Beetroot disese prediction

Table of contents

About the dataset:

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

```
! 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/beetroot", output="/content/dr
     ratio=(.8, .1, .1))

     Copying files: 136 files [00:46,  2.94 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

```
df_train = tf.keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/thesis_crop_dis_pre/beetroot/dataset_bet/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 107 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
```

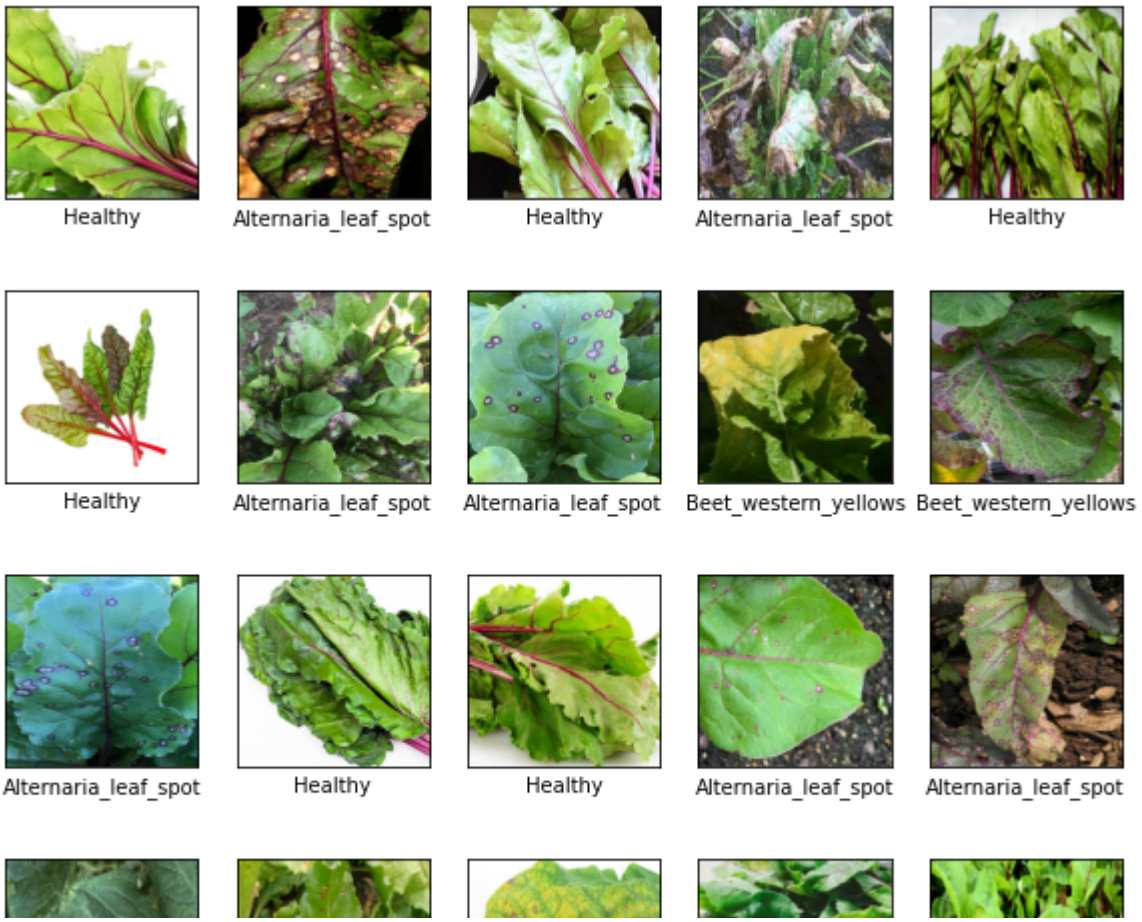
```
['Alternaria_leaf_spot', 'Beet_western_yellows', 'Healthy']
```

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

```
       [2.13828125e+02 2.27265625e+02 2.05953125e+02]
       [2.09952393e+02 2.23952393e+02 2.00952393e+02]
       ...
       [6.91726074e+01 2.91335449e+01 3.01530762e+01]
       [6.58034668e+01 2.46682129e+01 2.62241211e+01]
       [5.28923340e+01 2.08845215e+01 2.18845215e+01]]]


      [[[8.09302979e+01 7.76997070e+01 7.56254883e+01]
        [8.44139404e+01 6.86461792e+01 7.08865967e+01]
        [7.27825317e+01 6.61525269e+01 6.14019775e+01]
        ...
        [8.97628784e+01 8.73594360e+01 8.18566284e+01]
        [8.48298950e+01 8.32008667e+01 7.65817871e+01]
        [7.66216431e+01 7.36223145e+01 7.78482056e+01]]

       [[7.38498535e+01 7.53461914e+01 7.95707397e+01]
        [7.35344238e+01 7.13508301e+01 7.52961426e+01]
        [5.90563354e+01 5.23873291e+01 4.24352417e+01]
        ...
        [7.59206543e+01 7.10316772e+01 6.70925293e+01]
        [7.62002563e+01 7.15361938e+01 7.21963501e+01]
```

```
         [7.75506592e+01 7.17692871e+01 7.83806763e+01]]

        [[4.66647339e+01 4.77406006e+01 4.65017700e+01]
         [5.57771606e+01 5.22648315e+01 5.19367065e+01]
         [6.74901733e+01 6.40358887e+01 7.06574097e+01]
         ...
         [8.29878540e+01 7.57089233e+01 7.75452881e+01]
         [7.84767456e+01 7.28165894e+01 7.38087769e+01]
         [8.09447632e+01 7.19447632e+01 7.69447632e+01]]

        ...

        [[5.36907959e+01 5.10736084e+01 5.36712646e+01]
         [5.15076294e+01 5.02677002e+01 5.10656128e+01]
         [5.75891113e+01 5.44824829e+01 5.98914795e+01]
         ...
         [5.34787598e+01 5.52705688e+01 5.43588257e+01]
         [5.71317749e+01 6.32723999e+01 6.45419312e+01]
         [5.82962646e+01 5.16595459e+01 5.57711792e+01]]

        [[5.94259033e+01 5.27872925e+01 5.24746704e+01]
         [5.40327148e+01 5.49666748e+01 5.08981323e+01]
         [5.29701538e+01 4.94216309e+01 5.06163330e+01]
         ...
         [6.32034912e+01 5.78552246e+01 6.67849121e+01]
         [6.87354126e+01 6.88149414e+01 7.50496826e+01]
         [6.34418945e+01 6.19631958e+01 6.98438721e+01]]

        [[5.07918701e+01 4.58400879e+01 4.71870728e+01]
         [5.09257812e+01 5.40975342e+01 5.09393921e+01]
         [5.67774048e+01 4.58817139e+01 4.98092041e+01]
         ...
         [7.80919189e+01 7.53698730e+01 7.58760376e+01]
         [6.35960083e+01 6.33850708e+01 6.71369629e+01]
         [5.97167969e+01 5.61901245e+01 6.42546387e+01]]]]
      [0 2 2 1 1 0 1 2 0 0 0 1 2 1 0 2 0 0 0 2 0 2 0 2 0 2 0 1 0 2 2 1]
```

```
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

Beet western yellows Alternaria leaf spot Beet western yellows Healthy Healthy

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

```
Found 11 files belonging to 3 classes.
```

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

```
    shuffle=True,)
```

```
Found 17 files belonging to 3 classes.
```

Data preprocessing

Using tensorflows preprocessing layers

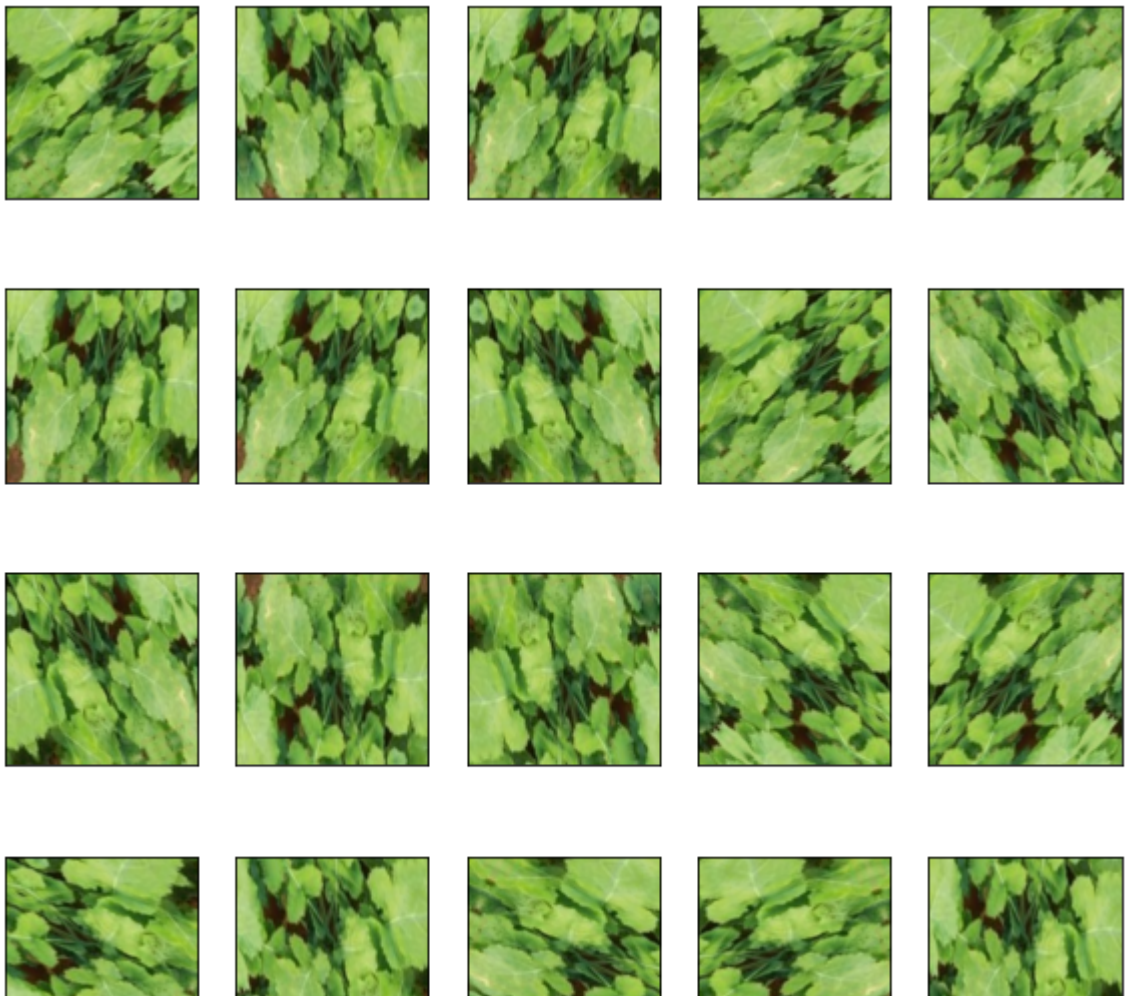
1. Resizing layers- used to change image length and widhth to (256,256)
2. Rescaling layers- to stanadradize the data
3. RandomZoom layers - to randomly zoom in or out on each axis of an image independently

```
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 [==============================] - 8s 2s/step - loss: 3.6170 - accuracy: 0.0000e+00
Epoch 2/5
4/4 [==============================] - 7s 2s/step - loss: 3.6110 - accuracy: 0.0000e+00
Epoch 3/5
4/4 [==============================] - 7s 2s/step - loss: 3.6096 - accuracy: 0.0000e+00
Epoch 4/5
4/4 [==============================] - 7s 2s/step - loss: 3.6053 - accuracy: 0.0000e+00
Epoch 5/5
4/4 [==============================] - 7s 2s/step - loss: 3.6049 - accuracy: 0.0000e+00
With optimizer:Adagrad
Epoch 1/5
4/4 [==============================] - 10s 2s/step - loss: 3.5383 - accuracy: 0.0093 - v
Epoch 2/5
4/4 [==============================] - 7s 2s/step - loss: 3.2045 - accuracy: 0.3364 - va
Epoch 3/5
4/4 [==============================] - 7s 2s/step - loss: 2.8350 - accuracy: 0.2710 - va
Epoch 4/5
4/4 [==============================] - 7s 2s/step - loss: 2.3616 - accuracy: 0.3364 - va
Epoch 5/5
4/4 [==============================] - 7s 2s/step - loss: 1.9066 - accuracy: 0.4299 - va
With optimizer:Adam
Epoch 1/5
4/4 [==============================] - 8s 2s/step - loss: 2.5711 - accuracy: 0.3178 - va
Epoch 2/5
4/4 [==============================] - 7s 2s/step - loss: 1.4043 - accuracy: 0.3738 - va
Epoch 3/5
4/4 [==============================] - 7s 2s/step - loss: 1.2525 - accuracy: 0.3551 - va
Epoch 4/5
4/4 [==============================] - 11s 2s/step - loss: 1.1931 - accuracy: 0.4206 - v
Epoch 5/5
4/4 [==============================] - 7s 2s/step - loss: 1.1652 - accuracy: 0.3271 - va
With optimizer:RMSprop
Epoch 1/5
4/4 [==============================] - 8s 2s/step - loss: 3.5320 - accuracy: 0.2523 - va
Epoch 2/5
4/4 [==============================] - 7s 2s/step - loss: 1.3230 - accuracy: 0.4206 - va
Epoch 3/5
4/4 [==============================] - 7s 2s/step - loss: 1.1452 - accuracy: 0.3364 - va
Epoch 4/5
4/4 [==============================] - 7s 2s/step - loss: 1.0656 - accuracy: 0.4299 - va
Epoch 5/5
4/4 [==============================] - 7s 2s/step - loss: 1.4492 - accuracy: 0.3551 - va
With optimizer:SGD
```

```
Epoch 1/5
4/4 [==============================] - 8s 2s/step - loss: 3.5874 - accuracy: 0.2617 - va
Epoch 2/5
4/4 [==============================] - 7s 2s/step - loss: 3.3037 - accuracy: 0.3925 - va
Epoch 3/5
4/4 [==============================] - 7s 2s/step - loss: 2.6460 - accuracy: 0.4299 - va
Epoch 4/5
4/4 [==============================] - 8s 2s/step - loss: 1.7267 - accuracy: 0.4299 - va
Epoch 5/5
4/4 [==============================] - 7s 2s/step - loss: 1.2867 - accuracy: 0.4299 - 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	0.000000	3.604947	0.000000	3.609094
1	Adagrad	0.429907	1.906603	0.363636	1.716519
2	Adam	0.327103	1.165200	0.454545	1.102630
3	RMSprop	0.355140	1.449209	0.363636	1.105333
4	SGD	0.429907	1.286734	0.363636	1.344995

```
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.build(input_shape = (32,256,256,3))


model.summary()
```

```
model.summary()
```

```
Model: "sequential_15"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_8 (Sequential)   (None, 256, 256, 3)       0

 conv2d_33 (Conv2D)          (32, 254, 254, 32)        896

 max_pooling2d_33 (MaxPoolin  (32, 84, 84, 32)         0
 g2D)

 conv2d_34 (Conv2D)          (32, 82, 82, 64)          18496

 max_pooling2d_34 (MaxPoolin  (32, 27, 27, 64)         0
 g2D)

 conv2d_35 (Conv2D)          (32, 25, 25, 64)          36928

 max_pooling2d_35 (MaxPoolin  (32, 8, 8, 64)           0
 g2D)

 flatten_11 (Flatten)        (32, 4096)                0

 dense_22 (Dense)            (32, 64)                  262208

 dense_23 (Dense)            (32, 37)                  2405

=================================================================
Total params: 320,933
Trainable params: 320,933
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 [==============================] - 7s 2s/step - loss: 1.1353 - accuracy: 0.4486 -
Epoch 3/30
4/4 [==============================] - 7s 2s/step - loss: 1.4043 - accuracy: 0.3271 -
Epoch 4/30
4/4 [==============================] - 7s 2s/step - loss: 1.2861 - accuracy: 0.2897 -
Epoch 5/30
4/4 [==============================] - 7s 2s/step - loss: 1.1032 - accuracy: 0.4019 -
Epoch 6/30
4/4 [==============================] - 7s 2s/step - loss: 1.0716 - accuracy: 0.4486 -
Epoch 7/30

4/4 [==============================] - 7s 2s/step - loss: 1.0264 - accuracy: 0.4299 -
Epoch 8/30
4/4 [==============================] - 7s 2s/step - loss: 1.0496 - accuracy: 0.4766 -
```

```
Epoch 9/30
4/4 [==============================] - 7s 2s/step - loss: 0.9874 - accuracy: 0.5701 -
Epoch 10/30
4/4 [==============================] - 7s 2s/step - loss: 0.9069 - accuracy: 0.5607 -
Epoch 11/30
4/4 [==============================] - 7s 2s/step - loss: 0.8522 - accuracy: 0.5981 -
Epoch 12/30
4/4 [==============================] - 7s 2s/step - loss: 0.8941 - accuracy: 0.6075 -
Epoch 13/30
4/4 [==============================] - 7s 2s/step - loss: 0.7558 - accuracy: 0.5981 -
Epoch 14/30
4/4 [==============================] - 7s 2s/step - loss: 0.7951 - accuracy: 0.5981 -
Epoch 15/30
4/4 [==============================] - 7s 2s/step - loss: 0.6957 - accuracy: 0.6822 -
Epoch 16/30
4/4 [==============================] - 7s 2s/step - loss: 0.6460 - accuracy: 0.7009 -
Epoch 17/30
4/4 [==============================] - 7s 2s/step - loss: 0.6925 - accuracy: 0.6822 -
Epoch 18/30
4/4 [==============================] - 7s 2s/step - loss: 0.5639 - accuracy: 0.7290 -
Epoch 19/30
4/4 [==============================] - 7s 2s/step - loss: 0.5539 - accuracy: 0.7477 -
Epoch 20/30
4/4 [==============================] - 7s 2s/step - loss: 0.4954 - accuracy: 0.8131 -
Epoch 21/30
4/4 [==============================] - 7s 2s/step - loss: 0.4069 - accuracy: 0.8318 -
Epoch 22/30
4/4 [==============================] - 7s 2s/step - loss: 0.3980 - accuracy: 0.8972 -
Epoch 23/30
4/4 [==============================] - 7s 2s/step - loss: 0.4123 - accuracy: 0.8411 -
Epoch 24/30
4/4 [==============================] - 7s 2s/step - loss: 0.3227 - accuracy: 0.8692 -
Epoch 25/30
4/4 [==============================] - 7s 2s/step - loss: 0.2771 - accuracy: 0.8879 -
Epoch 26/30
4/4 [==============================] - 7s 2s/step - loss: 0.3603 - accuracy: 0.8411 -
Epoch 27/30
4/4 [==============================] - 7s 2s/step - loss: 0.2837 - accuracy: 0.9159 -
Epoch 28/30
4/4 [==============================] - 7s 2s/step - loss: 0.1995 - accuracy: 0.9346 -
Epoch 29/30
4/4 [==============================] - 7s 2s/step - loss: 0.2230 - accuracy: 0.9252 -
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

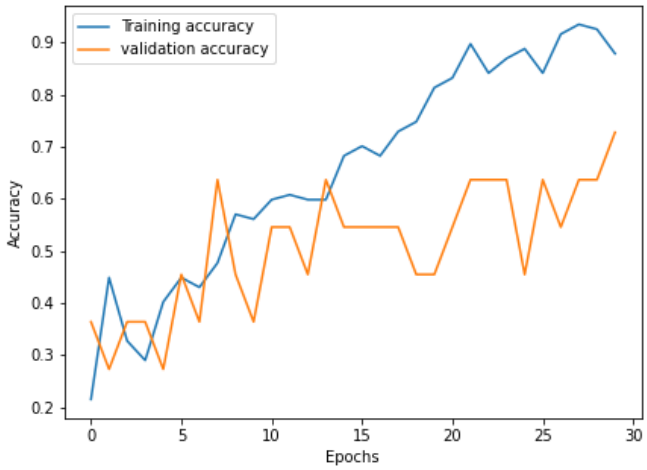
```
train_accu = history.history['accuracy']
val_accu = history.history['val_accuracy']
```

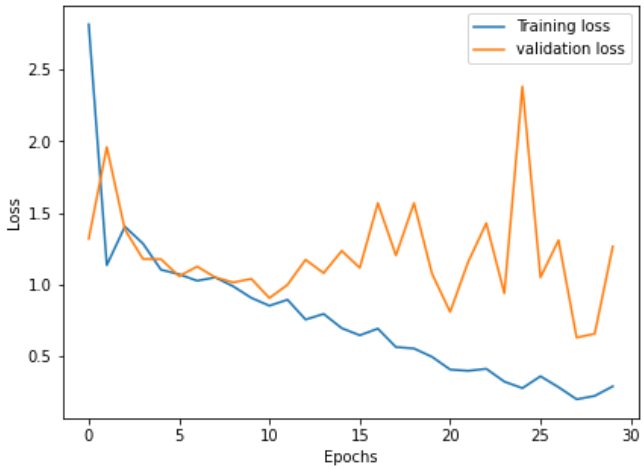
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']
```

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')
```





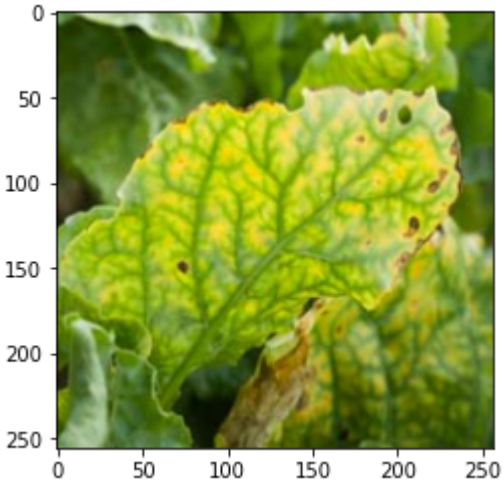

```
model.evaluate(df_test)
```

```
1/1 [==============================] - 4s 4s/step - loss: 0.6772 - accuracy: 0.7059
[0.6772037744522095, 0.7058823704719543]
```

```
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 Beet_western_yellows
predicted_label Beet_western_yellows
```



Saving keras model

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

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

Colab paid products - Cancel contracts here

✓ 0s completed at 6:19 PM