Potato disese prediction

Table of contents

About the dataset:

Dataset include images of 'Early_bligh','Late_blight' and 'Healthy' potato leaves. The images are collected from kaggle.

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.

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

```
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/potato", output="/content/driv
     ratio=(.8, .1, .1))
```

```
     Copying files: 2152 files [00:51, 42.16 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/potato/dataset_pot/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 1721 files belonging to 3 classes.
```

- 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
```

```
['Early_blight', 'Healthy', 'Late_blight']
```

```
len(classes)
```

```
3
```

```
len(df_train)
```

```
54
```

```
54*32
```

```
1728
```

- 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())
```

```
       [130. 124. 136.]
       [ 70.  64.  76.]
       ...
       [110. 108. 121.]
       [154. 152. 165.]
       [110. 108. 121.]]]


     [[[156. 155. 161.]
       [163. 162. 168.]
       [175. 174. 180.]
       ...
       [194. 193. 201.]
       [197. 196. 204.]
       [200. 199. 207.]]

      [[165. 164. 170.]
       [174. 173. 179.]
       [183. 182. 188.]
       ...
       [193. 192. 200.]
       [195. 194. 202.]
       [198. 197. 205.]]

      [[169. 168. 174.]
       [173. 172. 178.]
       [176. 175. 181.]
       ...
       [194. 193. 201.]
       [195. 194. 202.]
       [196. 195. 203.]]

      ...

      [[165. 163. 176.]
       [164. 162. 175.]
       [159. 157. 170.]
       ...
       [165. 163. 177.]
       [166. 164. 178.]
       [166. 164. 178.]]
```

```
          [[142. 140. 153.]
           [148. 146. 159.]
           [151. 149. 162.]
           ...
           [169. 167. 181.]
           [169. 167. 181.]
           [169. 167. 181.]]

          [[139. 137. 150.]
           [148. 146. 159.]
           [156. 154. 167.]
           ...
           [173. 171. 185.]
           [173. 171. 185.]
           [173. 171. 185.]]]]
       [0 2 2 2 1 0 2 2 2 0 1 2 0 1 2 2 0 2 2 0 2 0 1 0 1 0 2 0 0 2 0 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/potato/dataset_pot/val',
    labels='inferred',
    label_mode='int',
    class_names=None,
    color_mode='rgb',
    batch_size= batch_size,
    image_size= image_size,
    shuffle=True,
)
```

```
 Found 215 files belonging to 3 classes.
```

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

```
 Found 216 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
- 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,
    data_augumentation,
    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
```

- 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

```
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
54/54 [==============================] - 165s 3s/step - loss: 3.6084 - accuracy: 0.0000e
Epoch 2/5
54/54 [==============================] - 116s 2s/step - loss: 3.5585 - accuracy: 0.0000e
Epoch 3/5
54/54 [==============================] - 116s 2s/step - loss: 3.5118 - accuracy: 0.0128
Epoch 4/5
54/54 [==============================] - 116s 2s/step - loss: 3.4636 - accuracy: 0.2719
Epoch 5/5
54/54 [==============================] - 116s 2s/step - loss: 3.4113 - accuracy: 0.5026
With optimizer:Adagrad
Epoch 1/5
54/54 [==============================] - 117s 2s/step - loss: 2.5076 - accuracy: 0.3539
Epoch 2/5
54/54 [==============================] - 116s 2s/step - loss: 0.9509 - accuracy: 0.4956
Epoch 3/5
54/54 [==============================] - 115s 2s/step - loss: 0.9032 - accuracy: 0.5137
Epoch 4/5
54/54 [==============================] - 114s 2s/step - loss: 0.8905 - accuracy: 0.5293
Epoch 5/5
54/54 [==============================] - 118s 2s/step - loss: 0.8735 - accuracy: 0.5520
With optimizer:Adam
Epoch 1/5
54/54 [==============================] - 119s 2s/step - loss: 1.0379 - accuracy: 0.5468
Epoch 2/5
54/54 [==============================] - 119s 2s/step - loss: 0.6367 - accuracy: 0.7571
Epoch 3/5
54/54 [==============================] - 120s 2s/step - loss: 0.4935 - accuracy: 0.8053
Epoch 4/5
54/54 [==============================] - 134s 2s/step - loss: 0.3232 - accuracy: 0.8704
Epoch 5/5
54/54 [==============================] - 125s 2s/step - loss: 0.2973 - accuracy: 0.8733
With optimizer:RMSprop
Epoch 1/5
54/54 [==============================] - 128s 2s/step - loss: 1.0995 - accuracy: 0.5445
Epoch 2/5
54/54 [==============================] - 126s 2s/step - loss: 0.6270 - accuracy: 0.7600
Epoch 3/5
54/54 [==============================] - 120s 2s/step - loss: 0.4610 - accuracy: 0.8059
Epoch 4/5
54/54 [==============================] - 121s 2s/step - loss: 0.3999 - accuracy: 0.8298
Epoch 5/5
54/54 [==============================] - 122s 2s/step - loss: 0.3580 - accuracy: 0.8611
With optimizer:SGD
```

```
Epoch 1/5
54/54 [==============================] - 125s 2s/step - loss: 1.3687 - accuracy: 0.4399
Epoch 2/5
54/54 [==============================] - 127s 2s/step - loss: 0.9290 - accuracy: 0.4840
Epoch 3/5
54/54 [==============================] - 130s 2s/step - loss: 0.8940 - accuracy: 0.5293
Epoch 4/5
54/54 [==============================] - 125s 2s/step - loss: 0.8989 - accuracy: 0.5357
Epoch 5/5
54/54 [==============================] - 129s 2s/step - loss: 0.8767 - accuracy: 0.5723
```

- It is observed that adam optimizer performs better
- I have tried to train the model with 10 epochs, but the accuracies were consitent even from the third epoch and did not have any change till 10th. So i decided to run only for 5 epochs.
- Created a dataframe out of these results to compare the accuracy and loss values

```
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	0.502615	3.411250	0.516279	3.351572
1	Adagrad	0.552005	0.873510	0.479070	0.882518
2	Adam	0.873329	0.297284	0.925581	0.237606
3	RMSprop	0.861127	0.358046	0.795349	0.569786
4	SGD	0.572342	0.876666	0.465116	1.112441

- Since adam gives better accuracy training the model and checking for accuracy with the same

```
model = models.Sequential([
    data_preprocessing,
    data_augumentation,
    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() shows all the layers (networks) used in the model

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2D  (32, 84, 84, 32)         0
 )

 conv2d_1 (Conv2D)           (32, 82, 82, 64)          18496

 max_pooling2d_1 (MaxPooling  (32, 27, 27, 64)         0
 2D)

 conv2d_2 (Conv2D)           (32, 25, 25, 64)          36928

 max_pooling2d_2 (MaxPooling  (32, 8, 8, 64)           0
 2D)

 flatten (Flatten)           (32, 4096)                0

 dense (Dense)               (32, 64)                  262208

 dense_1 (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'])
```

Using train dataset to fit the model providing the validatioin dataset to validate with 10 epochs.

I am storing the results within the history variable which could be used to visualise the results

```
history = model.fit(df_train, epochs= 10, batch_size = 32, validation_data= df_vali)
```

```
Epoch 1/10
54/54 [==============================] - 141s 2s/step - loss: 1.0255 - accuracy: 0.5352
Epoch 2/10
54/54 [==============================] - 5s 83ms/step - loss: 0.5623 - accuracy: 0.7972
Epoch 3/10
54/54 [==============================] - 5s 81ms/step - loss: 0.4754 - accuracy: 0.8100
Epoch 4/10
54/54 [==============================] - 5s 82ms/step - loss: 0.3376 - accuracy: 0.8605
Epoch 5/10
54/54 [==============================] - 5s 81ms/step - loss: 0.3199 - accuracy: 0.8681
Epoch 6/10
54/54 [==============================] - 5s 82ms/step - loss: 0.2551 - accuracy: 0.8925
Epoch 7/10
54/54 [==============================] - 5s 83ms/step - loss: 0.2181 - accuracy: 0.9082
Epoch 8/10
54/54 [==============================] - 5s 82ms/step - loss: 0.1990 - accuracy: 0.9175
Epoch 9/10
54/54 [==============================] - 5s 81ms/step - loss: 0.1720 - accuracy: 0.9326
Epoch 10/10
54/54 [==============================] - 6s 103ms/step - loss: 0.1687 - accuracy: 0.9378
```

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']
```

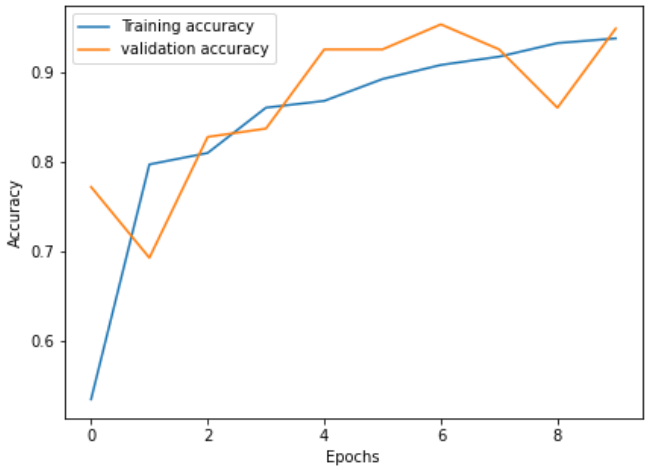
- Plotting training and validation accuracy and loss values with respect to each epochs

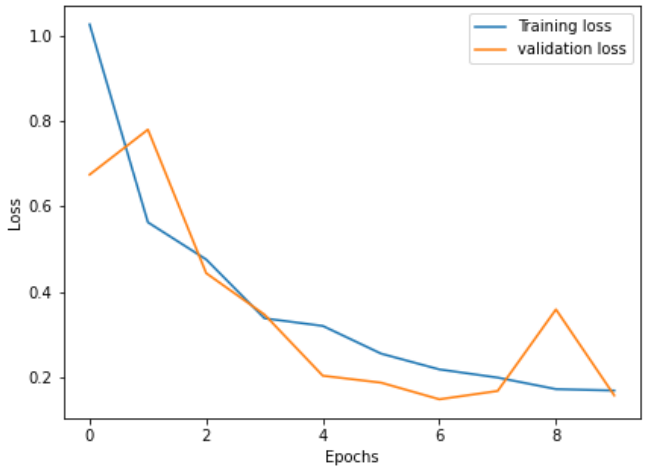
```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)
plt.plot(range(10),train_accu,label= 'Training accuracy')
plt.plot(range(10),val_accu,label = 'validation accuracy')
plt.legend()
```

```
plt.xlabel('Epochs')
plt.ylabel('Accuracy')

plt.subplot(1,2,2)
plt.plot(range(10),train_loss,label= 'Training loss')
plt.plot(range(10),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

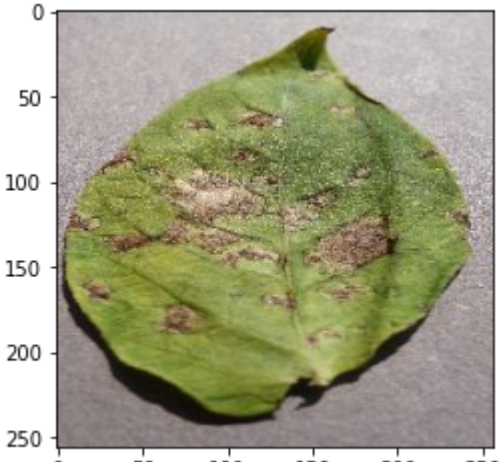
```
model.evaluate(df_test)
```

```
7/7 [==============================] - 26s 58ms/step - loss: 0.1119 - accuracy: 0.9444
[0.11185561120510101, 0.9444444179534912]
```

```
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 Early_blight
predicted_label Early_blight
```



Saving keras model

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

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

Colab paid products - Cancel contracts here