National College of Ireland

Configuration Manual

MSc Research Project
MSc. Data Analytics

Nikhil Awachat
Student ID: x21100446

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Nikhil Awachat
Student ID:	x21100446
Programme:	MSc. Data Analytics
Year:	2021 - 2022
Module:	MSc. Research Project
Supervisor:	Dr. Christian Horn
Submission Due Date:	15/08/2022
Project Title:	Plant Disease Classification Using Transfer Learning Methods
Word Count:	1039
Page Count:	11

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:	Nikhil Awachat
Date:	14th August 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

Nikhil Awachat
21100446

1 Introduction

The main objective of making this configuration manual will be to give all the details required to create this research with all the results obtained. To reproduce the results, this document includes snapshots of all the processes, including preprocessing, EDA, data augmentation, and model building, plus a summary and the design specification of all the models. The report follows a systematic structure to explain the research methodology. Section two gives all system hardware specifications for the main computer used for this study. Section three gives details about the dataset and the source information, with a link to the original dataset. Section four contains the snapshots of the data preprocessing and EDA. Section five deals with data augmentation and data split information. Section six has a detailed model building and specification and section seven consists of an evaluation.

2 Environment

2.1 Hardware Specification

The main system for this research has an AMD Ryzen 5 5500U Core CPU at 2.10 GHz, 8 GB of DDR4 RAM Memory with overclock at 3200 MHz, an AMD Radeon 6 Graphics card, SSD with 512 GB total memory, and has a Windows 10 64-bit OS.

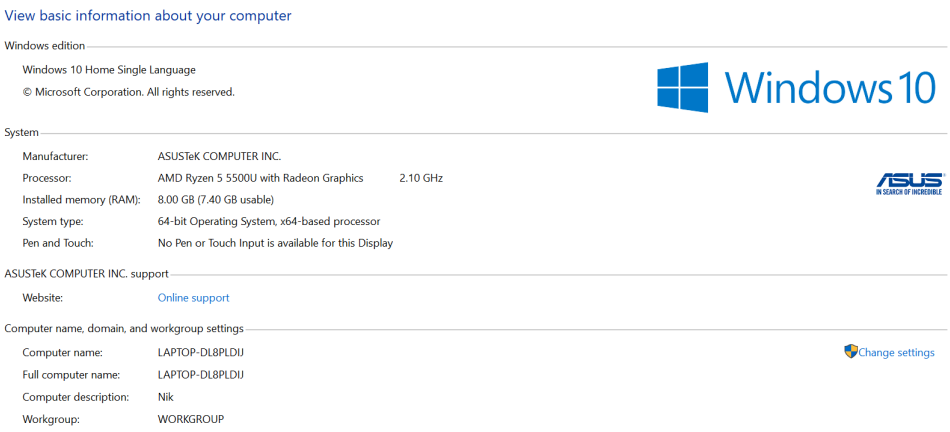

Figure 1: Specification for Hardware of the computer

2.2 Software Specification

1. Python (Version 3.8.12)
2. Jupyter Notebook (Version 6.4.8)

3 Data Collection

The dataset used for the entire research was taken from the Kaggle dataset repository. The link for the mentioned dataset is https://www.kaggle.com/datasets/shadabhussain/cgiar-computer-vision-for-crop-disease. There are 1486 raw pictures in the data, that are divided into training and validation sets categories. 876 pictures from three classes make up the training data. Although the images are accessible, they are not all the same size or shape. The dataset is collected from Ethiopia and Tanzania by an international organization on wheat crops.

4 Data Pre-processing and EDA

4.1 Data Balancing

The images are in jpg and jfif format which needs to be converted to a single format for better consistency, so the images were converted to the png format. The orignal data includes 3 classes—Healthy wheat, Steam rust, and Leaf rust and they are uniformly distributed. There is an imbalance because both the rust classes contain approximately twice as many photos as the Healthy wheat pictures.

```
'''Creating the balance dataset using flipping and rotating'''
c=0
List = os.listdir(source_path)
print(List)
# balanced_dataset_folder=r'balanced_dataset'
for x in List:
    folder= balanced_dataset_folder+r'/'+x+r'/'
    if os.path.exists(folder):
        pass
    else:
        os.makedirs(folder)
    path=os.path.join(source_path,x)
    images_folder = os.listdir(path)
    for images in images_folder:
        image_path = path + '/' + images
        if x == 'healthy_wheat':
            image1 = cv2.imread(image_path)
            cv2.imwrite(folder+x+str(c)+r'_.png',image1)
            c+=1
            if c>=17:
                image2 = cv2.flip(image1,1)
                cv2.imwrite(folder+x+str(c)+r'_.png',image2)
                c+=1

                image3 = cv2.cvtColor(image1, cv2.COLOR_BGR2RGB)
                cv2.imwrite(folder+x+str(c)+r'_.png',image3)
                c+=1
        else:
            image = cv2.imread(image_path)
            cv2.imwrite(folder+x+str(c)+r'_.png',image)
            c+=1
print('Count of total images in balanced dataset is: {}'.format(c))
```

Figure 2: Code to balanced data equally in all classes

To solve this issue, the 3 classes need to be balanced into an equal distribution, the remaining added images are augmented by a color change and flipping, which can be seen in the above image.

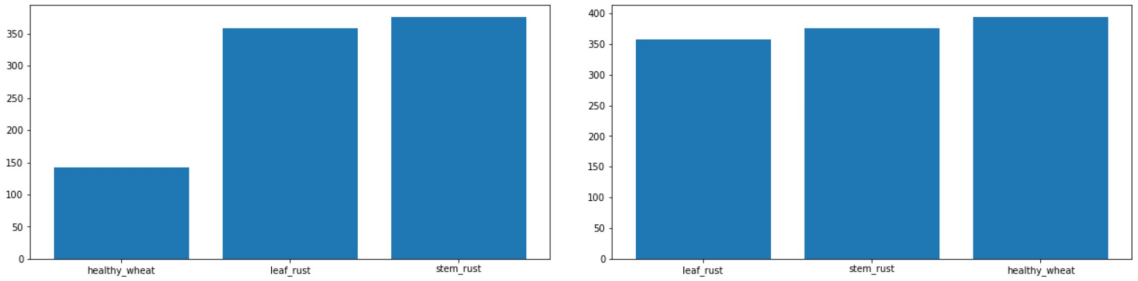

Figure 3: Dataset Distribution before and after balancing

4.2 Data Splitting

Due to the dataset's lack of a separate testing folder for each of the 3 categories, a process was introduced by creating two training and testing folders. The initial data was then divided between training and testing sets groups in a ratio of 75:25.

```
categories=['healthy_wheat', 'leaf_rust', 'stem_rust']
try:
    for category in categories:
        path = os.path.join("./train", category)
        os.makedirs(path)
        path = os.path.join("./test", category)
        os.makedirs(path)
    print("Folders created")
except:
    print("Folders already created")
```

```
Folders created
```

```
source_path=r"C:/Users/Nik/Desktop/Plant Disease/balanced_dataset/"
def generateData(lst,fnm):
    for i in range(len(lst)):
        if i<=len(lst)*0.75:
            destination="./train/"+fnm
            folder=fnm+"/"
        else:
            destination="./test/"+fnm
            folder=fnm+"/"
        shutil.copy(os.path.join(source_path,folder,lst[i]), destination)
```

Figure 4: Code for data split into train and test

The above code helps in splitting the data into train and test, first by creating folders for three classes in each of the train and test folders. The images are then split into a ratio of 75:25 into train and test folders. This is useful as the validation set is necessary to get the validation accuracy for all our models which is an important metric to check model performance.

5 Data Augmentation

5.1 Image Resize

Various data enhancement methods were performed on train data to achieve accurate results on the validating test set while preventing the model from overfitting. The original pictures were all in various dimensions, making it harder to train and test and requiring a lot of computing resources. As a result, all of the images were reduced to one size of 224 * 224. The images will be multiplied by 255 during the normalization procedure, which will replicate the image and utilize a scale of 0 to 1.

```
TRAIN_DIR = 'train/'
TEST_DIR = 'test/'
```

```
gen = ImageDataGenerator(rescale=1./255)
train = gen.flow_from_directory(directory=TRAIN_DIR, target_size=(224,224), batch_size=32,shuffle=True)
```

```
Found 848 images belonging to 3 classes.
```

```
gen = ImageDataGenerator(rescale=1./255)
test = gen.flow_from_directory(directory=TEST_DIR, target_size=(224,224), batch_size=32, shuffle=True)
```

```
Found 280 images belonging to 3 classes.
```

Figure 5: Code for image resize and rescalet

5.2 Augmentation

To check if augmentation would help in improving model efficiency, images are augmented by changing the color of the images to RGB and then by finding contours and doing a grayscale conversion which is shown in the code below.

```
count = 1
f = plt.figure(figsize=(50,20))
for file in leaf_rust_images[:8]:
    img = cv2.imread(file)
    img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
    ax = f.add_subplot(2, 4,count)
    ax = plt.imshow(img)
    ax = plt.title('Leaf Rust',fontsize= 30)
    count = count + 1
plt.show()
```

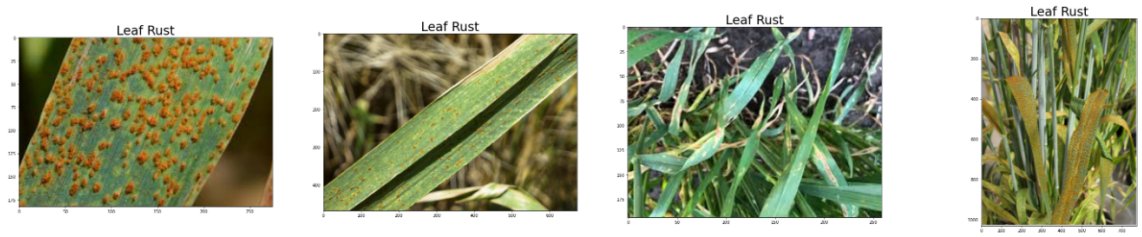

Figure 6: Image RGB conversion

```
count = 1
f = plt.figure(figsize=(50,20))
for file in healthy_wheat_images[:8]:
    img = cv2.imread(file)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    adapimg = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,11,3)
    cnts = cv2.findContours(adapimg, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = cnts[0] if len(cnts) == 2 else cnts[1]
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)
    for c in cnts:
        x,y,w,h = cv2.boundingRect(c)
        img_ROI = img[y:y+h, x:x+w]
        break
    ax = f.add_subplot(2, 4,count)
    ax = plt.imshow(img_ROI)
    ax = plt.title('Healthy Wheat',fontsize= 30)
    count = count + 1
plt.show()
```

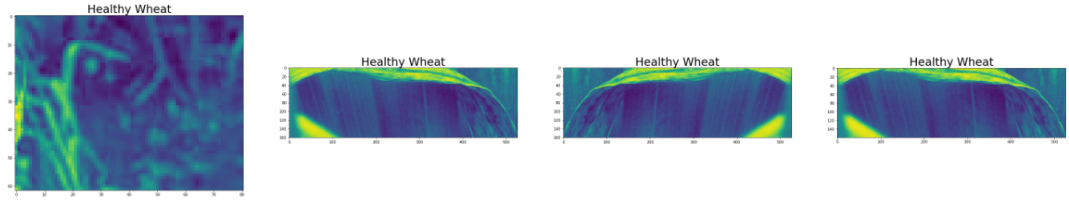

Figure 7: Image contours and Grayscale conversion

6 Model Building

The model is developed in this stage using various different approaches. CNN, VGG16, GoogleNet, and AlexNet are the four models created in this study. Certain parameters are set untrainable in order to reduce the complexity and computing power required for the algorithms, which reduces the total model parameters that may be used.

6.1 VGG16 Model

The first approach used is built on CNN and contains 16 layers in total known as VGG16. VGG16 is made up of 21 layers in total, but only 8 of them are trainable parameter layers. It contains thirteen convolution layers, five max-pooling layers, three dense layers, one SoftMax layer with three classes, and a flatten layer.

VGG16

```
model = Sequential()

#VGG16 Block 1
model.add(Conv2D(input_shape=(224,224,3),filters=64,kernel_size=(3,3),padding="same", activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same", activation="relu", trainable=False))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#VGG16 Block 2
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu"))
model.add(Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#VGG16 Block 3
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#VGG16 Block 4
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(MaxPool2D(pool_size=(2,2),strides=(2,2)))
```

```
#VGG16 Block 5
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
model.add(Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="relu", trainable=False))
```

```
model.add(Flatten(name='flatten'))
model.add(Dense(512, activation='relu', trainable=False))
model.add(Dense(4096, activation='relu', trainable=False))
model.add(Dropout(0.5))
model.add(Dense(units=3, activation="Softmax"))
```

```
import tensorflow as tf
opt = Adam(lr=0.01)
loss = tf.keras.losses.CategoricalCrossentropy()
model.compile(optimizer=opt, loss=loss, metrics=['accuracy'])
```

Figure 8: Code for VGG16 model

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 224, 224, 64)	1792
conv2d_14 (Conv2D)	(None, 224, 224, 64)	36928
max_pooling2d_4 (MaxPooling 2D)	(None, 112, 112, 64)	0
conv2d_15 (Conv2D)	(None, 112, 112, 128)	73856
conv2d_16 (Conv2D)	(None, 112, 112, 128)	147584
max_pooling2d_5 (MaxPooling 2D)	(None, 56, 56, 128)	0
conv2d_17 (Conv2D)	(None, 56, 56, 256)	295168
conv2d_18 (Conv2D)	(None, 56, 56, 256)	590080
conv2d_19 (Conv2D)	(None, 56, 56, 256)	590080
max_pooling2d_6 (MaxPooling 2D)	(None, 28, 28, 256)	0
conv2d_20 (Conv2D)	(None, 28, 28, 512)	1180160
conv2d_21 (Conv2D)	(None, 28, 28, 512)	2359808
conv2d_22 (Conv2D)	(None, 28, 28, 512)	2359808
max_pooling2d_7 (MaxPooling 2D)	(None, 14, 14, 512)	0
conv2d_23 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_24 (Conv2D)	(None, 14, 14, 512)	2359808
conv2d_25 (Conv2D)	(None, 14, 14, 512)	2359808
flatten (Flatten)	(None, 100352)	0
dense_3 (Dense)	(None, 512)	51380736
dense_4 (Dense)	(None, 4096)	2101248
dropout_1 (Dropout)	(None, 4096)	0
dense_5 (Dense)	(None, 3)	12291

Total params: 68,208,963
Trainable params: 87,939
Non-trainable params: 68,121,024

Figure 9: Summary VGG16 model

6.2 CNN Model

The next model that is built has 10 layers overall and is known as CNN. The CNN framework includes ten layers in total: Three convolutional layers, Three max-pooling layers, Two dense layers, a SoftMax layer with three classes, and a dropout layer.

```
# CNN
input_shape = (224, 224, 3)

model3 = Sequential()

model3.add(Conv2D(32, (3,3), input_shape=input_shape, activation='relu',data_format='channels_last'))
model3.add(MaxPooling2D((2,2), strides=(1,1), padding='same'),)

model3.add(Conv2D(64, (3,3), activation='relu'),)
model3.add(MaxPooling2D((2,2), strides=(1,1), padding='same'),)

model3.add(Conv2D(128, (3,3), activation='relu', trainable=False),)
model3.add(MaxPooling2D((2,2), strides=(1,1), padding='same'),)

model3.add(Flatten())

model3.add(Dense(128, activation='relu', trainable=False))
model3.add(Dense(64, activation='relu'))

model3.add(Dropout(0.5))
model3.add(Dense(3, activation='softmax'))
```

Figure 10: CNN model code

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 222, 222, 32)      896

max_pooling2d (MaxPooling2D  (None, 222, 222, 32)     0
)

conv2d_1 (Conv2D)            (None, 220, 220, 64)      18496

max_pooling2d_1 (MaxPooling  (None, 220, 220, 64)     0
2D)

conv2d_2 (Conv2D)            (None, 218, 218, 128)     73856

max_pooling2d_2 (MaxPooling  (None, 218, 218, 128)    0
2D)

flatten (Flatten)            (None, 6083072)           0

dense (Dense)                (None, 128)               778633344

dense_1 (Dense)              (None, 64)                8256

dropout (Dropout)            (None, 64)                0

dense_2 (Dense)              (None, 3)                 195

=================================================================
Total params: 778,735,043
Trainable params: 27,843
Non-trainable params: 778,707,200
```

Figure 11: CNN model summary

6.3 AlexNet Model

The next model used is AlexNet and it has a total of 19 layers. The AlexNet model includes Nineteen layers overall, including Three dropout networks, a SoftMax network with three classes, Five convolutions, three max-pooling networks, Five batch normalization networks, and two dense networks.

```python
model5 = keras.models.Sequential([
    keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shape=(224,224,3)),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), activation='relu', padding="same", trainable=False),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same", trainable=False),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same", trainable=False),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), activation='relu', padding="same", trainable=False),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3,3), strides=(2,2)),
    keras.layers.Flatten(),
    keras.layers.Dense(4096, activation='relu', trainable=False),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(4096, activation='relu', trainable=False),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(3, activation='softmax')
])
```

```python
opt = tensorflow.keras.optimizers.Adam(learning_rate=0.001)
model5.compile(optimizer=opt,loss="categorical_crossentropy",metrics=['accuracy'])
model5.summary()
```

Figure 12: AlexNet model code

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 54, 54, 96)        34944

 batch_normalization (BatchN  (None, 54, 54, 96)       384
 ormalization)

 max_pooling2d_3 (MaxPooling  (None, 26, 26, 96)       0
 2D)

 conv2d_4 (Conv2D)           (None, 26, 26, 256)       614656

 batch_normalization_1 (Batc  (None, 26, 26, 256)      1024
 hNormalization)

 max_pooling2d_4 (MaxPooling  (None, 12, 12, 256)      0
 2D)

 conv2d_5 (Conv2D)           (None, 12, 12, 384)       885120

 batch_normalization_2 (Batc  (None, 12, 12, 384)      1536
 hNormalization)

 conv2d_6 (Conv2D)           (None, 12, 12, 384)       1327488

 batch_normalization_3 (Batc  (None, 12, 12, 384)      1536
 hNormalization)

 conv2d_7 (Conv2D)           (None, 12, 12, 256)       884992

 batch_normalization_4 (Batc  (None, 12, 12, 256)      1024
 hNormalization)

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 flatten_1 (Flatten)         (None, 6400)              0

 dense_3 (Dense)             (None, 4096)              26218496

 dropout_1 (Dropout)         (None, 4096)              0

 dense_4 (Dense)             (None, 4096)              16781312

 dropout_2 (Dropout)         (None, 4096)              0

 dense_5 (Dense)             (None, 3)                 12291

=================================================================
Total params: 46,764,803
Trainable params: 49,987
Non-trainable params: 46,714,816
```

Figure 13: AlexNet model summary

6.4 GoogleNet Model

GoogleNet is the last model developed, which is made up of 22 levels. The 22 levels that make up the GoogleNet architecture are consisting of three dense fully connected layers, a SoftMax Dense network with three classes, Five convolution networks, Four max-pooling networks, Nine inception networks, and a dropout network.

```
def Inception_block(input_layer, f1, f2_conv1, f2_conv3, f3_conv1, f3_conv5, f4):
  # Input:
  # - f1: number of filters of the 1x1 convolutional layer in the first path
  # - f2_conv1, f2_conv3 are number of filters corresponding to the 1x1 and 3x3 convolutional layers in the second path
  # - f3_conv1, f3_conv5 are the number of filters corresponding to the 1x1 and 5x5  convolutional layer in the third path
  # - f4: number of filters of the 1x1 convolutional layer in the fourth path

  # 1st path:
  path1 = Conv2D(filters=f1, kernel_size = (1,1), padding = 'same', activation = 'relu')(input_layer)

  # 2nd path
  path2 = Conv2D(filters = f2_conv1, kernel_size = (1,1), padding = 'same', activation = 'relu')(input_layer)
  path2 = Conv2D(filters = f2_conv3, kernel_size = (3,3), padding = 'same', activation = 'relu')(path2)

  # 3rd path
  path3 = Conv2D(filters = f3_conv1, kernel_size = (1,1), padding = 'same', activation = 'relu')(input_layer)
  path3 = Conv2D(filters = f3_conv5, kernel_size = (5,5), padding = 'same', activation = 'relu')(path3)

  # 4th path
  path4 = MaxPooling2D((3,3), strides= (1,1), padding = 'same')(input_layer)
  path4 = Conv2D(filters = f4, kernel_size = (1,1), padding = 'same', activation = 'relu')(path4)

  output_layer = concatenate([path1, path2, path3, path4], axis = -1)

  return output_layer
```

Figure 14: Inception block of the GoogleNet

```
def GoogLeNet():
  # input layer
  input_layer = Input(shape = (224, 224, 3))

  # convolutional layer: filters = 64, kernel_size = (7,7), strides = 2
  X = Conv2D(filters = 64, kernel_size = (7,7), strides = 2, padding = 'valid', activation = 'relu')(input_layer)

  # max-pooling layer: pool_size = (3,3), strides = 2
  X = MaxPooling2D(pool_size = (3,3), strides = 2)(X)

  # convolutional layer: filters = 64, strides = 1
  X = Conv2D(filters = 64, kernel_size = (1,1), strides = 1, padding = 'same', activation = 'relu', trainable=False)(X)

  # convolutional layer: filters = 192, kernel_size = (3,3)
  X = Conv2D(filters = 192, kernel_size = (3,3), padding = 'same', activation = 'relu', trainable=False)(X)

  # max-pooling layer: pool_size = (3,3), strides = 2
  X = MaxPooling2D(pool_size= (3,3), strides = 2)(X)

  # 1st Inception block
  X = Inception_block(X, f1 = 64, f2_conv1 = 96, f2_conv3 = 128, f3_conv1 = 16, f3_conv5 = 32, f4 = 32)

  # 2nd Inception block
  X = Inception_block(X, f1 = 128, f2_conv1 = 128, f2_conv3 = 192, f3_conv1 = 32, f3_conv5 = 96, f4 = 64)

  # max-pooling layer: pool_size = (3,3), strides = 2
  X = MaxPooling2D(pool_size= (3,3), strides = 2)(X)

  # 3rd Inception block
  X = Inception_block(X, f1 = 192, f2_conv1 = 96, f2_conv3 = 208, f3_conv1 = 16, f3_conv5 = 48, f4 = 64)
```

```
# Extra network 1:
X1 = AveragePooling2D(pool_size = (5,5), strides = 3)(X)
X1 = Conv2D(filters = 128, kernel_size = (1,1), padding = 'same', activation = 'relu', trainable=False)(X1)
X1 = Flatten()(X1)
X1 = Dense(1024, activation = 'relu', trainable=False)(X1)
X1 = Dropout(0.7)(X1)
X1 = Dense(3, activation = 'softmax')(X1)


# 4th Inception block
X = Inception_block(X, f1 = 160, f2_conv1 = 112, f2_conv3 = 224, f3_conv1 = 24, f3_conv5 = 64, f4 = 64)

# 5th Inception block
X = Inception_block(X, f1 = 128, f2_conv1 = 128, f2_conv3 = 256, f3_conv1 = 24, f3_conv5 = 64, f4 = 64)

# 6th Inception block
X = Inception_block(X, f1 = 112, f2_conv1 = 144, f2_conv3 = 288, f3_conv1 = 32, f3_conv5 = 64, f4 = 64)

# Extra network 2:
X2 = AveragePooling2D(pool_size = (5,5), strides = 3)(X)
X2 = Conv2D(filters = 128, kernel_size = (1,1), padding = 'same', activation = 'relu', trainable=False)(X2)
X2 = Flatten()(X2)
X2 = Dense(1024, activation = 'relu', trainable=False)(X2)
X2 = Dropout(0.7)(X2)
X2 = Dense(3, activation = 'softmax')(X2)


# 7th Inception block
X = Inception_block(X, f1 = 256, f2_conv1 = 160, f2_conv3 = 320, f3_conv1 = 32,
                    f3_conv5 = 128, f4 = 128)
```

```
# max-pooling layer: pool_size = (3,3), strides = 2
X = MaxPooling2D(pool_size = (3,3), strides = 2)(X)

# 8th Inception block
X = Inception_block(X, f1 = 256, f2_conv1 = 160, f2_conv3 = 320, f3_conv1 = 32, f3_conv5 = 128, f4 = 128)

# 9th Inception block
X = Inception_block(X, f1 = 384, f2_conv1 = 192, f2_conv3 = 384, f3_conv1 = 48, f3_conv5 = 128, f4 = 128)

# Global Average pooling layer
X = GlobalAveragePooling2D(name = 'GAPL')(X)

# Dropoutlayer
X = Dropout(0.4)(X)

# output layer
X = Dense(3, activation = 'softmax')(X)

# model
model7 = Model(input_layer, [X, X1, X2], name = 'GoogLeNet')

return model7
```

Figure 15: GoogleNet model code

7 Evaluation

The metrics used to evaluate the models and get the model performance are shown below, including the train and validation accuracy and loss summary graph, and the confusion matrix to check the true positive and negative values.

```
# summarize history for accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```

Figure 16: Summary graph code for train and validation accuracy

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
target_names=['healthy_wheat','leaf_rust','stem_rust']


plt.figure(figsize=(10,10))
cm = confusion_matrix(y_test1, y_pred)
_=sns.heatmap(cm.T, annot=True, fmt='d', cbar=True, square=True, xticklabels=target_names,
              yticklabels=target_names)
plt.xlabel('Truth')
plt.ylabel('Predicted')
plt.savefig('./'+'confusion_matrix_onValidation.jpeg',dpi=93)
```

Figure 17: Confusion matrix code

8 Data Source

https://www.kaggle.com/datasets/shadabhussain/cgiar-computer-vision-for-crop-disease

9 GitHub Code

https://nbviewer.org/github/nikhilawachat123/Resaerch_Project_x21100446/blob/main/Nikhil_x21100446_PlantDiseaseClassification_Final.ipynb