Abstract

This project uses cutting-edge machine learning techniques to create a strong Convolutional Neural Network (CNN) model for the detection and classification of plant diseases in tomato crops in response to the serious problems posed by plant diseases, which endanger agricultural productivity and global food security. Using the extensive 20,600 photos of tomato leaves in a variety of health states from the PlantVillage Dataset on Kaggle, our model combines a complex architecture of convolutional layers, activation functions, batch normalization, and pooling layers. This is supplemented by fully connected and softmax layers which transform the network's raw output into class probabilities, enabling the precise identification of disease types. The CNN model was meticulously tuned through multiple iterations to optimize its performance under diverse conditions, accounting for variable lighting and camera settings which often complicate disease diagnosis in practical scenarios. By automating the disease detection process, the model promises to significantly enhance the efficiency and effectiveness of disease management strategies in agriculture. This helps to minimize crop damage and loss by lowering the labor and time needed for manual disease inspection and by improving the speed and accuracy of interventions. By providing a scalable solution that may be tailored to various crops and geographical areas, the creation and application of such a model mark a substantial advancement in agricultural technology and increase resilience to the mounting challenges posed by climate change and the pressures of a growing world food demand.

TABLE OF CONTENTS

1. Introduction:

Plant diseases have long been a significant challenge in agriculture, threatening food security, livelihoods, and ecosystems worldwide. The impact caused by these diseases extends beyond agricultural productivity, affecting economies, public health, and environmental sustainability. Despite advances in agricultural practices and scientific research, plant diseases continue to pose a formidable threat, exacerbated by factors such as globalization, climate change, and evolving pathogen dynamics. The Food and Agriculture Organization (FAO) estimates that plant diseases are responsible for the loss of up to 40% of global crop production annually [1].

Across the globe, plant diseases manifest in various forms, ranging from fungal, bacterial, viral, and nematode infections to abiotic stresses induced by environmental factors like drought, salinity, and pollution. These diseases afflict a vast array of crops, including staple food crops like wheat, rice, maize, and soybeans, as well as cash crops such as coffee, cocoa, and cotton. They affect approximately 20-30% of major food crops, leading to significant yield losses and economic damage [2]. Their impact is particularly pronounced in developing countries, where agriculture forms the backbone of the economy and smallholder farmers rely heavily on crop yields for sustenance and income. It is estimated the world's population is about to reach 9.7 billion by 2050, making food security a paramount. However, these plant diseases can jeopardize this goal by compromising crop yields and availability [3].

Detecting these types of diseases is a tedious and time consuming if done manually. So, there is a need for a automated system to help farmers to identify these disease by continuous monitoring of crops at every growth stage. In recent years, there has been growing interest in leveraging technological innovations, such as remote sensing, machine learning, and molecular diagnostics, for early detection and monitoring of plant diseases. These tools offer the potential to revolutionize disease management by enabling rapid and accurate diagnosis, targeted interventions, and real-time surveillance. The conventional image processing techniques such as thresholding, edge detection, and morphological operations are usually applied to images of plant leaves or tissues to identify the characteristic features associated with disease system. The major problem with these techniques are that under varying light conditions, camera settings are the varying plant phenotypes leading to false positives or negatives. To tackle this, machine learning based image detection techniques are used and also for more accurate detection.

In this project, we developed a CNN model for plant disease detection and classification. Following several trials, the CNN model's parameters were adjusted to improve the disease detection. A sequence of convolutional, activation, batch normalization, and pooling layers were used to build the CNN model. The fully connected and softmax layers come next, which convert the unprocessed network output into useful class probabilities. The CNN model produced runs with fewer parameters by selecting a smaller kernel size and lowering the number of layers.

2. Data Collection:

The dataset we used is obtained from Kaggle, a leading science competition platform for getting dataset, named 'PlantVillage Dataset' [4]. For detecting the disease in tomato leaves, classes have to be specified. Some of the classes with images are explained below (Fig 1.),

a) Bacterial Spot:
Bacterial spots on tomato leaves appear as dark, necrotic lesions encircled by a yellow halo; these diseases frequently result in defoliation and lower-quality fruit. The bacteria that causes the disease, Xanthomonas spp., is often dispersed by water splash or contaminated instruments. It poses a serious danger to tomato production globally [5].

b) Early Blight:
Early tomato leaf blight causes early defoliation and reduced yield. It appears as black, round lesions with concentric rings. Warm, humid weather aggravates the illness, which is brought on by the fungus Alternaria solani [6].

c) Tomato Yellow:
Tomato yellow leaf curl virus (TYLCV) results in upward-curling tomato leaves that are yellowish, inhibited in growth, and produce fewer fruits. It is spread by the whitefly Bemisia tabaci and is quite dangerous for tomato crops, especially in warm, tropical climates [7].

d) Healthy:
Tomato leaves that are in good health have a vivid green hue, a consistent texture, and no blemishes or discoloration. They encourage robust plant development and copious fruit production, which is indicative of healthy physiological processes and the lack of illness or stress [8].

a)

b)

c) d)

Fig. 1. Sample images of some of the tomato plant diseases obtained from the Kaggle Dataset

The entire dataset is approximately 690 MB in total size, comprising 20,600 sample images of various diseases of bell pepper, potato, tomato like bacterial spot, leaf mold, spider mite and also including the sample images of healthy leaves. This dataset has free to use license and hence can be used for any plant disease detection project or any work related to plant disease detection.

3. Methodology:

The purpose of this section is to demonstrate all the steps taken to detect plant disease which is really a crucial concern. To achieve this, we have built a Convolutional Neural Network (CNN) which is designed to understand images and find patterns in them, similar to how our brains process visual information. Here is the work flow diagram which demonstrates the steps used to execute the project. (Fig. 2)

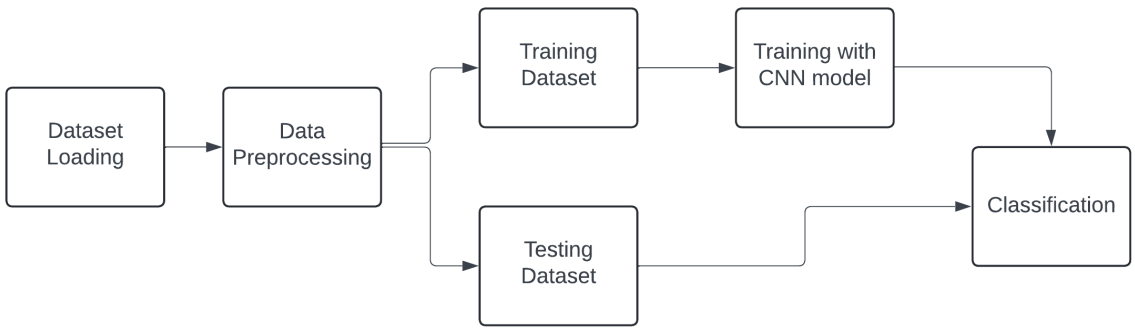

Fig. 2. Project Workflow

3.1 Data Exploration and Preprocessing:

The initial phase involves loading images and their associated labels from a designated directory. Resizing each image to a set size guarantees consistency across the dataset. The images are then stored in a list named images, while their labels reside in a separate list called labels.

Transitioning to NumPy arrays is essential for enhancing the efficiency of various machine learning operations. Converting the image and label lists into NumPy arrays streamlines mathematical computations and enhances compatibility with machine learning frameworks. To gain a visual grasp of the dataset, a function is crafted to exhibit random images alongside their corresponding labels. This function randomly selects images from the dataset and illustrates them, providing insights into the represented content based on their labels.

Prior to segmenting the dataset into training and testing subsets, the indices of the dataset undergo shuffling. This randomization process prevents any inherent order within the data that could potentially bias the training procedure. It's akin to shuffling a deck of cards before dealing them out. Following the index shuffling, the images and labels themselves are shuffled correspondingly. This step upholds the alignment between images and labels while injecting variability into the dataset.

The dataset is then partitioned into a training set and a testing set. The training set serves the purpose of model training, whereas the testing set is utilized to assess the model's performance. This segregation guarantees that the model isn't evaluated on the same data it was trained on, thus averting overfitting. Subsequently, the shapes of the training and testing sets are displayed to confirm the successful division process. Verifying the shapes ensures that the data is distributed as intended for training and testing purposes.

In essence, these procedures collectively ready the dataset for training a CNN model by loading, preprocessing, shuffling, and segmenting the data into suitable subsets. This sequence is fundamental in ensuring effective learning by the model and its ability to generalize effectively to unseen data.

3.2 Model Explanation:

As per the project goals, we aimed to create a Convolutional Neural Network (CNN) model from scratch. However, faced with unresolved issues, we decided to explore using the Keras library as an alternative solution. This approach was adopted to get the results for our project to proceed further.

3.2.1 CNN Model (Without Keras Library):

Implementation of Layers: The classes for each layer type are defined that include Conv2D, MaxPooling2D, BatchNormalization, Dense, Flatten Dropout, ReLU and Softmax. Each layer class implements the forward pass operation.

Forward Pass: The forward method of each class computes the output of the layer. Such as, the Conv2D class performs convolution and applies the activation function and MaxPooling2D performs max pooling. The output feature map of a convolutional layer O can be computed using,

$$O_{i,j,k} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{l=0}^{L-1} I_{i+m,j+n,l} \times F_{m,n,l,k} + b_k$$

Similarly, the output feature map O of a max-pooling layer is computed by taking the maximum value within each pooling window,

$$O_{i,j,k} = \max_{m,n} I_{(i \times \text{stride}[0]+m),(j \times \text{stride}[1]+n),k}$$

Model Architecture: The class CNN Model is used to define the CNN model, which aggregates instances of layer classes as its attributes. The architecture is similar to the CNN model (using Keras library), with convolutional, pooling, batch normalization, dense, and dropout layers. Batch normalization normalizes the activations of a layer by subtracting the mean and dividing by the standard deviation of the batch,

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

Dropout randomly sets a fraction of input units to zero during training to prevent overfitting:

$$\text{output} = \text{input} \times \text{mask}$$

Softmax activation converts the raw scores (logits) into probablitites for each class given by,

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

Training and Testing: The train method of the CNN Model class splits the data into training and testing sets and iterates over epochs. For every epoch, the model computes forward pass outputs, loss, performs backpropagation, and updates parameters with gradient descent.

3.2.2 CNN Model (Using Keras Library):

Convolutional and Pooling Layers: The Conv2D class provided by Keras library is used to define the convolution layers. Each layer specifies the number of filters, kernel size, activation function, strides and padding. The MaxPooling2D layer is used to perform max pooling with specified pool size and strides. BatchNormalization layer is used to apply batch normalization after each convolutional layer.

Dense Layers: The Dense class is used to define the dense layers, specifying the number of units and activation function. The Dropout layer is used to apply dropout regularization after each dense layer to prevent overfitting. Batch normalization is also applied after each dense layer.

Model Architecture: The model is a sequential model that allows layers to be added sequentially. The model architecture consists of alternating convolutional and pooling layers with connected layers. L2 regularization is applied to both kernel and bias terms in convolutional and dense layers having a coefficient of 0.01.

Training and Testing: The train method is used to train the model, which splits the data into training and testing sets and iterates over epochs. For each epoch, the model computes forward pass outputs, loss, performs backpropagation, and updates the parameters using gradient descent.

To sum up, the CNN Model (using Keras library) provides a high-level abstraction for building and training CNN models. It uses inbuilt layers and functionalities provided by Keras, enabling rapid prototyping and experimentation. On the contrary, the CNN Model (without Keras library) provides deeper in-depth control over the CNN model's architecture and operations. It requires defining each layer's forward pass operation and managing parameter updates manually.

For a better understanding, here is the model architecture used for building our CNN model (Fig 2.1)

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 55, 55, 96)	34,944
max_pooling2d (MaxPooling2D)	(None, 27, 27, 96)	0
batch_normalization (BatchNormalization)	(None, 27, 27, 96)	384
conv2d_1 (Conv2D)	(None, 23, 23, 256)	614,656
max_pooling2d_1 (MaxPooling2D)	(None, 11, 11, 256)	0
batch_normalization_1 (BatchNormalization)	(None, 11, 11, 256)	1,024
conv2d_2 (Conv2D)	(None, 9, 9, 384)	885,120
conv2d_3 (Conv2D)	(None, 7, 7, 384)	1,327,488
conv2d_4 (Conv2D)	(None, 5, 5, 256)	884,992
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 256)	0
batch_normalization_2 (BatchNormalization)	(None, 2, 2, 256)	1,024
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 4096)	4,198,400
dropout (Dropout)	(None, 4096)	0
batch_normalization_3 (BatchNormalization)	(None, 4096)	16,384
dense_1 (Dense)	(None, 4096)	16,781,312
dropout_1 (Dropout)	(None, 4096)	0
batch_normalization_4 (BatchNormalization)	(None, 4096)	16,384
dense_2 (Dense)	(None, 1000)	4,097,000
dropout_2 (Dropout)	(None, 1000)	0
batch_normalization_5 (BatchNormalization)	(None, 1000)	4,000
dense_3 (Dense)	(None, 20)	20,020

```
Total params: 28,883,132 (110.18 MB)
```

```
Trainable params: 28,863,532 (110.11 MB)
```

```
Non-trainable params: 19,600 (76.56 KB)
```

Fig. 2.1 Model Architecture

4. Results and Discussion:

4.1 Performance:

The performance of the model can be expressed in terms of accuracy. For accuracy calculation, we can use

$$Accuracy = \frac{TN+TP}{TN+FP+FN+TP}$$

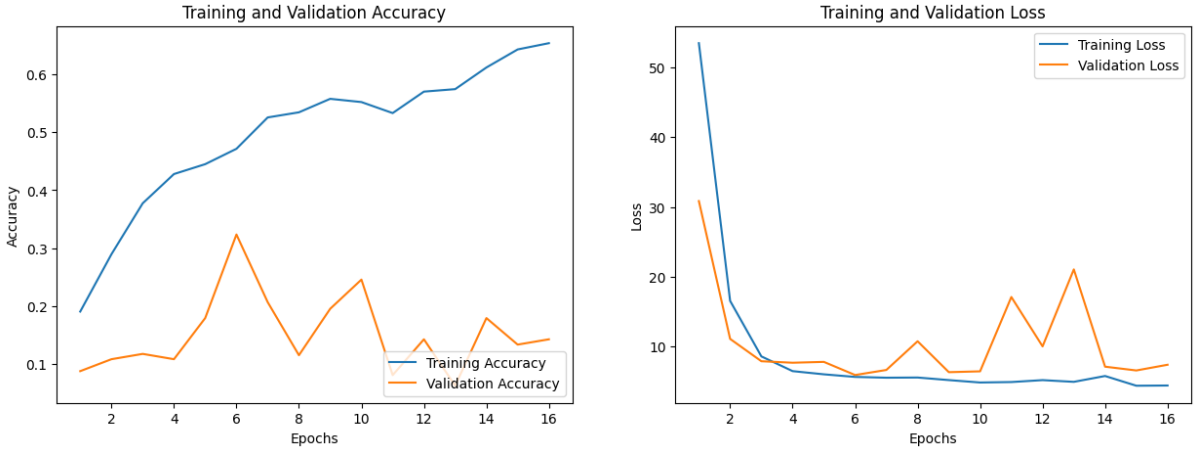

Fig. 3. Training Vs Validation Accuracy and Training vs Validation Loss

a) Training and Validation Accuracy:
The model appears to be learning from the training data as the training accuracy rises with time. More often than not, the Validation Accuracy is less than the Training Accuracy. This may be a sign of overfitting or a hint that the model is not generalizing effectively to fresh, untested data. The disparity implies that further data or regularization strategies would help this model's generalization.

b) Training and Validation Loss:
After a quick initial decline, the Training Loss subsequently declines more slowly. As the model begins to suit the training set, this is to be expected. Instead of the steady declining trend that you would normally hope to see, the Validation Loss exhibits significant variability. The model's predictions were significantly different from the predicted outputs on the validation set at those points, as indicated by the peaks in Validation Loss. Training Loss decreases sharply at the beginning and then continues to decrease at a slower rate. This is expected as the model starts to fit the training data. he Validation Loss, however, shows significant fluctuation and doesn't appear to have a stable decreasing trend, which is typically what you would want to see. The peaks in Validation Loss indicates that at those points, the model's predictions were quite off from the expected outputs on the validation set.

The validation metrics appear to vary a lot, which may be because the validation set is tiny or the model is sensitive to certain samples in the validation set. The general form of the curves indicates that training for more epochs may be helpful as the model may not have fully converged by the 16th epoch. There could be a number of reasons why the Validation Accuracy

is not increasing in tandem with the Training Accuracy, including the requirement for more varied training data, overfitting on the training data, or an overly complicated model.

4.2 Predicted Output:

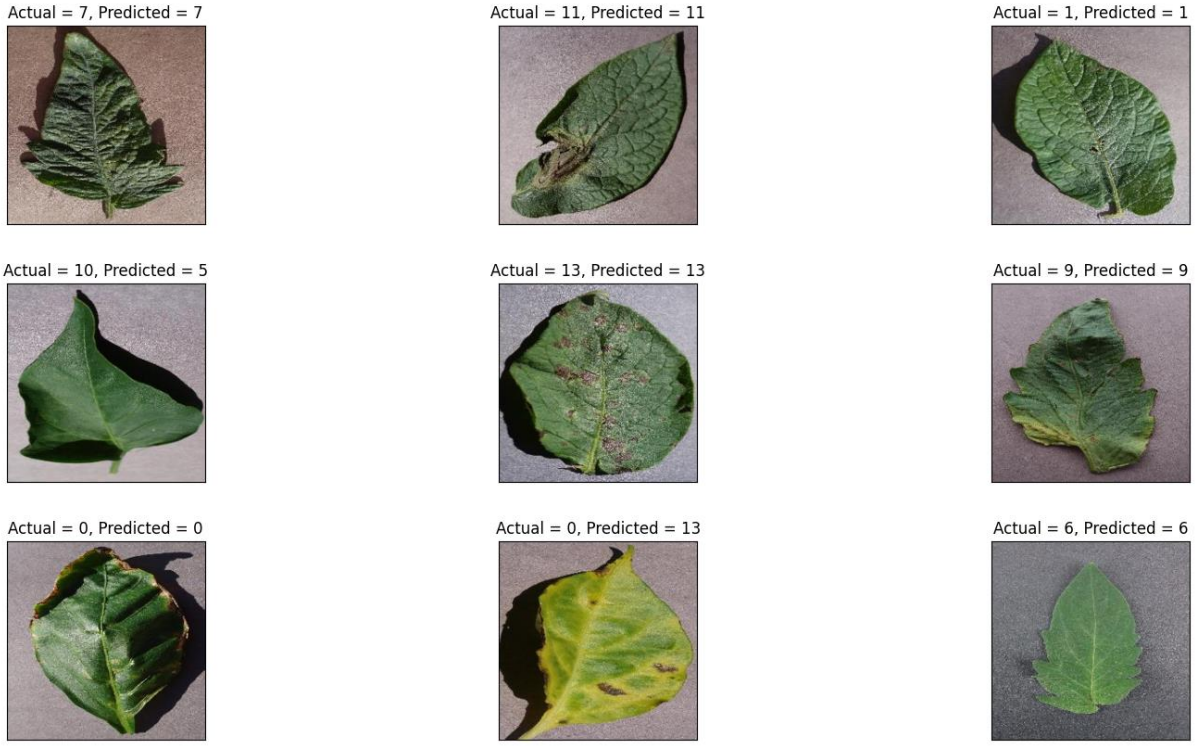

Fig. 3.1 The final output showing actual value of the leaf and the predicted value

From the above figure, it can be observed that the model correctly identified several diseases in the images, matching the actual labels with the predicted ones. This suggests that the features learned during training were representative of the validation data. There are also some instances where the model's prediction do not match the actual labels, where the predicted value of that image should be 0, but instead we got 13 as prediction.

4.3 Confusion Matrix:

Confusion Matrix is a powerful tool for understanding the performance of a classification model. From the confusion matrix of our classification model, we can observe that 'Tomato_Tomato_Mosaic_Virus', 'Potato_Early_blight' has highest numbers on the diagonal and so the model performs well in identifying these types of conditions.

'Tomato_Leaf_Mold' was accurately identified 47 times, while 'Tomato_healthy' and 'Tomato_Tomato_mosaic_virus' recorded 54 and 64 true positives, respectively. These results suggest that the model has effectively learned the distinct features characteristic of these diseases from the training data.

A notable number of false positives were observed for 'Pepper_bacterial_spot', which was incorrectly predicted 14 times as 'Pepper_healthy' and 5 times as 'Tomato_Target_Spot'. This

suggests a visual similarity in the symptoms that the model may be misinterpreting. Similarly, 'Tomato_Early_blight' was frequently misclassified as 'Tomato_Septoria_leaf_spot', indicating 31 instances where the model predicted 'Tomato_Early_blight' when it was actually 'Tomato_Septoria_leaf_spot'. This pattern of misclassification highlights specific challenges the model faces in differentiating between diseases with overlapping visual symptoms.

The confusion matrix also reveals specific instances where the model consistently confused one disease for another. For example, 'Tomato_Spider_mites_Two_spotted_spider_mite' was predicted as 'Tomato_Leaf_Mold' 37 times, reflecting a substantial error in the model's disease recognition capabilities. This consistent error could indicate that the training data for these two diseases share common features that the model is overemphasizing, leading to misinterpretations. Additionally, 'Tomato_Target_Spot' was often confused with 'Tomato_healthy', which could suggest that early or mild symptoms of the disease are not being captured effectively by the model.

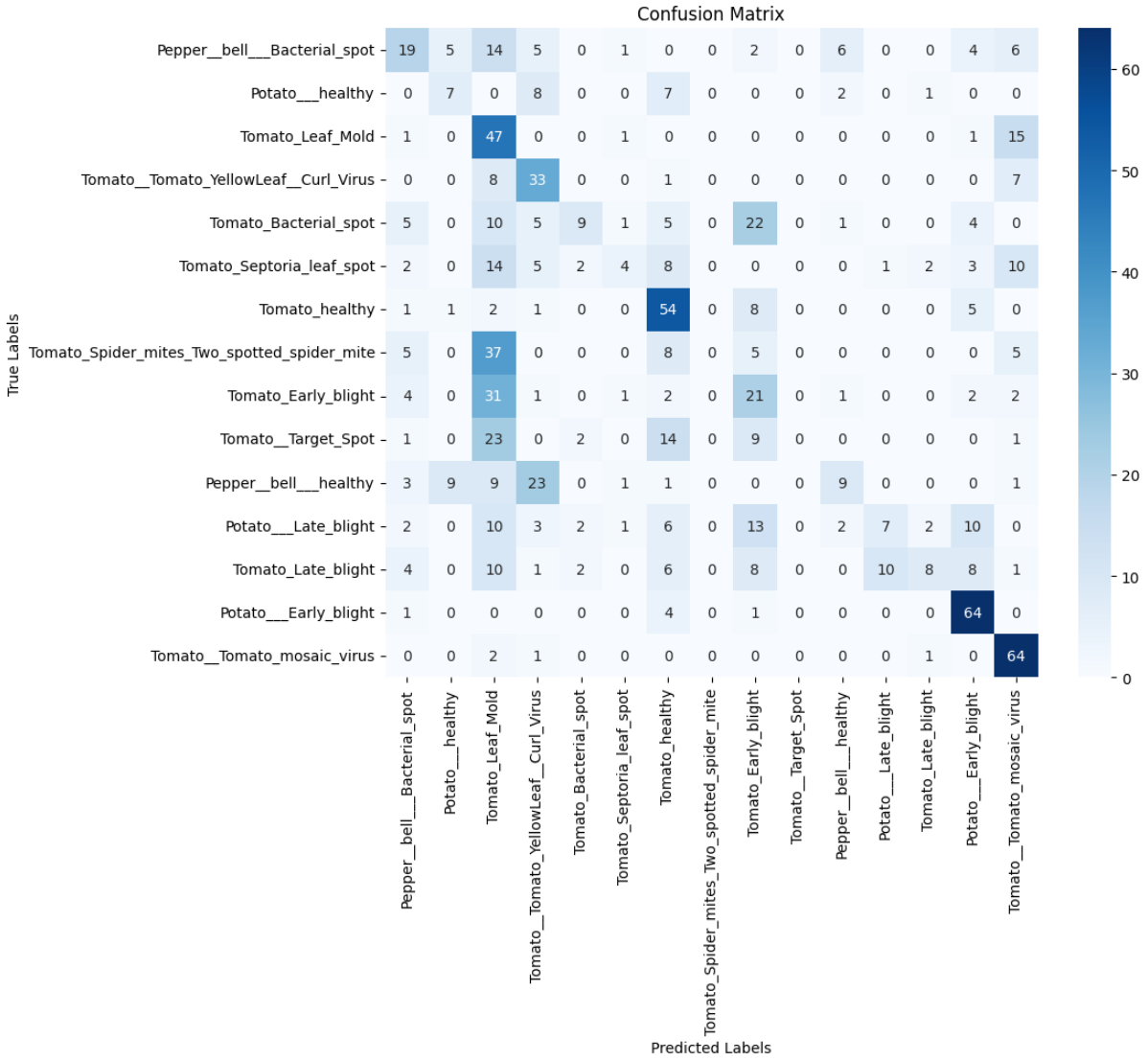

Fig. 3.2 Confusion Matrix – True Labels and Predicted Labels

5. Conclusion:

The CNN model developed in this project demonstrated promising results in the detection and classification of tomato plant diseases. Training on a large and varied dataset allowed the model to achieve a robust understanding of disease characteristics, although some challenges like misclassification and overfitting were encountered. The performance analysis, depicted through accuracy metrics and a confusion matrix, highlighted the model's strengths in recognizing certain diseases with high accuracy while also pointing out areas where improvement is needed. Future work could explore more sophisticated regularization techniques, augmented data to enhance model generalization, and expansion to other plant species. This project illustrates the potential of deep learning in transforming plant disease management, ultimately contributing to more sustainable agricultural practices.

References

1. Food and Agriculture Organization (FAO). (2019). The State of Food Security and Nutrition in the World 2019. Rome: FAO. Retrieved from http://www.fao.org/3/ca5162en/ca5162en.pdf

2. Savary, S., et al. (2019). The global burden of pathogens and pests on major food crops. Nature Ecology & Evolution, 3(3), 430-439. DOI: 10.1038/s41559-018-0793-y

3. Godfray, H. C. J., et al. (2010). Food Security: The Challenge of Feeding 9 Billion People. Science, 327(5967), 812-818. DOI: 10.1126/science.1185383

4. Emma Rex. (2023). Plant Disease Image Dataset. Kaggle. Retrieved from https://www.kaggle.com/datasets/emmarex/plantdisease/data https://github.com/spMohanty/PlantVillage-Dataset/tree/master/raw/color

5. Jones, J. B., et al. (2004). Bacterial Spot of Tomato and Pepper: The Current State of Our Understanding. Plant Health Progress, 10. DOI: 10.1094/PHP-2004-0707-01-RV

6. Daughtrey, M. L., et al. (1997). Early Blight of Tomato and Potato. The Plant Health Instructor. DOI: 10.1094/PHI-I-2000-1022-01

7. Moriones, E., & Navas-Castillo, J. (2000). Tomato Yellow Leaf Curl Virus, an Emerging Virus Complex Causing Epidemics Worldwide. Virus Research, 71(1-2), 123-134. DOI: 10.1016/S0168-1702(00)00216-6

8. Montesinos, E., & Bonaterra, A. (1996). Biological Control of Bacterial Spot of Tomato: Advantages and Limitations of Beneficial Microorganisms and Novel Techniques. In Plant-Microbe Interactions and Biological Control (pp. 93-110). Springer, Dordrecht. DOI: 10.1007/978-94-017-1284-2_6