MALWARE ANALYSIS IN DATA SCIENCE

REVIEW – 2

TITLE: MALWARE IMAGE CLASSIFICATION USING DEEP LEARNING MODELS.

Team Members

Aaryan Mehta: 20BAI1108

Ladi Jeevan Sai: 20BAI1293

Sai Nikhil: 20BAI1217

Sarvesh Chandak: 20BAI1221

Course name: Malware analysis in data science

Course code: CSE4053

Faculty: Dr. Jayasudha S

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Abstract

This study aims to explore the effectiveness of various deep learning models in classifying malware images using the Malimg dataset. Malware is a type of malicious software designed to damage, disrupt, or take control of computer systems. The detection and classification of malware are crucial in maintaining the security of computer systems.

In this study, we used the Malimg dataset, which contains over 9,000 malware images belonging to 25 different families. We trained and evaluated four deep learning models, namely VGG16, ResNet50, InceptionV3 using transfer learning techniques.

We also compared the performance of our models with previous studies that used the same dataset and found that our models outperformed the previous studies.

Our study demonstrates the effectiveness of using deep learning models for malware image classification. The results show that transfer learning techniques can be used to achieve high accuracy with relatively small datasets, such as the Malimg dataset. This study provides insights into the potential use of deep learning models for malware detection and classification, which can aid in improving the security of computer systems.

Objective

- To convert a malware into an image and analyze it.
- To develop a deep learning model that can accurately classify malware and non-malware images.
- To compare the performance of the CNN-based approach with other malware detection methods and evaluate its advantages and limitations.
- To improve the detection rate of unknown malware by using visual analysis instead of traditional methods.
- To reduce the number of false positives and false negatives in malware detection.
- To improve the overall efficiency.

Dataset

The Malimg Dataset contains 9339 malware images, belonging to 25 families/classes. Thus, our goal is to perform a multi-class classification of malware.

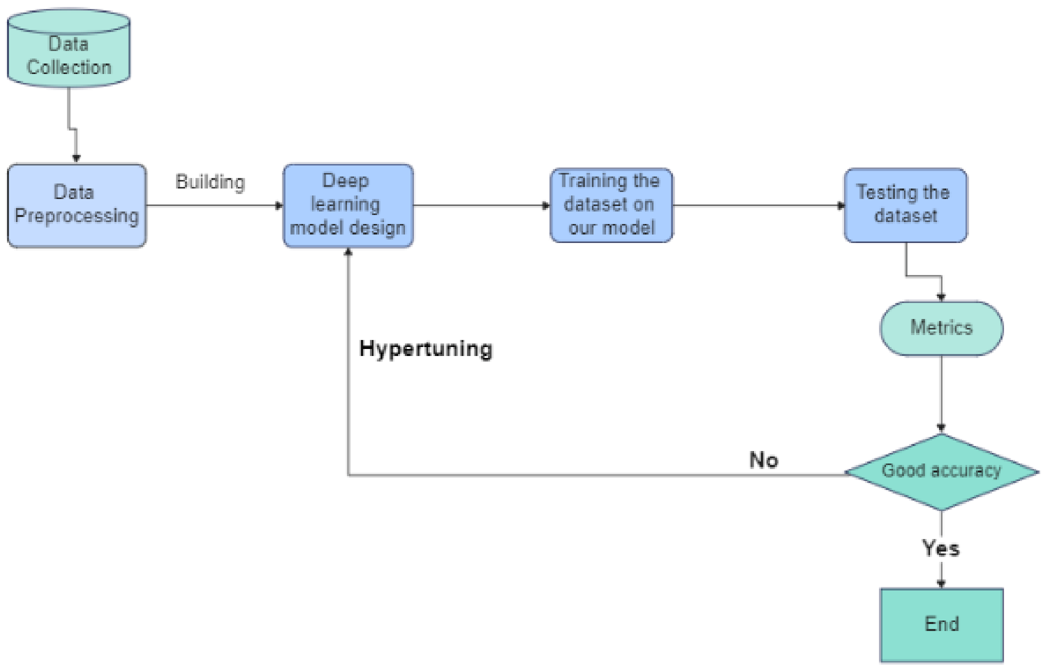
Classes of malware

- Adialer.C
- Agent.FYI
- Allaple.A
- Allaple.L
- Alueron.gen!J
- Autorun.K
- C2LOP.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

Link for the dataset:

https://www.kaggle.com/competitions/malware-classification/data

Architecture Diagram



Modules Planned

- **Data Collection and Preprocessing:**

This module involves collection of dataset of labeled malware and non-malware images, and preprocessing the dataset which we will use for training and testing. This may include tasks such as resizing, augumenting and normalizing the images.

- **Deep learning Model Design:**

In this module, we will use several deep learning algorithms like cnn, vgg, resnet and Inception architectures will be designed and implemented. This may include selecting the appropriate layers, number of filters and kernel size, optimizer, and loss function for the model.

- **Training and Validation:**

Our Deep Learning model will be trained on the preprocessed dataset and fine-tuned to improve its performance. This will be done by splitting the data into training and testing, and training the model on the training set.

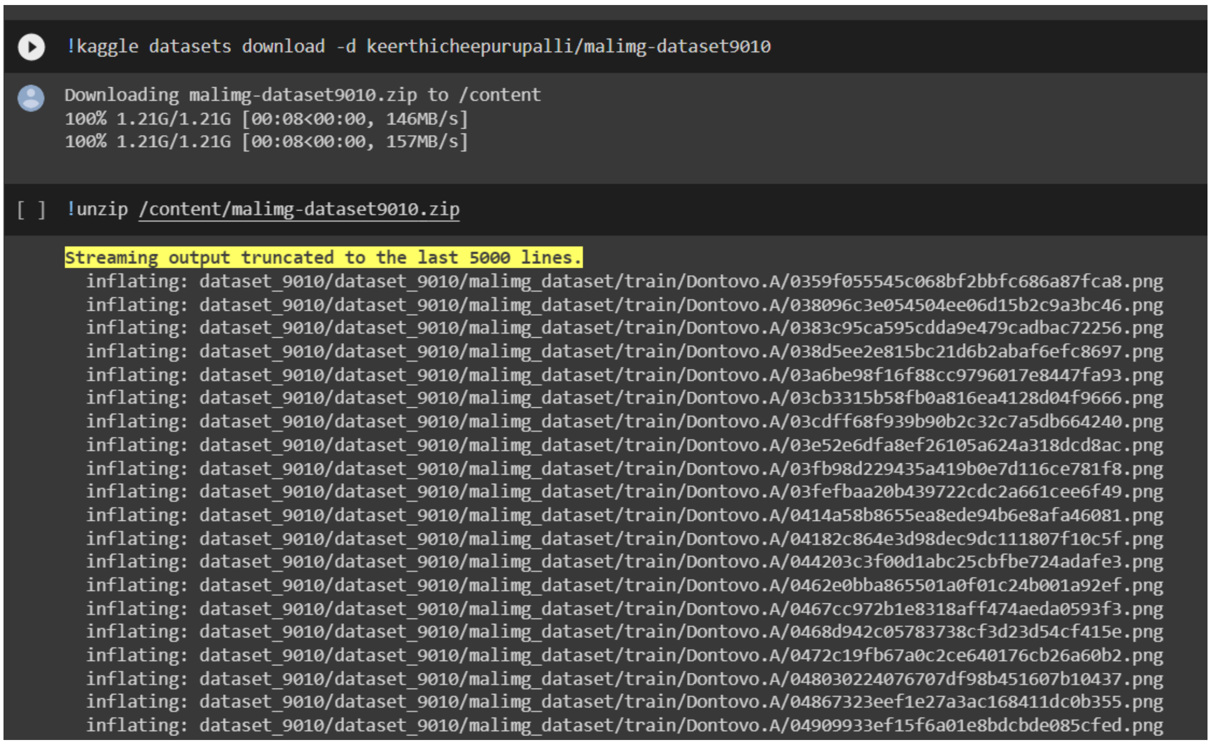
- **Testing and Evaluation:**

The trained models will be evaluated using the validation dataset and the test dataset. This will be done to check the accuracy, precision, recall, and other evaluation metric of the model on unseen data. Ultimately we will select the best model out of all.

Implementation

- **Pre-processing**

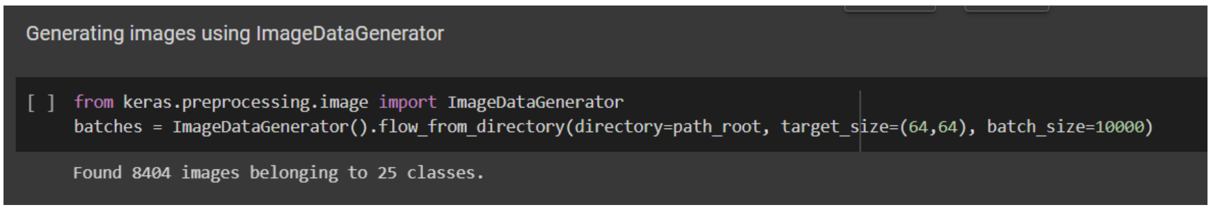
Calling dataset by Kaggle api and unzipping the dataset.



```
!kaggle datasets download -d keerthicheepurupalli/malimg-dataset9010

Downloading malimg-dataset9010.zip to /content
100% 1.21G/1.21G [00:08<00:00, 146MB/s]
100% 1.21G/1.21G [00:08<00:00, 157MB/s]

[ ] !unzip /content/malimg-dataset9010.zip

Streaming output truncated to the last 5000 lines.
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0359f055545c068bf2bbfc686a87fca8.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/038096c3e054504ee06d15b2c9a3bc46.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0383c95ca595cdda9e479cadbac72256.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/038d5ee2e815bc21d6b2abaf6efc8697.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/03a6be98f16f88cc9796017e8447fa93.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/03cb3315b58fb0a816ea4128d04f9666.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/03cdff68f939b90b2c32c7a5db664240.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/03e52e6dfa8ef26105a624a318dcd8ac.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/03fb98d229435a419b0e7d116ce781f8.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/03fefbaa20b439722cdc2a661cee6f49.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0414a58b8655ea8ede94b6e8afa46081.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/04182c864e3d98dec9dc111807f10c5f.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/044203c3f00d1abc25cbfbe724adafe3.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0462e0bba865501a0f01c24b001a92ef.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0467cc972b1e8318aff474aeda0593f3.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0468d942c05783738cf3d23d54cf415e.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/0472c19fb67a0c2ce640176cb26a60b2.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/048030224076707df98b451607b10437.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/04867323eef1e27a3ac168411dc0b355.png
  inflating: dataset_9010/dataset_9010/malimg_dataset/train/Dontovo.A/04909933ef15f6a01e8bdcbde085cfed.png
```

- **Generating images using imagedatagenerator**

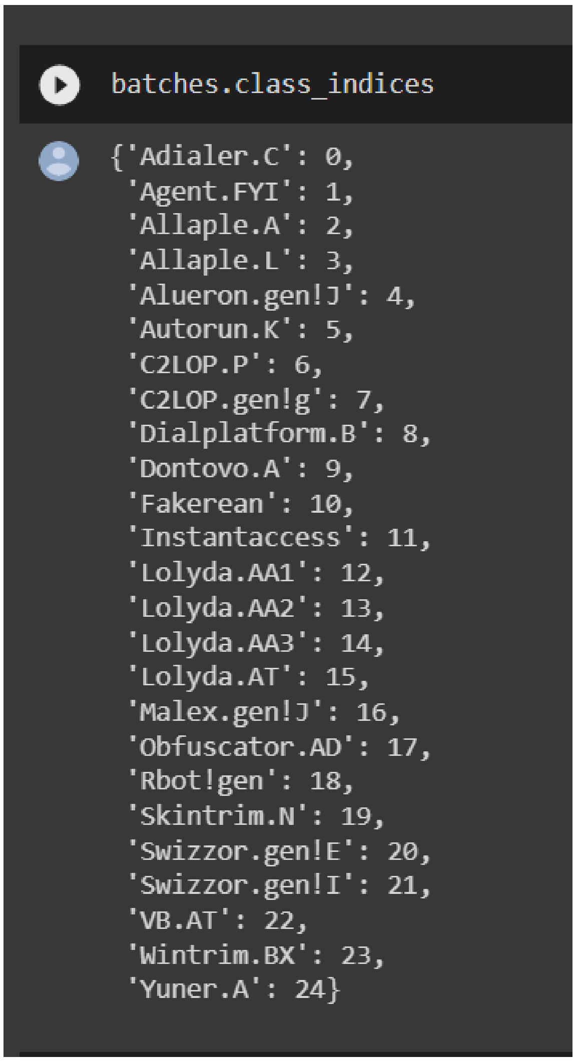
ImageDataGenerator.flow_from_directory() generates batches of normalized tensor image data from the respective data directories.



Generating images using ImageDataGenerator

```
[ ] from keras.preprocessing.image import ImageDataGenerator
    batches = ImageDataGenerator().flow_from_directory(directory=path_root, target_size=(64,64), batch_size=10000)

    Found 8404 images belonging to 25 classes.
```

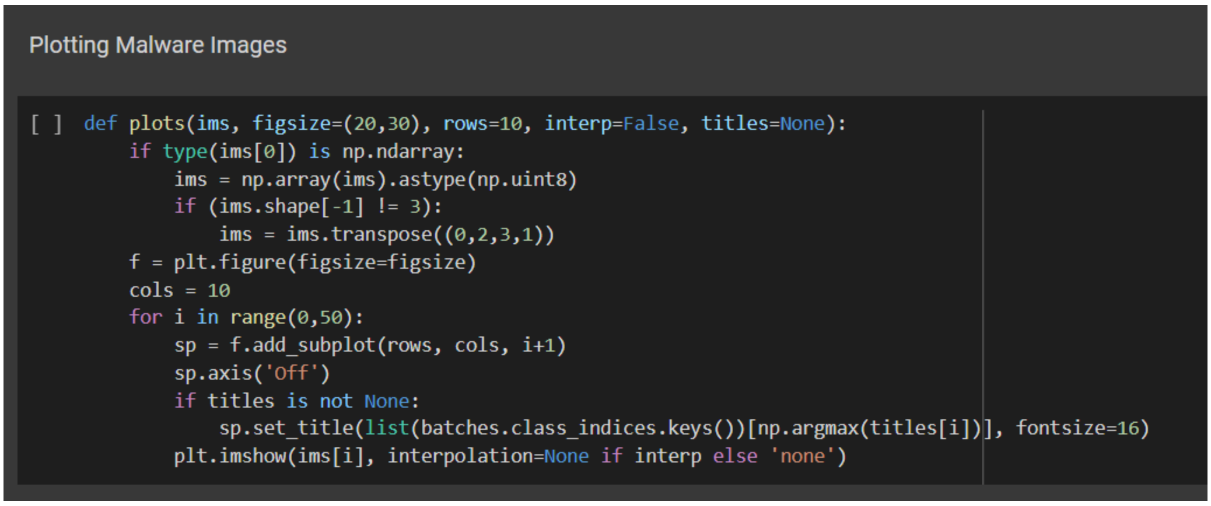
- **Classes of malware**



```
batches.class_indices
```

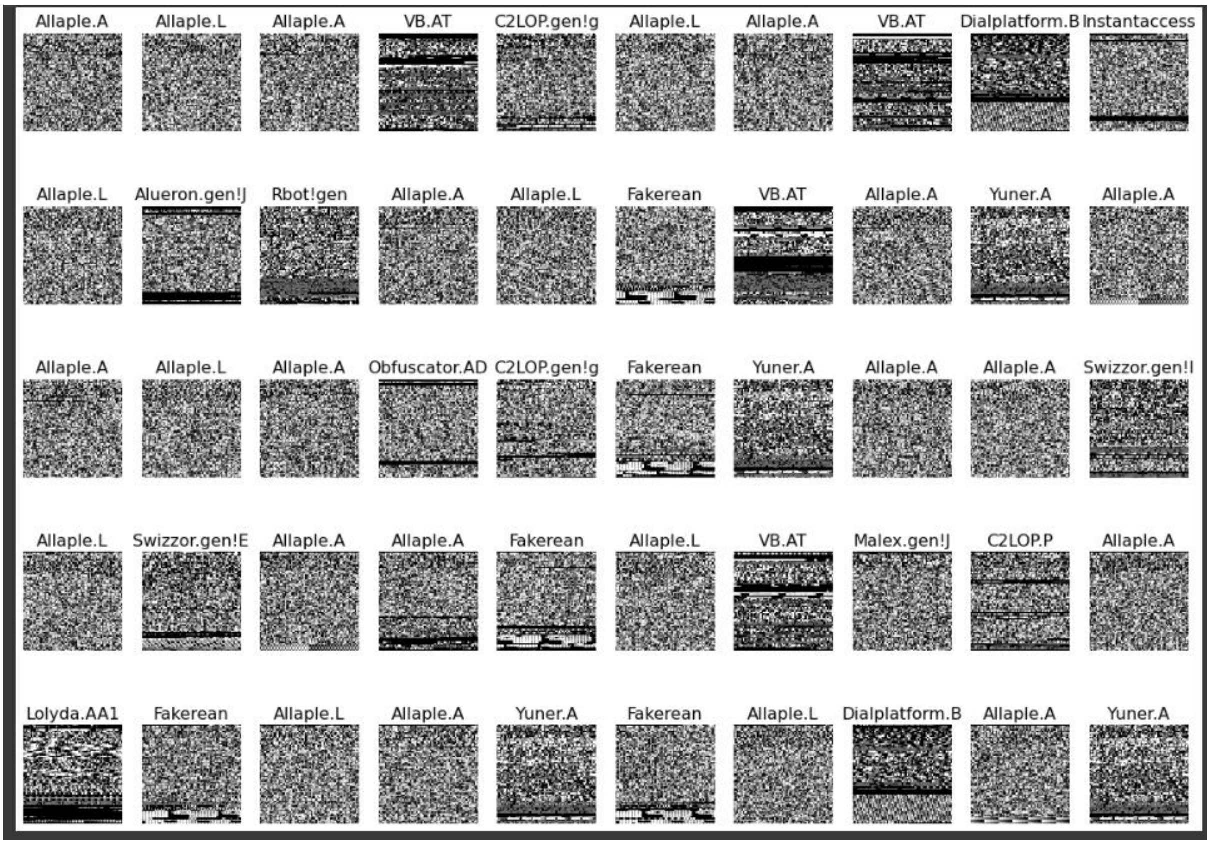
```
{'Adialer.C': 0,
 'Agent.FYI': 1,
 'Allaple.A': 2,
 'Allaple.L': 3,
 'Alueron.gen!J': 4,
 'Autorun.K': 5,
 'C2LOP.P': 6,
 'C2LOP.gen!g': 7,
 'Dialplatform.B': 8,
 'Dontovo.A': 9,
 'Fakerean': 10,
 'Instantaccess': 11,
 'Lolyda.AA1': 12,
 'Lolyda.AA2': 13,
 'Lolyda.AA3': 14,
 'Lolyda.AT': 15,
 'Malex.gen!J': 16,
 'Obfuscator.AD': 17,
 'Rbot!gen': 18,
 'Skintrim.N': 19,
 'Swizzor.gen!E': 20,
 'Swizzor.gen!I': 21,
 'VB.AT': 22,
 'Wintrim.BX': 23,
 'Yuner.A': 24}
```

- **Plotting malware images**



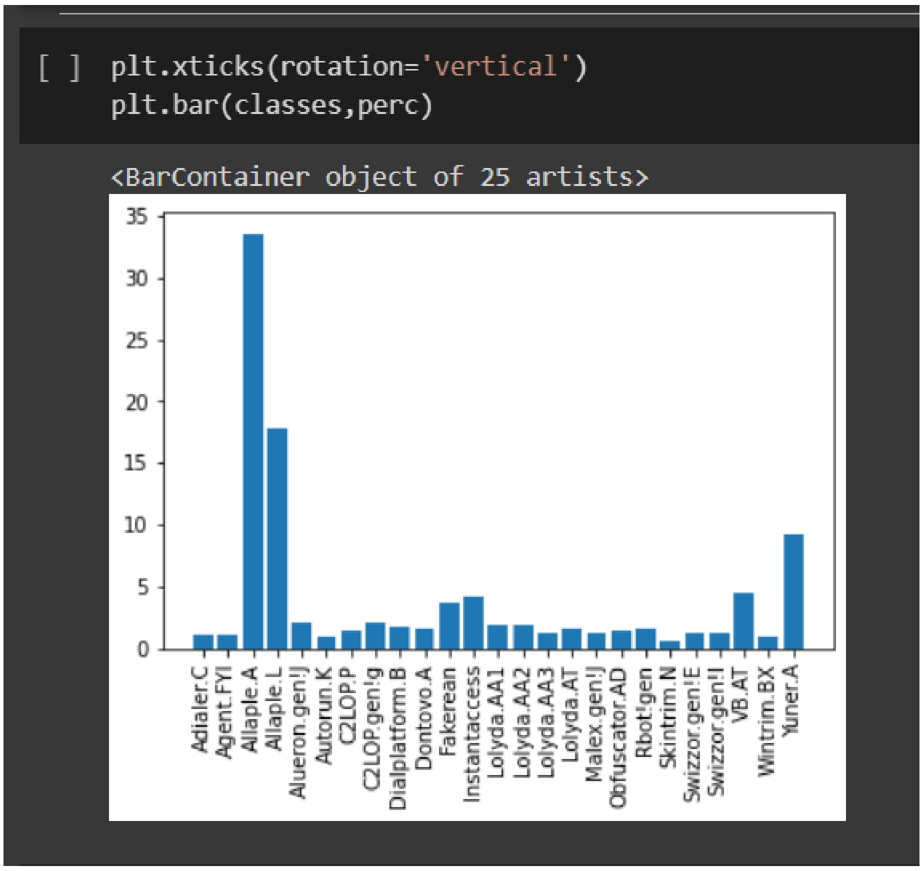
Plotting Malware Images

```
[ ] def plots(ims, figsize=(20,30), rows=10, interp=False, titles=None):
        if type(ims[0]) is np.ndarray:
            ims = np.array(ims).astype(np.uint8)
            if (ims.shape[-1] != 3):
                ims = ims.transpose((0,2,3,1))
        f = plt.figure(figsize=figsize)
        cols = 10
        for i in range(0,50):
            sp = f.add_subplot(rows, cols, i+1)
            sp.axis('Off')
            if titles is not None:
                sp.set_title(list(batches.class_indices.keys())[np.argmax(titles[i])], fontsize=16)
            plt.imshow(ims[i], interpolation=None if interp else 'none')
```



• Graphical visualition

Bar chart

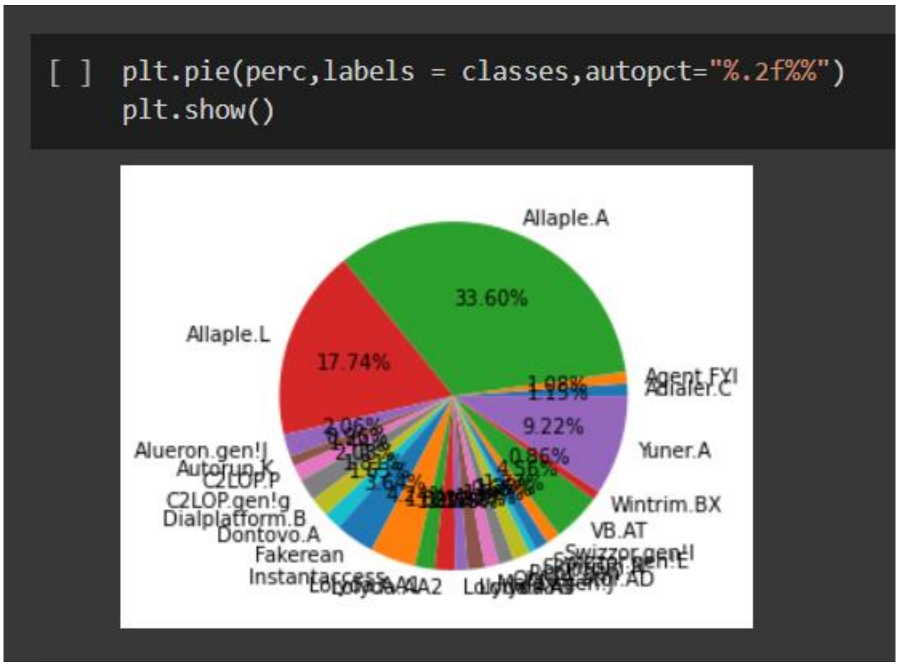


```
[ ] plt.xticks(rotation='vertical')
    plt.bar(classes,perc)
```

```
<BarContainer object of 25 artists>
```

Our dataset is quite unbalanced : a lot of Malwares belong to class 2 : **Allaple.A** and class 3 : **Allaple.L** !

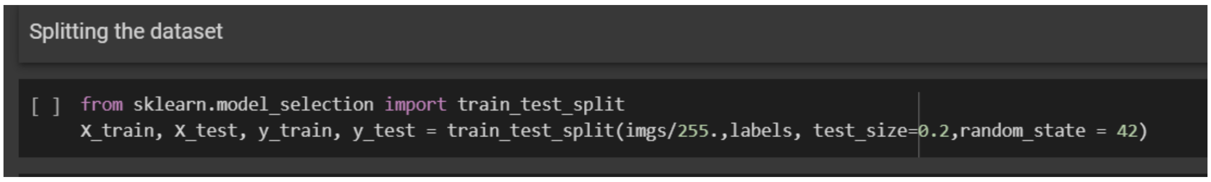
Pie chart



```
[ ]  plt.pie(perc,labels = classes,autopct="%.2f%%")
     plt.show()
```

- **Splitting the dataset**

Splitting the dataset in the ratio of 80 to 20



Splitting the dataset

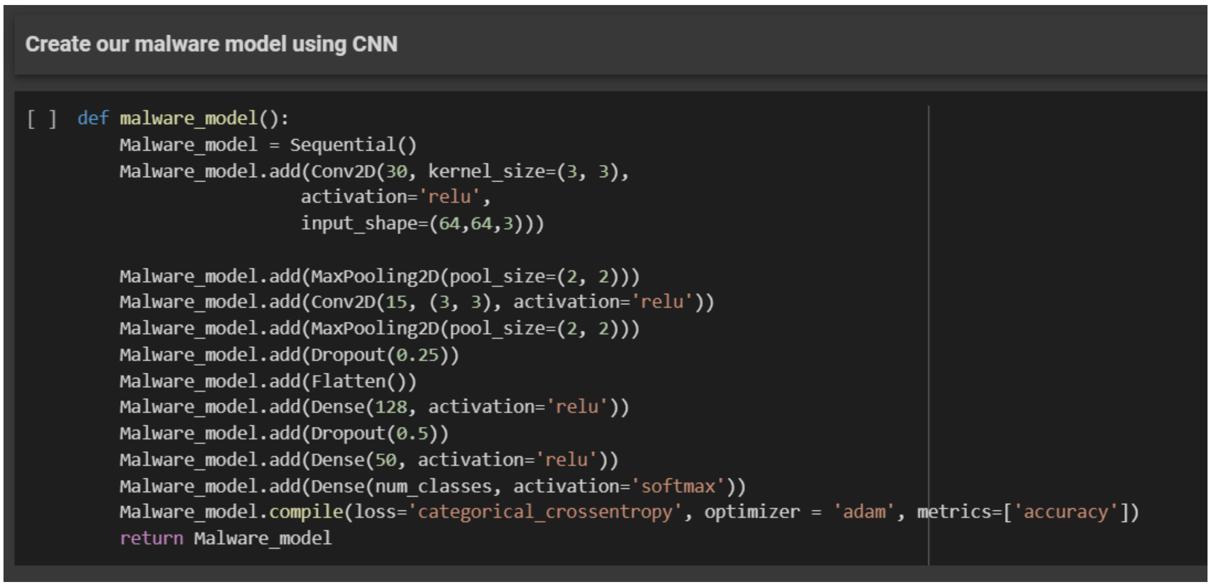
```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(imgs/255.,labels, test_size=0.2,random_state = 42)
```

<u>Creating our model using CNN</u>

Description of our CNN model:

- **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

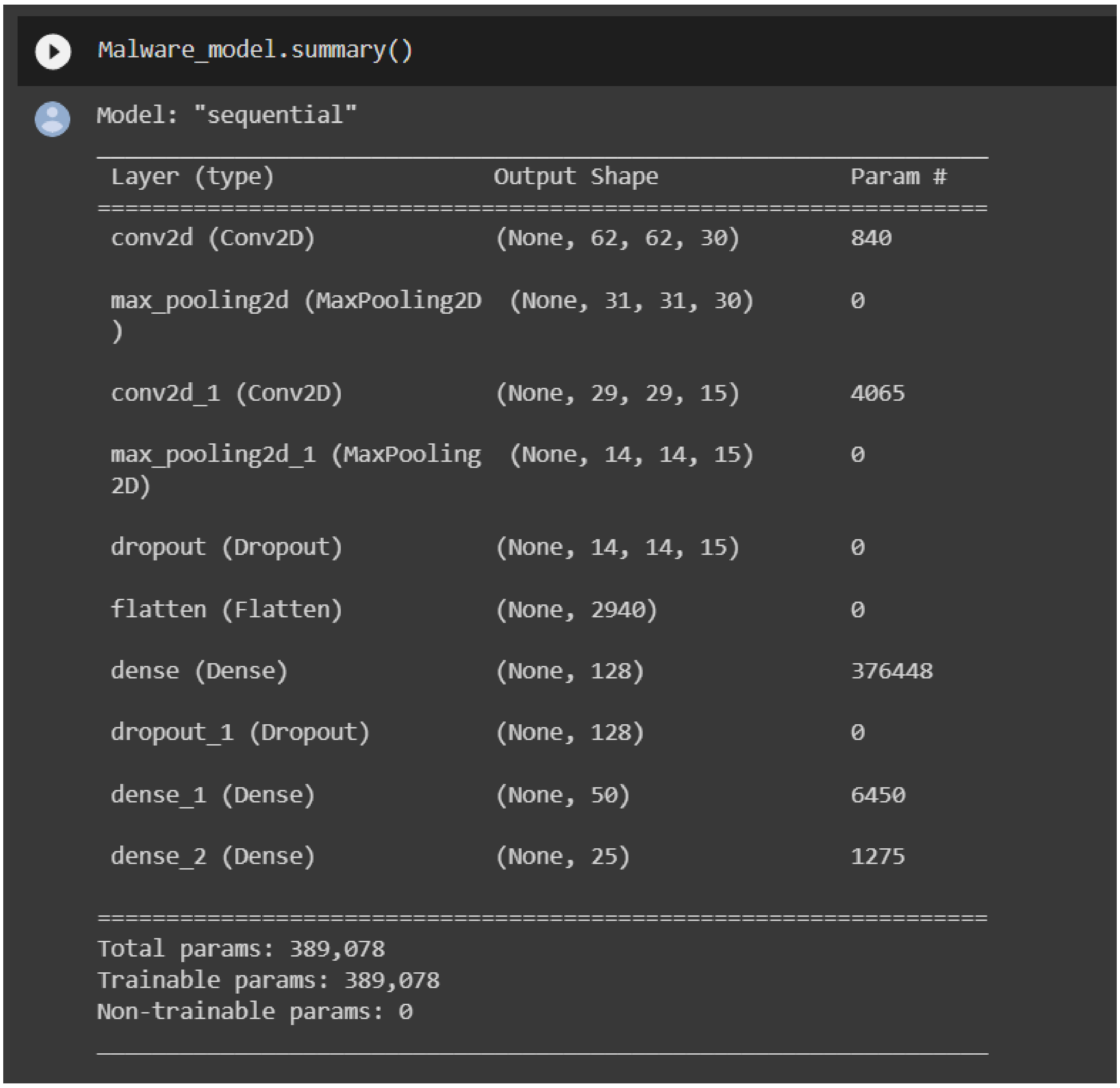
Input shape : 64 * 64 * 3



Create our malware model using CNN

```
[ ] def malware_model():
        Malware_model = Sequential()
        Malware_model.add(Conv2D(30, kernel_size=(3, 3),
                         activation='relu',
                         input_shape=(64,64,3)))

        Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
        Malware_model.add(Conv2D(15, (3, 3), activation='relu'))
        Malware_model.add(MaxPooling2D(pool_size=(2, 2)))
        Malware_model.add(Dropout(0.25))
        Malware_model.add(Flatten())
        Malware_model.add(Dense(128, activation='relu'))
        Malware_model.add(Dropout(0.5))
        Malware_model.add(Dense(50, activation='relu'))
        Malware_model.add(Dense(num_classes, activation='softmax'))
        Malware_model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics=['accuracy'])
        return Malware_model
```

- **Summary of our model**



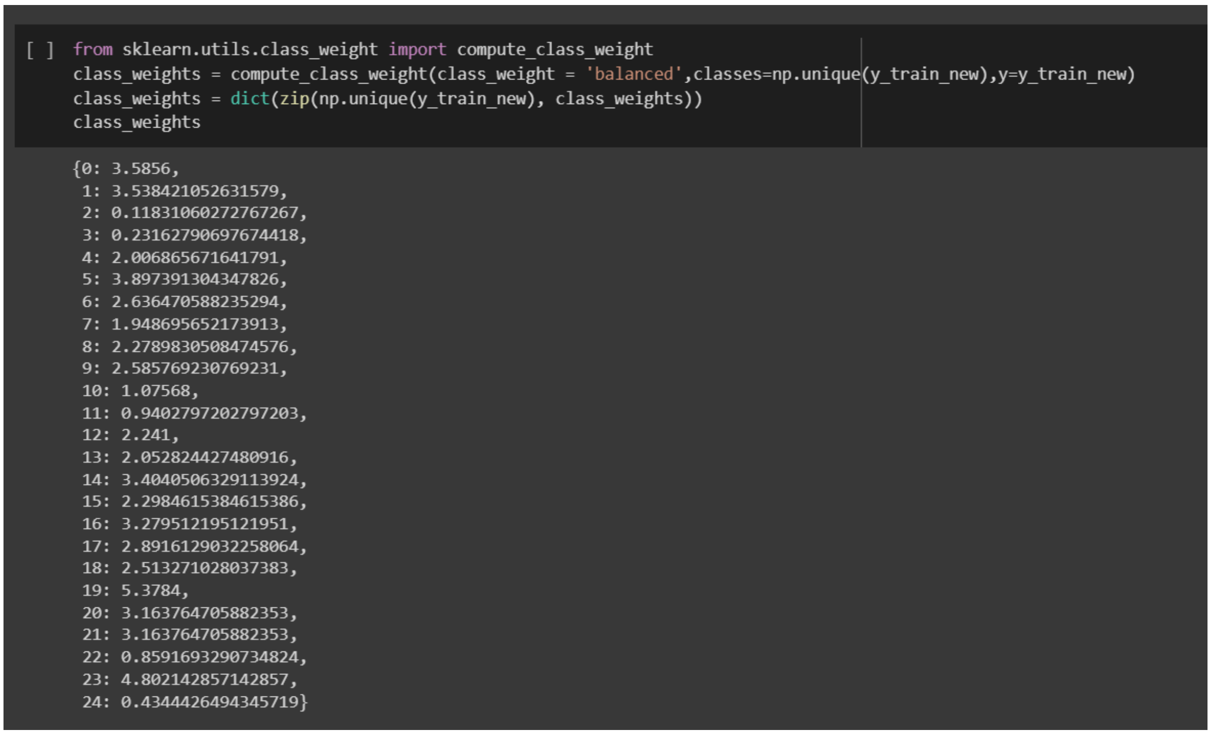
```
Malware_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 30)        840       
                                                                 
 max_pooling2d (MaxPooling2D  (None, 31, 31, 30)       0         
 )                                                               
                                                                 
 conv2d_1 (Conv2D)           (None, 29, 29, 15)        4065      
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 14, 14, 15)       0         
 2D)                                                             
                                                                 
 dropout (Dropout)           (None, 14, 14, 15)        0         
                                                                 
 flatten (Flatten)           (None, 2940)              0         
                                                                 
 dense (Dense)               (None, 128)               376448    
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 50)                6450      
                                                                 
 dense_2 (Dense)             (None, 25)                1275      
                                                                 
=================================================================
Total params: 389,078
Trainable params: 389,078
Non-trainable params: 0
_________________________________________________________________
```

- **Weight for each malware class**

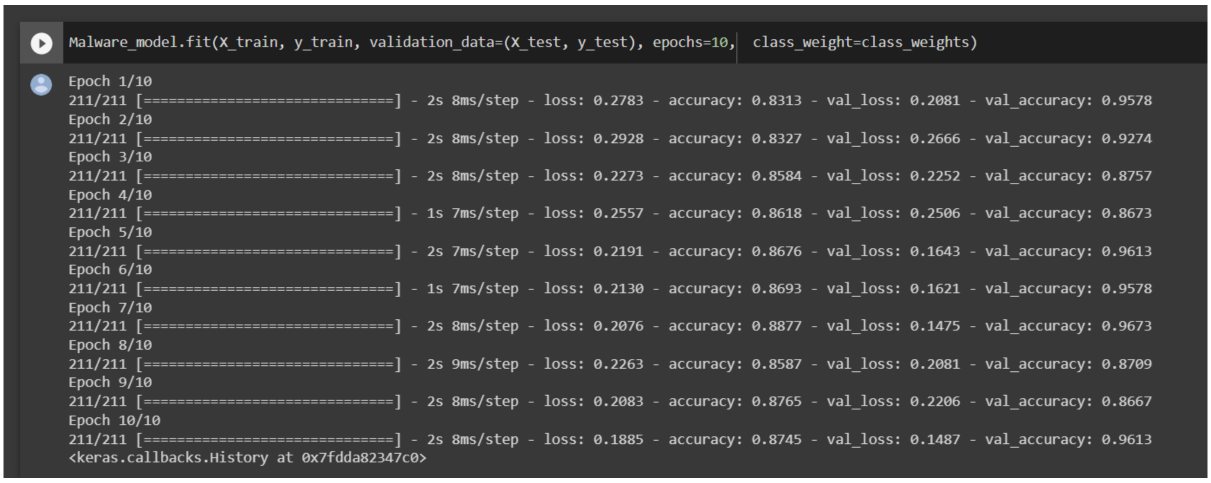
Several methods are available to deal with unbalanced data. In our case, we gave higher weight to minority class and lower weight to majority class.

class_weights uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data. To use this method, y_train must not be one hot encoded.



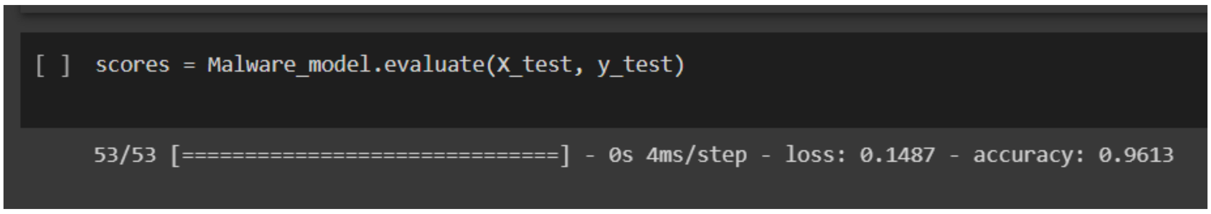
```
[ ] from sklearn.utils.class_weight import compute_class_weight
    class_weights = compute_class_weight(class_weight = 'balanced',classes=np.unique(y_train_new),y=y_train_new)
    class_weights = dict(zip(np.unique(y_train_new), class_weights))
    class_weights

    {0: 3.5856,
     1: 3.538421052631579,
     2: 0.11831060272767267,
     3: 0.23162790697674418,
     4: 2.006865671641791,
     5: 3.897391304347826,
     6: 2.636470588235294,
     7: 1.948695652173913,
     8: 2.2789830508474576,
     9: 2.585769230769231,
     10: 1.07568,
     11: 0.9402797202797203,
     12: 2.241,
     13: 2.052824427480916,
     14: 3.4040506329113924,
     15: 2.2984615384615386,
     16: 3.279512195121951,
     17: 2.8916129032258064,
     18: 2.513271028037383,
     19: 5.3784,
     20: 3.163764705882353,
     21: 3.163764705882353,
     22: 0.8591693290734824,
     23: 4.802142857142857,
     24: 0.4344426494345719}
```

- **Fitting our model**



```
Malware_model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=10,  class_weight=class_weights)

Epoch 1/10
211/211 [==============================] - 2s 8ms/step - loss: 0.2783 - accuracy: 0.8313 - val_loss: 0.2081 - val_accuracy: 0.9578
Epoch 2/10
211/211 [==============================] - 2s 8ms/step - loss: 0.2928 - accuracy: 0.8327 - val_loss: 0.2666 - val_accuracy: 0.9274
Epoch 3/10
211/211 [==============================] - 2s 8ms/step - loss: 0.2273 - accuracy: 0.8584 - val_loss: 0.2252 - val_accuracy: 0.8757
Epoch 4/10
211/211 [==============================] - 1s 7ms/step - loss: 0.2557 - accuracy: 0.8618 - val_loss: 0.2506 - val_accuracy: 0.8673
Epoch 5/10
211/211 [==============================] - 2s 7ms/step - loss: 0.2191 - accuracy: 0.8676 - val_loss: 0.1643 - val_accuracy: 0.9613
Epoch 6/10
211/211 [==============================] - 1s 7ms/step - loss: 0.2130 - accuracy: 0.8693 - val_loss: 0.1621 - val_accuracy: 0.9578
Epoch 7/10
211/211 [==============================] - 2s 8ms/step - loss: 0.2076 - accuracy: 0.8877 - val_loss: 0.1475 - val_accuracy: 0.9673
Epoch 8/10
211/211 [==============================] - 2s 9ms/step - loss: 0.2263 - accuracy: 0.8587 - val_loss: 0.2081 - val_accuracy: 0.8709
Epoch 9/10
211/211 [==============================] - 2s 8ms/step - loss: 0.2083 - accuracy: 0.8765 - val_loss: 0.2206 - val_accuracy: 0.8667
Epoch 10/10
211/211 [==============================] - 2s 8ms/step - loss: 0.1885 - accuracy: 0.8745 - val_loss: 0.1487 - val_accuracy: 0.9613
<keras.callbacks.History at 0x7fdda82347c0>
```

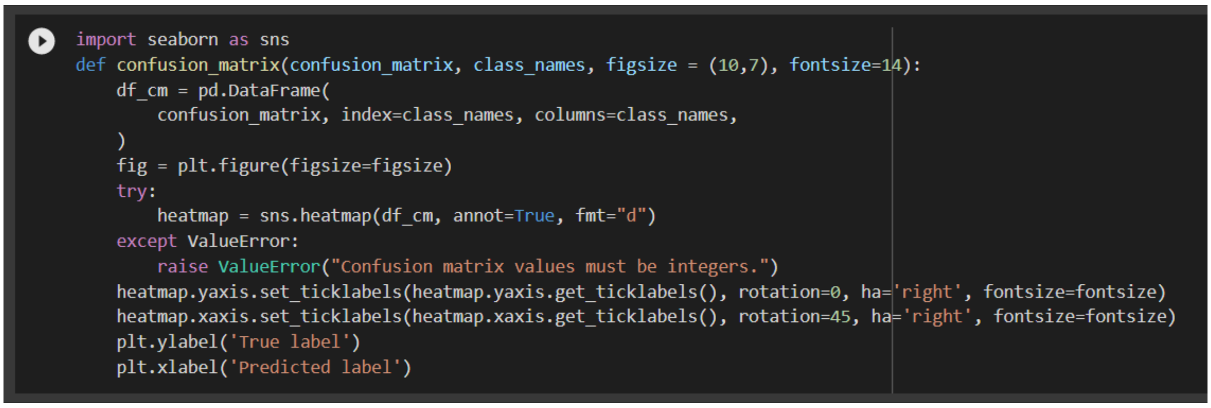
- **Accuracy of our model**



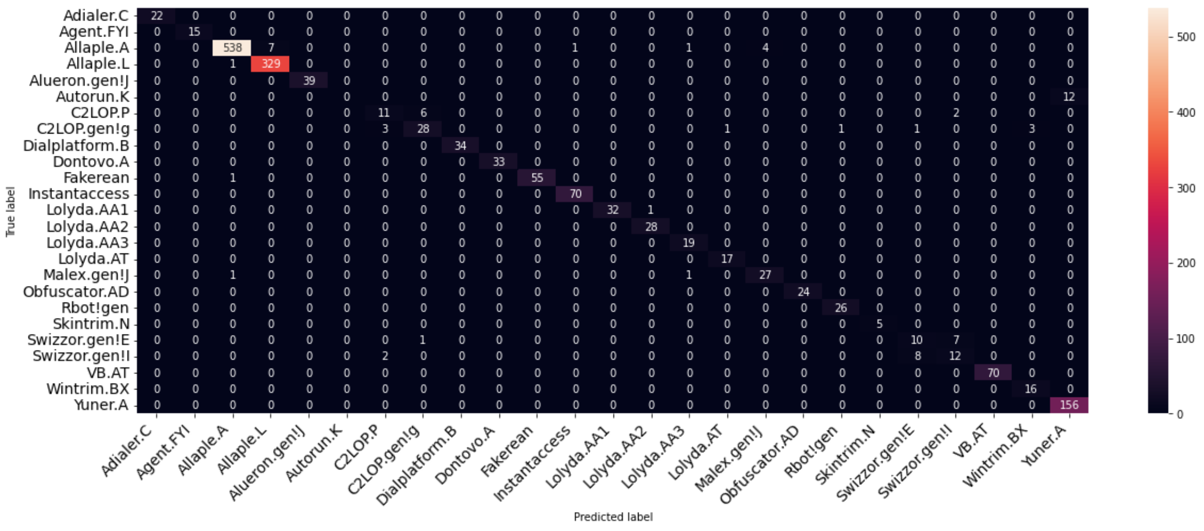
```
[ ]  scores = Malware_model.evaluate(X_test, y_test)

     53/53 [==============================] - 0s 4ms/step - loss: 0.1487 - accuracy: 0.9613
```

Accuracy: 96.13%

- **Confusion matrix**

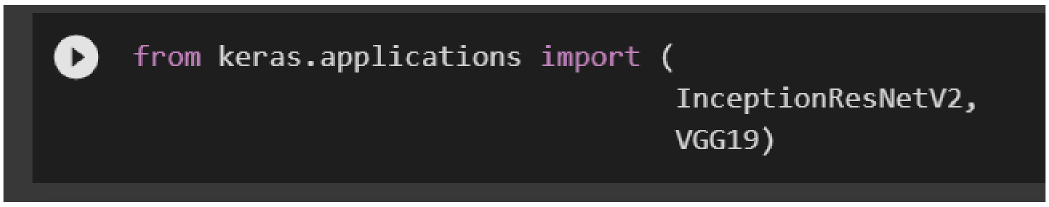


```
import seaborn as sns
def confusion_matrix(confusion_matrix, class_names, figsize = (10,7), fontsize=14):
    df_cm = pd.DataFrame(
        confusion_matrix, index=class_names, columns=class_names,
    )
    fig = plt.figure(figsize=figsize)
    try:
        heatmap = sns.heatmap(df_cm, annot=True, fmt="d")
    except ValueError:
        raise ValueError("Confusion matrix values must be integers.")
    heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', fontsize=fontsize)
    heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', fontsize=fontsize)
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
```



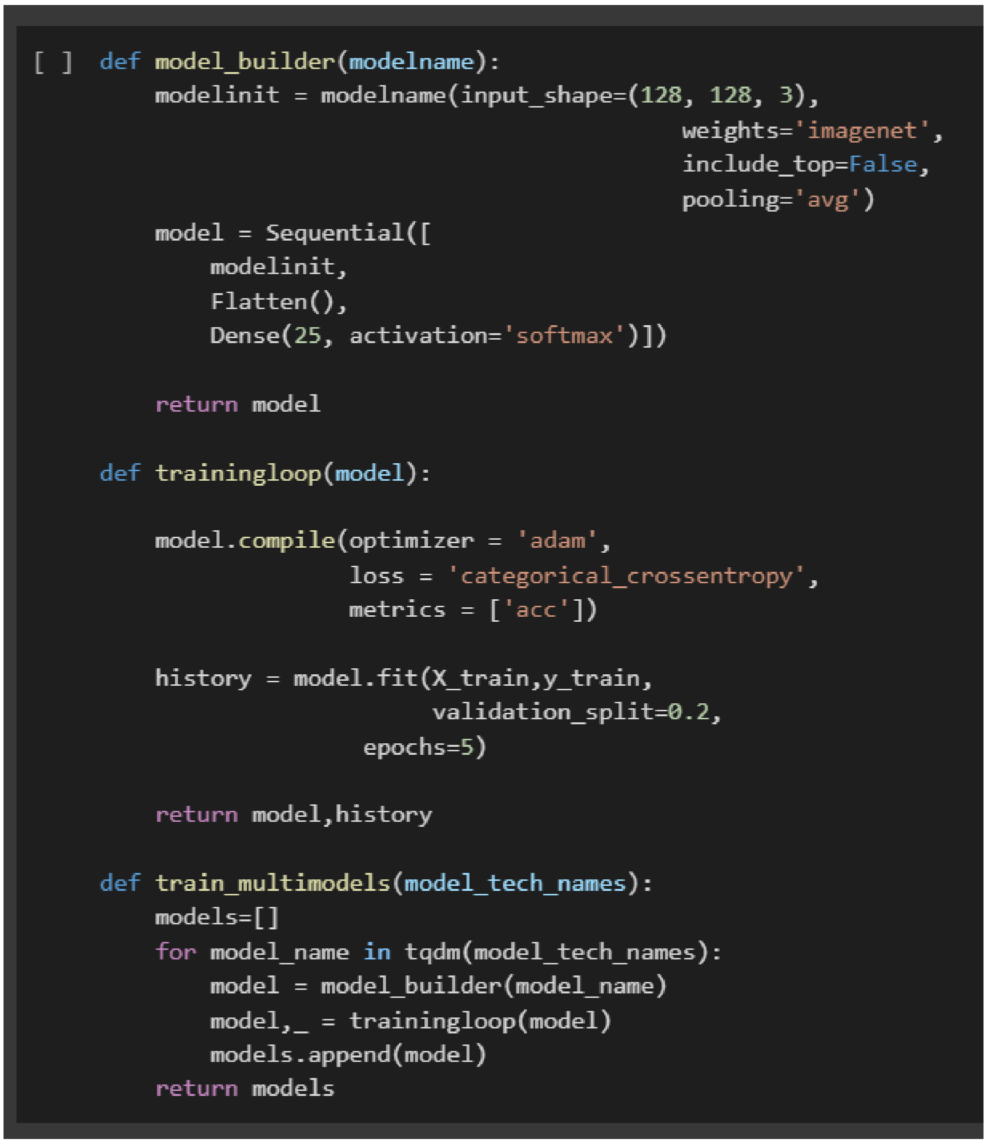
We can observe that although most of the malwares were well classified, **Autorun.K** is always mistaken for **Yuner.A**. This is because we have very few samples of **Autorun.K** in our training set. Moreover, **Swizzor.gen!E** is often mistaken with **Swizzor.gen!I**, because they come from really close famillies and thus could have similarities in their code

Using transfer learning



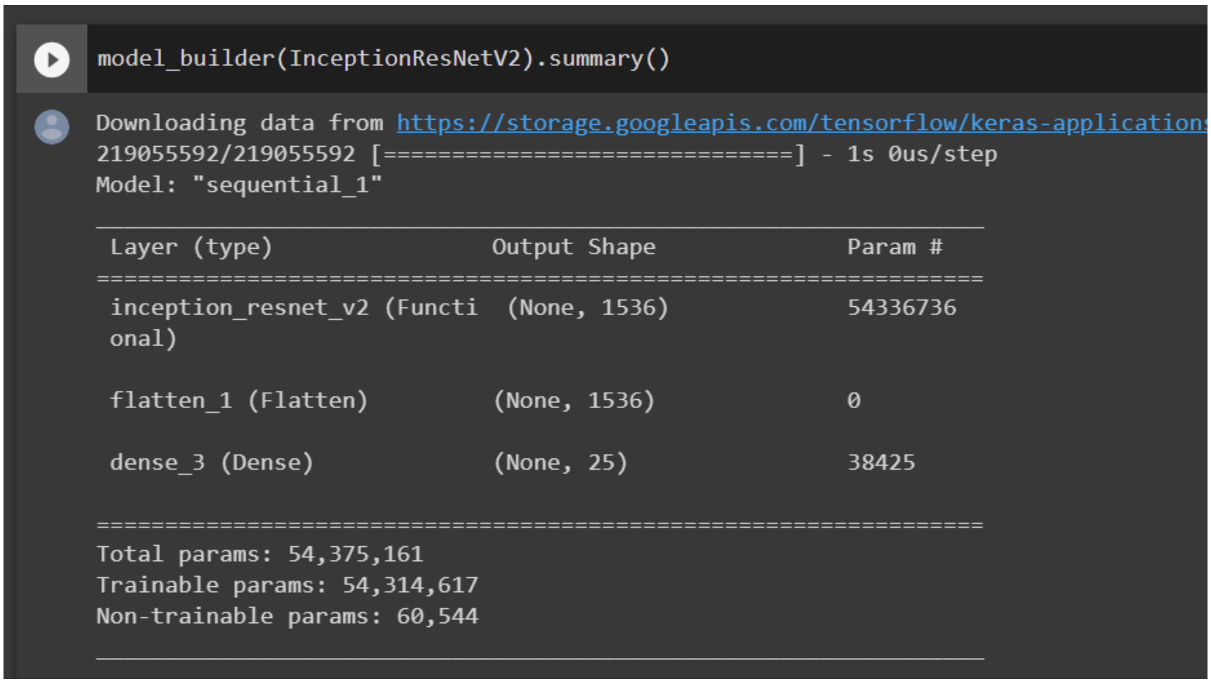
```
from keras.applications import (
                                 InceptionResNetV2,
                                 VGG19)
```

- **Building our model for transfer learning**



```
def model_builder(modelname):
    modelinit = modelname(input_shape=(128, 128, 3),
                                          weights='imagenet',
                                          include_top=False,
                                          pooling='avg')
    model = Sequential([
        modelinit,
        Flatten(),
        Dense(25, activation='softmax')])

    return model

def trainingloop(model):

    model.compile(optimizer = 'adam',
                  loss = 'categorical_crossentropy',
                  metrics = ['acc'])

    history = model.fit(X_train,y_train,
                        validation_split=0.2,
                   epochs=5)

    return model,history

def train_multimodels(model_tech_names):
    models=[]
    for model_name in tqdm(model_tech_names):
        model = model_builder(model_name)
        model,_ = trainingloop(model)
        models.append(model)
    return models
```

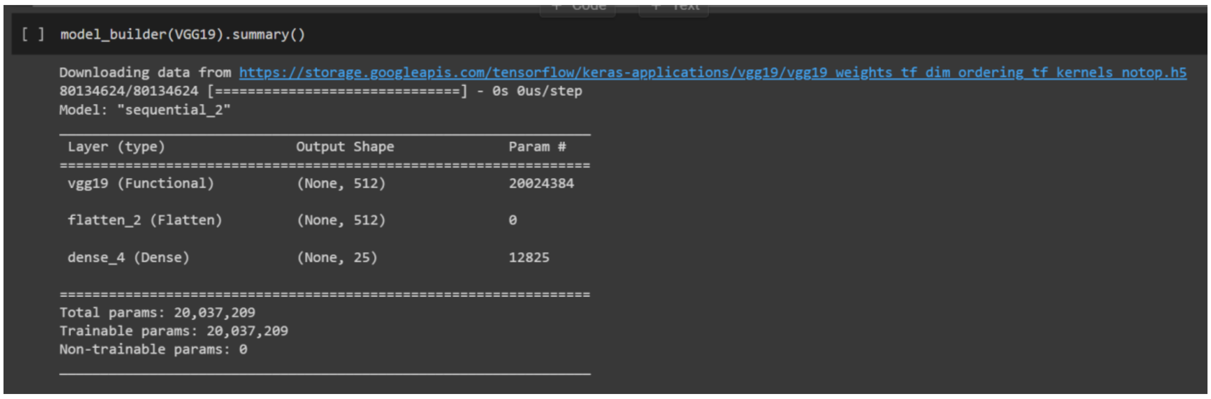
- **InceptionResNetV2 Summary**



```
model_builder(InceptionResNetV2).summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-application
219055592/219055592 [==============================] - 1s 0us/step
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_resnet_v2 (Functi  (None, 1536)             54336736
 onal)

 flatten_1 (Flatten)         (None, 1536)              0

 dense_3 (Dense)             (None, 25)                38425

=================================================================
Total params: 54,375,161
Trainable params: 54,314,617
Non-trainable params: 60,544
_________________________________________________________________
```

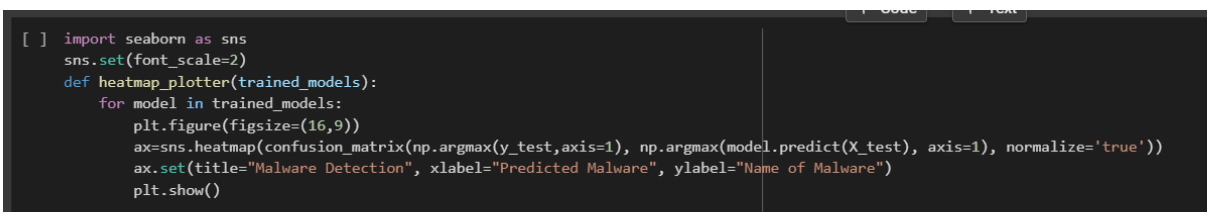
- **VGG19 Summary**



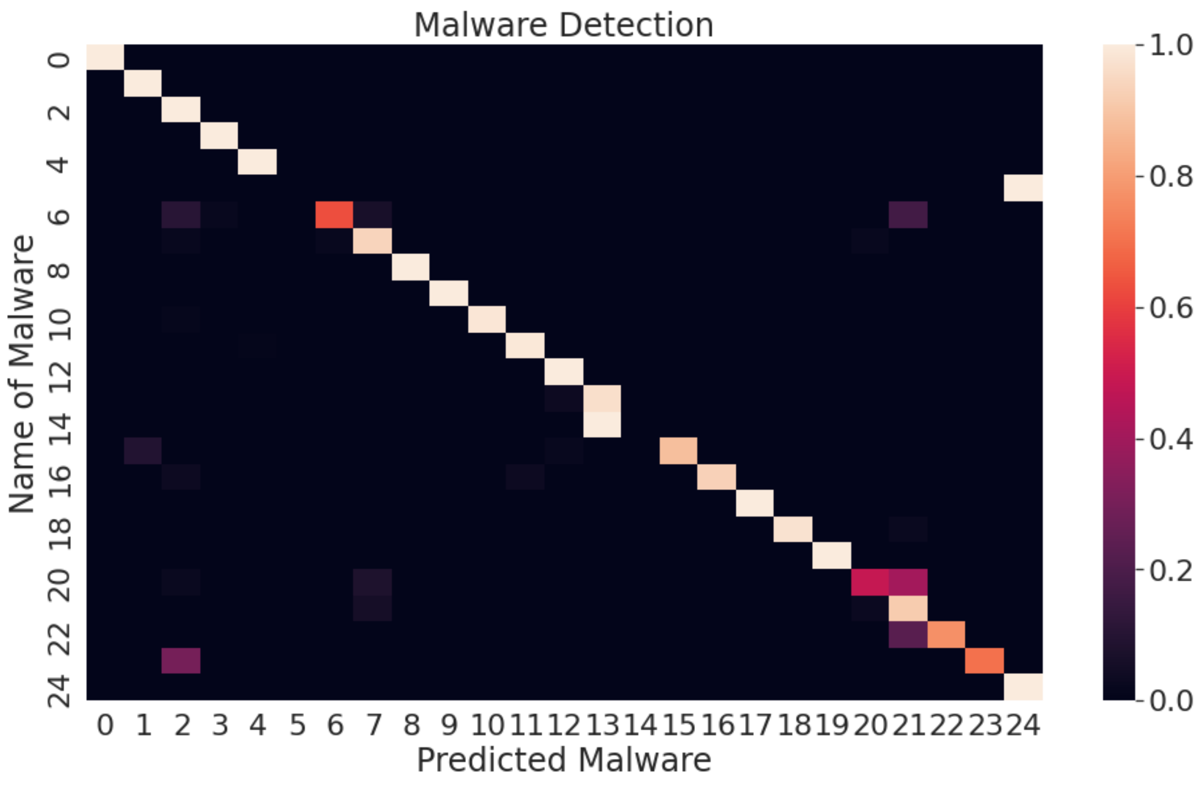
```
[ ] model_builder(VGG19).summary()
```

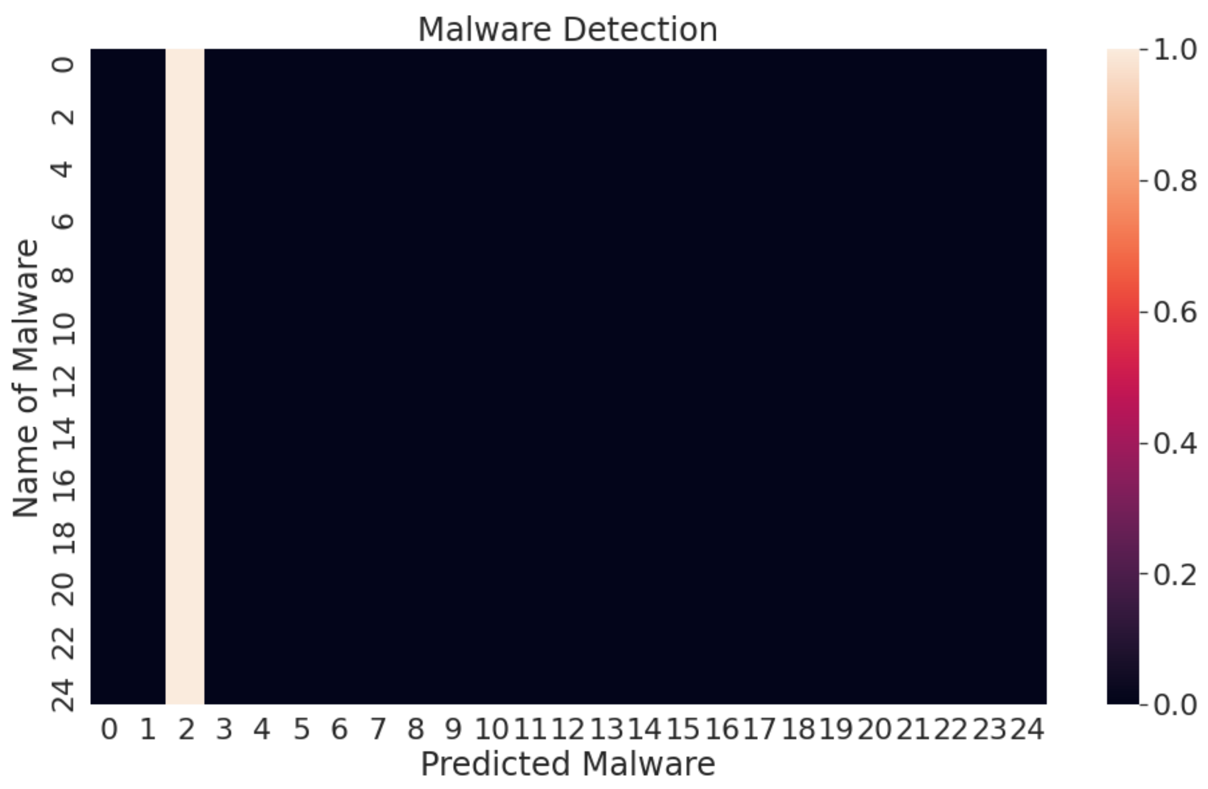
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 512)               20024384

 flatten_2 (Flatten)         (None, 512)               0

 dense_4 (Dense)             (None, 25)                12825

=================================================================
Total params: 20,037,209
Trainable params: 20,037,209
Non-trainable params: 0
_________________________________________________________________
```

- **Heatmap**

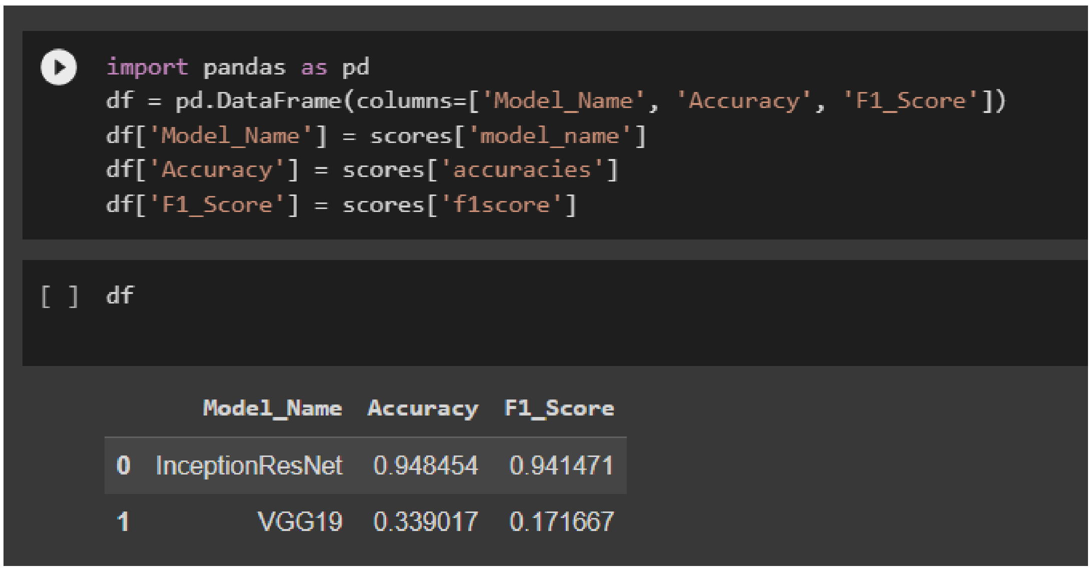


```
[ ]  import seaborn as sns
     sns.set(font_scale=2)
     def heatmap_plotter(trained_models):
         for model in trained_models:
             plt.figure(figsize=(16,9))
             ax=sns.heatmap(confusion_matrix(np.argmax(y_test,axis=1), np.argmax(model.predict(X_test), axis=1), normalize='true'))
             ax.set(title="Malware Detection", xlabel="Predicted Malware", ylabel="Name of Malware")
             plt.show()
```





- **Metrics**

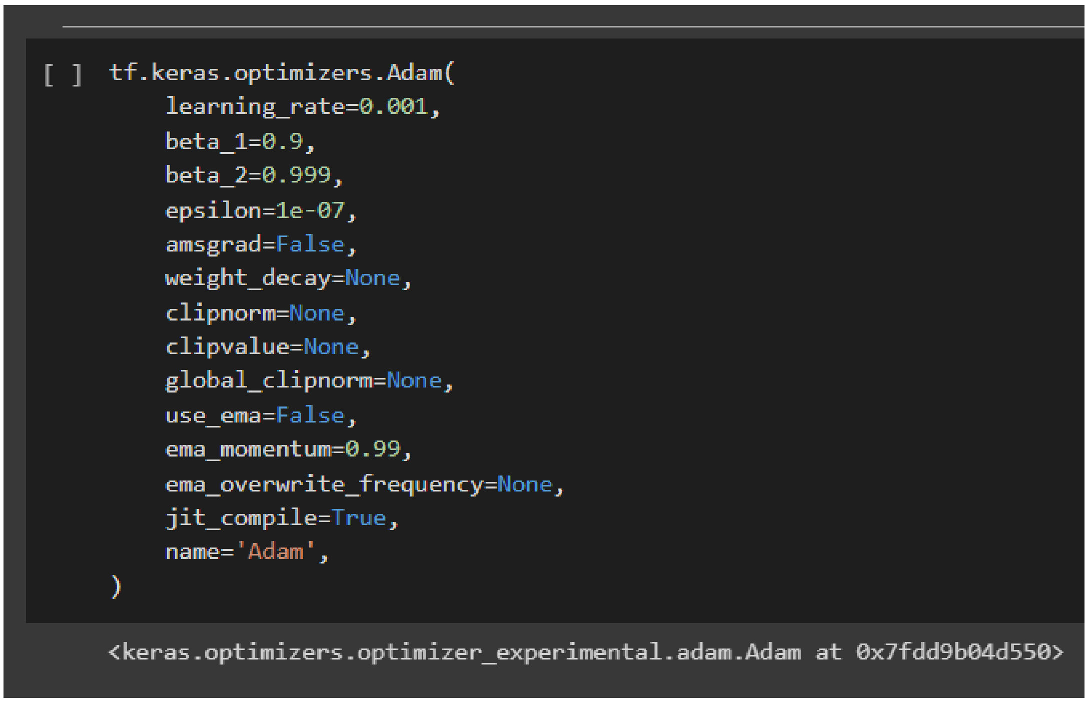


```
import pandas as pd
df = pd.DataFrame(columns=['Model_Name', 'Accuracy', 'F1_Score'])
df['Model_Name'] = scores['model_name']
df['Accuracy'] = scores['accuracies']
df['F1_Score'] = scores['f1score']
```

```
[ ] df
```

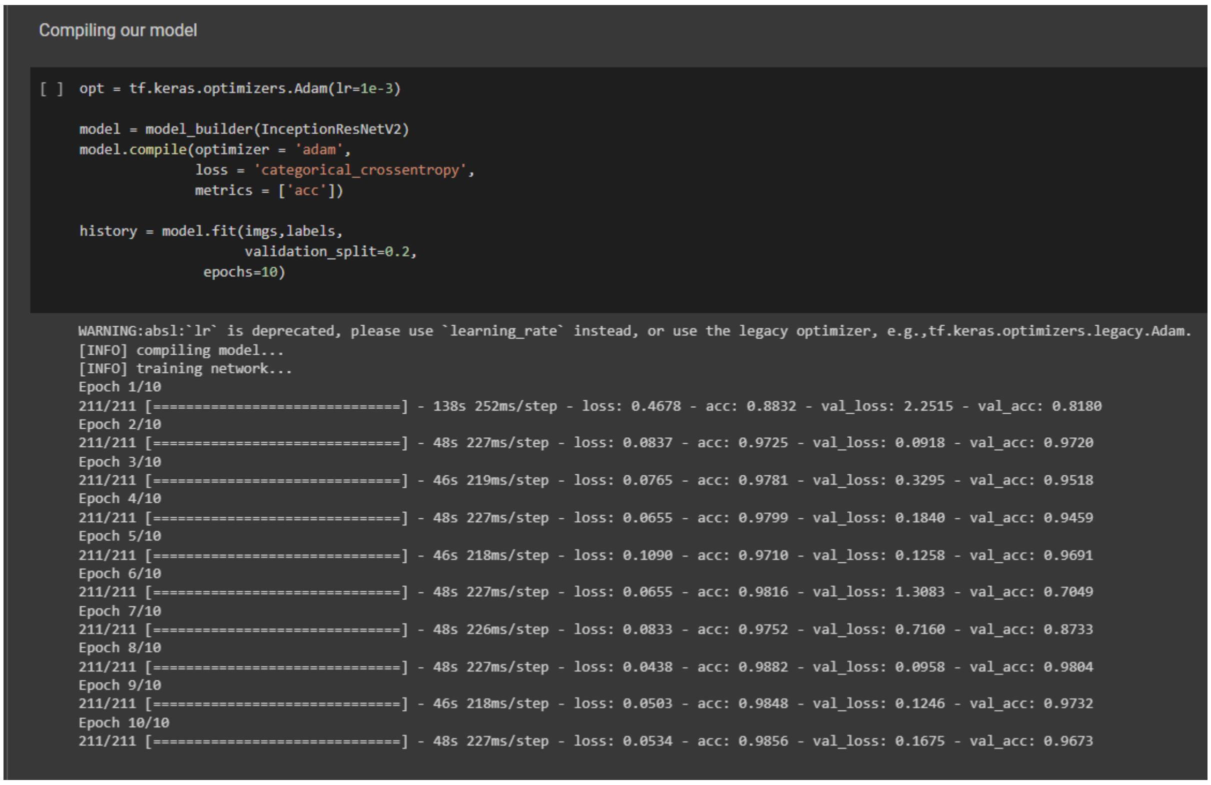
	Model_Name	Accuracy	F1_Score
0	InceptionResNet	0.948454	0.941471
1	VGG19	0.339017	0.171667

- **Hyperparameters of our model**



```
[ ]  tf.keras.optimizers.Adam(
         learning_rate=0.001,
         beta_1=0.9,
         beta_2=0.999,
         epsilon=1e-07,
         amsgrad=False,
         weight_decay=None,
         clipnorm=None,
         clipvalue=None,
         global_clipnorm=None,
         use_ema=False,
         ema_momentum=0.99,
         ema_overwrite_frequency=None,
         jit_compile=True,
         name='Adam',
     )

     <keras.optimizers.optimizer_experimental.adam.Adam at 0x7fdd9b04d550>
```

- <u>**Compiling our model**</u>



Compiling our model

```
[ ]  opt = tf.keras.optimizers.Adam(lr=1e-3)

     model = model_builder(InceptionResNetV2)
     model.compile(optimizer = 'adam',
                   loss = 'categorical_crossentropy',
                   metrics = ['acc'])

     history = model.fit(imgs,labels,
                         validation_split=0.2,
                    epochs=10)

     WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
     [INFO] compiling model...
     [INFO] training network...
     Epoch 1/10
     211/211 [==============================] - 138s 252ms/step - loss: 0.4678 - acc: 0.8832 - val_loss: 2.2515 - val_acc: 0.8180
     Epoch 2/10
     211/211 [==============================] - 48s 227ms/step - loss: 0.0837 - acc: 0.9725 - val_loss: 0.0918 - val_acc: 0.9720
     Epoch 3/10
     211/211 [==============================] - 46s 219ms/step - loss: 0.0765 - acc: 0.9781 - val_loss: 0.3295 - val_acc: 0.9518
     Epoch 4/10
     211/211 [==============================] - 48s 227ms/step - loss: 0.0655 - acc: 0.9799 - val_loss: 0.1840 - val_acc: 0.9459
     Epoch 5/10
     211/211 [==============================] - 46s 218ms/step - loss: 0.1090 - acc: 0.9710 - val_loss: 0.1258 - val_acc: 0.9691
     Epoch 6/10
     211/211 [==============================] - 48s 227ms/step - loss: 0.0655 - acc: 0.9816 - val_loss: 1.3083 - val_acc: 0.7049
     Epoch 7/10
     211/211 [==============================] - 48s 226ms/step - loss: 0.0833 - acc: 0.9752 - val_loss: 0.7160 - val_acc: 0.8733
     Epoch 8/10
     211/211 [==============================] - 48s 227ms/step - loss: 0.0438 - acc: 0.9882 - val_loss: 0.0958 - val_acc: 0.9804
     Epoch 9/10
     211/211 [==============================] - 46s 218ms/step - loss: 0.0503 - acc: 0.9848 - val_loss: 0.1246 - val_acc: 0.9732
     Epoch 10/10
     211/211 [==============================] - 48s 227ms/step - loss: 0.0534 - acc: 0.9856 - val_loss: 0.1675 - val_acc: 0.9673
```

<u>**Accuracy: 98.56%**</u>

Conclusion

In conclusion, deep learning models can be effectively used for malware image classification tasks using the malimg dataset. The malimg dataset consists of 9,335 malware images belonging to 25 different families, and it is a popular dataset used for this task.

Several deep learning models have been used for malware image classification using the malimg dataset, including Convolutional Neural Networks (CNNs) and Transfer Learning. These models have achieved high accuracy rates in correctly identifying the malware families present in the images.

One of the challenges in this task is the small size of the dataset, which can lead to overfitting. However, techniques such as data augmentation and transfer learning can help overcome this challenge.

Overall, deep learning models are a promising approach for malware image classification using the malimg dataset, and further research can explore the use of these models for other malware datasets and applications.