# Malware Classification Through Computer Vision

Prasann Viswanathan Iyer
*Department of Electrical Engineering, IIT Bombay*
Mumbai, India
prasann@iitb.ac.in

***Abstract*—We propose to apply a transformation to static malware byte files to convert them into RGB images. Following this, we use transfer learning and Decision Tree Classification on the resulting images. The objective of this Machine Learning Classification scheme is to see whether the transformation of Malware Byte Code to RGB images improves various evaluation metrics of classification, such as Accuracy, False Positive Rate, True Positive Rate and *F1* score. Finally, if time permits, we plan to explore the possibility of having a convex combination of the models.**

***Index Terms*—Classification, computer vision, decision trees, machine learning, static malware, transfer learning**

## I. Introduction

The term static malware means to classify the files without executing the application or monitoring its runtime behaviour. The textbook method of static malware classification is signature matching, which checks if strings of code match malicious patterns in our database. However, this approach is discarded as it is not robust to obfuscations or changes in code and is neither scalable when the number of signatures grows exponentially.

Consequentially, machine learning has proven to be quite effective for large-scale malware classification. However, feature selection for this classification task is cumbersome. It requires a disassembly step to make the code human-readable and is followed by counting the number of loops, binning the type of function calls, etc. Such processing results in large feature spaces, which further need to be reduced through techniques such as PCA and K-PCA. Therefore we desire a method without much manual effort in feature construction.

To solve these issues, our implementation will consider malware classification as a computer vision problem. The proposed method will consume the entire malware application binary irrespective of code obfuscation and is completely independent of signatures.

Finally, we test out the transformed data through a transfer learning neural network and compare its performance with a Decision Tree Classifier to see if converting byte code to RGB images improves on classification metrics.

## II. Methodology and Implementation

The inspiration [**?**] for our method came from visual analysis of the malware binaries rendered in grayscale, where it is seen that malware from the same family shares structural similarities while malware from other families displays distinctive structural or textural characteristics. The malware classification challenge may be approached as a vision classification task with the use of such visual examination.

### A. Preprocessing

Given a static binary, we map it directly to an array of integers between 0 and 255. Hence each binary is converted into a one-dimensional array $v \in [0, 255]$. Then the array v is normalized to $[0, 1]$ by dividing by 255. The normalized array $v$ is then reshaped into a two-dimensional array $v$ by the rules mentioned in the below table:

| File Size (in Kb) | Width |
|---|---|
| $(0, 10]$ | 32 or remove samples |
| $(10, 30]$ | 64 |
| $(30, 60]$ | 128 |
| $(60, 100]$ | 256 |
| $(100, 200]$ | 384 |
| $(200, 500]$ | 512 |
| $(500, 1000]$ | 768 |
| $(1000, 2000]$ | 1024 |
| $> 2000$ | 2048 |

Thus we reshape one-dimensional array $v$ into a two-dimensional array $v'$ using the above relationship.

The grey-scale photos are resized into a $m \times m$ matrix. To suit the input size of the pre-trained models using ImageNet, the option of $m$ is chosen to be 128. Using the size of 299 was desirable by the research paper specifications but the resulting parameters were too much for my limited RAM. We duplicate the channel three times to create RGB pictures after scaling the two-dimensional arrays into squared grey-scale images. Below you can see an example of a resultant generated image: It is

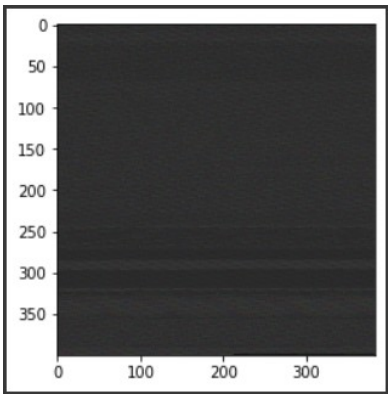


Fig. 1. Example of a Static Malware Image

also important to check the classes of Malware as they are distributed across training and cross-validation sets, which is shown in the bar chart below:

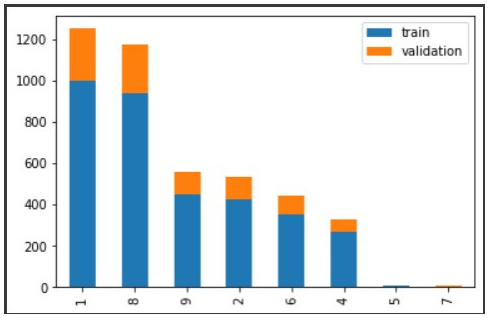


Fig. 2. Class Counts Bar Chart

### B. Training our Model

In the second stage, transfer learning is applied to the scaled and rearranged photos, with some of the layers frozen and the final few layers being retrained on malware images. Our architecture is adaptable to support a variety of deep learning neural networks, including Inception, VGG, ResNet, and DenseNet, because of the transfer learning strategy. We significantly reduce the time spent looking for neural network designs, parameters, and optimizers by using transfer learning. We also contrasted training from scratch with training using transfer learning. We have a lot fewer malware photos than there are images in ImageNet or other models, so training massive deep neural networks from scratch might not converge. Therefore, for comparison, we utilise scaled-down deep neural network models instead.

For the purpose of practical results generation I tried the Inception Model and the VGG model. The Inception model gave extremely poor results because it badly overfitted on the training samples. The VGG model on the other hand has fewer parameters and thus avoids overfitting. Below are the parameter sets for VGG:

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 image_input (InputLayer)    [(None, 128, 128, 3)]     0

 vgg16 (Functional)          (None, None, None, 512)   14714688

 flatten (Flatten)           (None, 8192)              0

 fc1 (Dense)                 (None, 512)               4194816

 fc2 (Dense)                 (None, 512)               262656

 predictions (Dense)         (None, 8)                 4104

=================================================================
Total params: 19,176,264
Trainable params: 19,176,264
Non-trainable params: 0
_________________________________________________________________
```

Fig. 3. Model Structure and Parameters

### C. Evaluation

As assessment criteria for our suggested technique, we employ classification accuracy, false positive rate, true positive rate, and F1 score (in binary classification). If our suggested approach exceeds all other chosen machine learning algorithms in terms of all these metrics on real-world datasets, we'll demonstrate this in a subsequent section. Unfortunately, the model does not perform exceedingly well and this is mainly due to my system limitations. I ran the code on 4307 malware files belonging to 9 classes and this in itself was more than 1.5Gb worth of data. The actual dataset is 500Gb large. With greater computation ability I do expect to generate much better results, up to the level of state-of-the-art malware classification models of accuracy 99%. Below we look at the results produced by the VGG transfer learning model:

Firstly we have accuracy and cross validation scores across different epochs: The graph shows consistent increase in

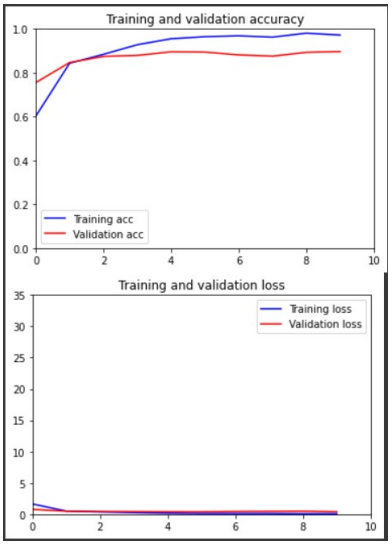


Fig. 4. Accuracy and Loss across training Epochs

accuracy on the testing set, which wasn't the case with the Inception model where there was significant overfitting. The VGG model is well-fit. Next we have the confusion matrix plotted across the 9 classes: We see clearly that the matrix

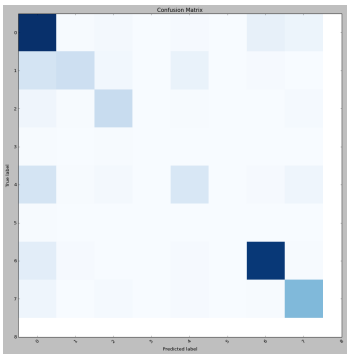

Fig. 5. Class-wise Confusion-Matrix

values for class 1 and 8 are the most promising and this is directly proportionate to the number of training samples available for both these classes. If the classes were well distributed, The matrix would have been ideal. Finally below I depict the performance scores of the model: These are decent

| | precision | recall | f1-score | N Obs |
|---|---|---|---|---|
| 0 | 0.65 | 0.86 | 0.74 | 251 |
| 1 | 0.90 | 0.43 | 0.58 | 107 |
| 2 | 0.72 | 0.79 | 0.75 | 66 |
| 3 | 0.00 | 0.00 | 0.00 | 2 |
| 4 | 0.60 | 0.38 | 0.46 | 88 |
| 5 | 0.00 | 0.00 | 0.00 | 1 |
| 6 | 0.90 | 0.89 | 0.89 | 235 |
| 7 | 0.79 | 0.86 | 0.82 | 112 |
| accuracy | | | 0.75 | 862 |
| macro avg | 0.57 | 0.52 | 0.53 | 862 |
| weighted avg | 0.77 | 0.75 | 0.74 | 862 |

Fig. 6. Precision Recall and F1 Scores

considering the limited training samples across classes.

### *D. Interpretation of Classification Results*

Once the results have been generated, we will interpret them to check whether they are consistent to avoid the case of the models performing poorly in real-life scenarios.
We effectively use the local interpretable model-agnostic explanation method to deliver interpretation. Super pixels, which are areas or patches of pixels next to one another, are the initial method we use to depict the virus pictures. The virus graphics incorporate the super-pixel representation as a visually comprehensible feature. Then, each superpixel has a binary vector imposed on it, where 0 denotes the lack of the superpixel and 1 denotes its existence. To train on the binary vectors and get the learned coefficients, we employ sparse linear classifiers. The areas of pixels associated with each super pixel's positive coefficients are those that contribute to the model's classification decision, while those associated with each super pixel's negative coefficients are those that do not. I wasn't able to do the super-pixel method as suggested by the paper.

### *E. Future Work Directions*

The study addressed a number of additional topics that I haven't been able to, primarily due to my lack of computing abilities. I'll go through a few of them below.

1) In order to compare it to the pre-trained model from Keras, the research investigated the process of building a smaller neural network from scratch. Unfortunately, I lack the computational skills necessary to investigate this. The article, however, was unable to compare the performance of the freshly trained model to that of the previously trained Keras model. As a result, I am not very concerned that not creating one from scratch would alter my conclusions.
2) The original research also investigated whether malware should be classified as either a benign file or a harmful file rather than according to their family. Unfortunately, the information I have prevents this from being achievable. I am unable to investigate this since the binary files I have access to are all malicious files. I do believe that as a whole, this topic would be more engaging and beneficial. Knowing if a file is harmful is more important to me than knowing what malware family it belongs to, at least.
3) I am aware that my data's validation performance is not exactly on par with the paper's claimed performance (my final model's accuracy is 90%, compared to the paper's reported accuracy of 99%). Furthermore, I am aware that the discrepancy between validation performance at 90% and training performance at 99% would undoubtedly point to overfitting. I believe I could improve performance if I knew which layers to include in the VGG16 model to assist prevent this.

### *F. Conclusion*

I'm using Keras pre-trained models for transfer learning for the first time, and I'm blown away by how easy the interface is to use and how precise a pre-trained model can be. Being able to perform so well on RGB photos of malware byte code with only a little fine tweaking is amazing considering that the majority of these models were trained on photographs of dogs, cats, and other pets.
I understand there are portions of the article, such as with the super-pixel technique, that I was unable to touch. Even though I made an effort to cover the essential elements, there were other little subtleties. It is worthwhile to investigate the overlooked methods if I come across a solid data collection that contains both harmful and benign byte code files.

### G. Flowchart

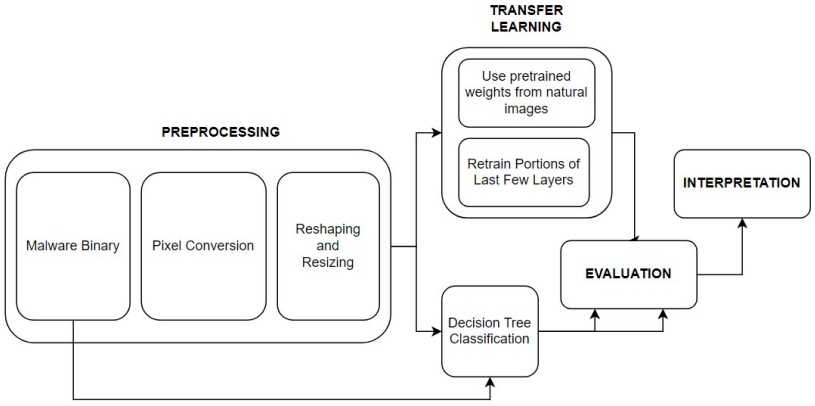


Fig. 7. The implementation Flow Diagram

## REFERENCES

B1 Chen, L. (2018). Deep Transfer Learning for Static Malware Classification. arXiv. https://doi.org/10.48550/arXiv.1812.07606