Plant Disease Classification

-by Satyadev Subudhi

Objective:

The aim of this project is to develop an accurate model capable of classifying and detecting various plant diseases from images. Utilizing the Plant Village Dataset, the project seeks to leverage advanced image processing and deep learning techniques to identify and categorize plant diseases effectively. This will ultimately contribute to improved crop management and agricultural productivity by enabling early intervention and treatment of plant diseases.

Dataset Description:

The dataset used for this project is sourced from the Plant Village Dataset on Kaggle website. The dataset originally comprises 20,639 images across 15 different classes of leaves. To expedite the testing phase and obtain results more efficiently, I selected a subset containing images from 5 specific classes, including those of potato and pepper plants. This strategic choice allows for quicker computation while maintaining a diverse representation of plant types within the subset.

Methodology:

- ### Loading the dataset:

  Loading potato and pepper images as tensorlow.data.dataset object.

```
dataset = tf.keras.preprocessing.image_dataset_from_directory( #working on a subset of data
    '/kaggle/input/2specplants/2specplants',
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

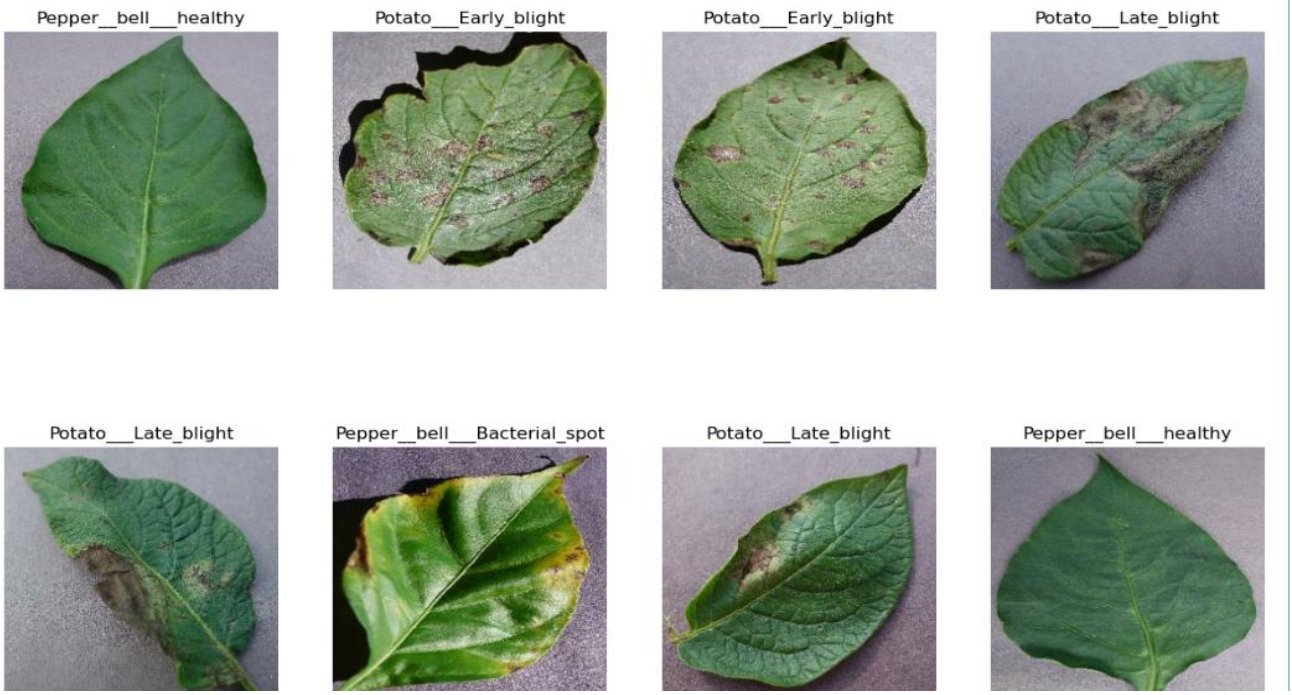
```
Found 4627 files belonging to 5 classes.
```

```
class_names = dataset.class_names
for i in range(len(class_names)):
    print(f"Class {i} = {class_names[i]}")

n_classes=len(class_names)
```

```
Class 0 = Pepper__bell___Bacterial_spot
Class 1 = Pepper__bell___healthy
Class 2 = Potato___Early_blight
Class 3 = Potato___Late_blight
Class 4 = Potato___healthy
```

- **Visualising the dataset images:**

In [76]:
```
# vizualization of dataset
plt.figure(figsize=(15, 15))
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")
```



- **Splitting the dataset:**

Training set= data which will be used for training model (60%)
Validation set = data to tested on while training (20%)
Testing set = data to tested on after training is done (20%)

In [78]:
```
#function to get train, test and validation data

def get_dataset_partitions_tf(ds, train_split=0.7, val_split=0.2, test_split=0.1, shuffle=True, shuffle_size=1000):
    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
```

In [79]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

- **Layers for resizing images and data augmentation:**

  Scaling the data and performing data augmentation (Rotating and flipping images)

```
#performing cache and prefetch operations
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)

#normalizing the data
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1./255),
])

#data augmentation
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:**

  In this project, we employed three advanced deep learning model architectures: MobileNetV2, ResNet, and VGG. Each model type has unique characteristics and advantages, making them suitable for different aspects of image classification tasks.

  - **MobileNetV2** is a lightweight, efficient deep learning model designed for mobile and embedded vision applications. It is known for its compact architecture, which makes it suitable for real-time applications on devices with limited computational power.
  - **ResNet** (Residual Network) is a highly influential model known for its introduction of residual learning, which allows training very deep networks without suffering from the vanishing gradient problem.
  - **VGG** (Visual Geometry Group) model is known for its simplicity and uniform architecture, which has been widely used for image classification tasks.

  After training the models, we find that **MobileNetV2** performs better than others in terms of accuracy.

- **Final Results:**

  The following are the final model architecture and results-

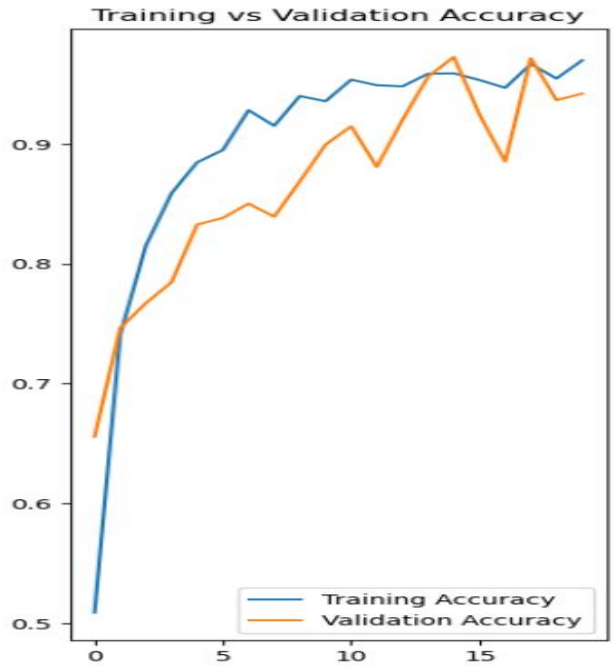
```
def create_mobilenetv2_model(input_shape, n_classes):
    base_model = tf.keras.applications.MobileNetV2(weights='imagenet', include_top=False, input_shape=input_shape)
    base_model.trainable = False  # Freeze the base model

    model = tf.keras.Sequential([
        resize_and_rescale,
        data_augmentation,
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(64, activation='relu'),
        layers.Dense(n_classes, activation='softmax')
    ])

    return model

input_shape = (IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
mobilenetv2_model = create_mobilenetv2_model(input_shape, n_classes)
mobilenetv2_model.summary()
```

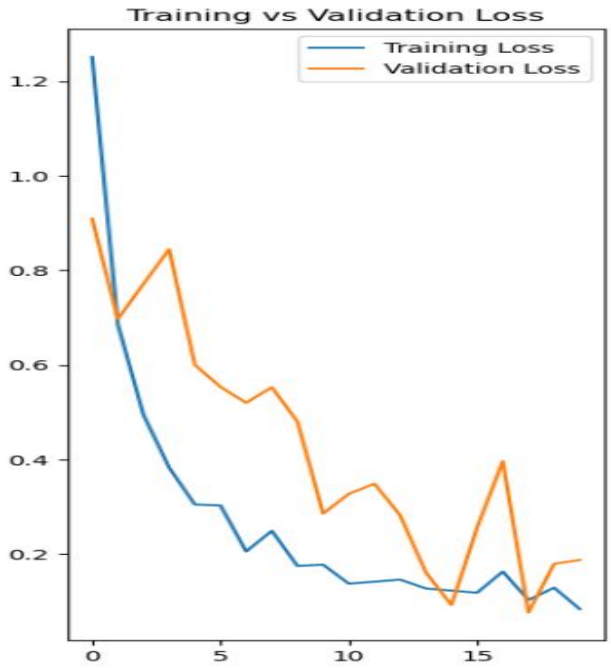
Defining hyperparameters like batch size = 32, number of epochs = 20 and using appropriate optimizers such as Adam optimizer and loss function like categorical cross entropy

```
input_shape = (256,256,3)
best_model = create_mobilenetv2_model(input_shape, n_classes)
best_model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
history = best_model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=20 # for better accuracy pick a higher number
)
```

```
87/87 ━━━━━━━━━━━━━━━━━━━━ 32s 367ms/step - accuracy: 0.9878 - loss: 0.0282 - val_accuracy: 0.9720 - val_loss: 0.0737
Epoch 18/20
87/87 ━━━━━━━━━━━━━━━━━━━━ 32s 370ms/step - accuracy: 0.9896 - loss: 0.0287 - val_accuracy: 0.9591 - val_loss: 0.1107
Epoch 19/20
87/87 ━━━━━━━━━━━━━━━━━━━━ 32s 367ms/step - accuracy: 0.9875 - loss: 0.0283 - val_accuracy: 0.9677 - val_loss: 0.0891
Epoch 20/20
87/87 ━━━━━━━━━━━━━━━━━━━━ 32s 365ms/step - accuracy: 0.9913 - loss: 0.0292 - val_accuracy: 0.9547 - val_loss: 0.1251
```

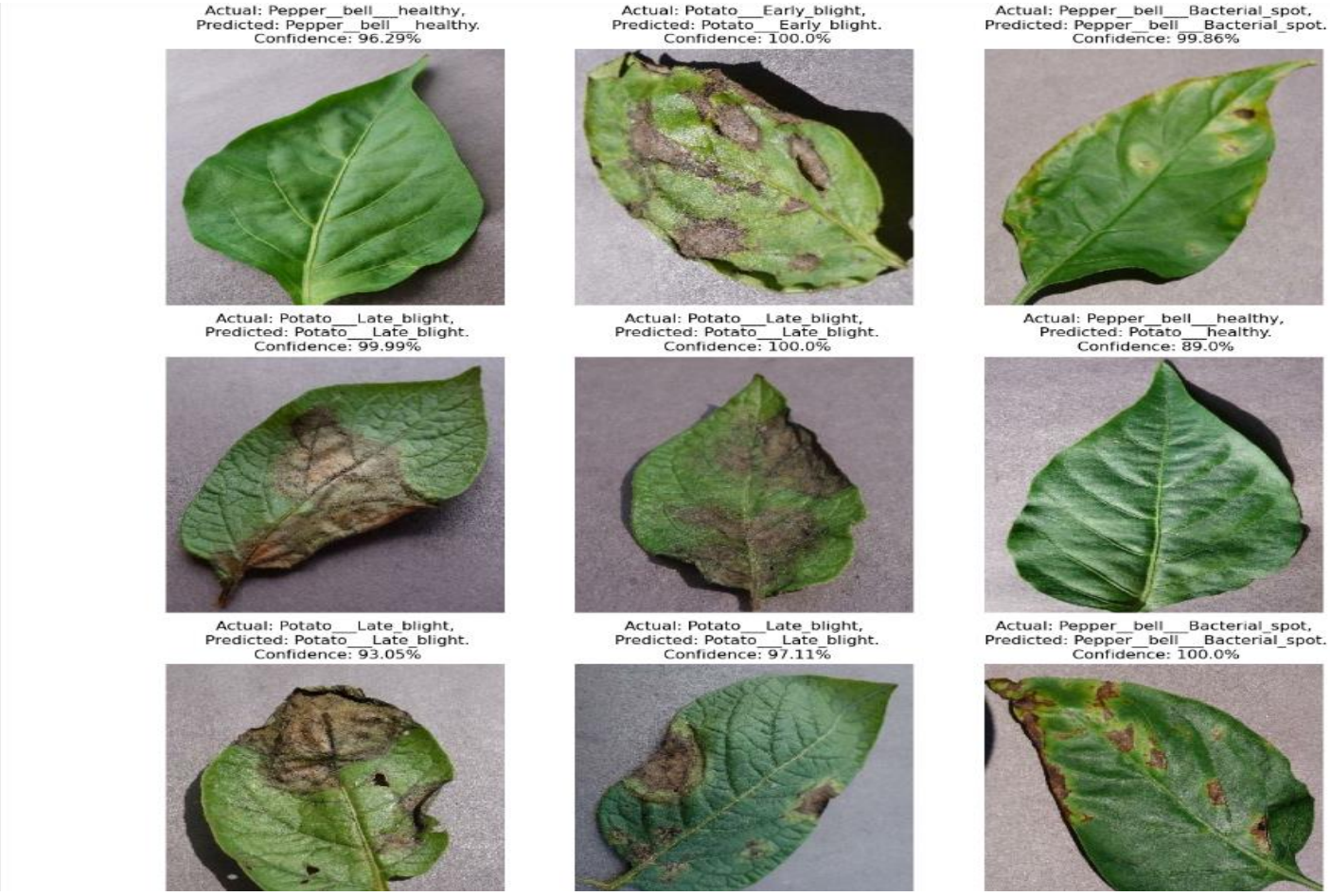
- **Training vs Validation plots:**





- **Evaluation on custom input:**

```
#creating a predict function for testing on sample inputs
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) #confidence level calculation
    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(best_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")
```



- **Classification report:**

Precision is defined as the ratio of true positive predictions to the total number of positive predictions
Recall is defined as the ratio of true positive predictions to the total number of actual positive instances
The F1 score is the harmonic mean of precision and recall

f1 score of class 4 is low because the dataset is imbalanced i.e it has very few images belonging to class 4 in the original dataset.

```
y_true = np.concatenate([y for x, y in test_ds], axis=0)
y_pred_prob = model_mob.predict(test_ds)
y_pred = np.argmax(y_pred_prob, axis=-1)
print(classification_report(y_true, y_pred, target_names=[str(i) for i in range(n_classes)]))
```

```
29/29 ━━━━━━━━━━━━━━━━━━━━ 3s 77ms/step
              precision    recall  f1-score   support

           0       0.29      0.30      0.29       181
           1       0.44      0.45      0.45       318
           2       0.33      0.33      0.33       202
           3       0.34      0.34      0.34       191
           4       0.08      0.09      0.09        23

    accuracy                           0.36       915
   macro avg       0.30      0.30      0.30       915
weighted avg       0.36      0.36      0.36       915
```

- **Challenges and Limitations:**
    - The original dataset is huge and requires a good gpu for training.
    - The given dataset is imbalanced due to lower number of images belonging to class 4 which leads to lower f1 score for that class.
    - Adding more layers in CNN architecture increases the computation time during training but increases the accuracy of model overall.

- **Future Work:**
    - The entire dataset can be used for training with better gpus and tpus of the machine.
    - Work needs to be done towards balancing the dataset.
    - More models need to explored for better results.
    - Adding more convolution layers and using dropout layers in the model architecture will reduce overfitting and improve the f1 scores.

- **Conclusion:**

The project successfully developed a deep learning model capable of detecting and classifying plant diseases from images with good accuracy. This model can be used to identify and categorize plant diseases effectively**.** This will ultimately contribute to improved crop management and agricultural productivity by enabling early intervention and treatment of plant diseases.

Link of the code → Plant_disease_classification (Kaggle notebook)