End-to-end deep learning project for classifying potato diseases.

The objective of this project is to develop an end-to-end deep learning solution for classifying potato leaf images into three categories: healthy, early blight, and late blight. The proposed solution involves the use of convolutinal neural networks (CNNs) 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:
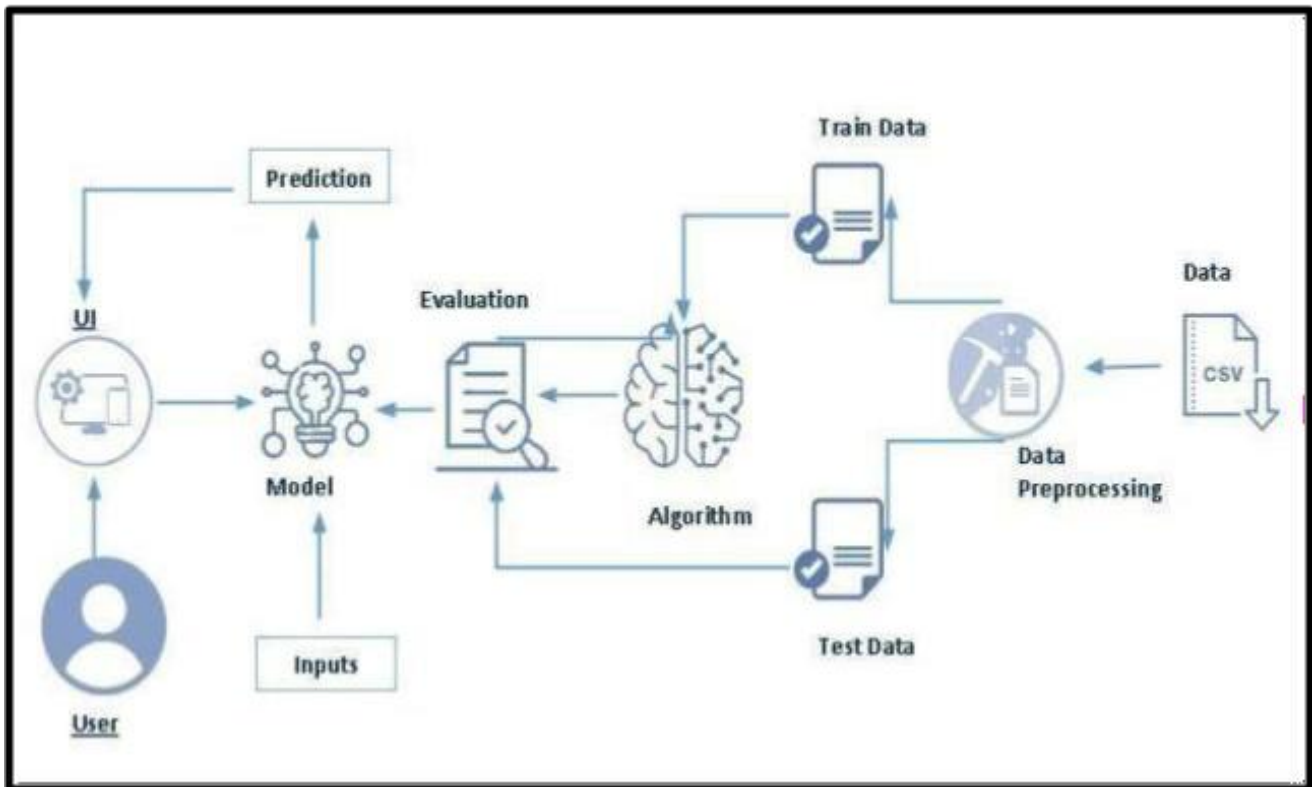
Project Flow:

- The user interacts with the UI to upload the file.
- Uploaded input is analysed by the model which is integrated/developed by you.
- Once the model analyses the input the prediction is showcased on the

UI. To accomplish this, we have to complete all the activities listed below,

- 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 Objectives:

By the end of this project you will:

- know fundamental concepts and techniques of Convolutional Neural Network.
- Gain a broad understanding of image data.
- Knowhow to pre-process/clean the data using different data preprocessing techniques.
- Know how to build a web application using the Streamlit framework.

Project Structure:

Create the Project folder which contains files as shown below
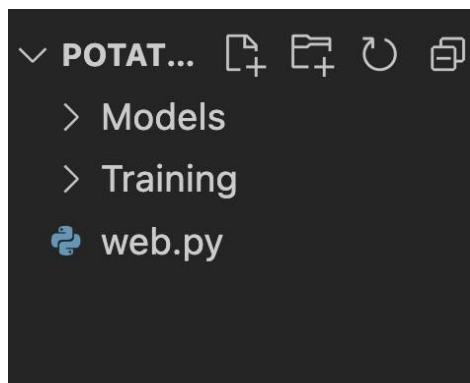
- We are building a Streamlit application which needs a python script web.py for a website.
- Model folder contains your saved models.
- The Training folder contains your data and code for building/training/testing the model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

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

1. Accurate prediction: The predictor 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 predictor must have a user-friendly interface that is easy to navigate and understand. 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 must be able to scale up based on the prediction from our product. The model should be able to handle any size of data without compromising on its accuracy or efficiency.

Activity 3: Literature Survey (Student Will Write)

A literature survey would involve researching and reviewing existing studies, articles, and other publications on the topic of project. The survey would aim to gather information on current systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous projects, and any relevant data or findings that could inform the design and implementation of the current project.
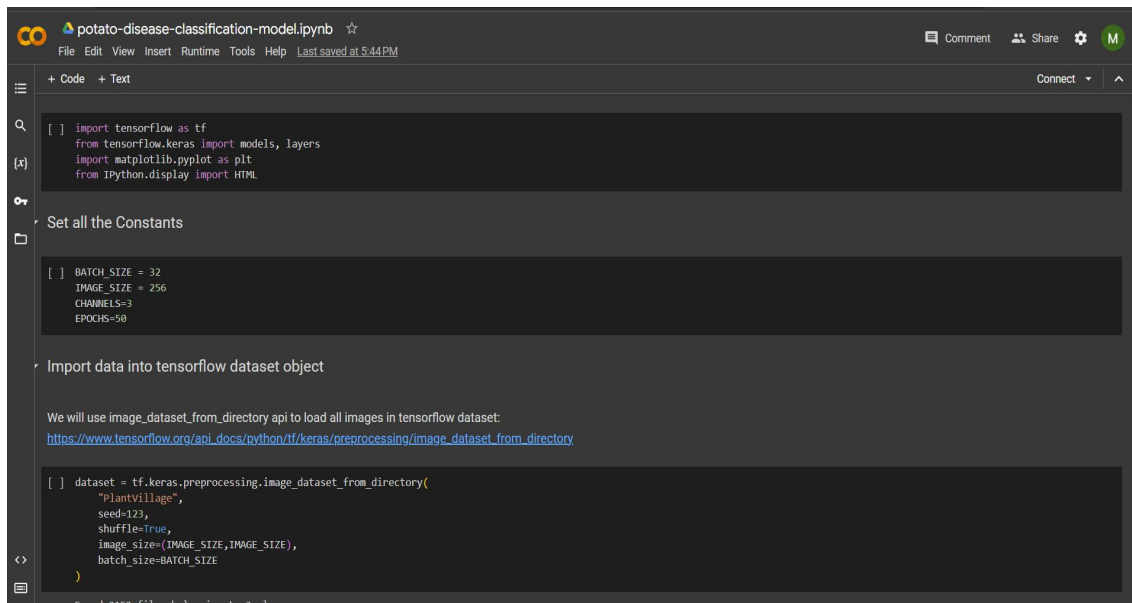
Activity 4: Social or Business Impact.

1. Improved potato crop yields: The potato leaf disease predictor can help farmers optimize their potato crop yields by providing accurate results. This can lead to increased crop yields and better crop quality, which can positively impact the income of farmers and the availability of good quality potatoes for consumers.

2. Reduction in production costs: The predictor can also help farmers reduce production costs by providing insights into optimal planting. This can lead to better resource allocation, reduced waste, and increased profitability for farmers.

Milestone 2: Data Collection & Preparation

DL depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this secti

Activity 1.1: Importing the libraries

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 of the dataset, we can use .class_names.

Activity 2.2: Visualizing the data

```
[ ]  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")
```

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.

The spilt will be in 8:1:1 ratio : train : test : validation respectively.

```
[ ]  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
```

Activity 4: Optimizing, Resize, Rescale and Augmentation of the data

```
[ ] 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)
```

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

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

Milestone 4: Model Building

Activity 1: Building a model

Now our data is ready and it's time to build the model.

Here we are going to build our model layer by layer using .Sequential() from keras.

Let's go !

```
[ ] 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()
```

```
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 (MaxPooling2   (32, 62, 62, 64)          0
_____
conv2d_2 (Conv2D)              (32, 60, 60, 64)          36928
_____
max_pooling2d_2 (MaxPooling2   (32, 30, 30, 64)          0
_____
conv2d_3 (Conv2D)              (32, 28, 28, 64)          36928
_____
max_pooling2d_3 (MaxPooling2   (32, 14, 14, 64)          0
_____
conv2d_4 (Conv2D)              (32, 12, 12, 64)          36928
_____
max_pooling2d_4 (MaxPooling2   (32, 6, 6, 64)            0
_____
conv2d_5 (Conv2D)              (32, 4, 4, 64)            36928
_____
max_pooling2d_5 (MaxPooling2   (32, 2, 2, 64)            0
_____
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
```

Activity 2: Compiling the model

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

Activity 3: Training the model

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

```
Epoch 1/50
54/54 [==============================] - 20s 255ms/step - loss: 0.8802 - accuracy: 0.5341 - val_loss: 0.8462 - val_accuracy: 0.5938
Epoch 2/50
54/54 [==============================] - 11s 196ms/step - loss: 0.6033 - accuracy: 0.7396 - val_loss: 0.6225 - val_accuracy: 0.6979
Epoch 3/50
54/54 [==============================] - 9s 172ms/step - loss: 0.3647 - accuracy: 0.8403 - val_loss: 0.3065 - val_accuracy: 0.8802
Epoch 4/50
54/54 [==============================] - 10s 176ms/step - loss: 0.2776 - accuracy: 0.8999 - val_loss: 0.2702 - val_accuracy: 0.8750
Epoch 5/50
54/54 [==============================] - 10s 179ms/step - loss: 0.2448 - accuracy: 0.8953 - val_loss: 0.1857 - val_accuracy: 0.9062
Epoch 6/50
54/54 [==============================] - 9s 174ms/step - loss: 0.2020 - accuracy: 0.9144 - val_loss: 0.2987 - val_accuracy: 0.9115
Epoch 7/50
54/54 [==============================] - 10s 185ms/step - loss: 0.1751 - accuracy: 0.9288 - val_loss: 0.1854 - val_accuracy: 0.9375
Epoch 8/50
54/54 [==============================] - 10s 180ms/step - loss: 0.1436 - accuracy: 0.9444 - val_loss: 0.2273 - val_accuracy: 0.9167
Epoch 9/50
54/54 [==============================] - 10s 175ms/step - loss: 0.1128 - accuracy: 0.9583 - val_loss: 0.1425 - val_accuracy: 0.9479
Epoch 10/50
54/54 [==============================] - 10s 179ms/step - loss: 0.1218 - accuracy: 0.9549 - val_loss: 0.2310 - val_accuracy: 0.9115
Epoch 11/50
54/54 [==============================] - 10s 179ms/step - loss: 0.1524 - accuracy: 0.9398 - val_loss: 0.0774 - val_accuracy: 0.9688
Epoch 12/50
54/54 [==============================] - 10s 186ms/step - loss: 0.1062 - accuracy: 0.9578 - val_loss: 0.1787 - val_accuracy: 0.9427
Epoch 13/50
54/54 [==============================] - 9s 172ms/step - loss: 0.1299 - accuracy: 0.9549 - val_loss: 0.0929 - val_accuracy: 0.9531
Epoch 14/50
54/54 [==============================] - 9s 169ms/step - loss: 0.0971 - accuracy: 0.9601 - val_loss: 0.1230 - val_accuracy: 0.9531
```

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_ds)

     8/8 [==============================] - 1s 14ms/step - loss: 0.0063 - accuracy: 1.0000
```

You can see above that we get 100.00% accuracy for our test dataset. This is considered to be a pretty good accuracy

```
[ ]  scores

     [0.006251859944313765, 1.0]
```

Scores is just a list containing loss and accuracy value

▾ Plotting the Accuracy and Loss Curves

```
[ ]  history

     <tensorflow.python.keras.callbacks.History at 0x7f3d98437e50>
```

You can read documentation on history object here: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/History

```
[ ]  history.params

     {'verbose': 1, 'epochs': 50, 'steps': 54}
```

```
[ ]  history.history.keys()

     dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

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

     list
```

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

     50
```

```
[ ]  history.history['loss'][:5] # show loss for first 5 epochs

     [0.8801848292350769,
      0.6033139228820801,
      0.3646925389766693,
      0.2776017189025879,
      0.24480397999286652]
```

```
history.history['accuracy']
```

```
[0.5214120149612427,
 0.7685185074806213,
 0.8449074029922485,
 0.8894675970077515,
 0.9224537014961243,
 0.9068287014961243,
 0.9259259104728699,
 0.9259259104728699,
 0.9357638955116272,
 0.9461805820465088,
 0.9560185074806213,
 0.9502314925193787,
 0.9542824029922485,
 0.9502314925193787,
 0.9589120149612427,
 0.9554398059844971,
 0.9513888955116272,
 0.9600694179534912,
 0.9554398059844971,
 0.9618055820465088,
 0.9728009104728699,
 0.9716435074806213,
 0.9629629850387573,
 0.9641203880310059,
 0.9710648059844971,
 0.9577546119689941,
 0.9751157164573669,
 0.9820601940155029,
 0.9722222089767456,
 0.9826388955116272,
 0.9826388955116272,
 0.9803240895271301,
 0.9785879850387573,
 0.9768518805503845,
 0.9774305820465088,
 0.9768518805503845,
 0.984375,
 0.9780092835426331,
 0.9855324029922485,
```

history.history['accuracy'] contains the track of accuracy achieved while you were training our model.

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

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

```python
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/wTraining and Validation Loss.
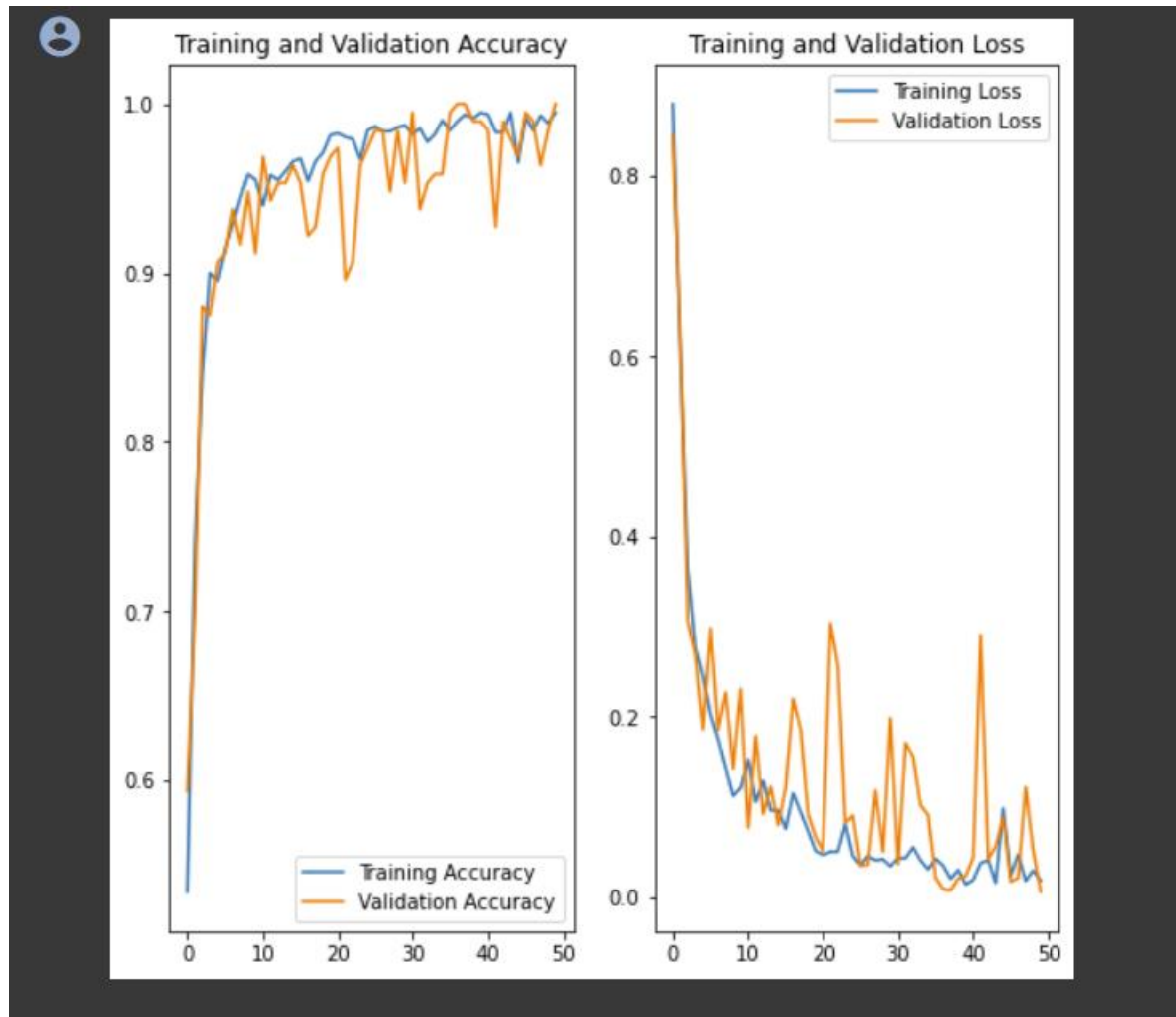
Activity 2: Model Evaluation with Visualization

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

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

```
[ ]  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: Potato___Early_blight
     predicted label: Potato___Early_blight
```
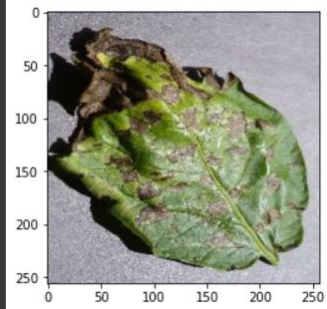

Now let us write a function for prediction, which returns the predicted class and the confidence of its prediction.

```python
[ ] 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
```

Evaluating a set of images from the test data set and visualizing there prediction and confidence.

```python
[ ] 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")
```

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.98%

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

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

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

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.92%

Actual: Potato___Early_blight,
Predicted: Potato___Early_blight.
Confidence: 99.87%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 99.88%

Actual: Potato___healthy,
Predicted: Potato___healthy.
Confidence: 99.53%

Actual: Potato___Late_blight,
Predicted: Potato___Late_blight.
Confidence: 88.21%

Activity 3: Save the model

This will allow us to save the .h5 format of the model.

Activity 4: Integrate with Web Framework

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

We will be using the streamlit package for our website development.

Streamlit is a free and open-source framework to rapidly build and share beautiful machine learning and data science web apps.

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

```
∨ POTAT...
  > Models
  > Training
  web.py
```
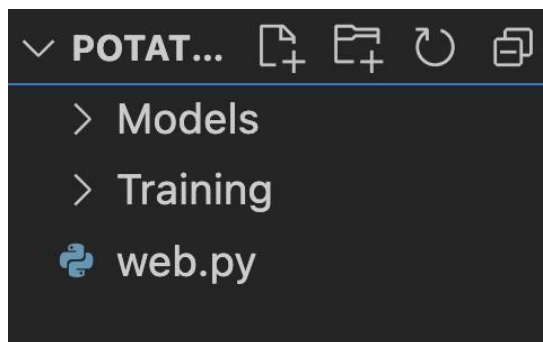
```python
import streamlit as st
import tensorflow as tf
from PIL import Image, ImageOps
import numpy as np
from io import BytesIO
```

Activity 4.2: Defining class names and loading the model:

```python
st.cache(allow_output_mutation=True)
CLASS_NAMES = ["Early Blight" , "Late Blight" , "Healthy"]
def load_model():
  model=tf.keras.models.load_model('./Models/potato.h5')
  return model
model = load_model()
```

Activity 4.3: Accepting the input from user and prediction

```python
st.write("""# Potato Leaf Disease Classification""")

file = st.file_uploader("Please upload an brain scan file", type=["jpg", "png"])

def import_and_predict(image_data, model):
        size = (255,255)
        image = ImageOps.fit(image_data, size, Image.ANTIALIAS)
        image = np.asarray(image)
        img_reshape = np.expand_dims(image,0)
        prediction = model.predict(img_reshape)
        return prediction
```

```python
if file is None:
    st.text("Please upload an image file")
else:
    image = Image.open(file)
    st.image(image, use_column_width=True)
    predictions = import_and_predict(image, model)
    index = np.argmax(predictions[0])
    predicted_class = CLASS_NAMES[index]
    confidence = np.max(predictions[0])
    st.write(predicted_class)
    st.write(confidence)
    print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(CLASS_NAMES[np.argmax(confidence)], 100 * np.max(confidence))
)
```

```
You can now view your Streamlit app in your browser.

Local URL: http://localhost:8501
Network URL: http://192.168.1.11:8501

For better performance, install the Watchdog module:

$ xcode-select --install
$ pip install watchdog
```

Milestone 5: Building Flask application

After the model is built, we will be integrating it to a web application so that normal users can also use it. The new users need to initially register in the portal. After registration users can login to browse the images to detect the condition of potatoes.

Activity 1: Build a python application.

Step 1: Load the required packages.

```python
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask,render_template,request
import os
import numpy as np
import pickle
```

Step 2: Initialize the flask app, load the model and configure the html pages Instance of Flask is created and the model is loaded using load_model from keras.

```python
app = Flask(__name__)
model = load_model(r"Potato_model.h5",compile = False)

@app.route('/')
def index():
    return render_template("index.html")

@app.route('/predict',methods = ['GET','POST'])
```

Step 3: Pre-process the frame and run.

```
def upload():
    if request.method=='POST':
        f = request.files['image']
        basepath=os.path.dirname(__file__)
        filepath = os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)
        img = image.load_img(filepath,target_size =(240,240))
        x = image.img_to_array(img)
        x = np.expand_dims(x,axis = 0)
        pred =np.argmax(model.predict(x),axis=1)
        index =['EarlyBlight','Healthy','LateBlight']
        text="The potato is : "+index[pred[0]]
    return text


if __name__=='__main__':
    app.run(debug=True)
```

Run the flask application using the run method. By default, the flask runs on port 5000. If the port is to be changed, an argument can be passed and the port can be modified.

Activity 2: Build the HTML page and execute

Build the UI where a home page will have details about the application and prediction page where a user is allowed to browse an image and get the predictions of images.

Step 1: Run the application

In the anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed, the localhost is activated on port 5000 and can be accessed through it.

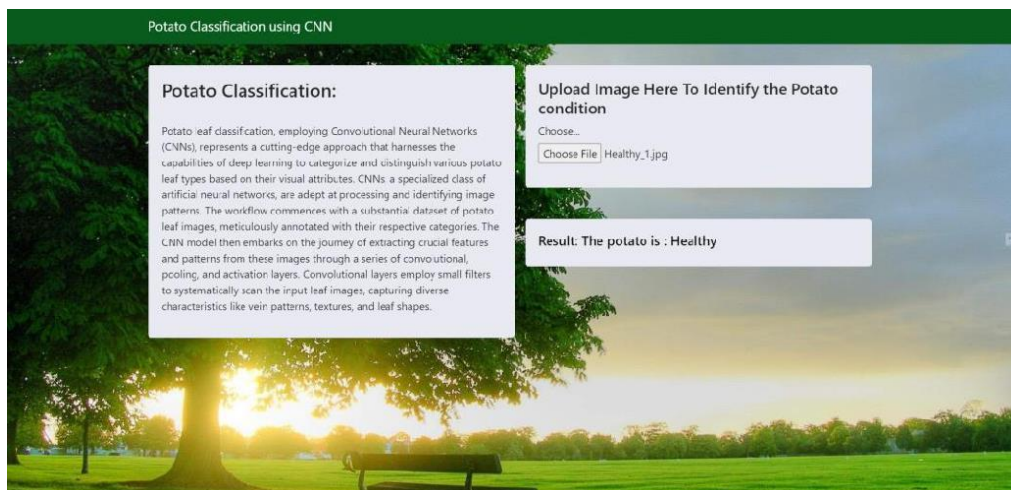Step 2: Open the browser and navigate to localhost:5000 to check your application The home page looks like this:
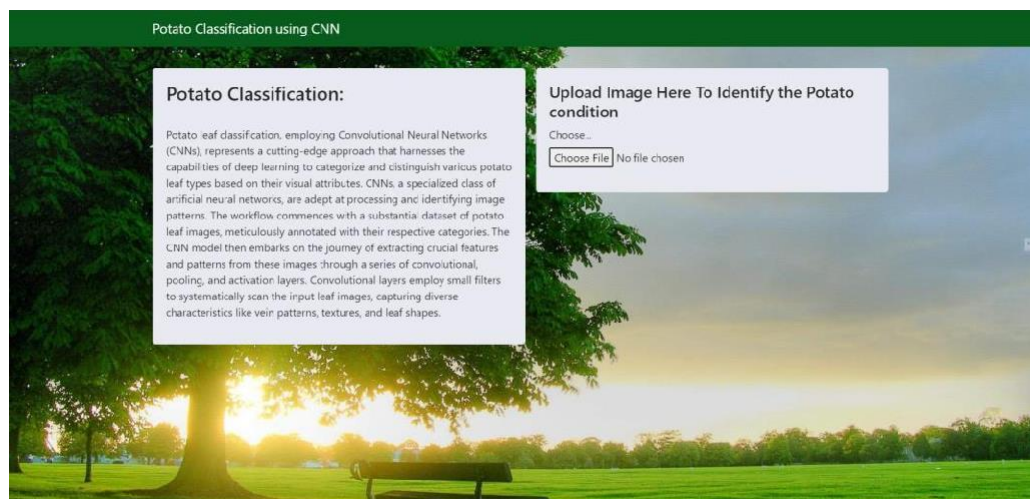
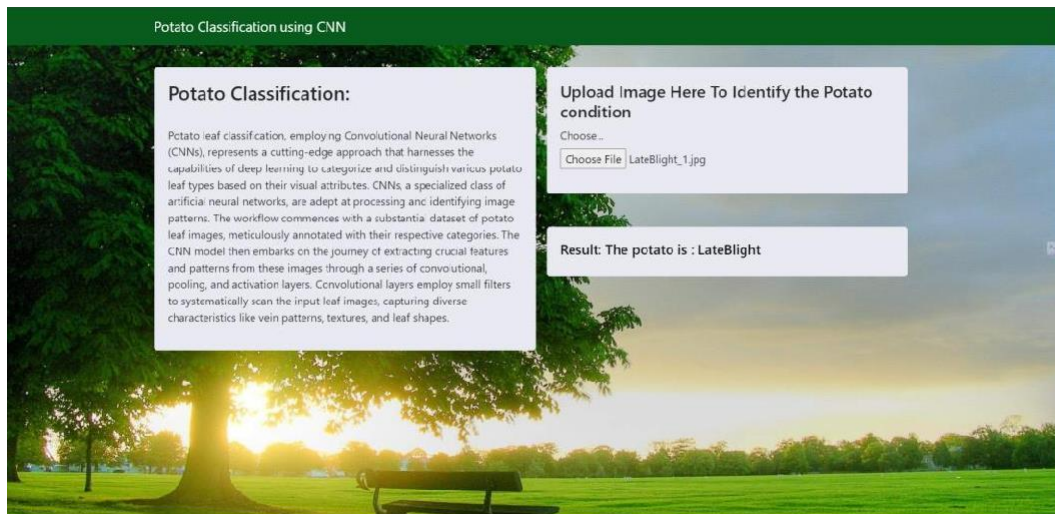Click on choose the file to upload and select an image to be classified

Classification-1 :
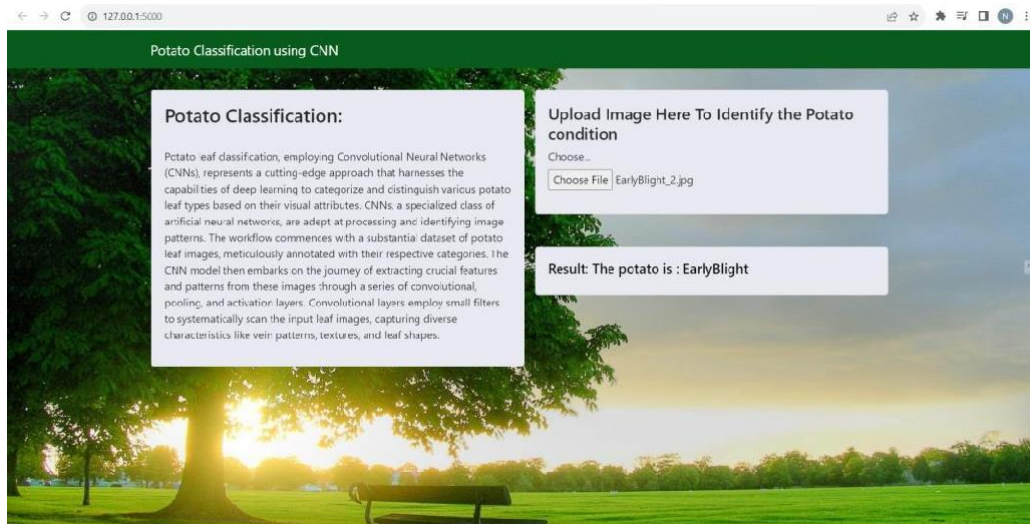
Classification-2 :

Classification-3:

Conclusion: By clicking the predict button we will predict the condition of using our ResNet50 model.