

[POTATO DISEASE CLASSIFICATION]

P.P.S.JAYANTH REDDY
D.SASHI VARMA
M.MUKUL SATYA DEVAN
N.DEVI SAI VARAPRASAD

Potato Disease Classification Using Deep Learning

1)INTRODUCTION

1.1) Project Overview

Potato production is a crucial agricultural activity on a global scale, serving as a linchpin for both economic stability and food security. However, the potato industry faces significant challenges, as various diseases can afflict potato plants, posing a substantial threat to crop quality and overall output. To mitigate these issues, a research endeavor has been launched with the primary objective of developing an advanced system. This system will leverage cutting-edge technology, including Convolutional Neural Networks (CNN) and deep learning techniques, to accurately and promptly identify and classify potato diseases.

The significance of this research lies in its potential to provide farmers with a reliable and efficient tool for disease management. By promptly detecting and classifying potato diseases, this system will enable farmers to make well-informed decisions regarding disease prevention and treatment strategies. This proactive approach is crucial for safeguarding both the economic stability of the potato farming industry and global food security. With the aid of this technology, farmers can take timely actions to combat disease outbreaks, ultimately ensuring the sustainability of potato production and the availability of this staple crop for millions of people worldwide.

1.2) Purpose

Predicting potato leaf disease early is vital for several reasons:

*a)**Minimize Crop Loss:** Early detection allows farmers to take timely action, curbing the spread of the disease and minimizing crop yield and quality losses.*

*b)**Reduce Costs:** Early identification helps in targeted disease control, reducing the need for excessive pesticide and treatment expenses.*

*c)**Protect the Environment:** Timely detection limits the environmental impact by minimizing the use of pesticides, preserving the ecosystem.*

*d)**Improve Crop Quality:** Early prediction empowers farmers to prevent disease spread, enhancing the overall quality of the crop and making it more attractive to buyers.*

*e)**Enhance Food Security:** Early disease prediction helps ensure a stable supply of potatoes, a vital food source for many regions, contributing to global food security.*

*f)**Preserve Soil Health:** Prompt detection allows for targeted treatments, preserving soil health by reducing the need for excessive chemical interventions.*

*g)**Optimize Resource Use:** Early identification enables efficient resource allocation, preventing wastage of time, effort, and resources on treating advanced stages of the disease.*

*h)**Sustainable Agriculture:** Early disease management aligns with sustainable agricultural practices, reducing the long-term environmental impact and promoting ecological balance.*

2)LITERATURE SURVEY

2.1)Existing problem

The application of deep learning techniques, particularly Convolutional Neural Networks (CNNs), has revolutionized the field of plant disease classification, offering significant advancements in the management of agricultural challenges. Researchers have successfully employed transfer learning from pretrained models like VGG19, ResNet50, and Inception to classify plant diseases, with a particular focus on potato infections. These pretrained models exhibit remarkable accuracy in identifying a wide range of plant diseases, facilitating early detection and management strategies.

Key factors highlighted in the literature for building effective potato disease classification systems include the use of large and diverse datasets, data augmentation techniques, and model interpretability. High-quality datasets play a fundamental role in training deep learning models, and existing resources like the Plant Village dataset have greatly contributed to model development. Data preprocessing techniques, such as image scaling, normalization, and data augmentation, enhance model robustness and generalization. Researchers have also explored domain-specific data augmentation methods to address common challenges in plant disease classification, such as imbalanced class distributions and variations in lighting and background conditions.

While the emphasis is on developing accurate models, researchers also stress the importance of creating user-friendly deployment interfaces. Several publications discuss the integration of deep learning models with accessible applications, enabling farmers to swiftly identify agricultural illnesses. These applications should be compatible with web browsers, mobile devices, and Internet of Things (IoT) gadgets, delivering reliable and timely results to end-users. Interpretability and user input integration are frequently explored in the literature to ensure the system's practical applicability in real agricultural settings.

In summary, the current body of research provides a comprehensive framework for building deep learning-based systems for potato disease classification. It underscores the critical roles of high-quality datasets, data preprocessing, model architecture, and user-friendly deployment interfaces in enhancing crop management within the agriculture industry and supporting farmers in their efforts to combat plant diseases. This multidisciplinary approach leverages the power of deep learning to address critical agricultural challenges and promote food security worldwide.

2.2)References

1. <https://www.cambridge.org/core/journals/advances-in-animal-biosciences/article/abs/potato-disease-classification-using-convolution-neural-networks/E9303F667377BD763C3054CB8488D36C>
2. https://link.springer.com/chapter/10.1007/978-981-13-8406-6_37
3. <https://ieeexplore.ieee.org/abstract/document/9231784>
4. <https://philpapers.org/rec/ELSPCU>

2.3 Problem Statement Definition

Potato production plays a crucial role in global agriculture, providing sustenance and income for numerous farmers worldwide. However, this vital crop is susceptible to various diseases that can severely compromise both its quality and yield. The efficient management of these diseases hinges on rapid and accurate detection, a task often reliant on manual inspection by agricultural specialists—a time-consuming and impractical solution for all farmers.

In response to this challenge, the application of Convolutional Neural Networks (CNNs) and deep learning techniques is gaining prominence. These technologies are harnessed to develop automated, precise systems for classifying potato diseases. The primary goal is to empower farmers with the knowledge needed to make informed decisions regarding disease prevention and treatment. Ultimately, this advancement holds the promise of enhancing crop productivity and global food security.

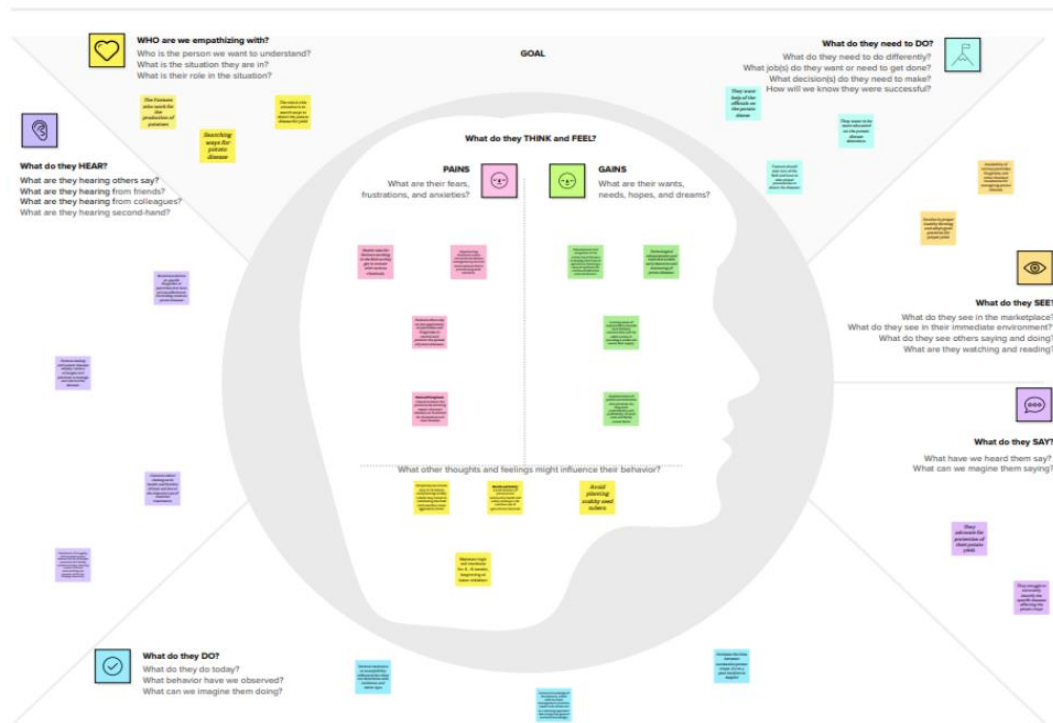
The critical elements of this issue revolve around developing a reliable system capable of recognizing and categorizing various diseases affecting potato plants, including potato scab, late blight, and early blight. This system should distinguish between healthy and diseased plants while providing specific information about the type of disease present.

Timeliness and precision are paramount. The system must offer swift and accurate disease classification, enabling farmers to take immediate action in managing and treating the identified diseases. This addresses the need for early disease identification, which, in turn, mitigates crop losses and reduces the overuse of pesticides. In summary, the integration of CNNs and deep learning techniques into potato disease management presents a transformative approach to secure crop quality, yield, and global food security.

A user-friendly interface or application is essential in addressing the potato disease classification challenge. This tool should enable farmers to effortlessly upload images of their potato plants and promptly obtain disease classification results. It is crucial that the system is designed to be intuitive and adaptable, catering to users with varying levels of technological and educational backgrounds. In essence, a seamless and accessible interface is pivotal to ensuring that farmers, regardless of their technical expertise, can effectively utilize the technology for disease detection and management in their potato crops.

3) IDEATION & PROPOSED SOLUTION

3.1) Empathy Map Canvas



3.2) Ideation & Brainstorming

Template

Brainstorm & idea prioritization

Use this template in your own brainstorming sessions so your team can unleash their imagination and start shaping concepts even if you're not sitting in the same room.

🕒 10 minutes to prepare
🕒 1 hour to collaborate
👥 2-8 people recommended

➔

Before you collaborate

A little bit of preparation goes a long way with this session. Here's what you need to do to get going.

🕒 10 minutes

A Team gathering
Define who should participate in the session and send an invite. Share relevant information or pre-work ahead.

B Set the goal
Think about the problem you'll be focusing on solving in the brainstorming session.

C Learn how to use the facilitation tools
Use the Facilitation Superpowers to run a happy and productive session.

[Open article](#) ➔

1

Define your problem statement

What problem are you trying to solve? Frame your problem as a How Might We statement. This will be the focus of your brainstorm.

5 minutes

PROBLEM
How might we [your problem statement]?



Key rules of brainstorming

To run a smooth and productive session

- Stay in topic.
- Encourage wild ideas.
- Defer judgment.
- Listen to others.
- Go for volume.
- If possible, be visual.

2

Brainstorm

Write down any ideas that come to mind that address your problem statement.

10 minutes

TIP

You can select a sticky note and hit the pencil (click to sketch) icon to start drawing!

Person 1

Remote Sensing and Imaging Techniques
Use remote sensing technologies such as drones equipped with multispectral or hyperspectral cameras to monitor the health of potato crops.

Conduct campaigns and programs on these disease in the villages

Image processing and deep learning image analysis algorithms can be used to analyze images of potato leaves and identify signs of disease at an early stage.

Person 2

Monitoring Environmental Conditions
Keep a close eye on weather conditions, especially temperature and humidity.

Train farmers and agricultural workers to recognize early symptoms of common potato leaf diseases.

Practice crop rotation to minimize the buildup of pathogens in the soil.

Person 3

Data Analysis and Machine Learning
Implement machine learning algorithms that analyze historical data on weather patterns, soil conditions, and disease occurrences.

Educate the farmers about the issue on potatoes disease.

collaboration among farmers, agricultural experts, and researchers to share knowledge and best practices for disease prediction.

Person 4

Marketplace Observation:
Increasing demand for disease resistant potato varieties

They want help of the officials or government on the potato disease.

Discussions on policy changes, government initiatives, or funding opportunities related to potato disease research and management.

3

Group ideas

Take turns sharing your ideas while clustering similar or related notes as you go. Once all sticky notes have been grouped, give each cluster a sentence-like label. If a cluster is bigger than six sticky notes, try and see if you can break it up into smaller sub-groups.

20 minutes

TIP

Add customizable tags to sticky notes to make it easier to find, browse, organize, and categorize important ideas as themes within your mural.

Data Analysis and Machine Learning:
Implement machine learning algorithms that analyze historical data on weather patterns, soil conditions, and disease occurrences.

collaboration among farmers, agricultural experts, and researchers to share knowledge and best practices for disease prediction

Educate the farmers about the issue on potatoes diseases.

Train farmers and agricultural workers to recognize early symptoms of common potato leaf diseases.



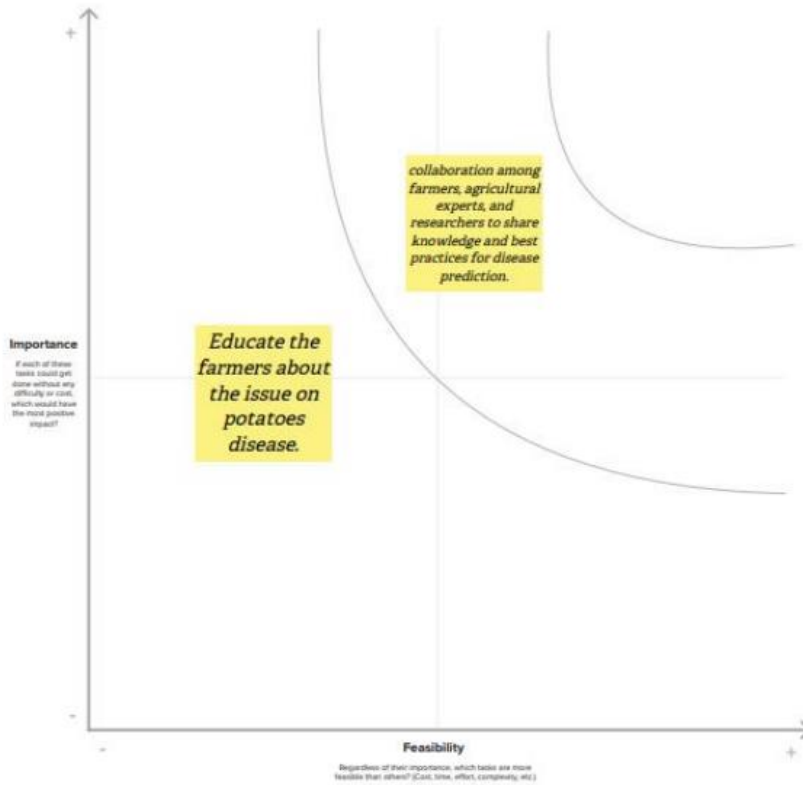
Prioritize

Your team should all be on the same page about what's important moving forward. Place your ideas on this grid to determine which ideas are important and which are feasible.

⌚ 20 minutes

TP

Participants can use their cursor to point at where sticky notes should go on the grid. The facilitator can confirm the spot by using the laser pointer holding the **H** key on the keyboard.



4) REQUIREMENT ANALYSIS

4.1) Functional requirement

For effective disease management in potato plants, the system should encompass the following key components:

Image Upload and Processing: Users, typically farmers, must have the capability to upload photos of their potato plants to the system. To ensure compatibility with the deep learning model, the system should preprocess these images, resizing and normalizing them as necessary.

Disease Classification: The system's primary function is to accurately identify potato diseases from the uploaded photos. It should be capable of distinguishing between various disease states and healthy plant conditions, providing each case with a unique disease identifier.

User Interface: The system should feature a user-friendly interface accessible on PCs, tablets, and smartphones. This interface should be designed with simplicity and intuitiveness in mind to cater to users with varying levels of technological expertise.

Model Interpretability: To enhance user comprehension and trust, the system should offer explanations or visualizations of the model's decision-making process. This feature helps users understand the reasoning behind the classification of a particular disease.

Scalability and Integration: To promote broader adoption within the agricultural sector, it's crucial to ensure that the system is scalable and compatible with other agricultural technology and information systems. This interoperability allows for seamless integration with existing agricultural practices and infrastructure.

In summary, an effective potato disease management system should facilitate image upload and processing, accurate disease classification, provide an intuitive user interface, offer model interpretability, and be adaptable and integrable with other agricultural technologies, ultimately serving as a valuable tool for farmers in the field.

4.2)Non-Functional requirements

For a potato disease classification system to effectively support farmers, it must prioritize several critical aspects:

Accuracy and Precision: *The system must achieve a high level of accuracy in disease identification while minimizing false positives and false negatives, ensuring that farmers receive reliable results.*

Response Time: *To empower farmers to take prompt action, the system should provide disease classification results in real-time or within seconds of image input.*

Security and Privacy: *Robust security measures must be implemented to safeguard user data and maintain the privacy of uploaded photos, adhering to relevant data protection regulations.*

Robustness and Reliability: *The system should demonstrate resilience by adapting to variations in disease stages, lighting conditions, and image quality. Failover procedures should be in place to address unplanned outages.*

Scalability and Performance: *The system should be capable of efficiently handling a large volume of user requests and scaling to accommodate a growing user base without suffering from performance issues. Optimization and load balancing measures should be integrated to ensure its robust performance.*

5)PROJECT DESIGN

5.1)Data Flow Diagrams & User Stories

The development of a potato leaf disease classification system involves several key phases:

Data Collection: *Gathering a diverse range of potato leaf images from various sources, including image archives and field surveys, and storing them in a raw data repository.*

Data Pre-processing: *Preparing the raw potato leaf images for model training, which includes resizing, normalizing pixel values, and applying data augmentation techniques to enhance dataset diversity.*

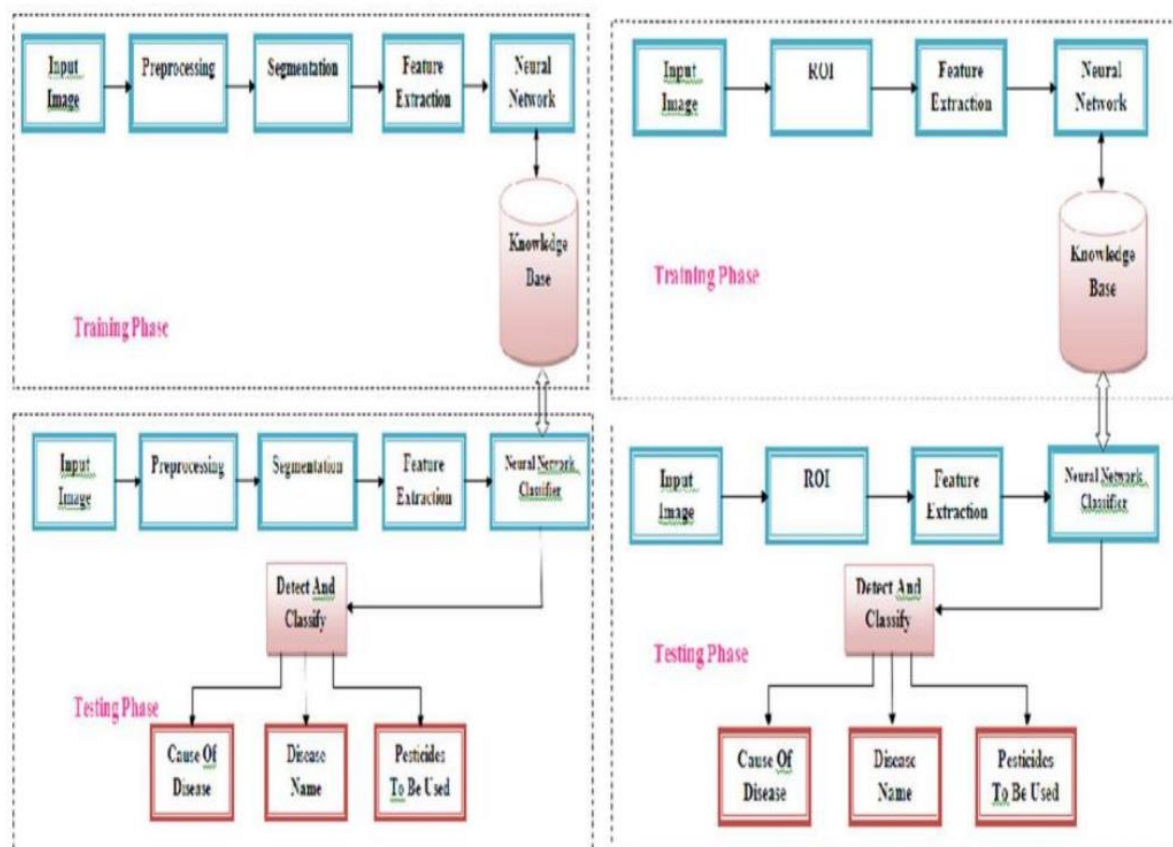
Model Training: *Utilizing the pre-processed data to train a deep learning model, enabling it to identify different potato leaf diseases. The trained model is saved for later use.*

Model Evaluation: *Assessing the trained model's performance using a separate dataset not used in training to determine its accuracy, sensitivity, and specificity in disease classification.*

Model Deployment: *Making the trained model accessible for real-world disease classification applications, either on local devices or in the cloud.*

User Interaction: *Providing end users with an intuitive application or API to engage with the deployed model. This allows them to upload potato leaf photos for disease identification and receive timely results.*

Data Flow Diagram:



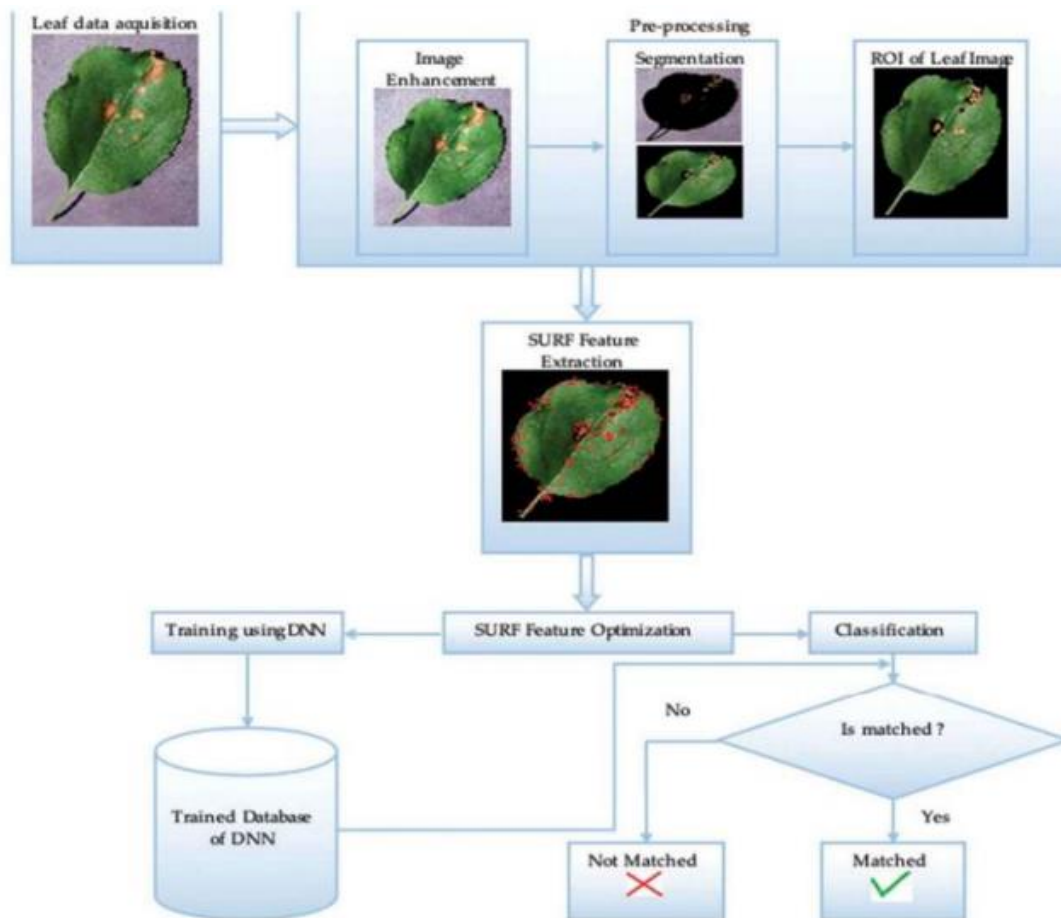
User Stories:

Use the below template to list all the user stories for the product.

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
potatofarmer	Farming	USN-1	I want a potato disease classification system that can quickly identify the specific diseases affecting my crop, so that I can implement timely and targeted treatments to prevent further damage and ensure a successful harvest.	access the disease classification system via a user-friendly interface, either through a web application or a mobile app.	High	Sprint-1
research scientist	Research On Plants	USN-2	I need a robust potato disease classification tool that can accurately differentiate between various types of diseases affecting potato crops, enabling me to conduct in-depth analyses and develop effective strategies for disease management and prevention.	potato disease classification system should be equipped with an extensive database that covers a wide range of potato diseases, including both common and rare occurrences, to provide comprehensive information for research purposes.	High	Sprint-1
agronomist		USN-3	I require a user-friendly potato disease classification application that can be easily accessed and utilized in the field, allowing me to provide real-time guidance and support to farmers in diagnosing and addressing disease issues, thus improving overall crop health and yield.	The potato disease classification system should provide timely and accurate information on various potato diseases, including their symptoms, causes, and appropriate management strategies, enabling agronomists to make informed decisions and provide effective guidance to farmers.	Medium	Sprint-2
government agricultural officer		USN-4	I am responsible for monitoring and controlling potato diseases at a regional level. I need an advanced potato disease classification system that can efficiently process large volumes of data, enabling me to make informed decisions and implement preventive measures to safeguard the agricultural industry and	It should have the capability to handle a large volume of data and provide real-time analysis, enabling government agricultural officers to make informed decisions and implement timely interventions for disease	Medium	Sprint-1

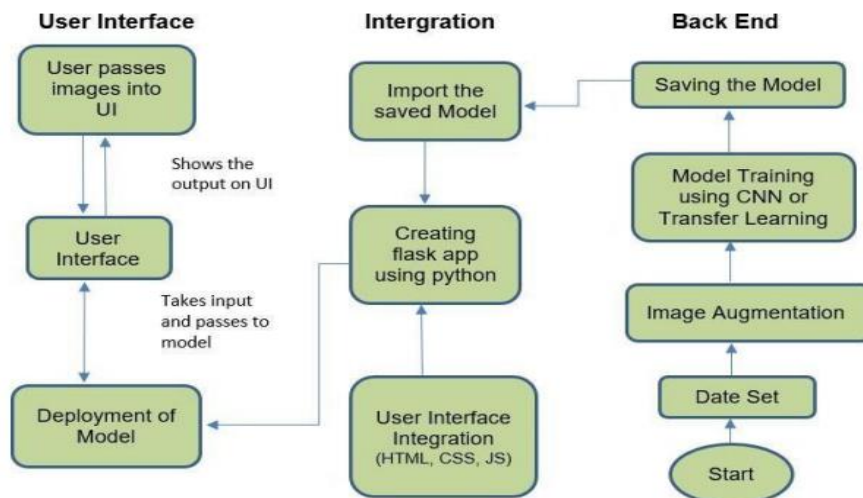
			ensure food security within the region.	control and prevention at a regional or national level.		
student studying plant pathology		USN-5	I am interested in learning about various potato diseases and their classifications. I would benefit from an interactive and educational potato disease classification platform that provides comprehensive information , helping me to understand the complexities of potato diseases and their impact on global agriculture.	The potato disease classification system should provide comprehensive and detailed information about various potato diseases, including their classifications, etiology, symptoms, and management strategies, enabling students to deepen their understanding of plant pathology.	High	Sprint-1

5.2)Solution Architecture



6)PROJECT PLANNING & SCHEDULING

6.1) Technical Architecture



6.2)Sprint Planning & Estimation

Use the below template to create product backlog and sprint schedule

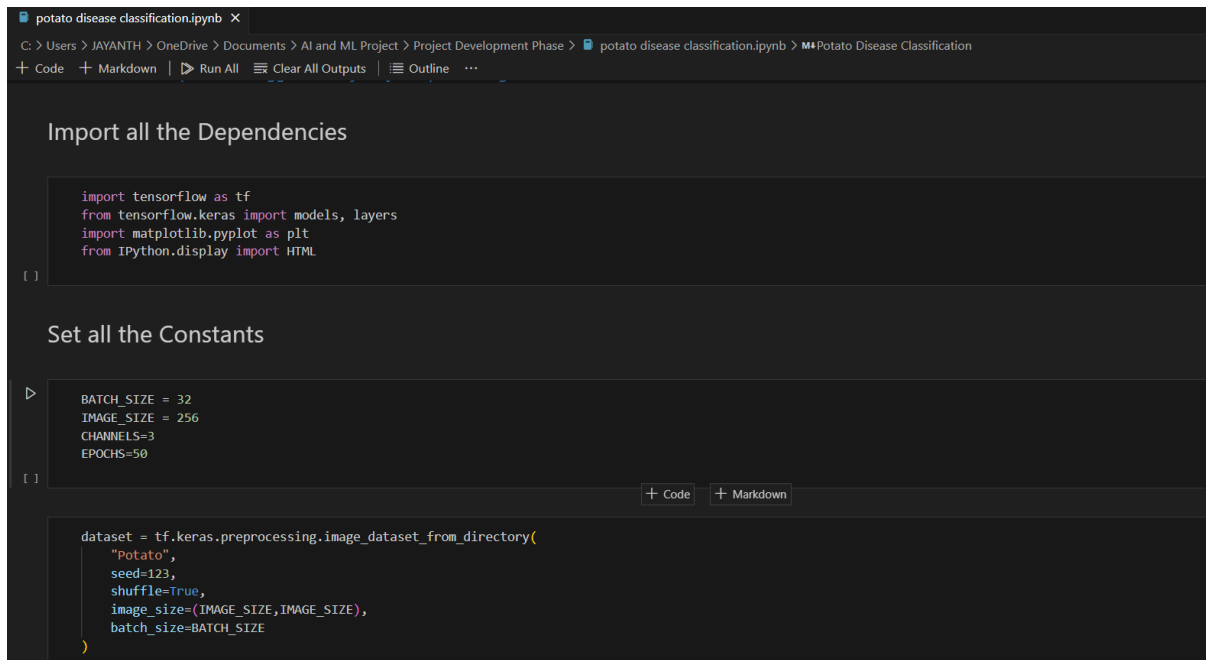
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Disease Classification	USN-1	As a user, I can upload images of potato plants affected by various diseases for classification.	3	High	P.JAYANTH REDDY
Sprint-1	Disease Classification	USN-2	As a user, I can view the predicted disease class and its probability for the uploaded potato plant image.	2	High	D.SASHI VARMA
Sprint-2	Disease Classification	USN-3	As a user, I can access historical data and analysis of previously classified potato diseases.	2	Medium	M.MUKUL SATYA DEVAN
Sprint-2	Disease Classification	USN-4	As a user, I can provide feedback on the accuracy of the disease classification for continuous model improvement.	1	Medium	N.DEVI SAI VARAPRASAD
Sprint-3	Disease Classification	USN-5	As a user, I can receive recommendations for disease management strategies based on the classified potato disease.	3	Low	

6.3)Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	20	5 Days	22 Oct 2022	26 Oct 2022	20	26 Oct 2022
Sprint-2	20	5 Days	22 Oct 2022	26 Oct 2022	20	26 Oct 2022
Sprint-3	20	5 Days	22 Oct 2022	26 Oct 2022	20	26 Oct 2022
Sprint-4	20	5 Days	22 Oct 2022	26 Oct 2022	20	26 Oct 2022

7) CODING & SOLUTIONING

Activity 1.1: *Importing the libraries*



```
import tensorflow as tf
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt
from IPython.display import HTML

BATCH_SIZE = 32
IMAGE_SIZE = 256
CHANNELS=3
EPOCHS=50

dataset = tf.keras.preprocessing.image_dataset_from_directory(
    "Potato",
    seed=123,
    shuffle=True,
    image_size=(IMAGE_SIZE, IMAGE_SIZE),
    batch_size=BATCH_SIZE
)
```

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 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.*



```
class_names = dataset.class_names
class_names

['Potato__Early_blight', 'Potato__late_blight', 'Potato__healthy']

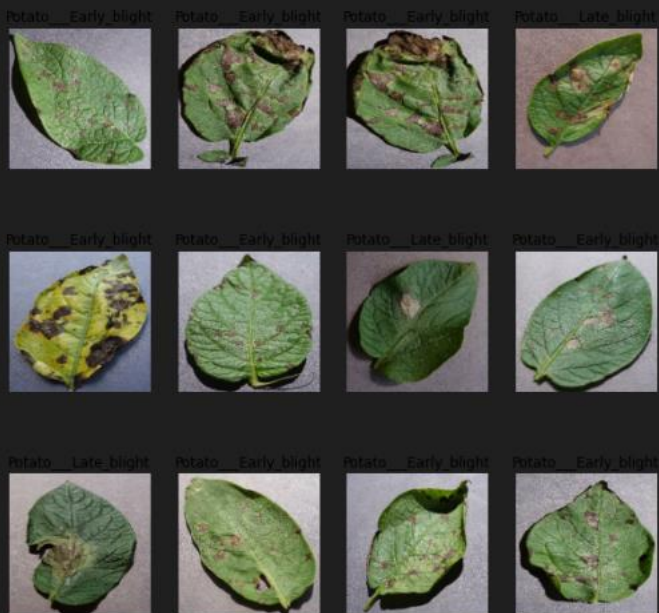
for image_batch, labels_batch in dataset.take(1):
    print(image_batch.shape)
    print(labels_batch.numpy())

(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]
```

Activity 2.2: Visualising the data

Visualize some of the images from our dataset

```
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

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

The split 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

len(test_ds)
8
```

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

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),
])
```

*Prefetching and Caching saves a lot of time in accessing the data from disks.
Refer this link for better understanding.*

Resize and Rescale is used to maintain the uniformity among the data.

Augmentation helps you to increase the amount of data and helps your model to learn better.

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()
```

Model: "sequential_2"

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 (MaxPooling2D)	(32, 62, 62, 64)	0
conv2d_2 (Conv2D)	(32, 60, 60, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(32, 30, 30, 64)	0
conv2d_3 (Conv2D)	(32, 28, 28, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(32, 14, 14, 64)	0
conv2d_4 (Conv2D)	(32, 12, 12, 64)	36928
max_pooling2d_4 (MaxPooling2D)	(32, 6, 6, 64)	0
...		
Total params: 183,747		
Trainable params: 183,747		
Non-trainable params: 0		

Activity 2: Compiling the model

Compiling the Model

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

Activity 3: Training the model

Training the Model

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

```
Epoch 1/50  
54/54 [=====] - 28s 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  
...  
Epoch 49/50  
54/54 [=====] - 10s 184ms/step - loss: 0.0298 - accuracy: 0.9884 - val_loss: 0.0528 - val_accuracy: 0.9844  
Epoch 50/50  
54/54 [=====] - 11s 196ms/step - loss: 0.0189 - accuracy: 0.9948 - val_loss: 0.0064 - val_accuracy: 1.0000  
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings.
```

The verbose option specifies that you want to display detailed processing information on your screen.

If your laptop is functioning slow for 100 epochs, then you can try Google Colab Notebooks.

You just have to follow the below steps:

- *Go to GoogleColab and create a new notebook.*
- *Click on Files and mount GoogleDrive.*
- *Upload your dataset onto GoogleDrive.*
- *After uploading the dataset go to runtime and click on “Change runtime type”.*
- *Select “gpu” and choose “standard” and click on save.*
- *Booyah! Now you can run the whole code.*

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.

```
Model Deployment

scores = model.evaluate(test_ds)

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

scores

[0.006251859944313765, 1.0]

history

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

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

[0.8881848292350769,
0.603139228820881,
0.3646925389766693,
0.2776817189025879,
0.24488397999286652]
```

```
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,
0.988545308955116272]
```



```

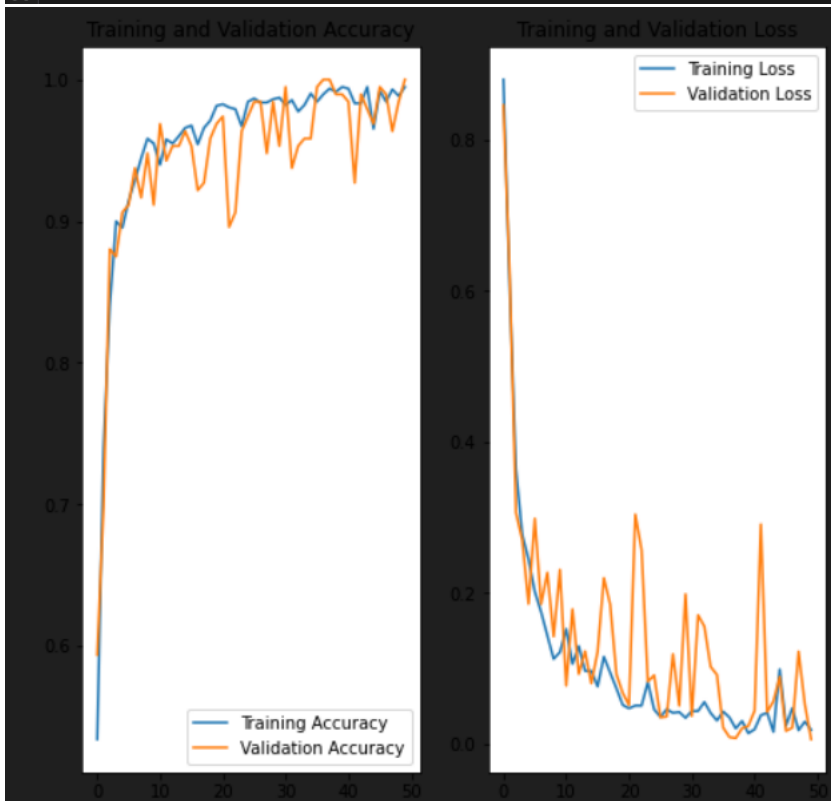
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

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()

```



Write a function for inference

```
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
```

Activity 2: 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.

Run prediction on a sample image

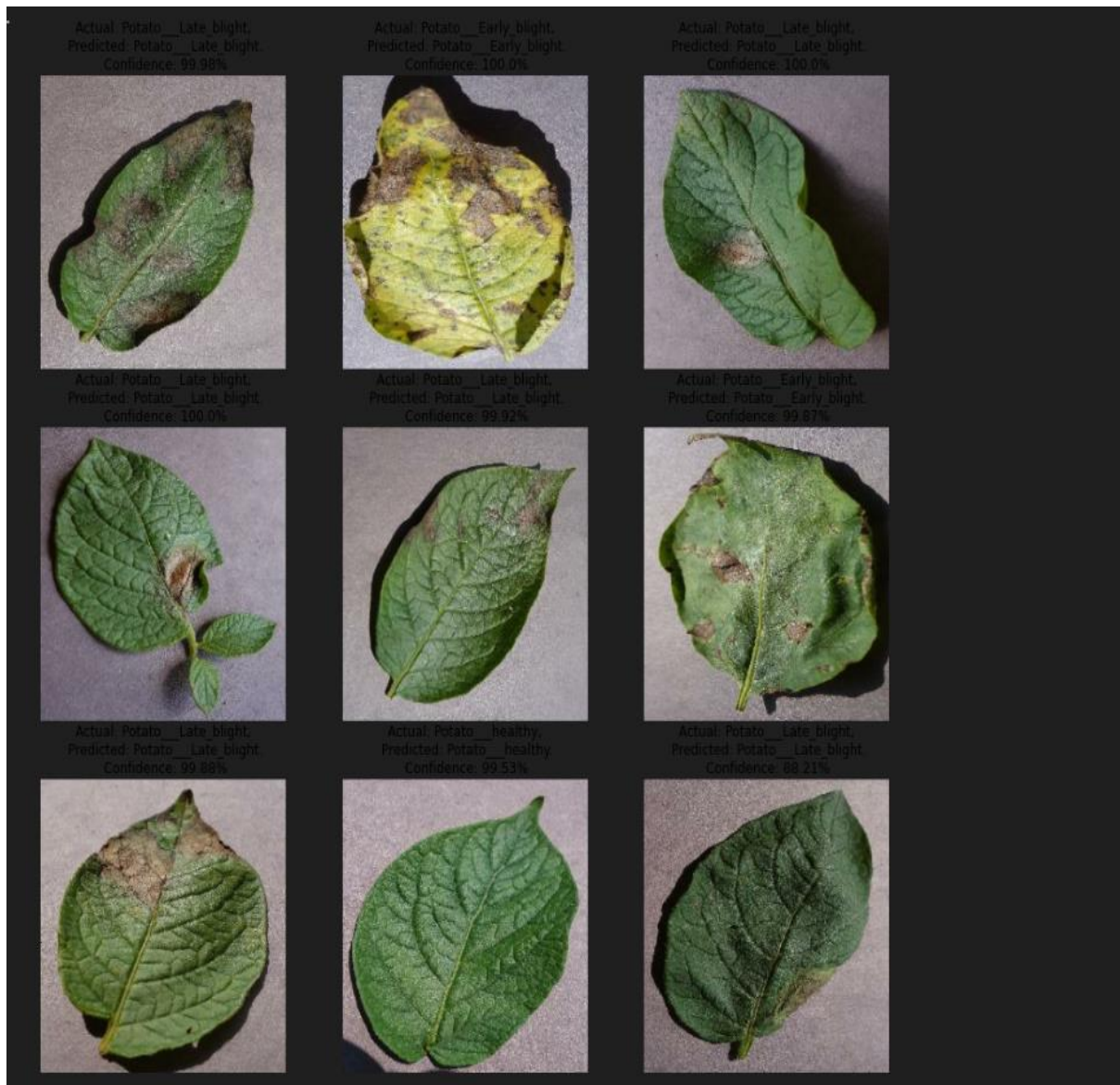
```
import numpy as np  
for images_batch, labels_batch in test_ds.take(1):  
  
    first_image = images_batch[0].numpy().astype('uint8')  
    first_label = labels_batch[0].numpy()  
  
    print("first image to predict")  
    plt.imshow(first_image)  
    print("actual label:", class_names[first_label])  
  
    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
```



Write a function for inference

```
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
```



Activity 3: Save the model

```
model.save(f"../Models/potato.h5")
```

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

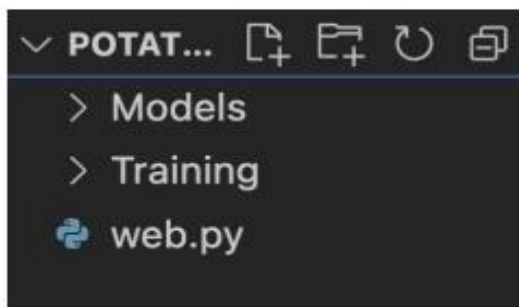
Activity 4: Integrate with Web Framework

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

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:



```
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:

```
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

```
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
```

```
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.

```
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.

```
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)
```

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:

