.

Output Layer and Softmax Function
The output layer is used to perform classification into malware classes. This layer consists of a number of neurons equal to the number of malware categories in the dataset, and uses the Softmax activation function. The Softmax function converts the output scores into probabilities that represent the likelihood that a particular input belongs to each malware class. .

Optimizer and Training Process
For the training process, we use the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.01 and a momentum of 0.9. SGD was chosen to provide more stable and directional weight updates compared to Adam, especially on relatively large and complex datasets. The use of momentum helps accelerates convergence by preserving the direction of weight updates, thereby avoiding excessive oscillations along steep slopes.

Measurement Method
Maldeb Dataset has 2 types of classification, namely benign and *malware,* so that there can be four possible classification outputs as shown in Figure 1.

		Predicted label	
		Benign	Malware (Intrusion)
True label	Benign	TN	FP
	Malware (Intrusion)	FN	TP

Figure 3. Confusion Matrix

There are 4 possibilities shown in Fig. 3 can be explained as follows:

a. *True Negative* (*TN*)
   True negative measurement results indicate the number of benign tumors that can be correctly identified.
b. *False Negative* (*FN*)
   false negative measurement results show that the classification results are benign, even though what was identified was *malware* .
c. *True Positive* (*TP*)
   True positive measurement results show the number of *malware* that was correctly identified.
d. *False Positive* (FP)

false positive measurement results indicate *malware* when in fact what is identified is benign. The four possible classification outputs are used to measure the model performance consisting of accuracy, precision, TPR (*Recall*) and F1- *score* with the following formula:

$$\text{Accuracy} = \frac{TP+TN}{TP+TN+FP+FN} \quad (1)$$

$$\text{Precision} = \frac{TP}{TP+FP} \quad (2)$$

$$TPR = \frac{TP}{TP+FN} \quad (3)$$

$$F - Score-= 2 \times \frac{Precision \times TPR}{Precision+TPR} \quad (4)$$

Accuracy (1) is the ratio of correctly identified malware. Precision (2) is the relevance in identifying *malware from the clarification results given by the output* . *True Positive Rate* (TPR) or recall (3) is the ratio of the success rate in identifying *malware* . *F-score* (4) is the accuracy rate calculated based on the precision and recall of a test.

RESULTS AND DISCUSSION

This experiment was conducted using the VGG16 model with variations of epochs (100, 250, 500) and optimizers (Adam, SGD) to detect malware. The results are shown in table 1.

Table 1. Experiment Result

Optimizer	*Epoch*	*Precision*	*Recall*	*F1-Score*	*Accuracy*
Adam	100	93	76	84	85
Adam	250	92	76	84	84
Adam	500	94	76	84	86
SGD	100	94	71	81	82
SGD	250	88	67	76	78
SGD	500	88	67	77	79

Adam Optimizer

The model trained with Adam showed consistent results and tended to be better than SGD across a range of epochs. At 100 epochs, the model with Adam

optimizer produced a precision of 93%, a recall of 76%, an F1-score of 84%, and an accuracy of 85%. This shows that the model is quite good at detecting malware accurately with few false positives or false negatives.
At 250 epochs, precision drops slightly to 92%, but recall remains stable at 76%. This results in a stable F1-score of 84 and a slight decrease in accuracy to 84%. This slight decrease is likely due to slight overfitting after a higher number of epochs.
At 500 epochs, precision has increased again to 94%, with recall remaining at 76%. However, the F1-score remains at 84, and accuracy has increased slightly to 86%. Adam seems to be quite stable in maintaining good performance even over longer training periods.

SGD Optimizer
The model trained using the SGD optimizer showed more varied results with performance that tended to be lower than Adam, especially in terms of recall. At 100 epochs, the model trained with SGD showed the same precision as Adam, which was 94%, but the recall was lower at 71%. This indicates that although the model was quite good at identifying detected malware, there were many cases of malware that were not detected (many false negatives). As a result, the F1-score dropped to 81, with an accuracy of 82%.
At 250 epochs, the model performance with SGD began to decline. Precision dropped to 88%, while recall remained low at 67%. The F1-score also dropped to 76, and accuracy only reached 78%. This decline may be due to SGD's inability to optimize the model more effectively as the number of epochs increases.
At 500 epochs, the downward trend continues. Precision remains at 88%, and recall does not improve, remaining at 67%. The F1-score and accuracy are at 77 and 79%, respectively, indicating that even though the model has been trained longer, its performance has not improved significantly and tends to be worse than Adam.
From these results, it is clear that the Adam optimizer is superior to SGD in terms of overall performance. Adam is able to maintain stable performance even when the number of epochs increases, while SGD tends to decrease in performance as the number of epochs increases. This may be due to Adam's adaptive ability to adjust the learning rate based on the gradient, allowing it to achieve faster and more stable convergence.
In contrast, SGD is more sensitive to the number of epochs and tends to get stuck in local minima or have difficulty optimizing the model properly. Although SGD has high precision values at the beginning of training, low recall indicates that the model often fails to detect malware, which in real-world scenarios can be fatal.
The low recall performance of SGD is a serious concern in malware detection applications. In security cases, errors in detecting malware (i.e., false negatives) are more harmful than false positives. Therefore, optimization with Adam is

more recommended for these malware detection tasks because it offers a better balance between precision and recall.

CONCLUSION

Based on the results of this experiment, it can be concluded that the Adam optimizer provides more stable and better performance compared to SGD in detecting malware using the VGG16-based transfer learning model. With higher precision, recall, F1-score, and accuracy, Adam is able to handle the overfitting problem better and offers better generalization to the validation data. In contrast, SGD shows less than optimal performance, especially in terms of recall, indicating that the model often fails to detect malware well. In malware detection, the use of the Adam optimizer is more recommended because of its ability to maximize detection and reduce false negatives .

REFERENCES

Firdausi, I., Lim, C., Erwin, A., & Nugroho, AS (2010). Analysis of machine learning techniques used in behavior-based malware detection. *Proceedings - 2010 2nd International Conference on Advances in Computing, Control and Telecommunication Technologies, ACT 2010* , 201–203. https://doi.org/10.1109/ACT.2010.33

Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2018). Classification of malware by using structural entropy on convolutional neural networks. *Proceedings of the 30th Innovative Applications of Artificial Intelligence Conference, IAAI 2018* , 7759–7764. https://doi.org/10.1609/aaai.v32i1.11409

Gibert, D., Mateu, C., Planes, J., & Vicens, R. (2019). Using convolutional neural networks for classification of malware represented as images. *Journal of Computer Virology and Hacking Techniques* , *15* (1), 15–28. https://doi.org/10.1007/s11416-018-0323-0

July, S., & Ismail, I. (2024). MalSSL — Self-Supervised Learning for Accurate and Label-Efficient Malware Classification. *IEEE Access* , *12* (March), 58823–58835. https://doi.org/10.1109/ACCESS.2024.3392251

Kalash, M., Rochan, M., Mohammed, N., Bruce, NDB, Wang, Y., & Iqbal, F. (2018). Malware Classification with Deep Convolutional Neural Networks. *2018 9th IFIP International Conference on New Technologies, Mobility and Security, NTMS 2018 - Proceedings , 2018 - January* , 1–5. https://doi.org/10.1109/NTMS.2018.8328749

Le, Q., Boydell, O., Namee, B. Mac, & Scanlon, M. (2018). Deep learning at the shallow end: Malware classification for non-domain experts. *Proceedings of the Digital Forensic Research Conference, DFRWS 2018*

USA , *26* , S118–S126. https://doi.org/10.1016/j.diin.2018.04.024

Rathore, H., Agarwal, S., Sahay, S. K., & Sewak, M. (2018). Malware detection using machine learning and deep learning. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* , *11297 LNCS* , 402–411. https://doi.org/10.1007/978-3-030-04780-1_28

Salas, M.I.P., de Geus, P.L., & Botacin, M.F. (2023). Enhancing Malware Family Classification in the Microsoft Challenge Dataset via Transfer Learning. *ACM International Conference Proceedings Series* , 156–163. https://doi.org/10.1145/3615366.3615374