Project Report Format

1. INTRODUCTION

1.1 Project Overview

Worldwide, growing potatoes is a crucial part of farming that helps with both making money and ensuring there's enough food. However, potato plants can get sick, and that's not great for the quality and amount of potatoes produced. It's important to quickly figure out and categorize these diseases to manage them effectively. The aim of this study is to develop a system using Convolutional Neural Networks (CNN) and deep learning methods to accurately identify and classify potato diseases. This way, farmers can make smart choices to prevent and treat these illnesses.

1.2 Purpose

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.

2. LITERATURE SURVEY

2.1 Existing problem

Researchers have effectively tackled the challenges of classifying plant diseases by using advanced deep learning methods, especially Convolutional Neural Networks (CNNs). They've found success in applying transfer learning from pretrained models like VGG19, ResNet50, and Inception, particularly in dealing with potato infections. These models, through their training, can accurately recognize a wide variety of plant diseases, aiding in early detection and control. The literature emphasizes the importance of large and diverse datasets, techniques like data augmentation, and the ability to interpret the models when developing a system to classify potato diseases.

Creating effective deep learning models for classifying plant diseases relies heavily on having top-notch datasets, as demonstrated by past research. The development of models has greatly improved thanks to the accessibility of established datasets like the Plant Village dataset and similar ones. Researchers enhance the robustness and applicability of these models by employing data preprocessing methods such as image scaling, normalization, and data augmentation. To overcome common challenges in plant disease classification, like uneven class distribution and variations in lighting and background, some studies have delved into specialized data augmentation techniques tailored to this specific domain.

While the literature stresses the importance of developing accurate models, it also underscores the need to create user-friendly deployment interfaces. Several publications have discussed the

integration of deep learning models with accessible applications, aiming to empower farmers in swiftly identifying agricultural diseases. These applications should be compatible with web browsers, mobile devices, and Internet of Things gadgets, providing users with reliable and timely results. Ensuring the practical usability of the system in real agricultural settings often involves exploring interpretability and integrating user input, as highlighted in the literature of this domain.

In the summary, the body of current research offers a solid framework for creating a deep learning-based system for classifying potato diseases. It emphasizes the crucial roles that high-quality datasets, data preparation, model architecture, and user-friendly deployment have in improving crop management in the agriculture industry and helping farmers.

2.2 References

1.https://www.cambridge.org/core/journals/advances-in-animalbiosciences/article/abs/potato-disease-classification-using-convolution-neuralnetworks/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

5. https://ieeexplore.ieee.org/abstract/document/8905128

2.3 Problem Statement Definition

Potatoes play a crucial role in global agriculture, serving as both a fundamental food source and a primary income stream for many farmers. However, potato plants are susceptible to various diseases that can significantly impact crop quality and productivity. Effectively managing these diseases relies on promptly and accurately detecting them. Traditional methods often involve manual inspection by agricultural experts, which can be time-consuming and inaccessible to all farmers. To address this challenge, automated and precise systems, utilizing Convolutional Neural Networks (CNNs) and deep learning techniques, are being developed for classifying potato diseases. This technology aims to empower farmers with informed decisions regarding disease prevention and treatment, ultimately boosting crop productivity and enhancing food security.

Important elements of problem solving statements:

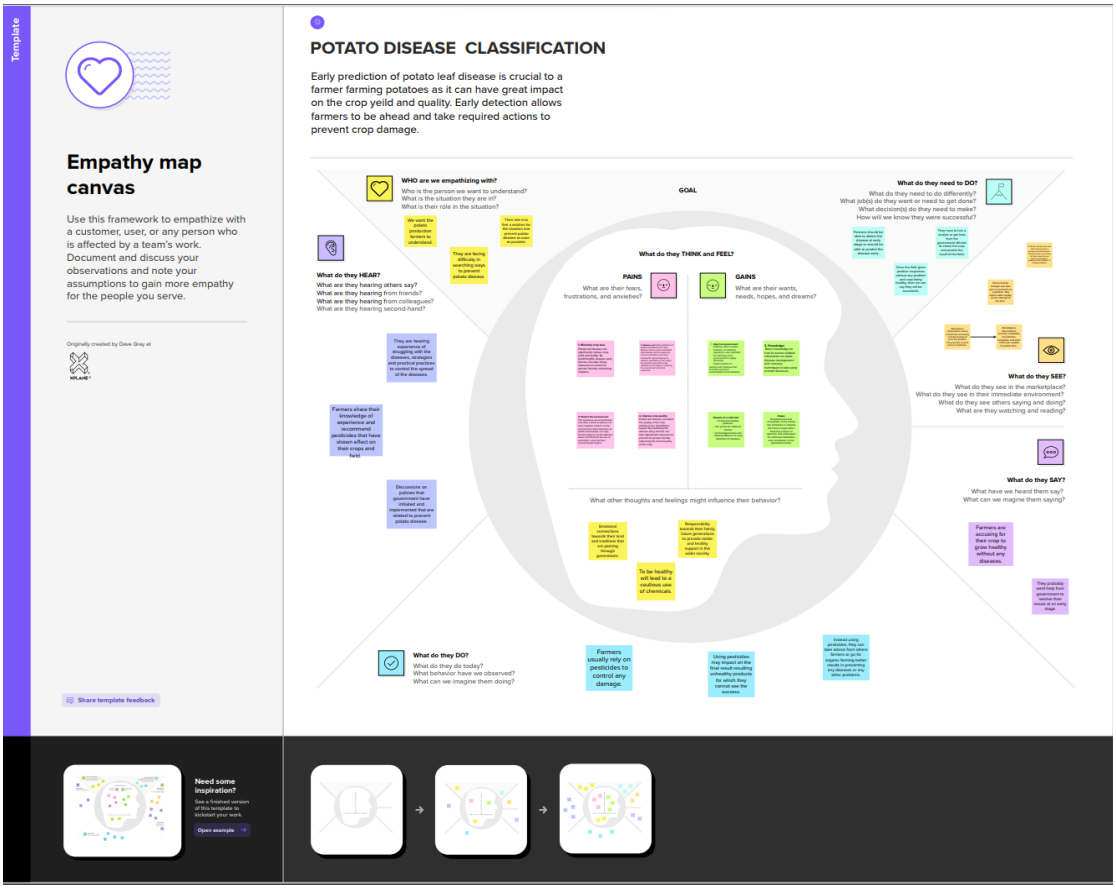
The main challenge is developing a trustworthy system for recognising and categorizing the several diseases that impact potato plants, including potato scab, late blight, and early blight. The technology should be able to distinguish between plants that are healthy and those that are ill, and it should give precise details on the kind of disease that is present.

Timeliness and Precision: The system should deliver accurate and quick disease classification data so that farmers can begin managing and treating the illnesses that have been recognised right away. This takes care of the requirement for early disease identification, which can cut down on crop losses and excessive pesticide use.

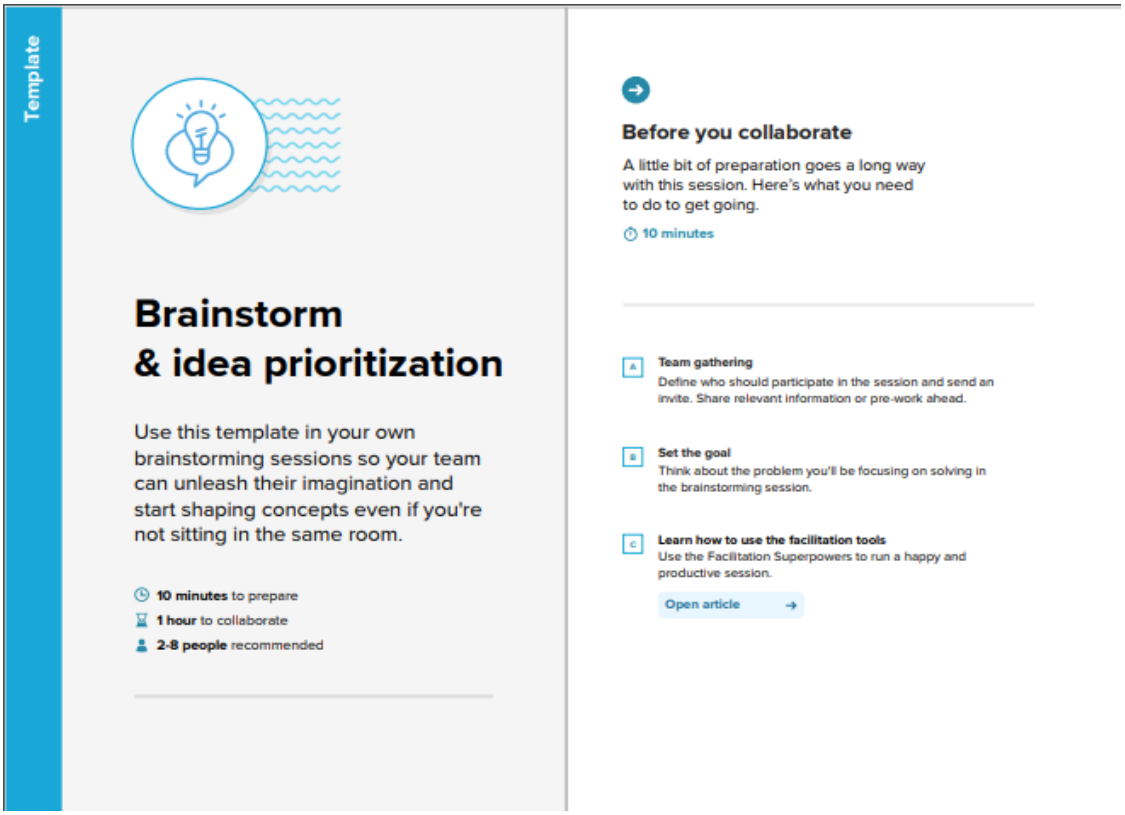
User-Friendly Interface: To solve this problem effectively, a user-friendly interface or application should be developed, allowing farmers to easily upload images of their potato plants and receive disease classification results. The system should be intuitive and adaptable to the technological and educational background of the users.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



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 predict potato leaf disease at an early stage?

Key rules of brainstorming

To run an smooth and productive session

- Stay in topic.
- Defer judgment.
- Go for volume.
- Encourage wild ideas.
- Listen to others.
- 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 [switch to sketch] icon to start drawing!

Person 1

Person 2

Person 3

Person 4

Person 5

Person 6

Person 7

Person 8

3

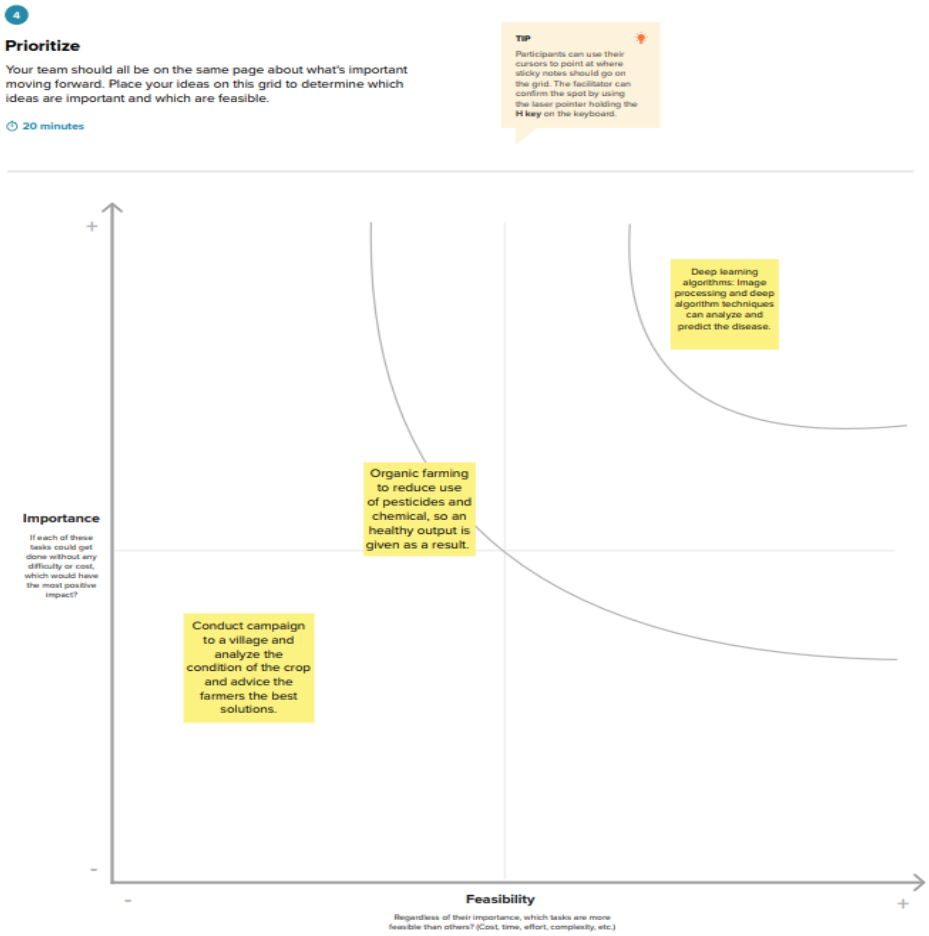
Group ideas

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 and 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.



4. **REQUIREMENT ANALYSIS**

4.1 Functional requirement

Image Upload and Processing: In order to classify diseases, users (farmers) should be able to upload photos of potato plants to the system. To guarantee that these photographs are compatible with the deep learning model, it should preprocess them, resizing and normalizing them as needed.

Disease Classification: The system's main job is to correctly identify potato illnesses from the supplied photos. It should be able to distinguish between distinct disease states and healthy plant states, giving each case a unique disease identifier.

User Interface: A user-friendly interface that is compatible with PCs, tablets, and smartphones must be provided by the system. It ought to offer a simple and intuitive user interface to suit consumers with varying degrees of technological expertise.

Model Interpretability: The system ought to provide justifications or illustrations of the model's decision-making procedure, giving users an understanding of the rationale behind the classification of a certain ailment. This feature improves user comprehension and trust.

Scalability and Integration: To enable wider acceptance and influence within the agricultural sector, make sure the system is scalable and compatible with other agricultural technology and information systems.

4.2 Non-Functional requirements

Accuracy and Precision: For farmers to receive trustworthy findings, the disease classification system needs to attain a high degree of accuracy. It should also minimise false positives and

false negatives by precisely identifying diseases.

Response Time: In order to enable farmers to take rapid action, the system should provide disease categorization results as soon as possible, ideally in real-time or within a few seconds of image input.

Security and Privacy: Put security measures in place to guard user information and guarantee the privacy of photos that users post. Images and personal data should be treated carefully and in accordance with applicable data protection laws.

Robustness and Reliability: Strong and dependable, the system should be able to adapt to changes in disease stages, lighting, and image quality. In the event of an unplanned outage, failover procedures ought to be included.

Scalability and Performance: Make sure the system is capable of managing a sizable volume of user requests and expanding to accommodate a rising user base without experiencing performance issues. Optimization and load balancing ought to be implemented.

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data collection: In this step, we collect various pictures of potato leaves from different places like fields, image libraries, or other sources. These pictures are then kept in a storage for raw data.

Data preprocessing: Raw potato leaf images are pre-processed to prepare them for model training. We might change their size, make sure the colors are set up in the right way, and use techniques to make the dataset more varied. This helps the model learn better.

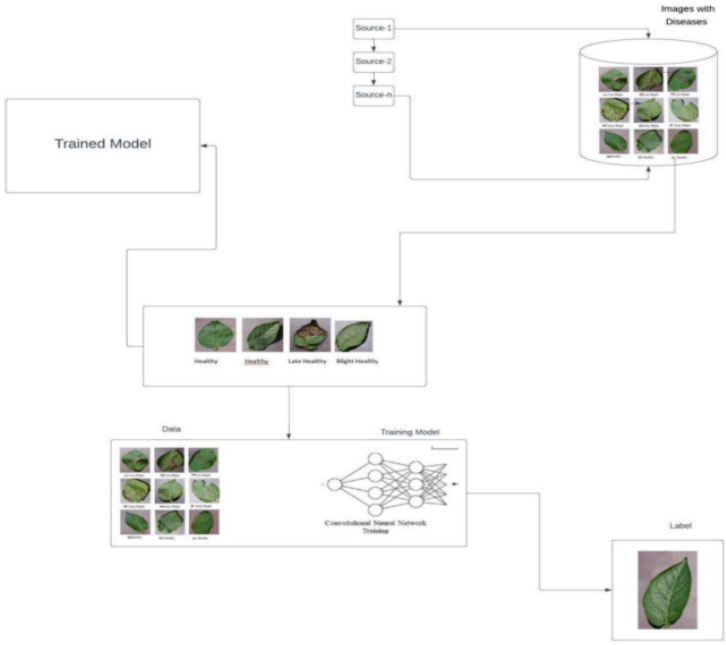
Model Training: The pre-processed data can now be used to train deep learning model, which learns to classify potato leaf disease. The trained model is saved for future use.

Model Evaluation: We check how well the trained model works by using a different set of pictures that it hasn't used before. We want to see how accurate it is at finding diseases and how good it is at not making mistakes.

Model Deployment: In this step, deployment of trained model is to be done, to make it accessible for real-world disease classification applications.

User interaction: End-users interact with the deployed model through a user-friendly application or API, enabling them to submit potato leaf images for disease identification and receive prompt results.

Data flow diagram:



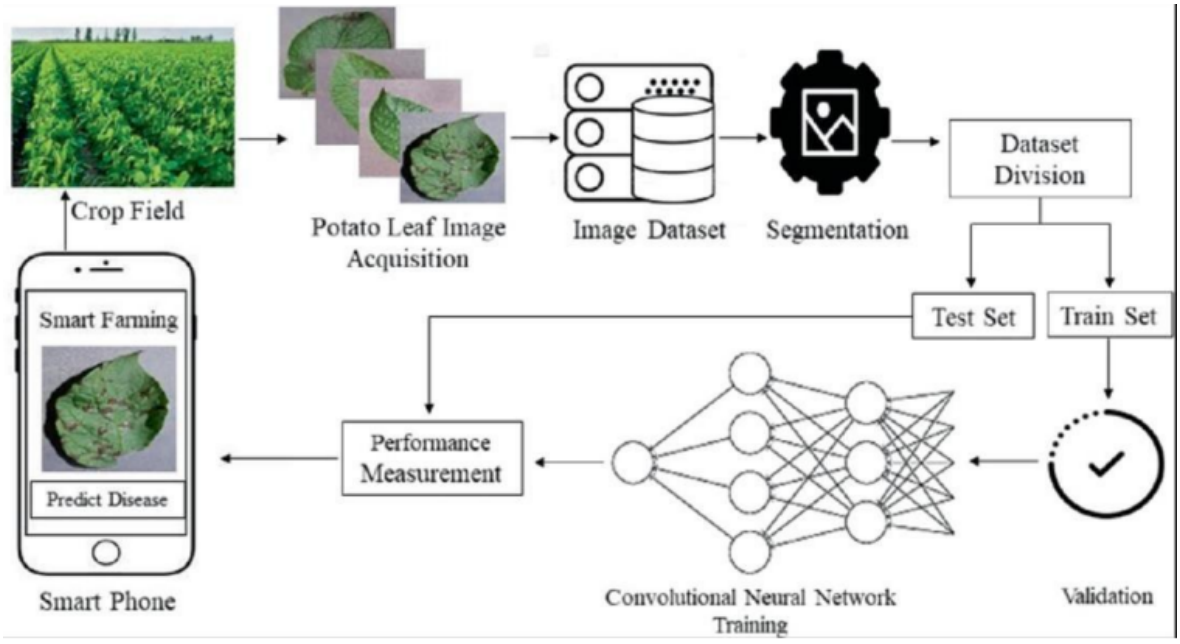
User Stories

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
Potato farmer	Registration	USN-1	I require a system for classifying potato diseases that can rapidly distinguish the particular ailments afflicting my crops. This will enable me to promptly apply precise treatments to thwart further harm and guarantee a productive harvest.	I can access my account / dashboard	High	Sprint-1
Research scientist	Research on plants	USN-2	Gain access to the disease classification system using an intuitive interface, accessible either through a web-based application or a mobile app.	The potato disease classification system needs to have a big database that includes many different potato diseases, even the ones that don't happen often. This way, it can offer lots of information for research purposes.	High	Sprint-1
agronomist		USN-3	I need a user-friendly potato disease classification app that's simple to use in the field. This app would enable me to offer immediate assistance to farmers in diagnosing and dealing with disease problems, ultimately enhancing crop	The potato disease classification system should offer prompt and precise details about different potato diseases, covering their	Medium	Sprint-2

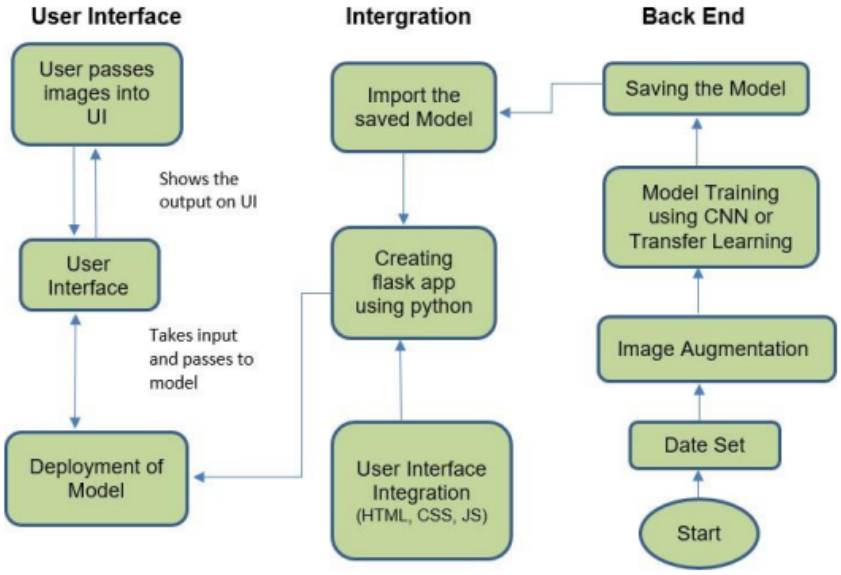
			health and yield.	symptoms, causes, and the right methods for managing them. This will empower agronomists to make well-informed choices and offer effective advice to farmers.		
Government agriculture officer		USN-4	I'm in charge of looking after potato diseases in a whole region. I need an advanced potato disease classification system for figuring out what diseases are there, especially for a big of data. This will help me make good decisions and take steps to stop these diseases from hurting our farms and making sure we have enough food in our region	It needs to be good at handling lots of data and quickly figuring things out. This way, government agricultural folks can use it to make smart choices and take quick actions to stop diseases from spreading in a big area, like a whole region or even the entire country.	Medium	Sprint-1
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	High	Sprint-1
				deepen their understanding of plant pathology.		

5.2 Solution Architecture



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

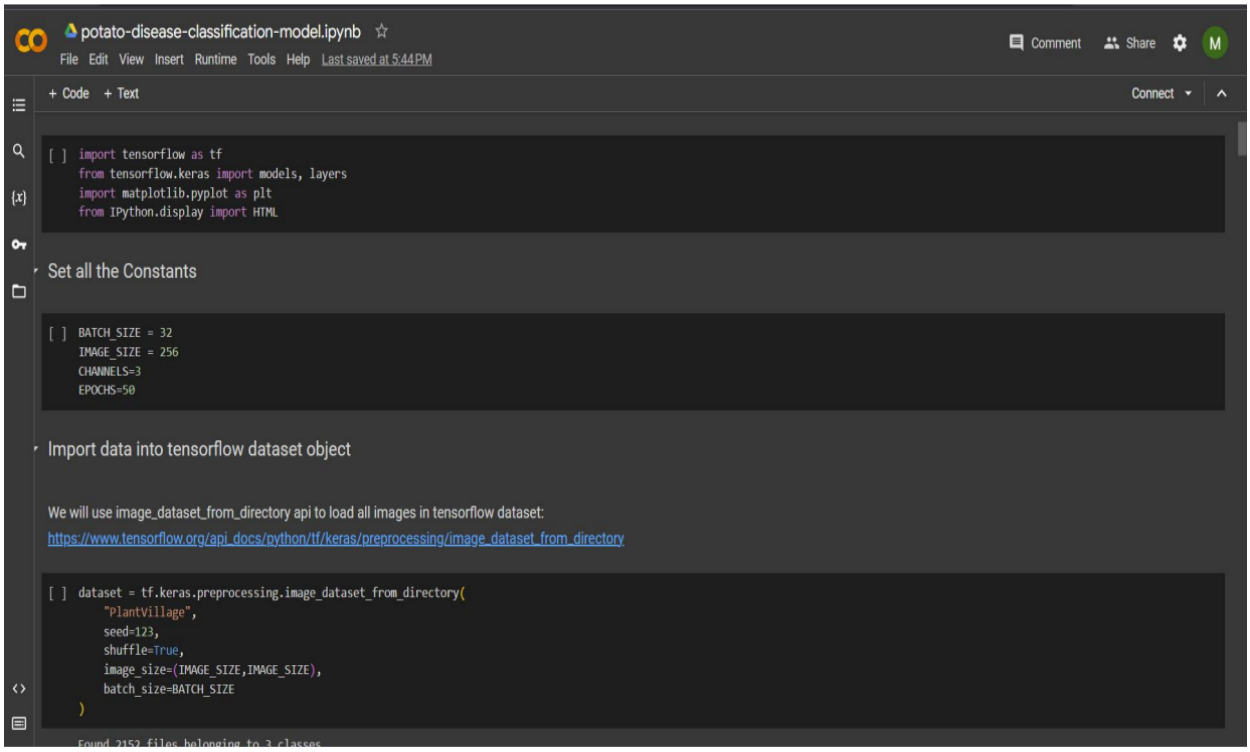
Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Disease Classification	USN-1	As a user, I can give images of affected potato leafs for classification of disease.	3	High	Pittam Harika
Sprint-1	Disease Classification	USN-2	As a user, I can get the information of the predicted disease class and its probability for that image.	2	High	Mummadi Nripesh Reddy
Sprint-2	Disease Classification	USN-3	As a user, I can have the access to view the history of data and analysis of previous disease classifications.	2	Medium	Mukkamalla Tarunika
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	
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	23 Oct 2023	27 Oct 2023	20	27 Oct 2023
Sprint-2	20	5 Days	23 Oct 2023	27 Oct 2023	20	27 Oct 2023
Sprint-3	20	5 Days	23 Oct 2023	27 Oct 2023	20	27 Oct 2023
Sprint-4	20	5 Days	23 Oct 2023	27 Oct 2023	20	27 Oct 2023

7. CODING & SOLUTIONING

Activity 1: importing the libraries



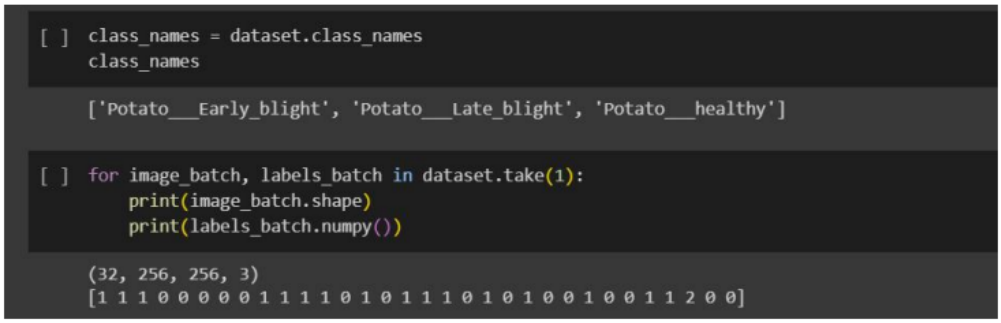
Activity 2: Data Preparation

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 visualize the data with labels.

● Extracting class names. ● Visualizing the data

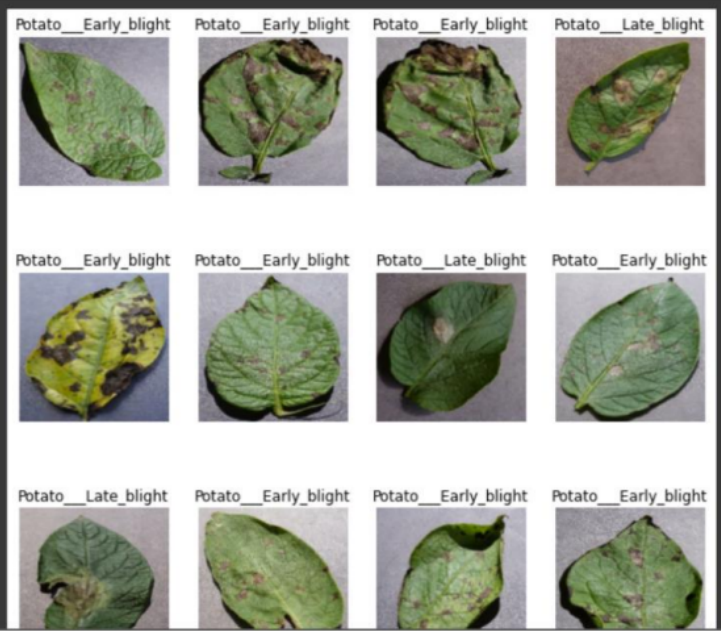
Activity 2.1: Extracting class names

● Let’s find the class names of our dataset first. To find the class name 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")
```



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

```
[ ] 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),
    ])
```

Prefetching and Caching saves a lot of time in accessing the data from disks. Refer to 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.

Model building:

Activity 1: Building a model

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

```
[ ] 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 (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

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

Activity 3: 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 [==============================] - 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
```

Model deployment

Activity 1: model evaluation

```
[ ] 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,
```

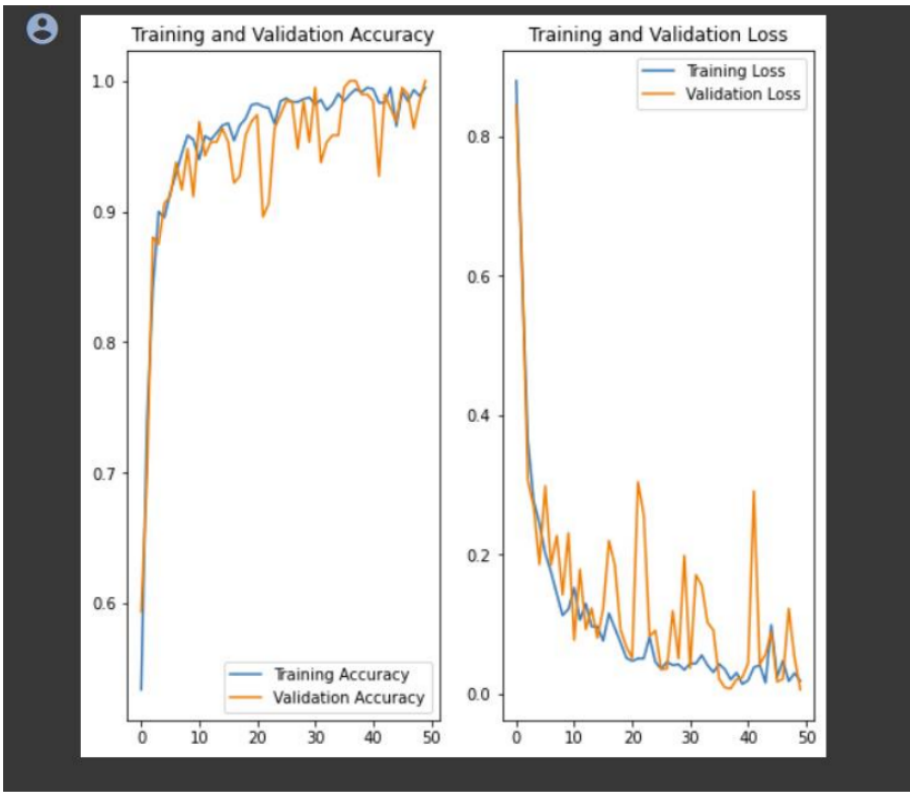
```
[ ]  acc = history.history['accuracy']
     val_acc = history.history['val_accuracy']

     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()
```

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 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.

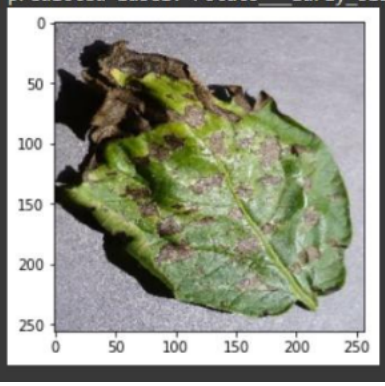
```
[ ]  import numpy as np
     for images_batch, labels_batch in test_ds.take(1):

         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
```



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

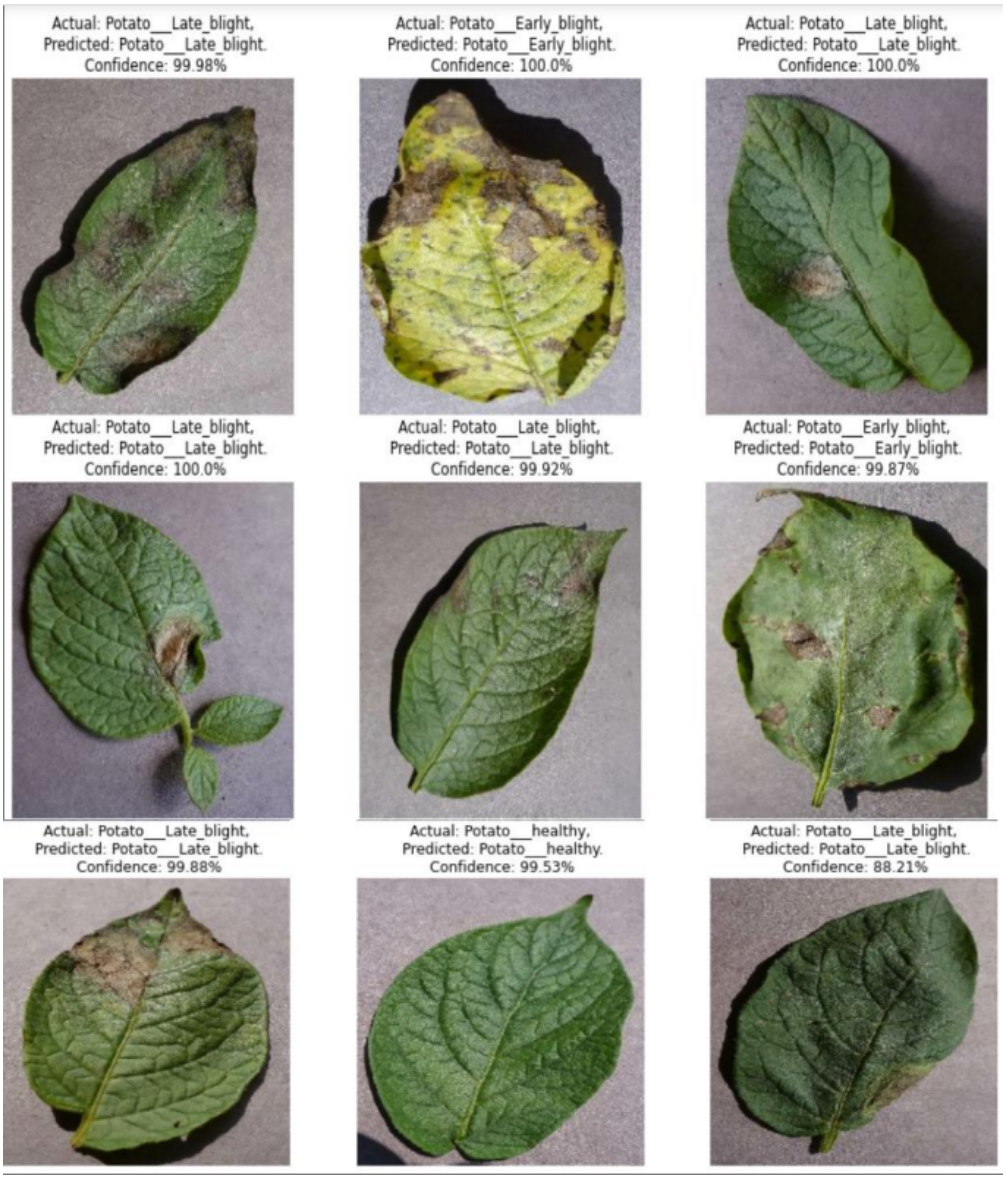
Now let us write a function for prediction, which returns the predicted class and the confidence of its prediction.

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

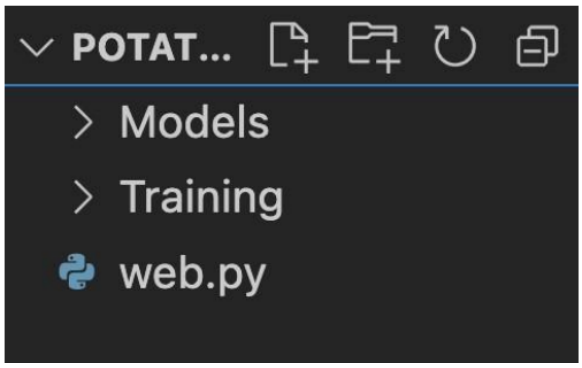
```
model.save(f"../Models/potato.h5")
```

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

Activity 3: 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 3.1: 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 3.2: 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 3.3: 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
```

Activity 4: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)
```

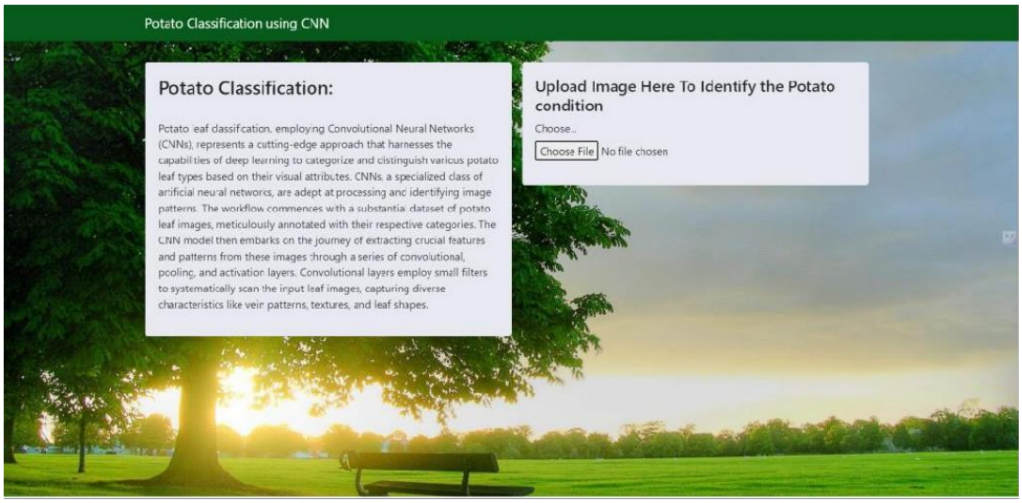
Run the flask application using the run method. By default, the flask runs on port 5000. If the port is to be changed, an argument can be passed and the port can be modified.

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.

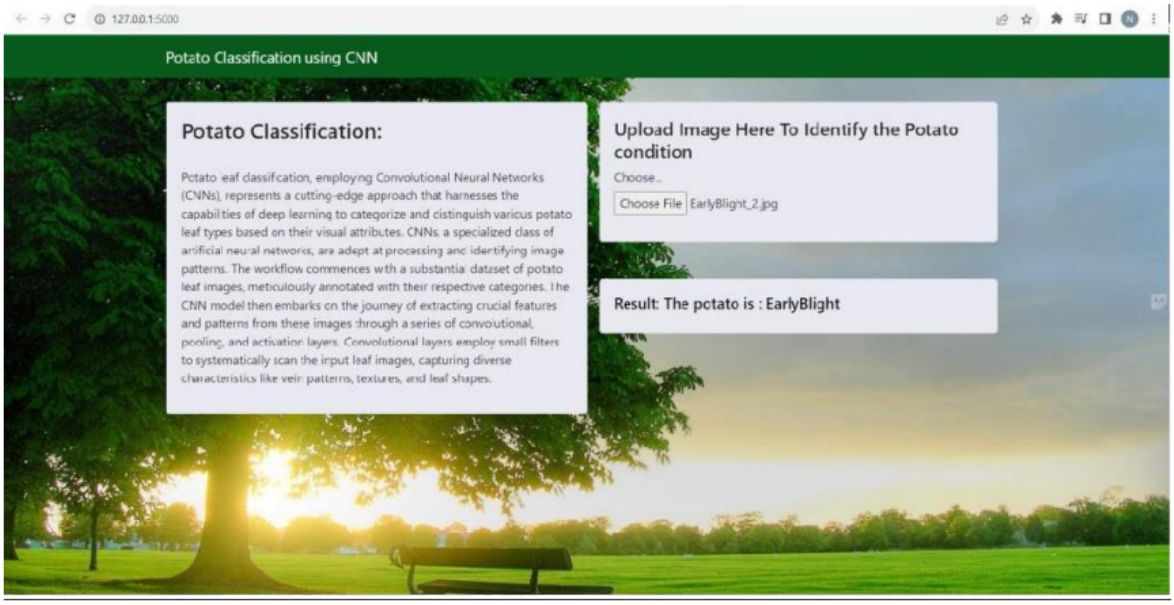
Step 1: Run the application In the anaconda prompt, navigate to the folder in which the flask app is present. When the python file is executed, the localhost is activated on port 5000 and can be accessed through it.

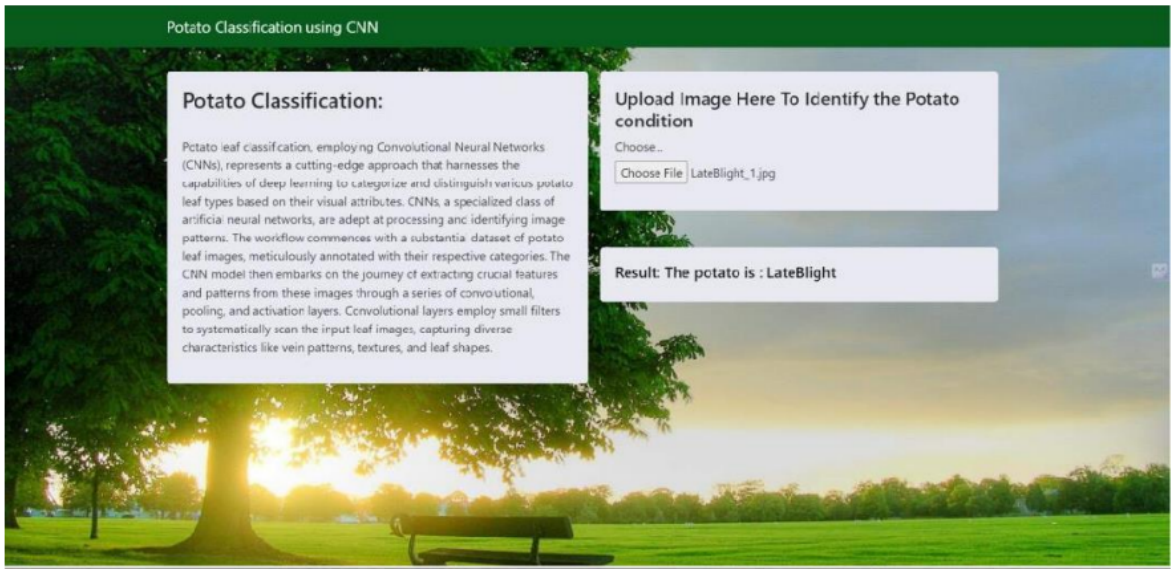
Step 2: Open the browser and navigate to localhost:5000 to check your applicationThe home page looks like this:

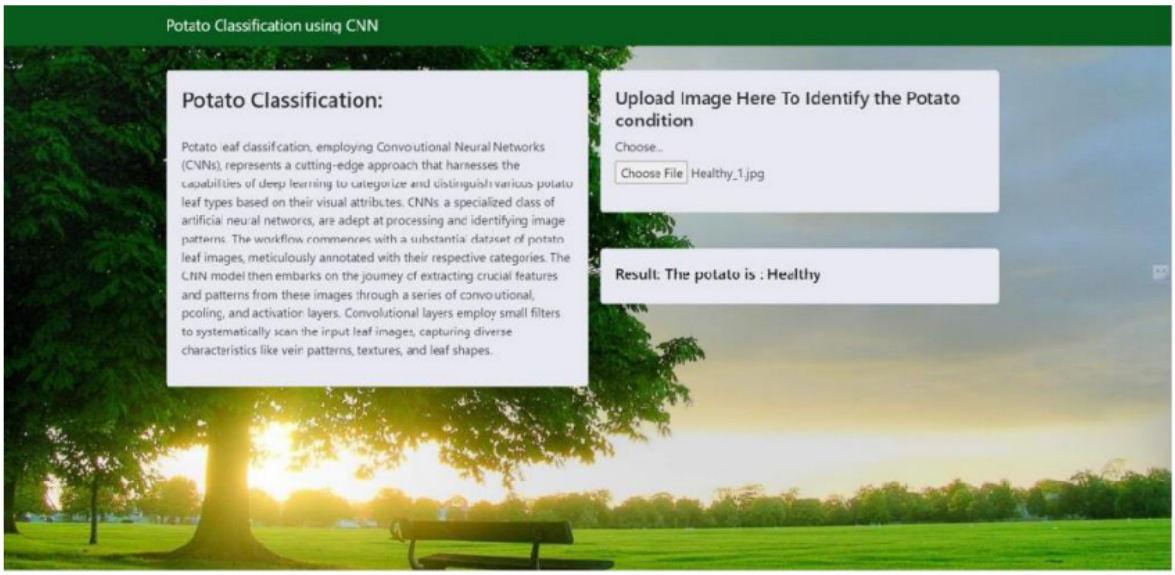


8. RESULTS

8.1 Output Screenshots







9. ADVANTAGES & DISADVANTAGES

Advantages:

Early Disease Detection: The technology makes it easier to identify potato illnesses early, enabling farmers to respond quickly to minimize crop damage. Higher crop yields and fewer financial losses may result from this.

Accurate Classification: When taught and maintained appropriately, deep learning models can yield remarkably accurate illness classification outcomes. Making educated judgements about illness prevention and treatment depends on this accuracy.

User-Friendly: The system's user interface and feedback mechanisms enable a broad spectrum of users, even those with low technical competence, to utilize it. It gives farmers the ability to control their crops with cutting-edge technologies.

Continuous Improvement: The accuracy and robustness of the model can be continuously enhanced by utilizing user feedback. As more data and feedback are collected, the system becomes increasingly reliable and effective. Scalability: The system can be scaled to accommodate a growing number of users and adapt to the specific needs of different agricultural regions and crops.

Disadvantages:

Model Complexity: Deep learning models, particularly CNNs, can be computationally intensive and may require high-performance hardware for training and inference. This could be a barrier for small-scale farmers with limited resources.

False Positives and Negatives: Like all machine learning models, false positives and false negatives can occur. Misclassifications can lead to unnecessary pesticide use or failure to address actual diseases, potentially causing harm or economic losses.

10. CONCLUSION

An encouraging approach for agriculture is the creation of a deep learning system, specifically Convolutional Neural Networks (CNNs), for the categorization of potato diseases. The prompt and precise identification of illnesses in potato plants by this technology allows for early detection, which can result in higher crop yields, lower financial losses, and improved food security. A wide range of users, including non-technical people, can utilize this technology because of its user-friendly interface and feedback systems. To guarantee the system's efficacy, however, issues with data quality, model complexity, interpretability, and continuing maintenance

must be properly handled. As the project moves forward, accuracy and relevance must be maintained by regular model changes and the incorporation of user feedback. To sum up, this project has the potential to greatly help farmers and the agricultural sector as a whole when implemented responsibly and thoughtfully.

11. FUTURE SCOPE

Expanded Crop Coverage: The underlying deep learning architecture can be extended to classify diseases in numerous other crops, while the current focus is on potato diseases. Numerous agricultural techniques can profit from its scalability, which also helps with crop health management.

AI-Driven Precision Agriculture: By integrating this system with more comprehensive precision agriculture techniques, resource allocation may be optimized, pesticide use can be decreased, and sustainable farming can be encouraged. The deep learning model is able to adjust to particular crop varieties and local conditions.

IoT Integration: The capabilities of the system can be improved by integration with Internet of Things (IoT) devices, such as automated irrigation systems and sensors. By gathering data in real time, these gadgets can enhance illness detection and prevention even further.

AI-Enhanced Recommendations: In addition to categorizing diseases, the system has the capacity to develop recommendations for disease management tactics, such as the choice of suitable pesticides, organic treatments, and preventive measures.

Mobile Apps and Accessibility: The creation of specialized mobile applications can improve accessibility and make it possible for farmers in isolated locations to use the technology for managing and diagnosing diseases.

12. APPENDIX

Source Code

The drive link for the ipynb file with the potato classification model using Resnet_50

https://drive.google.com/file/d/1Z5w5iEzjrg3eRSpLn6Pe8lCdwOTgP9vS/view?usp=drive_link

The github link for ipynb file

https://github.com/smartinternz02/SI-GuidedProject-601300-1697594000/blob/main/development%20phase/potato_disease_classification_model.ipynb

The github link for flask file

https://github.com/smartinternz02/SI-GuidedProject-601300-1697594000/blob/main/development%20phase/Flask/app.py

GitHub & Project Demo Link

https://drive.google.com/file/d/1TpJ0ctWRue9JbTVO1ldbTuHdxhuSIPFr/view?usp=drive_link