National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ritika Pramod Chendvenkar
Student ID: x19199473

School of Computing
National College of Ireland

Supervisor: Prof. Jorge Basilio

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Ritika Pramod Chendvenkar
Student ID:	x19199473
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Prof. Jorge Basilio
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	1031
Page Count:	14

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:	
Date:	21st September 2021

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

Ritika Pramod Chendvenkar
x19199473

1 Introduction

This configuration manual introduces the software and hardware requirements along with the details of programming codes written for model implementation in research project: "Identification and classification of leaf pests within the Indonesian Mango farms using Machine Learning".

2 System Configuration

2.1 Hardware Specifications

Table 1 shows the system hardware specifications on which the research was undertaken.

Table 1: Hardware Specifications

RAM	8 GB
Processor	Intel(R) Core(TM) i5-8300H
Speed	2.30 GHz
Operating System	Windows 10, 64 Bit
Storage	1 TB HDD
GPU	NVIDIA GeForce GTX1650

2.2 Software Specifications

- **Jupyter Notebook from Anaconda Distribution:**
  Anaconda Navigator is an open-source desktop graphical user interface (GUI) included in Anaconda distribution. It supports Jupyter Notebooks which is very helpful in implementation and execution of machine learning models on research data. Version 6.0.3 of Jupyter notebook was used for the implementation of this project which including all the stages namely, data augmentation, feature extraction, building machine learning models and evaluation of the models.

- **Streamlit:**
  Streamlit is an open-source web-based app framework that is used to create machine learning and data science-based apps using Python. It is easy to use, and no frontend development knowledge is needed. For the last stage of this project, we have used streamlit to build a webpage which allows users to upload the image of the infected leaf and the app tells the user what kind of pest it is.

- **GitHub:**
  GitHub is used to create repositories and store the model file for the webapp to run. Streamlit simply uses the GitHub repository to build the app. The model file and the built python app is pushed into the GitHub repository using GitBash.

- **Microsoft Excel 2019:**
  It was used for creating quick exploratory plots to obtain insights.

3 Development of Project

The major chunk of the codebase of this research work uses Python programming language. Majority of the stages including pulling the data, data augmentation, feature extraction, model implementation, evaluation and even development of the webapp, all this has been done using python language. The primary libraries used were Keras, TensorFlow, Sk-Learn (Scikit-Learn) along with numpy, pandas, matplotlib.

3.1 Data Preparation

The dataset downloaded from Mendeley data repository has been pulled into Jupyter notebook. The sections that follow below shed light on the end-to-end steps involved in the implementation of this research work. The original dataset consists of 510 images, which are not enough to train a machine learning model. The model may tend to overfit with such low number of samples in the training data. To address this issue, we have made use of data augmentation techniques to add variations in the images.

3.2 Data Augmentation

Augmentation is performed using Augmentor package in python. It involves creating a pipeline wherein each image is modified accordingly. The code for creating a pipeline to perform data augmentation after importing the required set of packages is shown in Figure 1. And the code snippet for the augmentation steps is shown in Figure 2.

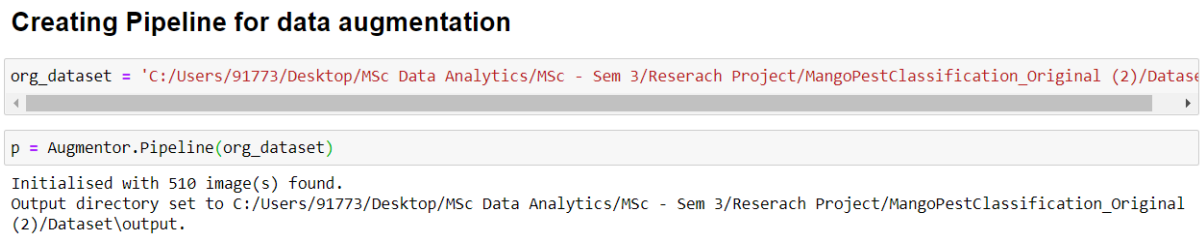

Figure 1: Pipeline creation for data augmentation

Data Augmentation

```
p.zoom_random(probability=0.5, percentage_area=0.8)
p.random_brightness(probability=0.3, min_factor=0.3, max_factor=1.2)
p.random_distortion(probability=1, grid_width=4, grid_height=4, magnitude=8)
p.rotate180(probability=0.6)
p.skew_tilt(probability=0.7, magnitude=1)
p.resize(probability=1, width=500, height=333, resample_filter='BICUBIC')
p.sample(12000)
```

```
Processing <PIL.Image.Image image mode=RGB size=500x333 at 0x1B5278333A0>: 100%|████████| 12000/12000 [04:59<00:00, 40.11 Sam
ples/s]
```

Figure 2: Data Augmentation techniques

3.3 Splitting Train and Test Folders

The next step is to split the created output folder into train and test folders containing classes. Figure 3 shows the code snippet and Figure 4 shows the snip of the folders before and after the split. The folders have been split in the ratio 80:20.

Splitting data into train and test folders

```
test_ratio = 0.20

for cls in classes_dir:
    os.makedirs(dest_dir +'train/' + cls)
    os.makedirs(dest_dir +'test/' + cls)
    src = root_dir + cls
    allFileNames = os.listdir(src)
    np.random.shuffle(allFileNames)
    train_FileNames, test_FileNames = np.split(np.array(allFileNames),
                                               [int(len(allFileNames)* (1 - test_ratio))])
    train_FileNames = [src+'/'+ name for name in train_FileNames.tolist()]
    test_FileNames = [src+'/' + name for name in test_FileNames.tolist()]
    print("******************************")
    print('Total images: ', len(allFileNames))
    print('Training: ', len(train_FileNames))
    print('Testing: ', len(test_FileNames))
    print("******************************")

    for name in train_FileNames:
        shutil.copy(name, dest_dir +'train/' + cls)

    for name in test_FileNames:
        shutil.copy(name, dest_dir +'test/' + cls)

    print("Copying Done!")
```

Figure 3: Splitting data into train and test folders

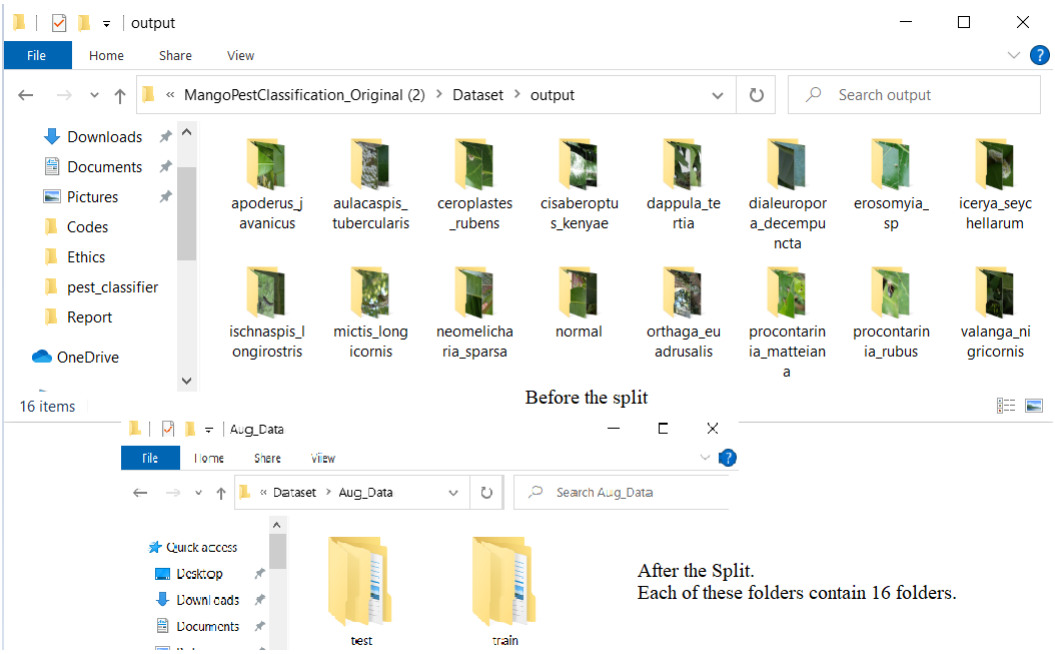

Figure 4: Before and After the split

3.4 Feature Extraction using GLCM

Gray Level Cooccurrence Matrix (GLCM) is a feature extraction technique that extracts thetexture features from the images. Using GLCM, we were successfully able to extract 80 features with 5 base texture features which are: homogeneity, contrast, energy, dissimilarity, and Angular Second Moment (ASM).

```
def GLCM(img_list_count, img_path):

    glcm_data = np.zeros((img_list_count, 80))
    class_name=[]
    count= 0

    for dir1 in tqdm(os.listdir(img_path)):
        for file in os.listdir(os.path.join(img_path, dir1)):
            image_path = os.path.join(img_path, dir1,  file)
            glcm_ = np.zeros(80)
            img = cv2.imread( image_path, cv2.COLOR_BGR2RGB)
            img = cv2.resize(img, (img_height, img_width),interpolation = cv2.INTER_AREA)

            img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            image = img_as_uint(img)
            image =  image.astype(np.uint8)
            glcm = greycomatrix(image, [1, 2, 3, 4], [0, np.pi/8,np.pi/4, 3*np.pi/8], 256, symmetric=True, normed=True)
            glcm_[:16]   = greycoprops(glcm, 'contrast').ravel()
            glcm_[16:32] = greycoprops(glcm, 'dissimilarity').ravel()
            glcm_[32:48] = greycoprops(glcm, 'homogeneity').ravel()
            glcm_[48:64] = greycoprops(glcm, 'energy').ravel()
            glcm_[64:80] = greycoprops(glcm, 'ASM').ravel()
            glcm_data[count,:] = glcm_
            count = count+1
            class_name.append(dir1)

    return pd.DataFrame(glcm_data), class_name
```

```
train_img_df, train_labels = GLCM(train_image_count,train_dir)
```

```
100%|██████████| 16/16 [09:30<00:00, 35.68s/it]
```

```
test_img_df, test_labels = GLCM(test_image_count,test_dir)
```

```
100%|██████████| 16/16 [02:21<00:00,  8.82s/it]
```

Figure 5: Code for GLCM feature extraction

```
# Checking the features generated by GLCM
train_img_df.head()
```

	0	1	2	3	4	5	6	7	8	9	...	70	71	72
0	181.326433	181.326433	362.467474	207.252202	557.493464	688.766251	362.467474	728.969781	960.987698	1051.694093	...	0.000141	0.000105	0.000089
1	320.511651	320.511651	432.445414	180.758388	834.983711	880.479881	432.445414	702.790409	1234.837023	1253.084329	...	0.000250	0.000200	0.000159
2	209.602298	209.602298	308.291158	140.858664	573.155345	624.927787	308.291158	545.866541	897.003353	923.118235	...	0.000220	0.000174	0.000142
3	284.582239	284.582239	393.099922	169.146461	779.931749	825.883852	393.099922	668.665354	1196.079044	1208.447254	...	0.000188	0.000152	0.000122
4	252.702394	252.702394	277.061192	62.282311	649.315738	650.783905	277.061192	367.665091	1025.390615	1016.298825	...	0.000197	0.000168	0.000113

5 rows × 80 columns

```
print(train_img_df.shape)
print(len(train_labels))
print(test_img_df.shape)
print(len(test_labels))
```

```
(9592, 80)
9592
(2408, 80)
2408
```

Figure 6: Extracted GLCM features

3.5 Feature Extraction / Dimensionality Reduction using PCA

The other technique used is the Principal Component Analysis (PCA), which is also a dimensionality reduction method. These 80 GLCM features are then subject to dimensionality reduction, in which we extract the most relevant features out of the 80, closest to the highest power of 2. So, as a part of this, we have outputted 64 of the most relevant texture features.

```
## Feature selection using PCA
from sklearn.decomposition import PCA

pca = PCA(n_components = 64)

train_img_pca = pca.fit_transform(train_img_df)
test_img_pca = pca.transform(test_img_df)
```

Figure 7: Code for PCA feature extraction

3.6 Splitting Train data into Train and Validation

We then split the train data into train and validation sets in the ratio 80:20.

```
# Splitting Data into Train, Val and Test Sets

X_train, X_val, y_train, y_val = train_test_split(train_img_pca,train_labels, test_size = 0.2,stratify = train_labels)

print('length X_train:', len(X_train))
print('length y_train:', len(y_train))

print('length X_val:', len(X_val))
print('length y_val:', len(y_val))

print('length X_test:',  len(test_img_pca))
print('length y_test:', len(test_labels))
```

```
length X_train: 7673
length y_train: 7673
length X_val: 1919
length y_val: 1919
length X_test: 2408
length y_test: 2408
```

Figure 8: Splitting into train and validation set

3.7 Model Implementation

This section shows the end-to-end steps involved in building and evaluating the machine learning models.

3.7.1 Importing the required libraries

As shown in figure 9, all the required libraries are imported.

```
# importing all the required Libraries
from collections import Counter
import cv2
import os
import glob
import skimage
import numpy as np
import regex as re
import pandas as pd
import seaborn as sns
from tqdm import tqdm
from PIL import Image
from os import listdir
import matplotlib.pyplot as plt
from tqdm import tqdm
from skimage.transform import resize
from collections import Counter
from sklearn.preprocessing import OneHotEncoder,LabelBinarizer
from numpy import save
from numpy import load

sns.set()

from sklearn.svm import SVC # SVC
from sklearn import metrics
from sklearn.utils import shuffle
from xgboost import XGBClassifier # XGBClassifier
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.utils import compute_class_weight
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split

import tensorflow  as tf
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.metrics import AUC
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.applications.vgg16 import VGG16 # VGG16
from tensorflow.keras.layers import Input, Dense, Dropout, BatchNormalization, Flatten, Activation, GlobalAveragePooling2D,Conv2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

from skimage import img_as_float, img_as_uint, img_as_int
from skimage.feature import greycomatrix, greycoprops
```

Figure 9: Importing required libraries

3.7.2 SVM with GLCM and PCA

Support Vector Machine was run and a combination of both, GLCM and PCA is used for feature extraction. The accuracy of the model was found to be 33.8%, increasing to 42.9% after hyperparameter tuning, which was still very low.

```
# train the model on train set
model = SVC()
model.fit(X_train, y_train)

# print validation prediction results
print()
print('--------- Validation Results --------------------')
val_predictions = model.predict(X_val)
print_performance_metrics(y_val, val_predictions)
print()
print()
print()

# print test prediction results
print('--------- Test Results --------------------')
test_predictions = model.predict(test_img_pca)
print_performance_metrics(test_labels, test_predictions)
print()
```

Figure 10: Implementation of SVM with GLCM and PCA

```
Accuracy: 0.338
Precision: 0.2977
Recall: 0.338
F1 Score: 0.2859
Cohen Kappa Score: 0.258
Matthews Corrcoef: 0.2633
                Classification Report:
                           precision    recall  f1-score   support

      apoderus_javanicus       0.22      0.41      0.29       293
  aulacaspis_tubercularis       0.00      0.00      0.00        62
      ceroplastes_rubens       0.00      0.00      0.00        57
     cisaberoptus_kenyae       0.00      0.00      0.00        93
          dappula_tertia       0.19      0.19      0.19       174
dialeuropora_decempuncta       0.22      0.47      0.30       133
             erosomyia_sp       0.00      0.00      0.00        60
      icerya_seychellarum       0.28      0.29      0.28       136
 ischnaspis_longirostris       0.00      0.00      0.00        35
       mictis_longicornis       0.60      0.86      0.71       402
    neomelicharia_sparsa       0.28      0.43      0.34       176
                  normal       0.41      0.19      0.26       165
     orthaga_euadrusalis       0.52      0.11      0.18       103
 procontarinia_matteiana       0.00      0.00      0.00       133
     procontarinia_rubus       0.50      0.01      0.02       121
     valanga_nigricornis       0.30      0.35      0.32       265

                accuracy                           0.34      2408
               macro avg       0.22      0.21      0.18      2408
            weighted avg       0.30      0.34      0.29      2408
```

Figure 11: Test Results of Implementation of SVM with GLCM and PCA

```
## Trying GridSearch CV to increase the accuracy
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(X_train, y_train)
# ran this for 15 minutes, manually observed the best and then fitted the model with those hyperparameters
```

Figure 12: GridSearchCV for Hyperparameter Tuning

3.7.3 SVM with GLCM only

Since the combination of GLCM and PCA did not give good accuracy, it was decided to only move forward with GLCM based feature extraction technique. The implementation is similar to 3.6.2, but without using PCA. The accuracy of the model was found to be 30.98% which was still lower than that with GLCM and PCA, but on performing hyperparameter tuning, the test accuracy increased to 43.36%.

```
# train the model on train set
model = SVC()
model.fit(X_train, y_train)

# print validation prediction results
print()
print('--------- Validation Results -------------------')
val_predictions = model.predict(X_val)
print_performance_metrics(y_val, val_predictions)
print()
print()
print()

# print test prediction results
print('--------- Test Results -------------------')
test_predictions = model.predict(test_img_df)
print_performance_metrics(test_labels, test_predictions)
print()
```

```
--------- Validation Results -------------------
Accuracy: 0.2824
Precision: 0.2047
Recall: 0.2824
F1 Score: 0.2193
Cohen Kappa Score: 0.1931
Matthews Corrcoef: 0.1998
                Classification Report:
                              precision    recall  f1-score   support

      apoderus_javanicus       0.16      0.28      0.20       234
 aulacaspis_tubercularis       0.00      0.00      0.00        49
      ceroplastes_rubens       0.00      0.00      0.00        45
     cisaberoptus_kenyae       0.00      0.00      0.00        74
          dappula_tertia       0.09      0.05      0.06       139
dialeuropora_decempuncta       0.18      0.62      0.28       106
            erosomyia_sp       0.00      0.00      0.00        47
     icerya_seychellarum       0.00      0.00      0.00       109
 ischnaspis_longirostris       0.00      0.00      0.00        28
      mictis_longicornis       0.51      0.81      0.62       321
    neomelicharia_sparsa       0.22      0.30      0.25       140
                  normal       0.25      0.09      0.13       131
     orthaga_euadrusalis       0.47      0.10      0.16        82
 procontarinia_matteiana       0.00      0.00      0.00       106
     procontarinia_rubus       0.00      0.00      0.00        96
     valanga_nigricornis       0.28      0.38      0.32       212

                accuracy                           0.28      1919
               macro avg       0.13      0.16      0.13      1919
            weighted avg       0.20      0.28      0.22      1919
```

Figure 13: Results of Implementation of SVM with GLCM only

```python
# train the model on train set
model = SVC(C= 1, gamma = 0.0001, kernel='rbf')
model.fit(X_train, y_train)

# print validation prediction results
print()
print('--------- HyperParameter Tuned Validation Results -------------------')
val_predictions = model.predict(X_val)
print_performance_metrics(y_val, val_predictions)
print()
print()
print()

# print test prediction results
print('--------- HyperParameter Tuned Test Results -------------------')
test_predictions = model.predict(test_img_df)
print_performance_metrics(test_labels, test_predictions)
print()
```

```
--------- HyperParameter Tuned Validation Results -------------------
Accuracy: 0.4336
Precision: 0.4615
Recall: 0.4336
F1 Score: 0.4215
Cohen Kappa Score: 0.3723
Matthews Corrcoef: 0.3751
                Classification Report:
                              precision    recall  f1-score   support

      apoderus_javanicus       0.36      0.47      0.41       234
 aulacaspis_tuberculariss       0.48      0.24      0.32        49
      ceroplastes_rubens       0.48      0.29      0.36        45
     cisaberoptus_kenyae       0.52      0.15      0.23        74
          dappula_tertia       0.34      0.29      0.31       139
dialeuropora_decempuncta       0.41      0.58      0.48       106
            erosomyia_sp       0.39      0.23      0.29        47
     icerya_seychellarum       0.34      0.49      0.40       109
 ischnaspis_longirostris       0.72      0.46      0.57        28
      mictis_longicornis       0.72      0.79      0.76       321
    neomelicharia_sparsa       0.37      0.34      0.35       140
                  normal       0.41      0.24      0.30       131
     orthaga_euadrusalis       0.75      0.18      0.29        82
 procontarinia_matteiana       0.31      0.31      0.31       106
     procontarinia_rubus       0.56      0.19      0.28        96
     valanga_nigricornis       0.30      0.52      0.38       212

                accuracy                           0.43      1919
               macro avg       0.47      0.36      0.38      1919
            weighted avg       0.46      0.43      0.42      1919
```

Figure 14: Results of Hyperparameter tuned Implementation of SVM with GLCM only

3.7.4 XGBoost and CatBoost

As the accuracy of the SVM classifier was lower, we decided to use more robust classifiers based on decision trees. We have made use of XGBoost classifier along with GLCM features to classify the images. The XGBoost classifier gave a test accuracy of 64.66%. CatBoost model gave a test accuracy of 65.82%, which was overall the best among all the classifiers.

```
# train the model on train set
model = XGBClassifier()
model.fit(X_train, y_train)

# print validation prediction results
print()
print('--------- XGBoost Validation Results -------------------')
val_predictions = model.predict(X_val)
print_performance_metrics(y_val, val_predictions)
print()
print()
print()

# print test prediction results
print('--------- XGBoost Test Results -------------------')
test_predictions = model.predict(test_img_df)
print_performance_metrics(test_labels, test_predictions)
print()
```

```
C:\Users\91773\anaconda3\lib\site-packages\xgboost\sklearn.py:892: U
precated and will be removed in a future release. To remove this war
lse when constructing XGBClassifier object; and 2) Encode your label
class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)

[17:38:11] WARNING: C:/Users/Administrator/workspace/xgboost-win64_r
0, the default evaluation metric used with the objective 'multi:soft
et eval_metric if you'd like to restore the old behavior.

--------- XGBoost Validation Results -------------------
Accuracy: 0.6524
Precision: 0.6504
Recall: 0.6524
F1 Score: 0.6482
Cohen Kappa Score: 0.6174
Matthews Corrcoef: 0.6178
```

Figure 15: Results of Implementation of XGBoost with GLCM only

```
# train the model on train set
model = CatBoostClassifier()

model.fit(X_train, y_train)

# print validation prediction results
print()
print('--------- CatBoost Validation Results -------------------')
val_predictions = model.predict(X_val)
print_performance_metrics(y_val, val_predictions)
print()
print()
print()

# print test prediction results
print('--------- CatBoost Test Results -------------------')
test_predictions = model.predict(test_img_df)
print_performance_metrics(test_labels, test_predictions)
print()
```

```
--------- CatBoost Validation Results -------------------
Accuracy: 0.6566
Precision: 0.6551
Recall: 0.6566
F1 Score: 0.6511
Cohen Kappa Score: 0.6224
Matthews Corrcoef: 0.6229
              Classification Report:
                           precision    recall  f1-score   support

      apoderus_javanicus       0.61      0.62      0.62       234
 aulacaspis_tuberculariS       0.82      0.63      0.71        49
     ceroplastes_rubens        0.82      0.60      0.69        45
    cisaberoptus_kenyae        0.59      0.41      0.48        74
         dappula_tertia        0.49      0.45      0.47       139
dialeuropora_decempuncta       0.69      0.80      0.74       106
           erosomyia_sp        0.47      0.32      0.38        47
     icerya_seychellarum       0.65      0.81      0.72       109
 ischnaspis_longirostris       0.61      0.61      0.61        28
```

Figure 16: Results of Implementation of CatBoost with GLCM only

3.7.5 Convolutional Neural Network (CNN)

As discussed in Wongbongkotpaisan and Phumeechanya (2021), CNN has proved to perform well with augmented data in classifying leaf diseases. They proposed a simple CNN architecture for classifying left pests and their model gave an accuracy of around 95.65%. We have used the same CNN architecture for our research. Convolutional Neural Network with 3 convolutional layers, 2 max pooling layers, 1 input and 1 output layer was applied on the data set. obtained from feature extraction with 5 epochs. The model gave a training accuracy of 92.7% for 5 epochs. And a test accuracy of 72.05%.

```
num_classes = 16

model = tf.keras.Sequential([
  tf.keras.layers.experimental.preprocessing.Rescaling(1./255),
  tf.keras.layers.Conv2D(32, (3,3) , activation='relu'),
  tf.keras.layers.MaxPooling2D((2,2)),
  tf.keras.layers.Conv2D(64, (3,3) , activation='relu'),
  tf.keras.layers.MaxPooling2D((2,2)),
  tf.keras.layers.Conv2D(128, (3,3) , activation='relu'),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(num_classes)
])
```

Figure 17: CNN Model Architecture

```
model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics='accuracy')
```

```
history = model.fit(train_ds,epochs=5,class_weight=class_weights)
```

```
WARNING:tensorflow:From C:\Users\91773\anaconda3\lib\site-packages\tensorflow\python\ops\array_ops.py:5043: calling gather (fro
m tensorflow.python.ops.array_ops) with validate_indices is deprecated and will be removed in a future version.
Instructions for updating:
The `validate_indices` argument has no effect. Indices are always validated on CPU and never validated on GPU.
Epoch 1/5
150/150 [==============================] - 299s 2s/step - loss: 2.0734 - accuracy: 0.3707
Epoch 2/5
150/150 [==============================] - 295s 2s/step - loss: 0.9841 - accuracy: 0.6799
Epoch 3/5
150/150 [==============================] - 293s 2s/step - loss: 0.5121 - accuracy: 0.8215
Epoch 4/5
150/150 [==============================] - 291s 2s/step - loss: 0.3390 - accuracy: 0.8832
Epoch 5/5
150/150 [==============================] - 293s 2s/step - loss: 0.2021 - accuracy: 0.9270
```

Figure 18: CNN Model Training

```
def print_performance_metrics(y_test, max_y_pred_test):
    """
        parameters
        ----------
        y_test : actual label (must be in non-one hot encoded form)
        y_pred_test : predicted labels (must be in non-one hot encoded form, common output of predict methods of classifers)

        returns
        -------
        prints the accuracy, precision, recall, F1 score, ROC AUC score, Cohen Kappa Score, Matthews Corrcoef and classification

    """
    print('Accuracy:', np.round(metrics.accuracy_score(y_test, max_y_pred_test),4))
    print('Precision:', np.round(metrics.precision_score(y_test, max_y_pred_test, average='weighted'),4))
    print('Recall:', np.round(metrics.recall_score(y_test, max_y_pred_test, average='weighted'),4))
    print('F1 Score:', np.round(metrics.f1_score(y_test, max_y_pred_test, average='weighted'),4))
    print('Cohen Kappa Score:', np.round(metrics.cohen_kappa_score(y_test, max_y_pred_test),4))
    print('Matthews Corrcoef:', np.round(metrics.matthews_corrcoef(y_test, max_y_pred_test),4))
    print('\t\tClassification Report:\n', metrics.classification_report(y_test, max_y_pred_test))
```

```
print('------------------------ Test Set Scores ------------------------')
print_performance_metrics(y_ground_truth_test_list, y_final_predictions)
```

```
------------------------ Test Set Scores ------------------------
Accuracy: 0.7205
Precision: 0.7378
Recall: 0.7205
F1 Score: 0.7211
Cohen Kappa Score: 0.6933
Matthews Corrcoef: 0.6956
                Classification Report:
               precision    recall  f1-score   support

           0       0.53      0.82      0.65       293
           1       0.87      0.76      0.81        62
           2       0.89      0.82      0.85        57
           3       0.63      0.59      0.61        93
           4       0.60      0.47      0.53       174
           5       0.94      0.88      0.91       133
           6       0.62      0.72      0.67        60
           7       0.90      0.83      0.87       136
           8       0.78      0.80      0.79        35
           9       0.87      0.85      0.86       402
          10       0.85      0.73      0.79       176
          11       0.82      0.72      0.76       165
          12       0.60      0.84      0.70       103
          13       0.63      0.67      0.65       133
          14       0.56      0.50      0.52       121
          15       0.73      0.52      0.61       265
```

Figure 19: CNN Model Testing

3.8 Addons in CNN for Webpage development

Assigning class weights for each class to develop the webpage.

```
from sklearn.utils import class_weight


class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(train_labels),
                                                  train_labels)

class_weights = dict(enumerate(class_weights))
```

```
C:\Users\91773\anaconda3\lib\site-packages\sklearn\utils\validation.py:68: FutureWarning: Pass cla
'aulacaspis_tubercularis' 'ceroplastes_rubens'
 'cisaberoptus_kenyae' 'dappula_tertia' 'dialeuropora_decempuncta'
 'erosomyia_sp' 'icerya_seychellarum' 'ischnaspis_longirostris'
 'mictis_longicornis' 'neomelicharia_sparsa' 'normal'
 'orthaga_euadrusalis' 'procontarinia_matteiana' 'procontarinia_rubus'
 'valanga_nigricornis'], y=['apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apc
s_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicu
'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_
nicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'a
rus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javani
s', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apode
javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus'
poderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_ja
cus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apc
s_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicu
'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_
nicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'a
rus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javani
s', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apoderus_javanicus', 'apode
```

```
# 0,1,2,3, are label encoded Pest varities
class_weights
```

```
{0: 0.5123931623931623,
 1: 2.446938775510204,
 2: 2.640969162995595,
 3: 1.6246612466124661,
 4: 0.8663294797687862,
 5: 1.1332703213610587,
 6: 2.529535864978903,
 7: 1.1020220588235294,
 8: 4.344202898550725,
 9: 0.3735202492211838,
 10: 0.855206847360913,
 11: 0.9124809741248098,
 12: 1.465770171149144 2,
 13: 1.1332703213610587,
 14: 1.2437759336099585,
 15: 0.5666351606805293}
```

Figure 20: Assigning class weights for Webpage

```python
def predict(image_path):
    classifier_model = "C:/Users/91773/Desktop/MSc Data Analytics/MSc - Sem 3/Reserach Project/Dataset/Aug_Data/cnn_with_weights_
    IMAGE_SHAPE = (224, 224,3)
    model = keras.models.load_model(classifier_model) #load the weights of model
    image = Image.open(image_path)
    test_image = image.resize((224,224))
    test_image = preprocessing.image.img_to_array(test_image)
    #test_image.shape = (224,224,3)
    test_image = np.expand_dims(test_image, axis=0)
    #test_image.shape = (1,224,224,3)

    class_names_with_indices =['apoderus_javanicus',
                               'aulacaspis_tubercularis',
                               'ceroplastes_rubens',
                               'cisaberoptus_kenyae',
                               'dappula_tertia',
                               'dialeuropora_decempuncta',
                               'erosomyia_sp',
                               'icerya_seychellarum',
                               'ischnaspis_longirostris',
                               'mictis_longicornis',
                               'neomelicharia_sparsa',
                               'normal',
                               'orthaga_euadrusalis',
                               'procontarinia_matteiana',
                               'procontarinia_rubus',
                               'valanga_nigricornis'
                               ]

    predictions = model.predict(test_image) #using model to predict image
    print(predictions)
    print()
    scores = tf.nn.softmax(predictions) # np.argamx
    scores = scores.numpy()
    print(np.argmax(scores))

    result = f"{class_names_with_indices[np.argmax(scores)]} with a { (100 * np.max(scores)).round(2) } % confidence."
    return result
```

Figure 21: Predict class function for Webpage

3.9 GitHub, Streamlit connectivity

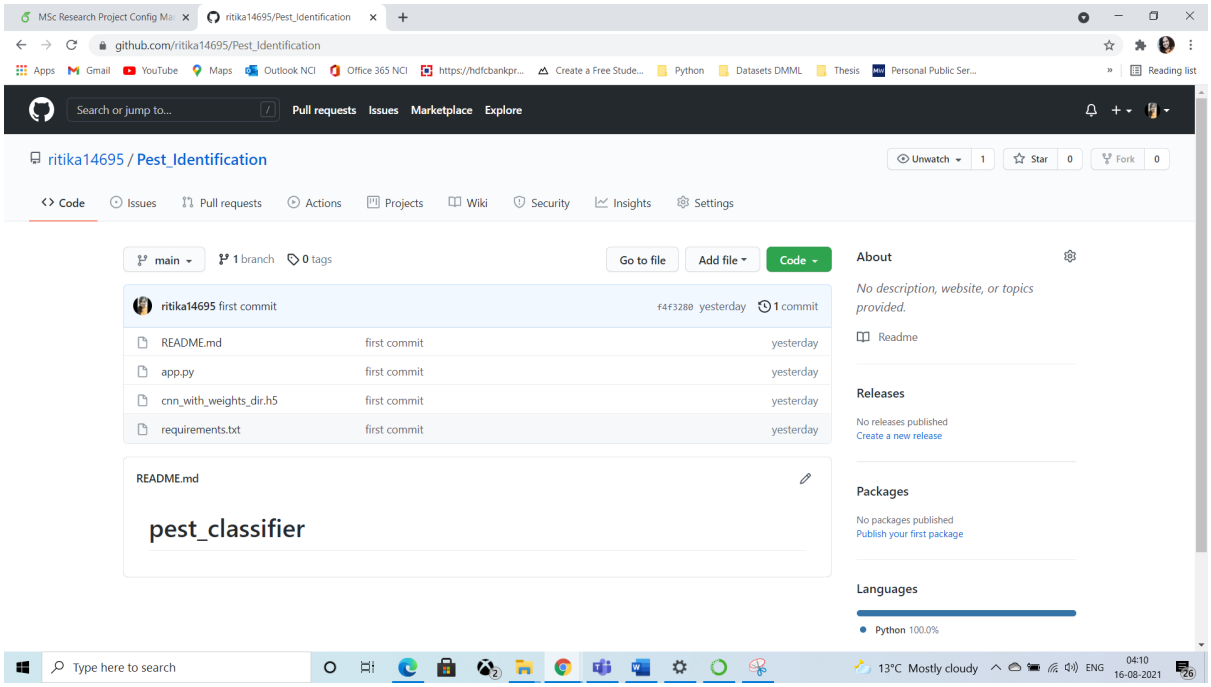

Figure 22: GitHub Public repository

```python
def main():
    file_uploaded = st.file_uploader("Upload Image File", type=["png","jpg","jpeg"])
    class_btn = st.button("Classify")
    if file_uploaded is not None:
        image = Image.open(file_uploaded)
        st.image(image, caption='Uploaded Image', use_column_width=True)

    if class_btn:
        if file_uploaded is None:
            st.write("Invalid command, please upload an image")
        else:
            with st.spinner('Model working....'):
                plt.imshow(image)
                plt.axis("off")
                predictions = predict(image)
                time.sleep(1)
                st.success('Classified')
                st.write(predictions)
                #st.pyplot(fig)
```

Figure 23: Python main() function for webpage

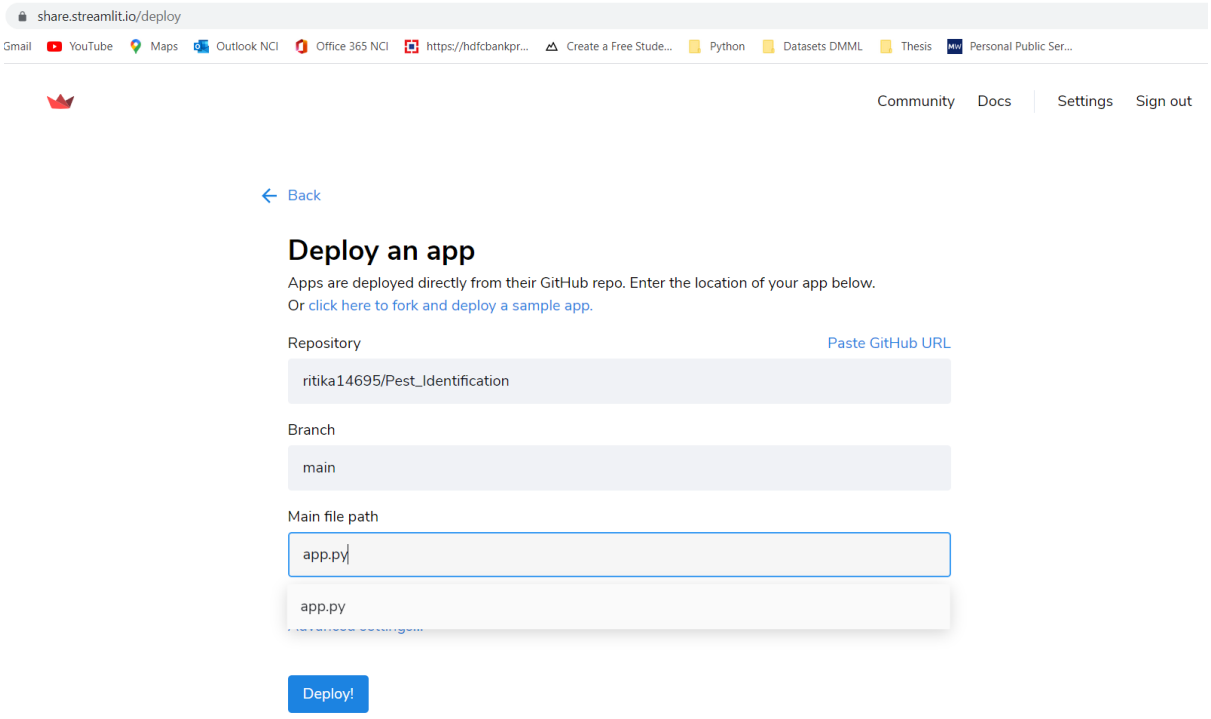

Figure 24: Deploying app using GitHub repository

References

Wongbongkotpaisan, J. and Phumeechanya, S. (2021). Plant leaf disease classification using local-based image augmentation and convolutional neural network, *2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 1023–1027.