Carrot diseases prediction

Table of contents

1. About the dataset
2. Importing libraries
3. Exploratory data analysis
4. Data preprocessing
5. Model building
6. Model evaluation

About the dataset:

Dataset include images of 'Life_spot' and 'Healthy' carrot 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/carrot", output="/content/driv
     ratio=(.8, .1, .1))
```

```
Copying files: 91 files [00:43,  2.09 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/carrot/dataset_carrot/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 72 files belonging to 2 classes.
```

```
len(df_train)
```

```
3
```

- Using dir function to check the builtin properties and methods of df_train

```
dir(df_train)
```

```
 '_setattr_tracking',
 '_shape_invariant_to_type_spec',
 '_single_restoration_from_checkpoint_position',
 '_structure',
 '_tf_api_names',
 '_tf_api_names_v1',
 '_trace_variant_creation',
 '_track_trackable',
 '_trackable_children',
 '_type_spec',
 '_unconditional_checkpoint_dependencies',
 '_unconditional_dependency_names',
 '_update_uid',
 '_variant_tensor',
 '_variant_tensor_attr',
 'apply',
 'as_numpy_iterator',
 'batch',
 'bucket_by_sequence_length',
 'cache',
 'cardinality',
 'choose_from_datasets',
 'class_names',
 'concatenate',
 'element_spec',
 'enumerate',
```

```
 'file_paths',
 'filter',
 'flat_map',
 'from_generator',
 'from_tensor_slices',
 'from_tensors',
 'get_single_element',
 'group_by_window',
 'interleave',
 'list_files',
 'map',
 'options',
 'padded_batch',
 'prefetch',
 'random',
 'range',
 'reduce',
 'rejection_resample',
 'repeat',
 'sample_from_datasets',
 'scan',
 'shard',
 'shuffle',
 'skip',
 'snapshot',
 'take',
 'take_while',
 'unbatch',
 'unique',
 'window',
 'with_options',
 'zip']
```

- 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
```

```
['Healthy', 'Life_spot']
```

```
len(classes)
```

```
2
```

```
len(df_train)
```

```
3
```

- As the images are loaded as batches, iterating over the first batch of images using take(1) method and retrieving the image and the label batches as tensorflow object. The image batch will 32 images of shape (256 * 256 * 3) and the label batch will have corresponding labels of 32 images
- Using numpy() function to convert tensorflow object to numpy array

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

```
   [5.02203369e+00 4.05328369e+00 9.05328369e+00]
   [6.43023682e+00 1.04302368e+01 1.34302368e+01]
   ...
   [8.17879028e+01 1.07787903e+02 8.15066528e+01]
   [7.62324829e+01 1.05232483e+02 7.42481079e+01]
   [7.59785767e+01 1.07306702e+02 7.23223267e+01]]]


 [[[5.63680725e+01 9.53680725e+01 2.36807251e+00]
   [5.65351562e+01 9.65351562e+01 0.00000000e+00]
   [6.79499969e+01 1.10949997e+02 3.88671875e+00]
   ...
   [1.15639328e+02 1.55596878e+02 3.63075562e+01]
   [6.52214050e+01 8.58398438e+01 5.34921875e+01]
   [8.12168274e+01 1.09167557e+02 5.12955780e+01]]

  [[5.36959381e+01 9.46959381e+01 0.00000000e+00]
   [5.67929688e+01 9.87929688e+01 0.00000000e+00]
   [6.76117249e+01 1.10611725e+02 4.20858765e+00]
   ...
   [1.13425003e+02 1.56186722e+02 2.58359222e+01]
   [5.22627869e+01 7.66963806e+01 2.65279541e+01]
   [1.05340622e+02 1.35895309e+02 6.82090607e+01]]

  [[5.46445312e+01 9.86520844e+01 6.36978149e-01]
   [5.52046814e+01 1.00204681e+02 1.48315430e-02]
   [7.45013275e+01 1.20510437e+02 1.17031555e+01]
   ...
   [1.09794281e+02 1.52782562e+02 2.03046875e+01]
   [6.86765900e+01 1.00964874e+02 2.41797028e+01]
   [1.18892426e+02 1.54220810e+02 7.06885223e+01]]

  ...

  [[8.59987030e+01 9.09987030e+01 8.69987030e+01]
   [8.60562592e+01 9.10562592e+01 8.50562592e+01]
   [7.98450775e+01 8.18450775e+01 7.68450775e+01]
   ...
   [1.02475250e+02 1.07475250e+02 1.03475250e+02]
   [9.92179871e+01 1.05217987e+02 1.01217987e+02]
   [8.30661469e+01 9.20661469e+01 8.70661469e+01]]

  [[9.27629700e+01 9.77629700e+01 9.37629700e+01]
   [7.50678253e+01 8.00678253e+01 7.40678253e+01]
   [8.39710846e+01 8.59710846e+01 8.09710846e+01]
```

```
    ...
    [9.40664062e+01 9.90664062e+01 9.50664062e+01]
    [9.67012482e+01 1.04922180e+02 1.00181870e+02]
    [7.75403137e+01 8.65403137e+01 8.15403137e+01]]

   [[9.00531769e+01 9.50531769e+01 9.10531769e+01]
    [7.19037476e+01 7.69037476e+01 7.09037476e+01]
    [7.57500153e+01 7.77500153e+01 7.27500153e+01]
    ...
    [9.91244812e+01 1.04124481e+02 1.00124481e+02]
    [9.16679688e+01 1.00468750e+02 9.55351562e+01]
    [7.24390106e+01 8.34390106e+01 7.74390106e+01]]]]
 [0 1 0 1 0 1 0 1 1 0 0 1 0 0 1 1 0 1 1 0 1 0 0 0 0 0 0 1 0 0 1 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

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

```
Found 8 files belonging to 2 classes.
```

```
df_test = tf.keras.utils.image_dataset_from_directory(
    directory = '/content/drive/MyDrive/thesis_crop_dis_pre/carrot/dataset_carrot/test',
    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 2 classes.
```

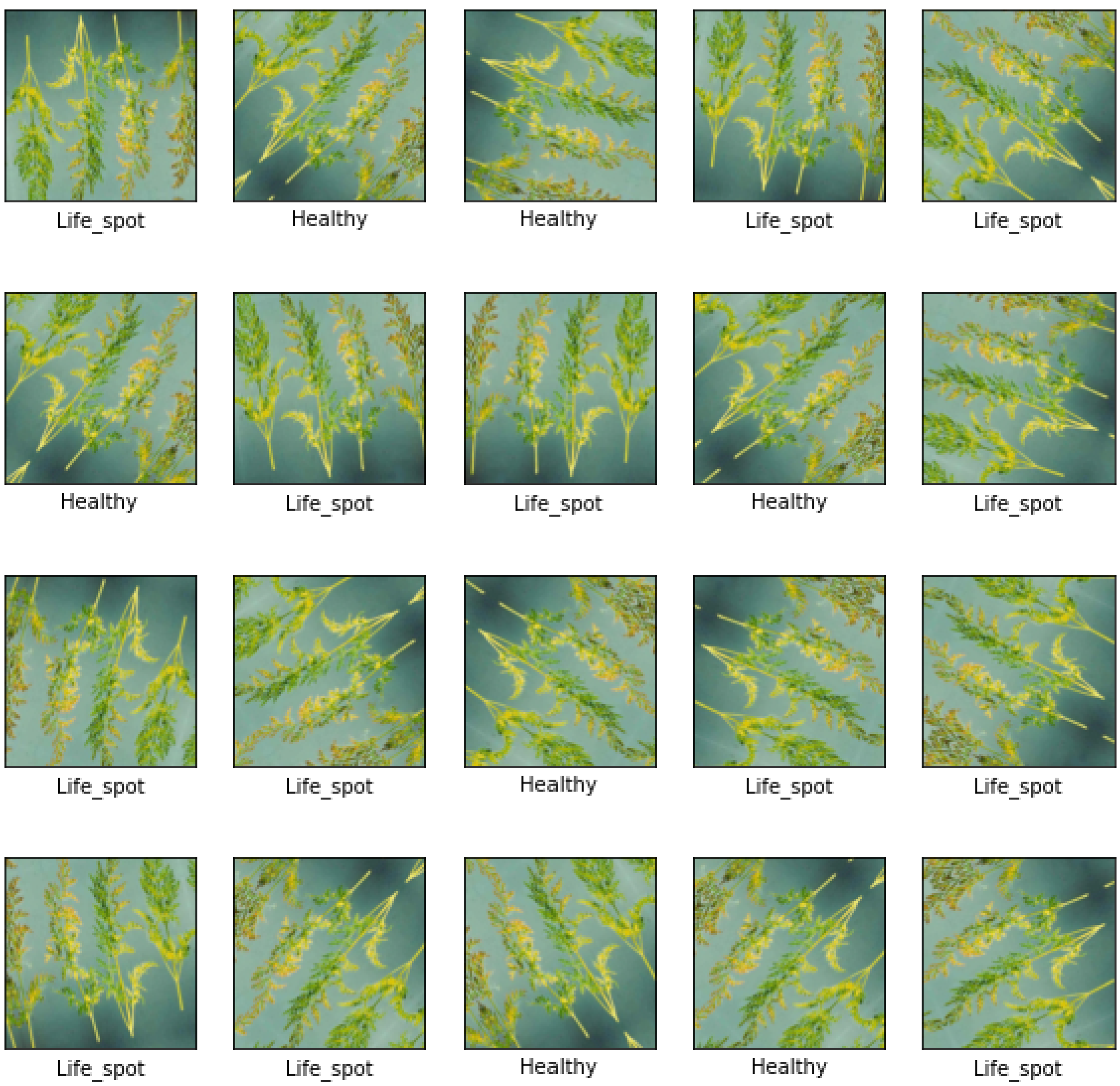
Data preprocessing

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),

                    ])


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.xlabel(classes[label[i]])
  plt.show()
```



Model building

Within the sequential layers having

data_preprocessing as the first layer consisting of resizing, rescaling and randomzoom preprocessing layers

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

Within the dense network Having a flatten layer, which helps to reduce the dimensionality of the input to single dimension

Followed by flatten layer having two deep layers where the first layer has 64 neurons and relu as its activation function, and the second is the output layer with 37 output categories and softmax as its activation function since it normalises the output

As CNN layers are used, there is not much necessary to use more dense layers CNN layers itself does the job

Have also tried different optimizer values

Using SparseCategoricalCrossentropy as loss function, since the traget value is not onehotencoded, also setting the from_logits = False as use softmax functione loss is normalised

Using accuracy as the metric to evaluate the model

Have built the model within a function to try different optimizers

```
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 = []


  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
3/3 [==============================] - 15s 1s/step - loss: 3.5079 - accuracy: 0.4167 - v
Epoch 2/5
3/3 [==============================] - 1s 86ms/step - loss: 3.5034 - accuracy: 0.4306 -
Epoch 3/5
3/3 [==============================] - 1s 84ms/step - loss: 3.5000 - accuracy: 0.4167 -
Epoch 4/5
3/3 [==============================] - 1s 85ms/step - loss: 3.4992 - accuracy: 0.4306 -
Epoch 5/5
3/3 [==============================] - 1s 85ms/step - loss: 3.4941 - accuracy: 0.4722 -
With optimizer:Adagrad
Epoch 1/5
3/3 [==============================] - 2s 135ms/step - loss: 3.5848 - accuracy: 0.0000e
Epoch 2/5
3/3 [==============================] - 1s 84ms/step - loss: 3.5422 - accuracy: 0.0139 -
Epoch 3/5
3/3 [==============================] - 1s 82ms/step - loss: 3.4977 - accuracy: 0.0278 -
```

```
Epoch 4/5
3/3 [==============================] - 1s 86ms/step - loss: 3.4433 - accuracy: 0.2083 -
Epoch 5/5
3/3 [==============================] - 1s 85ms/step - loss: 3.3828 - accuracy: 0.4167 -
With optimizer:Adam
Epoch 1/5
3/3 [==============================] - 2s 138ms/step - loss: 3.3070 - accuracy: 0.3611 -
Epoch 2/5
3/3 [==============================] - 1s 88ms/step - loss: 1.2793 - accuracy: 0.5139 -
Epoch 3/5
3/3 [==============================] - 1s 88ms/step - loss: 0.8334 - accuracy: 0.4861 -
Epoch 4/5
3/3 [==============================] - 1s 83ms/step - loss: 0.8441 - accuracy: 0.5139 -
Epoch 5/5
3/3 [==============================] - 1s 87ms/step - loss: 0.5958 - accuracy: 0.8611 -
With optimizer:RMSprop
Epoch 1/5
3/3 [==============================] - 2s 149ms/step - loss: 2.4590 - accuracy: 0.3056 -
Epoch 2/5
3/3 [==============================] - 1s 86ms/step - loss: 2.1143 - accuracy: 0.5000 -
Epoch 3/5
3/3 [==============================] - 1s 85ms/step - loss: 0.7238 - accuracy: 0.5833 -
Epoch 4/5
3/3 [==============================] - 1s 85ms/step - loss: 0.7278 - accuracy: 0.5417 -
Epoch 5/5
3/3 [==============================] - 1s 89ms/step - loss: 0.5658 - accuracy: 0.8194 -
With optimizer:SGD
Epoch 1/5
3/3 [==============================] - 2s 147ms/step - loss: 3.5018 - accuracy: 0.1528 -
Epoch 2/5
3/3 [==============================] - 1s 83ms/step - loss: 2.7736 - accuracy: 0.4861 -
Epoch 3/5
3/3 [==============================] - 1s 85ms/step - loss: 1.4652 - accuracy: 0.5000 -
Epoch 4/5
3/3 [==============================] - 1s 89ms/step - loss: 1.0808 - accuracy: 0.5139 -
Epoch 5/5
3/3 [==============================] - 1s 85ms/step - loss: 1.2758 - accuracy: 0.3889 -
```

```
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	0.472222	3.494104	0.5	3.501525
1	Adagrad	0.416667	3.382754	0.5	3.315003
2	Adam	0.861111	0.595817	0.5	0.859721
3	RMSprop	0.819444	0.565844	0.5	0.937859

```
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: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 256, 256, 3)       0

 conv2d_15 (Conv2D)          (32, 254, 254, 32)        896

 max_pooling2d_15 (MaxPoolin  (32, 84, 84, 32)         0
 g2D)

 conv2d_16 (Conv2D)          (32, 82, 82, 64)          18496

 max_pooling2d_16 (MaxPoolin  (32, 27, 27, 64)         0
 g2D)

 conv2d_17 (Conv2D)          (32, 25, 25, 64)          36928

 max_pooling2d_17 (MaxPoolin  (32, 8, 8, 64)           0
 g2D)

 flatten_5 (Flatten)         (32, 4096)                0

 dense_10 (Dense)            (32, 64)                  262208

 dense_11 (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
3/3 [==============================] - 1s 88ms/step - loss: 0.4026 - accuracy: 0.8611
Epoch 3/30
3/3 [==============================] - 1s 82ms/step - loss: 0.3505 - accuracy: 0.8750
Epoch 4/30
3/3 [==============================] - 1s 88ms/step - loss: 0.3199 - accuracy: 0.8889
Epoch 5/30
3/3 [==============================] - 1s 87ms/step - loss: 0.2462 - accuracy: 0.9167
Epoch 6/30
3/3 [==============================] - 1s 82ms/step - loss: 0.2167 - accuracy: 0.9167
Epoch 7/30
3/3 [==============================] - 1s 85ms/step - loss: 0.1943 - accuracy: 0.9444
Epoch 8/30
3/3 [==============================] - 1s 82ms/step - loss: 0.2146 - accuracy: 0.9167
Epoch 9/30
3/3 [==============================] - 1s 88ms/step - loss: 0.1568 - accuracy: 0.9444
Epoch 10/30
3/3 [==============================] - 1s 86ms/step - loss: 0.1952 - accuracy: 0.9028
Epoch 11/30
3/3 [==============================] - 1s 81ms/step - loss: 0.2088 - accuracy: 0.9444
Epoch 12/30
3/3 [==============================] - 1s 85ms/step - loss: 0.1863 - accuracy: 0.9167
Epoch 13/30
3/3 [==============================] - 1s 85ms/step - loss: 0.2906 - accuracy: 0.8472
Epoch 14/30
3/3 [==============================] - 1s 86ms/step - loss: 0.1851 - accuracy: 0.9444
Epoch 15/30
3/3 [==============================] - 1s 88ms/step - loss: 0.2899 - accuracy: 0.8889
Epoch 16/30
3/3 [==============================] - 1s 89ms/step - loss: 0.1380 - accuracy: 0.9722
Epoch 17/30
3/3 [==============================] - 1s 91ms/step - loss: 0.1934 - accuracy: 0.9167
Epoch 18/30
3/3 [==============================] - 1s 85ms/step - loss: 0.1045 - accuracy: 0.9861
Epoch 19/30
3/3 [==============================] - 1s 89ms/step - loss: 0.1106 - accuracy: 0.9722
Epoch 20/30
3/3 [==============================] - 1s 85ms/step - loss: 0.0948 - accuracy: 0.9861
Epoch 21/30
3/3 [==============================] - 1s 89ms/step - loss: 0.0753 - accuracy: 0.9722
Epoch 22/30
3/3 [==============================] - 1s 88ms/step - loss: 0.0557 - accuracy: 0.9861
Epoch 23/30
3/3 [==============================] - 1s 87ms/step - loss: 0.0416 - accuracy: 0.9861
```

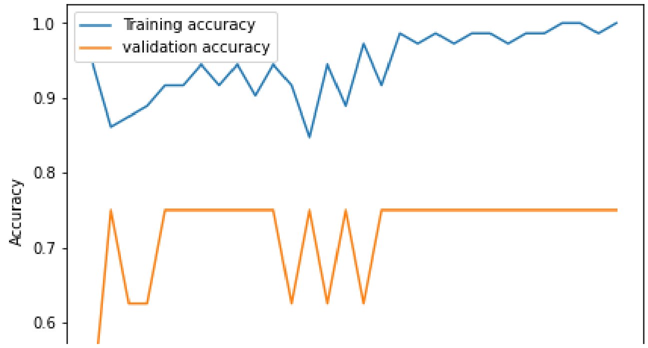
```
Epoch 24/30
3/3 [==============================] - 1s 87ms/step - loss: 0.0523 - accuracy: 0.9722
Epoch 25/30
3/3 [==============================] - 1s 86ms/step - loss: 0.0473 - accuracy: 0.9861
Epoch 26/30
3/3 [==============================] - 1s 88ms/step - loss: 0.0361 - accuracy: 0.9861
Epoch 27/30
3/3 [==============================] - 1s 88ms/step - loss: 0.0485 - accuracy: 1.0000
Epoch 28/30
3/3 [==============================] - 1s 86ms/step - loss: 0.0255 - accuracy: 1.0000
Epoch 29/30
3/3 [==============================] - 1s 82ms/step - loss: 0.0551 - accuracy: 0.9861
Epoch 30/30
3/3 [==============================] - 1s 85ms/step - loss: 0.0254 - accuracy: 1.0000
```

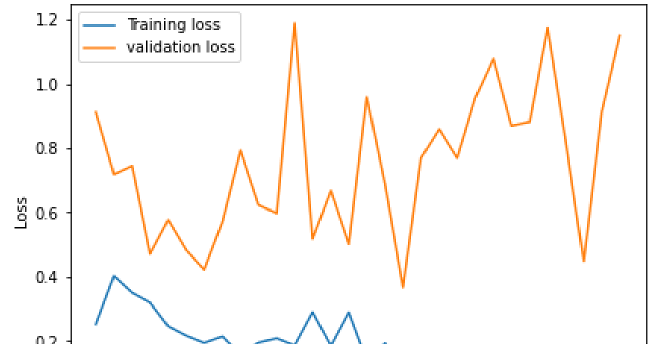
Model evaluation

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']


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 [==============================] - 3s 3s/step - loss: 0.1912 - accuracy: 0.9091
[0.19118906557559967, 0.9090909361839294]
```

```
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
```


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

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