CSE3501 J Component Project Report

Faculty:Prof. Prasad M

# **Title**

Image Malware Detection using Deep Learning

# **Team**

Pranaydeep Mayank 19BAI1026

Shekhar Iyer 19BAI1055

# **Abstract**

We propose a simple yet effective method for visualizing and classifying malware using image processing techniques. Malware binaries are visualized as gray-scale images, with the observation that for many malware families, the images belonging to the same family appear very similar in layout and texture. Motivated by this visual similarity, a classification method using standard image features is proposed. Neither disassembly nor code execution is required for classification.

Preliminary experimental results are quite promising with CNN classification accuracy on a malware database of 9,339 samples with 25 different malware families

# **Introduction**

Traditional approaches to analysing malware involve the extraction of binary signatures. Due to the rapid increase in the number of new malware signatures, it has become difficult to analyze them. Other techniques such as static code analysis or dynamic code analysis can also be utilized to identify and remove malicious software. Static analysis works by disassembling the code and exploring the control flow of the executable to look for malicious patterns. On the other hand, dynamic analysis works by executing the code in a virtual environment and a behavioral report characterizing the executable is generated based on the execution trace. Both these techniques have their pros and cons. Static analysis offers the most complete coverage but it usually suffers from code obfuscation. Dynamic analysis is more efficient and does not need the executable to be unpacked or decrypted. However, it is time intensive and resource consuming, thus raising scalability issues.

Moreover, some malicious behaviors might be unobserved because the environment does not satisfy the triggering conditions.

Hence converting malware byte code to grayscale image and then classifying it is much more efficient way of detecting malwares.

# **ALGORITHMS USED:**

# **1.)Convolutional Neural networks (CNN)**

Convolutional Neural networks are a subset of machine learning, and they are at the heart of deep learning algorithms. They are comprised of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network.

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

* Convolutional layer
* Pooling layer
* Fully-connected (FC) layer

# **2.)Transfer Learning Algorithms**

# **Expected Outcome**

Final deliverable of our project will be a deep learning model that converts a malware byte code to greyscale image and then detect it.

Trojan  DEEP LEARNING MODEL MALWARE

(grayscale image) DETECTED

(Bytecode file)

**Methodology:**

We will be using deep learning to make the classification.

1.) Convert Malware byte code to png image.

2.) Create deep learning models for classification of these images:

* CNN
* CNN with inception model
* CNN with Texture filters

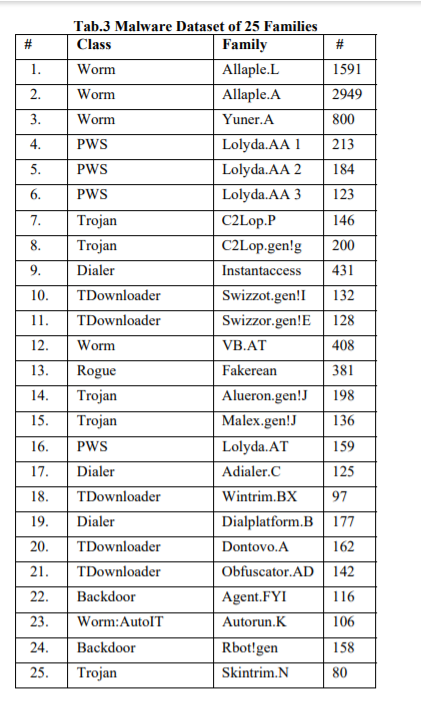
3.) Use this built model to predict any intruder malware in the system.

**Dataset Used:**

**Malimg Dataset.**

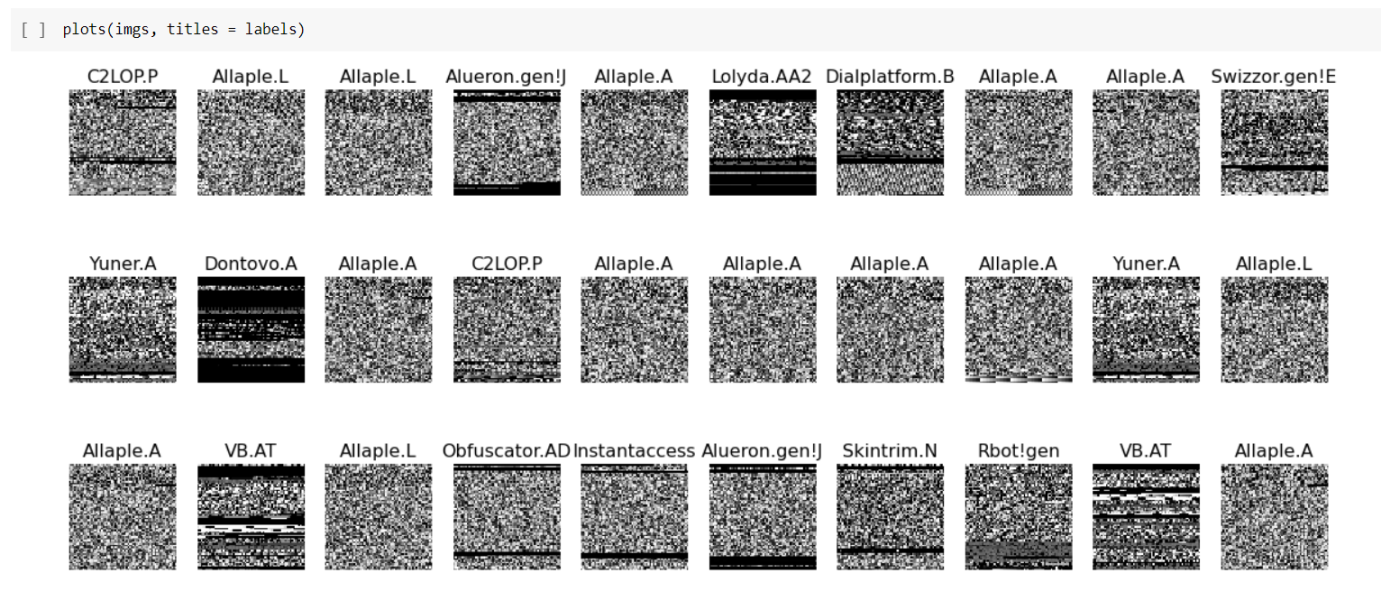
The Malimg dataset consists of 9339 images of 25 classes, and hence these samples require no pre-processing before applying image-based analysis.

Malware Classes:

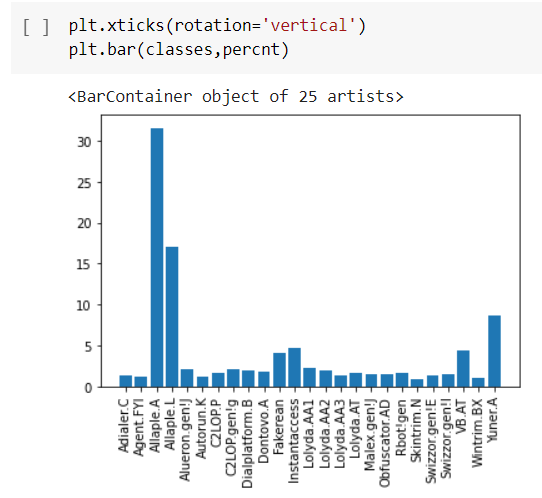


**Experiments and Results:**

**Malware Dataset after converting to grayscale Image:**



Dataset Distribution:

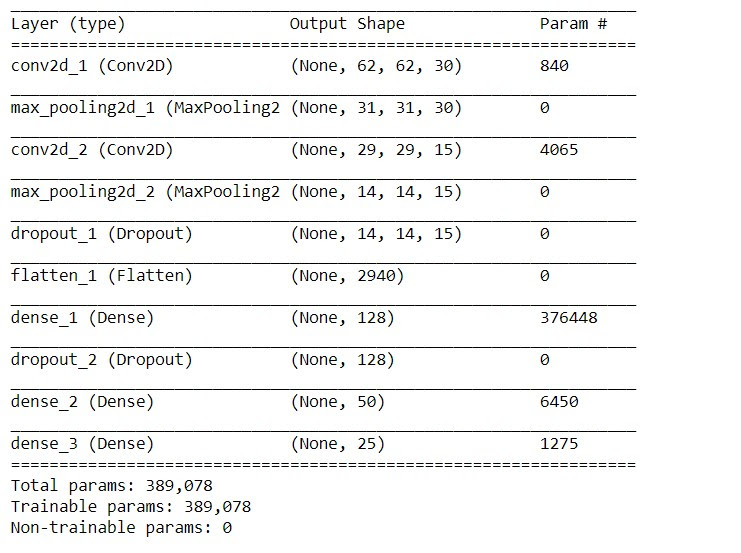


We can see that our dataset is quite unbalanced: a lot of Malwares belong to class 2 : Allaple.A and class 3 : Allaple.L

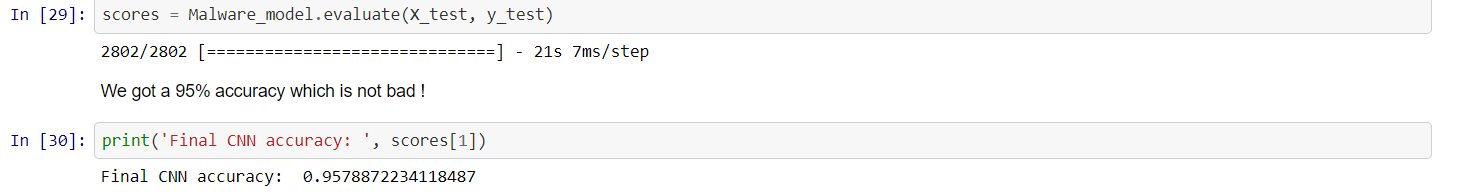
Built CNN model will have the following layers :

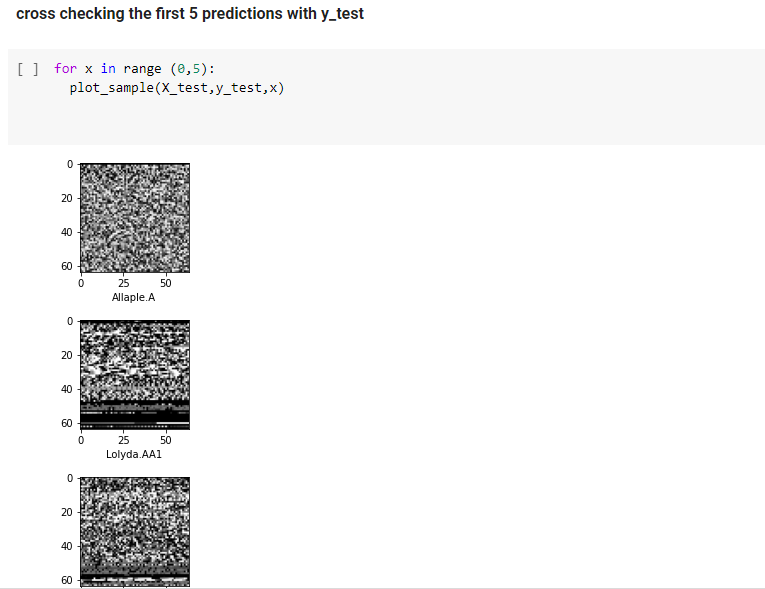
* Convolutional Layer : 30 filters, (3 \* 3) kernel size
* Max Pooling Layer : (2 \* 2) pool size
* Convolutional Layer : 15 filters, (3 \* 3) kernel size
* Max Pooling Layer : (2 \* 2) pool size
* DropOut Layer : Dropping 25% of neurons.
* Flatten Layer
* Dense/Fully Connected Layer : 128 Neurons, Relu activation function
* DropOut Layer : Dropping 50% of neurons.
* Dense/Fully Connected Layer : 50 Neurons, Softmax activation function
* Dense/Fully Connected Layer : num\_class Neurons, Softmax activation function
* Input shape : 64X 64 X3

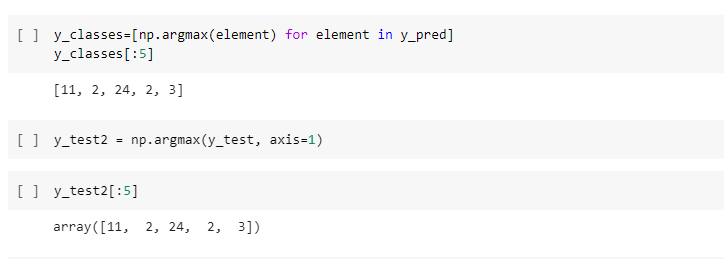
**CNN MODEL Description:**



Model Accuracy:

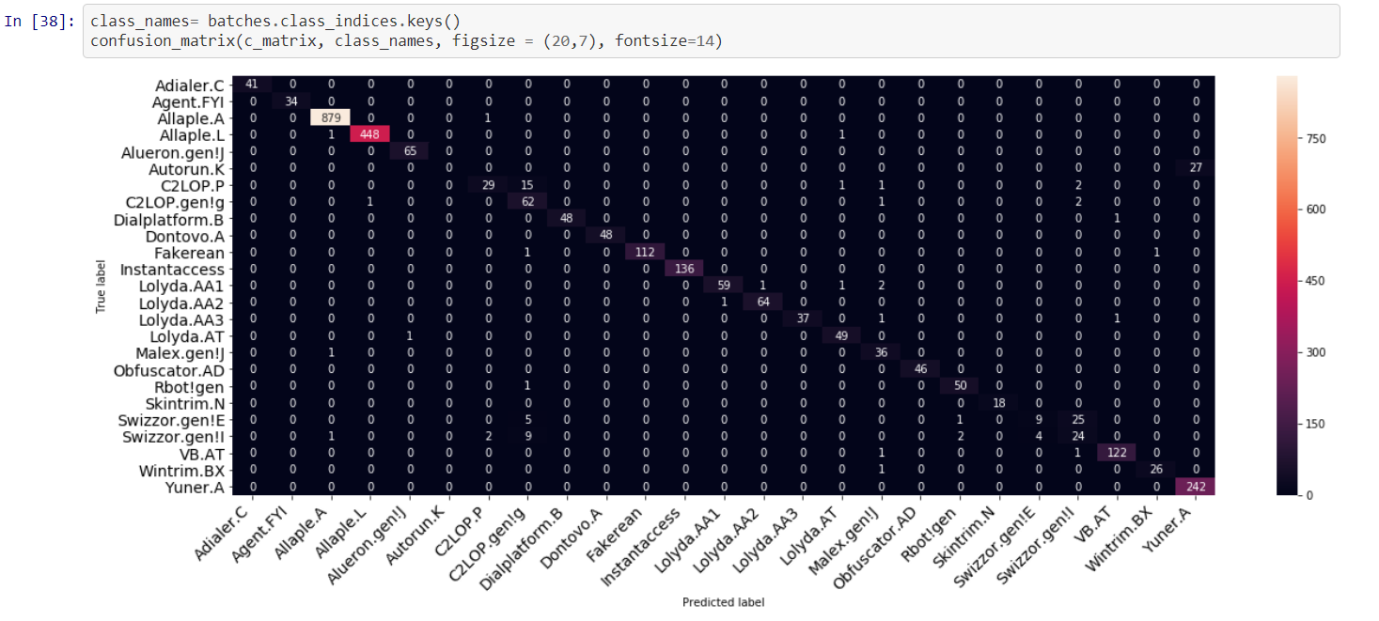






We can see that the first 5 prediction exactly matches to the first 5 Y\_test values.

Confusion Matrix:



We can observe that although most of the malwares were well classified, Autorun.K is always mistaken for Yuner.A.

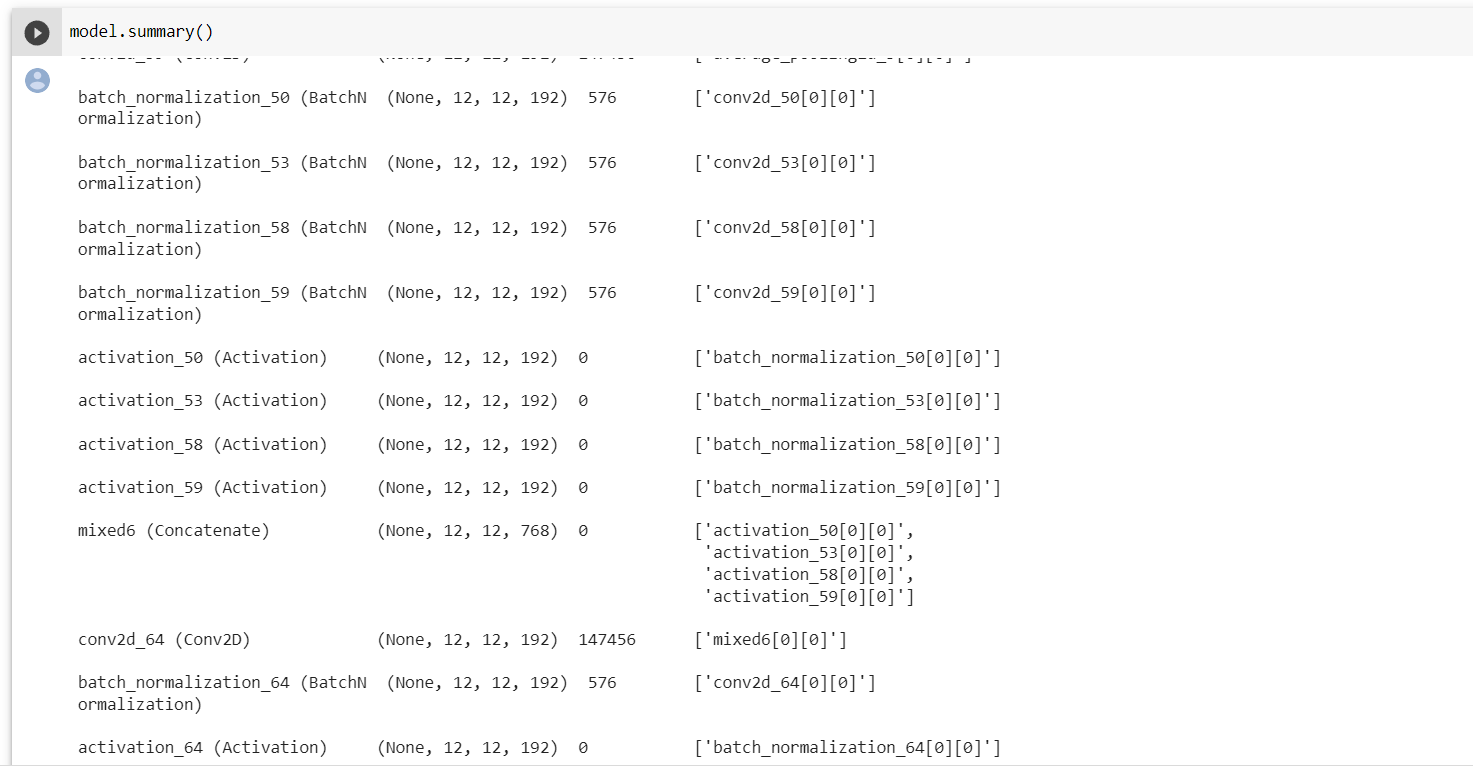
This is probably due to the fact that we have very few samples of Autorun.K in our training set. Moreover, Swizzor.gen!E is often mistaken with Swizzor.gen!l, which can be explained by the fact that they come from really close families and thus could have similarities in their code.

**Transfer Learning Model used for Malimg**

We have used ResNet50, InceptionNet, VggNet Architectures

Below given is the output for ResNet50 model





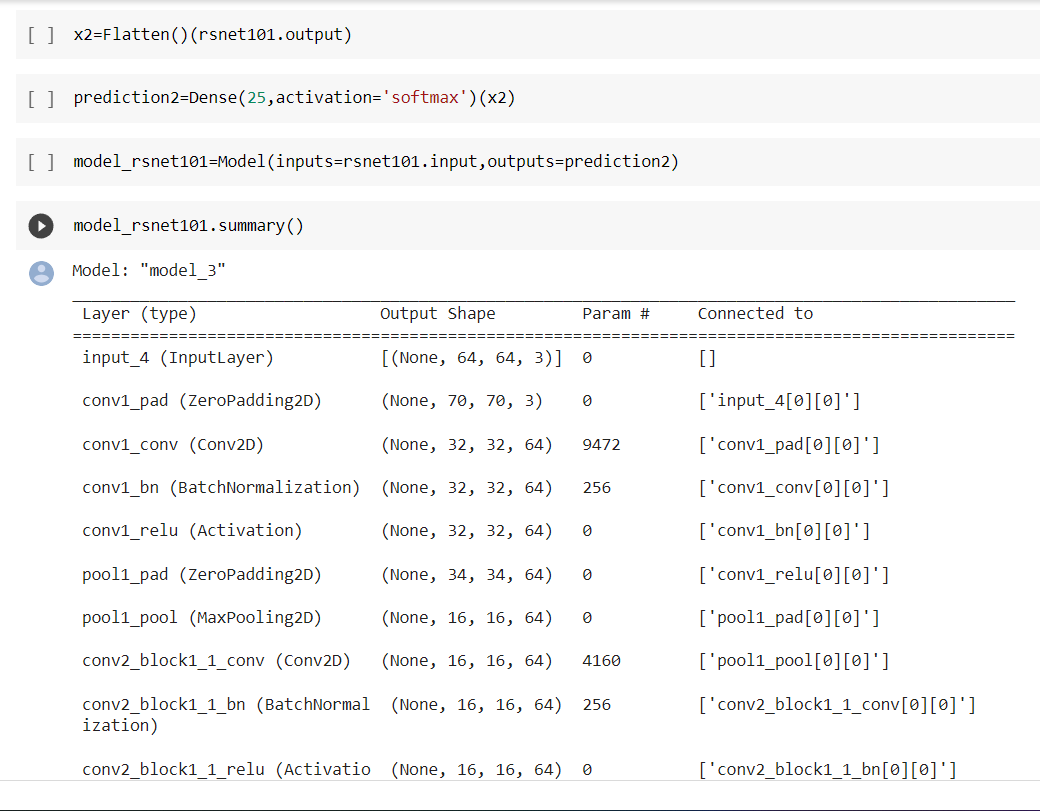
For InceptionV3 Architecture





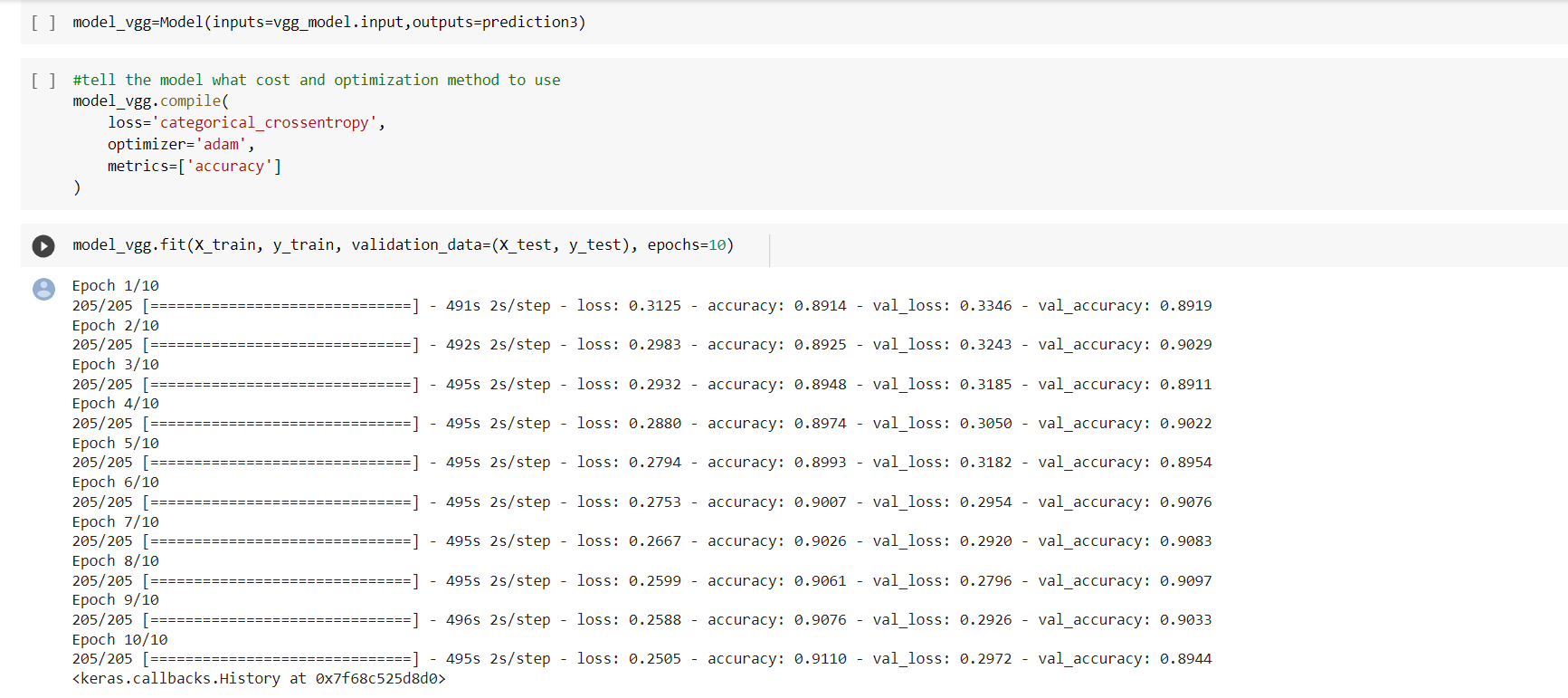
ResNet101 Architecture





VGG19 Architecture





From the above transfer Learning Techniques vgg19 gave the best results with an accuracy of 91%.

**CONCLUSION:**

The proposed methodology of converting a malware byte code to an image and then deploying a Deep Learning Model to detect it was successfully implemented .