Potato Leaf Plant Disease Detection and Classification using Convolutional Neural Networks (CNN)

[1]Aditya Dey,
Department of Computing Technologies, College of Engineering and Technology, Student of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur, 603203
India
[1]ad1180@srmist.edu.in

[2]Jayanto Sett,
Department of Computing Technologies, College of Engineering and Technology, Student of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur, 603203
India
[2]js9190@srmist.edu .in

[3*]Dr. R. Anto Arockia Rosaline,
Department of Computing Technologies, College of Engineering and Technology, Faculty of Engineering and Technology, SRM Institute of Science and Technology, Kattankulathur, 603203
India
[3*]antoaror@srmist.edu.in

Abstract – **The 'Potato Leaf Disease Detection' project harnesses Google's TensorFlow to combat early and late blight diseases in potato cultivation. This model promises swift and accurate disease identification, enabling timely interventions, reducing crop losses, and bolstering potato farming sustainability. It aims to revolutionize agriculture, enhancing global food security and innovation. Blight diseases like Phytophthora infestans and Alternaria Solani pose significant threats to potatoes, with conventional identification methods being slow and ineffective. By providing farmers with a reliable disease detection method, the project empowers timely actions, significantly reducing crop losses and promoting sustainability, essential for global food security. It aligns with broader agricultural innovation goals and supports the United Nations' Sustainable Development Goals.**

Keywords— convolutional neural network, detection system, plant disease, support vector machine

Introduction

Plant diseases pose an enduring global challenge, threatening agriculture and food security in a world marked by population growth and climate change. These diseases, caused by various pathogens like fungi, bacteria, viruses, nematodes, and phytoplasmas, disrupt plant functions, leading to visible symptoms like wilting, discoloration, necrosis, and deformities. However, the impact extends further, with up to 40% of global crop production lost, causing economic instability, higher food prices, reduced farmer incomes, and increased production costs.

Traditional plant disease diagnosis, relying on visual inspections by experts, has limitations such as subjectivity, labor-intensity, and scalability issues. Human variability and expertise play a role, hindering widespread application across agricultural landscapes.

In response to these challenges, a technological revolution is underway. It combines digital imaging, machine learning, and computer vision. Accessible digital imaging tools, from smartphones to drones, enable detailed plant and disease image capture, forming the basis of advanced disease detection systems. Machine learning and deep learning are emerging as powerful tools for automated image analysis, with computer vision enabling the interpretation of visual data, even identifying subtle disease symptoms that escape human perception.

Machine learning models, especially convolutional neural networks (CNNs), have been pivotal in accurately distinguishing healthy from diseased plants. This transformative approach promises a streamlined, accurate, and scalable method for plant disease detection and classification, holding great potential for securing agriculture and food production.

I. State-of-the-art

A: Plant Disease Detection: Datasets

Expert systems have revolutionized the identification and management of agricultural plants, significantly boosting plant production in agriculture due to technological advancements. These systems have led to the development of various disease detection models, notably PlantDoc and PlantVillage.

The PlantVillage dataset is the most extensive, containing over 163,000 images representing 14 crop species: Raspberry, Apple, Cherry, Orange, Blueberry, Pepper Bell, Squash, Tomato, Grape, Peach, Soybean, Strawberry, Corn, and Potato. These images include both healthy and diseased crop leaves and are accessible on the Kaggle platform.

For a specific model using approximately 20,600 images from the PlantVillage Dataset, the focus is solely on the Potato category. This model is trained on Potato-related images, including both healthy and diseased leaves with issues like Bacterial Spots, Blights, Viruses, Mold, and Spots.

The desired outcome is to identify 2,968 files belonging to three classes specifically related to Potato, while disregarding other plant species in the dataset.

B: Models for Plant Disease Detection

Comparing our literature survey with others in the field, where we focus on utilizing Convolutional Neural Networks (CNNs) and achieving an impressive 98.75% accuracy in plant disease detection, we can see the evolving landscape of research in this area.

In 2021, A. Lakshmanarao et al. presented their work at the International Conference on Artificial Intelligence and Machine Vision (AIMV), emphasizing the importance of ConvNets and the need for large-scale datasets in plant disease prediction and categorization. Their work underlines architectural advances, like new network structures and transfer learning, as key contributors to increased accuracy, although challenges remain regarding model interpretability and robustness.

In 2022, Dhruvi Gosai et al. analyzed an article discussing the significance of plant disease identification in agriculture and highlighted the dataset and machine learning techniques used for disease categorization.

At the 2023 ICAAIC, Kethsy Prabavathy et al. presented their study on "Plant Leaf Disease Detection using Machine Learning," emphasizing the dataset and data preprocessing methods and providing insights into the importance of disease detection in agriculture. They summarized experimental findings, focusing on the effectiveness of their machine learning models.

In 2023 (ICONSTEM), T. Priyaradhikadevi et al. discussed "Leaf Disease Detection Using Machine Learning Algorithm," addressing dataset and data preprocessing, and stressing the need for automated leaf disease diagnosis in agriculture. The study examined machine learning techniques, experiment findings, and their implications for crop protection and agricultural sustainability.

Emmanuel Moupojou et al. reviewed the "FieldPlant" dataset and deep learning methods for plant disease detection, highlighting the dataset's significance and structure. Their work focused on the importance of automated plant disease detection in agriculture.

Sannakki and Singh (2018) offered a comprehensive survey on tomato disease and pest detection, including CNN-based methods, while Samarendra et al. (2019) provided a review of various plant disease detection approaches, though not specific to CNNs.

In 2019 (ICACCS), U. Shruthi et al. conducted a thorough review of machine learning classification techniques for plant disease detection, emphasizing the importance of identifying plant diseases in agriculture and discussing machine learning research issues and future approaches.

At the 2019 (ICCSP), L. Sherly Puspha Annabel et al. reviewed machine learning techniques for plant leaf disease detection, providing a comparative analysis and addressing datasets, data preprocessing, and future research possibilities.

In 2023 (ICESC), Ajmeera Kiran et al. explored machine learning-integrated image processing methods for disease detection, emphasizing their potential to enhance crop protection and agricultural sustainability.

At the 2023 (ICCCI), Yogesh H. Bhosale et al. delved into the integration of deep neural networks, machine learning, image processing, and precision agriculture for plant disease management, underscoring the crucial role of automated disease detection in agriculture.

Pushkara Sharma et al. presented a paper in 2020, discussing the need for automating the classification of plant leaf diseases in agriculture. They explored the integration of image preprocessing with machine learning for disease classification and highlighted the potential impact on agricultural practices and future research goals.

References	Method	Plant	Accuracy
[12]	K-means Segmentation and Deep Learning Networks	Potato	97%
[09]	Deep Learning	Potato	95%
[08]	CNN	Potato	91%
[07]	CNN	Potato	93%
[06]	Multidimensional Fusion Atrous-CNN	Potato	89%
[05]	Deep Learning	Potato	94%
[01]	Hybrid Deep Learning	Potato	95%
This Work	Proposed CNN	Potato	99.37%

Table 1.1: Comparison of proposed and pre- trained model accuracies {Only for Potato}

To compare our proposed model with different Machine Learning Models, we down-sampled the dataset from 20.6k to 3k {2986} Potato plant instances. This enables us to evaluate and select the most efficient model for higher accuracy. We tested the new dataset with CNN, SVM, and KNN classifiers. Furthermore, we developed a hybrid model that combines CNN and SVM to showcase SVM's effectiveness in classifying a small dataset, supported by the robust CNN for larger datasets.

Method	Dataset Used	Size	Accuracy
CNN (Our model)	PlantVillage	2986	99.37%
SVM	PlantVillage	2000	55%
CNN	PlantVillage	2000	48%
KNN	PlantVillage	2000	9%
CNN+SVM (Hybrid)	PlantVillage	2000	57%

Table 1.2: Performance Comparison with ML algorithms {Only for Potato}

Our suggested model consists of 5 convolutional, 2 fully-connected and 3 max pooling layers. To introduce nonlinearity and resolve the vanishing gradients issue, activation function called ReLU is applied at each activation layer. At the output layer, softmax function is used. This classifies diseases by normalising the network's output into a distribution of probabilities across the expected output classes. Four dropout layers are used to cut off a few randomly selected neurons to prevent overfitting.

We have compared our proposed model to various Machine Learning Models by down- sampling our dataset (from 20.6k to 3k). This would help in choosing the better model and obtain higher accuracy as the efficiency differs from one dataset to the other. The new dataset was evaluated against CNN, SVM and KNN classifiers.

Additionally, we have devised a hybrid model that amalgamates CNN and the SVM algorithm to evaluate the precision of our dataset. This was done to prove the effectiveness of SVM in classifying a small dataset, with added support by the powerful CNN for larger dataset.

II. IMPLEMENTATION OF THE POTATO PLANT DISEASE DETECTION SYSTEM USING CNN

A: Dataset and Pre-Processing

The PlantVillage dataset is chosen for the project. It contains 20,639 RGB images of leaves belonging to 3 different plants: Bell Pepper, Potato and Tomato. They are divided into 15 classes, each depicting a disease or healthy plant. The PlantVillage dataset has served as a foundational resource in numerous research endeavors. Inclusivity of a diverse range of leaf images in the dataset is pivotal, as it enables the model to assimilate crucial variations throughout the training process. This inclusivity significantly enhances the capacity of the deep CNN model to generalize

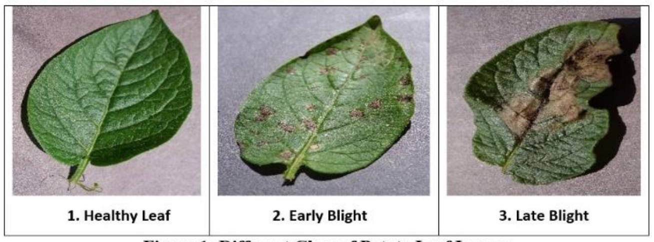


Figure 1: Different Class of Potato Leaf Images

effectively.

Fig 2.1 Random leaf images from PlantVillage dataset

The process in which image variations are created artificially is called image augmentation. To alter the images, transformations like shifting, shearing, scaling, zooming, and flipping are applied.

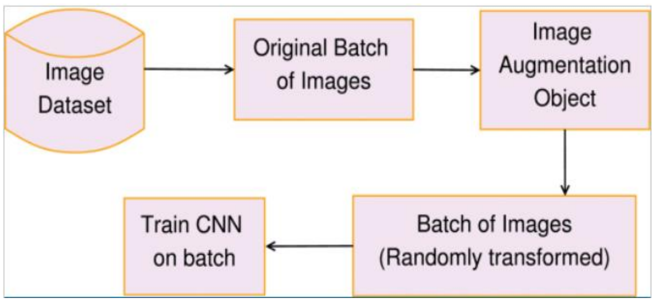


Image Dataset
Original Batch of Images
Image Augmentation Object
Batch of Images (Randomly transformed)
Train CNN on batch

Fig 2.2 Data Augmentation Process

By introducing small variations in the images, these transformations encourage diversity in the training set. Consequently, this lessens overfitting and enhances the model's ability to generalize.

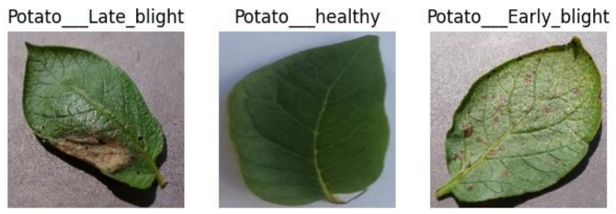


Fig 2.3 Examples of enhanced PlantVillage dataset leaves

B. CNN Model Architecture

A Deep Convolutional Neural Network, at its core, is an Artificial Neural Network (ANN) with deep learning and a feed-forward structure. This network comprises key components like the fully connected layer, pooling layer, and convolutional layer with non-linear activation units. What sets Deep CNN apart is its ability to dispense with the need for extensive feature engineering, making it advantageous compared to traditional machine learning methods. This approach has demonstrated its effectiveness across various domains, including precision agriculture, natural language processing (NLP), and tasks like text and image categorization. In the initial layer of a deep CNN, the primary task involves extracting numerous lower-level features.

1) Convolutional Layer

The convolutional layer plays a crucial role when applying a filter (kernel) to process a source image. This operation involves identifying local patterns within the previous layers, resulting in the creation of feature maps. Within the convolutional layer, there are two key elements: a linear convolutional operation and a non-linear activation unit.

When working with multi-channel images like RGB images, convolution is applied across multiple images simultaneously. In mathematical terms, this process can be expressed as:

$$\textbf{Conv (I, K)}_{x,y} = \sum_{i=1}^{nH}\sum_{j=1}^{nW}\sum_{k=1}^{nC} \textbf{K}_{i,j,k}\, \textbf{I}_{x+i-1,\, y+j-1,\, k}$$

In this context, the kernel is denoted as K(fh, fw, nC), and it undergoes convolution with an image represented as I(nH, nW, nC). While the image and kernel can have different dimensions, they share the same number of channels, nC. The image has width and height represented as nW and nH, and the kernel's dimensions are fh and fw. It's a common practice to treat the kernel as a window with odd dimensions, where both fh and fw are equal to a common value referred to as "f." This common value, "f," determines the size of the resulting feature map:

$$\textbf{Feature\_map}\ (o_H, o_W, z) = (\lfloor n_H + 2P - f/s + 1 \rfloor, \lfloor n_W + 2P - f/s + 1 \rfloor, z)$$

In the provided equation, the symbol "z" corresponds to the quantity of kernels applied in convolution with the input image. "s" denotes the stride, while "P" signifies the padding value. Rectified Linear Units (ReLU) stand out as a commonly employed activation function, responsible for activating neurons when the output of a convolution unit or any other linear transformation attains a value greater than or equal to zero. This can be expressed as follows:

$$f(z) = MAX(0, z)$$

2) *Pooling Layer*

To reduce the extensive number of parameters within activation maps, the pooling layer comes into play, effectively down sampling the feature maps produced by the convolutional layers. This approach aims to alleviate the computational burden, expedite the training process, and aid in mitigating overfitting. Among the various pooling techniques, max pooling stands as the most widely adopted, characterized by the mathematical expression provided below:

$$\text{Max\_Pooling: } y_j = \text{max\_i} \in R_j (P_i)$$

Max pooling involves selecting the highest value from each input patch. The dimensions of the resulting feature map can be determined using the following equation, where "R" signifies a receptive field composed of "P" pixels:

$$\text{Feature\_map } (o_H, o_W, z) = (\lfloor n_H + 2P - f/s + 1 \rfloor, \lfloor n_W + 2P - f/s + 1 \rfloor, n_C)$$

It's important to note that the pooling process only alters the dimensions n_H and n_W, while n_C remains unchanged.

3) *Fully Connected Layer (FC)*

Similar to conventional neural networks, convolutional neural networks (CNNs) also integrate fully connected (FC) or dense layers, usually positioned in the latter stages of the CNN. Output layers are established with a predetermined number of outputs. To accommodate these layers for 1-dimensional data, a flattened layer is utilized. This layer converts the 2-dimensional output from the preceding layers into a 1-dimensional representation. The fully connected layers apply a combination of linear and non-linear transformations. The description of these transformations is provided as follows:

$$z = W^T \cdot X + b$$
$$O = f(z)$$

The input feature map is denoted as "X," the weight as "W," the bias terms as "b," and the output of the fully connected layer as "O."

C. Model Training

During the model training phase, the CNN model's parameters must be optimized to minimize the error between the predicted and actual disease labels. Crucial components of this phase include:

- Loss Function: The model's training procedure must be guided by the choice of loss function. Especially for problems requiring numerous classes, categorical cross-entropy is a popular loss function in plant disease diagnosis.

- Hyperparameter Tuning: A few examples of hyperparameters that significantly affect a training process' efficacy include batch size, learning rate, and number of epochs to be trained. Finding the optimal collection of hyperparameters to provide the highest possible model performance is known as hyperparameter tuning.

- Regularization: Regularization techniques, such as dropout and L2 regularization, are employed in deep neural network training to mitigate the prevalent issue of overfitting.

- Early Stopping: Early stopping is a strategy to stop training a model when it starts to perform poorly on the validation set, suggesting that overfitting may be an issue.

- Model Checkpoints: If training is interrupted, the optimal model parameters are preserved, enabling the recovery of the top-performing model.

D. Model Evaluation

The model undergoes a comprehensive evaluation to determine its performance, which comprises several key components:

- **Validation Set Evaluation:** The model's performance is assessed using a validation set. These results are vital for gauging the model's predictive accuracy with new and unseen data and for making adjustments to hyperparameters.

- **Evaluation Metrics:** The model's effectiveness is measured using various evaluation metrics, including F1-Score, Precision, Confusion Matrix, and Accuracy.

- **Testing Set Evaluation:** The testing set serves as the ultimate assessment of the model's performance and efficiency. It provides an objective evaluation of the model's accuracy in identifying diseases in crops.

E. Real World Deployment

To make plant disease detection accessible and practical for farmers and agricultural stakeholders, the model needs to be deployed in the real world. It includes:

- **Real-time processing**: In scenarios where real-time disease detection is required, the system can be integrated into IoT devices or edge computing solutions for on-site analysis. This is particularly beneficial for early disease detection in the field.

- **Batch Processing**: In cases where real-time processing is not essential, batch processing can be employed. Collected plant images are periodically analyzed in bulk, and disease classification results are reported.

III. Proposed CNN Model

The integration of deep learning techniques, particularly Convolutional Neural Networks, has brought a significant transformation to plant disease detection. Nonetheless, the challenge of bridging the gap between advanced machine learning methods and their practical use by farmers persists.

To address this challenge, we propose an integrated system that allows farmers to directly input leaf images for disease diagnosis. This system takes the uploaded images, processes them, extracts relevant features, and classifies the images, facilitating swift interventions and minimizing crop losses. The disease detection process initiates with the user uploading a leaf image into the model, which then employs various Image Processing techniques and CNN layers to promptly generate results. These results assist the user in determining whether the plant is healthy or not.

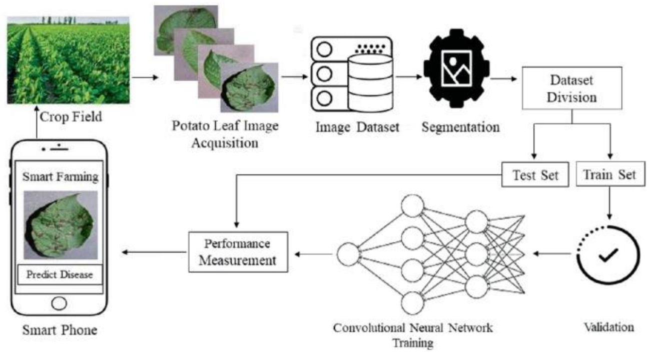


Fig 3.1 System Architecture

The integrated system for disease detection and classification in crops begins with the farmer's input of leaf images. The system is divided into three primary phases:

- Farmer’s Input
- Image Processing
- Training

A. Farmer Input: Capturing Leaf Images

The system initiates when a farmer or user encounters a plant displaying possible disease symptoms. To initiate the disease detection process, the farmer captures a digital image of the affected plant's leaf using a smartphone or another digital device that is equipped with a camera, including IoT devices. These captured images are then submitted to the plant disease detection system, often through a user-friendly application or platform. This accessibility ensures that farmers can utilize the system without requiring specialized technical expertise.

B. Preprocessing: Enhancing Image Quality

Upon receiving the input from the farmer, the image processing phase is initiated. This phase comprises three vital steps: Preprocessing, Feature Extraction, and Classification, all of which are fundamental for ensuring precise disease detection.

- **Image Denoising:** To enhance the quality of the input image, any noise or artifacts, including sensor noise or lighting variations, are eliminated using denoising filters or techniques. This step ensures a cleaner and more reliable input image.

- **Image Enhancement:** To improve the visibility of disease symptoms, adjustments are made to the image's contrast, brightness, and colour balance. Commonly employed techniques for this purpose include histogram equalization and contrast stretching.

- **Image Resizing:** Images can be resized to achieve a consistent resolution, ensuring uniformity in the data input to the system. Resizing aids in standardizing the image dimensions for consistent processing.

C. Feature Extraction: Identifying Disease- Related Patterns

After preprocessing, the system moves to feature extraction, a critical phase for understanding the image's content. Features are distinctive patterns or characteristics within the image that are relevant to identifying plant diseases. Feature extraction can encompass various techniques:

- **Colour-Based Features:** Colour features are extracted to determine variations in plant-leaf colour. It includes methods such as colour moments and colour histograms.

- **CNN Feature Extraction:** In modern plant disease detection systems, deep learning techniques, specifically CNNs, have demonstrated remarkable capability for automatic feature extraction. CNN layers automatically learn relevant features through convolution and pooling operations. Transfer learning with pre-trained CNN models can be employed for this purpose.

D. Classification: Disease Identification

The features extracted from the images serve as input for a classification model, which is responsible for identifying the disease type based on these features. This stage heavily relies on CNNs:

- **CNN Architecture**: The CNN architecture encompasses multiple convolutional and pooling layers, enabling it to capture hierarchical features. This architecture can be a custom design or a pre-trained model like VGG, Inception, or ResNet, fine-tuned for the specific plant disease classification task.

- **Training**: The CNN model undergoes training on a dataset containing labelled images of both healthy and diseased plants, encompassing early and late blight instances. During the training process, model parameters are fine-tuned using techniques such as backpropagation and gradient descent to minimize classification errors.

- **Classification Output**: The last layer of the CNN model generates a probability distribution across potential disease classes. The predicted disease type belongs to the class with the highest probability. The results can be presented as "healthy" or specify the exact disease type, such as early blight or late blight.

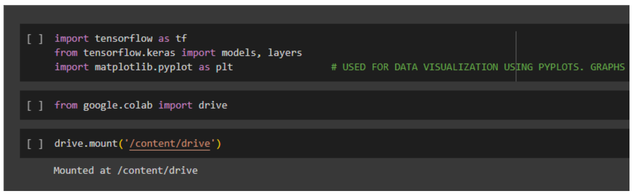
IV. IMPLEMENTATION

A: Proposed Recommender Library Modules

1. *NUMPY*
2. *SKLEARN*
3. *KERAS*
4. *TENSORFLOW*
5. *MATPLOTLIB*

B: Importing Dependencies/ Libraries

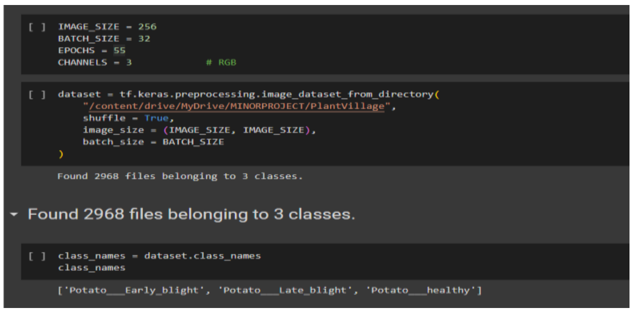
Importing TensorFlow
Importing Keras
Importing Matplotlib
Connecting Google Colab with Google Drive
Mounting the drive in the Google Colab



```
[ ] import tensorflow as tf
    from tensorflow.keras import models, layers
    import matplotlib.pyplot as plt                    # USED FOR DATA VISUALIZATION USING PYPLOTS. GRAPHS

[ ] from google.colab import drive

[ ] drive.mount('/content/drive')

    Mounted at /content/drive
```

C: Preprocessing Image Dataset

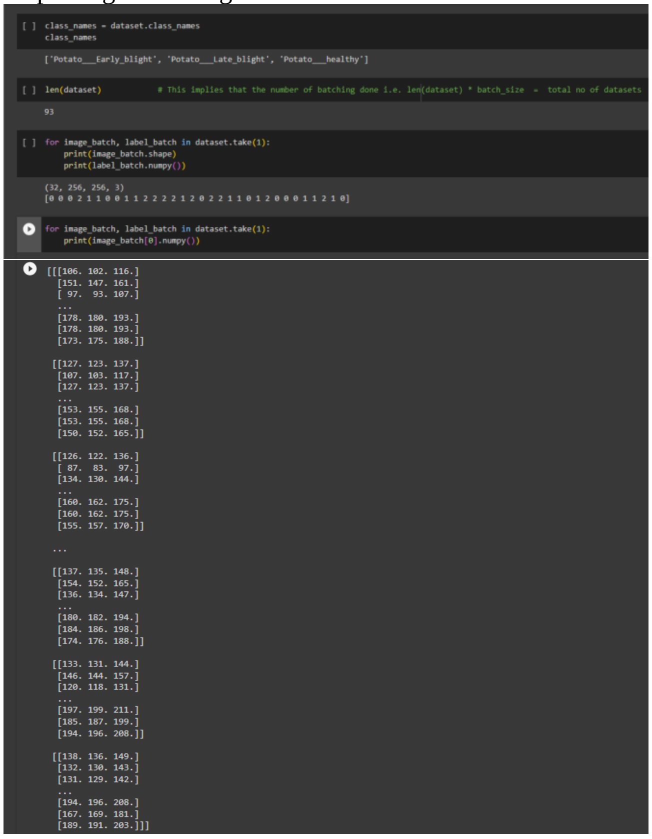
Constants and Variables Setup
Loading Image Dataset
Class Names Extraction
Class Names Output



```
[ ] IMAGE_SIZE = 256
    BATCH_SIZE = 32
    EPOCHS = 55
    CHANNELS = 3            # RGB

[ ] dataset = tf.keras.preprocessing.image_dataset_from_directory(
        "/content/drive/MyDrive/MINORPROJECT/PlantVillage",
        shuffle = True,
        image_size = (IMAGE_SIZE, IMAGE_SIZE),
        batch_size = BATCH_SIZE
    )

    Found 2968 files belonging to 3 classes.

▾ Found 2968 files belonging to 3 classes.

[ ] class_names = dataset.class_names
    class_names

    ['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']
```

D: Extracting information about Dataset

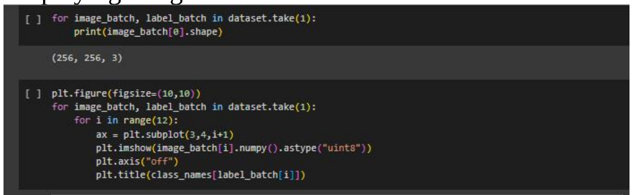
Batch Count
Inspecting Image Batch and Labels
Image Batch Shape
Label Batch Values
Inspecting First Image



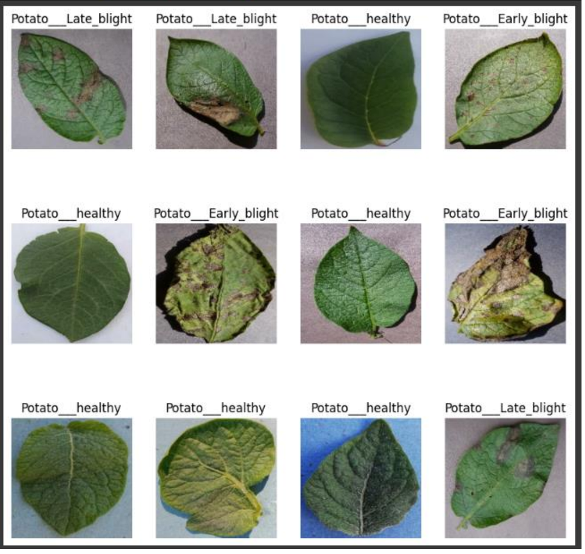
```
[ ] class_names = dataset.class_names
    class_names

    ['Potato___Early_blight', 'Potato___Late_blight', 'Potato___healthy']

[ ] len(dataset)          # This implies that the number of batching done i.e. len(dataset) * batch_size  =  total no of datasets

    93

[ ] for image_batch, label_batch in dataset.take(1):
        print(image_batch.shape)
        print(label_batch.numpy())

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

    for image_batch, label_batch in dataset.take(1):
        print(image_batch[0].numpy())

    [[[106. 102. 116.]
      [151. 147. 161.]
      [ 97.  93. 107.]
      ...
      [178. 180. 193.]
      [178. 180. 193.]
      [173. 175. 188.]]

     [[127. 123. 137.]
      [107. 103. 117.]
      [127. 123. 137.]
      ...
      [153. 155. 168.]
      [153. 155. 168.]
      [150. 152. 165.]]

     [[126. 122. 136.]
      [ 87.  83.  97.]
      [134. 130. 144.]
      ...
      [160. 162. 175.]
      [160. 162. 175.]
      [155. 157. 170.]]

     ...

     [[137. 135. 148.]
      [154. 152. 165.]
      [136. 134. 147.]
      ...
      [180. 182. 194.]
      [184. 186. 198.]
      [174. 176. 188.]]

     [[133. 131. 144.]
      [146. 144. 157.]
      [120. 118. 131.]
      ...
      [197. 199. 211.]
      [185. 187. 199.]
      [194. 196. 208.]]

     [[138. 136. 149.]
      [132. 130. 143.]
      [131. 129. 142.]
      ...
      [194. 196. 208.]
      [167. 169. 181.]
      [189. 191. 203.]]]
```

E: Visualization of subset of Images

Image Batch Shape Inspection
Creating a visualization Grid
Iterating through Images
Displaying Images with Labels

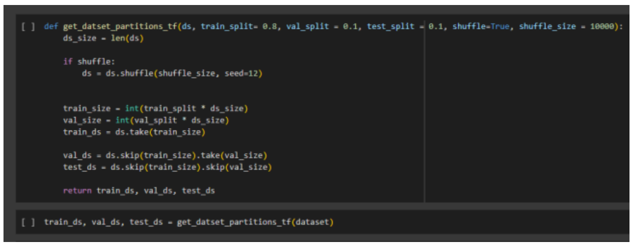


```
[ ] for image_batch, label_batch in dataset.take(1):
        print(image_batch[0].shape)

    (256, 256, 3)

[ ] plt.figure(figsize=(10,10))
    for image_batch, label_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.axis("off")
            plt.title(class_names[label_batch[i]])
```



F: Partitioning of dataset into Train, Test, Validation

Get Dataset Partitions TF
Dataset size Calculation
Shuffling Enabled 1000
Partitioning the Dataset
Return Values for Train, Test and Val Dataset

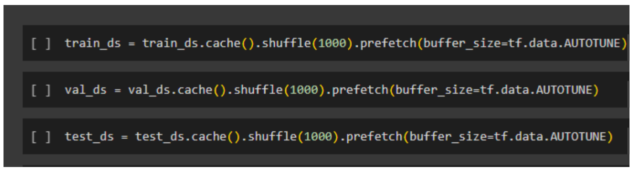


```
[ ] def get_datset_partitions_tf(ds, train_split= 0.8, val_split = 0.1, test_split = 0.1, shuffle=True, shuffle_size = 10000):
        ds_size = len(ds)

        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_datset_partitions_tf(dataset)
```

G: Operations on Train, Test and Val Datasets

Caching
Shuffling
Prefetching



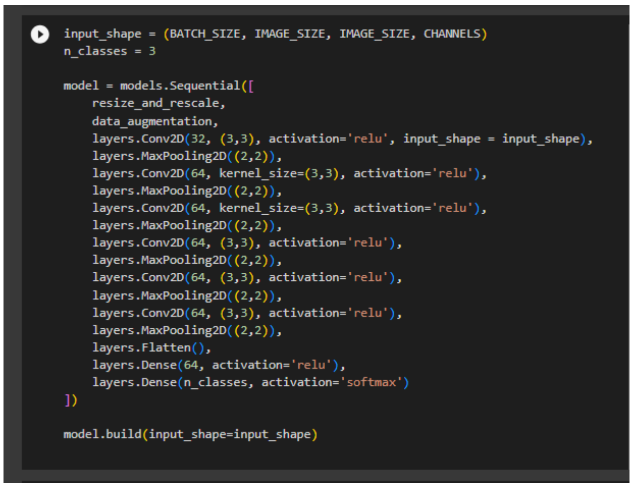
```
[ ] train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=tf.data.AUTOTUNE)
```

```
[ ] 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)
```

H: Defining CNN model using TensorFlow's Keras API

Model Definition
Input shape and number of Classes
Layer Stack
Flatten and Fully Connected Layers
Model Compilation

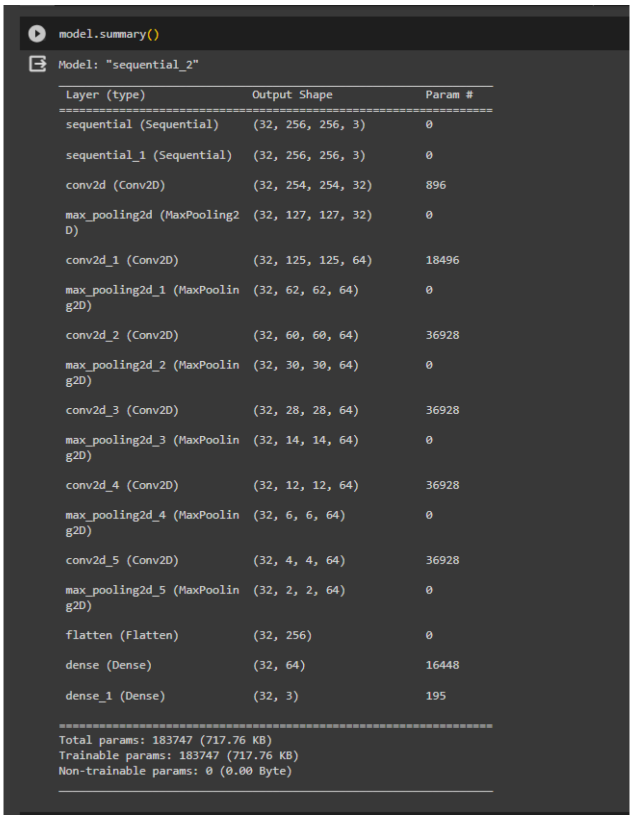


```
input_shape = (BATCH_SIZE, IMAGE_SIZE, IMAGE_SIZE, CHANNELS)
n_classes = 3

model = models.Sequential([
    resize_and_rescale,
    data_augmentation,
    layers.Conv2D(32, (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)
```

I: Model Summary

Layer Information
- Layer Type
- Output Shape
- Param

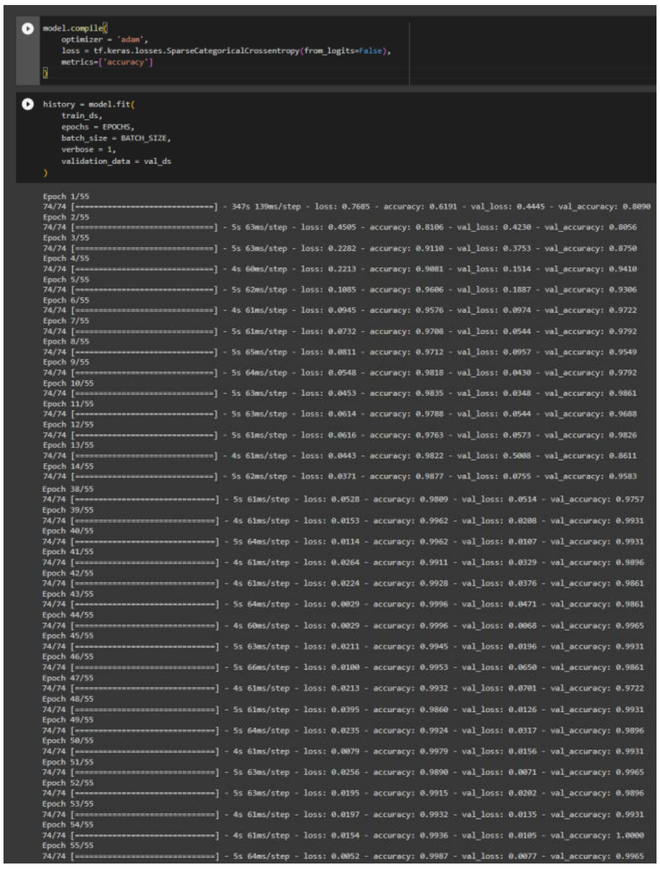
Total Parameters
Trainable vs Non-Trainable Parameters



```
model.summary()

Model: "sequential_2"
 _________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (32, 256, 256, 3)         0

 sequential_1 (Sequential)   (32, 256, 256, 3)         0

 conv2d (Conv2D)             (32, 254, 254, 32)        896

 max_pooling2d (MaxPooling2  (32, 127, 127, 32)        0
 D)

 conv2d_1 (Conv2D)           (32, 125, 125, 64)        18496

 max_pooling2d_1 (MaxPoolin  (32, 62, 62, 64)          0
 g2D)

 conv2d_2 (Conv2D)           (32, 60, 60, 64)          36928

 max_pooling2d_2 (MaxPoolin  (32, 30, 30, 64)          0
 g2D)

 conv2d_3 (Conv2D)           (32, 28, 28, 64)          36928

 max_pooling2d_3 (MaxPoolin  (32, 14, 14, 64)          0
 g2D)

 conv2d_4 (Conv2D)           (32, 12, 12, 64)          36928

 max_pooling2d_4 (MaxPoolin  (32, 6, 6, 64)            0
 g2D)

 conv2d_5 (Conv2D)           (32, 4, 4, 64)            36928

 max_pooling2d_5 (MaxPoolin  (32, 2, 2, 64)            0
 g2D)

 flatten (Flatten)           (32, 256)                 0

 dense (Dense)               (32, 64)                  16448

 dense_1 (Dense)             (32, 3)                   195

=================================================================
Total params: 183747 (717.76 KB)
Trainable params: 183747 (717.76 KB)
Non-trainable params: 0 (0.00 Byte)
 _________________________________________________________________
```

J: Compile and Train Neural Network

Model Compilation
Model Training 'model.fit()'
Model Training Process
Training History



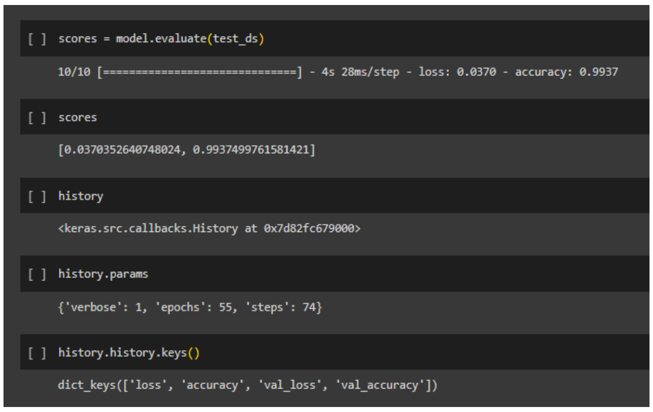
```
model.compile(
    optimizer = 'adam',
    loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy']
)
```

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

```
Epoch 1/55
74/74 [==============================] - 347s 139ms/step - loss: 0.7685 - accuracy: 0.6191 - val_loss: 0.4445 - val_accuracy: 0.8090
Epoch 2/55
74/74 [==============================] - 5s 63ms/step - loss: 0.4505 - accuracy: 0.8106 - val_loss: 0.4230 - val_accuracy: 0.8056
Epoch 3/55
74/74 [==============================] - 5s 63ms/step - loss: 0.2282 - accuracy: 0.9110 - val_loss: 0.3753 - val_accuracy: 0.8750
Epoch 4/55
74/74 [==============================] - 4s 60ms/step - loss: 0.2213 - accuracy: 0.9081 - val_loss: 0.1514 - val_accuracy: 0.9410
Epoch 5/55
74/74 [==============================] - 5s 62ms/step - loss: 0.1085 - accuracy: 0.9606 - val_loss: 0.1887 - val_accuracy: 0.9306
Epoch 6/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0945 - accuracy: 0.9576 - val_loss: 0.0974 - val_accuracy: 0.9722
Epoch 7/55
74/74 [==============================] - 5s 61ms/step - loss: 0.0732 - accuracy: 0.9708 - val_loss: 0.0544 - val_accuracy: 0.9792
Epoch 8/55
74/74 [==============================] - 5s 65ms/step - loss: 0.0811 - accuracy: 0.9712 - val_loss: 0.0957 - val_accuracy: 0.9549
Epoch 9/55
74/74 [==============================] - 5s 64ms/step - loss: 0.0548 - accuracy: 0.9818 - val_loss: 0.0430 - val_accuracy: 0.9792
Epoch 10/55
74/74 [==============================] - 5s 63ms/step - loss: 0.0453 - accuracy: 0.9835 - val_loss: 0.0348 - val_accuracy: 0.9861
Epoch 11/55
74/74 [==============================] - 5s 63ms/step - loss: 0.0614 - accuracy: 0.9788 - val_loss: 0.0544 - val_accuracy: 0.9688
Epoch 12/55
74/74 [==============================] - 5s 61ms/step - loss: 0.0616 - accuracy: 0.9763 - val_loss: 0.0573 - val_accuracy: 0.9826
Epoch 13/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0443 - accuracy: 0.9822 - val_loss: 0.5008 - val_accuracy: 0.8611
Epoch 14/55
74/74 [==============================] - 5s 62ms/step - loss: 0.0371 - accuracy: 0.9877 - val_loss: 0.0755 - val_accuracy: 0.9583
Epoch 38/55
74/74 [==============================] - 5s 61ms/step - loss: 0.0528 - accuracy: 0.9809 - val_loss: 0.0514 - val_accuracy: 0.9757
Epoch 39/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0153 - accuracy: 0.9962 - val_loss: 0.0208 - val_accuracy: 0.9931
Epoch 40/55
74/74 [==============================] - 5s 64ms/step - loss: 0.0114 - accuracy: 0.9962 - val_loss: 0.0107 - val_accuracy: 0.9931
Epoch 41/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0264 - accuracy: 0.9911 - val_loss: 0.0329 - val_accuracy: 0.9896
Epoch 42/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0224 - accuracy: 0.9928 - val_loss: 0.0376 - val_accuracy: 0.9861
Epoch 43/55
74/74 [==============================] - 5s 64ms/step - loss: 0.0029 - accuracy: 0.9996 - val_loss: 0.0471 - val_accuracy: 0.9861
Epoch 44/55
74/74 [==============================] - 4s 60ms/step - loss: 0.0029 - accuracy: 0.9996 - val_loss: 0.0068 - val_accuracy: 0.9965
Epoch 45/55
74/74 [==============================] - 5s 63ms/step - loss: 0.0211 - accuracy: 0.9945 - val_loss: 0.0196 - val_accuracy: 0.9931
Epoch 46/55
74/74 [==============================] - 5s 66ms/step - loss: 0.0100 - accuracy: 0.9953 - val_loss: 0.0650 - val_accuracy: 0.9861
Epoch 47/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0213 - accuracy: 0.9932 - val_loss: 0.0701 - val_accuracy: 0.9722
Epoch 48/55
74/74 [==============================] - 5s 61ms/step - loss: 0.0395 - accuracy: 0.9860 - val_loss: 0.0126 - val_accuracy: 0.9931
Epoch 49/55
74/74 [==============================] - 5s 64ms/step - loss: 0.0235 - accuracy: 0.9924 - val_loss: 0.0317 - val_accuracy: 0.9896
Epoch 50/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0079 - accuracy: 0.9979 - val_loss: 0.0156 - val_accuracy: 0.9931
Epoch 51/55
74/74 [==============================] - 5s 63ms/step - loss: 0.0256 - accuracy: 0.9890 - val_loss: 0.0071 - val_accuracy: 0.9965
Epoch 52/55
74/74 [==============================] - 5s 63ms/step - loss: 0.0195 - accuracy: 0.9915 - val_loss: 0.0202 - val_accuracy: 0.9896
Epoch 53/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0197 - accuracy: 0.9932 - val_loss: 0.0135 - val_accuracy: 0.9931
Epoch 54/55
74/74 [==============================] - 4s 61ms/step - loss: 0.0154 - accuracy: 0.9936 - val_loss: 0.0105 - val_accuracy: 1.0000
Epoch 55/55
74/74 [==============================] - 5s 64ms/step - loss: 0.0052 - accuracy: 0.9987 - val_loss: 0.0077 - val_accuracy: 0.9965
```

K: Evaluating a Trained ML Model

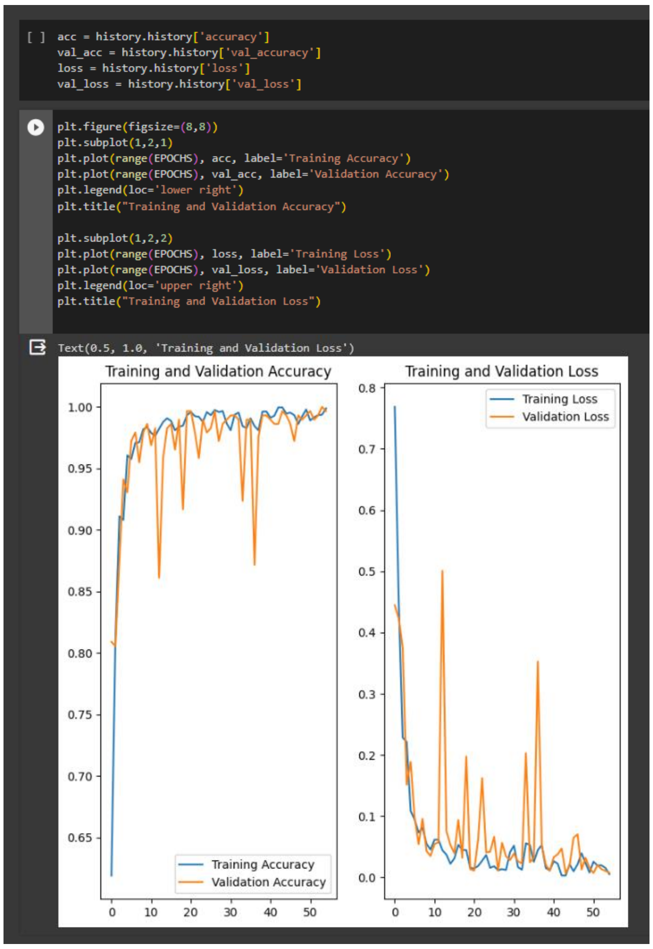
Model Evaluation
Score Variable
History Variable
History Params
Accuracy as 99.37% and Loss as 3.7%



```
[ ] scores = model.evaluate(test_ds)

    10/10 [==============================] - 4s 28ms/step - loss: 0.0370 - accuracy: 0.9937

[ ] scores

    [0.0370352640748024, 0.9937499761581421]

[ ] history

    <keras.src.callbacks.History at 0x7d82fc679000>

[ ] history.params

    {'verbose': 1, 'epochs': 55, 'steps': 74}

[ ] history.history.keys()

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

L: Data Visualization/ Analysis

Data Retrieval
Creating Subplots
Plotting Accuracy Curves
Plotting Loss Curves

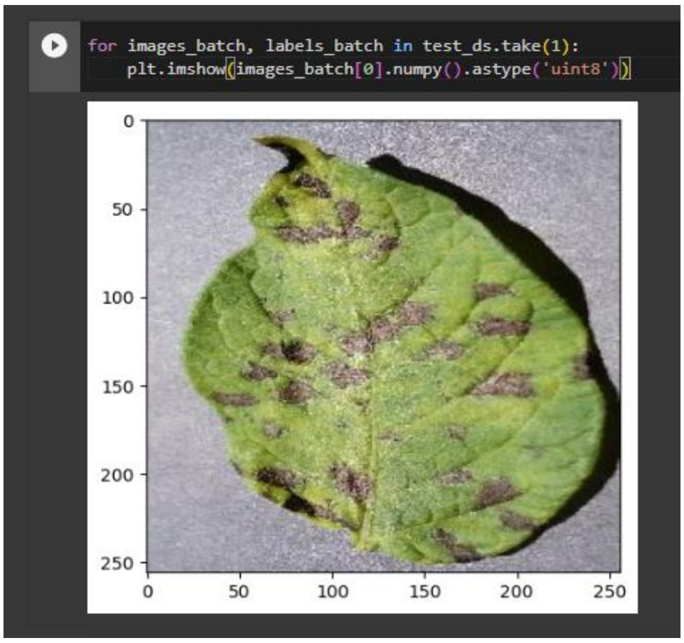


```
[ ] acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    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")

    Text(0.5, 1.0, 'Training and Validation Loss')
```

Training and Validation Accuracy track model performance during training and generalization to new data. Training Loss gradually decreases with more data, while Validation Loss fluctuates due to unseen data evaluation.

M: Visual Inspection of 1st Image

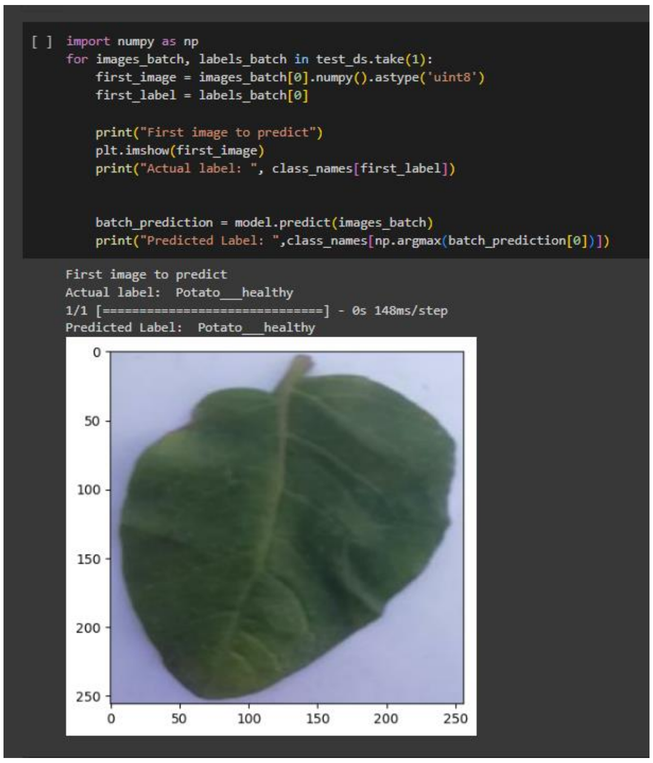
Data Extraction
Image Display
Accessing an image
Image Rendering
- plt.imshow()



```
for images_batch, labels_batch in test_ds.take(1):
    plt.imshow(images_batch[0].numpy().astype('uint8'))
```

N: Data Visualization/ Analysis

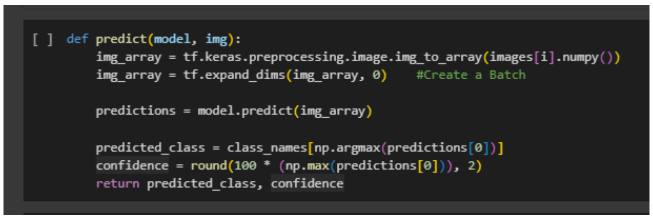
Batch Selection
Image and Label Extraction
Display Actual Label
Model Prediction
Display Predicted Label



```
[ ] 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]

        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___healthy
    1/1 [==============================] - 0s 148ms/step
    Predicted Label:  Potato___healthy
```

O: Prediction of Class and Confidence Label

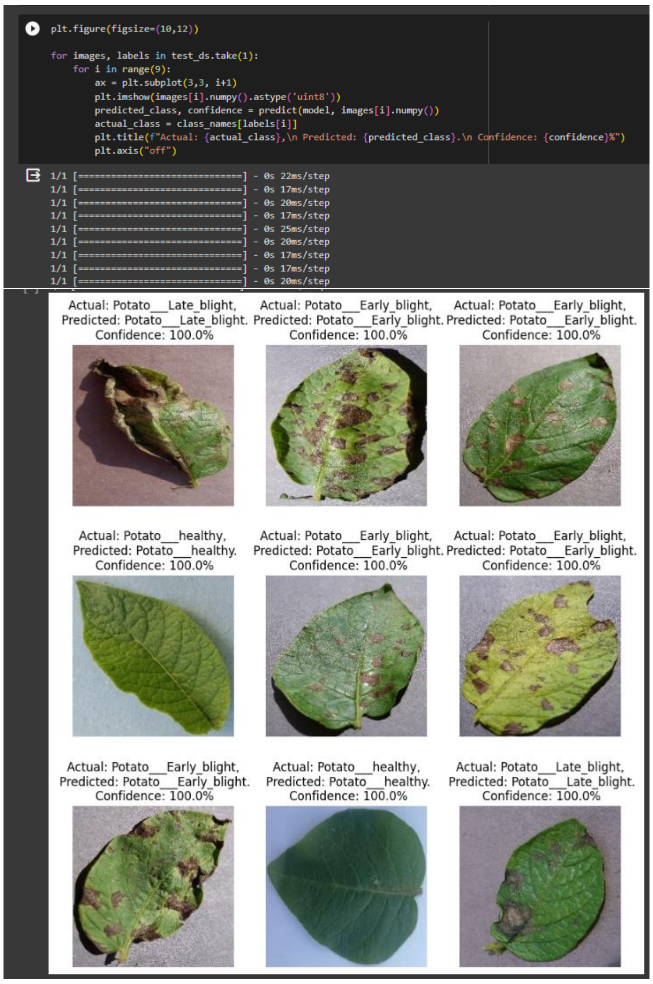
Image Preprocessing
Model Prediction
Class Label and Confidence



```
[ ] def predict(model, img):
        img_array = tf.keras.preprocessing.image.img_to_array(images[i].numpy())
        img_array = tf.expand_dims(img_array, 0)    #Create a Batch

        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
```

P: Visually Inspect set of Images in a grid of subplots

Creating Subplots
Iterating through Test Data
Image Display and Prediction
Titles and Labels
Overall Purpose



```
plt.figure(figsize=(10,12))

for images, labels in test_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3, i+1)
        plt.imshow(images[i].numpy().astype('uint8'))
        predicted_class, confidence = predict(model, images[i].numpy())
        actual_class = class_names[labels[i]]
        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")
        plt.axis("off")
```

```
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 20ms/step
```

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

The graph suggests that Training Accuracy improves steadily because it uses the training data. Validation Accuracy fluctuates as it tests the model on new data. Training Loss decreases gradually, but Validation Loss varies since it evaluates unseen data. More epochs can help reduce loss.

VI. Results and Discussion

The project has reached a successful conclusion, leveraging the PlantVillage dataset to construct a Convolutional Neural Network (CNN) model that achieved an impressive accuracy of 99.37%. Furthermore, a smaller dataset comprising precisely 2,968 images was employed for comparative analysis between deep learning and well-established machine learning algorithms. The project effectively explored and employed various machine learning and deep learning models for the detection and classification of plant diseases.

VII. Conclusion and Future Enhancement

In summary, the utilization of Convolutional Neural Networks (CNN) for plant disease detection represents an innovative strategy to address the substantial challenges confronting the agriculture sector. Conventional manual inspection methods are both inadequate and labor-intensive, exerting a considerable impact on economic sustainability and global food security. The application of CNN technology offers a promising solution to this predicament. These models have exhibited exceptional proficiency in image recognition, rendering them exceptionally well-suited for the intricate task of identifying and categorizing plant diseases. Through the training of CNNs on extensive datasets of plant images, the system can accurately differentiate between healthy and afflicted plants.

The future of plant disease detection employing CNN technology holds a spectrum of exciting prospects. By enhancing model accuracy, broadening coverage across various crops, integrating real-time monitoring, predictive capabilities, and risk assessment, ensuring accessibility, addressing ethical considerations, and fostering collaborative efforts, we can establish a sustainable, efficient, and equitable approach to plant disease management. This technological advancement transcends mere crop yield enhancement; it underscores the transformation of agriculture into a more sustainable and resilient sector, capable of nourishing a burgeoning global population while minimizing its environmental impact. The path ahead is marked by challenges, yet the potential rewards are nothing short of extraordinary.

VIII. References

[1] Fizzah Arshad, Muhammad Mateen, Shaukat Hayat, Maryam Wardah, Zaid Al-Huda, Yeong Hyeon Gu, Mugahed A. Al-antari, PLDPNet: End-to-end hybrid deep learning framework for potato leaf disease prediction, Alexandria Engineering Journal, Volume 78, 2023, Pages 406-418, ISSN 1110-0168, https://doi.org/10.1016/j.aej.2023.07.076.

[2] Sumita Mishra, Rishabh Sachan, Diksha Rajpal, Deep Convolutional Neural Network based Detection System for Real-time Corn Plant Disease Recognition, Procedia Computer Science. Volume 167,2020, Pages 2003-2010, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2020.03.236.

[3] Preeti Baser, Jatinder kumar R. Saini, Ketan Kotecha, TomConv: An Improved CNN Model for Diagnosis of Diseases in Tomato Plant Leaves, Procedia Computer Science, Volume 218, 2023, Pages 1825-1833, ISSN1877 0509, https://doi.org/10.1016/j.procs.2023.01.160.

[4] Dibya Jyoti Bora, Anil Kumar Gupta, Fayaz Ahmad Khan "Comparing the Performance of L*A*B* and HSV Color Spaces with Respect to Color Image Segmentation", arXiv:1506.01472.

[5] Md. Ashraful Islam, Md. Hanif Sikder. A Deep Learning Approach to Classify the Potato Leaf Disease. Journal of Advances in Mathematics and Computer Science, 2022, 37, pp.143 - 155. ff10.9734/jamcs/2022/v37i121735ff. ffhal-04015255f.

[6] Gao, W.; Xiao, Z.; Bao, T. Detection and Identification of Potato-Typical Diseases Based on Multidimensional Fusion Atrous-CNN and Hyperspectral Data. Appl. Sci. 2023, 13, 5023. https://doi.org/10.3390/app13085023.

[7] Varsha P. Gaikwad1, Vijaya Musande. Potato Plant leaf disease detection using CNN Model. Eur. Chem. Bull. 2023,12(1), 516-527

[8] Anushka Bangal, Dhiraj Pagar, Hemant Patil, Neha Pande "Potato Leaf Disease Detection and Classification using CNN" International Journal of Research Publication and Reviews, Vol 3, no 5, pp 1510-1515, May 2022.

[9] Deep Kothari , Harsh Mishra , Vishal Pandey , Mihir Gharat, Rashmi Thakur, 2022, Potato Leaf Disease Detection using Deep Learning, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 11, Issue 11 (November 2022).

[10] Aditi Singh and Harjeet Kaur 2021 IOP Conf. Ser.: Mater. Sci. Eng. 1022 012121

[11] R. Sujatha, Jyotir Moy Chatterjee, NZ Jhanjhi, Sarfraz Nawaz Brohi, Performance of deep learning vs machine learning in plant leaf disease detection, Microprocessors and Microsystems,Volume 80, 2021, 103615, ISSN 0141-9331, https://doi.org/10.1016/j.micpro.2020.103615.

[12] Md. Ashiqur Rahaman Nishad, Meherabin Akter Mitu, Nusrat Jahan, Predicting and Classifying Potato Leaf Disease using K-means Segmentation Techniques and Deep Learning Networks, Procedia Computer Science,Volume 212, 2022, Pages 220-229, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2022.11.006.