Configuration Manual

MSc Research Project
MSc in Data Analytics

Prateek Mata
Student ID: 21125279

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

Student Name:	Prateek Mata
Student ID:	21125279
Programme:	MSc in Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Catherine Mulwa
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	1549
Page Count:	15

Signature:	Prateek Mata
Date:	1st February 2023

Configuration Manual

Prateek Mata
21125279

1 Introduction

The purpose of this guide is to help any user set up the configuration on their own machine and get the results they want. Both the hardware and software requirements for setting up the environment are detailed in the accompanying documents. This guide features code snippets, exploratory data analysis visualizations, and model assessments.

2 Environment

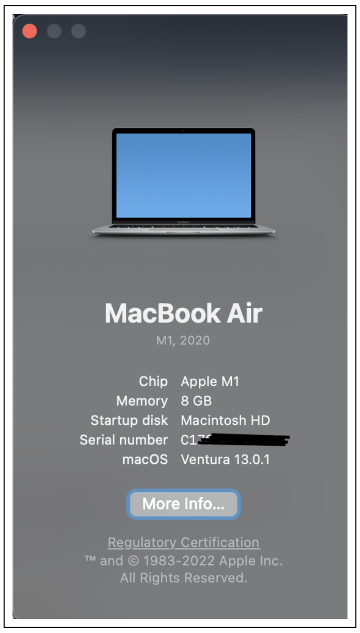
This section describes the entire environment necessary to execute the code solution. This consists of hardware and software configurations, Python libraries and packages, and Google Collaboratory setup.

2.1 Hardware Requirements

The hardware specification used to run this research project and the alternative requirements are summarised in Table1. Figure1 displays the system configuration of the machine used.

Table 1: Hardware Specification

Hardware	Used in this research project	Alternatives
System	MacBook Air M1 2020	Any Windows/Mac/Linux machine
OS	macOS Ventura 13.0.1	Any Windows/Macos/Linux Distribution
HardDisk	256GB	>10GB
Ram	8GB	>4GB
Processor	M1 Silicon	Any Intel/AMD/Apple Silicon
GPU	M1 Silicon	Any integrated/Nvidia/AMD



MacBook Air

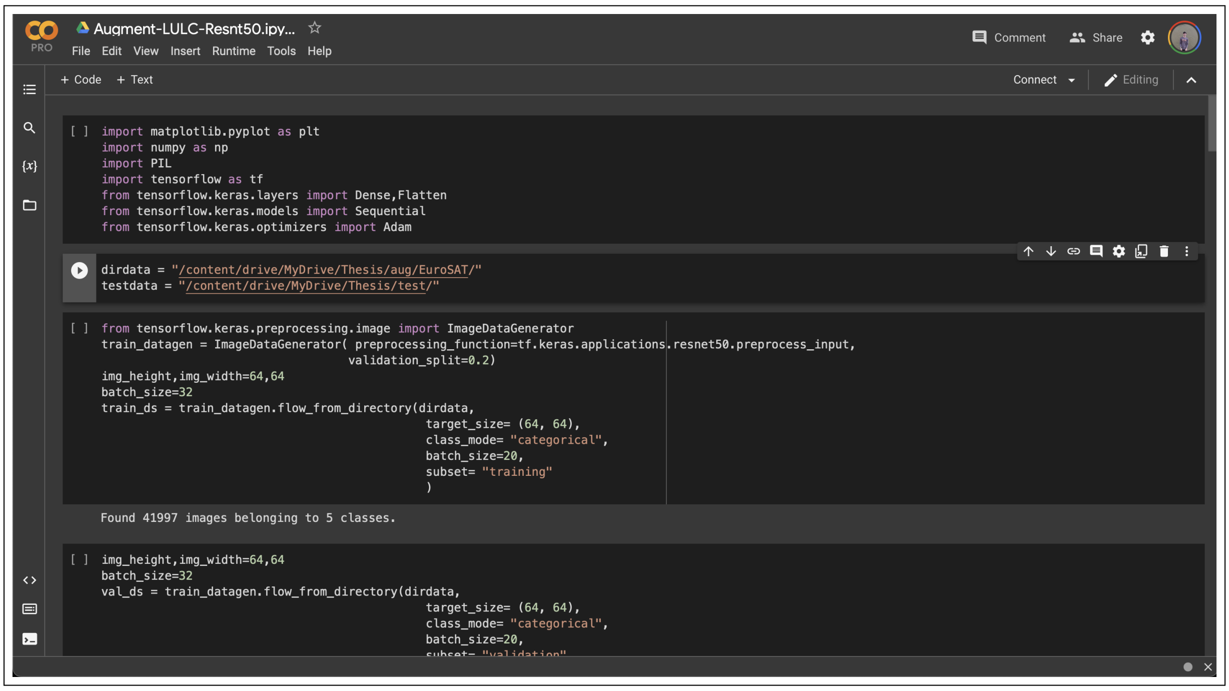
M1, 2020

```
         Chip  Apple M1
       Memory  8 GB
 Startup disk  Macintosh HD
Serial number  C1[redacted]
        macOS  Ventura 13.0.1
```

Figure 1: System Specification

2.2 Setting Up Google Colab Environment

Python was the programming language used for the project. As shown in Figure2, Google collab was used to implement the project. Colab enables the execution of Python code in the browser and is particularly suited to machine learning and deep learning. Colab is technically a hosted Jupyter notebook service that requires no configuration and provides free access to computing resources, including GPUs.



CO PRO
Augment-LULC-Resnt50.ipy...
File Edit View Insert Runtime Tools Help
Comment
Share
+ Code + Text
Connect
Editing

```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

```
dirdata = "/content/drive/MyDrive/Thesis/aug/EuroSAT/"
testdata = "/content/drive/MyDrive/Thesis/test/"
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator( preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
                                   validation_split=0.2)
img_height,img_width=64,64
batch_size=32
train_ds = train_datagen.flow_from_directory(dirdata,
                                              target_size= (64, 64),
                                              class_mode= "categorical",
                                              batch_size=20,
                                              subset= "training"
                                              )
```

```
Found 41997 images belonging to 5 classes.
```

```
img_height,img_width=64,64
batch_size=32
val_ds = train_datagen.flow_from_directory(dirdata,
                                            target_size= (64, 64),
                                            class_mode= "categorical",
                                            batch_size=20,
                                            subset= "validation"
```

Figure 2: Google Colab Overview

Three datasets were used one was acquired from EuroSAT's official github repository and has two variants one with image augmentation applied to it and one without augmentation,the other was custom made by downloading satellites images from European space agency's online portal. The zip files for these datasets can be downloaded[1] and have to be uploaded to google drive. Google drive has to be mounted before to colab before the codebase can access the directories in it using the mount drive button shown in Figure3.

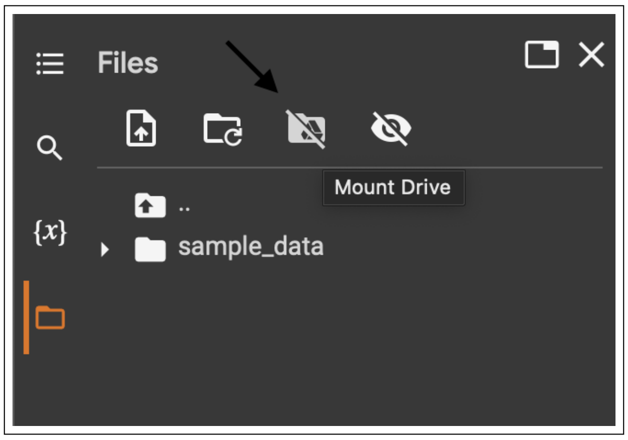
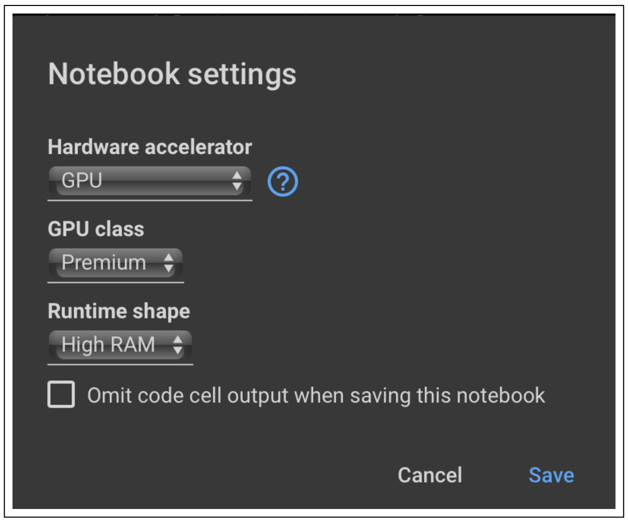


Figure 3: Mount google drive to colab

After mounting the drive a runtime has to be assigned and GPU has to be selected for faster processing time as shown in Figure4. The candidate was using Google colab pro and thus had premium GPU's but the standard GPU's can be used as well and works perfectly fine.



Notebook settings

Hardware accelerator: GPU

GPU class: Premium

Runtime shape: High RAM

☐ Omit code cell output when saving this notebook

Figure 4: Assign Runtime

After doing the steps mentioned above the colab environment is set up and ready to run the codebase for this research project.

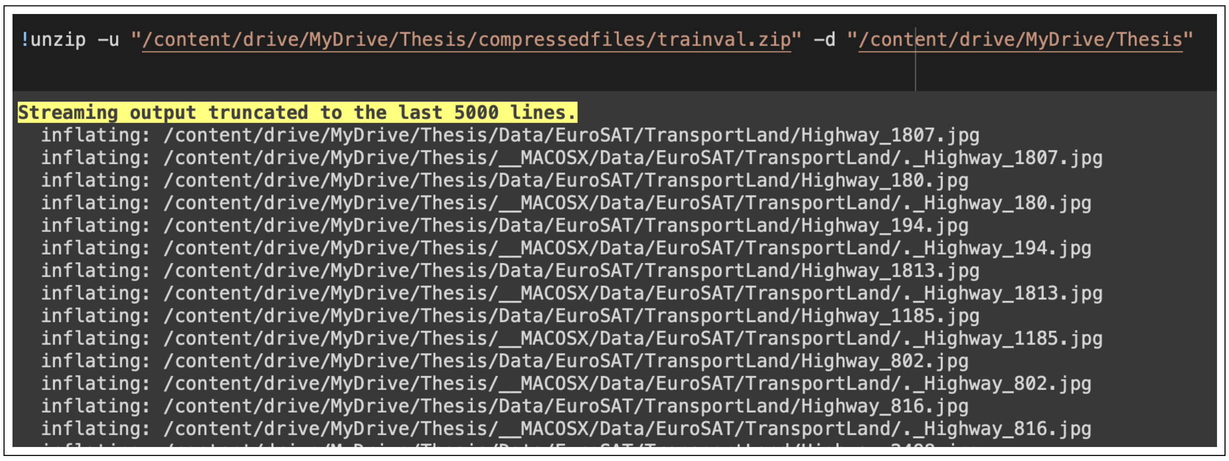
[1]https://drive.google.com/drive/folders/17egtXi4cyVt63wmMDZcay-dpZZXFxs1Q?usp=share_link

3 Implementation

This section provides step-by-step instructions for replicating the research project using the provided code base, including data acquisition, model construction, training, results, and visualizations.

3.1 Reading the Data

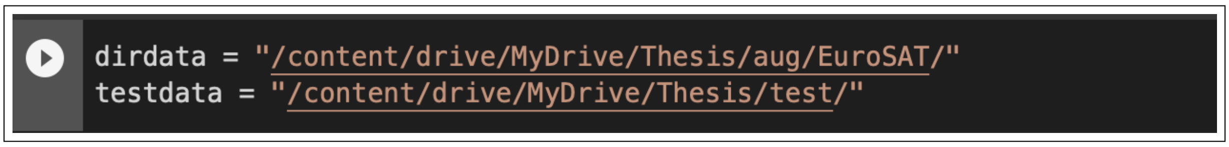
The zip files in google drive have to be extracted first to get the directory path, the compressed files already have train and test folder inside which there are different sub folders which represent the different classes. Figure5 shows the code snippet to unzip the files and store it in drive.



```
!unzip -u "/content/drive/MyDrive/Thesis/compressedfiles/trainval.zip" -d "/content/drive/MyDrive/Thesis"

Streaming output truncated to the last 5000 lines.
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_1807.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_1807.jpg
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_180.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_180.jpg
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_194.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_194.jpg
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_1813.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_1813.jpg
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_1185.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_1185.jpg
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_802.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_802.jpg
  inflating: /content/drive/MyDrive/Thesis/Data/EuroSAT/TransportLand/Highway_816.jpg
  inflating: /content/drive/MyDrive/Thesis/__MACOSX/Data/EuroSAT/TransportLand/._Highway_816.jpg
```

Figure 5: Unzipping the compressed files in drive

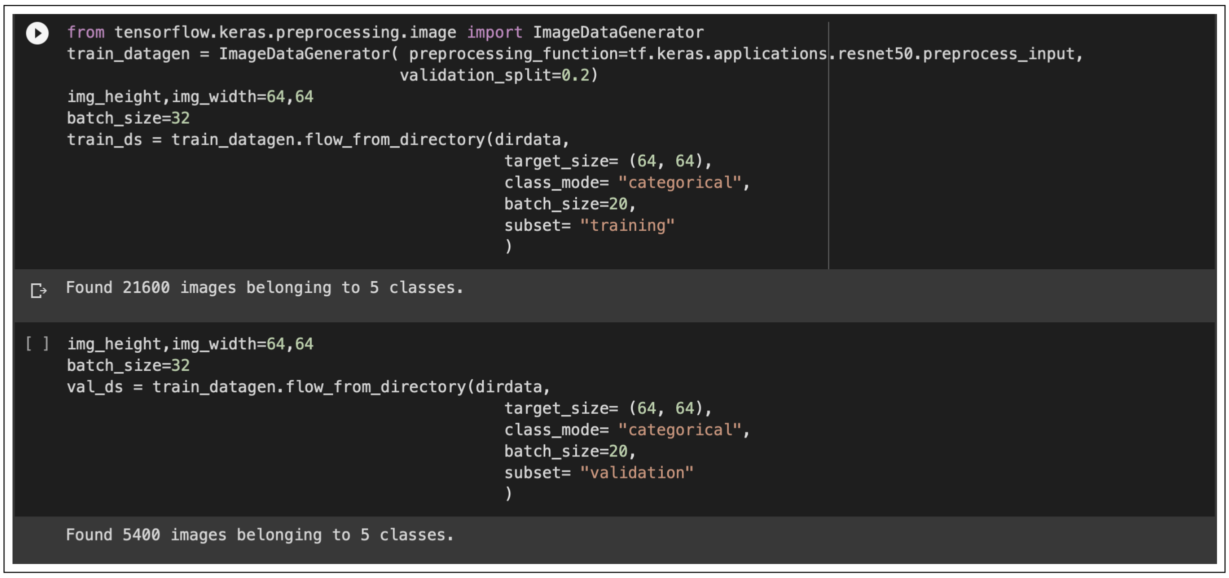
The directories for training and testing data are assigned to fetch data seen in Figure6.



```
dirdata = "/content/drive/MyDrive/Thesis/aug/EuroSAT/"
testdata = "/content/drive/MyDrive/Thesis/test/"
```

Figure 6: Assign the directories to fetch data

3.2 Data Preparation

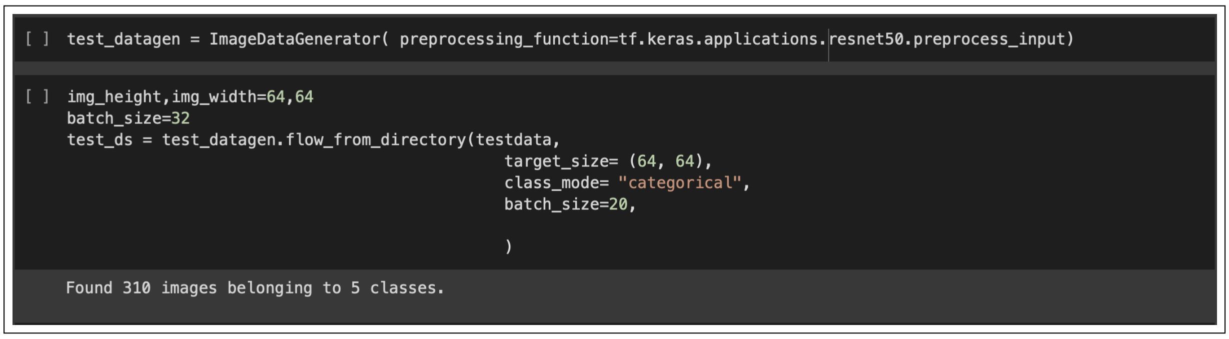


```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator( preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
                                  validation_split=0.2)
img_height,img_width=64,64
batch_size=32
train_ds = train_datagen.flow_from_directory(dirdata,
                                            target_size= (64, 64),
                                            class_mode= "categorical",
                                            batch_size=20,
                                            subset= "training"
                                            )

Found 21600 images belonging to 5 classes.
```

```
img_height,img_width=64,64
batch_size=32
val_ds = train_datagen.flow_from_directory(dirdata,
                                            target_size= (64, 64),
                                            class_mode= "categorical",
                                            batch_size=20,
                                            subset= "validation"
                                            )

Found 5400 images belonging to 5 classes.
```

Figure 7: Training and Validation split after preprocessing

After reading the data, the next step is data preparation. This step begins by creating a data generator using the ImageGenerator function from the keras preprocessing library. This data generator loads the datasets splits it into training and validation sets and from the training directory and test set from the test directory, it preprocesses the image so it could fit the model, gives them a constant size of 64x64 and categroises into the classes. Figure7 and 8 show the training and validation split and test set respectively.



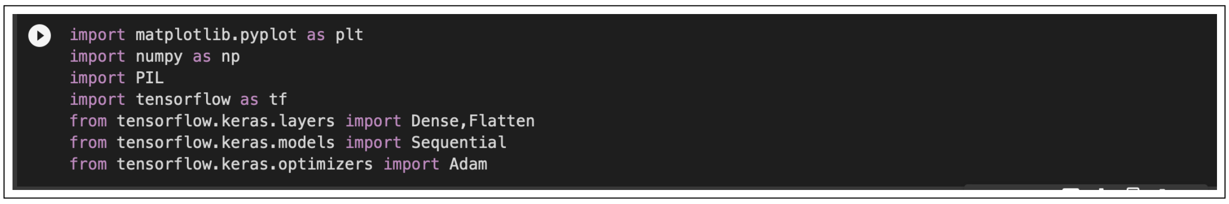
```
test_datagen = ImageDataGenerator( preprocessing_function=tf.keras.applications.resnet50.preprocess_input)
```

```
img_height,img_width=64,64
batch_size=32
test_ds = test_datagen.flow_from_directory(testdata,
                                            target_size= (64, 64),
                                            class_mode= "categorical",
                                            batch_size=20,

                                            )

Found 310 images belonging to 5 classes.
```

Figure 8: Test data after preprocessing using ImageGenerator

3.3 Model Training of ResNet-50

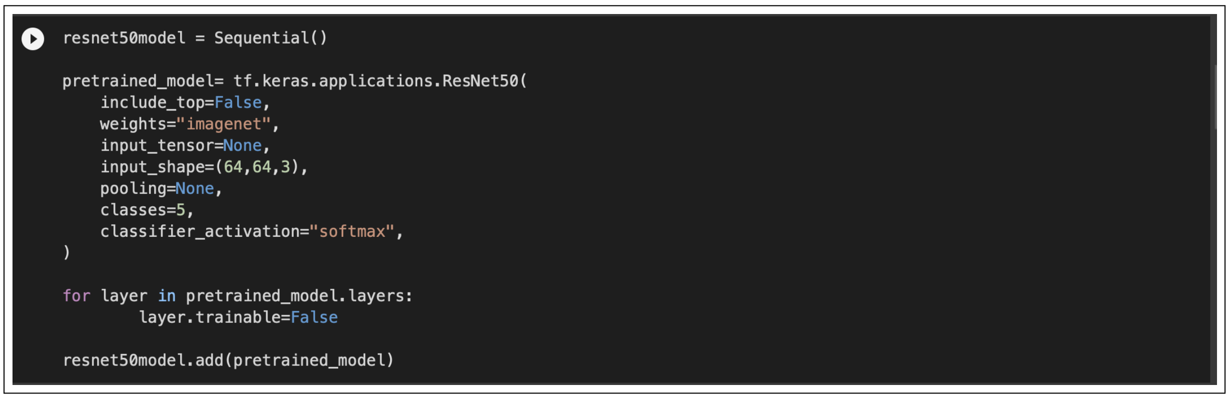
Import the important libraries as shown in Figure9



```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
```

Figure 9: Importing the libraries

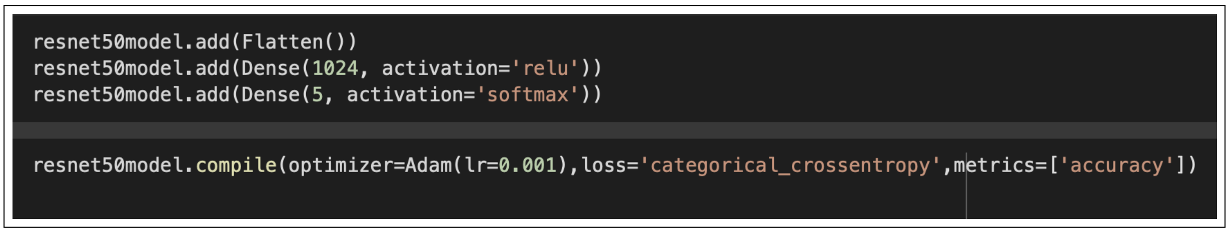
The pretrained model for ResNet50 is then initialised(Figure 10)



```
resnet50model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(64,64,3),
    pooling=None,
    classes=5,
    classifier_activation="softmax",
)

for layer in pretrained_model.layers:
        layer.trainable=False

resnet50model.add(pretrained_model)
```

Figure 10: Initialising the ResNet-50 model

Three new layers are added to the pretrained ResNet50 and the model is compiled using the parameters as shown in Figure 11.

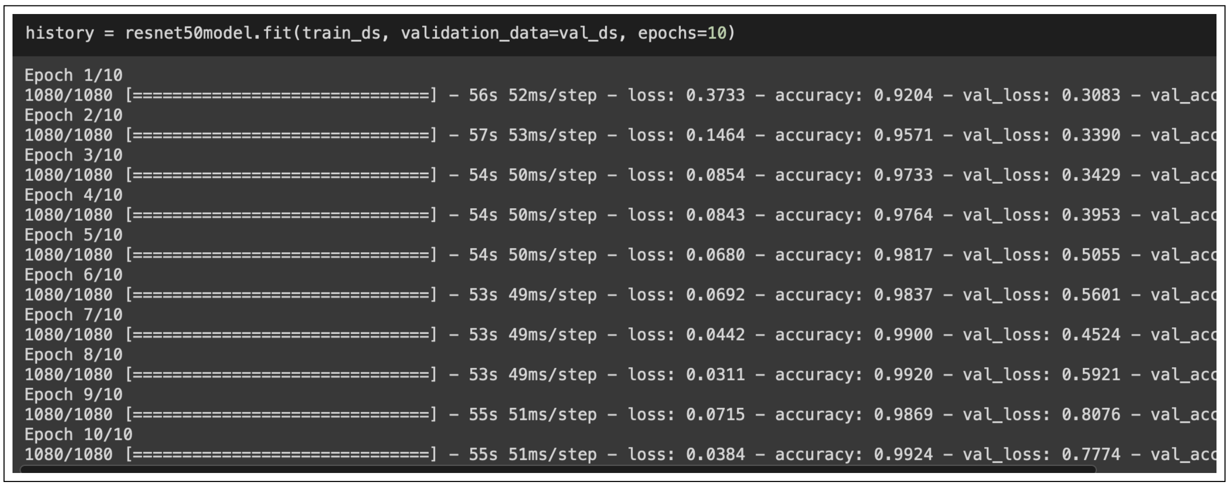


```
resnet50model.add(Flatten())
resnet50model.add(Dense(1024, activation='relu'))
resnet50model.add(Dense(5, activation='softmax'))
```

```
resnet50model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

Figure 11: Adding layers to ResNet-50

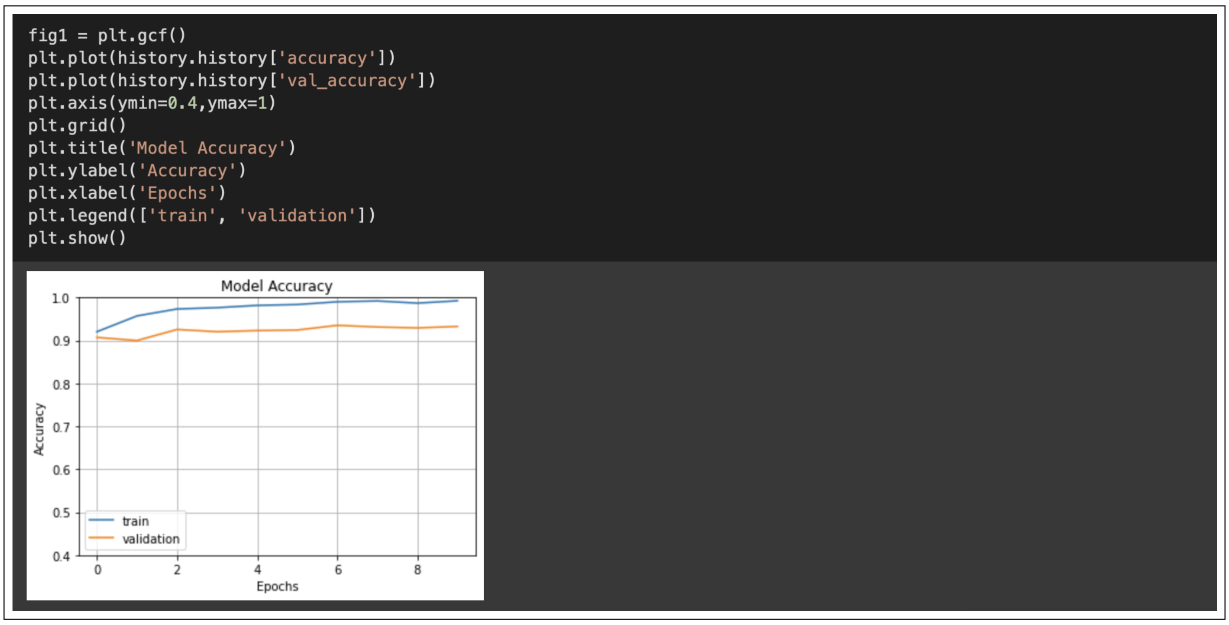
The model is then fitted with the the training and validation sets from both augmented and non augmented images to build two seperate models and run for 10 epochs each.(Figure 12)



```
history = resnet50model.fit(train_ds, validation_data=val_ds, epochs=10)

Epoch 1/10
1080/1080 [==============================] - 56s 52ms/step - loss: 0.3733 - accuracy: 0.9204 - val_loss: 0.3083 - val_acc
Epoch 2/10
1080/1080 [==============================] - 57s 53ms/step - loss: 0.1464 - accuracy: 0.9571 - val_loss: 0.3390 - val_acc
Epoch 3/10
1080/1080 [==============================] - 54s 50ms/step - loss: 0.0854 - accuracy: 0.9733 - val_loss: 0.3429 - val_acc
Epoch 4/10
1080/1080 [==============================] - 54s 50ms/step - loss: 0.0843 - accuracy: 0.9764 - val_loss: 0.3953 - val_acc
Epoch 5/10
1080/1080 [==============================] - 54s 50ms/step - loss: 0.0680 - accuracy: 0.9817 - val_loss: 0.5055 - val_acc
Epoch 6/10
1080/1080 [==============================] - 53s 49ms/step - loss: 0.0692 - accuracy: 0.9837 - val_loss: 0.5601 - val_acc
Epoch 7/10
1080/1080 [==============================] - 53s 49ms/step - loss: 0.0442 - accuracy: 0.9900 - val_loss: 0.4524 - val_acc
Epoch 8/10
1080/1080 [==============================] - 53s 49ms/step - loss: 0.0311 - accuracy: 0.9920 - val_loss: 0.5921 - val_acc
Epoch 9/10
1080/1080 [==============================] - 55s 51ms/step - loss: 0.0715 - accuracy: 0.9869 - val_loss: 0.8076 - val_acc
Epoch 10/10
1080/1080 [==============================] - 55s 51ms/step - loss: 0.0384 - accuracy: 0.9924 - val_loss: 0.7774 - val_acc
```

Figure 12: Fitting the model with train and validation sets

A graph is plotted as shown in Figure 13 to show the training and validation accuracy over the epochs

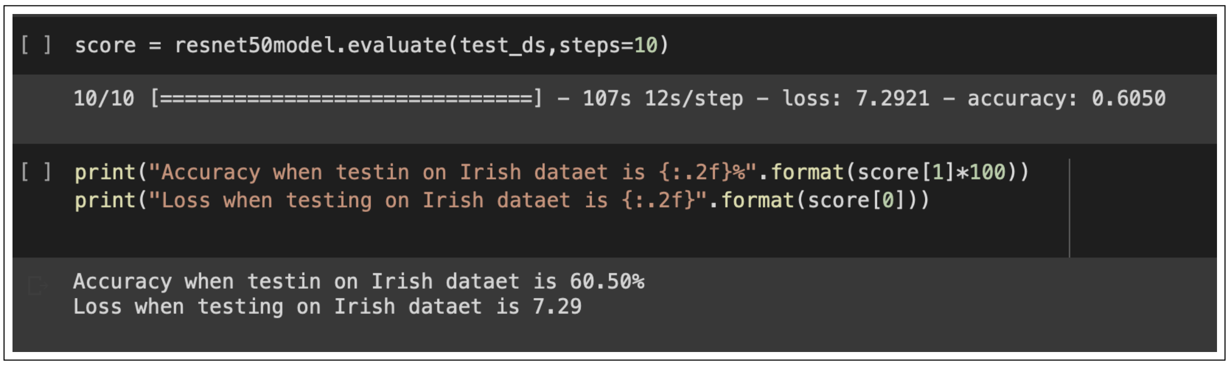


```
fig1 = plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.show()
```

Figure 13: Plotting accuracy over the epochs

3.4 Evaluation and Results of ResNet-50

The fitted model is then evaluated using the test set of Irish images which gives accuracy and loss as shown in Figure 14



```
[ ]  score = resnet50model.evaluate(test_ds,steps=10)

     10/10 [==============================] - 107s 12s/step - loss: 7.2921 - accuracy: 0.6050

[ ]  print("Accuracy when testin on Irish dataet is {:.2f}%".format(score[1]*100))
     print("Loss when testing on Irish dataet is {:.2f}".format(score[0]))

     Accuracy when testin on Irish dataet is 60.50%
     Loss when testing on Irish dataet is 7.29
```

Figure 14: Evaluate the model using the test set

After the evaluation confusion matrix for the fitted model with train and validation sets as well as confusion matrix for evaluation with test set is plotted using the confusion matrix from sklearn as seen in Figure 15

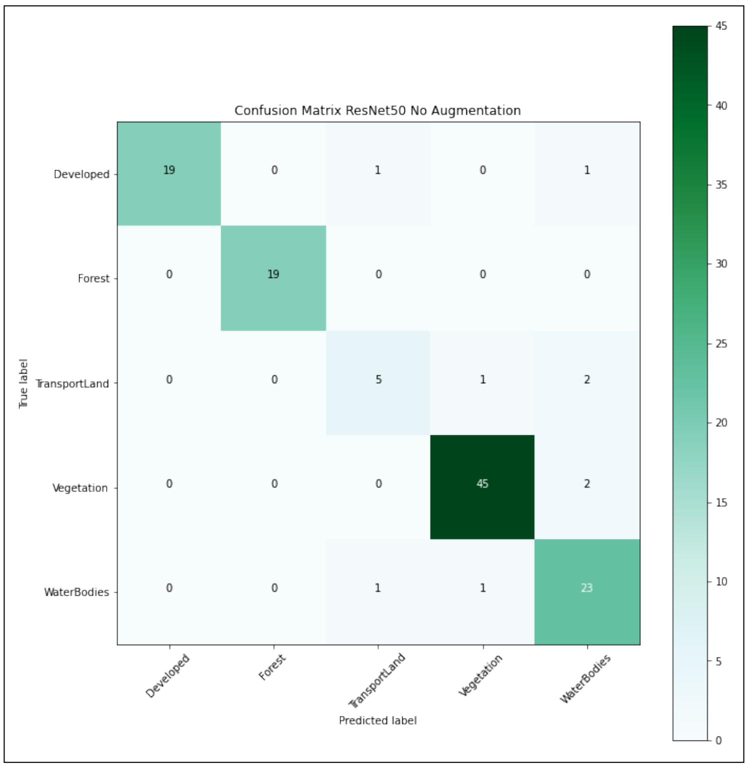


Figure 15: Confusion matrix for ResNet-50(Training and validation split) without augmentation

The confusion matrix is plotted for both training and test datasets using the two ResNet-50 models trained on training set with and without Image augmentation. The Figures 16, 17 and 18 show the rest of the confusion matrices for the ResNet-50 models.

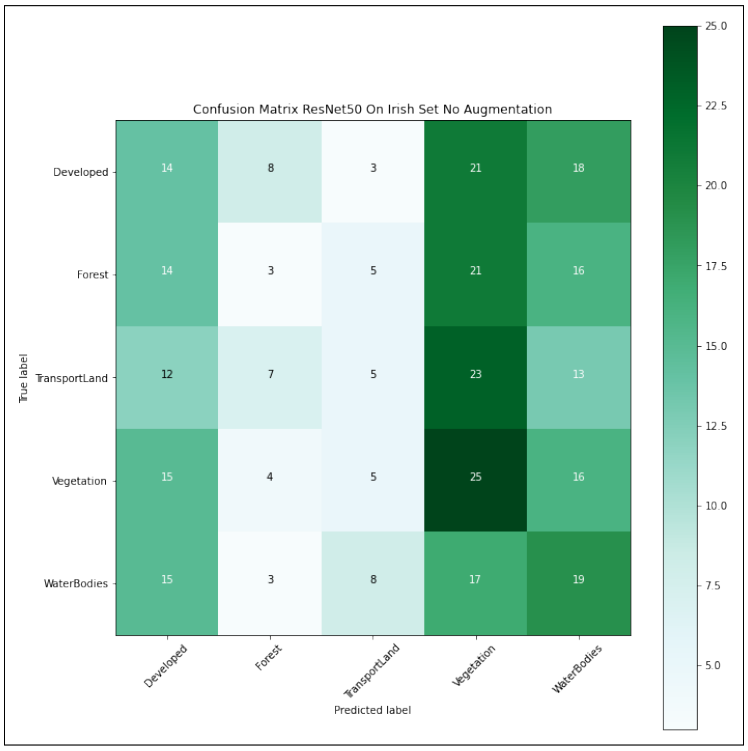


Figure 16: Confusion matrix for ResNet-50(Test Set) without augmentation

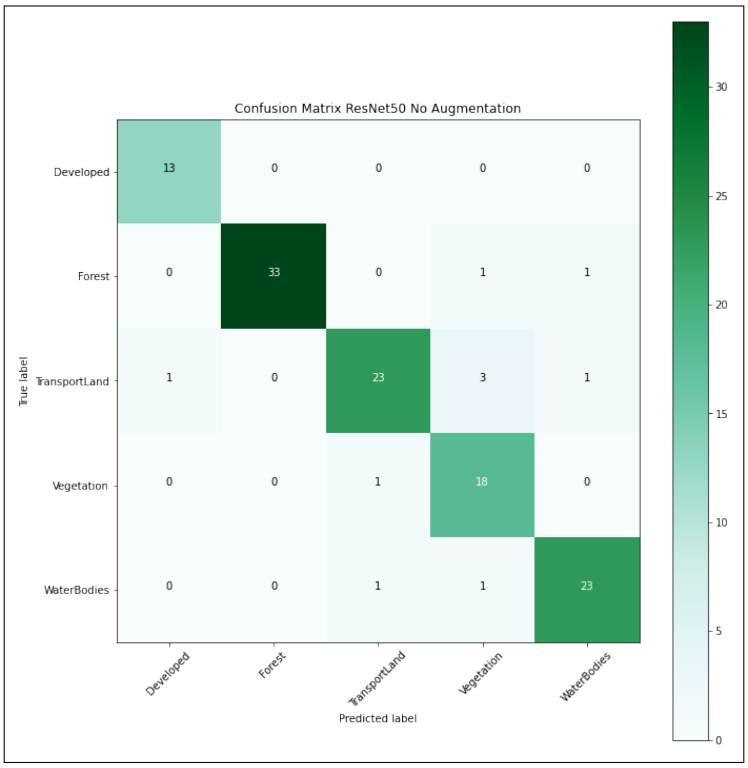


Figure 17: Confusion matrix for ResNet-50(Training and validation split) with augmentation

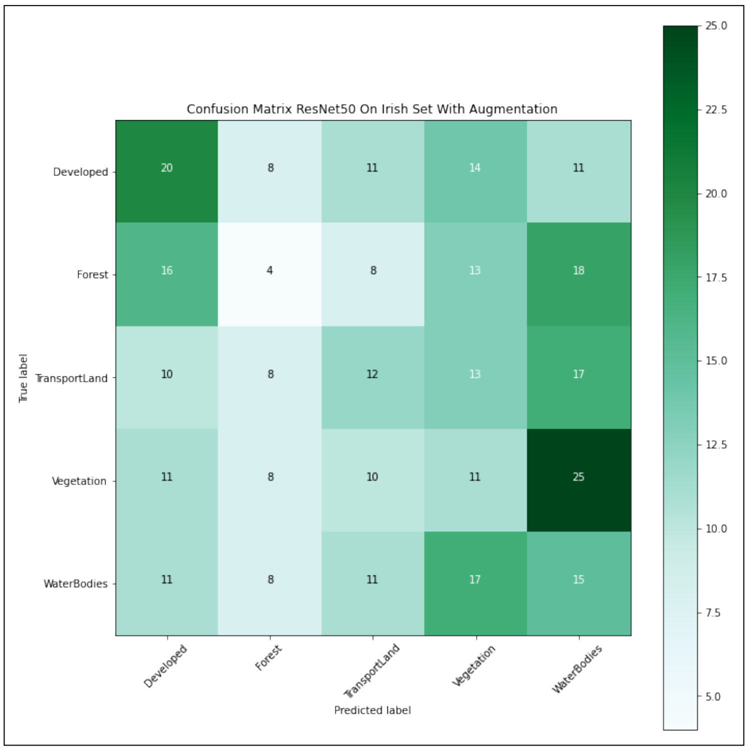
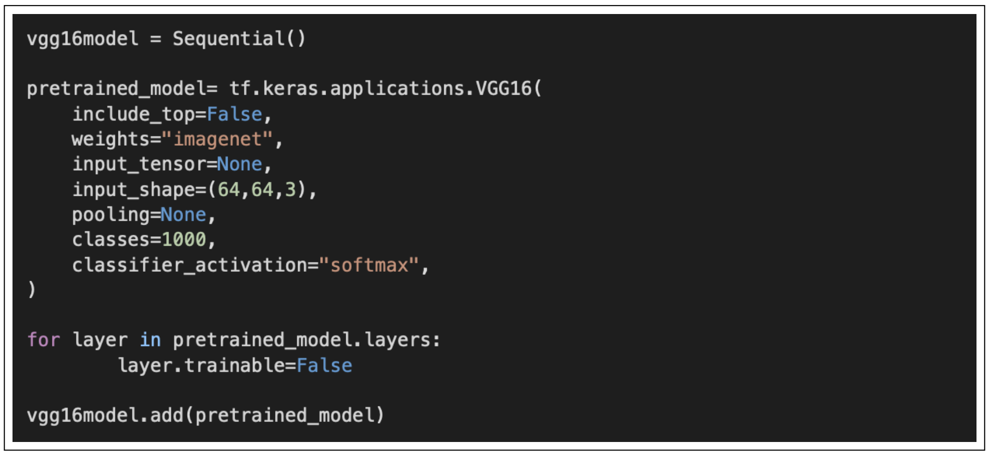


Figure 18: Confusion matrix for ResNet-50(Test set) with augmentation

3.5 Model Training of VGG-16

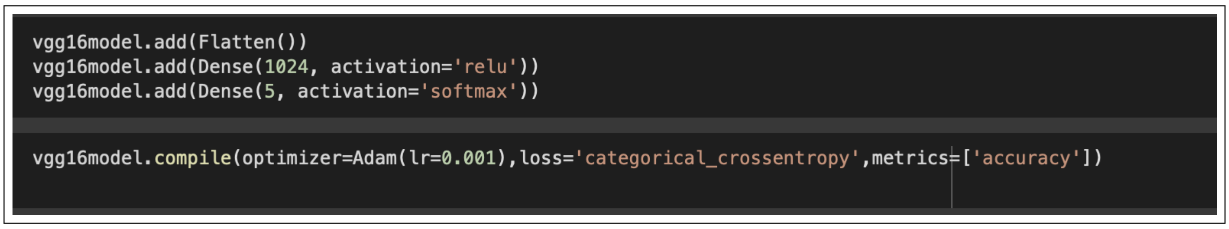
Similar to the previous models all the steps till section 3.2 are to be carried out, the VGG-16 model is then initialised as shown in Figure 19



```
vgg16model = Sequential()

pretrained_model= tf.keras.applications.VGG16(
    include_top=False,
    weights="imagenet",
    input_tensor=None,
    input_shape=(64,64,3),
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)

for layer in pretrained_model.layers:
        layer.trainable=False

vgg16model.add(pretrained_model)
```

Figure 19: Initialising the VGG-16 model

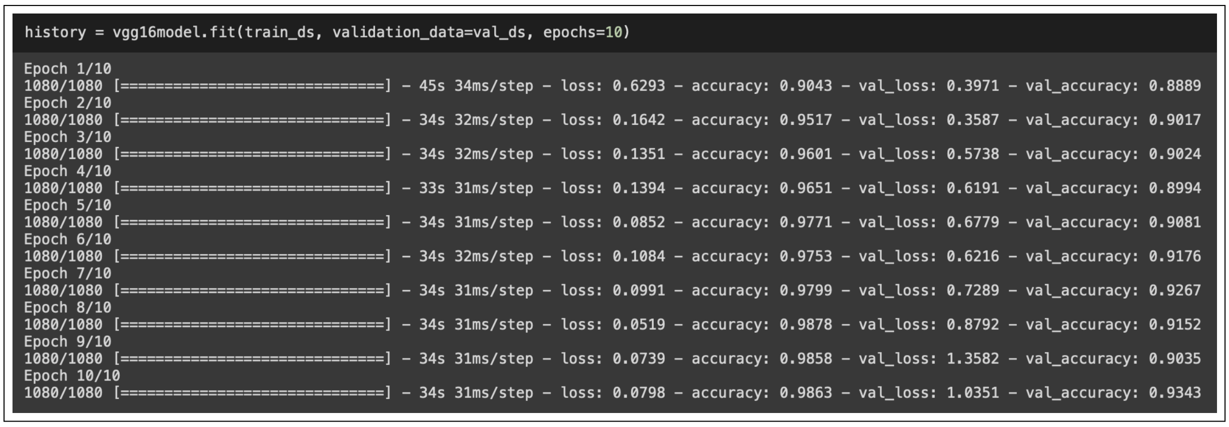
The pretrained VGG-16 is given three new layers, and the model is built using the parameters shown in Figure 20.



```
vgg16model.add(Flatten())
vgg16model.add(Dense(1024, activation='relu'))
vgg16model.add(Dense(5, activation='softmax'))

vgg16model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

Figure 20: Adding layers to VGG-16

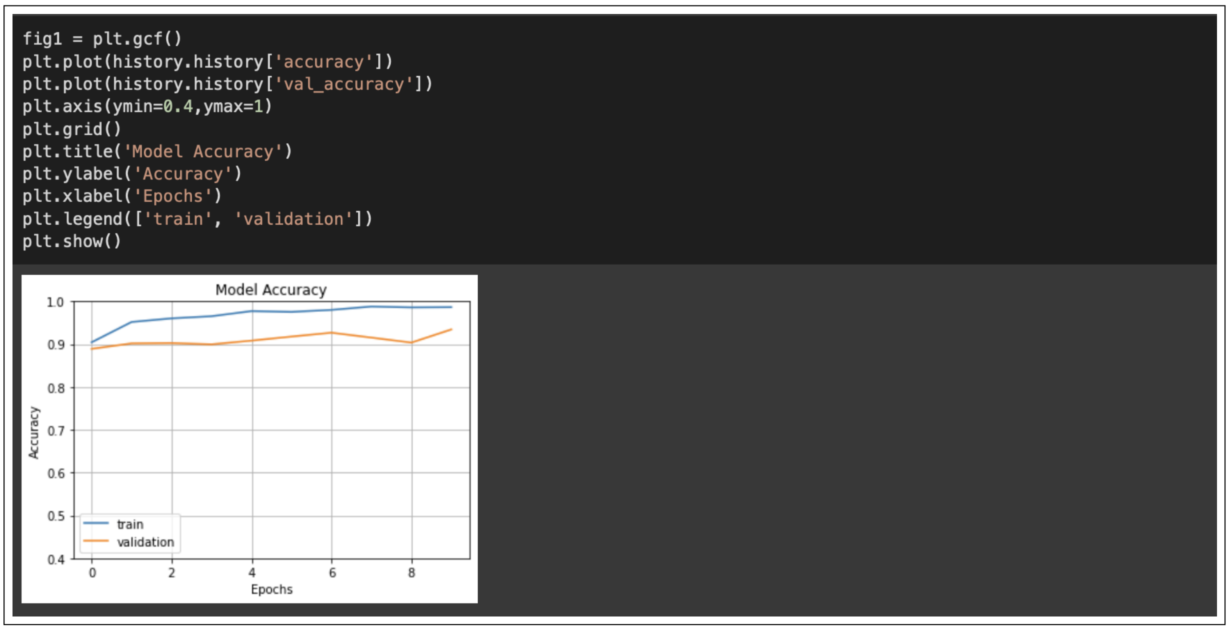
The model is then fitted with the the training and validation sets from both augmented and non augmented images to build two separate models and run for 10 epochs each.(Figure 21)



```
history = vgg16model.fit(train_ds, validation_data=val_ds, epochs=10)

Epoch 1/10
1080/1080 [==============================] - 45s 34ms/step - loss: 0.6293 - accuracy: 0.9043 - val_loss: 0.3971 - val_accuracy: 0.8889
Epoch 2/10
1080/1080 [==============================] - 34s 32ms/step - loss: 0.1642 - accuracy: 0.9517 - val_loss: 0.3587 - val_accuracy: 0.9017
Epoch 3/10
1080/1080 [==============================] - 34s 32ms/step - loss: 0.1351 - accuracy: 0.9601 - val_loss: 0.5738 - val_accuracy: 0.9024
Epoch 4/10
1080/1080 [==============================] - 33s 31ms/step - loss: 0.1394 - accuracy: 0.9651 - val_loss: 0.6191 - val_accuracy: 0.8994
Epoch 5/10
1080/1080 [==============================] - 34s 31ms/step - loss: 0.0852 - accuracy: 0.9771 - val_loss: 0.6779 - val_accuracy: 0.9081
Epoch 6/10
1080/1080 [==============================] - 34s 32ms/step - loss: 0.1084 - accuracy: 0.9753 - val_loss: 0.6216 - val_accuracy: 0.9176
Epoch 7/10
1080/1080 [==============================] - 34s 31ms/step - loss: 0.0991 - accuracy: 0.9799 - val_loss: 0.7289 - val_accuracy: 0.9267
Epoch 8/10
1080/1080 [==============================] - 34s 31ms/step - loss: 0.0519 - accuracy: 0.9878 - val_loss: 0.8792 - val_accuracy: 0.9152
Epoch 9/10
1080/1080 [==============================] - 34s 31ms/step - loss: 0.0739 - accuracy: 0.9858 - val_loss: 1.3582 - val_accuracy: 0.9035
Epoch 10/10
1080/1080 [==============================] - 34s 31ms/step - loss: 0.0798 - accuracy: 0.9863 - val_loss: 1.0351 - val_accuracy: 0.9343
```

Figure 21: Fitting the model with train and validation sets

A graph is plotted as shown in Figure 22 to show the training and validation accuracy over the epochs

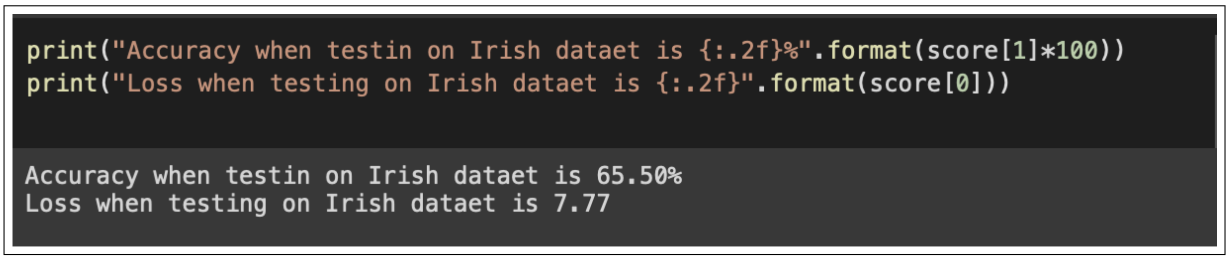


```
fig1 = plt.gcf()
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.axis(ymin=0.4,ymax=1)
plt.grid()
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'validation'])
plt.show()
```

Figure 22: Plotting accuracy over the epochs

3.6 Evaluation and Results of VGG-16

Using the test set of Irish images, the fitted model is then evaluated, yielding accuracy and loss as shown in Figure 23



```
print("Accuracy when testin on Irish dataet is {:.2f}%".format(score[1]*100))
print("Loss when testing on Irish dataet is {:.2f}".format(score[0]))
```

```
Accuracy when testin on Irish dataet is 65.50%
Loss when testing on Irish dataet is 7.77
```

Figure 23: Evaluate the model using the test set

After the evaluation confusion matrix for the fitted model with train and validation sets as well as confusion matrix for evaluation with test set is plotted using the confusion matrix from sklearn as seen in Figure 24

The confusion matrix is plotted for both training and test datasets using the two ResNet-50 models trained on training set with and without Image augmentation. The Figures25, 26 and 27 show the rest of the confusion matrices for the VGG16 models.

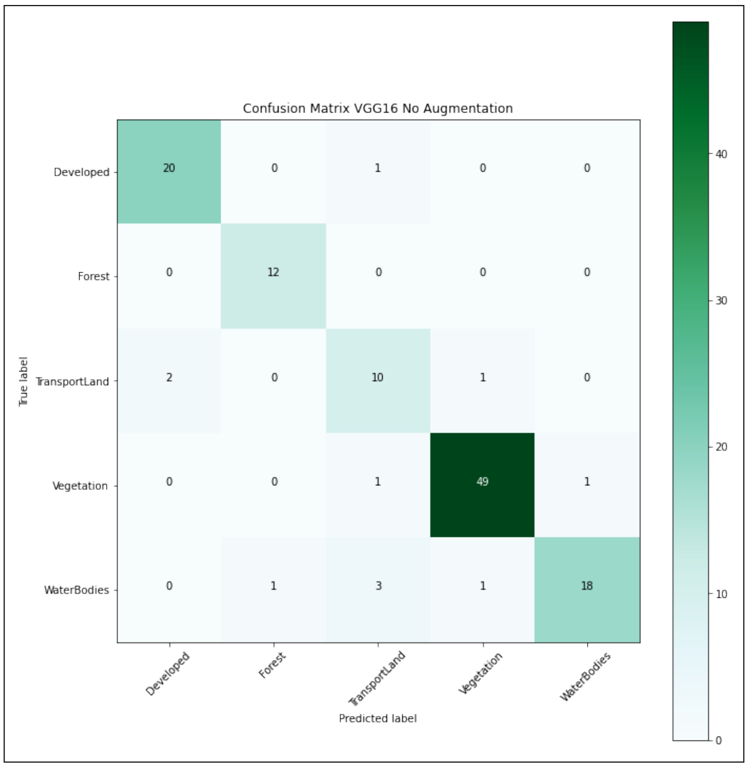


Figure 24: Confusion matrix for VGG-16(Training and validation split) without augmentation

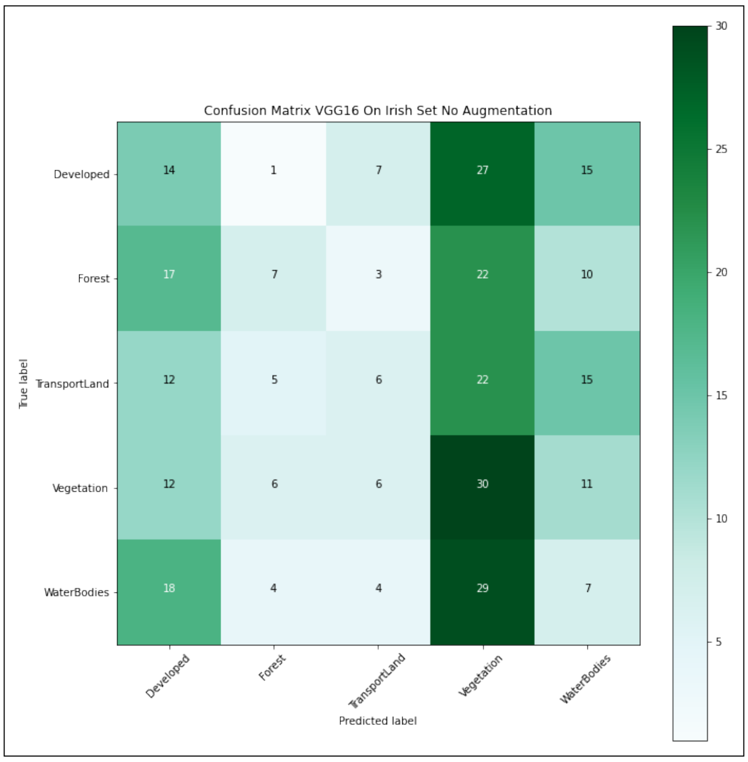


Figure 25: Confusion matrix for VGG-16(Test Set) without augmentation

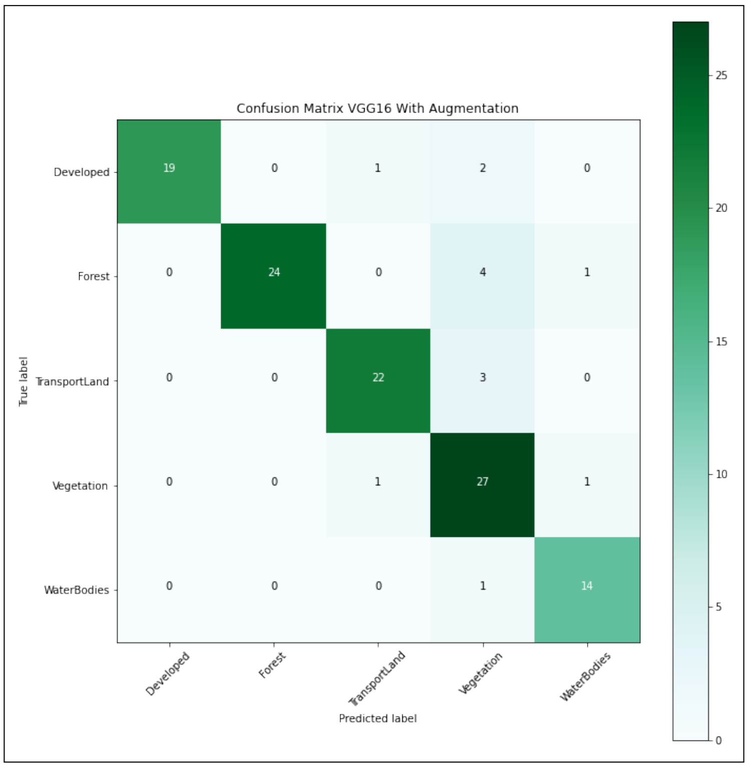


Figure 26: Confusion matrix for VGG-16(Training and validation split) with augmentation

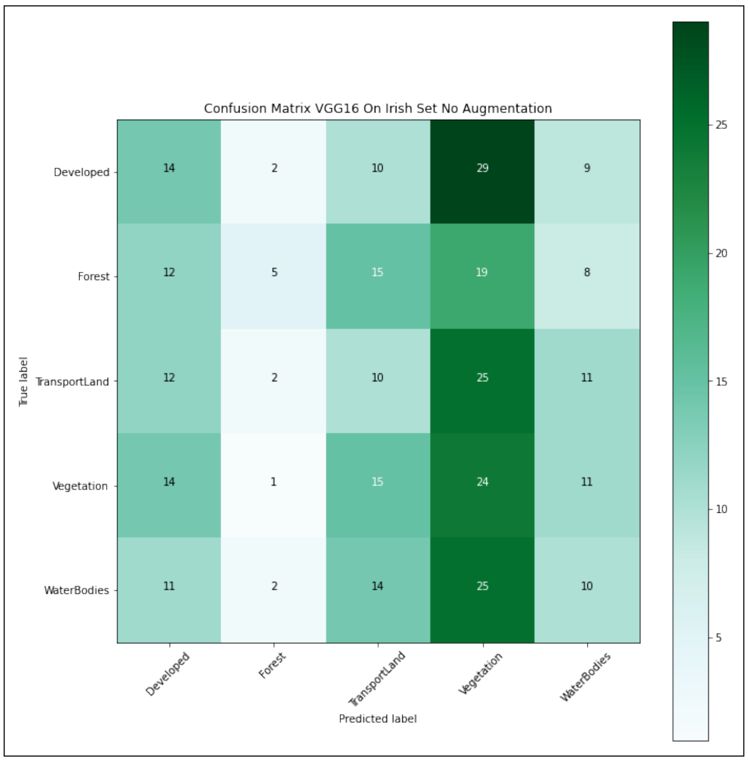


Figure 27: Confusion matrix for VGG16(Test set) with augmentation

4 Conclusion

Performing the steps mentioned in the sections above, the codebase for this reserach project can be replicated and implemented to yield similar results and better understand the working of this project and contribute to it.

References

sklearn.metrics.confusion_matrix (n.d.). https://scikit-learn/stable/modules/generated/sklearn.metrics.confusion_matrix.html. [Online;].

Team, K. (n.d.a). Keras documentation: Model training APIs, https://keras.io/api/models/model_training_apis/. [Online;].

Team, K. (n.d.b). Keras documentation: Resnet and ResNetV2, https://keras.io/api/applications/resnet/. [Online;].

Team, K. (n.d.c). Keras documentation: Vgg16 and VGG19, https://keras.io/api/applications/vgg/. [Online;].