National College of Ireland

Configuration Manual

MSc Research Project
MSc in Data Analytics

Ria Arora
Student ID: x20231385

School of Computing
National College of Ireland

Supervisor: Prof. Prashanth Nayak

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ria Arora		
Student ID:	20231385		
Programme:	MSc in Data Analytics	**Year:**	2021-22
Module:	Research Project		
Lecturer:	Prof. Prashant Nayak		
Submission Due Date:	15-12-2022		
Project Title:	Cotton Plant Disease Prediction Using ResNet50		
Word Count:	1596	**Page Count:**	**10**

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 authors' written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ria Arora

Date: 15-12-2022

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 reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	□

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

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

Configuration Manual

Ria Arora
Student ID: x20231385

1 Introduction

This handbook is a step-by-step manual that shows all the steps performed while doing this project on Cotton Plant Disease detection using Transfer Learning. This report helps the reader to check the results and analyze the outputs. This manual may also be useful for team members who want to learn more about the capabilities and features of the project.

1.1 Project Overview

This project aims to identify cotton disease in its leaves and plants. The model that is used for it is ResNet50 architecture. Data augmentation is done by rotating the images to 15, 30, and 90 degrees both clockwise and anti-clockwise. Data augmentation is performed in the ratio 1:1, 1:2, and 1:3. Fine tuning is performed by adjusting the hyperparameters, Adam optimizer is used and the loss function is categorical_crossentropy.

2 Pre-requisites

Programming Language: Python.
Development Tools: Jupyter Notebook

3 Software Installation

This is a list of instructions for installing and setting up the Anaconda Distribution, which is a popular platform for data science and machine learning. The instructions are as follows:

1. Install and download the Anaconda Distribution by visiting the Anaconda website and following the instructions on the download page.
2. Download the Anaconda Distribution package by selecting the appropriate package for your operating system and following the instructions on the download page.
3. Install the Anaconda Distribution by following the instructions provided with the downloaded package.
4. From the Anaconda Navigator, create a new Anaconda environment by clicking the "Create" button and specifying the desired settings for the environment.
5. Establish a setting for Jupyter Notebook by opening the Anaconda Navigator and navigating to the "Environments" tab. Select the environment where you want to use Jupyter Notebook, and then click the "Install" button next to the "Jupyter" package.

6. To use Jupyter Notebook, open the Anaconda Navigator and click the "Launch" button next to the Jupyter package in the environment where you want to use it. This will open Jupyter Notebook in your web browser, where you can create and run notebook files.

Installed GPU:

```
(RP) D:\College_Material\Sem3\Research_Project\yolov7>nvcc --version
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2022 NVIDIA Corporation
Built on Wed_Jun__8_16:59:34_Pacific_Daylight_Time_2022
Cuda compilation tools, release 11.7, V11.7.99
Build cuda_11.7.r11.7/compiler.31442593_0
```

Figure 1. GPU installation

Data Augmentation is performed using Roboflow:

Outputs per training example: 1,2,3
90° Rotate: Clockwise, Counter-Clockwise
Rotation: Between -15° and +15°
Bounding Box: Rotation: Between -30° and +30°

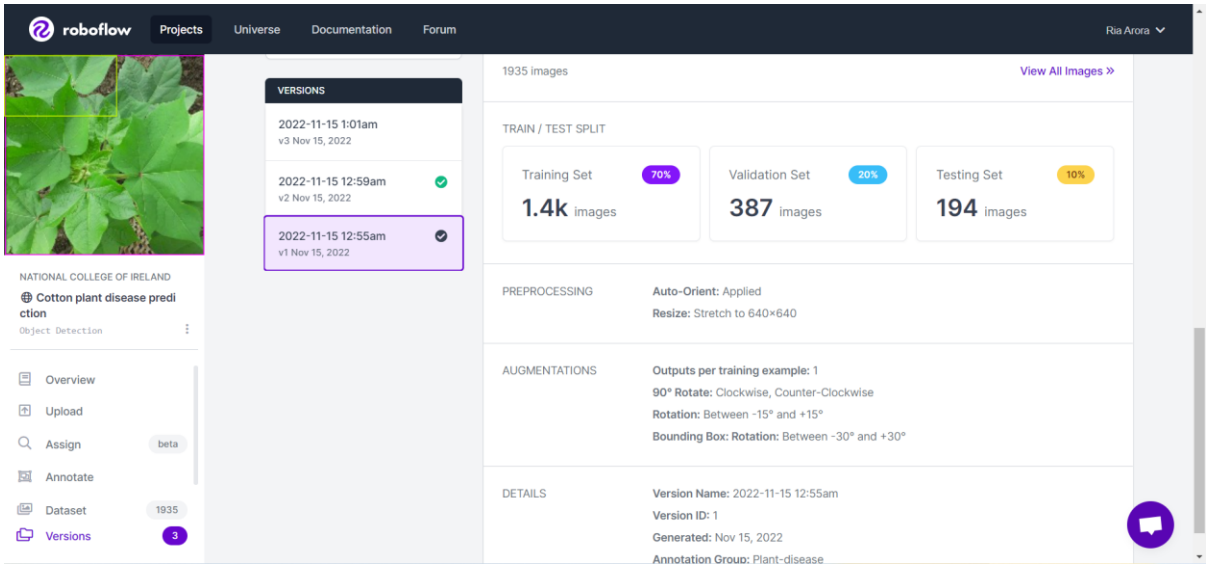

Figure 2. Roboflow used for data augmentation

4 Project Implementation Guide

This section shows all the codes and its explanation and includes the steps taken for project completion.

4.1 ResNet50 baseline model

This code below is a Python script that is importing several libraries, such as NumPy, pandas, seaborn, matplotlib, and TensorFlow. These libraries are used for machine learning and data science tasks. The code also includes some other lines that set up some basic parameters for image processing, such as setting the matplotlib inline backend and importing a ResNet50 model from Keras. It also sets up a basic neural network using the Sequential API in Keras. Finally, it imports some utility functions from the sklearn library for evaluating machine learning models.

```
import numpy as np
import pandas as pd
import os
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Input, Lambda, Dense, Flatten
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.applications.resnet50 import ResNet50
from keras import utils, callbacks
from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers, models
from tensorflow.keras.optimizers import Adam
from keras.losses import CategoricalCrossentropy
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn import metrics
```

Figure 3. Importing all the libraries

The below code is used for checking the GPU.

```
sess = tf.compat.v1.Session(config=tf.compat.v1.ConfigProto(log_device_placement=True))
```

```
Device mapping:
/job:localhost/replica:0/task:0/device:GPU:0 -> device: 0, name: NVIDIA GeForce RTX 2060, pci bus id: 0000:01:00.0, compute cap
ability: 7.5
```

Figure 4. Code to check GPU

The below code is used to re-size all the images.

```
IMAGE_SIZE = [224, 224]

train_path = 'Datasets/train'
valid_path = 'Datasets/test'
```

Figure 5. Code for image resizing

Importing the ResNet50 model and pre-processing the layer to the front of the ResNet.

```
resnet = ResNet50(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

Figure 6. ResNet50 model

Training the existing models. Here, the trainable layer is taken as false which means we are not training existing weights.

```
for layer in resnet.layers:
    layer.trainable = False
```

Figure 7. Model training

To get the output classes the below code is used.

```
folders = glob('Datasets/train/*')
```

Figure 8. Output classes are obtained

In this line of code, the Flatten layer from the TensorFlow library is being used to flatten the output of the resnet model. The Flatten layer takes the output tensor from the previous layer and flattens it into a 1-dimensional tensor, which can be used as input to the next layer in the neural network. This is a common step in the preprocessing of data for use in a neural network, and it allows the output of the previous layer to be fed into a dense layer, which is a fully connected layer of neurons in the neural network.

```
x = Flatten()(resnet.output)
```

Figure 9. Flatten layer is added

In this line of code, a Dense layer is added to the neural network. This layer is connected to the output of the Flatten layer, and it has 4 output units and uses the softmax activation function.

```
prediction = Dense(4, activation='softmax')(x)
```

Figure 10. The dense layer is added and the softmax activation function is used

The model object is created as below.

```
model = Model(inputs=resnet.input, outputs=prediction)
```

Figure 11. The model object is created

The loss function and optimizer are defined here.

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

Figure 12. Model.compile()

Used the Image Data Generator to import the images from the dataset.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   interpolation_order=1,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)
```

Figure 13. ImageDataGenerator

Here, the target size is taken the same as the image size which is 224,224, batch size is taken at 20 and the loss function used is categorical cross-entropy.

```
training_set = train_datagen.flow_from_directory('Datasets/train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')
```

```
Found 1951 images belonging to 4 classes.
```

```
test_set = test_datagen.flow_from_directory('Datasets/test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

```
Found 18 images belonging to 4 classes.
```

Figure 14. Target size, batch size, and class mode are defined

During training, the **fit_generator** method will iterate over the batches of data generated by the generator function and use them to update the model's weights. After each batch, it will evaluate the model's performance on a validation set, if one is provided, and adjust the model's internal parameters accordingly. At the end of the training, the model will have learned to make predictions on new data based on the patterns it has identified in the training data.

```
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=20,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
```

Figure 15. Model fit generator

The training and validation accuracy graph is plotted.

```
acc=history.history['accuracy']
val_acc=history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']

epochs=range(len(acc))

fig = plt.figure(figsize=(14,7))
plt.plot(epochs, acc, 'r', label="Training Accuracy")
plt.plot(epochs, val_acc, 'b', label="Validation Accuracy")
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc='lower right')
plt.show()
```

Figure 16. The training and validation accuracy graph is plotted

The training and validation loss graph is plotted.

```
fig = plt.figure(figsize=(14,7))
plt.plot(epochs, loss, 'r', label="Training Loss")
plt.plot(epochs, val_loss, 'b', label="Validation Loss")
plt.legend(loc='upper right')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and validation loss')
```

```
Text(0.5, 1.0, 'Training and validation loss')
```

Figure 17. The training and validation loss graph is plotted.

4.2 Fine-tuning data augmentation dataset (1:1)

The below steps show the additional changes made for running the fine-tuning model with data augmentation 1:1.

The location is changed here for training on different data. The target size should be the same as initiated for the image size.

```
training_set = train_datagen.flow_from_directory('../Data/class_data_11/train',
                                                 target_size = (224, 224),
                                                 batch_size = 8,
                                                 class_mode = 'categorical')
```

```
Found 1951 images belonging to 4 classes.
```

```
#Change Location here for training on different data
test_set = test_datagen.flow_from_directory('../Data/class_data_11/test',
                                            target_size = (224, 224),
                                            batch_size = 8,
                                            class_mode = 'categorical')
```

```
Found 36 images belonging to 5 classes.
```

```
#Change Location here for training on different data
valid_set = test_datagen.flow_from_directory('../Data/class_data_11/val',
                                            target_size = (224, 224),
                                            batch_size = 8,
                                            class_mode = 'categorical')
```

```
Found 324 images belonging to 4 classes.
```

Figure 18. New data location

The trainable layer in this case is taken True which means we are training existing weights.

```
for layer in resnet.layers:
    layer.trainable = True
```

Figure 19. Existing weights are trained

This code defines a function called create_model that creates a Keras model using the Sequential API. The model consists of several layers. A pre-trained ResNet model, which is imported but not shown in this code snippet. This is typically a convolutional neural network (CNN) trained on a large dataset to recognize a wide range of images. By using a pre-trained model, the model created by this function can benefit from the knowledge and features learned by the pre-trained model. A flatten layer, which flattens the output of the ResNet model from a 3D tensor to a 1D tensor. This is necessary because the next layer in the model is a dense (fully-connected) layer, which requires a 1D input tensor. A dropout layer with a dropout rate of 0.5. This layer randomly sets half of the input units to 0 at each update during training, which helps prevent overfitting and improves the generalizability of the model. A dense layer with 64 units and the ReLU activation function. This layer is a fully-connected layer that maps the output of the dropout layer to a 64-dimensional space. The ReLU activation function ensures that the output of this layer is always non-negative. Another dropout layer with a dropout rate of 0.5. This is the same as the previous dropout layer and serves the same purpose. A dense layer with 4 units and the softmax activation function. This is the output layer of the model, which maps the output of the previous layer to a 4-dimensional space and applies the softmax activation function to produce probabilities for each of the 4 classes. After the layers are added to the model, the compile method is called to configure the model for training. This function specifies the Adam optimizer, categorical cross-entropy loss, and accuracy metric for evaluating the model. The create_model function returns the constructed model.

```
from keras import models, regularizers, layers, optimizers, losses, metrics
from keras.models import Sequential
def create_model():
    model = models.Sequential()
    model.add(resnet)
    model.add(layers.Flatten())
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(64, activation='relu',kernel_regularizer=regularizers.l2(0.001)))
    model.add(layers.Dropout(0.5))
    model.add(layers.Dense(4, activation='softmax'))

    model.compile(optimizer=optimizers.Adam(lr=1e-5),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
    return model
```

Figure 20. Model architecture

```
def load_model():
    model = tf.keras.models.load_model('./Resnet50_model_12')
    return model
```

```
model = create_model()
print(model.summary())
```

```
Model: "sequential_2"
```

Figure 21. Loading and creating the model

This code defines an EarlyStopping callback in Keras. The EarlyStopping callback is used to stop the training of a model when it stops improving, which is determined by monitoring a given metric over time. In this case, the EarlyStopping callback is configured to monitor the validation loss (i.e., the loss on a validation dataset) and to stop the training when the loss stops decreasing for 5 consecutive epochs. Additionally, when the training is stopped, the callback restores the weights of the model that resulted in the lowest validation loss. This helps prevent overfitting and ensures that the model with the best performance on the validation set is used.

```
earlystopping = callbacks.EarlyStopping(monitor="val_loss", mode="min",
                                        patience=5, restore_best_weights = True)
```

```
epochs = 20
model=create_model()
with tf.device('/gpu:0'):
    history = model.fit(training_set,
                        epochs=epochs,
                        verbose=1,
                        validation_data=valid_set, shuffle=True, callbacks = [earlystopping])
```

```
Epoch 1/20
```

Figure 22. Model fit

```
model.save('./Resnet50_model_11',save_format='tf')
```

Figure 23. The model is saved in TensorFlow format

```
loaded_model = tf.keras.models.load_model('./Resnet50_model_11')
```

Figure 24. Loading the saved model

```
metrics = pd.DataFrame(model.history.history)
print("The model metrics are")
metrics
```

Figure 25. Metrics evaluation

4.3 Fine-tuning data augmentation dataset (1:2)

The data location is changed for the new data.

```
# Make sure you provide the same target size as initialied for the image size
#Change Location here for training on different data
training_set = train_datagen.flow_from_directory('./Data/class_data_12/train',
                                                 target_size = (224, 224),
                                                 batch_size = 4,
                                                 class_mode = 'categorical')
```

```
Found 2707 images belonging to 4 classes.
```

```
#Change Location here for training on different data
test_set = test_datagen.flow_from_directory('./Data/class_data_12/test',
                                            target_size = (224, 224),
                                            batch_size = 4,
                                            class_mode = 'categorical')
```

```
Found 194 images belonging to 4 classes.
```

```
#Change Location here for training on different data
valid_set = test_datagen.flow_from_directory('./Data/class_data_12/valid',
                                            target_size = (224, 224),
                                            batch_size = 4,
                                            class_mode = 'categorical')
```

```
Found 387 images belonging to 4 classes.
```

```
print(len(training_set))
```

```
677
```

Figure 26. New dataset location

This code defines a function called load_model that loads a pre-trained Keras model from a file on disk. The function uses the load_model method from the tf. Keras. models module, which takes the path of the file containing the pre-trained model as an argument. The function then returns the loaded model.

```
def load_model():
    model = tf.keras.models.load_model('./Resnet50_model_11')
    return model
```

Figure 27. Model loading

The new model is saved and loaded.

```
epochs = 20
model=create_model()
with tf.device('/gpu:0'):
    history = model.fit(training_set,
                        epochs=epochs,
                        verbose=1,
                        validation_data=valid_set, shuffle=True, callbacks = [earlystopping])
```

```
model.save('./Resnet50_model_12',save_format='tf')
```

```
loaded_model = tf.keras.models.load_model('./Resnet50_model_12')
```

Figure 28. The new model is saved and loaded

4.4 Fine-tuning 1:3

The Dataset location is changed again and the new model is saved and loaded.

```
# Make sure you provide the same target size as initialied for the image size
training_set = train_datagen.flow_from_directory('./Data/class_data_13/train',
                                                 target_size = (224, 224),
                                                 batch_size = 20,
                                                 class_mode = 'categorical')
```

```
Found 3256 images belonging to 4 classes.
```

```
test_set = test_datagen.flow_from_directory('./Data/class_data_13/test',
                                            target_size = (224, 224),
                                            batch_size = 20,
                                            class_mode = 'categorical')
```

```
Found 581 images belonging to 4 classes.
```

```
valid_set = test_datagen.flow_from_directory('./Data/class_data_13/valid',
                                            target_size = (224, 224),
                                            batch_size = 20,
                                            class_mode = 'categorical')
```

```
Found 387 images belonging to 4 classes.
```

```
print(len(training_set))
```

```
163
```

Figure 29. New data location

```
def load_model():
    model = tf.keras.models.load_model('./Resnet50_model_12')
    return model
```

```
model = load_model()
print(model.summary())
```

```
Model: "sequential"
```

```
model.save('./Resnet50_model_13',save_format='tf')
loaded_model = tf.keras.models.load_model('./Resnet50_model_13')
```

Figure 30. The model is saved and the new model is loaded