National College of Ireland

Configuration Manual

MSc Research Project
Masters of Science in Data Analytics

Steve Mendonca
Student ID: 20226691

School of Computing
National College of Ireland

Supervisor: Dr. Abid Yaqoob

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Steve Mendonca
Student ID:	20226691
Programme:	Masters of Science in Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Abid Yaqoob
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	786
Page Count:	8

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:	15th December 2022

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

Steve Mendonca
20226691

1 Introduction

The Configuration Manual gives a detailed explanation of all the processes and technologies used during the research project. It is important to have a detailed manual that gives information about the project as this gives in information about the background of the research and an insight about how the technology works. It also gives intricate details that cannot be found in the report as the configuration manual is purely technical. It gives a step by step walkthrough of the project and also consists of the results obtained during this project.

2 Environment Specifications

This section gives information about the environment necessary for the successful run of the project.

2.1 Hardware Specifications

The hardware specifications that were used during the research are given below:

- Operating System: Windows 10
- Processor: AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
- RAM: 8GB

2.2 Software Specifications

The software specifications required during the course of this research is given below:

- Python 3.9.7[1]
- Anaconda Navigator 2.3.2[2]
- Jupyter Notebook[3]

[1]https://www.python.org/
[2]https://www.anaconda.com/
[3]https://jupyter.org/

3 Libraries and Packages

Various packages and Libraries used for the research are seen in Figure 1. The use of the libraries are given below.

```
import tensorflow
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten, Conv2D, MaxPooling2D, BatchNormalization, Dropout
from tensorflow.keras.models import Model
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.applications.densenet import DenseNet121
from tensorflow.keras.preprocessing import image
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator,load_img
from tensorflow.keras.models import Sequential
import cv2
import keras
from skimage.transform import rotate
from skimage.util import random_noise
from skimage.exposure import adjust_gamma
from skimage.filters import gaussian
import numpy as np
import glob, os
from sklearn.metrics import f1_score, confusion_matrix, accuracy_score, classification_report
from tensorflow.python.framework.ops import tensor_id
```

Figure 1: Libraries code snippet

- **tensorflow**[4]**:** This library is used for data modeling and model building. All layers can be called using this package.
- **matplotlib**[5]**:** This library is used for visualizations and data insight. The line graphs in this research are made using this library.
- **cv2**[6]**:** This library resolves all problems regarding computer vision problems.
- **keras**[7]**:** Keras is an API that helps easily call the tensorflow functions.
- **skimage**[8]**:** This library helps make edits on images and change the outline of them. It was used for image augmentation in this research.
- **numpy**[9]**:** This library is used to work with arrays. Apart from arrays it also works with matrices.

4 Dataset Description

The dataset used for the research is a cotton plant disease dataset[10]. The images are taken from a field in Asia with a growth of cotton crops. This dataset consists of four classes that are, *Healthy plants, Diseased plants, Healthy leaves and Diseased leaves*. There are a total of 2310 images in this dataset. The dataset is split into three groups that are, train, test and val. This split is made prior to the download of data and does not need to be done during implementation.

[4]https://www.tensorflow.org/
[5]https://matplotlib.org/
[6]https://pypi.org/project/opencv-python/
[7]https://keras.io/
[8]https://scikit-image.org/docs/stable/api/skimage.html
[9]https://numpy.org/
[10]https://www.kaggle.com/datasets/janmejaybhoi/cotton-disease-dataset

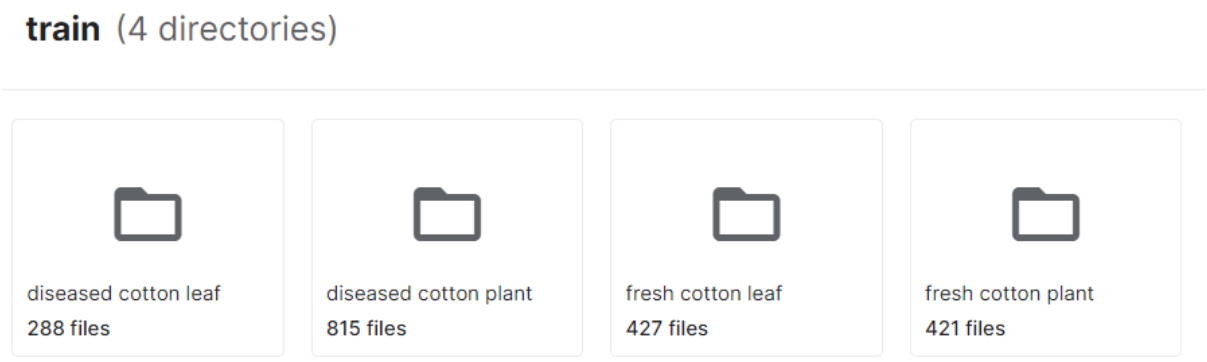

Figure 2: Data Directory

5 Data Augmentation

Since the number of images are low, the problem of overfitting could arise. Augmentation is done to avoid this problem. The code snippet is given in Figure 3.

```
#Augmentation.
for filename in glob.glob('train/*/aug*'):
    os.remove(filename)
#creates 5 more images for each image and writes the image to the respective folders.
#TF transforms was not used.
files = glob.glob('train/*/*.jpg',recursive=True)
for i in files:
    aug = cv2.imread(i)
    aug1 = adjust_gamma(aug, gamma=0.5,gain=1).astype(np.uint8)
    cv2.imwrite('train/' + i.split('/')[1] + '/aug1_' + i.split('/')[2], aug1)

    aug1 = adjust_gamma(aug, gamma=2, gain=1).astype(np.uint8)
    cv2.imwrite('train/' + i.split('/')[1] + '/aug2_' + i.split('/')[2], aug1)

    aug1 = np.fliplr(aug)
    cv2.imwrite('train/' + i.split('/')[1] + '/aug3_' + i.split('/')[2], aug1)

    aug1 = np.flipud(aug)
    cv2.imwrite('train/' + i.split('/')[1] + '/aug4_' + i.split('/')[2], aug1)

    aug1 = (random_noise(aug)*255).astype(np.uint8)
    cv2.imwrite('train/' + i.split('/')[1] + '/aug5_' + i.split('/')[2], aug1)
###TILL HERE IS AUGMENTATION
#Run only once if possible
```

Figure 3: Augmentation code snippet

The 5 types of augmentation techniques like ‘random_noise’, ‘fliplr’, etc. that are used can be seen in Figure 1.

6 Implementation

The work done for the implementation of this research is given in this section.

6.1 Defining image size

```
# re-size all the images to this
length = 256
height = 256
IMAGE_SIZE = [length,  height]
```

Figure 4: Size Definition code snippet

6.2 Model Definition

```
#Select from custom, vgg16 and densenet121
model_name = 'custom'
```

Figure 5: Model Definition code snippet

6.3 Data Loading

```
train_path = '/train'
valid_path = '/val'
classes = glob.glob('train/*')

#Data Loader
train_datagen = ImageDataGenerator()
val_datagen = ImageDataGenerator()
training_set = train_datagen.flow_from_directory('train',
                                                 target_size = (length,  height),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
val_set = val_datagen.flow_from_directory('val',
                                          target_size = (length,  height),
                                          batch_size = 32,
                                          class_mode = 'categorical')
```

Figure 6: Data Loading code snippet

7 Model Building

This section provides the code and outputs of the models that were used during the research.

7.1 VGG16

Figure 7 highlights the code used to build the VGG16 model. The weights from the 'imagenet' dataset is used for this research. The output of the was then flattened and put in an fully connected output layer.

```python
if model_name == "vgg16":
    """ vgg16
    """
    model = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
    for layer in model.layers:
        layer.trainable = False
    x = Flatten()(model.output)
    prediction = Dense(len(classes), activation='softmax')(x)
    model = Model(inputs=model.input, outputs=prediction)
    model.compile(
        loss='categorical_crossentropy',
        optimizer='Adam',
        metrics=['accuracy']
    )
    check_point = tensorflow.keras.callbacks.ModelCheckpoint(f'model_{model_name}.h5', monitor='accuracy',save_best_only=True)
```

Figure 7: VGG16 code snippet

```
Accuracy:0.9683794466403162
F1 Score:0.9683794466403162
Confusion Matrix:
[[41  0  2  0]
 [ 0 75  0  3]
 [ 1  0 65  0]
 [ 0  1  1 64]]
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.95      0.96        43
           1       0.99      0.96      0.97        78
           2       0.96      0.98      0.97        66
           3       0.96      0.97      0.96        66

    accuracy                           0.97       253
   macro avg       0.97      0.97      0.97       253
weighted avg       0.97      0.97      0.97       253
```

Figure 8: VGG16 Confusion Matrix snippet

Figure 8 shows the scores of the VGG16 model. The accuracy, precision, recall and F1 score can be seen that the VGG16 model produced.

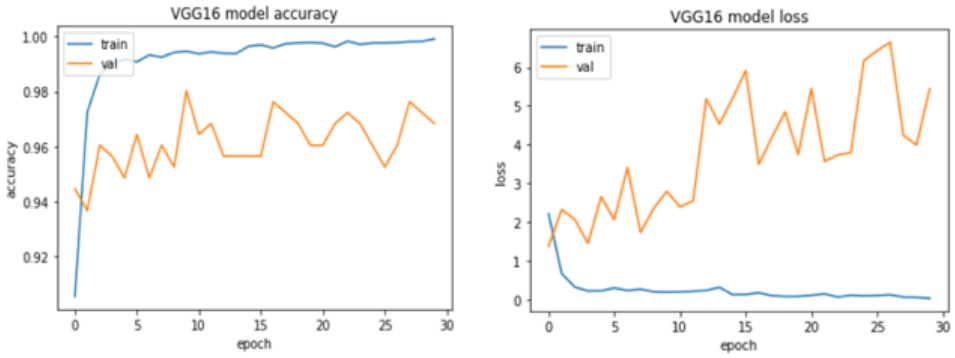

Figure 9: VGG16 Plots snippet

Figure 9 shows the Plot history of the VGG16 model. The accuracy and model loss over the course of 30 epochs can be seen.

7.2 DenseNet121

The code snippet can be seen in figure 10 that was used for the DenseNet121 model. Similar to the VGG16 model, the output is first flattened and then put through a fully connected dense layer with Softmax activation function.

Figure 11 shows the Scores and confusion matrix by the DenseNet121 model.

```python
elif model_name == "densenet121":
    """ Densenet121
    """
    model = DenseNet121(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
    for layer in model.layers:
        layer.trainable = False
    x = Flatten()(model.output)
    prediction = Dense(len(classes), activation='softmax')(x)
    model = Model(inputs=model.input, outputs=prediction)
    model.compile(
        loss='categorical_crossentropy',
        optimizer='Adam',
        metrics=['accuracy']
    )
    check_point = tensorflow.keras.callbacks.ModelCheckpoint(f'model_{model_name}.h5', monitor='accuracy', save_best_only=True)
```

Figure 10: DenseNet121 code snippet

```
Accuracy:0.8063241106719368
F1 Score:0.8063241106719368
Confusion Matrix:
[[39  2  2  0]
 [ 0 78  0  0]
 [ 4  3 59  0]
 [ 0 38  0 28]]
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.91      0.91        43
           1       0.64      1.00      0.78        78
           2       0.97      0.89      0.93        66
           3       1.00      0.42      0.60        66

    accuracy                           0.81       253
   macro avg       0.88      0.81      0.80       253
weighted avg       0.87      0.81      0.79       253
```

Figure 11: DenseNet121 Confusion Matrix snippet

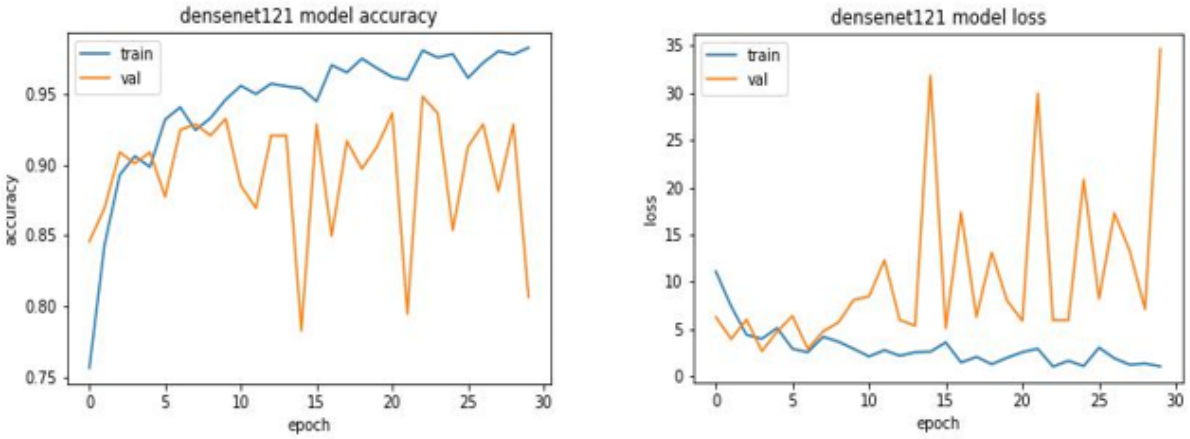

Figure 12: DenseNet121 Plots snippet

Figure 12 shows the history plots of the accuracy and model loss of the DenseNet121 model.

7.3 Custom CNN model

```
elif model_name == "custom":
    """ custom CNN
    """
    model = tensorflow.keras.models.Sequential()
    # Convolutional layer
    model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(256,256, 3)))
    # Pooling layer
    model.add(MaxPooling2D(pool_size=(2,2)))
    # Batch Normalization
    model.add(BatchNormalization())

    model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2,2)))
    model.add(BatchNormalization())

    model.add(Dropout(0.2))

    model.add(Flatten())

    model.add(Dense(128, activation='relu'))

    model.add(Dropout(0.3))
    model.compile(
        loss='categorical_crossentropy',
        optimizer='Adam',
        metrics=['accuracy']
    )
    check_point = tensorflow.keras.callbacks.ModelCheckpoint(f'model_{model_name}.h5', monitor='accuracy', save_best_only=True)
```

Figure 13: CNN code snippet

Figure 13 shows all the layers used in the custom CNN model with the parameters and activations used throughout the model.

```
Accuracy:0.9604743083003953
F1 Score:0.9604743083003953
Confusion Matrix:
[[43  0  0  0]
 [ 0 74  0  4]
 [ 2  1 63  0]
 [ 1  2  0 63]]
Classification Report:
              precision    recall  f1-score   support

           0       0.93      1.00      0.97        43
           1       0.96      0.95      0.95        78
           2       1.00      0.95      0.98        66
           3       0.94      0.95      0.95        66

    accuracy                           0.96       253
   macro avg       0.96      0.96      0.96       253
weighted avg       0.96      0.96      0.96       253
```

Figure 14: CNN Confusion Matrix snippet

Figure 14 gives the scores and confusion matrix of the custom CNN model. The accuracy, F1 score, precision and recall can be observed.

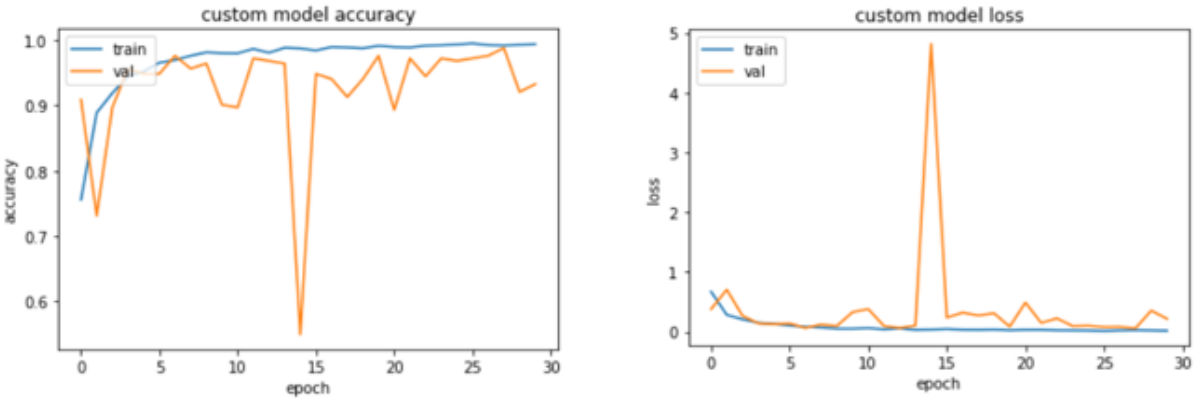

Figure 15: CNN Plots snippet

Figure 15 gives the history plots of the accuracy and model loss of the custom CNN model through the course of 30 epochs.