Potato Leaf Disease Classification

Import all the Dependencies

```
import numpy as np
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras import layers, Sequential, models
import pathlib

from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 7182255403736550898
xla_global_id: -1
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 14415560704
locality {
  bus_id: 1
  links {
  }
}
incarnation: 1598529688220453900
physical_device_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0, compute capability: 7.5"
xla_global_id: 416903419
]
```

Set all the Constants

```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=20
```

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

```
Mounted at /content/drive
```

Import data into tensorflow dataset object

```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "/content/drive/MyDrive/content/drive/Training",
    seed=12,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

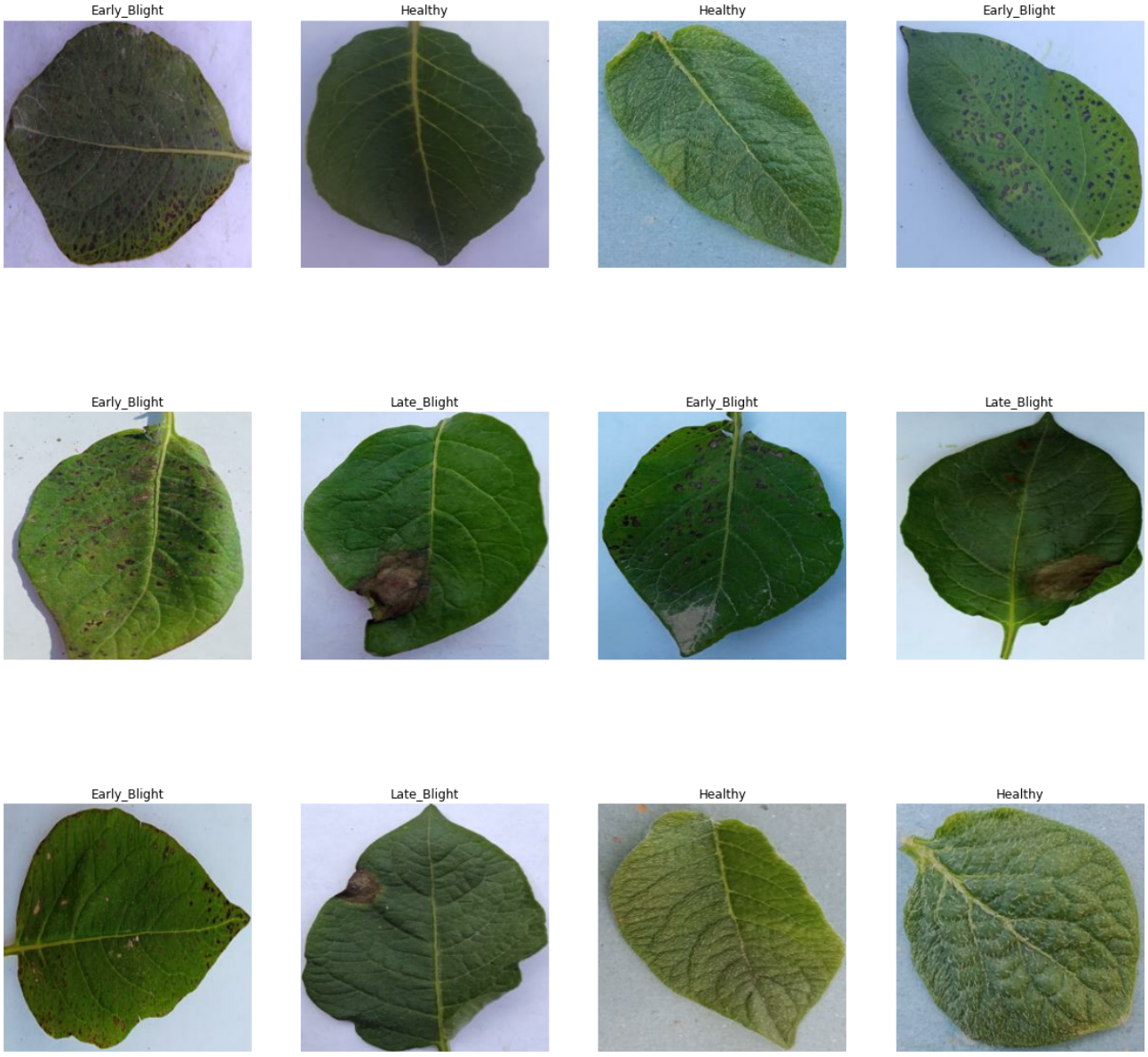
```
Found 3666 files belonging to 3 classes.
```

```
class_names = dataset.class_names
class_names
```

```
['Early_Blight', 'Healthy', 'Late_Blight']
```

Visualize some of the images from our dataset

```
plt.figure(figsize=(20, 20))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```



Train, Test, validation data split

```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1, shuffle=True, shuffle_size=10000):
    assert (train_split + test_split + val_split) == 1

    ds_size = len(ds)

    if shuffle:
        ds = ds.shuffle(shuffle_size, seed=12)

    train_size = int(train_split * ds_size)
    val_size = int(val_split * ds_size)

    train_ds = ds.take(train_size)
    val_ds = ds.skip(train_size).take(val_size)
    test_ds = ds.skip(train_size).skip(val_size)
```

```
    return train_ds, val_ds, test_ds
```

```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

```
print("Size of Data is :{0} \nBatch size of Training Data is :{1}\nBatch size of Validation Data is :{2} \nBatch size of T
```

```
Size of Data is :115
Batch size of Training Data is :92
Batch size of Validation Data is :11
Batch size of Testing Data is :12
```

Cache, Shuffle, and Prefetch the Dataset

```
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Building the Model

Creating a Layer for Resizing and Normalization

Before we feed our images to network, we should be resizing it to the desired size. Moreover, to improve model performance, we should normalize the image pixel value (keeping them in range 0 and 1 by dividing by 256). This should happen while training as well as inference. Hence we can add that as a layer in our Sequential Model.

You might be thinking why do we need to resize (256,256) image to again (256,256). You are right we don't need to but this will be useful when we are done with the training and start using the model for predictions. At that time somone can supply an image that is not (256,256) and this layer will resize it

```
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1./255),
])
```

Data Augmentation

Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.

```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
```

Applying Data Augmentation to Train Dataset

```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model Architecture

We use a CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation.

```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    layers.Conv2D(32, kernel_size = (3,3), activation='relu', input_shape=input_shape),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64,  kernel_size = (3,3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(n_classes, activation='softmax'),
])

model.build(input_shape=input_shape)
```

```
model.summary()
```

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

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

max_pooling2d (MaxPooling2D  (32, 127, 127, 32)        0
)

conv2d_1 (Conv2D)            (32, 125, 125, 64)        18496

max_pooling2d_1 (MaxPooling  (32, 62, 62, 64)          0
2D)

conv2d_2 (Conv2D)            (32, 60, 60, 64)          36928

max_pooling2d_2 (MaxPooling  (32, 30, 30, 64)          0
2D)

conv2d_3 (Conv2D)            (32, 28, 28, 64)          36928

max_pooling2d_3 (MaxPooling  (32, 14, 14, 64)          0
2D)

conv2d_4 (Conv2D)            (32, 12, 12, 64)          36928

max_pooling2d_4 (MaxPooling  (32, 6, 6, 64)            0
2D)

conv2d_5 (Conv2D)            (32, 4, 4, 64)            36928

max_pooling2d_5 (MaxPooling  (32, 2, 2, 64)            0
2D)

flatten (Flatten)            (32, 256)                 0

dense (Dense)                (32, 64)                  16448

dense_1 (Dense)              (32, 3)                   195

=================================================================
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

Compiling the Model

We use `adam` Optimizer, `SparseCategoricalCrossentropy` for losses, `accuracy` as a metric

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

Training The Model

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)
```

```
Epoch 1/20
92/92 [==============================] - 311s 384ms/step - loss: 1.0065 - accuracy: 0.4734
Epoch 2/20
92/92 [==============================] - 28s 302ms/step - loss: 0.7126 - accuracy: 0.7225
Epoch 3/20
92/92 [==============================] - 29s 317ms/step - loss: 0.5739 - accuracy: 0.7730
Epoch 4/20
92/92 [==============================] - 28s 303ms/step - loss: 0.4251 - accuracy: 0.8338
Epoch 5/20
92/92 [==============================] - 29s 311ms/step - loss: 0.3133 - accuracy: 0.8870
Epoch 6/20
92/92 [==============================] - 28s 301ms/step - loss: 0.2989 - accuracy: 0.8870
Epoch 7/20
92/92 [==============================] - 28s 308ms/step - loss: 0.2313 - accuracy: 0.9181
Epoch 8/20
92/92 [==============================] - 28s 300ms/step - loss: 0.2024 - accuracy: 0.9317
Epoch 9/20
92/92 [==============================] - 28s 308ms/step - loss: 0.1805 - accuracy: 0.9345
Epoch 10/20
92/92 [==============================] - 28s 304ms/step - loss: 0.1557 - accuracy: 0.9498
Epoch 11/20
92/92 [==============================] - 30s 327ms/step - loss: 0.1340 - accuracy: 0.9529
Epoch 12/20
92/92 [==============================] - 28s 304ms/step - loss: 0.1035 - accuracy: 0.9662
Epoch 13/20
92/92 [==============================] - 29s 311ms/step - loss: 0.1115 - accuracy: 0.9601
Epoch 14/20
92/92 [==============================] - 28s 303ms/step - loss: 0.0929 - accuracy: 0.9659
Epoch 15/20
92/92 [==============================] - 28s 308ms/step - loss: 0.0904 - accuracy: 0.9679
Epoch 16/20
92/92 [==============================] - 28s 300ms/step - loss: 0.0921 - accuracy: 0.9703
Epoch 17/20
92/92 [==============================] - 29s 311ms/step - loss: 0.0818 - accuracy: 0.9713
Epoch 18/20
92/92 [==============================] - 28s 299ms/step - loss: 0.0708 - accuracy: 0.9771
Epoch 19/20
92/92 [==============================] - 29s 316ms/step - loss: 0.0911 - accuracy: 0.9700
Epoch 20/20
92/92 [==============================] - 28s 300ms/step - loss: 0.0778 - accuracy: 0.9768
```

```
scores = model.evaluate(test_ds)
```

```
12/12 [==============================] - 6s 34ms/step - loss: 0.1997 - accuracy: 0.9427
```

Scores is just a list containing loss and accuracy value

Plotting the Accuracy and Loss Curves

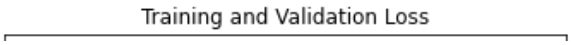
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

```
#graphs for accuracy and loss of training and validation data
plt.figure(figsize = (20,20))
plt.subplot(2,3,1)
plt.plot(range(EPOCHS), acc, label = 'Training Accuracy')
plt.plot(range(EPOCHS), val_acc, label = 'Validation Accuracy')
plt.legend(loc = 'lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(2,3,2)
plt.plot(range(EPOCHS), loss, label = 'Training Loss')
plt.plot(range(EPOCHS), val_loss, label = 'Validation Loss')
plt.legend(loc = 'upper right')
plt.title('Training and Validation Loss')
```

Text(0.5, 1.0, 'Training and Validation Loss')



Running prediction on a sample images

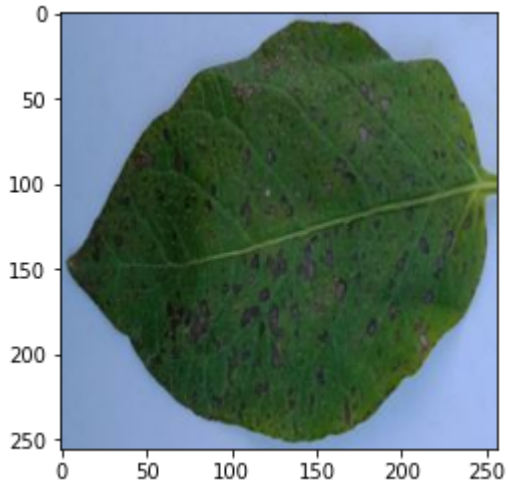
```
import numpy as np
for images_batch, labels_batch in test_ds.take(1):

    first_image = images_batch[0].numpy().astype('uint8')
    first_label = labels_batch[0].numpy()

    print("first image to predict")
    plt.imshow(first_image)
    print("actual label:",class_names[first_label])

    batch_prediction = model.predict(images_batch)
    print("predicted label:",class_names[np.argmax(batch_prediction[0])])
```

```
first image to predict
actual label: Early_Blight
1/1 [==============================] - 0s 241ms/step
predicted label: Early_Blight
```



Function to test for inference

```
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)
    print(predictions)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

Now running inference on few sample images

```
plt.figure(figsize=(15, 20))
for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))

        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
```

```
1/1 [==============================] - 0s 252ms/step
[[9.9999785e-01 2.0459451e-08 2.1367530e-06]]
1/1 [==============================] - 0s 24ms/step
[[6.8031495e-06 4.8416882e-06 9.9998832e-01]]
1/1 [==============================] - 0s 22ms/step
[[1.1667284e-11 2.1669587e-12 1.0000000e+00]]
1/1 [==============================] - 0s 24ms/step
[[9.999927e-01 6.550550e-12 7.306685e-06]]
1/1 [==============================] - 0s 25ms/step
[[9.9983239e-01 1.9847275e-05 1.4763210e-04]]
1/1 [==============================] - 0s 24ms/step
[[9.9997675e-01 2.1040007e-08 2.3262031e-05]]
1/1 [==============================] - 0s 22ms/step
[[9.9952888e-01 4.3672426e-05 4.2741140e-04]]
1/1 [==============================] - 0s 22ms/step
[[1.2278304e-02 2.3653038e-06 9.8771930e-01]]
1/1 [==============================] - 0s 26ms/step
[[1.3738289e-04 7.4737240e-05 9.9978787e-01]]
```