MALWARE CLASSIFICATION THROUGH TRANSFER LEARNING

Priyansu Panda

ABSTRACT:

"A Malware is a generic term that describes any malicious code or program that can be harmful to systems". Malware compromises privacy, allows unauthorised access to system resources, and engages in other abusive behaviours. In 2017 Kaspersky Lab detected 360,000 new malicious files per day. Given the motivation of malware authors to write such programmes for illegal financial gain, this alarmingly high number is likely to rise in the future.
Traditional antivirus products have dominated the security market for many years and are generally extremely effective. However, malware authors' ability to invent technology and tactics has grown exponentially over decades of development. According to research, traditional antivirus products have rarely managed to succeed in detecting unknown malware over the last decade.

By converting the malware files into images, we can turn the problem into an image classification which can be solved by Deep Learning Neural networks. The malware files basically exist in .asm or. bytes file containing meta data. It can be converted to RGB images to apply a neural network. Here we have considered Xception model from Transfer Learning to build the concerned neural network.

TECHNOLOGY STACK:

- Image Processing
- Convolutional Neural Network
- Transfer Learning – Xception Architecture

LIBRARY USED:

Pandas, Numpy, Seaborn, Matplotlib, Scipy, Tensorflow, Keras, Scikit Learn

TRANSFER LEARNING:

Transfer learning is a deep learning approach in which knowledge is transferred from one model to another. Transfer Learning is remarkably useful in Natural Language Processing, Speech Recognition and Computer vision. Its advantages

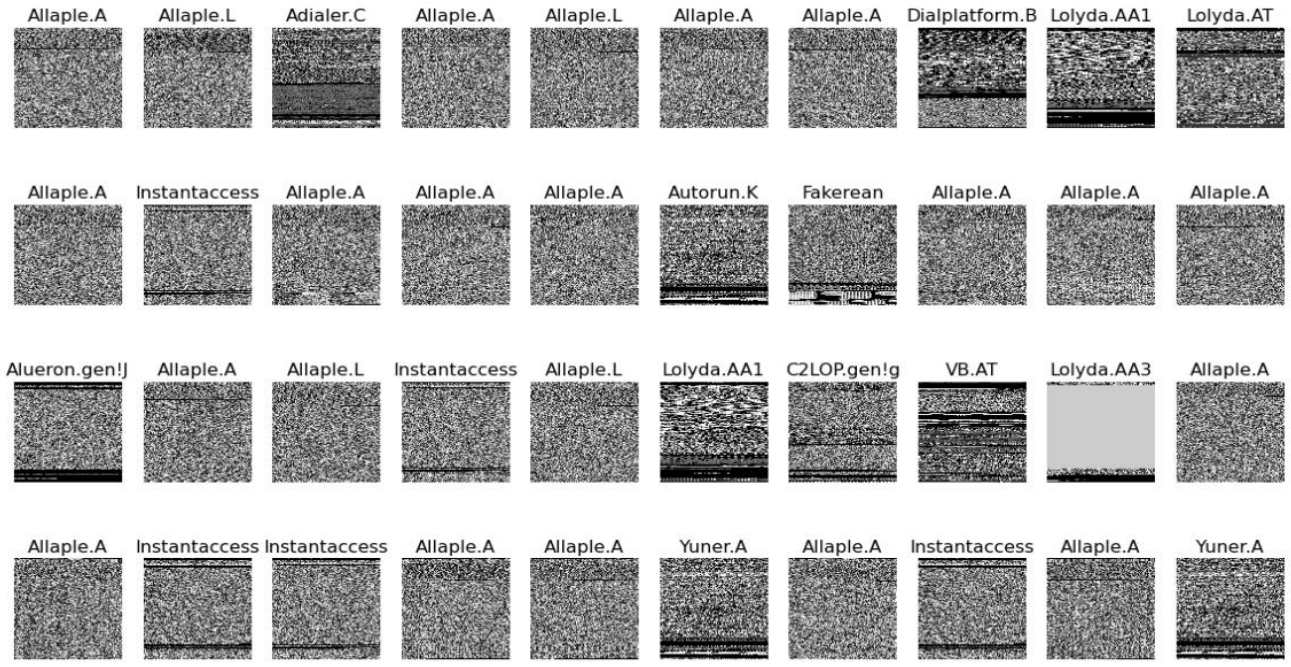
include less training time, feature extraction and it is used in the place of lack of data. Keras provides access to a number of top-performing pre-trained models that were developed for image recognition tasks. Here we have considered Xception architecture.

Xception model which is developed by Francois Chollet is based on the Inception architecture. Here Inception modules have been replaced with depthwise separable convolutions. As it is a stronger version of Inception architecture, it was proposed to be named as “Xception” which stands for “Extreme inception”. The fundamental concept of Xception i.e. depthwise separable convolutions has two parts as depthwise convolutions and pointwise convolutions. Xception architecture can be used through Keras and TensorFlow.

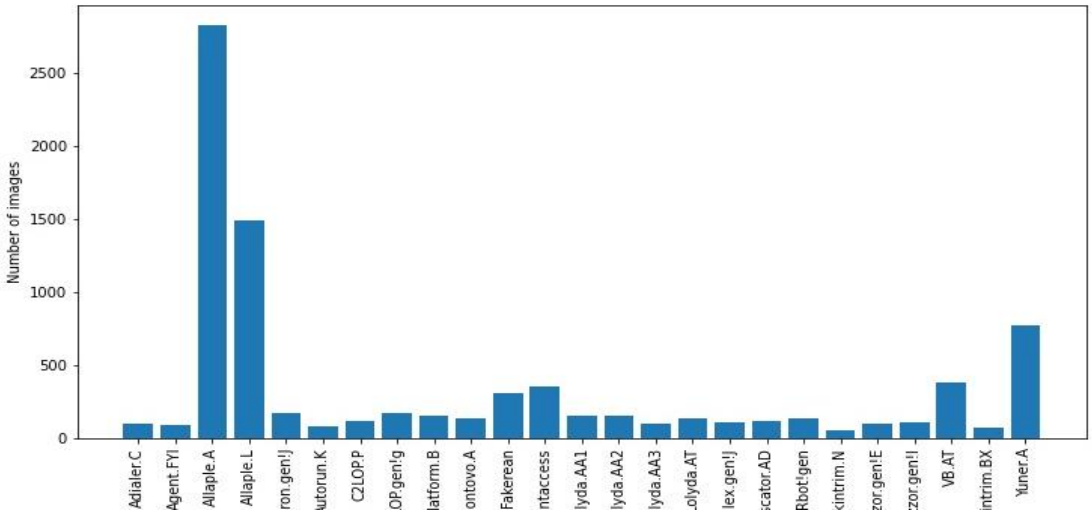
DATA ANALYSIS:

For the project we have taken the “malimg” dataset provided by Kaggle. The pri mary advantage of the malimg dataset is that it had already converted the malwa re files into RGB images which decreases the computational cost & time. It cons ists of 9339 malware images divided into 25 different families of malware. The t raining data contains 8404 images. The 25 families of malware can be described as {'Adialer.C','Agent.FYI','Allaple.A','Allaple.L','Alueron.gen!J','Autorun.K','C 2LOP.P','C2LOP.gen!g','Dialplatform.B','Dontovo.A','Fakerean','Instantaccess',' Lolyda.AA1','Lolyda.AA2','Lolyda.AA3','Lolyda.AT','Malex.gen!J','Obfuscator. AD','Rbot!gen','Skintrim.N','Swizzor.gen!E’,'Swizzor.gen!I','VB.AT','Wintrim. BX','Yuner.A'}

Here are the sample images from our data.



The shape of the images was set to be (128X128X3) for better preprocessing. Th e number of images in each family was represented graphically.



There were a great number of images from two specific classes i.e. “Allaple A” & “Allaple L”. Due to this our data is unbalanced towards the other classes. To resolve this issue we have assigned equal weights to each class through class_weight while building the neural network model.

BUILDING MODEL:

After splitting the data into training and testing sets, we defined a function to build a CNN model using the Xception model. Along with the Xception model, a GlobalAveragePooling layer and for output a Dense layer as also added. For compiling the model, “adam” optimizer along with “categorical_crossentropy” loss function is defined. “Accuracy” was set to be the metrics parameter. Then the model was fitted to the training data set with 15 epochs.

```
def create_model(base_model):
    base_model.trainable = True
    global_average_layer = tf.keras.layers.GlobalAveragePooling2D()(base_model.output)
    prediction_layer = tf.keras.layers.Dense(25, activation='softmax')(global_average_layer)
    model = tf.keras.models.Model(inputs=base_model.input, outputs=prediction_layer)
    model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.0001), loss = "categorical_crossentropy", metrics=["accuracy"])
    return model
```

```
base_model = Xception(input_shape=(128,128,3), include_top=False, weights="imagenet")
model = create_model(base_model)
```

```
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=15,  class_weight=class_weights)
```

(Code snippet of building the model)

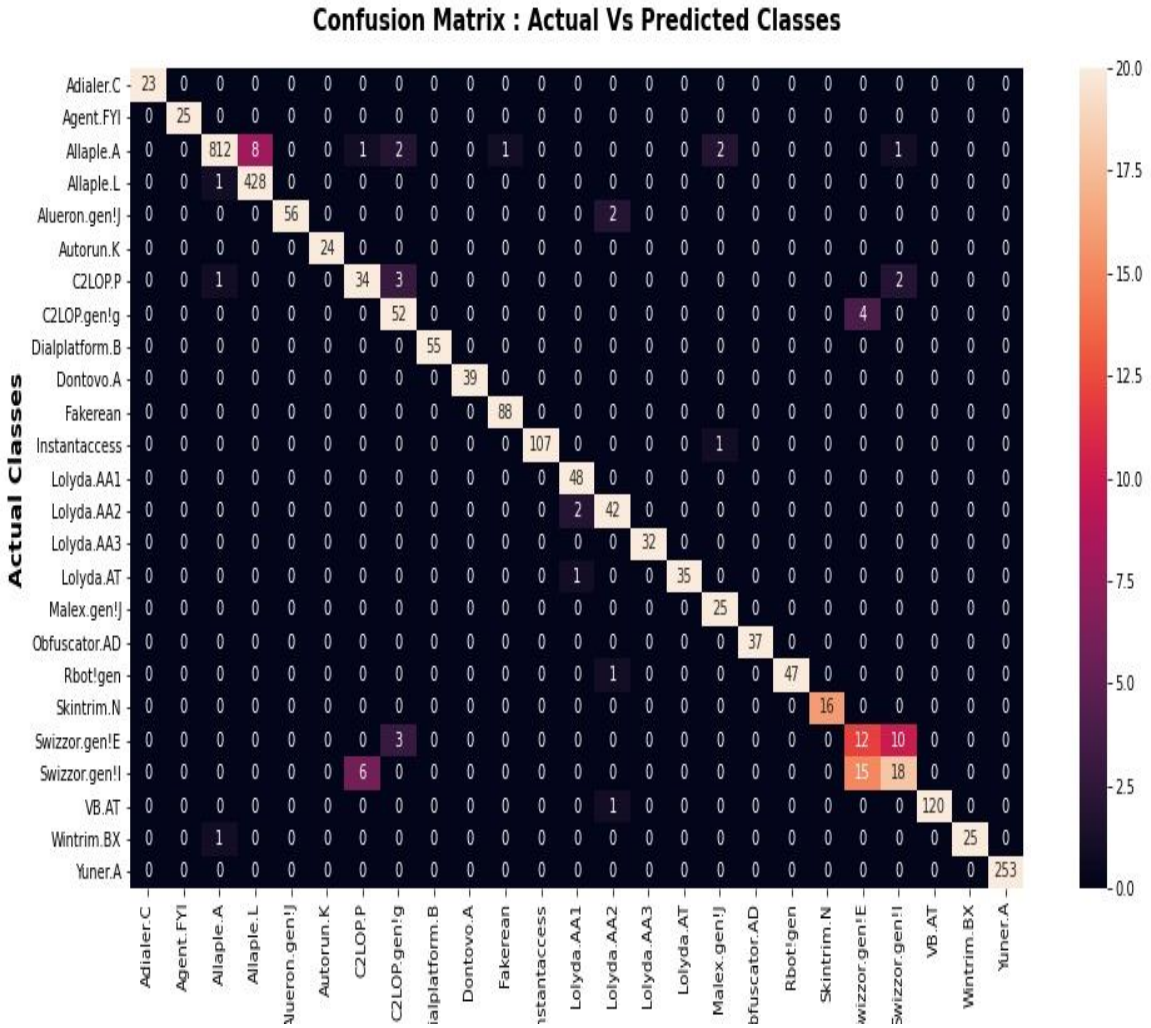
PERFORMANCE ANALYSIS:

The accuracy of the Xception model was found out to be 97.26 % which is a good measure of the architecture.

```
scores = model.evaluate(X_test, y_test)
print("Accuracy of the model: ", scores[1])
```

```
79/79 [==============================] - 173s 2s/step - loss: 0.1299 - accuracy: 0.9726
Accuracy of the model:  0.9726407527923584
```

Also, the confusion matrix was presented graphically for better understanding of the predictions.



From the above observation, we can say that all malwares are classified excellently. But sometimes “Swizzor.gen!E” is misinterepted as “Swizzor.gen!L” and vice-versa. This can be due to the fact that both the malwares come from same families and very similar in their respective codes.

USE CASE:

- The proposed architecture can be used in anti-virus vendors to detect unknown malwares.

FURTHER IMPROVEMENTS:

- An updated dataset “Microsoft Malware Dataset” can be used in place of Malimg dataset for better training of the model.
- Different Transfer learning architecture such as InceptionV3, ResNet50 can be compared & then used for building the model.

For entire code you can check out my Github .