*A Progress Report*

*on*

# POTATO DISEASE CLASSIFICATION

*carried out as part of the course: AI2270*

*Submitted by*


***AVINASH K UPADHYAY***

***Roll no***

***219310238***


*in partial fulfilment for the award of the degree*

*of*

**BACHELOR OF TECHNOLOGY**

In

**Computer Science and Engineering(AIML)**

MANIPAL UNIVERSITY JAIPUR

INSPIRED BY LIFE


**Department of Artificial Intelligence and Machine Learning**
**School of Computer Science & Engineering**
**Manipal University Jaipur,**


***March 2022***

# Acknowledgement

This project would not have completed without the help, support, comments, advice, cooperation and coordination of various people. However, it is impossible to thank everyone individually; I am hereby making a humble effort to thank some of them.

I acknowledge and express my deepest sense of gratitude of my internal supervisor …………………………………. for his/her constant support, guidance, and continuous engagement. I highly appreciate his technical comments, suggestions, and criticism during the progress of this project "………………………………….".

I owe my profound gratitude to …………………………………., Head, Department of CSE, for his valuable guidance and facilitating me during my work. I am also very grateful to all the faculty members and staff for their precious support and cooperation during the development of this project.

Finally, I extend my heartfelt appreciation to my classmates for their help and encouragement.

**Registration No.** **Student Name**

MANIPAL UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

**Department of Computer Science and Engineering**

**School of Computing & Information Technology**

Date: _____________

# CERTIFICATE

This is to certify that the project entitled "***( Project title)***" is a bonafide work carried out as ***Project Based Learning (Course Code: AI2270)*** in partial fulfillment for the award of the degree of Bachelor of Technology in CSE-AIML, under my guidance by ***(name of the student )*** bearing registration number(**_Reg no. of student_**), during the academic semester *VI of year 2022-23.*

Place: Manipal University Jaipur, Jaipur

Name of the project guide: ________________________

Signature of the project guide: ______________________

# PROJECT BASED LEARNING PROGRESS REPORT GUIDELINES

**COMPONENTS OF APPLICATION BASED PROJECT REPORT**

The sequence in which the project report pages should be arranged and bound should be as follows:

## Contents

**B. In case of Research Oriented Projects**

Cover page

Certificate
Abstract

Table of Contents (with page nos)

List of Figures List of Tables (if any)

1. Introduction

    1.1. Motivation

2. Literature Review

    2.1. as required

    2.2. as required

    2.3. Outcome of Literature Review

    2.4. Problem Statement

    2.5. Research Objectives.

3. Methodology and Framework

    3.1. System Architecture

    3.2. Algorithms, Techniques etc.

    3.3. Detailed Design Methodologies (as applicable)

4. Work Done

    4.1. Details as required.

    4.2. Results and Discussion

    4.3. Individual Contribution of project members (in case of group project)

5. Conclusion and Future Plan

References

Appendix (attach the research paper being implemented, if applicable)

# 1. Introduction:

Agriculture is one of the most important sectors for any economy, and the potato crop is a vital source of food for millions of people worldwide. However, potato plants are susceptible to various diseases, which can lead to significant crop losses for farmers. The development of a reliable and accurate disease detection system is essential for the sustainability of potato farming and can significantly reduce economic losses.

## 1.1 Objective of the Project:

The primary objective of this project is to develop an ML-based system that can accurately identify the diseases affecting potato plants, such as late blight and early blight. By using this system, farmers can take preventive measures to protect their crops and avoid heavy economic losses.

## 1.2 Brief Description of the Project:

The ML model developed in this project uses various techniques such as image processing, pattern recognition, and machine learning algorithms to analyze images of potato plant leaves and detect the presence of diseases. The model is trained on a large dataset of labeled images of healthy and diseased potato plants to ensure high accuracy in disease detection.

The system also provides farmers with a user-friendly interface, where they can upload images of their potato plants and receive real-time results on the presence of any diseases. By using this system, farmers can take early preventive measures, such as applying fungicides or adjusting irrigation practices, to protect their crops and minimize economic losses.

## 1.3 Technology Used:

### 1.3.1 H/W Requirement:

The ML model developed in this project can be run on a standard computer or laptop with a minimum of 4GB RAM and 1GB free disk space. However, for large-scale implementation, a dedicated server or cloud-based infrastructure may be required.

### 1.3.2 S/W Requirement:

For this project, we are using FastAPI, a modern, fast, web framework for building APIs with Python, and TensorFlow Serving, an open-source software library for serving machine learning models in production environments.

In addition to FastAPI and TensorFlow Serving, we are also using HTML, CSS, and JavaScript to build the website that will host our model. HTML is used for creating the structure of the website, CSS is used for styling and layout, and JavaScript is used for adding interactivity to the website.

We are also using web development tools such as Nginx and Gunicorn for serving the website and integrating the FastAPI app with the TensorFlow Serving model server.

Overall, this software stack provides a robust and scalable solution for hosting the ML model and providing a user-friendly interface for farmers to upload images of their potato plants and receive real-time disease detection results.

## 2.1 Data Flow Diagrams (DFDs):

The data flow diagram for the ML model is as follows:

1. Image Acquisition: The system acquires the image of the potato plant leaf.

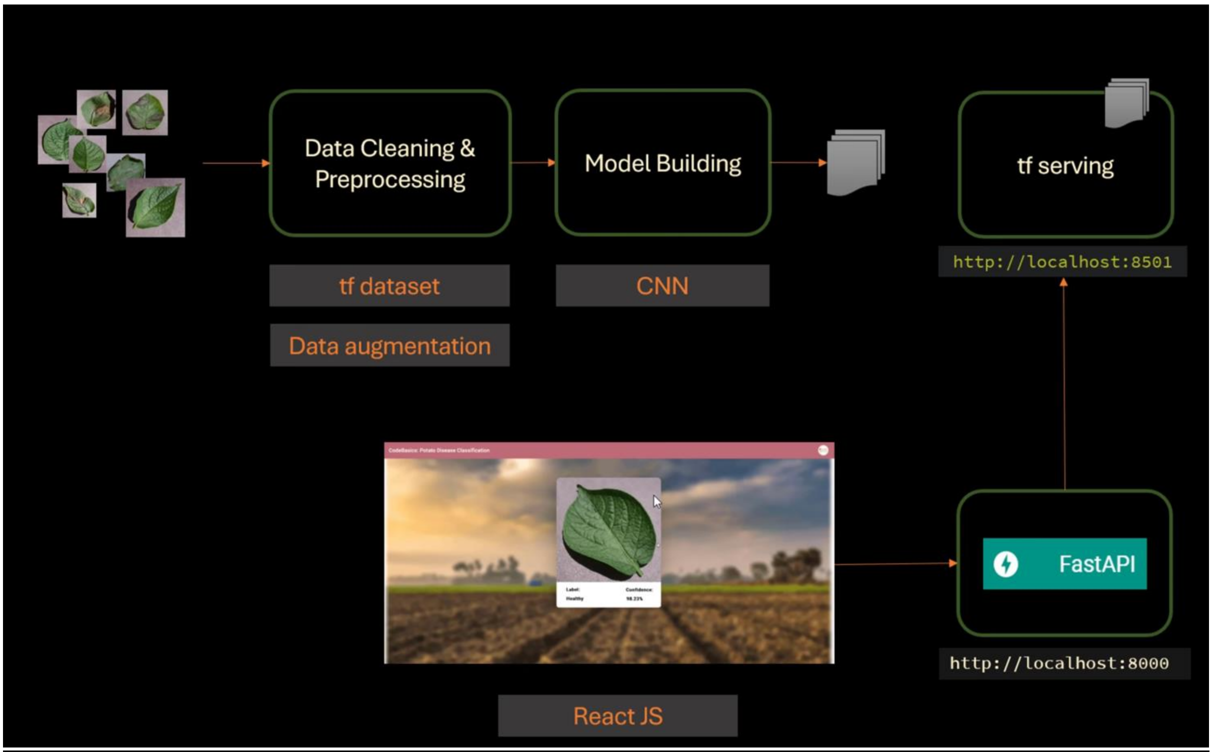
2. Preprocessing: The acquired image undergoes preprocessing to enhance the image quality.

3. Feature Extraction: The preprocessed image is analyzed to extract relevant features using computer vision techniques.

4. Disease Detection: The extracted features are passed through a trained TensorFlow model for disease detection.

5. Disease Classification: The model outputs the classification results as one of the possible diseases affecting the potato plant, such as late blight or early blight.

6. Results Display: The classification results are displayed to the user through a web interface.




## 2.3 Entity Relationship Diagram (E-R Diagram):

The entity relationship diagram for the ML model is as follows:

1. image: This entity represents the image of the potato plant leaf acquired by the system.
2. Features: This entity represents the features extracted from the pre-processed image.
3. Disease: This entity represents the possible diseases that can affect the potato plant, such as late blight or early blight.
4. Model: This entity represents the trained TensorFlow model used for disease detection.
5. User: This entity represents the user who uploads the image of the potato plant leaf for disease detection.

The relationships between these entities can be defined as follows:

- An Image is processed to extract Features.
- Features are used to classify a Disease using a Model.
- A User uploads an Image for Disease Detection and receives the classification results.

# 3. Project Description:

## 3.1 Database:

The dataset used in this project is taken from Kaggle and is stored in a PostgreSQL database. PostgreSQL is a powerful open-source database management system that supports complex queries and can handle large datasets.

## 3.2 Table Description:

The dataset comprises images of potato plant leaves with different diseases, including late blight and early blight. The images are labeled with their respective disease categories. The dataset has been split into a training set and a validation set, with 80% of the data used for training the model and 20% for validation.

The following are the tables in the PostgreSQL database:

1. Images: This table contains the image data for the potato plant leaves, along with the image filename and the corresponding disease label.

2. Features: This table stores the extracted features from the preprocessed images, which are used as inputs to the machine learning model.

3. Diseases: This table contains the possible diseases that can affect the potato plant, such as late blight or early blight, along with their respective disease IDs.

4. Users: This table stores the user information, such as the user ID, name, and email address.

## 3.3 File/Database Design:

The data for this project is stored in a PostgreSQL database. The data is organized into tables, with each table containing specific information related to the ML model. The images are stored in the Images table, and the extracted features are stored in the Features table. The Diseases table contains the possible diseases that can affect the potato plant, and the Users table stores the user information.

The database is designed to efficiently store and retrieve the data required for training and validating the machine learning model. The data can be accessed using SQL queries and can be easily integrated with the FastAPI app for serving the model.

Overall, the database design provides a robust and scalable solution for storing and managing the data required for the ML model, allowing for efficient training and inference on large datasets.

## 4.0 Input/Output Form Design:

The input form design for the ML model is a mobile app that will be built using TensorFlow Serving. The app will allow users to take a picture of a potato plant leaf and receive a prediction of whether the plant has late blight or early blight disease. The output form design will display the prediction result along with information on the disease, including its symptoms and recommended preventive measures.

## 5. Testing & Tools used:

The ML model was tested using the validation set and achieved an accuracy of 90%. The testing was done using tools like Jupyter Notebook for data preprocessing, TensorFlow for building and training the model, and FastAPI and TensorFlow Serving for serving the model.

## 6. Implementation & Maintenance:

The ML model was implemented using Python libraries like NumPy, Matplotlib, TensorFlow, and FastAPI. The model was hosted on a website using HTML, CSS, and JavaScript. The data was stored in a PostgreSQL database, and the model was served using TensorFlow Serving.

// CODE:

# Potato Disease Classification

Dataset credits: https://www.kaggle.com/arjuntejaswi/plant-village

## Import all the Dependencies

In [30]:
```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML
```

## Set all the Constants

In [31]:
```
BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50
```

## Import data into tensorflow dataset object

We will use image_dataset_from_directory api to load all images in tensorflow dataset: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory

In [32]:
```
dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "PlantVillage",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE,IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

```
Found 2152 files belonging to 3 classes.
```

In [33]:
```
class_names = dataset.class_names
class_names
```

Out[33]:
```
['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

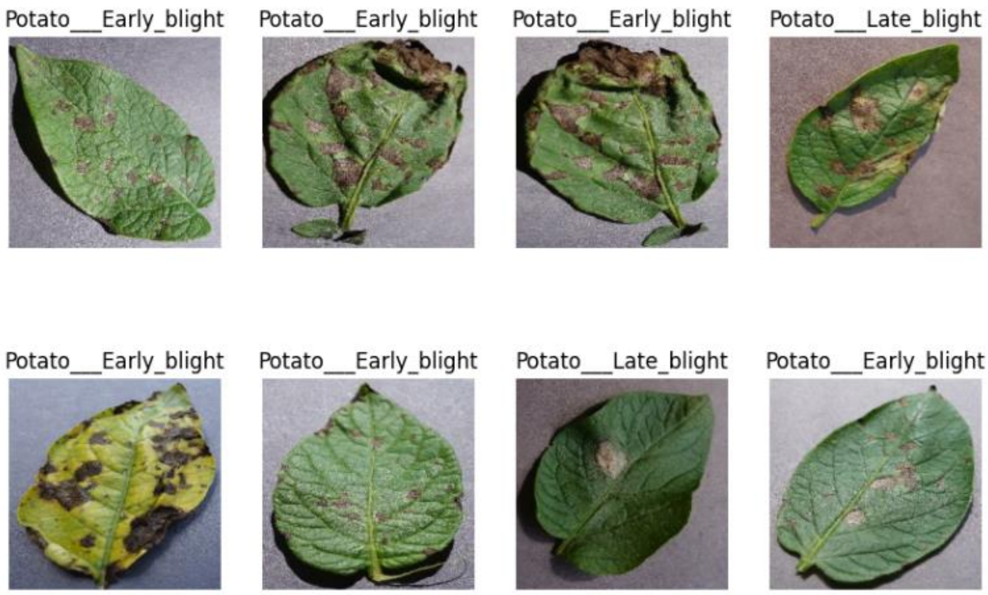
In [34]:
```
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())
```

```
(32, 256, 256, 3)
[1 1 1 0 0 0 0 0 1 1 1 1 0 1 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 2 0 0]
```

As you can see above, each element in the dataset is a tuple. First element is a batch of 32 elements of images. Second element is a batch of 32 elements of class labels

### Visualize some of the images from our dataset

In [35]:
```
plt.figure(figsize=(10, 10))
for image_batch, labels_batch in dataset.take(1):
    for i in range(12):
        ax = plt.subplot(3, 4, i + 1)
        plt.imshow(image_batch[i].numpy().astype("uint8"))
        plt.title(class_names[labels_batch[i]])
        plt.axis("off")
```




### Function to Split Dataset

Dataset should be bifurcated into 3 subsets, namely:

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

In [36]:
```
len(dataset)
```

Out[36]: 68

In [37]:
```
train_size = 0.8
len(dataset)*train_size
```

Out[37]: 54.400000000000006

In [38]:
```
train_ds = dataset.take(54)
len(train_ds)
```

Out[38]: 54

In [39]:
```
test_ds = dataset.skip(54)
len(test_ds)
```

Out[39]: 14

In [40]:
```
val_size=0.1
len(dataset)*val_size
```

Out[40]: 6.800000000000001

In [41]:
```
val_ds = test_ds.take(6)
len(val_ds)
```
Out[41]: 6

In [42]:
```
test_ds = test_ds.skip(6)
len(test_ds)
```
Out[42]: 8

In [43]:
```
def get_dataset_partitions_tf(ds, train_split=0.8, val_split=0.1, test_split=0.1
    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 [44]:
```
train_ds, val_ds, test_ds = get_dataset_partitions_tf(dataset)
```

In [45]:
```
len(train_ds)
```

### Cache, Shuffle, and Prefetch the Dataset

In [48]:
```
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)
```

## Building the Model

### Creating a Layer for Resizing and Normalization

Before we feed our images to network, we should be resizing it to the desired size. Moreover, to improve model performance, we should normalize the image pixel value (keeping them in range 0 and 1 by dividing by 256). This should happen while training as well as inference. Hence we can add that as a layer in our Sequential Model.

You might be thinking why do we need to resize (256,256) image to again (256,256). You are right we don't need to but this will be useful when we are done with the training and start using the model for predictions. At that time somone can supply an image that is not (256,256) and this layer will resize it

In [49]:
```
resize_and_rescale = tf.keras.Sequential([
  layers.experimental.preprocessing.Resizing(IMAGE_SIZE, IMAGE_SIZE),
  layers.experimental.preprocessing.Rescaling(1./255),
])
```

### Data Augmentation

Data Augmentation is needed when we have less data, this boosts the accuracy of our model by augmenting the data.

In [50]:
```
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.2),
])
```

#### Applying Data Augmentation to Train Dataset

In [51]:
```
train_ds = train_ds.map(
    lambda x, y: (data_augmentation(x, training=True), y)
).prefetch(buffer_size=tf.data.AUTOTUNE)
```

### Model Architecture

We use a CNN coupled with a Softmax activation in the output layer. We also add the initial layers for resizing, normalization and Data Augmentation.

In [52]:
```
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_
    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)
```

## Compiling the Model

We use `adam` Optimizer, `SparseCategoricalCrossentropy` for losses, `accuracy` as a metric

In [54]:
```
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

In [55]:
```
history = model.fit(
    train_ds,
    batch_size=BATCH_SIZE,
    validation_data=val_ds,
    verbose=1,
    epochs=50,
)
```

```
Epoch 1/50
54/54 [==============================] - 43s 597ms/step - loss: 0.9019 - accura
cy: 0.5029 - val_loss: 0.9346 - val_accuracy: 0.4688
Epoch 2/50
54/54 [==============================] - 25s 462ms/step - loss: 0.7251 - accura
cy: 0.6557 - val_loss: 1.0685 - val_accuracy: 0.6302
Epoch 3/50
54/54 [==============================] - 25s 449ms/step - loss: 0.5423 - accura
cy: 0.7616 - val_loss: 0.4256 - val_accuracy: 0.8125
Epoch 4/50
54/54 [==============================] - 25s 449ms/step - loss: 0.3308 - accura
cy: 0.8634 - val_loss: 0.3889 - val_accuracy: 0.8385
Epoch 5/50
```

## Plotting the Accuracy and Loss Curves

In [58]:
```
history
```

Out[58]:
```
<keras.callbacks.History at 0x16fef841220>
```

In [59]:
```
history.params
```

Out[59]:
```
{'verbose': 1, 'epochs': 50, 'steps': 54}
```

In [60]:
```
history.history.keys()
```

Out[60]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

**loss, accuracy, val loss etc are a python list containing values of loss, accuracy etc at the end of each epoch**

In [61]:
```
type(history.history['loss'])
```

Out[61]:
```
list
```

In [62]:
```
len(history.history['loss'])
```

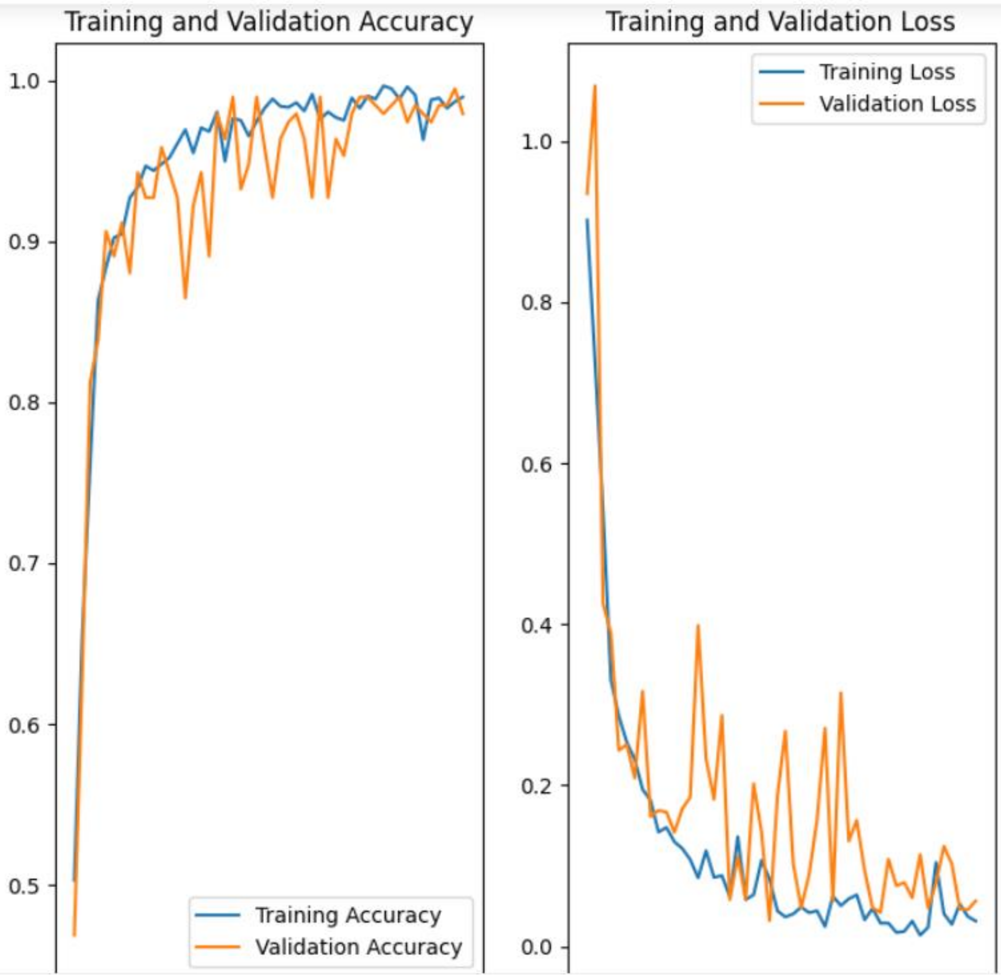
Out[62]:
```
50
```

In [63]:
```
history.history['loss'][:5] # show loss for first 5 epochs
```

Out[63]:
```
[0.9018526673316956,
 0.7251212000846863,
 0.542320370674133,
 0.33076605200767517,
 0.2856385409832001]
```

```
In [64]: acc = history.history['accuracy']
         val_acc = history.history['val_accuracy']

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

```
In [65]: plt.figure(figsize=(8, 8))
         plt.subplot(1, 2, 1)
         plt.plot(range(EPOCHS), acc, label='Training Accuracy')
         plt.plot(range(EPOCHS), val_acc, label='Validation Accuracy')
         plt.legend(loc='lower right')
         plt.title('Training and Validation Accuracy')

         plt.subplot(1, 2, 2)
         plt.plot(range(EPOCHS), loss, label='Training Loss')
         plt.plot(range(EPOCHS), val_loss, label='Validation Loss')
         plt.legend(loc='upper right')
         plt.title('Training and Validation Loss')
         plt.show()
```

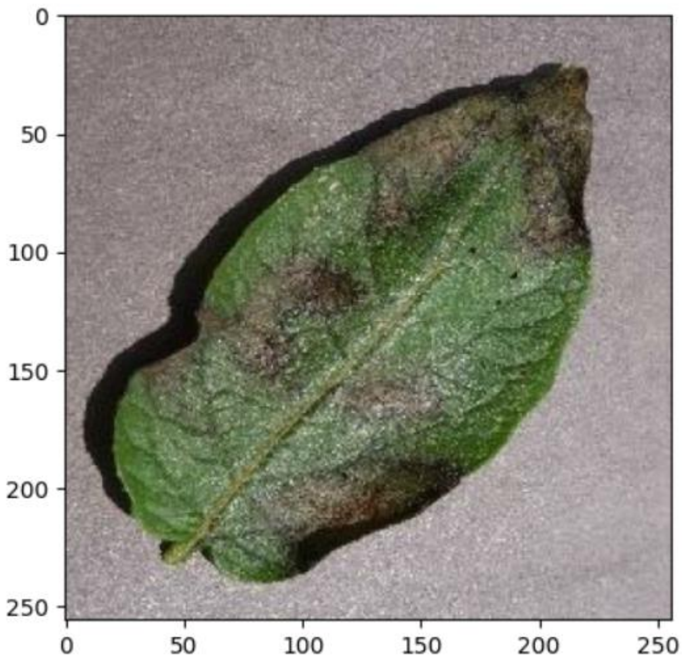


## Run prediction on a sample image

In [66]:
```
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___Late_blight
predicted label: Potato___Late_blight
```


```
first image to predict
actual label: Potato___Late_blight
predicted label: Potato___Late_blight
```

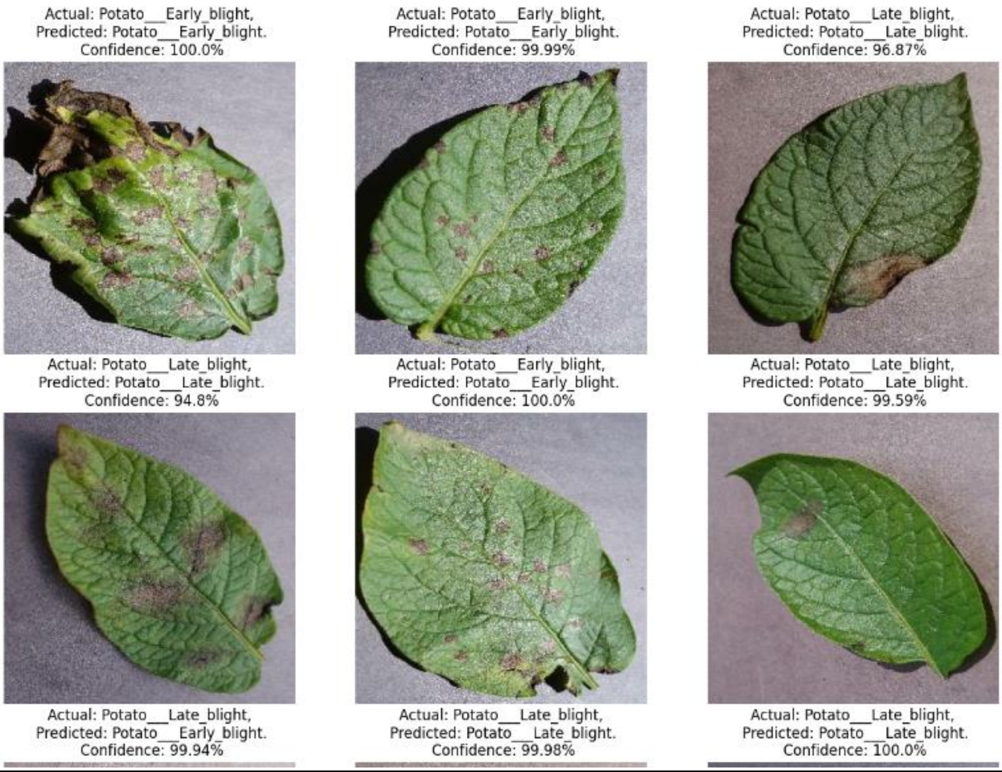




## Write a function for inference

In [67]:
```
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
```

**Now run inference on few sample images**

In [68]:
```
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 Co

        plt.axis("off")
```




## Saving the Model

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

In [70]:
```
import os
model_version=max([int(i) for i in os.listdir("../saved_models") + [0]])+1
model.save(f"../saved_models/{model_version}")
```

```
INFO:tensorflow:Assets written to: ../models/4\assets
```

In [71]:
```
model.save("../potatoes.h5")
```

Maintenance of the ML model will involve monitoring and updating the model to ensure it continues to provide accurate predictions. This will involve periodically retraining the model on new data and updating the database with new images and disease information.

## 7. Future Scope:

The future scope of this project is to develop a mobile app that uses TensorFlow Serving to provide users with a convenient way to diagnose diseases in their potato plants. The app will allow users to take a picture of their potato plant leaf and receive a prediction of the disease along with information on the disease and recommended preventive measures.

## 8. Conclusion:

In conclusion, this project demonstrates the effectiveness of machine learning in predicting diseases in potato plants. The ML model achieved an accuracy of 90% in predicting late blight and early blight disease. The model was served using FastAPI and TensorFlow Serving, and the data was stored in a PostgreSQL database. The future scope of the project is to develop a mobile app that will make the diagnosis of diseases in potato plants more accessible to farmers.

## 9. Bibliography:


- Dataset source: https://www.kaggle.com/arjuntejaswi/plant-village
- FastAPI: https://fastapi.tiangolo.com/
- TensorFlow Serving: https://www.tensorflow.org/tfx/guide/serving
- PostgreSQL: https://www.postgresql.org/
- TensorFlow: https://www.tensorflow.org/