Module Name: SE4050 - Deep Learning Assignment

Group memebr details:

IT21302862 - Sri Samadhi L.A.S.S

IT21251900 - Rajapaksha R.M.S.

IT21178054 -

IT21360428 - Monali G.M.N.

Import all the Dependencies

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

Set all the Constants

```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

Import data into tensorflow dataset object

```
import zipfile
import os
import tensorflow as tf

# Step 1: Unzip the file from Google Drive to a local directory
zip_path = '/content/drive/MyDrive/DL_Project/plants.zip'

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall('temp_dir')

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    'temp_dir',
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 2152 files belonging to 1 classes.
```

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

```
['PlantVillage']
```

```
import os

# Path to the dataset directory
dataset_dir = '/content/temp_dir/PlantVillage'

# Get the list of subdirectories (class names)
class_names = [d for d in os.listdir(dataset_dir) if os.path.isdir(os.path.join(dataset_dir, d))]

print(class_names)
```

```
['Potato___healthy', 'Potato___Late_blight', 'Potato___Early_blight']
```

```
len(dataset)
```

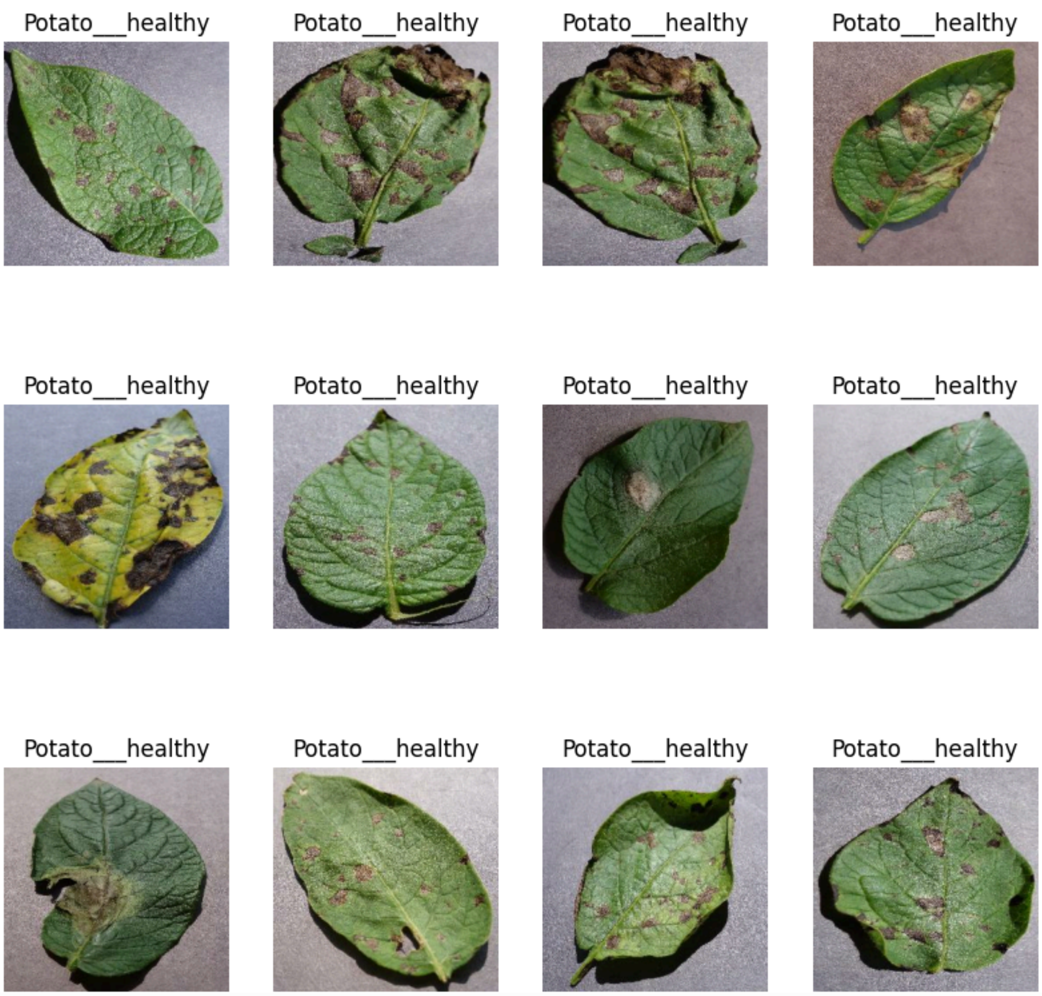
```
68
```

```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Visualize some of the images from our dataset

```
plt.figure(figsize=(10, 10))
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")
```



Function to Split Dataset Dataset should be bifurcated into 3 subsets, namely: **bold text**

Training: Dataset to be used while training Validation: Dataset to be tested against while training Test: Dataset to be tested against after we trained a model

```
len(dataset)
```

```
68
```

```
train_size = 0.8
len(dataset)*train_size
```

```
54.400000000000006
```

```
train_ds = dataset.take(54)
len(train_ds)
```

```
54
```

```
test_ds = dataset.skip(54)
len(test_ds)
```

```
14
```

```
val_size=0.1
len(dataset)*val_size
```

```
6.800000000000001
```

```
val_ds = test_ds.take(6)
len(val_ds)
```

```
6
```

```
test_ds = test_ds.skip(6)
len(test_ds)
```

```
8
```

```
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)
```

```
len(train_ds)
```

```
54
```

```
len(val_ds)
```

```
6
```

```
len(test_ds)
```

```
8
```

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)
```

IT21302862 - Building the Model

Creating a Layer for Resizing and Normalization

Before 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.

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

Data Augmentation

This boosts the accuracy of our model by augmenting the data.

```
import tensorflow as tf

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

IT21302862 - 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)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
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)	18,496
max_pooling2d_1 (MaxPooling2D)	(32, 62, 62, 64)	0
conv2d_2 (Conv2D)	(32, 60, 60, 64)	36,928
max_pooling2d_2 (MaxPooling2D)	(32, 30, 30, 64)	0
conv2d_3 (Conv2D)	(32, 28, 28, 64)	36,928
max_pooling2d_3 (MaxPooling2D)	(32, 14, 14, 64)	0
conv2d_4 (Conv2D)	(32, 12, 12, 64)	36,928
max_pooling2d_4 (MaxPooling2D)	(32, 6, 6, 64)	0
conv2d_5 (Conv2D)	(32, 4, 4, 64)	36,928
max_pooling2d_5 (MaxPooling2D)	(32, 2, 2, 64)	0
flatten (Flatten)	(32, 256)	0
dense (Dense)	(32, 64)	16,448
dense_1 (Dense)	(32, 3)	195

IT21302862 - 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']
)
```

```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
)
```

```
Epoch 1/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 254s 5s/step - accuracy: 0.9165 - loss: 0.2327 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 2/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 267s 5s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 3/50
54/54 ━━━━━━━━━━━━━━━━━━━━ 257s 5s/step - accuracy: 1.0000 - loss: 0.0000e+00 - val_accuracy: 1.0000 - val_loss: 0.0000e+00
Epoch 4/50
38/54 ━━━━━━━━━━━━━━━━━━━━ 1:13 5s/step - accuracy: 1.0000 - loss: 0.0000e+00
```

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

```
scores
```

Plotting the Accuracy and Loss Curves

```
history
```

```
history
```

```
history.params
```

```
history.history.keys()
```

```
type(history.history['loss'])
```

```
len(history.history['loss'])
```

```
history.history['loss'][:5]
```

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

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

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 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(1, 2, 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')
plt.show()
```

Run prediction on a sample image

```
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])])
```

Write a function 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)

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

```
plt.figure(figsize=(15, 15))
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")
```

Saving the Model

We append the model to the list of models as a new version

```
import os
model_version=max([int(i) for i in os.listdir("../models") + [0]])+1
model.save(f"../models/{model_version}")
```

```
model.save("../potatoes.h5")
```

Double-click (or enter) to edit