Application of AI/ML in Agriculture

Course Instructor: Prof. Amey Pathak

Steps Involved in this Project

> **1. Data collection**
>
> **2. Data Preprocessing**
>
> **3. Model building**
>
> **4. Model Evaluation**
>
> **5. Model Testing and Implementation**

1. Data Collection:

In this project, readymade data is used obtained from website Kaggle.

Dataset Link: https://www.kaggle.com/abdallahalidev/plantvillage-dataset

Importing the required libraries into Colab

```
# NumPy: Fundamental library for numerical operations with support for large arrays and matrices.
import numpy as np

# Pandas: Powerful data manipulation and analysis library with Series and DataFrame data structures.
import pandas as pd

# TensorFlow: Open-source machine learning library for building, training, and deploying models.
import tensorflow as tf

# Matplotlib: Widely-used plotting library in Python for creating diverse visualizations.
import matplotlib.pyplot as plt

# Keras: High-level neural networks API running on top of TensorFlow for easy model development.
from keras import layers, Sequential, models



print("All libraries succesfully installed and loaded !!")
```

```
All libraries succesfully installed and loaded !!
```

Loading data into Google colab

```
#Check Instructions above for mounting Google Drive
# Mount Google Drive in Google Colab to access files stored in Google Drive directly.
from google.colab import drive

# The '/content/drive' path is a common mounting point for accessing Google Drive content in Colab.
drive.mount('/content/drive')
print("Drive Mounted Successfully")
```

Show hidden output

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

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

```
!unzip /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Copy_of_Potato_Dataset.zip -d /content/drive/MyDrive/Colab_Notebooks/AI_
```

```
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_841.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_842.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_843.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_844.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_845.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_846.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_847.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_848.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_849.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_85.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_850.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_851.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_852.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_853.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_854.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_855.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_856.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_857.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_858.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_859.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_86.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_860.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_861.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_862.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_863.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_864.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_865.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_866.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_867.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_868.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_869.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_87.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_870.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_871.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_872.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_873.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_874.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_875.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_876.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_877.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_878.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_879.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_88.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_880.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_881.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_882.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_883.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_884.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_885.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_886.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_887.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_888.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_889.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_89.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_890.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_891.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_892.jpg
inflating: /content/drive/MyDrive/Colab_Notebooks/AI_in_Agriculture/Potato_unzip/Potato Dataset/Healthy/Healthy_893.jpg
```

Initializing Batch Size, Image size and epochs

```
# Set the batch size for training data.
BATCH_SIZE = 32

# Define the image size for preprocessing and model input.
IMAGE_SIZE = 256

# Specify the number of channels in the images (3 for RGB).
CHANNELS = 3

# Set the number of epochs for training the model.
EPOCHS = 10

print("Batch size, Image size, channels and epochs INITIALIZED !!!!")
```

```
Batch size, Image size, channels and epochs INITIALIZED !!!!
```

```
class_names = dataset.class_names
print("The dataset consists of classes:",class_names)
```

```
The dataset consists of classes: ['EarlyBlight', 'Healthy', 'LateBlight']
```

Visualizing the images from our dataset

```
#image specifications
for image_batch, label_batch in dataset.take(1):
  print(image_batch.shape)
```

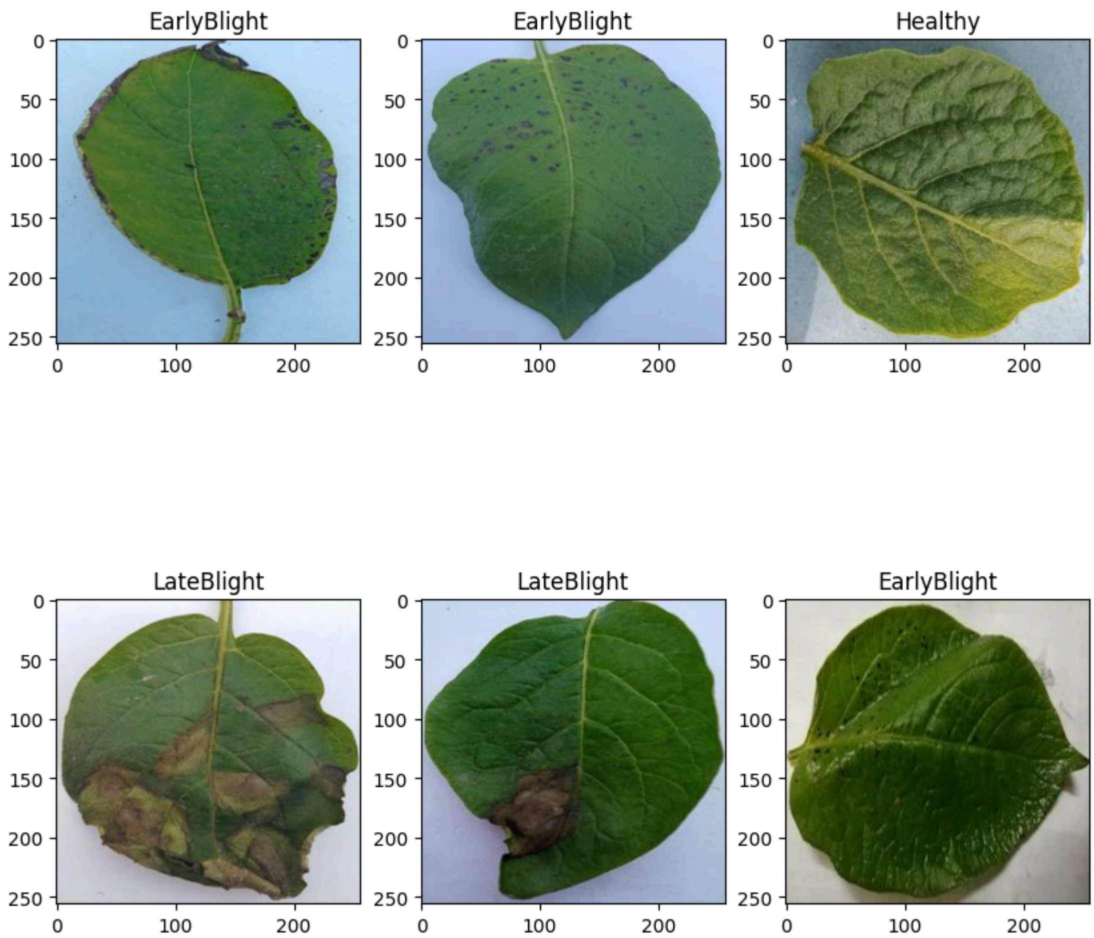
```
(32, 256, 256, 3)
```

The output indicates that the image batch consists of 32 images with dimensions of 256 pixels in height, 256 pixels in width, and 3 channels, representing the RGB color information. Each image is a 3D array with shape (256, 256, 3).

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

```
[2 0 1 2 2 1 2 0 2 2 1 1 1 2 1 0 1 2 2 1 1 1 0 1 1 2 1 0 1 0 2 1]
```

```
# Visualizing the images in the dataset
plt.figure(figsize=(10, 10))

for image_batch, label_batch in dataset.take(1):
  for i in range(6):
    ax = plt.subplot(2, 3, i + 1)
    plt.imshow(image_batch[i].numpy().astype("uint8"))
    plt.title(class_names[label_batch[i]])


#This code picks randomly some images in the dataset
```



Data Preprocessing

Data Splitting for Model Training:

```
#set the train, validation and test splits in decimals , example 80 % as 0.8.
#NOTE: the total train + val + test should be equal to  1

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 Test Data is: {3}
    len(dataset),
    len(train_ds),
    len(val_ds),
    len(test_ds)
))
```

```
Size of Data is: 94
Batch size of Training Data is: 75
Batch size of Validation Data is: 9
Batch size of Test Data is: 10
```

```
# Cache the training dataset in memory for faster access during training.
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

# Cache the validation dataset in memory for faster access during model evaluation.
val_ds = val_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)

# Cache the test dataset in memory for faster access during model testing.
test_ds = test_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

Image Preprocessing for Model Input:

```
# Sequential model for resizing images to IMAGE_SIZE and rescaling pixel values
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.Rescaling(1./255),
])
```

Enhancing Model Robustness with Data Augmentation:

```
 # Sequential model for data augmentation, including random horizontal and vertical flips, and random rotation
data_augmentation = tf.keras.Sequential([
    layers.RandomFlip("horizontal_and_vertical"),
    layers.RandomRotation(0.2),
])

# Applying data augmentation to the training dataset using the map function and prefetching for performance
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

CNN Model Architecture

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)

print("CNN architecture has been built sucessfully !")

```
model.summary()
```

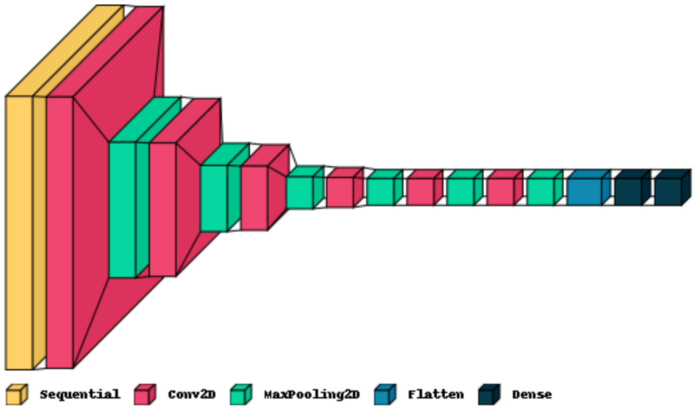
```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d_6 (Conv2D)           (32, 254, 254, 32)        896

 max_pooling2d_6 (MaxPoolin  (32, 127, 127, 32)        0
 g2D)

 conv2d_7 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_7 (MaxPoolin  (32, 62, 62, 64)          0
 g2D)

 conv2d_8 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_8 (MaxPoolin  (32, 30, 30, 64)          0
 g2D)

 conv2d_9 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_9 (MaxPoolin  (32, 14, 14, 64)          0
 g2D)

 conv2d_10 (Conv2D)          (32, 12, 12, 64)          36928

 max_pooling2d_10 (MaxPooli  (32, 6, 6, 64)            0
 ng2D)

 conv2d_11 (Conv2D)          (32, 4, 4, 64)            36928

 max_pooling2d_11 (MaxPooli  (32, 2, 2, 64)            0
 ng2D)

 flatten_1 (Flatten)         (32, 256)                 0

 dense_2 (Dense)             (32, 64)                  16448

 dense_3 (Dense)             (32, 3)                   195

=================================================================
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Visualizing CNN architecture

```
# Installing the Library required for visualization as this library is not installed in colab
# For this Python PIP is used.
!pip install visualkeras
```

```
Collecting visualkeras
  Downloading visualkeras-0.0.2-py3-none-any.whl (12 kB)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (9.4.0)
Requirement already satisfied: numpy>=1.18.1 in /usr/local/lib/python3.10/dist-packages (from visualkeras) (1.25.2)
Collecting aggdraw>=1.3.11 (from visualkeras)
  Downloading aggdraw-1.3.18-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (993 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 993.7/993.7 kB 1.3 MB/s eta 0:00:00
Installing collected packages: aggdraw, visualkeras
Successfully installed aggdraw-1.3.18 visualkeras-0.0.2
```

```
# After installation of library, now it canbbe imported
import visualkeras
visualkeras.layered_view(model, legend=True, scale_xy=0.8)
```



Compiling the Model

```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
print("Model sucessfully Comppiled!")
```

```
Model sucessfully Comppiled!
```

Model Training

```
#Model training using the compiled model
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=EPOCHS,
)

print("Model successfuly trained")
```

```
Epoch 1/10
75/75 [==============================] - 368s 423ms/step - loss: 1.0584 - accuracy: 0.4306 - val_loss: 0.9209 - val_accuracy: 0.607
Epoch 2/10
75/75 [==============================] - 21s 278ms/step - loss: 0.9502 - accuracy: 0.5380 - val_loss: 0.8687 - val_accuracy: 0.5556
Epoch 3/10
75/75 [==============================] - 21s 273ms/step - loss: 0.8574 - accuracy: 0.6066 - val_loss: 0.8089 - val_accuracy: 0.6701
Epoch 4/10
75/75 [==============================] - 21s 279ms/step - loss: 0.8520 - accuracy: 0.6079 - val_loss: 0.7018 - val_accuracy: 0.7118
Epoch 5/10
75/75 [==============================] - 21s 278ms/step - loss: 0.6035 - accuracy: 0.7609 - val_loss: 0.5321 - val_accuracy: 0.8194
Epoch 6/10
75/75 [==============================] - 20s 273ms/step - loss: 0.5167 - accuracy: 0.8064 - val_loss: 0.3463 - val_accuracy: 0.8715
Epoch 7/10
75/75 [==============================] - 21s 280ms/step - loss: 0.3979 - accuracy: 0.8411 - val_loss: 0.3183 - val_accuracy: 0.8819
Epoch 8/10
75/75 [==============================] - 21s 274ms/step - loss: 0.3289 - accuracy: 0.8809 - val_loss: 0.2467 - val_accuracy: 0.9028
Epoch 9/10
75/75 [==============================] - 21s 280ms/step - loss: 0.2786 - accuracy: 0.8972 - val_loss: 0.3620 - val_accuracy: 0.8681
Epoch 10/10
75/75 [==============================] - 20s 269ms/step - loss: 0.2150 - accuracy: 0.9189 - val_loss: 0.2710 - val_accuracy: 0.8958
Model successfuly trained
```

Model Evaluation

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

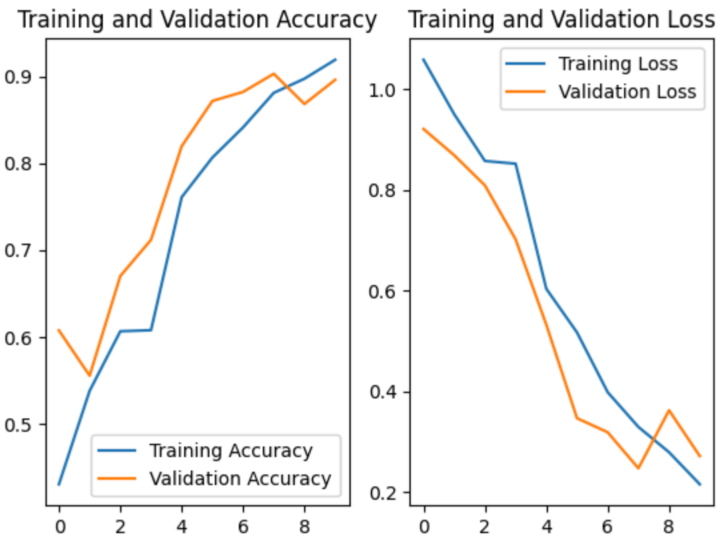
```
10/10 [==============================] - 14s 28ms/step - loss: 0.1883 - accuracy: 0.9312
```

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

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

Visualization of Training Dynamics:

```
#graphs for accuracy and loss of training and validation data
plt.figure(figsize = (10,10))
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')
```



Model Prediction and Evaluation

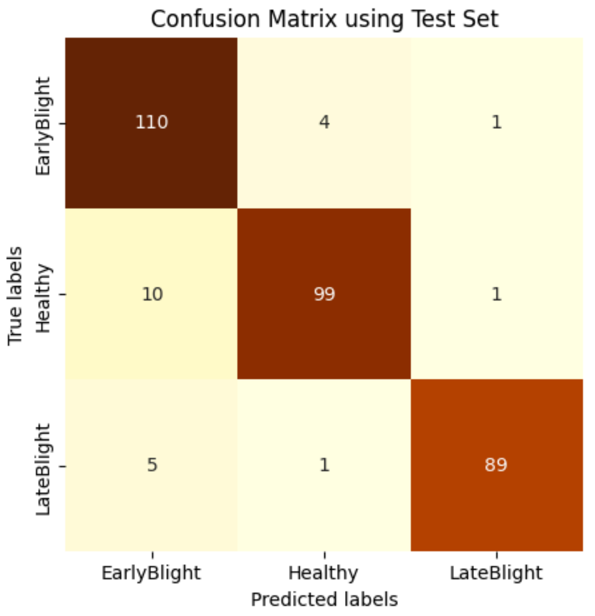
```
import numpy as np

lbls_true = []
lbls_pred = []  # predicted integer labels
pred_confs = []  # confidences

for imgs, lbls in test_ds:
    lbls_true.extend(lbls.numpy().tolist())

    pred_imgs = model.predict(imgs)
    for pred_img in pred_imgs:
        lbls_pred.append(np.argmax(pred_img))
        pred_confs.append(np.max(pred_img))
```

```
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 38ms/step
```

```
from sklearn.metrics import confusion_matrix

# changing integer labels to class names
lbls_true_names = list(map(lambda x: class_names[x], lbls_true))
lbls_pred_names = list(map(lambda x: class_names[x], lbls_pred))

# getting confusion matrix
cf_matrix = confusion_matrix(lbls_true_names, lbls_pred_names, labels=class_names)
cf_matrix_title = 'Confusion Matrix using Test Set'
print(cf_matrix_title)
print(cf_matrix)
```

```
Confusion Matrix using Test Set
[[110   4   1]
 [ 10  99   1]
 [  5   1  89]]
```

```
import seaborn as sns

#plotting confusion matrix
plt.figure(figsize=(5, 5))
sns.heatmap(cf_matrix, annot=True, fmt='g',
            xticklabels=class_names, yticklabels=class_names,
            cbar=False, cmap='YlOrBr')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title(cf_matrix_title)
plt.show()
```




```
# getting precisions
pred_sums = np.sum(cf_matrix, axis=0)
prec_matrix = cf_matrix / pred_sums
precs = prec_matrix.diagonal()

# getting recalls
true_sums = np.sum(cf_matrix, axis=1)
recall_matrix = cf_matrix / true_sums
recalls = recall_matrix.diagonal()

# getting f1 measures
f1s = (2 * precs * recalls) / (precs + recalls)


# displaying evaluations
evals_df = pd.DataFrame(data={'Precision': precs, 'Recall': recalls, 'F1': f1s},
                        index=class_names)
pd.options.display.float_format = '{:.2%}'.format
evals_df
```

	Precision	Recall	F1
EarlyBlight	88.00%	95.65%	91.67%
Healthy	95.19%	90.00%	92.52%
LateBlight	97.80%	93.68%	95.70%

Saving a Model

```
# Save the model to a file
model.save('potato_disease_model_v1.h5')  # You can replace 'potato_disease_model_v1.h5' with your desired file name
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file vi
  saving_api.save_model(
```

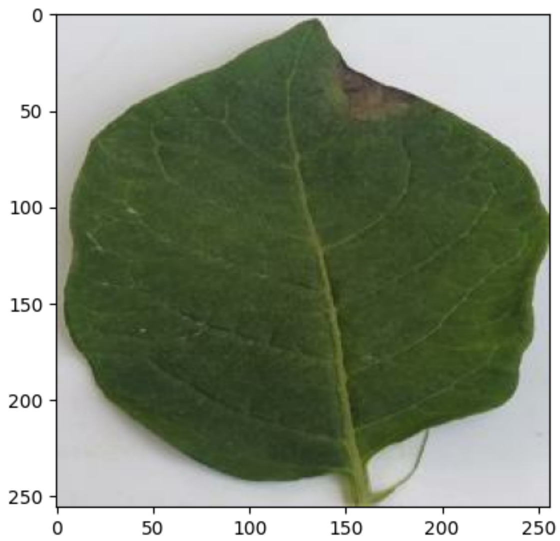
Start coding or generate with AI.

```
# Optionally, download the saved model file to your local machine
from google.colab import files
files.download('potato_disease_model_v1.h5')
```

Prediction and confidence interval

```
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: LateBlight
1/1 [==============================] - 0s 42ms/step
predicted label: LateBlight
```




```
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=(14, 14))
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 27ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
```