Cabbage disese prediction

Table of contents

About the dataset:

Dataset include images of 'Alternaria', 'Bacterial_soft_rot' and 'Healthy' cabbage leaves. The images are collected from various internet sources.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
! pip install split-folders --quiet
```

Importing necessary libraries

1. Using pandas library to load dataset and data processing
2. Numpy to work with arrays and matrices
3. Matplotlib for data visualization
4. Using splitfolders , splitting the data into train, test and validation dataset
5. Using tensorflow and keras libraries for model building and training.

```
import pandas as pd
import numpy as np
#import splitfolders
from tensorflow import keras
```

```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras import layers, models
```

- Before exploring the dataset splitting the data into training , testing and validation dataset. Since I am using tensorflows "image_dataset_from_directory" to load the images, they donot have an option to split the dataset into three directories, they provide only train and validation split. However, I would like to have train, test and validation split. And using splitfolder function to do the same.
- The splifolders function splits the data with respect to the ratios. Therefore it takes the input dataset directory , the destination directory to save the splitted datasets and the ratio of the split as parameters.
- Here, by passing the dataset directory into splitfolder function, splitting the data into train, test and validation dataset providing the ratio as 80% , 10% and 10% respectively and storing it in a new folder (dataset- foldername).

```
splitfolders.ratio("/content/drive/MyDrive/thesis_crop_dis_pre/cabbage", output="/content/dri
     ratio=(.8, .1, .1))

     Copying files: 139 files [00:48,  2.86 files/s]
```

Image size and batch size are assigned with default values

```
image_size = (256, 256)
batch_size = 32
```

Exploratory data analysis

Since its an image data using tensorflows " image_dataset_from_directory " to load the datasets.

Loading the train data passing parameters such as

1. directory, which gives the path of the train data
2. labels is set to 'inferred' (since i would like to have the same label names from the directory)
3. with lables_mode as int, encoding the labels as integers
4. using a default batch size (32) to train the data
5. providing default image size (256,256)
6. also providing image channel (color_node) as 3 ie., rgb

7. since we have seperate datasets for validation, here the validation_split is set to None

- image_size and batch_size values could be assigned to a variable since we make use of it again.

```
df_train = tf.keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/thesis_crop_dis_pre/cabbage/dataset_cab/train',
    labels='inferred',
    label_mode ='int',
    class_names=None,
    color_mode ='rgb',
    batch_size = batch_size,
    image_size = image_size,
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
)
```

```
     Found 111 files belonging to 3 classes.
```

Since I have assigned the labels to inferred, we can get the list of class names using the method "class_names" and the names would match the subdirectory names of the data

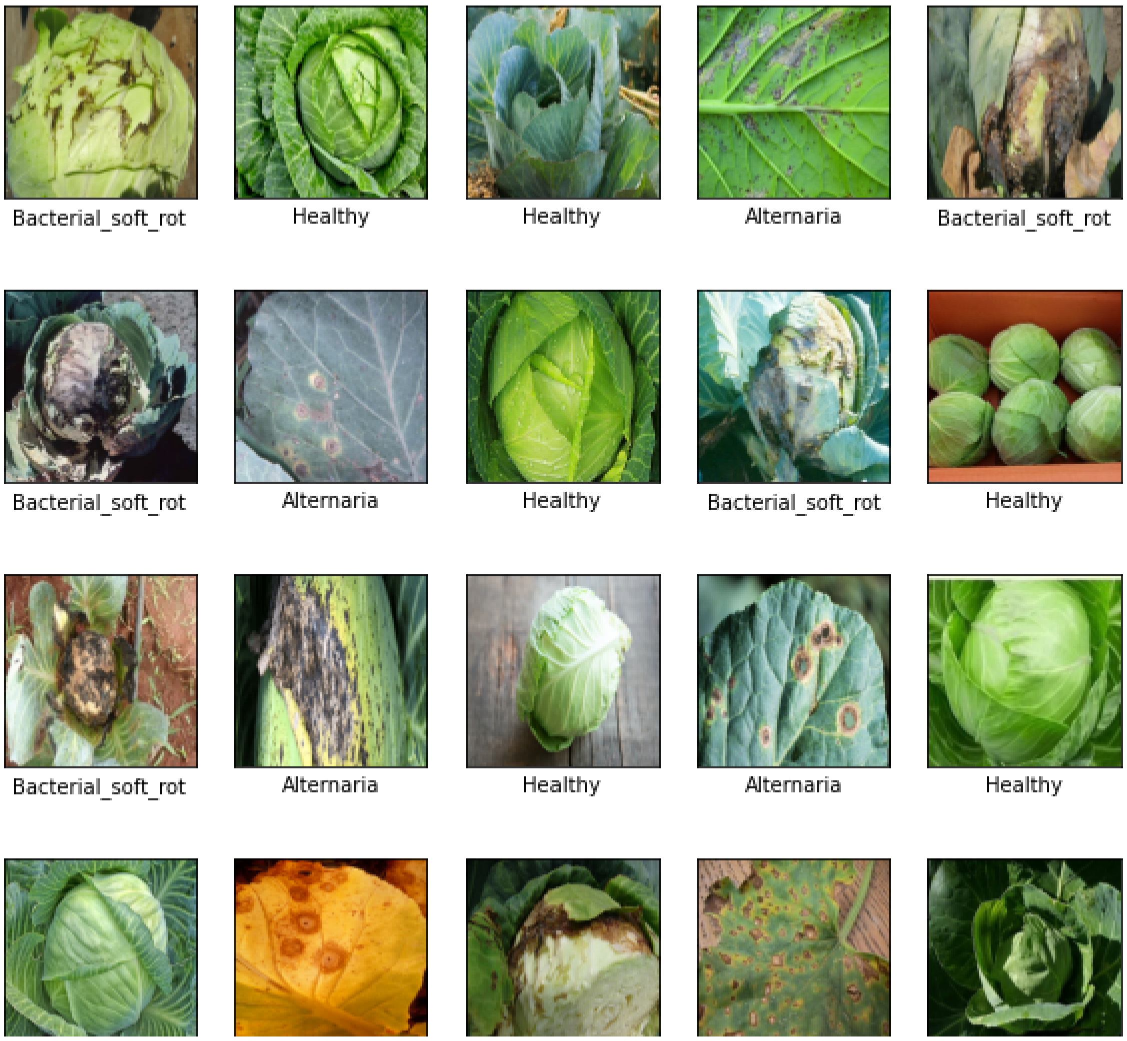
```
classes = df_train.class_names
classes
```

```
     ['Alternaria', 'Bacterial_soft_rot', 'Healthy']
```

```
for image, label in df_train.take(2):
  print(image.numpy())
  print(label.numpy())
```

```
        [2.54000000e+02 2.54000000e+02 2.54000000e+02]
        [2.54000000e+02 2.54000000e+02 2.54000000e+02]
        ...
        [2.54000000e+02 2.54000000e+02 2.54000000e+02]
        [2.54000000e+02 2.54000000e+02 2.54000000e+02]
        [2.54000000e+02 2.54000000e+02 2.54000000e+02]]]


      [[[8.56796875e+01 8.76796875e+01 3.86796875e+01]
        [8.63750000e+01 8.63750000e+01 3.23750000e+01]
        [8.93750000e+01 8.73750000e+01 3.83750000e+01]
        ...
        [4.05625000e+01 8.15625000e+01 6.75625000e+01]
        [3.63750000e+01 8.03750000e+01 6.73750000e+01]
        [3.90000000e+01 7.90000000e+01 6.80000000e+01]]

       [[7.82265625e+01 8.12500000e+01 3.41796875e+01]
        [8.01250000e+01 8.31250000e+01 3.01250000e+01]
        [8.79375000e+01 8.59375000e+01 3.79375000e+01]
        ...
        [4.10000000e+01 8.10000000e+01 7.00000000e+01]
        [3.88750000e+01 8.08750000e+01 7.08750000e+01]
        [3.91875000e+01 8.11875000e+01 7.11875000e+01]]

       [[6.77656250e+01 7.30156250e+01 3.16406250e+01]
        [8.13203125e+01 8.34375000e+01 3.74375000e+01]
        [8.65703125e+01 8.59453125e+01 3.95703125e+01]
        ...
        [4.08750000e+01 8.11250000e+01 7.17500000e+01]
        [4.08750000e+01 8.10000000e+01 7.30000000e+01]
        [4.01640625e+01 8.31640625e+01 7.41640625e+01]]

       ...

       [[1.10234375e+01 1.82734375e+01 2.61484375e+01]
        [1.09296875e+01 2.01796875e+01 2.70546875e+01]
        [1.20000000e+01 1.92500000e+01 2.71250000e+01]
        ...
        [1.37500000e+00 5.12500000e+00 4.12500000e+00]
        [1.37500000e+00 5.12500000e+00 4.12500000e+00]
        [1.25000000e+00 5.00000000e+00 4.00000000e+00]]

       [[1.00000000e+01 2.00000000e+01 2.20000000e+01]
        [1.10000000e+01 2.10000000e+01 2.30000000e+01]
        [8.00000000e+00 2.20000000e+01 2.30000000e+01]
        ...
        [1.00000000e+00 5.00000000e+00 4.00000000e+00]
        [1.00000000e+00 5.00000000e+00 4.00000000e+00]
        [1.00000000e+00 5.00000000e+00 4.00000000e+00]]

       [[9.00000000e+00 2.00000000e+01 2.40000000e+01]
        [9.00000000e+00 2.00000000e+01 2.40000000e+01]
        [9.00000000e+00 2.00000000e+01 2.40000000e+01]
        ...
        [1.00000000e+00 5.00000000e+00 4.00000000e+00]
        [1.00000000e+00 5.00000000e+00 4.00000000e+00]
        [1.00000000e+00 5.00000000e+00 4.00000000e+00]]]]
     [1 0 1 2 1 2 0 0 2 0 2 2 2 1 1 0 0 1 0 0 0 1 0 2 0 2 1 0 0 1 2 0]
```

```
for image, label in df_train.take(1):
  plt.figure(figsize=(10,10))
  for i in range(20):
    plt.subplot(4,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(image[i].numpy().astype('int'), cmap=plt.cm.binary)
    plt.xlabel(classes[label[i]])
  plt.show()
```



Loading validation and test dataset from the directory again through "image_dataset_from_directory" providing the default batch size and image size

```
df_vali = tf.keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/thesis_crop_dis_pre/cabbage/dataset_cab/val',
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    batch_size= batch_size,
    image_size= image_size,
    shuffle=True,
)
```

```
     Found 12 files belonging to 3 classes.
```

```
df_test = tf.keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/thesis_crop_dis_pre/cabbage/dataset_cab/test',
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    batch_size= batch_size,
    image_size= image_size,
    shuffle=True,)
```

```
     Found 16 files belonging to 3 classes.
```

Data preprocessing

Using tensorflows preprocessing layers

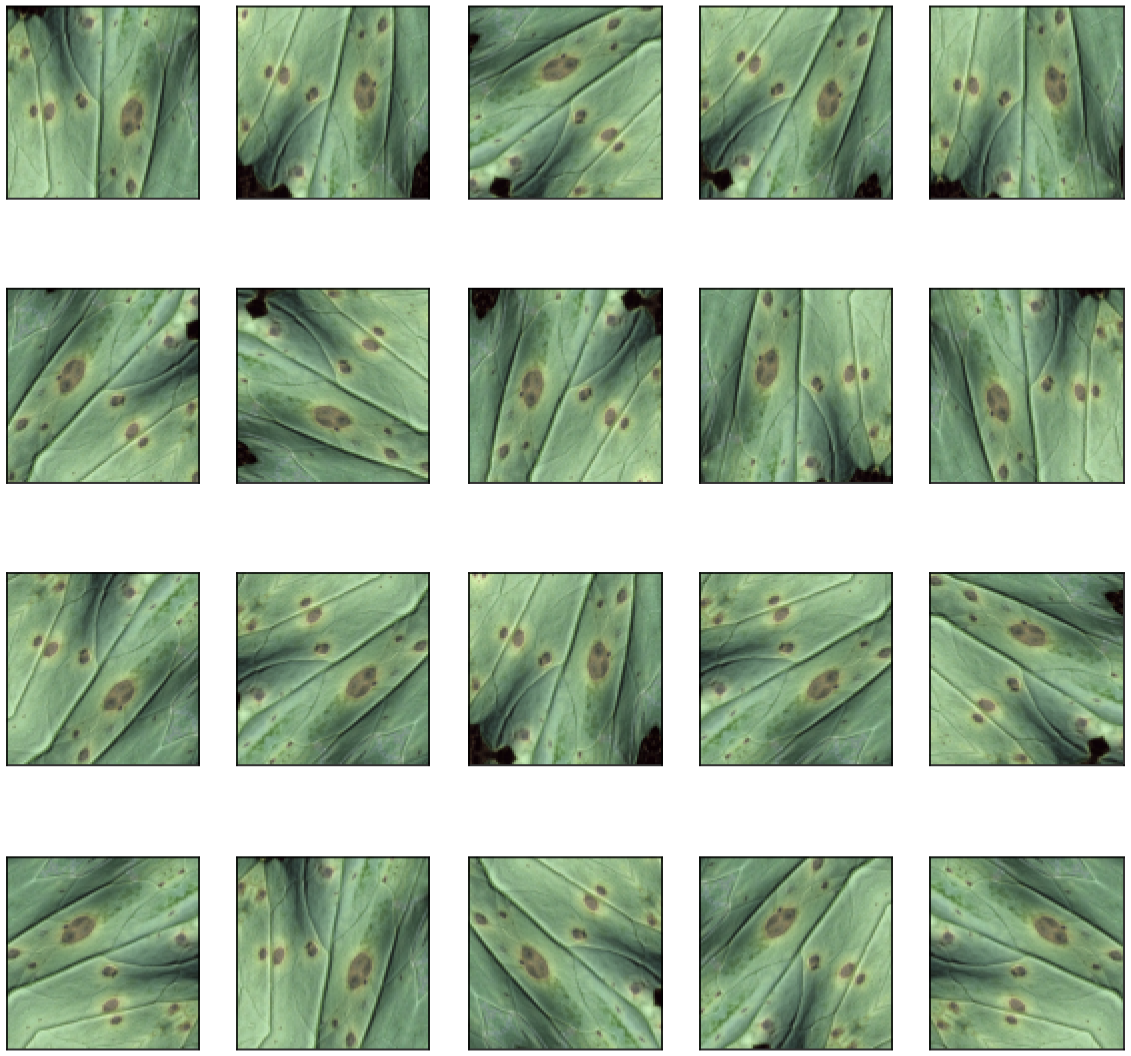
1. Resizing layers- used to change image length and widhth to (256,256)
2. Rescaling layers- to stanadradize the data
3. RandomZoom layers - to randomly zoom in or out on each axis of an image independently

```
data_preprocessing = tf.keras.Sequential([

                                layers.experimental.preprocessing.Resizing(256,256),
                                layers.experimental.preprocessing.Rescaling(1./255, input_sha
                                layers.experimental.preprocessing.RandomZoom(0.2)

                    ])


data_augumentation = tf.keras.Sequential([
                                layers.experimental.preprocessing.RandomFlip('horizontal_and_
                                layers.experimental.preprocessing.RandomRotation(0.2),

                    ])
```

Before using data augumentation on the dataset, I have tried to visualisize how it looks with this particular dataset

Using RandomFlip and RandomRotation layers

```
for image, label in df_train.take(1):
  plt.figure(figsize=(10,10))
  for i in range(20):
    augmented_images = data_augumentation(image)
    plt.subplot(4,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(augmented_images[0].numpy().astype('int'), cmap=plt.cm.binary)
  plt.show()
```



Model building

Within the sequential layers having

1. data_preprocessing as the first layer consisting of resizing, rescaling and randomzoom preprocessing layers
2. Secondly having stack of convolutional 2D and Maxpooling layers. In this case, using 4 layers of each (tried different counts), the convolutional layers take filters to find the features of the images and its size which is given by the kernel_size and an activation function

- The first convolutional layer takes the input shape (256,256,3) which is image height, width and the rgb mode , 32 filters with size of 3* 3 and relu as its activation function. And the following convolutional layers take same kernel size and activation function but the filter value as 64
- Using maxpooling layers as it extracts the main and sharp features from the images providing size of 3 * 3

3. Within the dense network

- Having a flatten layer, which helps to reduce the dimensionality of the input to single dimension

```
def check_opt(optimizers):

  model = models.Sequential([
    data_preprocessing,
    layers.Conv2D(filters=32, kernel_size= (3, 3), activation='relu', input_shape=(256, 256,
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(filters=64, kernel_size= (3, 3), activation='relu'),
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(filters=64, kernel_size= (3, 3), activation='relu'),
    layers.MaxPooling2D((3, 3)),
    layers.Flatten(),
    layers.Dense(64, activation='relu' ),
    layers.Dense(37, activation = 'softmax'),

    ])


  model.compile(optimizer= optimizers,
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
              metrics=['accuracy'])

  return model


optimizers = ['Adadelta', 'Adagrad', 'Adam', 'RMSprop', 'SGD']


train_acc = []
val_acc = []
train_loss =[]
val_loss = []
```

Trying out the following five optimizers and storing it in an empty lists

Looping through all the optimizers and applying to the function

Using model.fit to train the training dataset and storing it in a variable. The fit model returns an history callback object which could be called to return the loss and accuracy values

Appending the final epoch results to the list. Here since I use only 5 epochs appending the fifth loss and accuracy value with index value 4

```
for i in optimizers:
  model = check_opt(i)
  print('With optimizer:'+ i)
```

```
history = model.fit(df_train, epochs= 5, batch_size = 32,validation_data= df_vali)

train_acc.append(history.history['accuracy'][4])
val_acc.append( history.history['val_accuracy'][4])

train_loss.append(history.history['loss'][4])
val_loss.append(history.history['val_loss'][4])
```

```
With optimizer:Adadelta
Epoch 1/5
4/4 [==============================] - 14s 3s/step - loss: 3.6087 - accuracy: 0.2072 - v
Epoch 2/5
4/4 [==============================] - 9s 2s/step - loss: 3.6076 - accuracy: 0.2072 - va
Epoch 3/5
4/4 [==============================] - 9s 2s/step - loss: 3.6043 - accuracy: 0.2072 - va
Epoch 4/5
4/4 [==============================] - 9s 2s/step - loss: 3.6029 - accuracy: 0.2072 - va
Epoch 5/5
4/4 [==============================] - 9s 2s/step - loss: 3.6050 - accuracy: 0.2072 - va
With optimizer:Adagrad
Epoch 1/5
4/4 [==============================] - 9s 2s/step - loss: 3.5540 - accuracy: 0.0721 - va
Epoch 2/5
4/4 [==============================] - 9s 2s/step - loss: 3.4853 - accuracy: 0.3694 - va
Epoch 3/5
4/4 [==============================] - 9s 2s/step - loss: 3.4019 - accuracy: 0.3964 - va
Epoch 4/5
4/4 [==============================] - 9s 2s/step - loss: 3.2987 - accuracy: 0.3964 - va
Epoch 5/5
4/4 [==============================] - 9s 2s/step - loss: 3.1587 - accuracy: 0.3964 - va
With optimizer:Adam
Epoch 1/5
4/4 [==============================] - 10s 2s/step - loss: 2.7063 - accuracy: 0.3243 - v
Epoch 2/5
4/4 [==============================] - 9s 2s/step - loss: 1.2312 - accuracy: 0.3874 - va
Epoch 3/5
4/4 [==============================] - 9s 2s/step - loss: 1.4794 - accuracy: 0.3694 - va
Epoch 4/5
4/4 [==============================] - 9s 2s/step - loss: 1.1238 - accuracy: 0.4144 - va
Epoch 5/5
4/4 [==============================] - 9s 2s/step - loss: 1.1867 - accuracy: 0.2523 - va
With optimizer:RMSprop
Epoch 1/5
4/4 [==============================] - 10s 2s/step - loss: 3.1375 - accuracy: 0.2973 - v
Epoch 2/5
4/4 [==============================] - 9s 2s/step - loss: 1.8336 - accuracy: 0.4865 - va
Epoch 3/5
4/4 [==============================] - 9s 2s/step - loss: 1.3576 - accuracy: 0.3153 - va
Epoch 4/5
4/4 [==============================] - 9s 2s/step - loss: 1.2229 - accuracy: 0.3874 - va
Epoch 5/5
4/4 [==============================] - 9s 2s/step - loss: 1.2045 - accuracy: 0.4595 - va
With optimizer:SGD
```

```
Epoch 1/5
4/4 [==============================] - 10s 2s/step - loss: 3.5479 - accuracy: 0.0090 - v
Epoch 2/5
4/4 [==============================] - 9s 2s/step - loss: 3.0675 - accuracy: 0.3964 - va
Epoch 3/5
4/4 [==============================] - 9s 2s/step - loss: 2.1576 - accuracy: 0.3964 - va
Epoch 4/5
4/4 [==============================] - 9s 2s/step - loss: 1.5129 - accuracy: 0.4324 - va
Epoch 5/5
4/4 [==============================] - 9s 2s/step - loss: 1.2342 - accuracy: 0.3964 - va
```

```
data = {'Optimizers': ['Adadelta', 'Adagrad', 'Adam', 'RMSprop', 'SGD'], 'Training_accuracy':
         'Training_loss': [train_loss[0],train_loss[1],train_loss[2],train_loss[3],train_loss
         'Validation_accuracy' : [val_acc[0],val_acc[1],val_acc[2],val_acc[3],val_acc[4]],

         'Validation_loss': [val_loss[0],val_loss[1],val_loss[2],val_loss[3],val_loss[4]],
```

```
df = pd.DataFrame(data)
```

```
df
```

	Optimizers	Training_accuracy	Training_loss	Validation_accuracy	Validation_loss
0	Adadelta	0.207207	3.604986	0.166667	3.612076
1	Adagrad	0.396396	3.158702	0.416667	2.986845
2	Adam	0.252252	1.186702	0.416667	1.091377
3	RMSprop	0.459459	1.204498	0.333333	1.164194
4	SGD	0.396396	1.234197	0.500000	1.395111

```
model = models.Sequential([
    data_preprocessing,
    layers.Conv2D(filters=32, kernel_size= (3, 3), activation='relu',padding ='same', input_s
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(filters=64, kernel_size= (3, 3), activation='relu',padding ='same'),
    layers.MaxPooling2D((3, 3)),
    layers.Conv2D(filters=64, kernel_size= (3, 3), activation='relu',padding ='same'),
    layers.MaxPooling2D((3, 3)),
    layers.Flatten(),
    layers.Dense(64, activation='relu' ),
    layers.Dense(37, activation = 'softmax'),

])
model.build(input_shape = (32,256,256,3))
model.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0         
                                                                 
 conv2d_26 (Conv2D)          (32, 256, 256, 32)        896       
                                                                 
 max_pooling2d_26 (MaxPoolin  (32, 85, 85, 32)         0         
 g2D)                                                            
                                                                 
 conv2d_27 (Conv2D)          (32, 85, 85, 64)          18496     
                                                                 
 max_pooling2d_27 (MaxPoolin  (32, 28, 28, 64)         0         
 g2D)                                                            
                                                                 
 conv2d_28 (Conv2D)          (32, 28, 28, 64)          36928     
                                                                 
 max_pooling2d_28 (MaxPoolin  (32, 9, 9, 64)           0         
 g2D)                                                            
                                                                 
 flatten_8 (Flatten)         (32, 5184)                0         
                                                                 
 dense_16 (Dense)            (32, 64)                  331840    
                                                                 
 dense_17 (Dense)            (32, 37)                  2405      
                                                                 
=================================================================
Total params: 390,565
Trainable params: 390,565
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer= 'Adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits = False),
          metrics=['accuracy'])
```

```
history = model.fit(df_train, epochs= 30, batch_size = 32, validation_data= df_vali)
```

```
Epoch 2/30
4/4 [==============================] - 9s 2s/step - loss: 1.7367 - accuracy: 0.4054 -
Epoch 3/30
4/4 [==============================] - 9s 2s/step - loss: 1.2193 - accuracy: 0.3874 -
Epoch 4/30
4/4 [==============================] - 9s 2s/step - loss: 1.1351 - accuracy: 0.3964 -
Epoch 5/30
4/4 [==============================] - 9s 2s/step - loss: 1.1082 - accuracy: 0.3153 -
Epoch 6/30
4/4 [==============================] - 9s 2s/step - loss: 1.0453 - accuracy: 0.5135 -
Epoch 7/30

4/4 [==============================] - 10s 2s/step - loss: 1.0501 - accuracy: 0.4595
Epoch 8/30
4/4 [==============================] - 9s 2s/step - loss: 1.0429 - accuracy: 0.5225 -
```

```
Epoch 9/30
4/4 [==============================] - 9s 2s/step - loss: 0.9797 - accuracy: 0.6036 -
Epoch 10/30
4/4 [==============================] - 9s 2s/step - loss: 0.9453 - accuracy: 0.5225 -
Epoch 11/30
4/4 [==============================] - 10s 2s/step - loss: 0.8904 - accuracy: 0.6757
Epoch 12/30
4/4 [==============================] - 10s 2s/step - loss: 0.9074 - accuracy: 0.6036
Epoch 13/30
4/4 [==============================] - 10s 2s/step - loss: 0.8639 - accuracy: 0.6126
Epoch 14/30
4/4 [==============================] - 10s 2s/step - loss: 0.8661 - accuracy: 0.6306
Epoch 15/30
4/4 [==============================] - 9s 2s/step - loss: 0.7909 - accuracy: 0.6757 -
Epoch 16/30
4/4 [==============================] - 9s 2s/step - loss: 0.7652 - accuracy: 0.7297 -
Epoch 17/30
4/4 [==============================] - 10s 2s/step - loss: 0.7385 - accuracy: 0.6577
Epoch 18/30
4/4 [==============================] - 10s 2s/step - loss: 0.7435 - accuracy: 0.7117
Epoch 19/30
4/4 [==============================] - 9s 2s/step - loss: 0.7218 - accuracy: 0.7117 -
Epoch 20/30
4/4 [==============================] - 10s 2s/step - loss: 0.6241 - accuracy: 0.7748
Epoch 21/30
4/4 [==============================] - 9s 2s/step - loss: 0.5554 - accuracy: 0.7838 -
Epoch 22/30
4/4 [==============================] - 10s 2s/step - loss: 0.5956 - accuracy: 0.7838
Epoch 23/30
4/4 [==============================] - 10s 2s/step - loss: 0.4504 - accuracy: 0.8468
Epoch 24/30
4/4 [==============================] - 10s 2s/step - loss: 0.5632 - accuracy: 0.7568
Epoch 25/30
4/4 [==============================] - 10s 2s/step - loss: 0.5456 - accuracy: 0.7748
Epoch 26/30
4/4 [==============================] - 10s 2s/step - loss: 0.3842 - accuracy: 0.8829
Epoch 27/30
4/4 [==============================] - 10s 2s/step - loss: 0.3513 - accuracy: 0.8829
Epoch 28/30
4/4 [==============================] - 10s 2s/step - loss: 0.3353 - accuracy: 0.8829
Epoch 29/30
4/4 [==============================] - 9s 2s/step - loss: 0.3873 - accuracy: 0.8649 -
Epoch 30/30
```

Model evaluation

- Evaluating the model by comparing the accuracy and loss in training and validation dataset using callback function(history) and visualizing with matplotlib

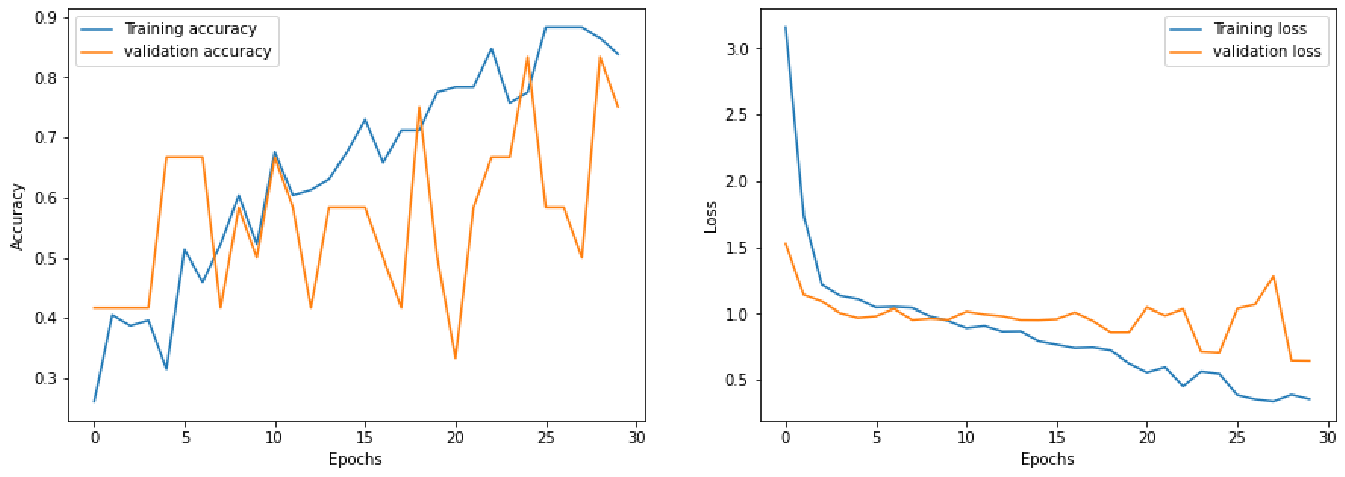
```
train_accu = history.history['accuracy']
val_accu = history.history['val_accuracy']
```

```
train_loss = history.history['loss']
val_loss = history.history['val_loss']
```

- Plotting training and validation accuracy and loss values with respect to each epochs

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(range(30),train_accu,label= 'Training accuracy')
plt.plot(range(30),val_accu,label = 'validation accuracy')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.subplot(1,2,2)
plt.plot(range(30),train_loss,label= 'Training loss')
plt.plot(range(30),val_loss,label = 'validation loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
Text(0, 0.5, 'Loss')
```



Evaluating the test data

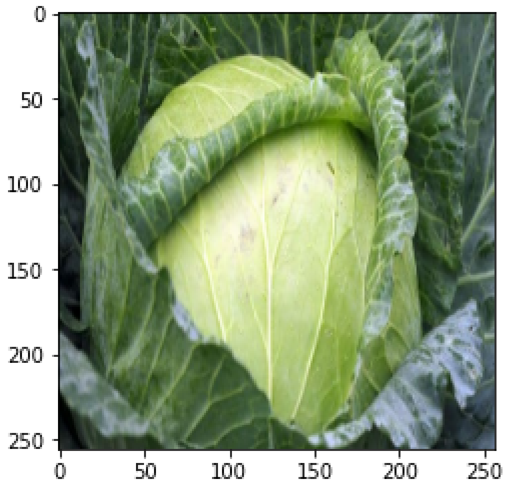
- Using evaluate method to evaluate the test datset and checking predicted and the actual label using predict function

```
model.evaluate(df_test)
```

```
1/1 [==============================] - 5s 5s/step - loss: 0.8487 - accuracy: 0.5000
[0.8486647009849548, 0.5]
```

```
for image, label in df_test.take(1):
  plt.imshow(image[1].numpy().astype('int'))
  print('actual label', classes[label[1]])

  prediction = model.predict(image)
  print('predicted_label',classes[np.argmax(prediction[1])] )
```

```
actual label Healthy
predicted_label Healthy
```



Saving keras model

```
save_path = '/content/drive/MyDrive/thesis_crop_dis_pre/cab_save_model.h5'
model.save(save_path)

model = keras.models.load_model(save_path)
```