National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Harshitha Somanna
Student ID: x22150366

School of Computing
National College of Ireland

Supervisor: Christina Hava Muntean

National College of Ireland
MSc Project Submission Sheet
School of Computing

Student Name:	Harshitha Poolakanda Somanna
Student ID:	x22150366
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Christina Hava Muntean
Submission Due Date:	14/12/2023
Project Title:	A Deep Learning-Based System for Plant Disease Detection and Classification in Arabica Coffee Leaves
Word Count:	800
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Harshitha Poolakanda Somanna
Date:	14th December 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: A Deep Learning-Based System for Plant Disease Detection and Classification in Arabica Coffee Leaves

Harshitha Poolakanda Somanna
x22150366

1. Introduction

This document provides details on the hardware and software components that are used in building A Deep Learning-Based System for Plant Disease Recognition and Classification in Arabica Coffee Leaves. The steps for setting up, creating, running, and testing this research using the suggested framework are discussed in this manual.

2. Hardware Specification.

Table 1. Hardware Specification

Hardware Used	**Specification**
Operating System	Windows 10 Pro Version 22H2
RAM	16.0 GB
Processor	Intel(R) Core(TM) i5-6300U CPU @ 2.40GHz 2.50 GHz
System Type	64-bit operating system, x64-based processor

3. Software Specification

Table 2. Software Specification

Software Used	**Version**
Anaconda Navigator	2022.10
Python	3.9.13

4. Required Libraries

Table 3. Libraries Used

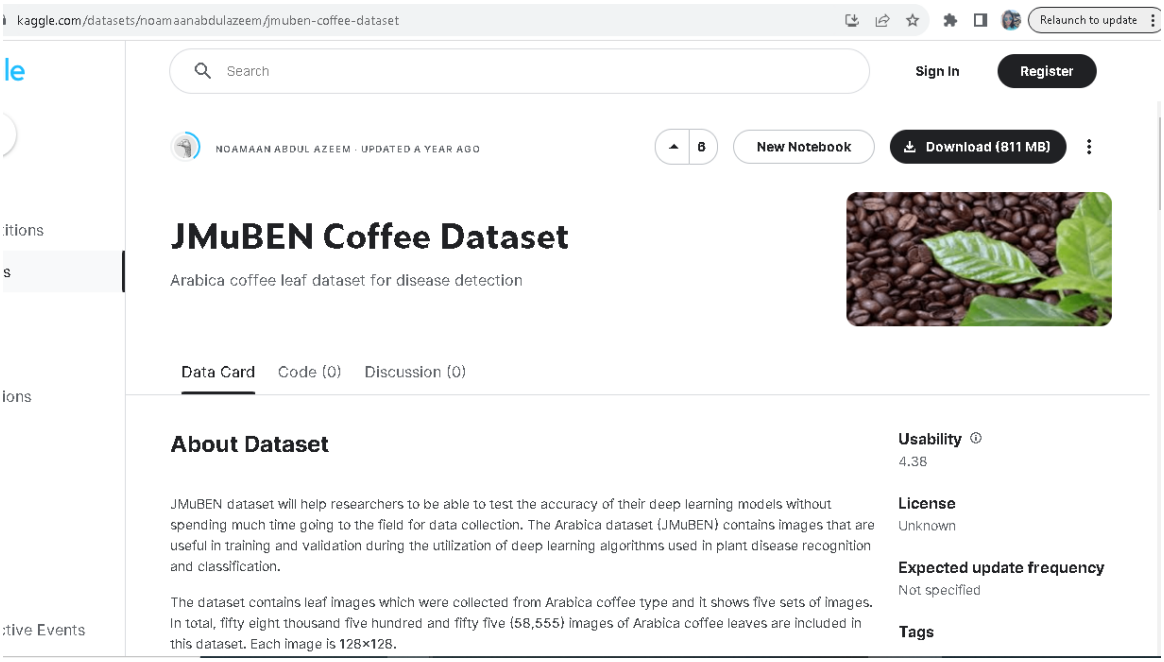
matplotlib	fuzzywuzzy
numpy	warnings
pandas	random
tensorflow	PIL
seaborn	imblearn
cv2	keras
itertools	tensorflow_addons
pathlib	

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as img
import cv2
import itertools
import pathlib
import warnings
import os
import random
import time
import gc
from IPython.display import Markdown, display
from PIL import Image
from random import randint
warnings.filterwarnings('ignore')
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import matthews_corrcoef as MCC
from sklearn.metrics import balanced_accuracy_score as BAS
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
import keras
from sklearn.utils import shuffle
from tensorflow import keras
from keras import Sequential
from keras import layers
import tensorflow as tf
import tensorflow_addons as tfa
from tensorflow.keras.preprocessing import image_dataset_from_directory
from keras.utils.vis_utils import plot_model
from tensorflow.keras import Sequential, Input
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.layers import Dense, Dropout,SeparableConv2D, Activation, BatchNormalization, Flatten, GlobalAveragePo
from tensorflow.keras.layers import Conv2D, Flatten
from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator as IDG
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, plot_confusion_matrix
from tensorflow.keras import layers, models, optimizers, callbacks
from tensorflow.keras.applications import MobileNetV2, InceptionV3, VGG16, EfficientNetV2S
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Add, AveragePooling2D, Dense, AvgPool2D,BatchNormalization, ReLU, DepthwiseConv2D
```

Fig 1. Importing the required libraries

5. Dataset

The dataset is available in the kaggle website and is free to download without any permission. Dataset consists of 58,405 images representing five classes, Phoma (6571 images), Cescospora (7681 images), Rust (8192 images), Healthy (18983 images), and Miner (16978 images).
Link: https://www.kaggle.com/datasets/noamaanabdulazeem/jmuben-coffee-dataset



5.1 Data Pre-Processing

The dataset is resized to a size of 128*128 and stored in a separate folder.

```
# input directory
input_dir = r'C:\Users\Admin\Documents\Coffee leaves\JMuBEN'
# output directory
output_dir = r'C:\Users\Admin\Documents\Coffee leaves\JMuBEN_resized'

if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Classes
classes = ['Healthy', 'Miner', 'Leaf rust', 'Cerscospora', 'Phoma']

# Target size
IMG_SIZE = (128, 128)

for class_name in classes:
    class_input_dir = os.path.join(input_dir, class_name)
    class_output_dir = os.path.join(output_dir, class_name)

    # Creating output directory for the class
    if not os.path.exists(class_output_dir):
        os.makedirs(class_output_dir)

    # Iterate through the files in the class directory
    for filename in os.listdir(class_input_dir):
        if filename.endswith('.jpg'):
            # Loading and resizing the image
            img_path = os.path.join(class_input_dir, filename)
            img = Image.open(img_path)
            img = img.resize(IMG_SIZE)

            # Saving the resized image to the output directory
            output_path = os.path.join(class_output_dir, filename)
            img.save(output_path)

print("All images have been resized and saved to the output directory.")
```

Fig 2. Resizing the images to 128x128 size

The dataset is iterated over three sets of data for all five classes and is inturn split into train, test and validation set for all five classes in the ratio 0.7, 0.2 and 0.1 respectively.

```
input_path =  r'C:\Users\Admin\Documents\Coffee leaves\JMuBEN_resized'
output_path = r'C:\Users\Admin\Documents\Coffee leaves\JMuBEN_resized_split'

if os.path.exists(output_path):
    shutil.rmtree(output_path)

for i in range(3):
    os.makedirs(f"{output_path}/experiment{i+1}")

split_folders = os.listdir(output_path)

for i in range(3):
    split_folder = f"{output_path}/{split_folders[i]}"
    splitfolders.ratio(input_path, output=split_folder, seed=1337+i, ratio=(0.7, 0.1, 0.2))
```

Fig 3. Splitting the dataset into 3 experiments and also into train, test and val set and storing in a separate directory for all 3 experiments.

6. Implementation of the Model

A series of steps is carried on for implementing the model and the steps are as follows

6.1 Setting the Hyperparameters

```
batchSize = 32
imageSize = (128, 128)
labes = 5
inputShape = (128, 128, 3)
numberOfEpochs = 30
patience = 30
```

Fig 4. Setting the Hyperparameters

6.2 Function to calculate computational complexity

```
def computational_complexity(model_h5_path):

    model = tf.keras.models.load_model(model_h5_path)

    flops = tf.profiler.experimental.profile(model, options=tf.profiler.experimental.ProfileOptionBuilder.float_operation())

    gflops = flops.total_float_ops / 1e9

    return gflops
```

Fig 5. Code snippet to calculate computational complexity

6.3 Function for Model evaluation and Model Progress

```
def model_progress(history, path, i):

    acc = history.history["acc"]
    val_acc = history.history["val_acc"]

    loss = history.history["loss"]
    val_loss = history.history["val_loss"]

    epochs = range(1, len(acc) + 1)

    plot_graph(epochs, acc, "Train", val_acc, "Validation", "Train and Validation Accuracy", "Epochs", "Accuracy", path, "Ac
    plot_graph(epochs, loss, "Train", val_loss, "Validation", "Train and Validation Loss", "Epochs", "Loss", path, "Loss", i

    return len(acc)
```

```
def model_evaluation(y_true, y_pred):

    accuracy = accuracy_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred, average="weighted")
    precision = precision_score(y_true, y_pred, average="weighted")
    recall = recall_score(y_true, y_pred, average="weighted")

    return accuracy, f1, precision, recall
```

Fig 6. Code snippet to for mode progess and model evaluation

6.4 Saving the metrics in a .csv file

```python
def metrics_evaluation(total_params, gflops, epochs, training_time, accuracy, f1, precision, recall, i, path, model_name):

    metrics = {
        "Experiment": str(i),
        "Total Params": total_params,
        "GFLOPS": gflops,
        "Epochs": epochs,
        "Training Time (sec)": training_time,
        "Test Accuracy": accuracy,
        "Test F1 Weighted": f1,
        "Test Precision Weighted": precision,
        "Test Recall Weighted": recall
    }

    df_new = pd.DataFrame(data=[metrics])
    file_path = f"{path}/metrics_{model_name}.csv"

    if not os.path.exists(path):
        os.makedirs(path)

    if os.path.exists(file_path):
        df = pd.read_csv(file_path)
        df = df.append(df_new, ignore_index=True)
        df.to_csv(file_path, header=True, index=False)
    else:
        df_new.to_csv(file_path, header=True, index=False)
```

Fig 7. Code snippet for metrics evaluation

6.5 Generation of confusion matrix

```python
def confusion_matrix_scorer(model, class_names, test_generator, y_true_test, path, model_name, i):

    class estimator:
        _estimator_type = ''
        classes_= []

        def __init__(self, model, classes):
            self.model = model
            self._estimator_type = "classifier"
            self.classes_ = classes

        def predict(self, X):
            y_prob= self.model.predict(X)
            y_pred = y_prob.argmax(axis=1)
            return y_pred


    classifier = estimator(model, class_names)

    cm = plot_confusion_matrix(estimator=classifier, X=test_generator, y_true=y_true_test, xticks_rotation=45, cmap="Blues")
    cm.ax_.set_title(f"Confusion Matrix - {model_name}")
    cm.ax_.set_xlabel("Predicted labels")
    cm.ax_.set_xticklabels(class_names)
    cm.ax_.set_ylabel("True labels")
    cm.ax_.set_yticklabels(class_names)

    file_path = f"{path}/{i}-ConfusionMatrix.jpg"

    plt.savefig(file_path, dpi=115, bbox_inches="tight")
    plt.close()
    plt.clf()
```

Fig 8. Code snippet to generate confusion matrix

6.6 Generating the dataframe

```python
def data_image_generator(train_dir, validation_dir, test_dir):

    train_datagen = ImageDataGenerator(rescale=1./255)
    validation_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
        train_dir,
        target_size=imageSize,
        batch_size=batchSize,
        class_mode="categorical")

    validation_generator = validation_datagen.flow_from_directory(
        validation_dir,
        target_size=imageSize,
        batch_size=batchSize,
        class_mode="categorical")

    test_generator = test_datagen.flow_from_directory(
        test_dir,
        target_size=imageSize,
        batch_size=1,
        class_mode="categorical",
        shuffle = False)

    return train_generator, validation_generator, test_generator
```

Fig 9. Code snippet to generate train, test and val dataframe

6.7 Model implementation for CNN

```python
def cnn(input_shape=(128, 128, 3), num_labels=5):
    model = models.Sequential()

    # Convolutional Layers
    model.add(layers.Conv2D(32, (5, 5), input_shape=input_shape, activation='relu', name="conv2d_1"))
    model.add(layers.MaxPooling2D(pool_size=(3, 3), name="max_pooling2d_1"))
    model.add(layers.Conv2D(32, (3, 3), activation='relu', name="conv2d_2"))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), name="max_pooling2d_2"))
    model.add(layers.Conv2D(64, (3, 3), activation='relu', name="conv2d_3"))
    model.add(layers.MaxPooling2D(pool_size=(2, 2), name="max_pooling2d_3"))

    # Flatten Layer
    model.add(layers.Flatten(name="flatten_1"))

    # Dense Layers
    model.add(layers.Dense(512, activation='relu'))
    model.add(layers.Dropout(0.25))
    model.add(layers.Dense(num_labels, activation='softmax'))

    return model
```

Fig 10. Code snippet for CNN implementation

6.8 Model implementation for MobileNetV2

```python
def mobilenetv2():
    mobilenetv2 = tf.keras.applications.MobileNetV2(
        input_shape=INPUT_SHAPE,
        alpha=1.0,
        include_top=True,
        weights=None,
        input_tensor=None,
        pooling="max",
        classes=NUM_LABELS,
        classifier_activation=None)

    last = mobilenetv2.layers[-2]
    x = layers.Flatten()(last.output)
    x = layers.Dense(512, activation="relu")(x)
    predictions = layers.Dense(NUM_LABELS, activation="softmax")(x)

    mobilenetv2_architecture = models.Model(mobilenetv2.input, predictions)
    return mobilenetv2_architecture
```

Fig 10. Code snippet for MobileNetV2 implementation

6.9 Model implementation for VGG16

```python
def vgg16():

    # Load pre-trained VGG16 model without top layers
    vgg16 = tf.keras.applications.VGG16(
        include_top=False,  # Set to False to exclude the top layers
        weights=None,  # Use pre-trained weights
        input_tensor=None,
        input_shape=INPUT_SHAPE,
        pooling='max',  # Use 'max' pooling
        classes=NUM_LABELS,  # Number of classes in your classification task
        classifier_activation=None)

    # Freeze the layers of the pre-trained VGG16 model
    for layer in vgg16.layers:
        layer.trainable = False

    # Get the output of the last convolutional layer in VGG16
    last = vgg16.output

    # Flatten the output
    x = layers.Flatten()(last)

    # Add a dense layer with 512 units and ReLU activation
    x = layers.Dense(512, activation="relu")(x)

    # Add the final dense layer with the number of classes and softmax activation
    predictions = layers.Dense(NUM_LABELS, activation="softmax")(x)

    # Create a new model with VGG16 as the input and the custom dense layers
    vgg16_architecture = models.Model(vgg16.input, predictions)

    return vgg16_architecture
```

Fig 11. Code snippet for VGG16 implementation

6.10 Model implementation for EfficientNetB0

```python
def efficientnet():
    efficientnet = tf.keras.applications.EfficientNetB0(
        include_top=False,
        weights=None,
        input_tensor=None,
        input_shape=inputShape,
        pooling='max',  # or 'avg' for average pooling
        classes=labes,
        classifier_activation=None)

    last = efficientnet.get_layer(index=-1)
    x = layers.Flatten()(last.output)
    x = layers.Dense(512, activation="relu")(x)
    predictions = layers.Dense(labes, activation="softmax")(x)

    efficientnet_architecture = models.Model(efficientnet.input, predictions)
    return efficientnet_architecture
```

Fig 12. Code snippet for EfficientNetB0 implementation

6.11 Model Compilation

The steps required for model compilation for CNN is shown in the code snippet. The same follows for EfficientNetB0, VGG16 and MobileNetV2.

```python
for i in range(1, 4):
    print(f">> EXPERIMENT {i}")

        model_architecture = cnn()

        model_architecture.compile(loss="categorical_crossentropy", optimizer=optimizers.RMSprop(learning_rate=0.0004),
                                   metrics=["acc"])
        model_architecture.summary()

        # Training model and verifying training time
        print(f"\n>> Training {model_name} model for Experiment {i}...")
        beginning = time.time()

        history = model_architecture.fit(
            train_generator,
            steps_per_epoch=100,
            epochs=EPOCHS,
            validation_data=validation_generator,
            validation_steps=50,
            callbacks=[csv_logger, early_stopping, model_checkpoint],
            verbose=0)

        end = time.time()
        training_time = end - beginning

        # Saving the best model
        model_architecture.save(f"{artifacts_folder}/{model_name}_exp{i}.h5")
        print(f"Best model for {model_name} saved for Experiment {i}.")

        # Getting GFLOPS
        gflops = get_gflops(f"{artifacts_folder}/{model_name}_exp{i}.h5")

        # Model progress
        model_evaluation_path = f"{evaluation}/{model_name}"
        total_epochs = model_training_progress(history, model_evaluation_path, i)
        print(f"Progress model for {model_name} saved for Experiment {i}.")

        # Loading model architecture and weights
        model_architecture_json = model_architecture.to_json()
        model_architecture = tf.keras.models.model_from_json(model_architecture_json)
        model_architecture.load_weights(f"{checkpoints_path}/checkpoint_{model_name}")
```

```
    # Testing model
    print(f"\nTesting {model_name} model for Experiment {i}...")
    STEP_SIZE_TEST = test_generator.n // test_generator.batch_size
    test_generator.reset()

    y_pred = model_architecture.predict(test_generator, steps=STEP_SIZE_TEST, verbose=1)
    y_pred = np.argmax(y_pred, axis=1)

    # Mapping predicted integers to labels
    labels = (train_generator.class_indices)
    labels = dict((value, key) for key, value in labels.items())

    y_pred = [labels[element] for element in y_pred]

    y_true = test_generator.filenames
    y_true = [element[:element.find('/')] for element in y_true]

    # Extract labels from directory names in filenames
    y_true = [filename.split("\\")[0] for filename in test_generator.filenames]

    total_params = model_architecture.count_params()

    # Evaluating model and saving metrics
    accuracy, f1, precision, recall = evaluate_model(y_true, y_pred)
    metrics_evaluation(total_params, gflops, total_epochs, training_time, accuracy, f1, precision, recall, i, model_eval
    print("Model evaluated.")

    # Plotting confusion matrix
    class_names = os.listdir(test_dir)

    # Debug label extraction
    true_labels = []

    threshold = 80

    for filename in y_true:
        match, score = process.extractOne(filename, labels.values(), scorer=fuzz.token_set_ratio)
        if score >= threshold:
            for key, value in labels.items():
                if filename == value:
                    true_labels.append(key)
                    break

      y_true = y_true[:len(true_labels)]

      print("Length of y_true:", len(y_true))
      print("Length of true_labels:", len(true_labels))

      confusion_matrix_scorer(model_architecture, class_names, test_generator, true_labels, model_evaluation_path,
                              model_name, i)
      print("Confusion Matrix saved.")
      print("\n\n\n")

   # Timing for each experiment
   experiment_end = time.time()
   experiment_time = experiment_end - total_beginning
   print(f"\n>> Experiment {i} completed. Total time for Experiment {i}: {experiment_time:.2f} seconds.\n")

# Timing for the entire loop
total_end = time.time()
total_time = total_end - total_beginning
print(f"\n>> All experiments completed. Total time for all experiments: {total_time:.2f} seconds.\n")
```

Fig 13. Code snippet for model compilation of CNN

```
# Timing for the entire loop
total_beginning = time.time()

for i in range(1, 4):
    print(f">> EXPERIMENT {i}")

        # Creating and compiling model
        model_architecture = mobilenetv2()

        model_architecture.compile(loss="categorical_crossentropy", optimizer=optimizers.RMSprop(learning_rate=0.0004),
                                   metrics=["acc"])
        model_architecture.summary()

        # Training model and verifying training time
        print(f"\n>> Training {model_name} model for Experiment {i}...")
        beginning = time.time()

        history = model_architecture.fit(
            train_generator,
            steps_per_epoch=100,
            epochs=EPOCHS,
            validation_data=validation_generator,
            validation_steps=50,
            callbacks=[csv_logger, early_stopping, model_checkpoint],
            verbose=0)

        end = time.time()
        training_time = end - beginning

         # Saving the best model
        model_architecture.save(f"{artifacts_folder}/{model_name}_exp{i}.h5")
        print(f"Best model for {model_name} saved for Experiment {i}.")

        # Getting GFLOPS
        gflops = get_gflops(f"{artifacts_folder}/{model_name}_exp{i}.h5")

        # Model progress
        model_evaluation_path = f"{evaluation}/{model_name}"
        total_epochs = model_training_progress(history, model_evaluation_path, i)
        print(f"Progress model for {model_name} saved for Experiment {i}.")
```

Fig 14. Code snippet for model compilation of MobileNetV2

```
# Timing for the entire loop
total_beginning = time.time()

for i in range(1, 4):
    print(f">> EXPERIMENT {i}")

        # Creating and compiling model
        model_architecture = vgg16()

        model_architecture.compile(loss="categorical_crossentropy", optimizer=optimizers.RMSprop(learning_rate=0.0004),
                                   metrics=["acc"])
        model_architecture.summary()

        # Training model and verifying training time
        print(f"\n>> Training {model_name} model for Experiment {i}...")
        beginning = time.time()

        history = model_architecture.fit(
            train_generator,
            steps_per_epoch=100,
            epochs=EPOCHS,
            validation_data=validation_generator,
            validation_steps=50,
            callbacks=[csv_logger, early_stopping, model_checkpoint],
            verbose=0)

        end = time.time()
        training_time = end - beginning

        # Saving the best model
        model_architecture.save(f"{artifacts_folder}/{model_name}_exp{i}.h5")
        print(f"Best model for {model_name} saved for Experiment {i}.")

        # Getting GFLOPS
        gflops = get_gflops(f"{artifacts_folder}/{model_name}_exp{i}.h5")

        # Model progress
        model_evaluation_path = f"{evaluation}/{model_name}"
        total_epochs = model_training_progress(history, model_evaluation_path, i)
        print(f"Progress model for {model_name} saved for Experiment {i}.")
```

Fig 15. Code snippet for model compilation of VGG16

```python
# Timing for the entire loop
total_beginning = time.time()

for i in range(1, 4):
    print(f">> EXPERIMENT {i}")

        # Creating and compiling model
        model_architecture = efficientnet()

        model_architecture.compile(loss="categorical_crossentropy", optimizer=optimizers.RMSprop(learning_rate=0.0004),
                                   metrics=["acc"])
        model_architecture.summary()

        # Training model and verifying training time
        print(f"\n>> Training {model_name} model for Experiment {i}...")
        beginning = time.time()

        history = model_architecture.fit(
            train_generator,
            steps_per_epoch=100,
            epochs=EPOCHS,
            validation_data=validation_generator,
            validation_steps=50,
            callbacks=[csv_logger, early_stopping, model_checkpoint],
            verbose=0)

        end = time.time()
        training_time = end - beginning

        # Saving the best model
        model_architecture.save(f"{artifacts_folder}/{model_name}_exp{i}.h5")
        print(f"Best model for {model_name} saved for Experiment {i}.")

        # Getting GFLOPS
        gflops = get_gflops(f"{artifacts_folder}/{model_name}_exp{i}.h5")

        # Model progress
        model_evaluation_path = f"{evaluation}/{model_name}"
        total_epochs = model_training_progress(history, model_evaluation_path, i)
        print(f"Progress model for {model_name} saved for Experiment {i}.")
```

Fig 16. Code snippet for model compilation of EfficientNet

6.12 Model Training

```
>> Training cnn model for Experiment 3...
Best model for cnn saved for Experiment 3.
Progress model for cnn saved for Experiment 3.

Testing cnn model for Experiment 3...
11685/11685 [==============================] - 74s 6ms/step
Model evaluated.
Length of y_true: 11685
Length of true_labels: 11685
11685/11685 [==============================] - 73s 6ms/step
Confusion Matrix saved.




>> Experiment 3 completed. Total time for Experiment 3: 3456.49 seconds.
```

Fig 17. Model training progress for CNN

```
>> Training mobilenetv2 model for Experiment 3...
Best model for mobilenetv2 saved for Experiment 3.
Progress model for mobilenetv2 saved for Experiment 3.

Testing mobilenetv2 model for Experiment 3...
11685/11685 [==============================] - 260s 22ms/step
Metrics saved to CSV:
   Experiment  Total Params    GFLOPS  Epochs  Training Time (sec)
0           3       2916421  0.008748      30          5552.077375

   Test Accuracy  Test F1 Weighted  Test Precision Weighted  \
0       0.324947          0.159388                  0.10559

   Test Recall Weighted
0              0.324947
Model evaluated.
Length of y_true: 11685
Length of true_labels: 11685
11685/11685 [==============================] - 259s 22ms/step
```

Fig 17. Model training progress for MobileNetV2

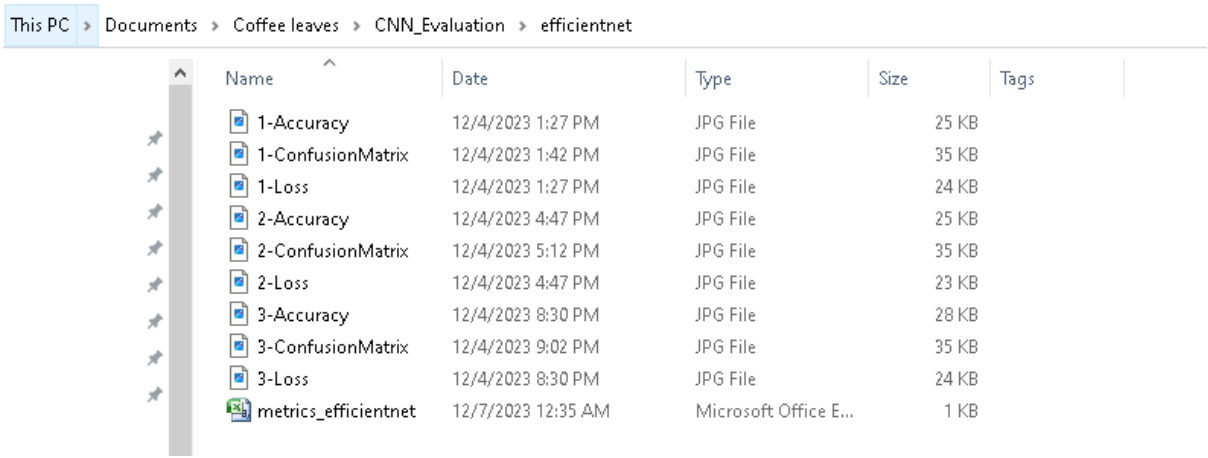


Fig 18. Path where the evaluation metrics are saved

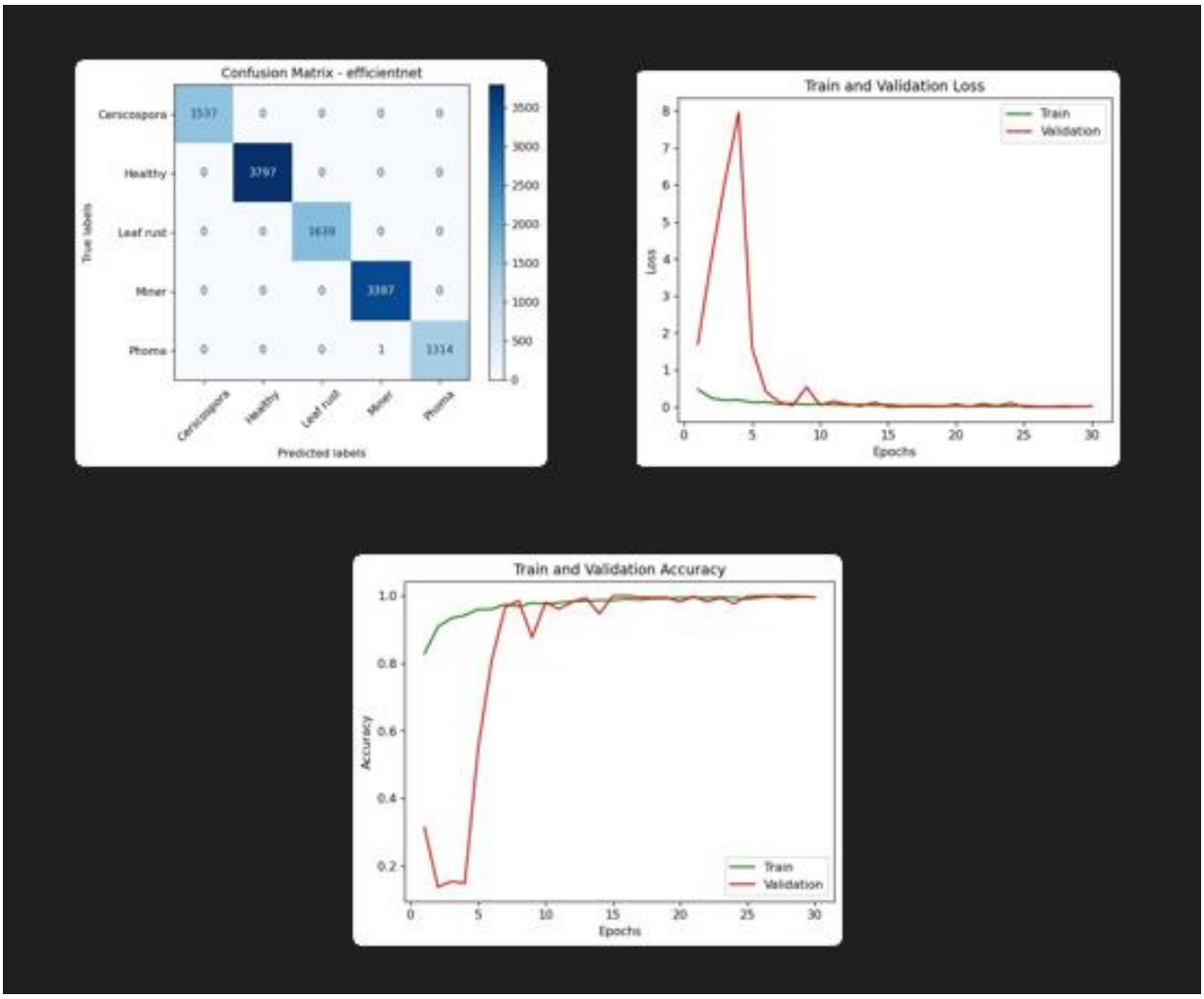


Fig 19. Training and Validation curve per epoch run for EfficientNetB0

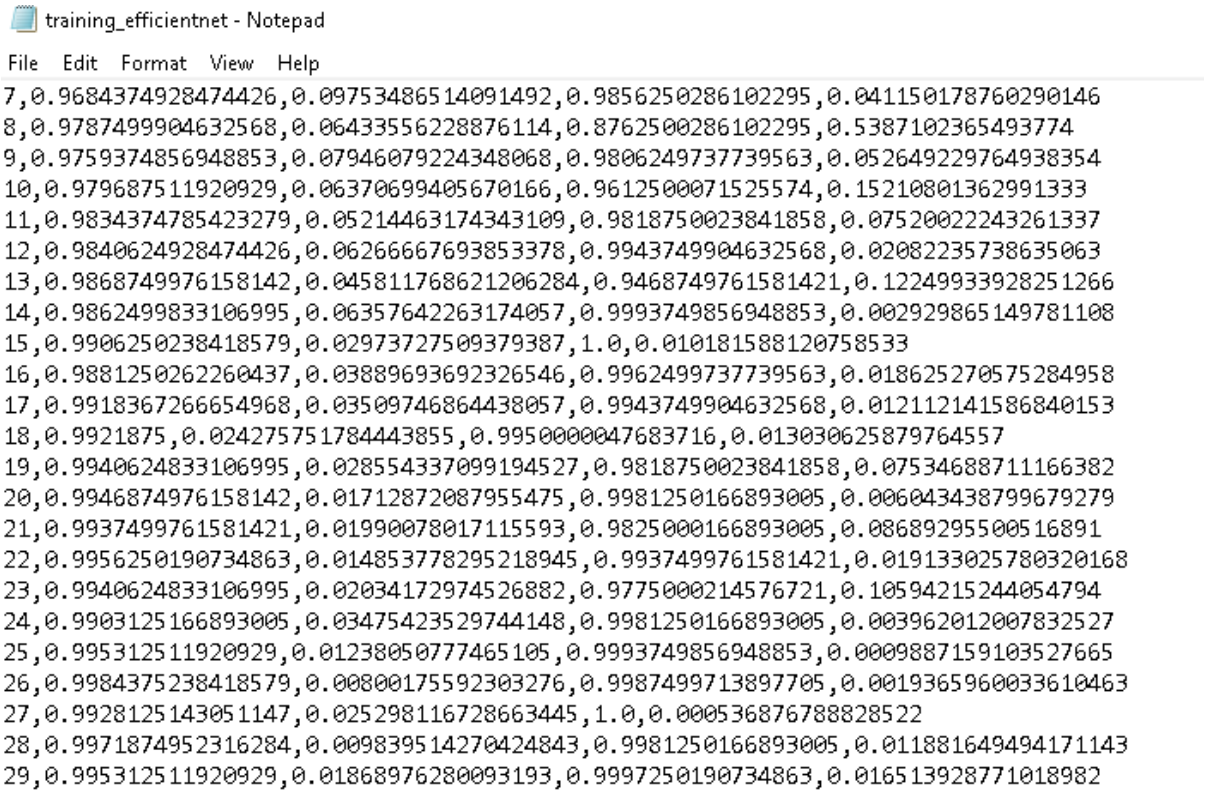


Fig 20. Model training progress for every epoch run saved as a .txt file for EfficientNetB0