Potato Disease Classification Using Deep Learning

1. INTRODUCTION

1.1 Project Overview

Potato cultivation is a significant agricultural activity globally, contributing to food security and economic stability. However, potato plants are susceptible to various diseases, which can reduce crop yield and quality. Timely detection and classification of these diseases are crucial for effective disease management. In this project, we aim to develop a system that uses Convolutional Neural Networks (CNN) and deep learning techniques to classify potato diseases accurately and assist farmers in making informed decisions about disease control and treatment.

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

Many studies have successfully applied deep learning techniques, especially Convolutional Neural Networks (CNNs), to plant disease classification tasks. Researchers have employed transfer learning from pretrained models like VGG19, ResNet50, and Inception to efficiently classify plant diseases, including potato diseases. These models have demonstrated high accuracy in identifying various plant diseases, aiding in early detection and management. Literature in this area emphasizes the importance of large and diverse datasets, data augmentation strategies, and model interpretability, which are crucial aspects to consider when developing a potato disease classification system.

Prior research has highlighted the significance of high-quality datasets in training deep learning models for plant disease classification. Existing datasets like the Plant Village dataset and others have played a vital role in model development. Researchers have employed data preprocessing techniques, such as image resizing, normalization, and data augmentation, to improve the robustness and generalization of models. Some studies have explored domain-specific data augmentation methods to handle issues like imbalanced class distribution and variations in lighting and backgrounds, which are common challenges in plant disease classification.

While the development of accurate models is important, the literature also underscores the need for user-friendly deployment interfaces. Several studies have discussed the integration of deep learning models with user-friendly applications that empower farmers to easily classify diseases in their crops. These applications should be accessible on mobile devices, web platforms, or IoT devices, and they should provide quick and reliable results to end-users. Literature in this domain often explores interpretability and user feedback integration to ensure the practical utility of the system in real-world agricultural settings.

In conclusion, the existing literature provides a strong foundation for developing a potato disease classification system using deep learning. It highlights the critical role of dataset quality, data preprocessing, model architecture, and user-friendly deployment, with a focus on practical applications to benefit farmers and improve crop management in the agriculture sector.

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

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

2.3 Problem Statement Definition

Potato cultivation is a crucial component of global agriculture, serving as a staple food source and a source of income for many farmers. However, potato plants are susceptible to various diseases that can significantly impact crop yield and quality. The accurate and timely identification of these diseases is essential for effective disease management. Traditional disease identification methods often require manual inspection by agricultural experts, which can be time-consuming and may not be readily available to all farmers. To address this challenge, the problem at hand is to develop an automated and accurate system for classifying potato diseases using Convolutional Neural Networks (CNNs) and deep learning techniques. This system will assist farmers in making informed decisions about disease control and treatment, ultimately improving crop yield and food security.

Key aspects of the problem statement:

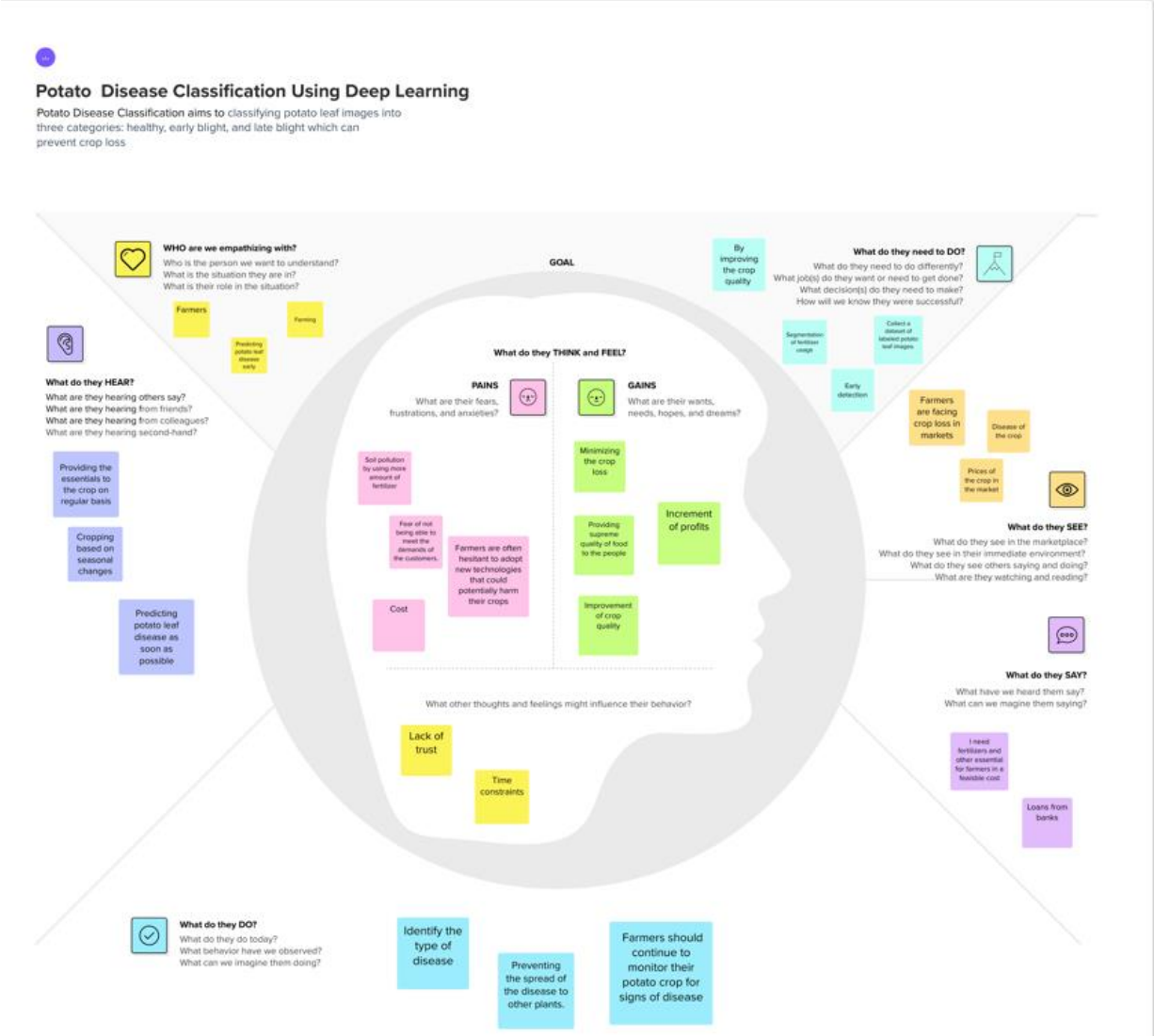
Disease Identification: The primary problem is to create a reliable method for identifying and classifying various diseases that affect potato plants, such as late blight, early blight, potato scab, and others. The system should accurately differentiate between healthy plants and diseased ones, providing specific information about the type of disease present.

Timeliness and Precision: The system should provide timely and precise disease classification results, enabling farmers to take immediate action to manage and treat the identified diseases. This addresses the need for early disease detection, which can significantly reduce crop losses and the overuse of pesticides.

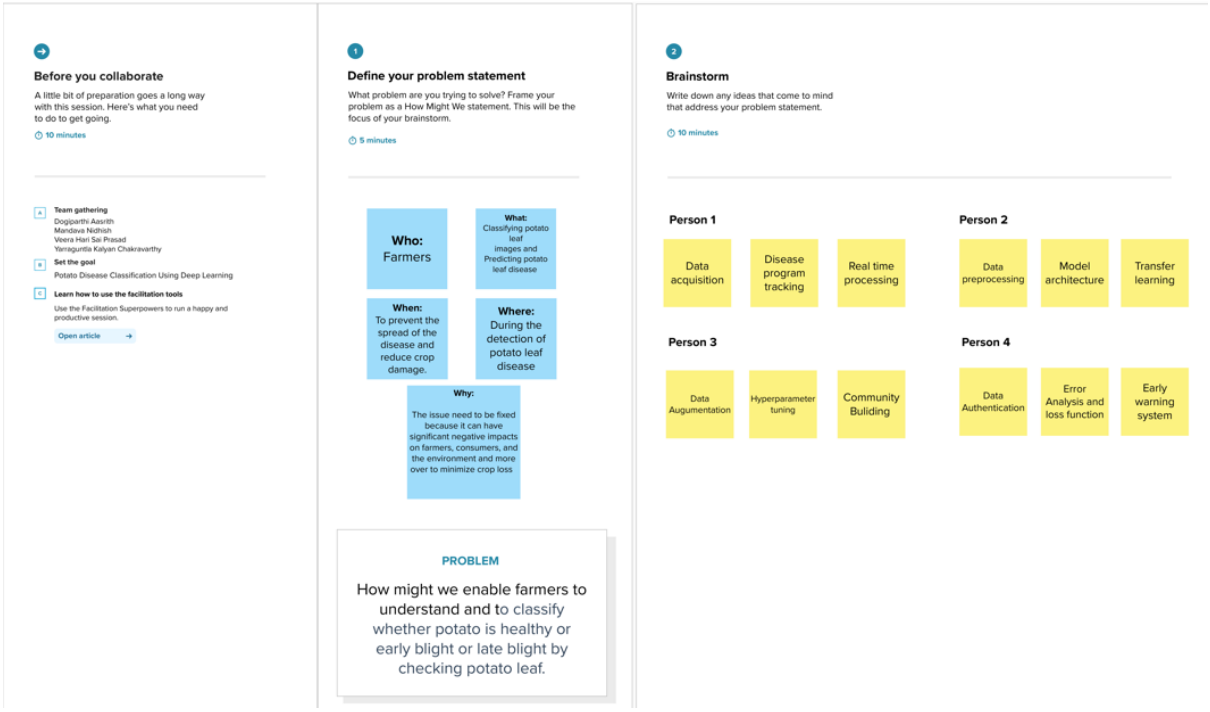
User-Friendly Interface: To solve this problem effectively, a user-friendly interface or application should be developed, allowing farmers to easily upload images of their potato plants and receive disease classification results. The system should be intuitive and adaptable to the technological and educational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

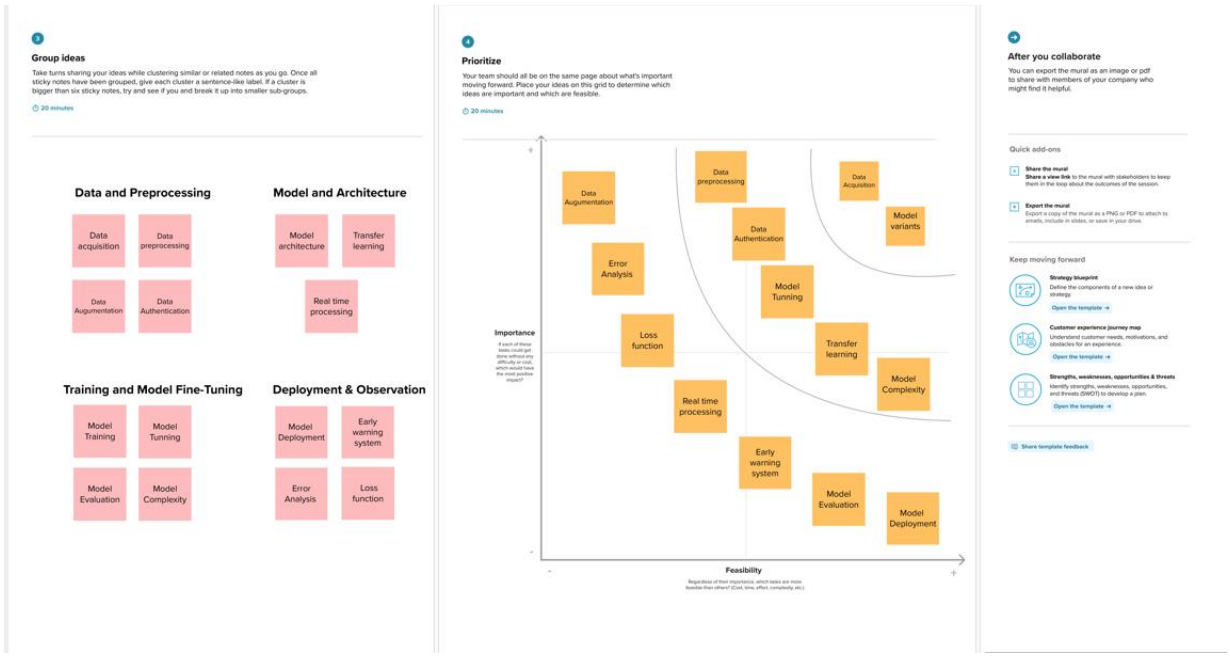
3.IDEATION & PROPOSED SOLUTION

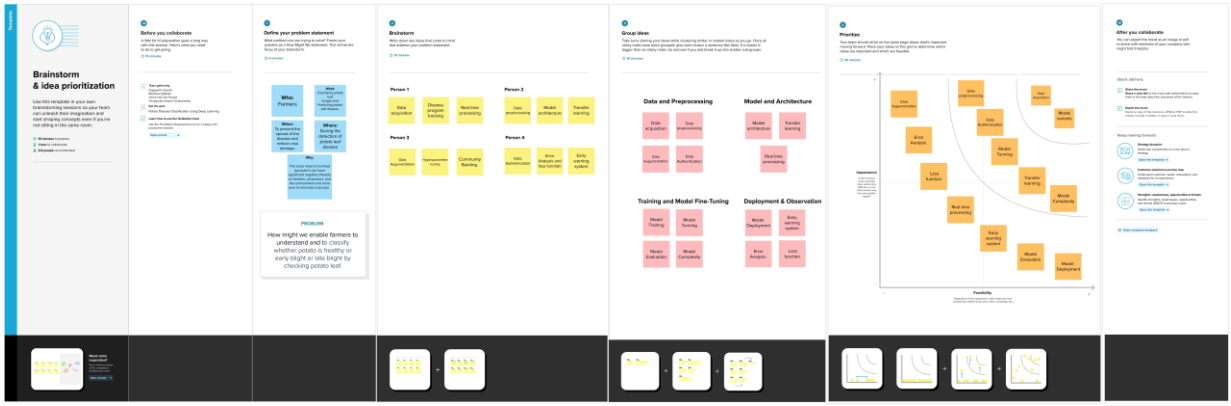
3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming







4.REQUIREMENT ANALYSIS

4.1 Functional requirement

Image Upload and Processing: The system should allow users (farmers) to upload images of potato plants for disease classification. It should preprocess these images, including resizing and normalization, to ensure compatibility with the deep learning model.

Disease Classification: The primary function of the system is to accurately classify potato diseases based on the input images. It should differentiate between healthy plants and various disease states, providing specific disease identification for each case.

User Interface: The system must offer a user-friendly interface accessible on various devices, such as smartphones, tablets, and computers. It should provide a straightforward and intuitive user experience to accommodate users with different levels of technological proficiency.

Model Interpretability: The system should offer explanations or visualizations of the model's decision-making process, providing insights into why a particular disease classification was made. This feature enhances user trust and understanding.

Scalability and Integration: Ensure that the system is scalable and can be integrated with other agricultural technologies and information systems, allowing for broader adoption and impact within the agricultural sector.

4.2 Non-Functional requirements

Accuracy and Precision: The system must achieve a high level of accuracy in disease classification to ensure reliable results for farmers. It should also provide precise disease identification, minimizing false positives and false negatives.

Response Time: The system should deliver disease classification results promptly, ideally in real-time or within a few seconds of image submission, enabling farmers to take immediate action.

Security and Privacy: Implement security measures to protect user data and ensure the privacy of uploaded images. Personal information and images should be handled with care and in compliance with relevant data protection regulations.

Robustness and Reliability: The system should be robust and reliable, capable of handling variations in image quality, lighting conditions, and disease stages. It should also have failover mechanisms in case of unexpected downtime.

Scalability and Performance: Ensure that the system can handle a potentially large number of user requests and adapt to growing user bases without compromising performance. Load balancing and optimization should be in place.

5.PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data Collection: This phase entails gathering a diverse dataset of potato leaf images from sources such as field surveys, image repositories, or other means. The collected images are then stored in a raw data repository.

Data Pre-processing: Raw potato leaf images are pre-processed to prepare them for model training. This can include resizing images, normalizing pixel values, and applying data augmentation techniques to enhance dataset diversity.

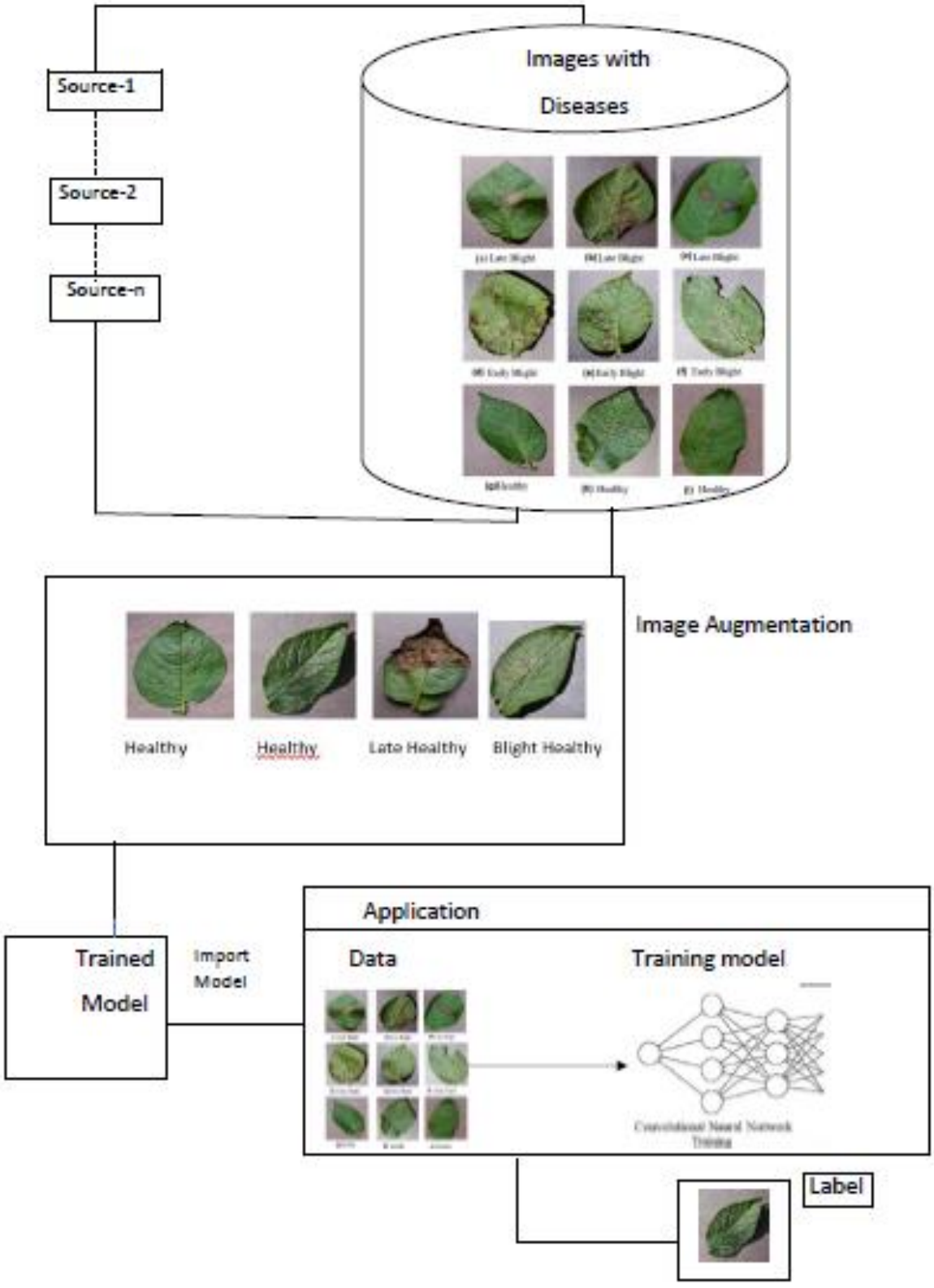
Model Training: In this stage, the pre-processed data is utilized to train a deep learning model, which learns to classify potato leaf diseases. The trained model is saved for future use.

Model Evaluation: The performance of the trained model is assessed using a separate dataset not used in training, to gauge its accuracy, sensitivity, and specificity in disease classification.

Model Deployment: This step involves deploying the trained model to make it accessible for real-world disease classification applications, either on local devices or in the cloud.

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:

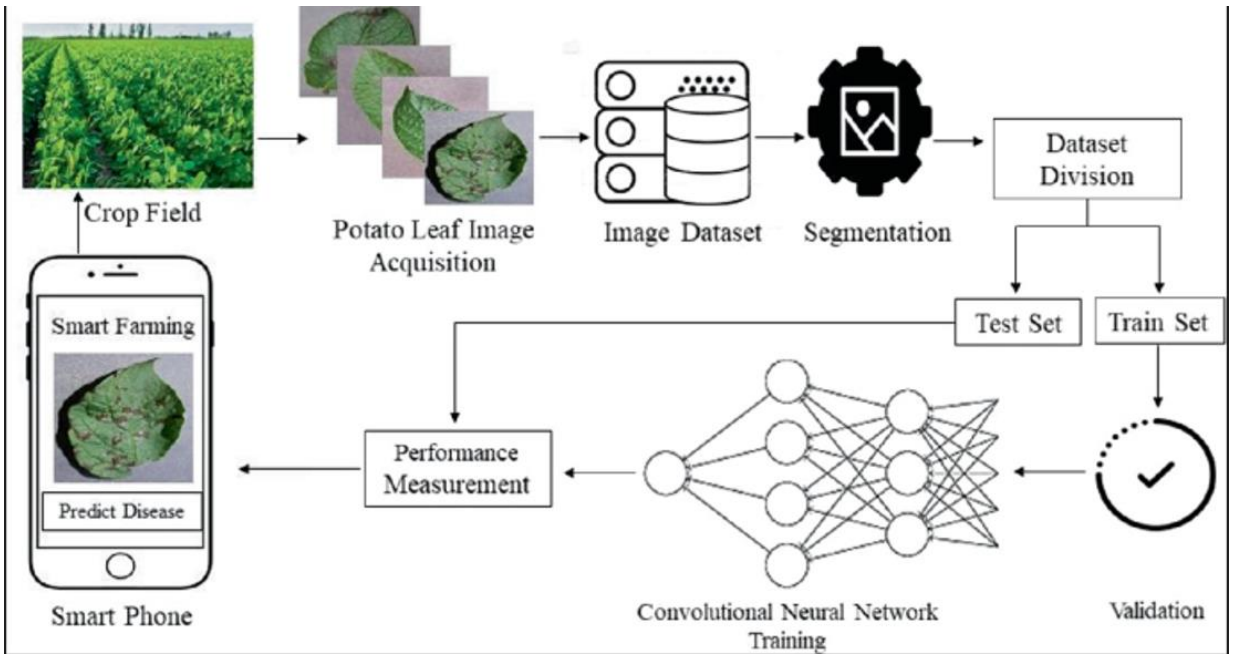


User Stories:

User Type	Functional Requirement	User Story Number	User Story/Task	Acceptance criteria	Priority	Release
	Project setup and infrastructure	User – 1	To develop a project for potato leaf classification with the help of required tools and frame works.	Successful Configuration	High	Sprint 1
Coastal Guards	Development environment	User - 2	Compile comprehensive dataset of images showcasing various types of potato diseases to train a deep learning model for potato disease classification.	Gathered a diverse dataset of images depicting various types of diseases	High	Sprint 1
Individuals	Data collection	User – 3	Prior to model training, perform pre-processing tasks, including resizing the images, standardizing pixel values, and dividing the dataset into training and test subsets.	Pre-processing and the splitting of the plant village dataset	High	Sprint 2
Researchers and Academics	Data pre-processing	User – 4	Explore and evaluate various deep learning architectures (E.g.: CNN) to select the most suitable model for potato classification.	Exploring the different deep learning models	High	Sprint 2
Organizations	Model Development	User – 5	Train the designated deep learning model using the pre-processed dataset and closely observe its performance on the validation set.	Validation using the test part of the dataset	High	Sprint 3
Institutions	Training	User – 6	Integrate data augmentation methods to enhance the model's resilience and accuracy.	Test it with the augmented dataset	Medium	Sprint 4

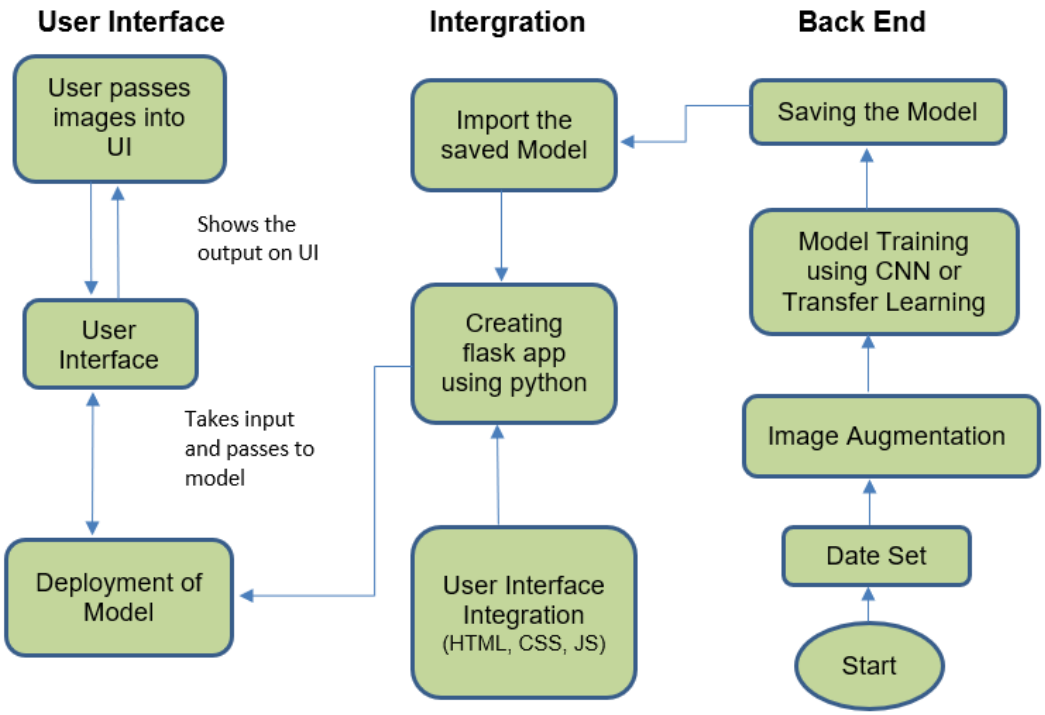
	Model Deployment	User – 7	Deploy the trained deep learning model as an API or web service and make it accessible for potato classification.	Check the scalability of the model	Medium	Sprint 4
	Testing and Quality Assurance	User – 8	Thoroughly test the model and web interface, identify and report bugs, fine-tune parameters, and optimize performance based on user feedback.	Creating the web application	Medium	Sprint 6

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	Project setup & Infrastructure	USN-1	Set up the development environment with the required tools and frameworks to start the potato leaf disease classification project	3	High	Aasrith
Sprint 1	Development environment	USN-2	Gather a diverse dataset of images containing different types of potato leaves (Potato Early Blight, Potato Late Blight, and Potato healthy leaf) for training the deep learning model.	2	High	Aasrith
Sprint 2	Data collection	USN-3	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets.	2	High	Nidhish
Sprint 2	Data preprocessing	USN-4	Explore and evaluate different deep learning architectures (e.g. CNNs) to select the most suitable model for potato leaf classification	3	High	Nidhish
Sprint 3	Model development	USN-5	Train the selected deep learning model using the preprocessed dataset and monitor its performance on the validation set.	3	High	Kalyan
Sprint 3	Training	USN-6	Implement data augmentation techniques like rotation and flipping to improve the model's robustness and accuracy.	2	Medium	Kalyan
Sprint 4	Model deployment & Integration	USN-7	Deploy the trained deep learning model as an API or web service to make it accessible for potato leaves classification. Integrate the model's API into a user-friendly web interface for users to upload images and receive potato leaves classification results.	2	Medium	Hari
Sprint 5	Testing & quality assurance	USN-8	conduct thorough testing of the model and web interface to identify and report any issues or bugs. fine-tune the model hyperparameters and optimize its performance based on user feedback and testing results.	3	Medium	Hari

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	5	3 Days 26 Oct 2023 29 Oct 2023	5	29 Oct 2023
Sprint-2	5	3 Days 29 Oct 2023 31 Oct 2023	5	30 Oct 2023
Sprint-3	5	2 Days 1 Nov 2023 3 Nov 2023	5	3 Nov 2023
Sprint-4	2	2 Days 3 Nov 2023 5 Nov 2023	2	4 Nov 2023
Sprint-5	3	1 Days 5 Nov 2023 6 Nov 2023	3	6 Nov 2023

7. CODING & SOLUTIONING

Data Pre-Processing

The dataset images are to be pre-processed before giving it to the model.

Import ImageDataGenerator Library and Configure it

ImageDataGenerator class is used to augment the images with different modifications like considering the rotation, flipping the image etc.

```
[ ]  from tensorflow.keras.preprocessing.image import ImageDataGenerator

     training_data = keras.preprocessing.image_dataset_from_directory(
         '/content/Dataset',
         batch_size = 70,
         image_size =(240,240),

         shuffle = True,
         seed =123,
         subset ='training',
         validation_split=0.15,
         )
```

Apply ImageDataGenerator functionality to Train and Validation set

Specify the path of both the folders in flow_from_directory method. We are importing the images in 240x240 pixels.

```
training_data = keras.preprocessing.image_dataset_from_directory(
    '/content/Dataset',
    batch_size = 70,
    image_size =(240,240),

    shuffle = True,
    seed =123,
    subset ='training',
    validation_split=0.15,
    )
validation_data =keras.preprocessing.image_dataset_from_directory(
    '/content/Dataset',
    batch_size = 70,
    image_size =(240,240),

    shuffle = True,
    seed =123,
    validation_split =0.15,
    subset ='validation',
    )
```

```
Found 4072 files belonging to 3 classes.
Using 3462 files for training.
Found 4072 files belonging to 3 classes.
Using 610 files for validation.
```

Building the model

We will be creating the pre-trained VGG19 model and ResNet50 model for predicting custom classes and choose the model with the highest accuracy.

Creating and compiling the model

VGG19

Firstly, import the necessary libraries and define a function for creation of the model. Next, call the pre trained model with the parameter include_top=False, as we need to use the model for predicting custom images. Next, add dense layers to the top model. Finally, add an output layer with the number of neurons equal to out output classes.

```
[ ] from tensorflow.keras.applications import VGG19
    from tensorflow.keras.models import Model
    from tensorflow.keras.layers import Flatten, Dense
    from tensorflow.keras.optimizers import Adam
```

```
[ ] base_model = VGG19(weights='imagenet', include_top=False, input_shape=(300, 300, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80134624/80134624 [==============================] - 0s 0us/step
```

```
[ ] for layer in base_model.layers:
        layer.trainable = False

    x = Flatten()(base_model.output)
    x = Dense(256, activation='relu')(x)
    x = Dense(128, activation='relu')(x)
    output = Dense(3, activation='softmax')(x)  # Replace 'num_classes' with your actual number of classes
```

```
[ ] model = Model(inputs=base_model.input, outputs=output)
    model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

ResNet 50

Firstly, import the necessary libraries and define a function for creation of the model. Next, call the pre trained model with the parameter include_top=False, as we need to use the model for predicting custom images. Next, add dense layers to the top model. Finally, add an output layer with the number of neurons equal to out output classes.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

training_data = keras.preprocessing.image_dataset_from_directory(
    '/content/Dataset',
    batch_size = 70,
    image_size =(240,240),

    shuffle = True,
    seed =123,
    subset ='training',
    validation_split=0.15,
    )
validation_data =keras.preprocessing.image_dataset_from_directory(
    '/content/Dataset',
    batch_size = 70,
    image_size =(240,240),

    shuffle = True,
    seed =123,
    validation_split =0.15,
    subset ='validation',
    )
```

```
resnet_model = Sequential()
pretrained_model= ResNet50(include_top=False,
                   input_shape=(240,240,3),
                   pooling='avg',
                   weights='imagenet')
for layer in pretrained_model.layers:
  layer.trainable=False
resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(BatchNormalization())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(BatchNormalization())
resnet_model.add(Dense(4, activation='softmax'))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5
94765736/94765736 [==============================] - 5s 0us/step
```

```
resnet_model.compile(
    loss='sparse_categorical_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)
```

VGG19

Model Summary

Total params: 30674755

Trainable params: 10650371

Non-trainable params: 20024384

```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 300, 300, 3)]     0
 block1_conv1 (Conv2D)       (None, 300, 300, 64)      1792
 block1_conv2 (Conv2D)       (None, 300, 300, 64)      36928
 block1_pool (MaxPooling2D)  (None, 150, 150, 64)      0
 block2_conv1 (Conv2D)       (None, 150, 150, 128)     73856
 block2_conv2 (Conv2D)       (None, 150, 150, 128)     147584
 block2_pool (MaxPooling2D)  (None, 75, 75, 128)       0
 block3_conv1 (Conv2D)       (None, 75, 75, 256)       295168
 block3_conv2 (Conv2D)       (None, 75, 75, 256)       590080
 block3_conv3 (Conv2D)       (None, 75, 75, 256)       590080
 block3_conv4 (Conv2D)       (None, 75, 75, 256)       590080
 block3_pool (MaxPooling2D)  (None, 37, 37, 256)       0
 block4_conv1 (Conv2D)       (None, 37, 37, 512)       1180160
 block4_conv2 (Conv2D)       (None, 37, 37, 512)       2359808
 block4_conv3 (Conv2D)       (None, 37, 37, 512)       2359808
 block4_conv4 (Conv2D)       (None, 37, 37, 512)       2359808
 block4_pool (MaxPooling2D)  (None, 18, 18, 512)       0
 block5_conv1 (Conv2D)       (None, 18, 18, 512)       2359808
 block5_conv2 (Conv2D)       (None, 18, 18, 512)       2359808
 block5_conv3 (Conv2D)       (None, 18, 18, 512)       2359808
 block5_conv4 (Conv2D)       (None, 18, 18, 512)       2359808
 block5_pool (MaxPooling2D)  (None, 9, 9, 512)         0
 flatten (Flatten)           (None, 41472)             0
 dense (Dense)               (None, 256)               10617088
 dense_1 (Dense)             (None, 128)               32896
 dense_2 (Dense)             (None, 3)                 387
=================================================================
Total params: 30674755 (117.01 MB)
Trainable params: 10650371 (40.63 MB)
Non-trainable params: 20024384 (76.39 MB)
_________________________________________________________________
```

Resnet_50

Model Summary

Total params: 24649092

Trainable params: 1056260

Non-trainable params: 23592832

```
resnet_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 flatten (Flatten)           (None, 2048)              0

 batch_normalization (Batch  (None, 2048)              8192
 Normalization)

 dense (Dense)               (None, 512)               1049088

 batch_normalization_1 (Bat  (None, 512)               2048
 chNormalization)

 dense_1 (Dense)             (None, 4)                 2052

=================================================================
Total params: 24649092 (94.03 MB)
Trainable params: 1056260 (4.03 MB)
Non-trainable params: 23592832 (90.00 MB)
_________________________________________________________________
```

8. PERFORMANCE TESTING

8.1 Performance Metrics

VVG19

Training Accuracy: 94.81

Validation Accuracy: 75.52

```
history = model.fit(train_batch,
                    steps_per_epoch=len(train_batch),
                    epochs=20,
                    validation_data=valid_batch,
                    validation_steps=len(valid_batch))
```

```
Epoch 1/20
51/51 [==============================] - 137s 2s/step - loss: 1.1687 - accuracy: 0.6563 - val_loss: 0.8760 - val_accuracy: 0.5916
Epoch 2/20
51/51 [==============================] - 96s 2s/step - loss: 0.3884 - accuracy: 0.8460 - val_loss: 0.5634 - val_accuracy: 0.7626
Epoch 3/20
51/51 [==============================] - 97s 2s/step - loss: 0.3373 - accuracy: 0.8678 - val_loss: 0.7003 - val_accuracy: 0.7036
Epoch 4/20
51/51 [==============================] - 96s 2s/step - loss: 0.2695 - accuracy: 0.9006 - val_loss: 0.6073 - val_accuracy: 0.7651
Epoch 5/20
51/51 [==============================] - 95s 2s/step - loss: 0.2752 - accuracy: 0.8978 - val_loss: 0.7050 - val_accuracy: 0.7491
Epoch 6/20
51/51 [==============================] - 96s 2s/step - loss: 0.2926 - accuracy: 0.8972 - val_loss: 0.7410 - val_accuracy: 0.6974
Epoch 7/20
51/51 [==============================] - 98s 2s/step - loss: 0.3115 - accuracy: 0.8819 - val_loss: 1.1235 - val_accuracy: 0.6384
Epoch 8/20
51/51 [==============================] - 97s 2s/step - loss: 0.3296 - accuracy: 0.8751 - val_loss: 0.7098 - val_accuracy: 0.7491
Epoch 9/20
51/51 [==============================] - 95s 2s/step - loss: 0.1918 - accuracy: 0.9276 - val_loss: 0.5112 - val_accuracy: 0.7983
Epoch 10/20
51/51 [==============================] - 99s 2s/step - loss: 0.1811 - accuracy: 0.9310 - val_loss: 0.6497 - val_accuracy: 0.7749
Epoch 11/20
51/51 [==============================] - 99s 2s/step - loss: 0.2209 - accuracy: 0.9159 - val_loss: 0.4945 - val_accuracy: 0.8057
Epoch 12/20
51/51 [==============================] - 97s 2s/step - loss: 0.1491 - accuracy: 0.9506 - val_loss: 0.5268 - val_accuracy: 0.8106
Epoch 13/20
51/51 [==============================] - 97s 2s/step - loss: 0.1335 - accuracy: 0.9509 - val_loss: 0.5203 - val_accuracy: 0.8093
Epoch 14/20
51/51 [==============================] - 98s 2s/step - loss: 0.1890 - accuracy: 0.9270 - val_loss: 0.6613 - val_accuracy: 0.7798
Epoch 15/20
51/51 [==============================] - 97s 2s/step - loss: 0.1250 - accuracy: 0.9555 - val_loss: 0.7076 - val_accuracy: 0.7774
Epoch 16/20
51/51 [==============================] - 98s 2s/step - loss: 0.1852 - accuracy: 0.9310 - val_loss: 0.4994 - val_accuracy: 0.8093
Epoch 17/20
51/51 [==============================] - 97s 2s/step - loss: 0.1498 - accuracy: 0.9488 - val_loss: 0.9279 - val_accuracy: 0.7478
Epoch 18/20
51/51 [==============================] - 96s 2s/step - loss: 0.1551 - accuracy: 0.9423 - val_loss: 0.5196 - val_accuracy: 0.8155
Epoch 19/20
51/51 [==============================] - 97s 2s/step - loss: 0.1776 - accuracy: 0.9307 - val_loss: 0.9083 - val_accuracy: 0.7294
Epoch 20/20
51/51 [==============================] - 98s 2s/step - loss: 0.1404 - accuracy: 0.9481 - val_loss: 0.7183 - val_accuracy: 0.7552
```

Resnet_50

Accuracy

Training Accuracy: 99.65

Validation Accuracy: 97.21

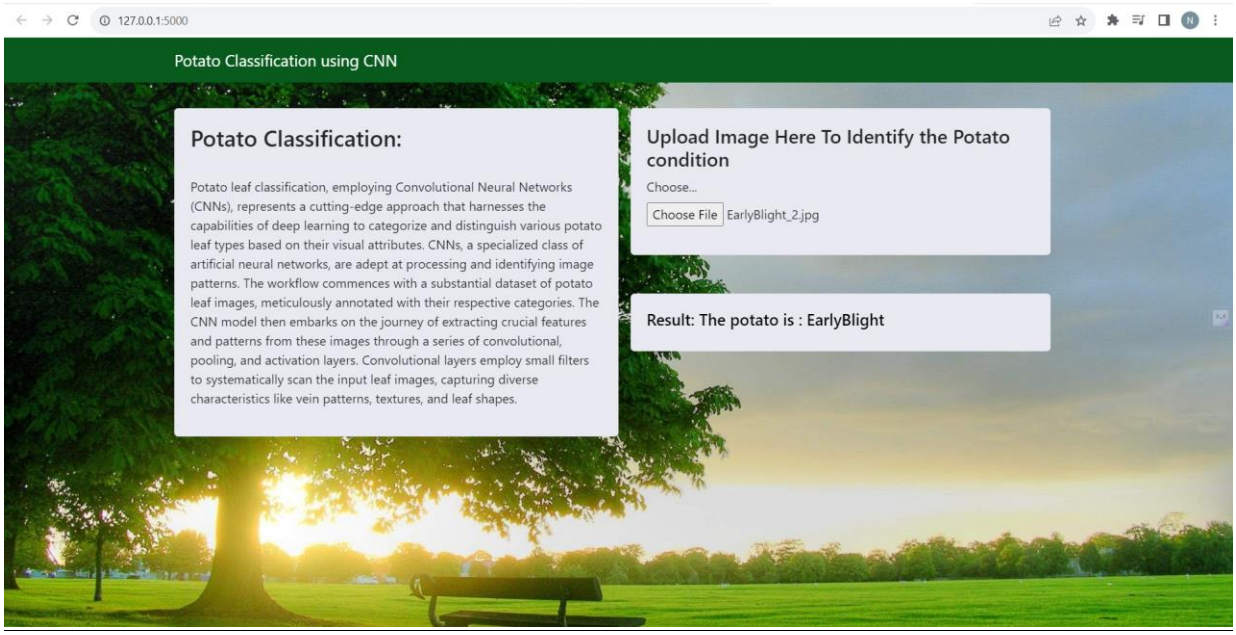
```
[ ]  history = resnet_model.fit(training_data,
                        steps_per_epoch=len(training_data),
                        epochs=20,
                        validation_data=validation_data,
                        validation_steps=len(validation_data))
```

```
Epoch 1/20
50/50 [==============================] - 32s 358ms/step - loss: 0.3138 - accuracy: 0.9113 - val_loss: 0.2253 - val_accuracy: 0.9426
Epoch 2/20
50/50 [==============================] - 15s 283ms/step - loss: 0.0657 - accuracy: 0.9786 - val_loss: 0.1386 - val_accuracy: 0.9639
Epoch 3/20
50/50 [==============================] - 14s 278ms/step - loss: 0.0279 - accuracy: 0.9916 - val_loss: 0.1251 - val_accuracy: 0.9574
Epoch 4/20
50/50 [==============================] - 15s 286ms/step - loss: 0.0209 - accuracy: 0.9928 - val_loss: 0.1143 - val_accuracy: 0.9590
Epoch 5/20
50/50 [==============================] - 15s 282ms/step - loss: 0.0167 - accuracy: 0.9939 - val_loss: 0.0891 - val_accuracy: 0.9623
Epoch 6/20
50/50 [==============================] - 15s 282ms/step - loss: 0.0093 - accuracy: 0.9974 - val_loss: 0.1041 - val_accuracy: 0.9590
Epoch 7/20
50/50 [==============================] - 15s 283ms/step - loss: 0.0140 - accuracy: 0.9954 - val_loss: 0.0904 - val_accuracy: 0.9639
Epoch 8/20
50/50 [==============================] - 15s 284ms/step - loss: 0.0068 - accuracy: 0.9980 - val_loss: 0.0966 - val_accuracy: 0.9689
Epoch 9/20
50/50 [==============================] - 15s 283ms/step - loss: 0.0060 - accuracy: 0.9991 - val_loss: 0.0915 - val_accuracy: 0.9689
Epoch 10/20
50/50 [==============================] - 15s 286ms/step - loss: 0.0074 - accuracy: 0.9974 - val_loss: 0.0876 - val_accuracy: 0.9689
Epoch 11/20
50/50 [==============================] - 15s 285ms/step - loss: 0.0069 - accuracy: 0.9980 - val_loss: 0.0920 - val_accuracy: 0.9672
Epoch 12/20
50/50 [==============================] - 15s 285ms/step - loss: 0.0043 - accuracy: 0.9997 - val_loss: 0.1078 - val_accuracy: 0.9689
Epoch 13/20
50/50 [==============================] - 15s 300ms/step - loss: 0.0040 - accuracy: 0.9991 - val_loss: 0.1001 - val_accuracy: 0.9623
Epoch 14/20
50/50 [==============================] - 15s 287ms/step - loss: 0.0025 - accuracy: 0.9994 - val_loss: 0.1086 - val_accuracy: 0.9656
Epoch 15/20
50/50 [==============================] - 15s 288ms/step - loss: 0.0068 - accuracy: 0.9977 - val_loss: 0.1400 - val_accuracy: 0.9557
Epoch 16/20
50/50 [==============================] - 15s 288ms/step - loss: 0.0107 - accuracy: 0.9965 - val_loss: 0.1128 - val_accuracy: 0.9607
Epoch 17/20
50/50 [==============================] - 15s 287ms/step - loss: 0.0095 - accuracy: 0.9965 - val_loss: 0.1217 - val_accuracy: 0.9623
Epoch 18/20
50/50 [==============================] - 15s 286ms/step - loss: 0.0166 - accuracy: 0.9939 - val_loss: 0.1332 - val_accuracy: 0.9574
Epoch 19/20
50/50 [==============================] - 15s 286ms/step - loss: 0.0134 - accuracy: 0.9945 - val_loss: 0.1144 - val_accuracy: 0.9689
Epoch 20/20
50/50 [==============================] - 17s 339ms/step - loss: 0.0082 - accuracy: 0.9965 - val_loss: 0.1230 - val_accuracy: 0.9721
```

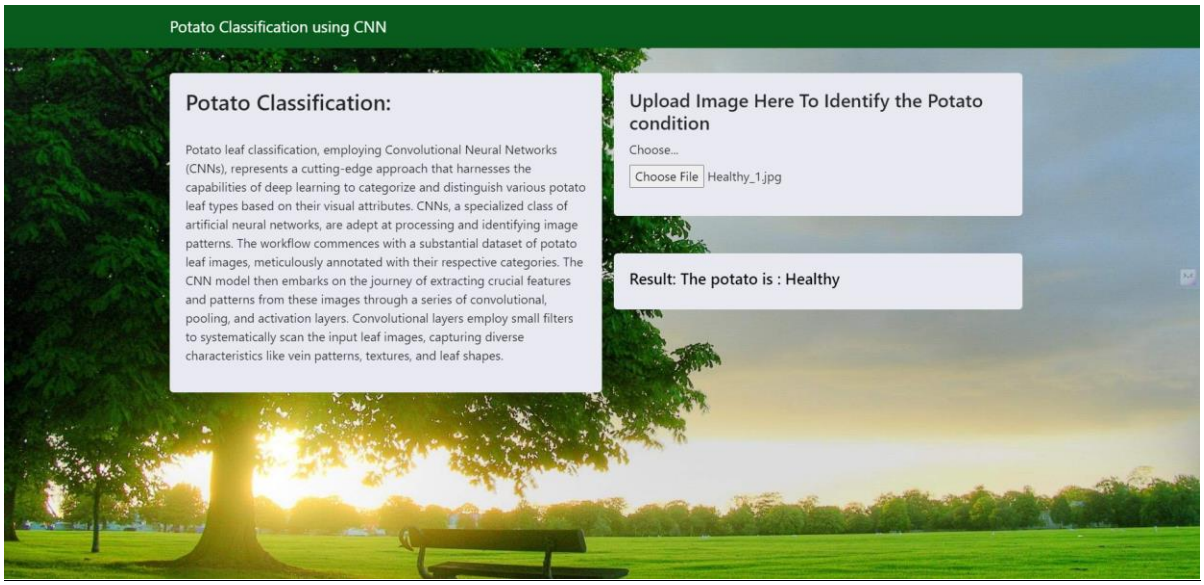
9. RESULTS

9.1 Output Screenshots

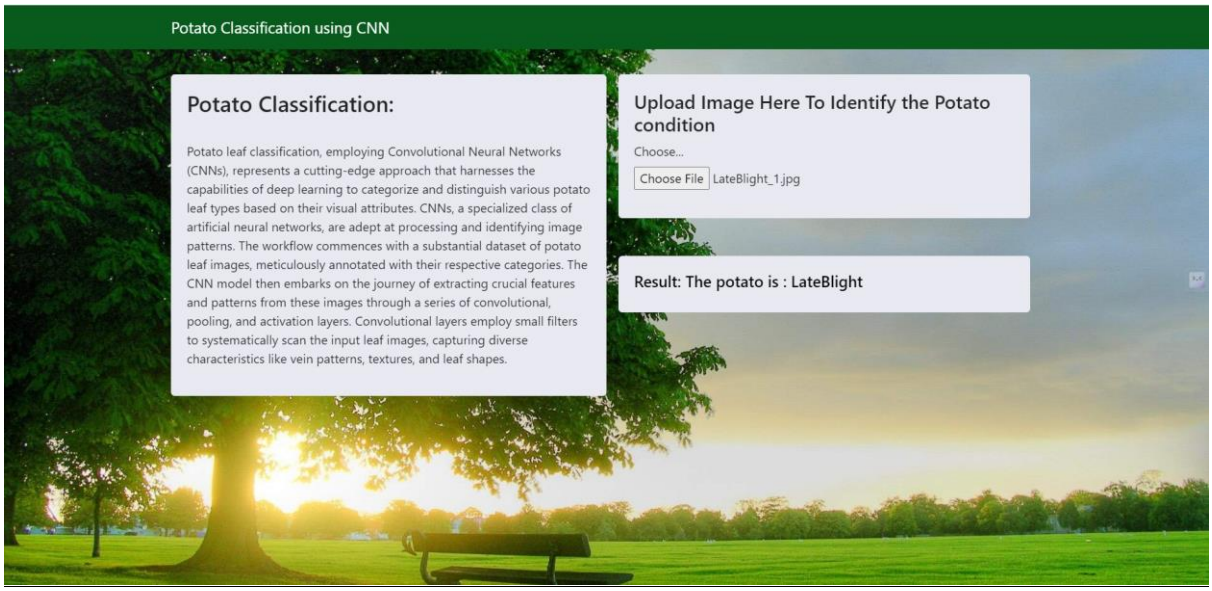
Output-1



Output-2



Output-3



10. ADVANTAGES & DISADVANTAGES

Advantages:

Early Disease Detection: The system facilitates early detection of potato diseases, allowing farmers to take prompt action to mitigate crop damage. This can lead to higher crop yields and reduced economic losses.

Accurate Classification: Deep learning models, if properly trained and maintained, can provide highly accurate disease classification results. This accuracy is crucial for making informed decisions about disease control and treatment.

User-Friendly: The user interface and feedback mechanisms make the system accessible to a wide range of users, including those with limited technical expertise. It empowers farmers to leverage advanced technology for crop management.

Continuous Improvement: User feedback can be used to continuously improve the model's accuracy and robustness. 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:

Data Requirements: Deep learning models require large and diverse datasets for training, which can be challenging to obtain, especially for rare or region-specific diseases. Acquiring and annotating such datasets can be time-consuming and costly.

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.

Interpretability: While deep learning models can achieve high accuracy, they are often considered "black boxes," making it difficult to explain why a specific classification decision was made. Model interpretability is a challenge and can impact user trust and acceptance.

Maintenance and Updates: Continuous model maintenance and updates are required to keep the system effective. This includes retraining the model with new data and addressing evolving disease patterns.

Privacy and Security: Handling user-generated data, including images, requires robust security measures to protect user privacy. Data breaches or misuse of information can have legal and ethical implications.

Dependency on Technology: The system relies on technology infrastructure, including internet access and compatible devices. In regions with poor connectivity or limited access to devices, the system may not be as effective.

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.

11. CONCLUSION

The development of a potato disease classification system using deep learning, particularly Convolutional Neural Networks (CNNs), offers a promising solution for agriculture. This system's ability to swiftly and accurately identify diseases in potato plants provides early detection, which can lead to increased crop yields, reduced economic losses, and enhanced food security. The user-friendly interface and feedback mechanisms make this technology accessible to a diverse user base, including non-technical users. However, challenges related to data quality, model complexity, interpretability, and ongoing maintenance must be carefully addressed to ensure the system's effectiveness. Regular model updates and user feedback integration are essential to maintaining accuracy and relevance as the project progresses. In conclusion, this project has the potential to significantly benefit farmers and the broader agricultural industry when implemented responsibly and thoughtfully.

12. FUTURE SCOPE

Expanded Crop Coverage: While the current focus is on potato diseases, the underlying deep learning framework can be extended to classify diseases in various other crops. This scalability can benefit a wide range of agricultural practices and contribute to crop health management.

AI-Driven Precision Agriculture: Incorporating this system into broader precision agriculture practices can optimize resource allocation, reducing the use of pesticides and promoting sustainable farming. The deep learning model can adapt to specific regional conditions and crop varieties.

IoT Integration: Integration with the Internet of Things (IoT) devices, such as sensors and automated irrigation systems, can enhance the system's capabilities. These devices can collect real-time data, further improving disease detection and prevention.

AI-Enhanced Recommendations: The system can evolve to not only classify diseases but also provide recommendations for disease management strategies, including the selection of appropriate pesticides, organic treatments, and preventive measures.

Mobile Apps and Accessibility: Development of dedicated mobile applications can increase accessibility, enabling farmers in remote areas to utilize the technology for disease identification and management.

13. APPENDIX

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

https://drive.google.com/file/d/1UiFtWxk5U2ZYYhEl8tft5WP5Mv1qUdhy/view?usp=sharing

The github link for the ipynb file

https://github.com/smartinternz02/SI-GuidedProject-600098-1697456559/blob/main/4.Development%20Phase/Potato_Disease_Classification_Resnet50.ipynb

The github link for the flask source code

https://github.com/smartinternz02/SI-GuidedProject-600098-1697456559/tree/main/4.Development%20Phase

Project Demo Video link

https://drive.google.com/file/d/1hnF6w1YgRrTLm2VuHH6EiiWiuCZrS9W5/view?usp=sharing