End-to-end deep learning project for Potato Disease Classification.

TEAM-592248

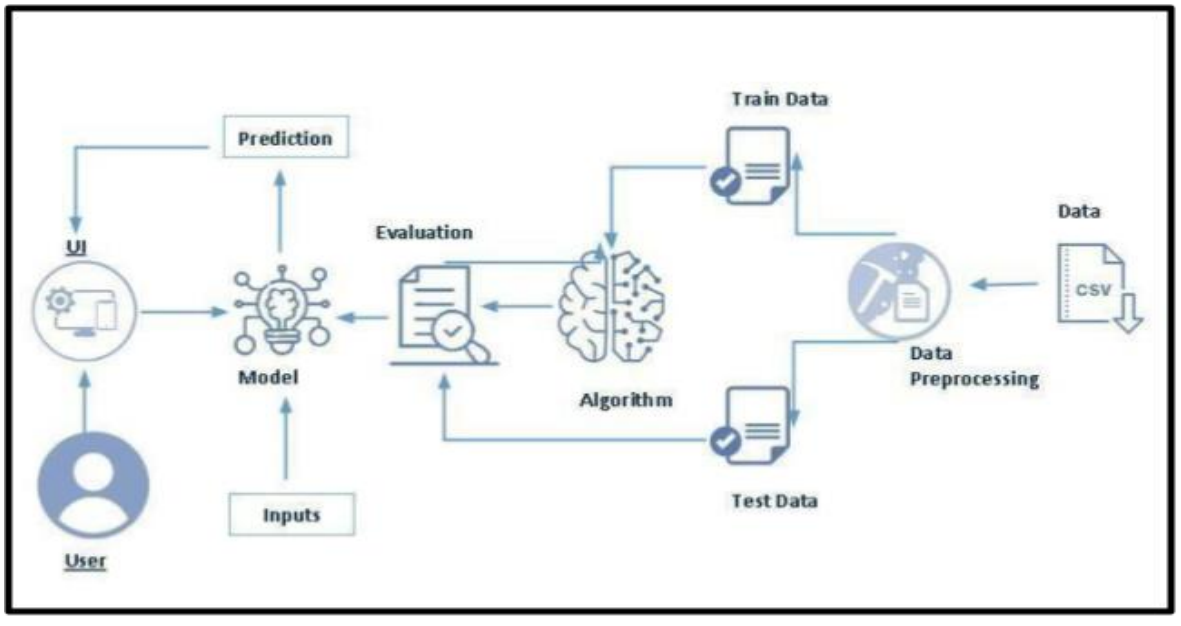
The objective of this project is to develop an end-to-end deep learning solution for classifying potato Diseases images into three categories: healthy, early blight, and late blight. The proposed solution involves the use of convolutional neural networks (CNNs) and Transfer Learning techniques to extract relevant features from the input images and classify them into one of the three categories.

Predicting potato leaf disease as soon as possible is crucial because it can have significant impacts on crop yield and quality. Early detection allows farmers to take prompt action to prevent the spread of the disease and reduce crop damage. Here are some reasons why predicting potato leaf disease early is essential:

1. **Minimize crop loss:** Potato leaf disease can significantly reduce crop yield and quality. By predicting the disease early, farmers can take timely measures to control its spread, thereby minimizing crop loss.
2. **Reduce cost:** Early detection of potato leaf disease can help farmers reduce costs associated with disease control measures, such as pesticides and other treatments. By identifying the disease early, farmers can target the specific area of the crop affected, which helps in reducing the overall cost of control measures.
3. **Protect the environment:** The excessive use of pesticides and other control measures can have negative impacts on the environment. Early detection of potato leaf disease can help farmers target only the affected areas and minimize the use of pesticides, reducing their environmental impact.
4. **Improve crop quality:** Potato leaf disease can affect the quality of the crop, making it less desirable to buyers. By predicting the disease early, farmers can take appropriate measures to prevent its spread, thereby improving the overall quality of the crop.

Let us look at the Technical Architecture of the project.

Technical Architecture:



Project Flow:

- The user uses the UI or web application to upload the file.
- Uploaded images or inputs is analyzed by the model which is integrated/developed.
- The model predicts between the three classes we named and gives specific output in the user interface we have interacted before.
- Define Problem / Problem Understanding
    - 0 Specify the business problem
    - Business requirements
    - Literature Survey.
    - Social or Business Impact.
- Data Collection & Preparation
    - Collect the dataset
    - Data Preparation
- Exploratory Data Analysis
    - 0 Descriptive statistical
    - Visual Analysis
- Model Building
    - Building a model.
    - 0 Training the model.
    - Testing the model
- Model Deployment
    - 0 Save the best model
    - Integrate with Web Framework
- Project Demonstration & Documentation
    - 0 Record explanation Video for project end to end solution

- Project Documentation-Step by step project development procedure

Prior Knowledge:

To complete this project, you must require following software's , concepts and packages

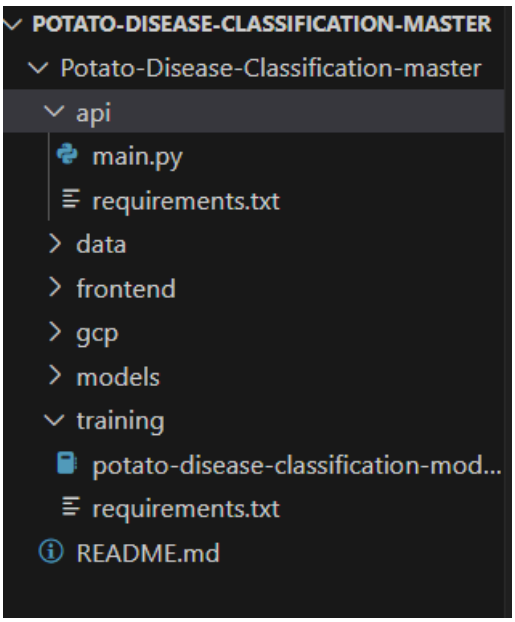
- VS Code :
  - Refer to the link below to download VS Code.
  - Link : https://code.visualstudio.com/download
- Create Project:
  - Open VS Code.
  - Create a Folder and name it "Potato_Disease_Classification" .
- Deep Learning Concepts
  - CNN: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add
  - Sequential Models : https://towardsdatascience.com/coding-a-convolutional-neural-network-cnn-using-ker a s-sequential-api-ec5211126875
  - Transfer Learning techniques: https://www.tensorflow.org/tutorials/images/transfer_learning
- Web concepts:
  - Get the gist on Fast Api: https://www.geeksforgeeks.org/fastapi-introduction/

Project Objectives:

By the end of this project you will:

- know fundamental concepts and techniques of Convolutional Neural Network and integration with transfer learning Techniques.
- This wide outline then focuses analytical approaches, aligns techniques to targets and allows full utilization of images' analytical potential.
- Get to Know to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Fast Api and hands on Node Js/React Js framework.

Project Structure:



Create the Project folder which contains files as shown below

- We are building a Fast Api application which needs a python script main.py for a website.
- model's folder contains your saved models.
- The training folder contains your data and code for building/training/testing the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

The manual identification of potato diseases is time-consuming, subjective, and prone to human error. To improve efficiency, accuracy, and scalability in disease diagnosis, there is a need to develop an automated potato disease classification system using advanced image processing and machine learning. This system should accurately categorize common potato diseases like late blight and scab from images. The goal is to provide farmers with a scalable, easy-to-use tool to make data driven decisions about disease management and protection. Overall, automating potato disease identification through AI and computer vision can empower more effective interventions against outbreaks.

Activity 2: Business requirements

Here are some potential business requirements for an ecommerce product delivery estimation predictor using deep learning:

1. Accurate prediction: The Model we train must be able to accurately predict the stage of leaf degradation. The accuracy of the prediction is crucial for farmers, agribusinesses, and other stakeholders to make informed decisions on the production.

2. User-friendly interface: The Model must have a user-friendly interface that is easy to navigate and understand as most of the Target audience are farmers and the should get the prediction in less time. The interface should present the results of the predictor in a clear and concise manner to enable farmers and other stakeholders to make informed decisions.

3. Scalability: The predictor system will leverage a cloud-based infrastructure and optimized parallel processing to enable efficient analysis of large image volumes. Edge computing integration will facilitate scalability in remote field locations. The machine learning models are designed for performance efficiency across devices.

Activity 3: Literature Survey (Student Will Write)

Potato disease classification has been a subject of significant research, leveraging various technological approaches. In 2018, a study focused on the "Application of Machine Learning Techniques for Potato Disease Detection" explored the integration of image processing and classification algorithms for identifying diseases affecting potatoes. Subsequent to this, a 2019 literature review delved into the realm of "Image Processing Techniques for Potato Disease Detection," discussing the diverse methodologies within computer vision and highlighting associated challenges and opportunities. The year 2020 witnessed research specifically dedicated to "Potato Disease Classification Using Deep Learning Techniques," examining the application of convolutional neural networks (CNNs) and comparing their efficacy to traditional machine learning approaches. Preceding this, a 2017 paper titled "An Overview of Potato Diseases: A Review" provided a foundational resource, offering a comprehensive understanding

Activity 4: Social or Business Impact.

By enabling early identification of potato diseases, the proposed solution enhances food security and environmental sustainability through increased yields and targeted disease management, while also economically empowering smallholder farmers. The user-friendly and accessible interface ensures ease of adoption and satisfaction. Farmers receive real-time feedback and tailored recommendations that translate into prompt and effective disease management actions. Continuous improvement and adaptability to emerging challenges, along with scalability to diverse agricultural practices, provide ongoing value. In summary, this solution delivers positive social impacts and prioritizes customer satisfaction through an intelligent, easy-to use tool that provides practical, customizable insights for sustainable potato farming.

Milestone 2: Data Collection & Preparation

DL depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset.

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

DATASET : ***Click Me and install only potato related files.***

Make sure your directory looks like
belowTraining
|_PlantVillage
|_ Potato_Early_blight
|_ Potato_healthy
|_ Potato_Late_blight
|_Training.ipynb

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Under the Training folder create a training.ipynb file in jupyter and start writing code in this file.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

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

Activity 1.2: Set ImageSize, BatchSize and Channels.

Set the values as shown below

```
Seting up all the Constants

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

Activity 1.2: Read the Dataset

Our dataset contains .jpg images. We can read the dataset with the help of tensorflow.

In tensorflow we have a function called image_dataset_from_directory() to read the dataset. As a parameter we have to give the directory of the images and other parameters as shown below.

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
        '../data/dataset/train',
        target_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE,
        class_mode="sparse",
)
✓ 0.1s                                                    Python
Found 1506 images belonging to 3 classes.
```

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the deep learning model. We have to extract the class names of the data and let's try to visualise the data with labels.

- Extracting class names.
- Visualising the data

Activity 2.1: Extracting class names

- Let's find the class names of our dataset first. To find the class names ofthe dataset, we can use .class_names.

```
class_names = list(train_generator.class_indices.keys())
class_names
```

✓ 0.0s Python

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

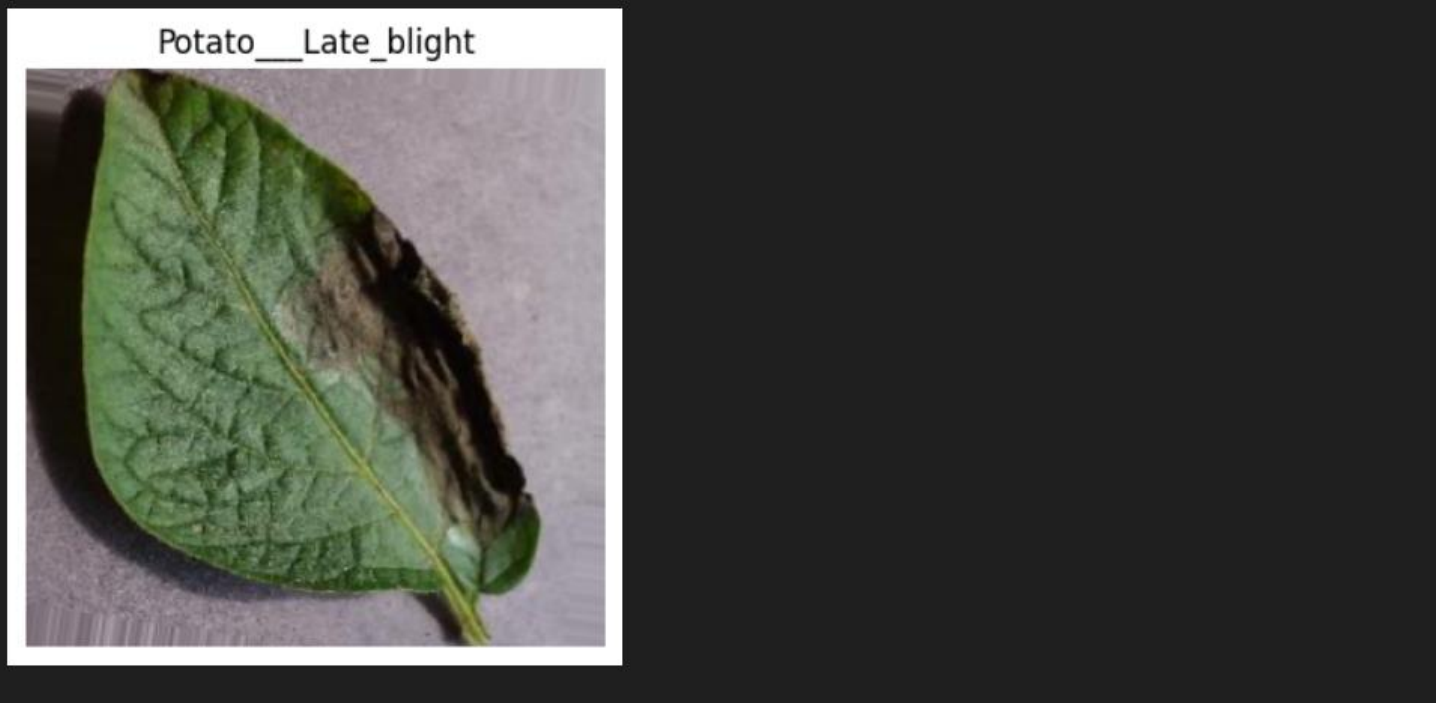
Activity 2.2: Visualising the data

We have created batches of the data and we can access the batch

Let's visualize some of the images from our dataset

```
plt.figure(figsize=(12,4))

for image_batch, label_batch in train_generator:
    plt.imshow(image_batch[0])
    plt.title(class_names[int(label_batch[0])])
    plt.axis("off")
    break
```

✓ 2.0s



- Image batch contains images and label batch contains labels of the particular image.
- Below code subplots 9 images from the 1st batch of the extracted dataset.

```python
for images, labels in test_generator:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

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

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

        plt.axis("off")
    break
```

✓ 2.3s Python

```
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 31ms/step
```

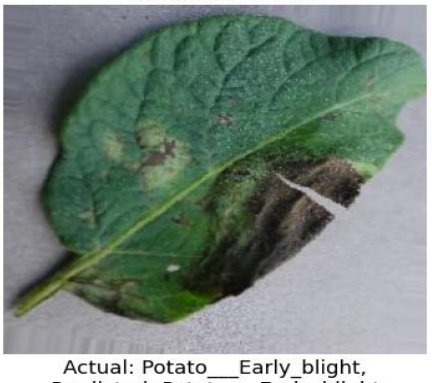
Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.91%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

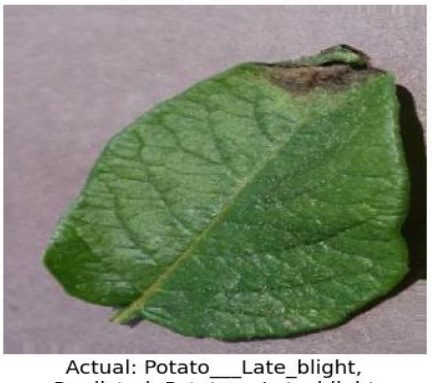
Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.99%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.89%



Actual: Potato___Late_blight,
Predicted: Potato___healthy.
Confidence: 93.81%



Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.94%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%

You might get different images as we assigned the shuffling parameter to True while reading the dataset

Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

Which are not suitable for our dataset.

Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

This is not needed for our current dataset as it is an image dataset. Visual analysis is mostly used for numerical data.

Activity 3: Splitting data into train and test and validation sets

Now let's split the Dataset into train, test and validation sets. First split the dataset into train and test sets.

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True
)

train_generator = train_datagen.flow_from_directory(
        '../data/dataset/train',
        target_size=(IMAGE_SIZE, IMAGE_SIZE),
        batch_size=BATCH_SIZE,
        class_mode="sparse",
)
```

✓ 0.1s Python

```
Found 1506 images belonging to 3 classes.
```

```python
validation_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True)

validation_generator = validation_datagen.flow_from_directory(
        '../data/dataset/val',
        target_size=(IMAGE_SIZE,IMAGE_SIZE),
        batch_size=32,
        class_mode="sparse"
)
```

✓ 0.0s Python

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

```python
test_datagen = ImageDataGenerator(
        rescale=1./255,
        rotation_range=10,
        horizontal_flip=True)

test_generator = test_datagen.flow_from_directory(
        '../data/dataset/test',
        target_size=(IMAGE_SIZE,IMAGE_SIZE),
        batch_size=32,
        class_mode="sparse"
)
```

✓ 0.0s Python

```
Found 431 images belonging to 3 classes.
```

Milestone 4: Model Building

Activity 1: Building a model

Now the data is augmented and completely preprocessed we can head to the next part that is model building.

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

model = models.Sequential([
    layers.InputLayer(input_shape=input_shape),
    layers.Conv2D(32, 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,  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'),
])
```

✓ 0.2s Python

```
model.summary()
```

✓ 0.0s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      896

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 64)        36928

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 64)       0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 64)        36928

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 64)       0
 2D)

 conv2d_4 (Conv2D)           (None, 12, 12, 64)        36928
...
Total params: 183,747
Trainable params: 183,747
Non-trainable params: 0
_________________________________________________________________
```

Activity 2: Compiling the model

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

✓ 0.0s

Activity 3: Training the model

```
history = model.fit(
    train_generator,
    steps_per_epoch=47,
    batch_size=BATCH_SIZE,
    validation_data=validation_generator,
    validation_steps=6,
    verbose=1,
    epochs=EPOCHS,
)
```

```
Epoch 1/20
47/47 [==============================] - 106s 2s/step - loss: 0.9011 - accuracy: 0.5400 - val_loss: 0.7863 - val_accuracy: 0.7344
Epoch 2/20
47/47 [==============================] - 79s 2s/step - loss: 0.6680 - accuracy: 0.7198 - val_loss: 0.4601 - val_accuracy: 0.8073
Epoch 3/20
47/47 [==============================] - 83s 2s/step - loss: 0.4296 - accuracy: 0.8304 - val_loss: 0.5007 - val_accuracy: 0.8281
Epoch 4/20
47/47 [==============================] - 91s 2s/step - loss: 0.3014 - accuracy: 0.8826 - val_loss: 0.1960 - val_accuracy: 0.9010
Epoch 5/20
47/47 [==============================] - 93s 2s/step - loss: 0.1675 - accuracy: 0.9335 - val_loss: 0.1308 - val_accuracy: 0.9479
Epoch 6/20
47/47 [==============================] - 83s 2s/step - loss: 0.2395 - accuracy: 0.9037 - val_loss: 0.1317 - val_accuracy: 0.9479
Epoch 7/20
47/47 [==============================] - 80s 2s/step - loss: 0.1203 - accuracy: 0.9600 - val_loss: 0.1287 - val_accuracy: 0.9375
Epoch 8/20
47/47 [==============================] - 84s 2s/step - loss: 0.2024 - accuracy: 0.9227 - val_loss: 0.1433 - val_accuracy: 0.9427
Epoch 9/20
47/47 [==============================] - 114s 2s/step - loss: 0.1538 - accuracy: 0.9403 - val_loss: 0.0854 - val_accuracy: 0.9635
Epoch 10/20
47/47 [==============================] - 86s 2s/step - loss: 0.1391 - accuracy: 0.9417 - val_loss: 0.1773 - val_accuracy: 0.9271
Epoch 11/20
47/47 [==============================] - 84s 2s/step - loss: 0.0745 - accuracy: 0.9701 - val_loss: 0.0800 - val_accuracy: 0.9688
Epoch 12/20
47/47 [==============================] - 82s 2s/step - loss: 0.0782 - accuracy: 0.9729 - val_loss: 0.0600 - val_accuracy: 0.9792
Epoch 13/20
...
Epoch 19/20
47/47 [==============================] - 89s 2s/step - loss: 0.0645 - accuracy: 0.9756 - val_loss: 0.0346 - val_accuracy: 0.9948
Epoch 20/20
47/47 [==============================] - 3601s 78s/step - loss: 0.0393 - accuracy: 0.9858 - val_loss: 0.0223 - val_accuracy: 0.9948
```

Milestone 5: Model Deployment

Activity 1: Model Evaluation

As we have the record of accuracy stored in the history variable. We can try visualising the performance of our model.

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

✓ 13.1s

```
14/14 [==============================] - 12s 892ms/step - loss: 0.0606 - accuracy: 0.9791
```

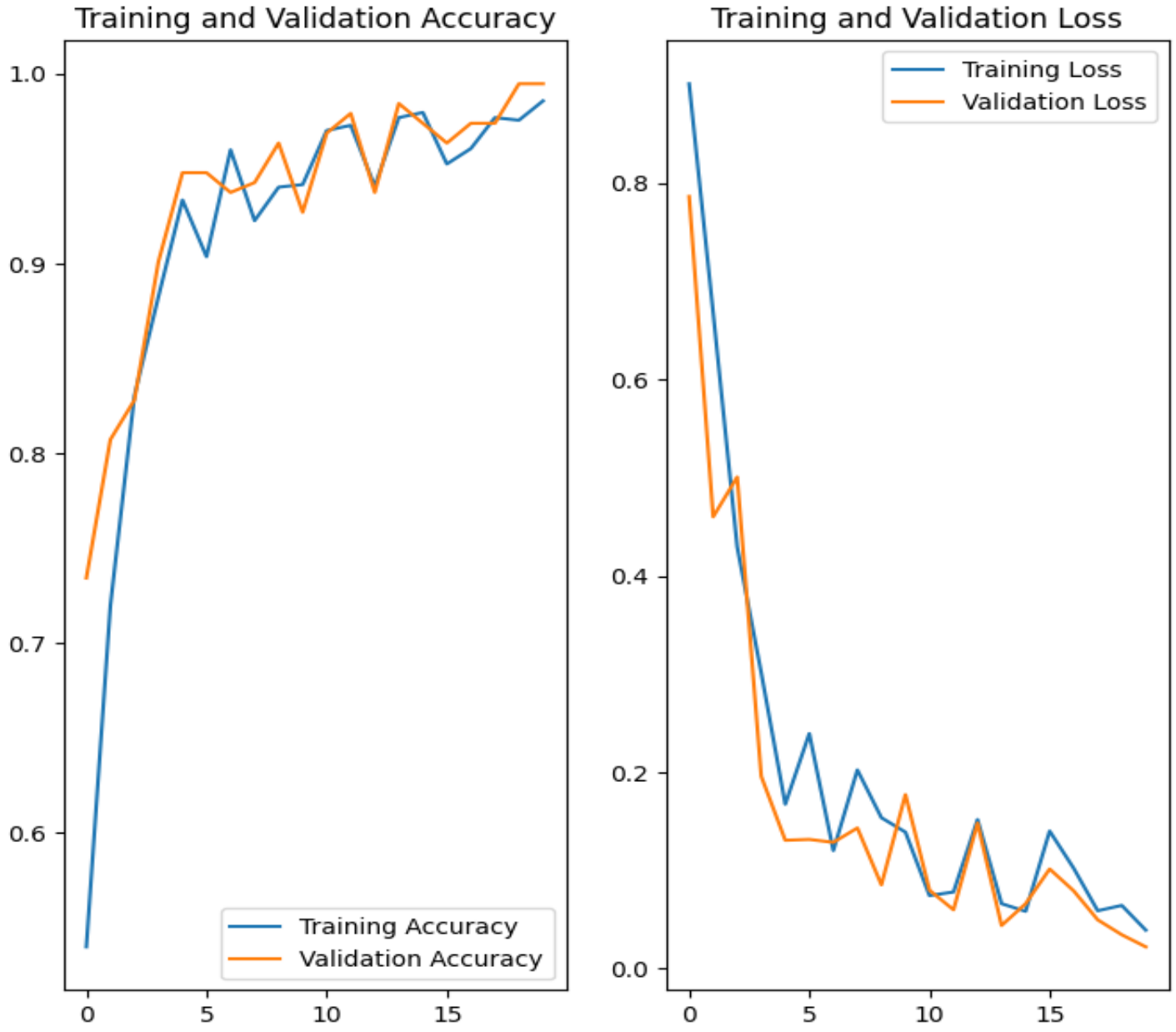
Activity 2: Plotting the Accuracy and loss curves

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

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

✓ 0.0s

```
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()
```

The above code plots two graphs, one graph b/w Training and Validation Accuracy and the other graphs b/w Training and Validation Loss.



Activity3: Model Evaluation with Visualisation

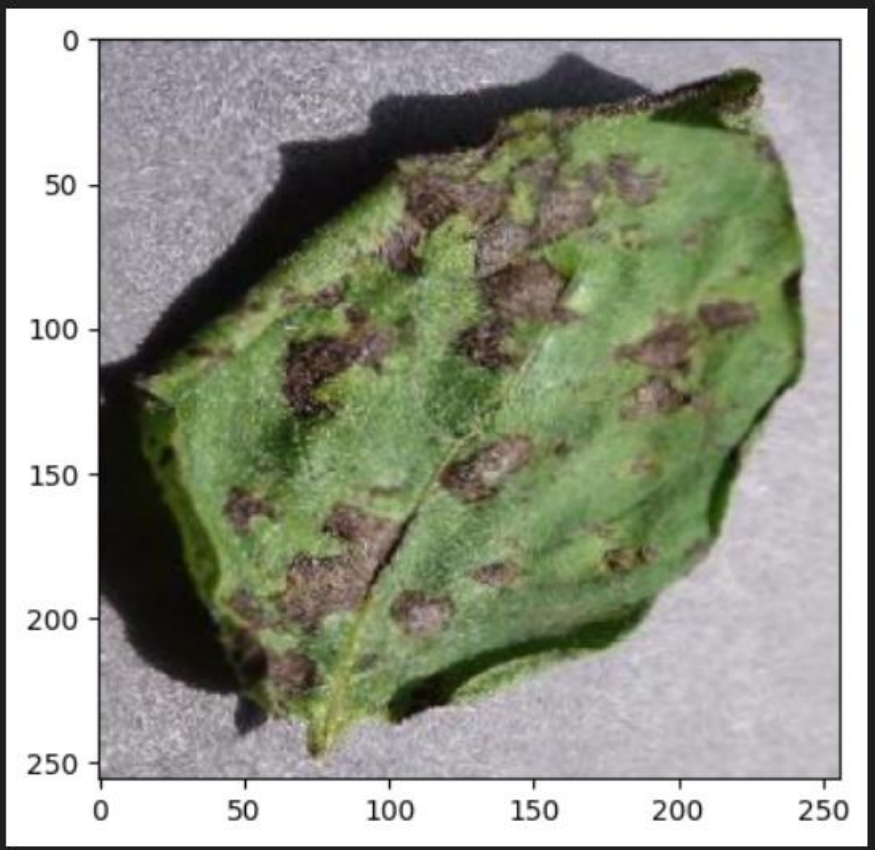
First we'll select one image from the test dataset and try to predict using the trained model. Then we'll try to print the actual class and predicted class of that particular image.

III. Prediction on a sample image

```
import numpy as np

for image_batch, label_batch in test_generator:

    first_image = image_batch[0]
    first_label = int(label_batch[0])

    print("First image to predict")
    plt.imshow(first_image)
    print(f"Actual label: {class_names[first_label]}")

    batch_prediction = model.predict(image_batch)
    print(f"Predicted label: {class_names[np.argmax(batch_prediction[0])]}")

    break
```

Each time you run this code; you'll get a different image to predict.

```
First image to predict
Actual label: Potato___Early_blight
1/1 [==============================] - 0s 455ms/step
Predicted label: Potato___Early_blight
```



Now let us write a function for prediction, which returns the predicted class and the confidence of its prediction.

Let's write a function for inference

```python
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    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
```

17] ✓ 0.0s Python

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

for images, labels in test_generator:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

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

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

        plt.axis("off")
    break
```

] ✓ 2.3s Python

```
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 31ms/step
```

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.91%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.99%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.89%

Actual: Potato___Late_blight,
Predicted: Potato___healthy.
Confidence: 93.81%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 100.0%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.94%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 100.0%

Activity 4: Saving the model

IV. Saving the Model

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

0.1s Python

This allows us to save the model and use it for the next step for deployment or integrating with the frontend.

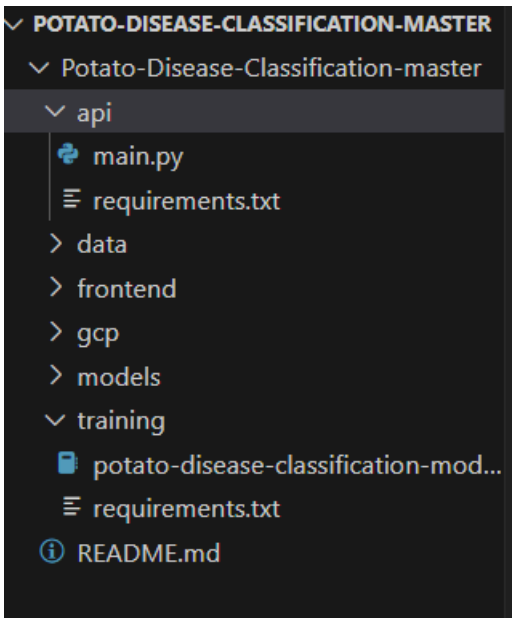
Activity 5: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built.

We will be using the fast Api package for our website development.

The "fast" API in Python refers to a framework that enables the easy deployment of serverless functions on cloud platforms. It abstracts away the complexities of serverless computing, allowing developers to focus on writing functions in Python and seamlessly deploying them as serverless applications without the need to manage infrastructure.

Activity 4.1: Create a main.py file and import necessary packages:




```
from fastapi import FastAPI, File, UploadFile
from fastapi.middleware.cors import CORSMiddleware
from io import BytesIO
from PIL import Image

import uvicorn
import numpy as np
import tensorflow as tf
```

```
from google.cloud import storage
from PIL import Image
import tensorflow as tf
import numpy as np
```

These package especially fast Api and some packages like node.js shows different not found error for that you need to download necessary files in the desktop u are using.

Activity 4.2: Defining class names and loading the model:

```
MODEL = tf.keras.models.load_model("../models/potatoes.h5")
CLASS_NAMES = ['Potato Early Blight', 'Potato Late Blight', 'Potato Healthy']

def read_file_as_image(file) -> np.ndarray:
    image = np.array(Image.open(BytesIO(file)))

    return image
```

Activity 4.3: Accepting the input from user and prediction

```
app = FastAPI()

origins = [
    "http://localhost",
    "http://localhost:3000"
]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

```python
@app.post("/predict")
async def predict(file: UploadFile):

    image = read_file_as_image(await file.read())
    img_batch = np.expand_dims(image, 0)

    predictions = MODEL.predict(img_batch)

    predicted_class = CLASS_NAMES[np.argmax(predictions[0])]
    confidence = round(float(np.max(predictions[0])) * 100, 2)

    return {'class': predicted_class, 'confidence': confidence}


if __name__ == "__main__":
    uvicorn.run(app, host="localhost", port=8000)
```

Activity 4.4: Accepting the input from user and prediction and run the main.py in gcp file.

```python
def predict(request):
    global model
    if model is None:
        download_blob(
            BUCKET_NAME,
            "models/potatoes.h5",
            "/tmp/potatoes.h5",
        )
        model = tf.keras.models.load_model("/tmp/potatoes.h5")

    image = request.files["file"]

    image = np.array(Image.open(image).convert("RGB").resize((256, 256))) # image resizing

    image = image/255 # normalize the image in 0 to 1 range

    img_array = tf.expand_dims(image, 0)
    predictions = model.predict(img_array)

    print("Predictions:",predictions)

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

    # set CORS headers
    headers = {
        'Access-Control-Allow-Origin': '*'
    }

    response = {"class": predicted_class, "confidence": confidence}

    return (response, 200, headers)
```

Activity 4.5: navigating to terminal to run the web server

```
## Front End Usage :

1. Go to the **frontend** directory:

`cd frontend`

2. Install dependencies:

`npm install`

3. Run the app:

`npm run start`
```

First, we go to terminal and navigate to the frontend directory and use npm run start in terminal to host and start the website.

```
PS C:\Users\pavan\Downloads\Potato-Disease-Classification-master> cd Potato-Disease-Classification-master
PS C:\Users\pavan\Downloads\Potato-Disease-Classification-master\Potato-Disease-Classification-master> cd frontend
PS C:\Users\pavan\Downloads\Potato-Disease-Classification-master\Potato-Disease-Classification-master\frontend> npm run start
Compiled successfully!
```

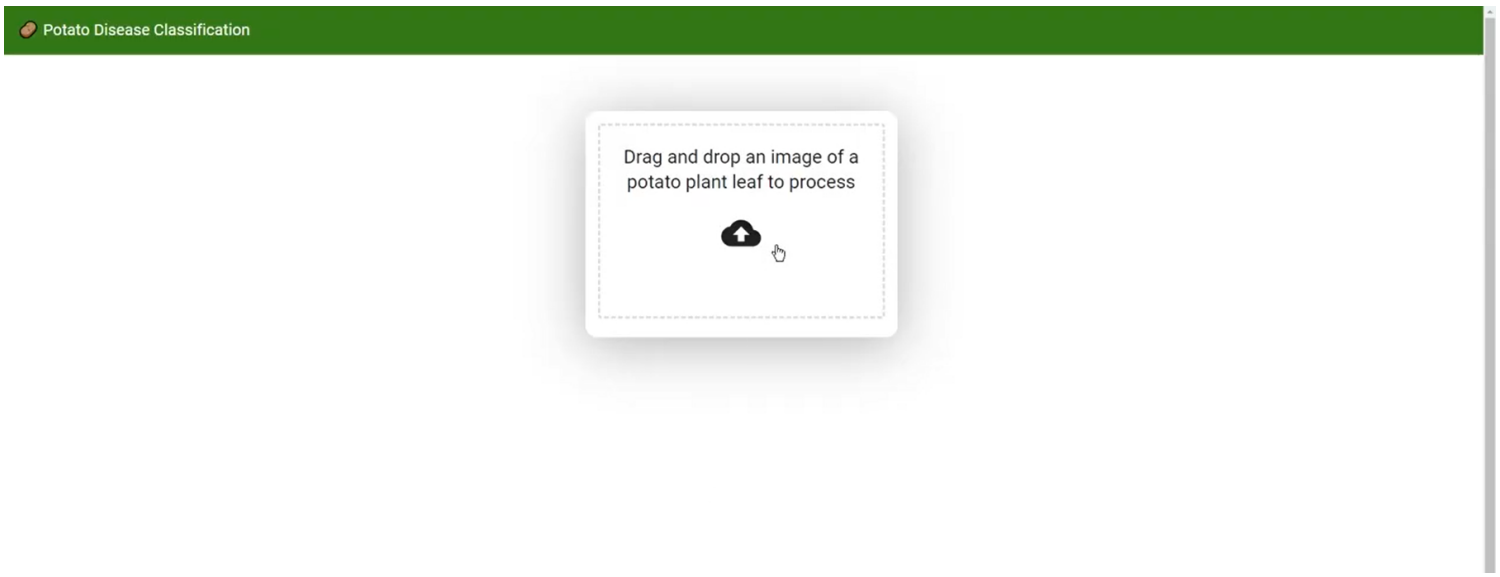
You could get errors on npm like npm not defined error for that u need to install the npm in desktop and run the npm install command in terminal.
https://nodejs.org/en

Thus, we will get a window like below after successful compiling of website.

```
You can now view frontend in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://192.168.29.203:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

HOW DOES YOUR WEBSITE LOOK LIKE ?

- Before uploading the picture:



- After uploading the picture: