**Deep Malware: Malware Image Classification using Deep Learning**

**Project Goal:**

The aim of this project is to develop a Malware Image Classification using Deep Learning

**Problem Statement:**

The rapid development in the field of communication and networks has increased the size and complexity of the network. Due to these reasons, many malwares are generated that create a challenge for systems to detect these malwares accurately. Moreover, the presence of malicious software (malware) with the aim of launching various malware files within the network cannot be ignored. Although, there are numerous efforts by the researchers to develop procedures for automatic classification of malware. The methods of manually analyzing malware file are very time-consuming. Lately, deep learning-based methods are being used for the classification of malware. In this paper, we present a rapid and accurate malware classification based on different Convolutional Neural Network (CNN) architectures—including a custom CNN as well as commodity off-the shelf CNN architectures such as VGG-16, ResNet-50, Inceptionv3 models. This has been demonstrated on benchmark datasets of Malimg dataset, which is consists of malware images that were obtained after conversion of Malware binaries

**Dataset Used**

The Malimg dataset is basically consists of 9458 malware samples which were split into 25 classes. The major feature of this dataset is that they are not providing malware samples once, but alternatively their images as they seem on disk. In a related way to the work in. Bytes of executable files are inconsequential assigned to floats, which will later be elucidate as pixel values of the grayscale image. Unsurprisingly, the categories in the dataset are imbalanced the largest class (‘Allaple.A’) contains 2949 samples, While the smallest class contains only 80 samples. In Figure 2 shows some of the random samples taken from dataset. It is clear that the images in each category have different styles that allow to distinguish between the samples of a family no matter what samples they are in another family. Dataset is illustrated in figure as shown below.

Dataset Division The dataset is divided into two subsets: training data and test data. From each class, 70% images are allocated for training the network while remaining 30% are allocated for testing the trained network

Data Set Link :

https://drive.google.com/drive/folders/1UdnPOtpZu-sz-vYdYcFQj01iGZHYStZx?usp=sharing



**Experimental phase:**

**Requirements**

The main requirements are listed below, You can run the code in Google Colab Directly

* Python 3.6.10
* Numpy
* Keras 2.7.0
* Tensorflow 2.7.0
* Pillow 6.1
* Scikit-learn
* Pandas
* Seaborn
* Matplotlib
* Tqdm

**Experiment:**

**Prepare Data**

* We rescale by 1.0/255 to normalize the rgb values if they are in range 0-255 the values are too high for good model performance

**Training:**

* Data distribution:

70% Training and 30% Test.

* Loss Function:

Categorical Cross entropy

* Evaluation Metrics:

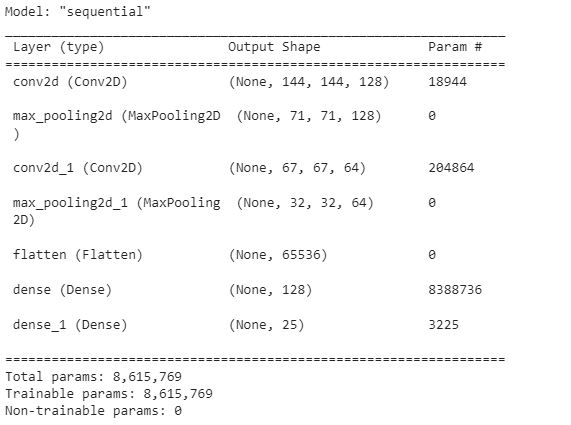
Accuracy

* Optimizer:

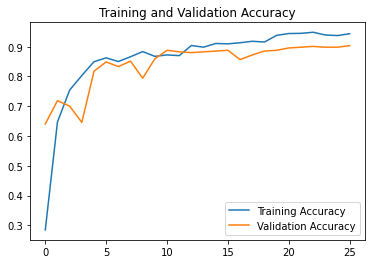
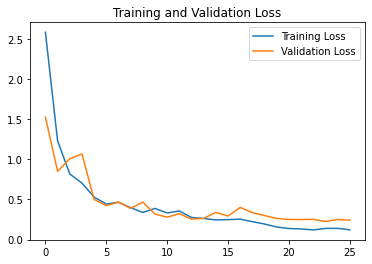
Adam Optimizer

* Reducing Learning Rate on plateau feature to improve training.
* Use Early Stopping Method

**Results:**

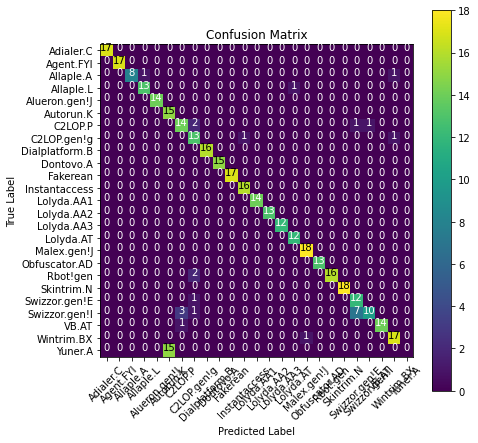
**Custom CNN:**

**Training and Validation Accuracy** **Training and Validation Loss**

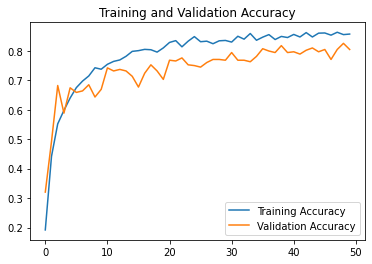
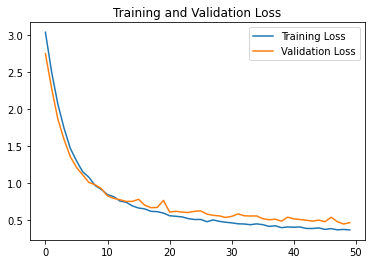


**Confusion Matrix**

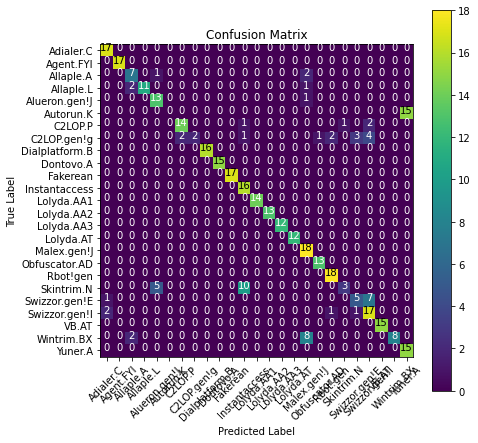
**Accuracy 0.90**

****

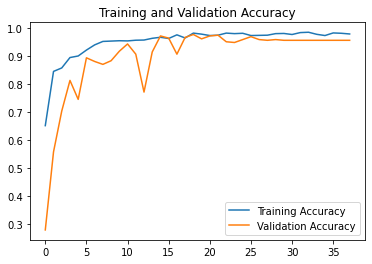
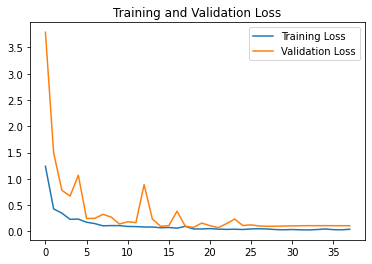
**VGG16 Model:**

**Training and Validation Accuracy** **Training and Validation Loss**

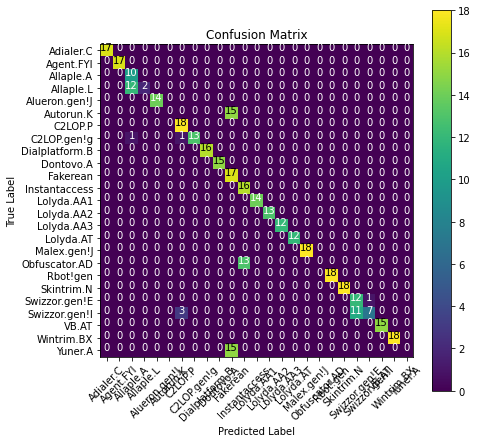
**Confusion Matrix**

**Accuracy 0.80**

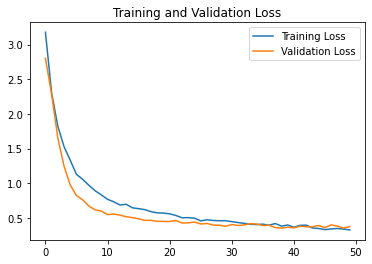
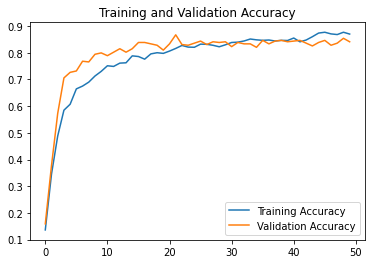
**ResNet50 Model:**

**Training and Validation Accuracy** **Training and Validation Loss**

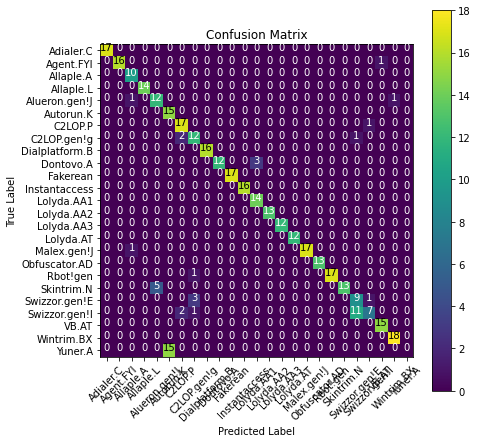
**Confusion Matrix**

**Accuracy 0.81**

**InceptionV3 Model:**

**Training and Validation Accuracy** **Training and Validation Loss**

**Confusion Matrix**

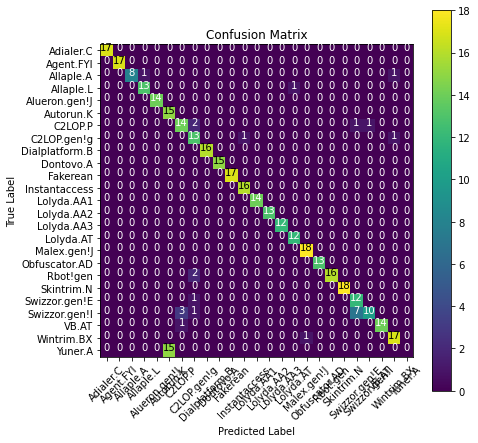
**Accuracy 0.87**

**Ensemble Model:**

### **Combining the best models of VGG16, Resnet50 & InceptionV3**

**Confusion Matrix**

**Accuracy 0.92**

****