Transfer Learning with MobileNetV2 for Tomato Leaf Disease Classification

Sartaj Sudheer Babu
Student Number *220684536*
Supervisor Name: Tayyab Ahmad Ansari
MSc Internet of Things

Abstract—Tomato plants are highly vulnerable to various leaf disease, which can considerably affect the growth and yield. Effective illness treatment and prevention depend on the early identification and categorization of these disorders. In this project, a transfer-learning based strategy for categorising tomato leaf disease using mobileNetV2 as the base model is utilised. MobileNetV2 model is pretrained on ImageNet dataset, which can aid in the feature extraction process involved in transfer learning. The final model is a combination of both pretrained MobileNetV2 model and a classifier network. The class sample count disproportion is tackled by runtime data augmentation. The PlantVillage dataset is the major strength of the project and it is used for both training and evaluation process. The quality of the model is determined by assessing the model with various evaluation metrics.

Index Terms—MobileNetV2, transfer learning, CNN, Tomato leaf diseases, Pretrained models, Computer Vision

I. INTRODUCTION

The food demand in world is increasing day by day, and therefore the need of detecting plant diseases using plant parts is crucial. Moreover, detecting plant diseases in time is beneficial for better production and help us to make preventive actions against the spread of diseases. Due to crop diseases, a huge number of farmers are unable to get the profit they forecast. Tomatoes are one of the most versatile and widely used ingredients in culinary preparations worldwide. Tomatoes are one of the most used ingredients for cooking and they are well known for their nutritional value. The cultivation of this famous fruit is detrimentally impacted by pest's attack and lack of proper environmental conditions. (Panno et al., 2021)

Identification of tomato leaf disease is the first step towards preventive action and which in turn can help in the optimal production of tomatoes. At the same time, manual identification has its own downsides, it takes a lot of time, and can be incorrectly identified as another disease etc This may result in the use high amount of pesticides or less pesticides. Lacking prior knowledge, the visual examination of plants may lead to incorrect disease prognosis. Crop damage can be caused by misdiagnosing diseases and using too many or too few pesticides. On the other hand, manually identifying tomato illnesses through careful plant observation takes time. The visual inspection of plants might result in inaccurate disease diagnosis if preceding knowledge is lacking. Misdiagnosis of illnesses and overuse or underuse of pesticides are two factors that might harm crops. These process of identifying can be easily achieved using a lot of machine learning methodologies, in this project a CNN (Convolutional Neural Networks) model is proposed. (Liakos et al., 2018)

Deep learning models are having an upper hand over other machine learning algorithm provided that we have a reasonable amount of data is used for training. Likewise, CNN has the ability to find intricate patterns and edges in an image, therefore CNN is considered as a good choice for image related projects. The project proposes a lightweight transfer learning based MobileNetV2 (Sandler et al., 2018) model for detecting diseases on tomato leaves and finally evaluating the trained model using evaluation metrics to measure the performance.

II. LITERATURE REVIEW

Traditional illness categorization and detection methods are demanding of labour, frequently inaccurate, and were based on hand crafted feature

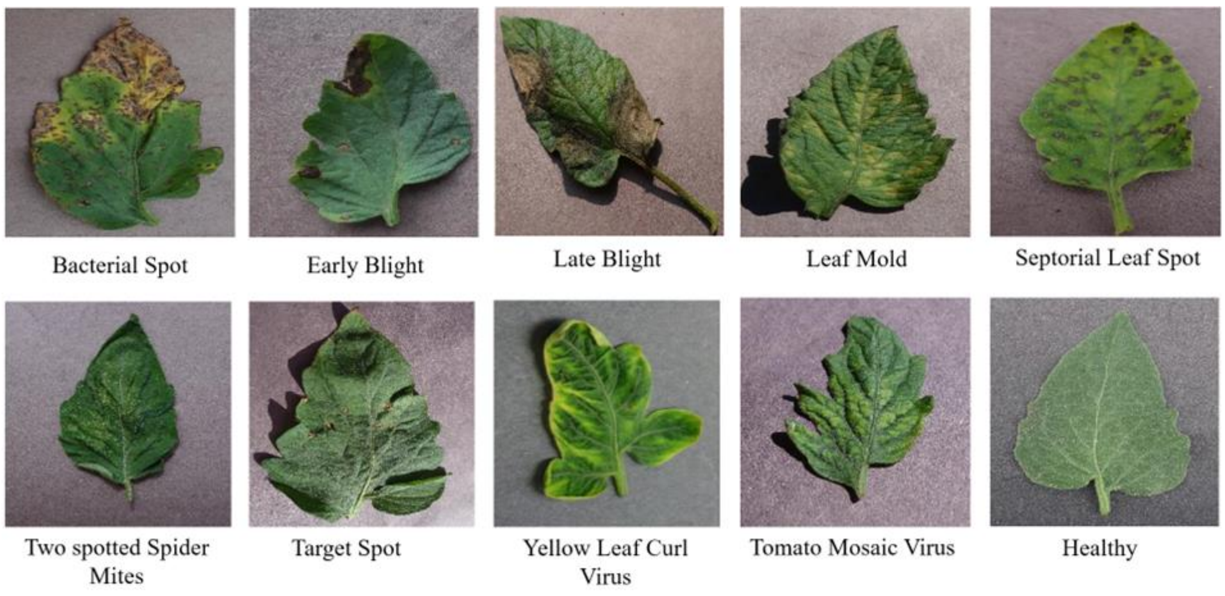


Fig 1. Sample pictures of tomato leaves from the 10 classes in the PlantVillage dataset.

extraction methods. Methods of machine learning (ML) relied on rigorous pre-processing operations, such as cropping the area of interest, background removal, images undergo filtering for better feature extraction, Changing the colour grading, and scaling the image. Traditional machine learning algorithms perform better when there are only a few number of classes, and because of the higher sophistication brought on by these pre-processing procedures, and they were unable to generalise to larger sizes (Ahmed et al., 2022). An overview of studies employing conventional machine learning methods for identifying plant diseases was published by (Yang and Guo, 2017). However, with the help PlantVillage dataset, which include diseased images of tomato plant, deep learning models were able to perform different analysis on tomato leaves. Some of these works include real time localization of diseases (Liu and Wang, 2020), segmenting leaves from different backgrounds (Ngugi, Abdelwahab and Abo-Zahhad, 2020), visualising features learned at different layers of CNN model (Fuentes et al., 2017), combining leaf classification and segmentation (Yang et al., 2020). For leaf disease classification, transfer learning-based approaches are widely used, investigated with various pretrained models, Fine-tuned the models from different depths and studied the impact hyperparameter choices. These researches suggested using CNN architecture like GoogleNet (Wu, Chen and Meng, 2020), AlexNeT (Rangarajan, Purushothaman and Ramesh, 2018), ResNet (Zhang et al., 2018), MobileNetV2(Ahmed et al., 2022). These transfer learning-based models which are pretrained using huge datasets performed better when compared with simple machine learning models. The accessibility of appropriate labelled dataset is crucial in categorising the classes. The dataset should also be able to represent the real data, using which the inference is done.

III. DATASET

TABLE1: Sample Distribution across dataset

Class Label	Sample count
Bacterial Spot	2127
Early Blight	1000
Late Blight	1909
Leaf Mold	952
Septoria Leaf Spot	1771
Two-spotted Spider Mites	1676
Target Spot	1404
Yellow Leaf Curl Virus	5357
Tomato Mosaic Virus	373
Healthy	1591
Total	18160

The PlantVillage repository (Hughes and Salathe, 2016) contains 61,486 images of 14 different crops that have been labelled by specialists in plant pathology. There are 18,160 tomato leaf images in

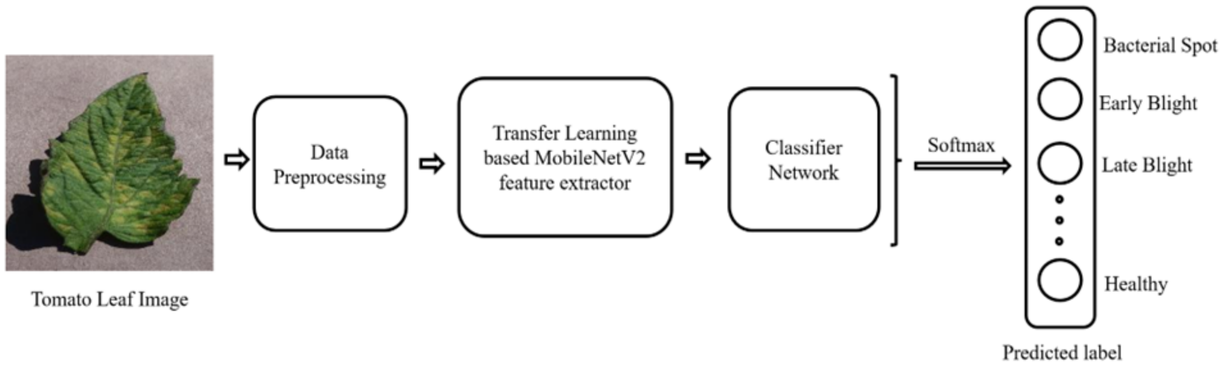


Fig. 2. Tomato Leaf Disease Classification Architecture

that, including 1 healthy tomato leaf class and 9 diseased tomato leaf classes. This dataset provides a wide range of illness and samples of leaves with varying degree of disease infection. Figure 1 shows a simple example photo from every class. The dataset clearly contains disproportion when the number of samples in each class in considered, since all classes have noticeably dissimilarity in sample count, as shown in Table 1. For example, the yellow Leaf curl virus class, the sample count is 5357, which is significantly higher than other classes. Whereas, sample count in Tomato Mosaic Virus is 373. This can cause the model unable to learn properly from the classes with lower sample count, which in turn effects the generalization ability of the model (Chawla et al., 2002). It is possible to address this issue using a variety of methods like under sampling, oversampling, data augmentation guaranteeing that the model can detect every sickness in the same way.

IV. METHODOLOGY

The PlantVillage repository accommodate images of different leaves and their illness classes, out of that, only the classes containing tomato leaf images are taken and is split into 3 different portions which are training set with 60% of total images, validation set with 20 % of total images and test set with 20% of total images. The training and validation set is now converted to zip file and then uploaded to google colab storage and then extracted to original size. Now the GPU devices are activated within the colab for speed up the training process and all the necessary libraries required are imported. Next, the base model, MobileNetV2 which already trained using the ImageNet (Russakovsky et al., 2015) dataset is loaded without the top layer and the parameters are set to trainable mode for fine tuning. A new classification head is created to be attached on top of the base model to make the complete model. The count of units in the last layer of the model is set to 10 because of the ten classes in the dataset, it outputs predictions of each class. Then the training images have to undergo certain preprocessing step before the actual training process. Once the data pre-processing stage is completed, the configurations for the training process are set. Before the data is being injected for actual training process of learning the parameters, data augmentation is performed on it. Additionally, batch size is set to 16 and the image is resized to a size that match the input layer of the final model. Categorial Cross-entropy and Adam is specified as the loss function and optimizer respectively. Finally, the model is trained for 100 epochs, after every epoch the model is saved, and based on the valuation accuracy the best model is selected. The selected model is saved in HDF5 format. The figure 2 shows the CNN architecture through which the image has to go through during inference.

The effectiveness of the finalised model is then assessed using the test with metrics like accuracy, F1-Score, precision, recall. The model is also examined based on its model size and parameter count. All the aspects discussed in the methodology are further discussed in the coming sections.

A. Data Pre-Processing

Additionally, the original dataset has a class imbalance, as was already highlighted. The most popular strategy for addressing this is to under sample or oversample specific classes (Wu, Chen and Meng, 2020). In spite of somewhat balancing the dataset, it has significant disadvantages of its own. As a result, the model's capacity to generalise is eventually hampered. Under-sampling, by which the sample counts of the majority class are decreased, may exclude some of the difficult pictures for particular classes that might contain crucial details for the model, and this may affects the model ability to correctly classify those classes. Because of the drawbacks of the under-sampling and oversampling, number of images are kept the same Resizing and scaling the images were done as part of the preprocessing. (224, 224, 3) was the form of the pre-processed dataset. By rescaling the pixel intensity values range were transformed from (0,255) to (0,1). Runtime data augmentation is used to address the data imbalance among classes.

B. Data Augmentation

The runtime augmentation performed on the dataset include shearing, horizontal flip, zooming, width shifting and height shifting. Shearing, width shift, height shift and zooming factor is taken as 0.2. Contrary to conventional methods of increasing the number of samples by data augmentation, runtime augmentation is employed. Runtime augmentation ensures that model encounters the different augmentations of a single image during each epoch, it can reduce overfitting as the model does not come across the same image in every epoch. Figure 3 shows the transformation of an image under different data augmentation techniques.

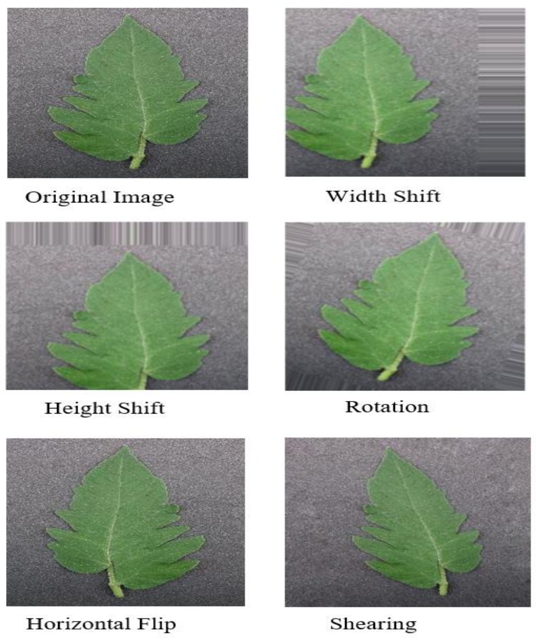


Fig. 3. Data Augmentation

C. Transfer Learning Based Mobilenetv2 Feature Extractor

Transfer learning is a technique in deep learning approach that allows information gained from one task or dataset to be applied to another activity or dataset that is closely related. Transfer learning uses the pre-trained weights and information from a pre-trained model as a starting point for the new task rather than training a model from scratch. Transfer learning minimises the time and computing resources needed for training by using pre-trained models. Instead of starting from scratch when training a model, employ a pre-trained model that has previously picked up important characteristics from a sizable dataset. The model can converge more quickly thanks to this initialization, which gives it a head start. Collecting and labelling a large amount of data for a particular activity can be difficult and time-consuming in many real-world settings. By utilising previously labelled data, transfer learning enables you to get around the data scarcity issue. When you have little data for the target job, the pre-trained model gives you a good initialization, enabling successful learning. You can alter the pre-trained model to meet the needs of your target task by changing or adding particular layers. Because of its adaptability, transfer learning may be used for a variety of purposes and encourages the reuse of previously learned models in many fields.

The ImageNet dataset is used in order to pre-train MobileNetV2. The ImageNet dataset is a large dataset made up of millions of labelled photos categorised into tens of thousands of categories. For several computer vision applications, such as image classification, it has been commonly used as a benchmark dataset.

There are 155 layers in the MobileNetV2 model, which is accessible in the Keras Applications module. Both trainable and non-trainable layers are included in this.

The input picture given to the model is of the shape (224, 224, 3), which corresponds to an RGB image with a resolution of 224x224 pixels. The first convolutional block in the MobileNetV2 model analyses the input picture. An ordinary convolutional layer with 32 filters makes up this block, and then comes the batch normalisation layer and a Rectified Linear Unit (ReLU) activation function. This first block aids in the input image's low-level feature extraction.

The bottleneck block is one of MobileNetV2's key components. The expansion layer, the depthwise convolution layer, and the pointwise convolution layer are the three primary layers that make up the bottleneck block. This block layout preserves key characteristics while reducing computational complexity and model size.

The bottleneck block's expansion layer is its first layer. It tries to increase the number of channels in the feature maps, enabling more expressive ability and capturing intricate patterns. It entails a 1x1 convolution operation with a rectified linear unit (ReLU) activation function. The model can learn and represent richer features with help of this expansion.

After the expansion layer follows the depthwise convolution layer. It performs a distinct spatial convolution on each input channel of the feature maps. It is carried out using a tiny kernel size, usually 3x3, effectively collecting spatial information. By separating the spatial and channel-wise convolutions, the depthwise convolution decreases computational complexity and results in a substantial parameter reduction.

The pointwise convolution layer is applied after the depthwise convolution. As a bottleneck layer, it executes a 1x1 convolution to change the number of channels. While maintaining crucial characteristics, this technique aids in further reducing computational costs and model size. Through the activation function, it also gives the network additional nonlinearity.

Residual connections are included in MobileNetV2 in addition to bottleneck blocks. The residual connection allows information to pass from previous levels to subsequent ones by connecting a block's input straight to its output. This skip link facilitates gradient flow during training, resolving the vanishing gradient issue and enhancing information flow through the network.

In order to balance accuracy and computing performance, the MobileNetV2 design makes use of bottleneck blocks, expansion layers, depthwise convolution layers, pointwise convolution layers, and residual connections. As a result, it may be deployed on devices with limited resources and yet perform well in image classification.

D. Modified Top Layers

Instead of, employing the retrieved features from the pretrained models directly for numerical prediction. As a move towards improving the extracted features from the feature extractor network we used a fusion of dense layer, batch normalization layer and a dropout layer (Ahmed et al., 2022). The activations of the preceding layer are normalised by batchnormalization layer (Ioffe and Szegedy, 2015). By standardising the input to each layer and increasing the effectiveness of the optimisation process, it aids in stabilising and accelerating the training process. It may enhance generalisation and lessen the likelihood of overfitting. The output from the base model is connected to the batchnormalization layer followed by a dense layer comprising of 128 units with ReLU (Glorot, Bordes and Yoshua Bengio, 2011) activation. Next comes the Dropout layer (Srivastava et al., 2014) with a dropout rate of 0.5, induces a regularization effect to prevent overfitting. This layer is followed by another Dense layer with 64 units and ReLU activation. Then comes second batchnormalization layer in the classifier network. Finally, a Softmax (Goodfellow, Bengio and Courville, 2016) activated Dense layer with 10 units, gives the probability of each class. At the time of predicting the class these probabilities are passed to argmax function, which outputs the index of the class with highest probability. Figure 4 shows the custom classifier network attached t the top of the base model.

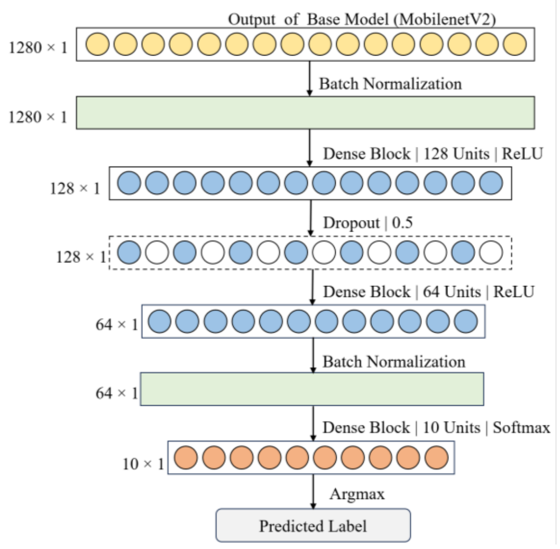

Output of Base Model (MobilenetV2)
1280 × 1
Batch Normalization
1280 × 1
Dense Block | 128 Units | ReLU
128 × 1
Dropout | 0.5
128 × 1
Dense Block | 64 Units | ReLU
64 × 1
Batch Normalization
64 × 1
Dense Block | 10 Units | Softmax
10 × 1
Argmax
Predicted Label

Fig. 4. Classifier Network

V. TRAINING SETUP

The proposed system was trained in Google Colab using TensorFlow, Keras, and other essential tools in a Python environment. Every study was carried out using a 15 GB VRAM NVIDIA Tesla V100 GPU and an Intel Xeon CPU with 2vCPU @ 2.20 GHz.

From each class, 60% of the photos were used for training, 20% for validation, and 20% for testing.

The batch size for the training process is selected as 16. Since smaller batch sizes are frequently noisier, they aid in the creation of a regularisation effect and lower the generalisation error. The training is done for 100 epochs using Adam (Kingma and Ba, 2014) optimization algorithm with a learning rate of 0.001, Adam optimization algorithm performs better than other optimization algorithms for computer vision problems. This optimization algorithm optimises the loss function which is categorial cross-entropy. It calculates the difference between the predicted probability distribution and true distribution (one-hot encoded) of the target classes. Model Checkpoints are used to save the model after every epoch and the one that give the best validation accuracy is taken.

The feature extractor part of the network is initialised using the pretrained weights from ImageNet Dataset (Russakovsky et al., 2015) and the classifier head is initialized with random weights.

VI. EVALUATION METRICS

A. Accuracy

Accuracy is the proportion of all correctly predicted images to all predicted images is known as accuracy. Test set is used for calculating the accuracy of the model, which are hidden from the model during training, to have a better idea of its generalisation capacity (Ahmed et al., 2022).

Accuracy $= \frac{X}{Y} \times 100\%$

Here, Y is the total number of images in the test set, and X represents the total number of examples for which the model's label prediction was correct

B. Precision

The ratio of the total number of true positive image predictions made on all classes to the sum of total number of true positive forecasts and false-positive predictions made across all classes is known as precision. In a situation with multiple classes, precision is used to compare the images that were correctly classified as belonging to each class among all the images that were classified as belonging to that class.

$$Precision_c = \frac{TrueP_C}{TrueP_C + FalseP_C}$$

Here, $TrueP_C$ represents the total amount of images classified as c correctly, while $FalseP_C$ represents the total number of images classified as c incorrectly.

When classes are unbalanced, the macro-average precision is determined by calculating the precision for every class independently and then averaging the results. This makes sure that the model is penalised equally for each incidence of a false positive in any class.

For a set of classes C,

$$\text{Macro Average Precision} = (\sum_{c \epsilon C} Precision_c)/|C|$$

Here, $Precision_c$ is the precision of class c, and $|C|$ is the total number of classes.

C. Recall

Recall is the ratio of the sum of the total number of true positive predictions among all the classes and the sum of the total number of true positive predictions and false-negative predictions among all classes (Ahmed et al., 2022).

$$Recall_c = \frac{TrueP_C}{TrueP_C + FalseN_C}$$

Here, $TrueP_C$ represents the number of images classified as class c correctly, while $FalseN_C$ represents the number of images classified as other classes incorrectly.

When classes are unbalanced, the macro average recall is determined, where the average of each class's individual recollection is used. This makes sure that the model is penalised equally for every false-negative occurrence in every class.

For a set of classes C,

$$\text{Macro Average Recall} = (\sum_{c \epsilon C} Recall_c)/|C|$$

Here, $Recall_c$ is the recall of class c, and $|C|$ is the total number of classes.

D. Parameter Count

The learnable weights of the model are referred to as parameters, and depending on the optimisation technique used, they are modified throughout the backward propagation phase. The total number of parameters in a model gives us an estimate for the time the model will take during training and also influences the size of the model and the inference process (Ahmed et al., 2022).

$$Parameter\ Count = \sum_{i=1}^{La} Parameters_i$$

Here, $Parameters_i$ is the number of trainable weights in the i^{th}layer and La represents number of layers in the finalised model.

E. Model Size

Models after training are saved as an HDF5 file. It consists the model's architecture, weights that are trained, and the status of the optimizer used. For performing inference on single images, the saved model can be loaded using different deployment methods. Model size is defined as the size of the model when it is saved in HDF5 format and MB (Mega Byte) is the standard unit for model size (Ahmed et al., 2022).

F. F1-Score

The F1-Score is the accuracy and recall weighted average that takes into account both the quantity of erroneous positive predictions and false negative predictions. A high F1-Score is essential when dealing with an unbalanced dataset to lessen the likelihood of making false positive and false negative predictions (Ahmed et al., 2022).

F1-Score for every class c can be measured using the formula given below:

$$F1 - Score_c = \frac{2 \times Precsion_C \times Recall_C}{Precision_C + Recall_C}$$

Here, $Precsion_C$ and $Recall_C$ are the precision and recall of class c respectively.

However, for disproportionate classes macro average F1-score is used where the F1-score for each and every class is determined separately and their mean is taken.

$$Macro\ Average\ F1 - Score = \frac{\sum_{c \in C} FScore_c}{|C|}$$

Here, $FScore_c$ is the F1-Score for class c and $|C|$ is the total number of classes.

VII. RESULTS AND DISCUSSION

The model was trained for 100 epochs. However, the best model was selected in the 96th epoch based on the validation accuracy. The training process took around 4 hours to complete 100 epochs. The plot of loss and accuracy against epochs are shown in figure 5 and 6 respectively.

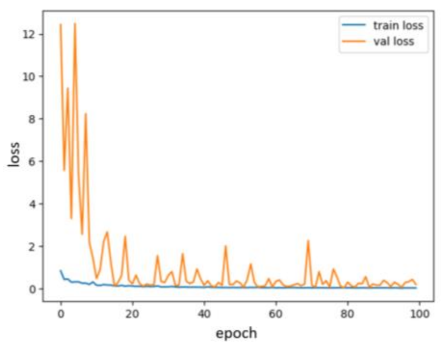


Fig. 5. Plot of training and validation loss against epochs

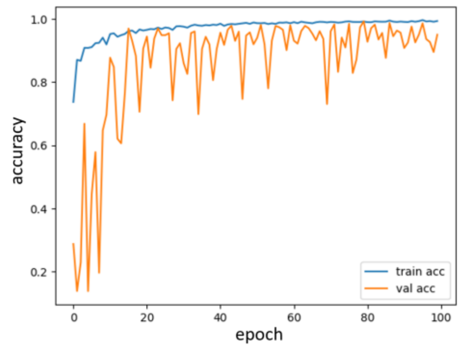
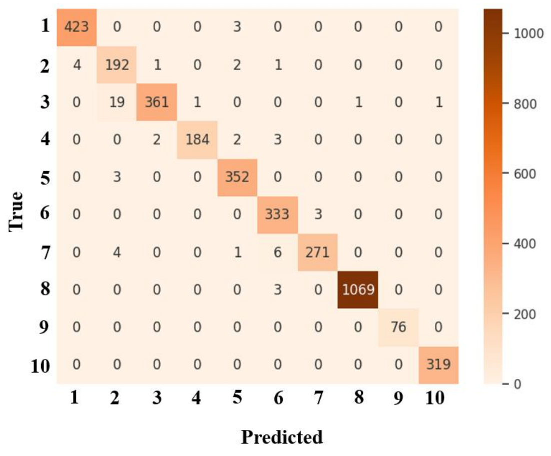


Fig. 6. Plot of training and validation accuracy against epochs



True \ Predicted	1	2	3	4	5	6	7	8	9	10
1	423	0	0	0	3	0	0	0	0	0
2	4	192	1	0	2	1	0	0	0	0
3	0	19	361	1	0	0	0	1	0	1
4	0	0	2	184	2	3	0	0	0	0
5	0	3	0	0	352	0	0	0	0	0
6	0	0	0	0	0	333	3	0	0	0
7	0	4	0	0	1	6	271	0	0	0
8	0	0	0	0	0	3	0	1069	0	0
9	0	0	0	0	0	0	0	0	76	0
10	0	0	0	0	0	0	0	0	0	319

Fig. 7. Confusion matrix

Reference	**Number of classes**	**Image Count**	**Accuracy (%)**	**Model Size (MB)**
(Durmuş, Güneş and Kırcı, 2017)	10	N/A	94.30	2.94
(Brahimi, Boukhalfa and Moussaoui, 2017)	9	14828	99.18	23.06
(Tm et al., 2018)	10	18160	94.85	156.78
(Rangarajan, Purushothaman and Ramesh, 2018)	7	13262	97.49	350.25
(Zhang et al., 2018)	9	5550	97.28	98.29
(Bir, Kumar and Singh, 2020)	10	15000	98.60	15.59
(Maeda-Gutiérrez et al., 2020)	10	18160	99.39	23.06
(Wu, Chen and Meng, 2020)	5	5300	94.33	23.06
(Abbas et al., 2021)	10	16012	97.11	27.58
(Ahmed et al., 2022)	10	18160	99.30	9.60
This model	**10**	**18160**	**98.35**	**28.7**

Table 2: Comparison of model results with other models

Class Label	**Sample Count**	**Precision**	**Recall**	**F1-Score**
Bacterial Spot	426	0.99	0.99	0.99
Early Blight	200	0.88	0.96	0.92
Late Blight	383	0.99	0.94	0.97
Leaf Mold	191	0.99	0.96	0.98
Septoria Leaf Spot	355	0.98	0.99	0.98
Two-spotted Spider Mite	336	0.96	0.99	0.98
Target Spot	282	0.99	0.96	0.97
Yellow Leaf Curl Virus	1072	1.00	1.00	1.00
Tomato Mosaic Virus	76	1.00	1.00	1.00
Healthy	319	1.00	1.00	1.00

TABLE 3: Per Class Sample Count, Precision, Recall, F1-Score for the test set.

The model has been able to provide an accuracy of 98.4% on the test set with a sample size of 3640. The confusion matrix based on the test set has shown in figure 7. The numbering in the confusion matrix from 1 to 10 corresponds to leaf diseases in the same order of illness listed in Table1. Comparison of the model obtained is done with other latest and up to date models for tomato leaf disease classification in Table 2(Ahmed et al., 2022). The model utilised all the classes and all the images inside those classes of tomato leaf from the PlantVillage dataset. The models with higher accuracy and lesser model size are considered better. The proposed model was able to achieve an accuracy of 98.35% with a model size of 28.7%. The highest accuracy of 99.39% is achieved by Maeda-Gutiérrez et al., 2020, with a model size of 23.06MB. But the model proposed by Ahmed et al., 2022, stands out from the reaming model because not only they were able to achieve an accuracy of 99.30, but with much lesser model size of 9.6 MB. The proposed model could be improved in regard to accuracy and model size. Additionally, Table 3 provides the F1-score, recall, and accuracy for each class and Table 4 contains the overall accuracy, Macro Average precision, Macro average recall, Macro average F1-score, Parameters Count, Model size.

Metric	Value
Accuracy	98.35%
Macro Average Precision	0.978
Macro Average Recall	0.979
Macro Average F1-Score	0.979
Parameters Count	2,430,154
Model Size	28.7 MB

TABLE 4: metrics

VIII. CONCLUSION

In order to fulfil the rising need for food production, quick and precise diagnosis of leaf diseases is very important. The data augmentation technique is performed between each epoch to tackle the issue of class sample count imbalance. With the help of all these pipeline elements, the model was able to concentrate on the illness areas and extract high-level characteristics, resulting in an accuracy of 98.4 %. However, one drawback of the PlantVillage dataset is that the samples were collected in a lab setting. Images of tomato leaves obtained from the field with various settings can be used for additional research. These pictures can include background clutter and occlusion. Furthermore, each of the samples utilised in our investigation has just one illness. Another difficult problem to address would be identifying several illnesses on a single leaf. The identification of the degree of infection on leaves may also be an objective of categorization. Last but not least, this study may be expanded to categorise illnesses from a wider variety of crops.

IX. REFERENCES

[1] Abbas, A., Jain, S., Gour, M. and Vankudothu, S. (2021). Tomato plant disease detection using transfer learning with C-GAN synthetic images. Computers and Electronics in Agriculture, 187, p.106279. doi:https://doi.org/10.1016/j.compag.2021.106279.

[2] Ahmed, S., Hasan, Md.B., Ahmed, T., Sony, Md.R.K. and Kabir, Md.H. (2022). Less is More: Lighter and Faster Deep Neural Architecture for Tomato Leaf Disease Classification. IEEE Access, pp.1–1. doi:https://doi.org/10.1109/access.2022.3187203.

[3] Attallah, O. (2023). Tomato Leaf Disease Classification via Compact Convolutional Neural Networks with Transfer Learning and Feature Selection. Horticulturae, 9(2), p.149. doi:https://doi.org/10.3390/horticulturae9020149.

[4] Bir, P., Kumar, R. and Singh, G. (2020). Transfer Learning based Tomato Leaf Disease Detection for mobile applications. doi:https://doi.org/10.1109/gucon48875.2020.9231174.

[5] Brahimi, M., Boukhalfa, K. and Moussaoui, A. (2017). Deep Learning for Tomato Diseases: Classification and Symptoms Visualization. Applied Artificial Intelligence, 31(4), pp.299–315. doi:https://doi.org/10.1080/08839514.2017.1315516.

[6] Chawla, N.V., Bowyer, K.W., Hall, L.O. and Kegelmeyer, W.P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research, 16(16), pp.321–357. doi:https://doi.org/10.1613/jair.953.

[7] Durmuş, H., Güneş, E.O. and Kırcı, M. (2017). Disease detection on the leaves of the tomato plants by using deep learning. [online] IEEE Xplore. doi:https://doi.org/10.1109/Agro-Geoinformatics.2017.8047016.

[8] Fuentes, A., Im, D.H., Yoon, S. and Dong Sun Park (2017). Spectral Analysis of CNN for Tomato Disease Identification. pp.40–51. doi:https://doi.org/10.1007/978-3-319-59063-9_4.

[9] Glorot, X., Bordes, A. and Yoshua Bengio (2011). Deep Sparse Rectifier Neural Networks. Proc. 14th Int. Conf. Artif. Intell. Statist., 15, pp.315–323.

[10] Goodfellow, I., Bengio, Y. and Courville, A. (2016). Deep Learning. Cambridge, Massachusetts: The Mit Press.

[11] Hughes, D.P. and Salathe, M. (2016). An open access repository of images on plant health to enable the development of mobile disease diagnostics. arXiv:1511.08060 [cs]. [online] Available at: https://arxiv.org/abs/1511.08060.

[12] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.

[13] Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. Computer Science. [online] doi:https://doi.org/10.48550/arXiv.1412.6980.

[14] Liakos, K.G., Busato, P., Moshou, D., Pearson, S. and Bochtis, D. (2018). Machine Learning in Agriculture: A Review. Sensors (Basel, Switzerland), [online] 18(8), p.2674. doi:https://doi.org/10.3390/s18082674.

[15] Liu, J. and Wang, X. (2020). Tomato Diseases and Pests Detection Based on Improved Yolo V3 Convolutional Neural Network. Frontiers in Plant Science, 11. doi:https://doi.org/10.3389/fpls.2020.00898.

[16] Maeda-Gutiérrez, V., Galván-Tejada, C.E., Zanella-Calzada, L.A., Celaya-Padilla, J.M., Galván-Tejada, J.I., Gamboa-Rosales, H., Luna-García, H., Magallanes-Quintanar, R., Guerrero Méndez, C.A. and Olvera-Olvera, C.A. (2020). Comparison of Convolutional Neural Network Architectures for Classification of Tomato Plant Diseases. Applied Sciences, [online] 10(4), p.1245. doi:https://doi.org/10.3390/app10041245.

[17] Ngugi, L.C., Abdelwahab, M. and Abo-Zahhad, M. (2020). Tomato leaf segmentation algorithms for mobile phone applications using deep learning. Computers and Electronics in Agriculture, 178, p.105788. doi:https://doi.org/10.1016/j.compag.2020.105788.

[18] Panno, S., Davino, S., Caruso, A.G., Bertacca, S., Crnogorac, A., Mandić, A., Noris, E. and Matić, S. (2021). A Review of the Most Common and Economically Important Diseases That Undermine the Cultivation of Tomato Crop in the Mediterranean Basin. Agronomy, 11(11), p.2188. doi:https://doi.org/10.3390/agronomy11112188.

[19] Rangarajan, A.K., Purushothaman, R. and Ramesh, A. (2018). Tomato crop disease classification using pre-trained deep learning algorithm. Procedia Computer Science, 133, pp.1040–1047. doi:https://doi.org/10.1016/j.procs.2018.07.070.

[20] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, [online] 115(3), pp.211–252. doi:https://doi.org/10.1007/s11263-015-0816-y.

[21] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. and Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. doi:https://doi.org/10.1109/cvpr.2018.00474.

[22] Srivastava, N., Hinton, G.E., Krizhevsky, A., Ilya Sutskever and Ruslan Salakhutdinov (2014). Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res., 15(1), pp.1929–1958.

[23] Tm, P., Pranathi, A., SaiAshritha, K., Chittaragi, N.B. and Koolagudi, S.G. (2018). Tomato Leaf Disease Detection Using Convolutional Neural Networks. [online] IEEE Xplore. doi:https://doi.org/10.1109/IC3.2018.8530532.

[24] Wu, Q., Chen, Y. and Meng, J. (2020). DCGAN-Based Data Augmentation for Tomato Leaf Disease Identification. IEEE Access, 8, pp.98716–98728. doi:https://doi.org/10.1109/access.2020.2997001.

[25] Yang, G., Chen, G., He, Y., Yan, Z., Guo, Y. and Ding, J. (2020). Self-Supervised Collaborative Multi-Network for Fine-Grained Visual Categorization of Tomato Diseases. IEEE Access, [online] 8, pp.211912–211923. doi:https://doi.org/10.1109/ACCESS.2020.3039345.

[26] Yang, X. and Guo, T. (2017). Machine learning in plant disease research. European Journal of BioMedical Research, 3(1), p.6. doi:https://doi.org/10.18088/ejbmr.3.1.2017.pp6-9.

[27] Zhang, K., Wu, Q., Liu, A. and Meng, X. (2018). Can Deep Learning Identify Tomato Leaf Disease? Advances in Multimedia, 2018, pp.1–10. doi:https://doi.org/10.1155/2018/6710865.