Using Residual Network for Plant Leaf Disease Image Classification

Ayman M. Mahmoud*, Romany F. Mansour, Gamal F. Badran

Mathematics and Computer Department, Faculty of Science, New Valley University, El-Kharja 72511, Egypt

Doi: 10.21608/nujbas.2025.370104.1033

Article information: Submit Date 20 Mar. 2025; Revise Date 3 May 2025; Accept Date 20 May 2025

Abstract

Many people in charge of agriculture field especially orchards, use examination to classify the quality of their harvest visually with expert eyes. This process is often very expensive and requires an abundance of labor, inaccurate due to lack of experience in some cases as well, however, with modern technology we can use deep learning for image classification. Generally, the industry is not always interested in the most accurate technique for a given problem but, in many cases, it is interested in the most appropriate technique for the expected result from that given problem. There should also be a balance between the computational cost and the accuracy. This paper uses deep learning with residual neural network to classify some common fruit tree leaf diseases such as apple scab, apple black rot, peach bacterial spot, potato late blight, and grape black measles, … using their digital images. We have adopted the resNet18 for image classification with Cross Entropy Loss as a loss function. Our optimizer here is Stochastic Gradient Descent. We compare different batch sizes as instance 1, 8, 32, 64, …. The result of changing the batch sizes highlights the improvement of the model performance at the expense of time. We choose 64 as batch size then we compare different values of learning rates starting from 0.01 to 0.05. When we reach the value 0.04 for the learning rate our model achieves a good classification accuracy of 95.1%. Finally, pointing out our future vision to enhance our model to increase accuracy.

Keywords: Deep Learning; Plant Village; Neural Network; Classification; Residual Network.

Introduction

The orchard economy is important for boosting economic growth in rural areas and increasing farmers' income. It also helps improve agricultural production [1]. However, recent climate changes and environmental issues have led to more leaf diseases in fruit trees. These diseases, such as Powdery mildew, leaf scorch, black rot, Cedar apple rust, esca, and bacterial spot, cause significant losses for fruit farmers [2]. To reduce these losses, it is crucial to accurately identify the leaf diseases affecting fruit trees [3]. Traditionally, experienced farmers have done this by hand, but this method is time-consuming, inefficient, and often leads to mistakes [4]. Therefore, we need a faster and more accurate way to classify fruit tree leaf diseases to help with prevention and control [5].
Deep learning has significantly advanced the field of computer vision due to its powerful capability to autonomously extract intricate features from images [6]. This technology is increasingly being integrated into agricultural applications, particularly for the recognition of crop diseases [7].
One notable study by, Elfatimi *et al.* [8] introduced a novel classification method that employs the Mobile NetV2 architecture to recognize different diseases affecting bean leaves. This method not only streamlined the classification process but also achieved a recognition accuracy surpassing 92%, underscoring the reliability of MobileNetV2 in agricultural applications. In another promising study, Vakalapudi & Kumar [9] utilized the EfficientNetB4 model, a state-of-the-art convolutional neural network architecture, to identify various species of pepper leaf diseases. This approach achieved an impressive classification accuracy of 92.19%, demonstrating the model's effectiveness in differentiating between subtle visual signs of disease on pepper leaves.
Zhang *et al.* [10] research provided insights into apple leaf disease recognition through a network specifically tailored for scenarios with limited training samples, referred to as SDINet. His results were notable, with the method yielding commendable recognition rates, which are crucial for real-world applications where data may be scarce. Furthermore, Zhang *et al.* [10] enhanced the preselected box configurations utilized in the SSD (Single Shot Multibox Detector) model and assessed its performance on a dataset specifically focused on watermelon leaf diseases. This refinement led to an outstanding average accuracy of 92.4%, proving the robustness of his approach in a practical setting. Zhang et al. conducted a comprehensive analysis of two prominent models, DeepLabV3+ and U-Net. By scrutinizing their strengths and limitations, he proposed a hybrid two-phase model that incorporates both architectures for the classification of cucumber leaf disease severity. This innovative approach managed to achieve an average classification accuracy of 92.85%, effectively addressing the challenges posed by complex agricultural contexts. Zhou *et al.* [1] made significant strides in the

* Corresponding authors: ayman27@sci.nvu.edu.eg

classification of rice disease through the development of an improved ShuffleNetV2 model. This advanced model achieved a remarkable recognition accuracy of 96.6%, reflecting its potential for high-stakes agricultural monitoring where precise disease identification is essential.

Ding *et al.* **[11]** took a different approach by employing an image segmentation SVM (Support Vector Machine) classifier in conjunction with ResNet18, a well-established deep learning model, to identify specific apple leaf diseases. This combination showcased the versatility of machine learning techniques in tackling disease detection.

Together, these studies illustrate the rapid advancements in deep learning applications for crop disease recognition and highlight the potential for improving agricultural productivity through technology.

Previous studies have successfully identified pests and diseases within specific categories of fruits and vegetables. However, a comprehensive model that can detect multiple diseases across both fruits and vegetables has yet to be developed **[12]**. In real-world orchard settings, different fruit tree species are often interplanted. Research has demonstrated that strategic interplanting can help reduce the spread of pests and diseases; however, there is a lack of studies focused on identifying various diseases that affect these trees. This gap in research can be attributed to several inherent challenges, including: Leaves may share comparable shapes, yet they can exhibit unique symptoms of disease. This highlights the inadequacy of relying solely on leaf shape for accurate disease identification during feature extraction. Furthermore, different diseases can show similar symptoms across various fruit tree species **[13]**; for example, black spot affects apple leaves, while powdery mildew impacts cherry leaves. Therefore, a robust disease detection model must take into account a broader range of visual features beyond just leaf shape. Additionally, the same disease can present differently across various leaf types, further complicating the identification process.

In this study, a residual neural network model was employed as the primary tool for plant leaf disease classification **[14]**. We have adopted the resNet18 model with Cross Entropy Loss as a loss function and our optimizer is stochastic gradient descent **[15]**. We use the RGB images of project as our dataset for training and evaluation of our model which consists of 54305 images containing different plant leaf diseases. The images are organized in 38 folders each folder contains the same disease, so the classes are the names of the folders.

Dataset

To accurately simulate the conditions of an orchard, this study selected a variety of fruit tree leaf disease datasets from the publicly available Plant Village project Which can be easily downloaded from Kaggle. The datasets primarily include apple scab, apple black rot, apple cedar rust, apple healthy, blueberry healthy, cherry healthy, cherry powdery mildew, grape black rot, grape esca, grape leaf spot, orange huanglongbing, peach bacterial spot, strawberry leaf scorch, … . All sample images are in RGB format with three color channels, organized in 38 folders and each image has a resolution of 256×256 pixels. The number of leaf disease datasets and examples of some images can be seen in Fig. 1.

The classes for the classification are the names of the folders and it is the name of the disease. The classes are: Apple scab, Apple Black rot, Apple Cedar apple rust, Apple healthy, Blueberry healthy, Cherry including sour healthy, Cherry (including sour) Powdery mildew, Corn (maize) *Cercospora* leaf spot Gray leaf spot, Corn (maize) Common rust , Corn (maize) healthy, Corn (maize) Northern Leaf Blight, Grape Black rot, Grape Esca (Black Measles), Grape healthy, Grape Leaf blight (Isariopsis Leaf Spot), Orange *Haunglongbing* (Citrus greening), Peach Bacterial spot, Peach healthy, Pepper bell Bacterial spot, Pepper bell healthy, Potato Early blight, Potato healthy, Potato Late blight, Raspberry healthy, Soybean healthy, Squash Powdery mildew, Strawberry healthy, Strawberry Leaf scorch, Tomato Bacterial spot, Tomato Early blight, Tomato healthy, Tomato Late blight, Tomato Leaf Mold, Tomato Septoria leaf spot, Tomato Spider mites Two-spotted spider mite, Tomato Target Spot, Tomato mosaic virus and Tomato Yellow Leaf Curl Virus.

Methodology

We divided the dataset randomly (which consists of 54,305 images) into training data and evaluating data at ratio of 8:2 so the training set consists of 43,444 images, and the evaluating set is 10,861 images as illustrated in Fig 2.

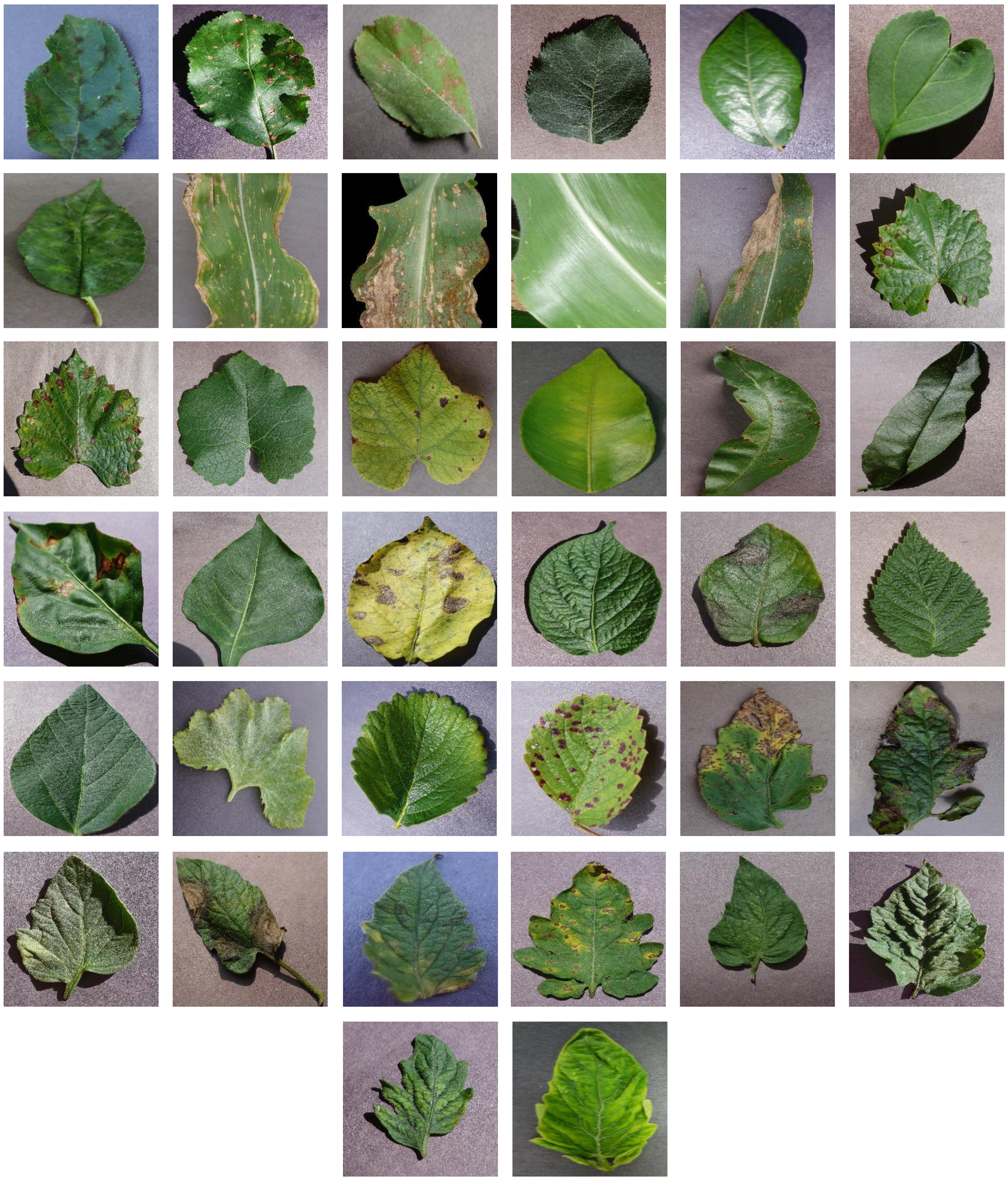
Fig. 1. Examples of different plant leaf diseases

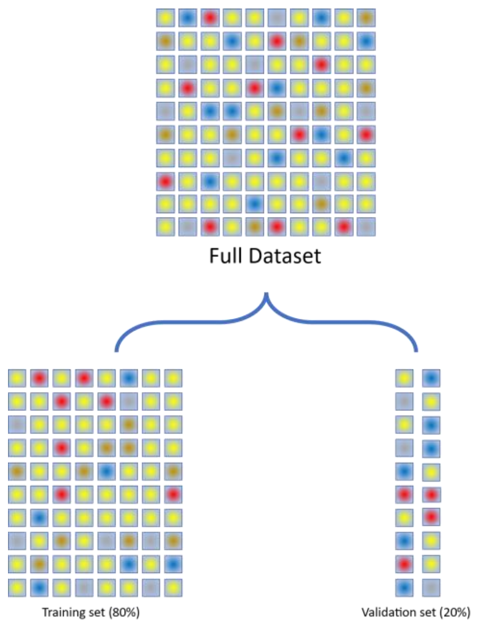

Fig. 2. Dividing the dataset into train and validation set

This paper Uses ResNet-18 for Image Classification. We implement a deep learning pipeline for image classification using the ResNet-18 architecture. It loads a dataset of plant images, applies data augmentation, trains the model on a GPU (if available), and evaluates the performance. The architecture of resNet-18 implementing in this paper is illustrated in Fig. 3.

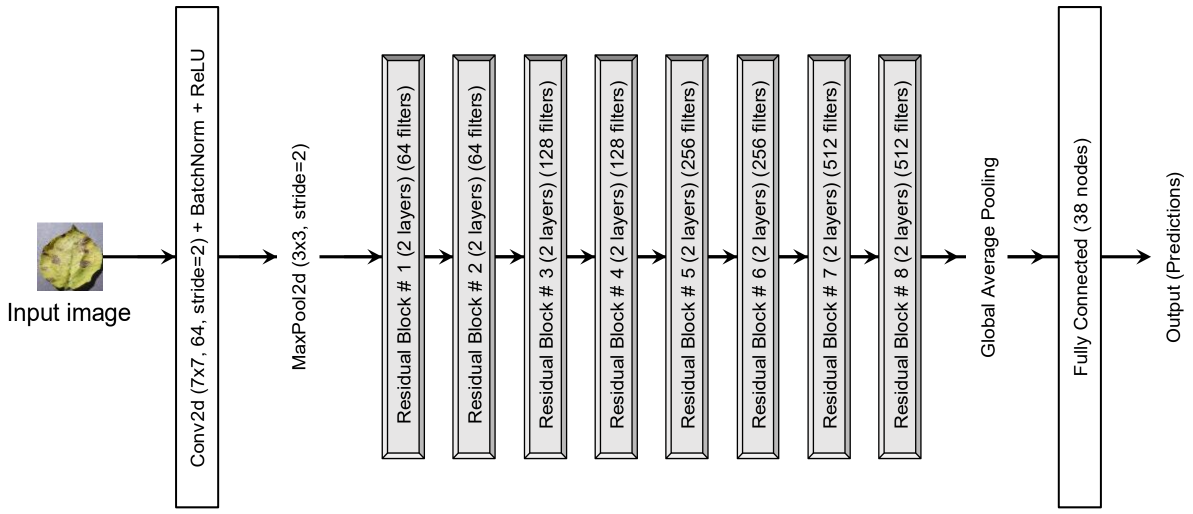
Input image
Conv2d (7x7, 64, stride=2) + BatchNorm + ReLU
MaxPool2d (3x3, stride=2)
Residual Block # 1 (2 layers) (64 filters)
Residual Block # 2 (2 layers) (64 filters)
Residual Block # 3 (2 layers) (128 filters)
Residual Block # 4 (2 layers) (128 filters)
Residual Block # 5 (2 layers) (256 filters)
Residual Block # 6 (2 layers) (256 filters)
Residual Block # 7 (2 layers) (512 filters)
Residual Block # 8 (2 layers) (512 filters)
Global Average Pooling
Fully Connected (38 nodes)
Output (Predictions)

Fig. 3. Architecture of resNet-18

The ResNet18 model we used in this paper is highly proficient in training deep neural networks, primarily due to its implementation of skip connections that effectively address the vanishing gradient problem. Its modular architecture not only facilitates easy scaling and modification but also achieves impressive performance with fewer parameters than many other networks, resulting in faster processing and reduced risk of overfitting. ResNet18 demonstrates a remarkable capacity for learning complex features and is commonly employed as a backbone for a variety of computer vision tasks. Furthermore, pre-trained weights are available to simplify transfer learning. Additionally, it offers superior computational efficiency compared to deeper models in the ResNet family, making it a practical option for a wide range of applications.

The initial layer of resNet-18 is a convolution layer, and it is responsible for procession the input image and prepare it for the coming stages. This layer has kernel size = 7×7, consists of 64 filters, stride = 2. The new size (width/height) of the new output image is given by:

$$\text{output size} = \frac{\text{input size} - \text{kernel size} + 2 \times \text{padding}}{\text{stride}} + 1 \tag{1}$$

BatchNorm2d uses to normalizes the output from the convolutional layer to make the training process more stable and faster. It keeps the mean (average) of the output close to 0 and its standard deviation close to 1. The activation function of the nodes in this layer is ReLU (Rectified linear unit) to add nonlinearity to the model. It makes all negative values zero and keeps positive values as they are

$$ReLU(x) = \max(0, x) \tag{2}$$

MaxPool2d is considered as a pooling operation rather than a layer. It used in Convolutional Neural Networks (CNNs) to reduce the dimensionality of a feature map while preserving the most important features. It takes the maximum value from each specified window (such as 2x2 or 3x3) and uses it as output. We can illustrate how the Maxpool2D in our model with this example in Fig. 4:

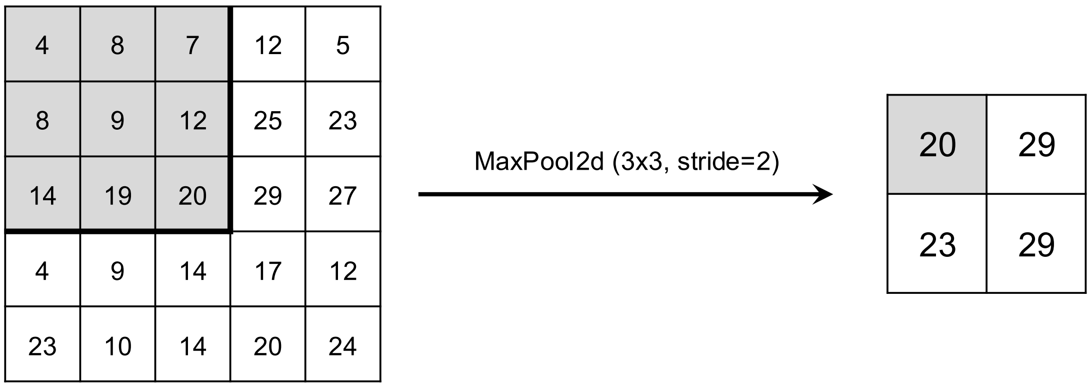

Fig. 4. Maxpool2D (3x3, stride=2)

The first stride the MaxPool2D will work with the matrix $\begin{bmatrix} 4 & 8 & 7 \\ 8 & 9 & 12 \\ 14 & 19 & 20 \end{bmatrix}$. The maximum number here is 20.

Since, the stride = 2 then the second matrix we will deal with is $\begin{bmatrix} 7 & 12 & 5 \\ 12 & 25 & 23 \\ 20 & 29 & 27 \end{bmatrix}$ and the maximum number in it is 29. After finishing the first row the filter will continue down by 2 columns to deal with the matrix

$\begin{bmatrix} 14 & 19 & 20 \\ 4 & 9 & 14 \\ 23 & 10 & 14 \end{bmatrix}$ to get the maximum number 23. The final number we get from the last matrix $\begin{bmatrix} 20 & 29 & 27 \\ 14 & 17 & 12 \\ 14 & 20 & 24 \end{bmatrix}$ is 29.

Every residual block consists of 2 layers but the output of the second layer in the residual block will not be the activation function of the weighted sum, but it will be the value of the input of the first layer plus the activation function of it as illustrated in Fig. 5.

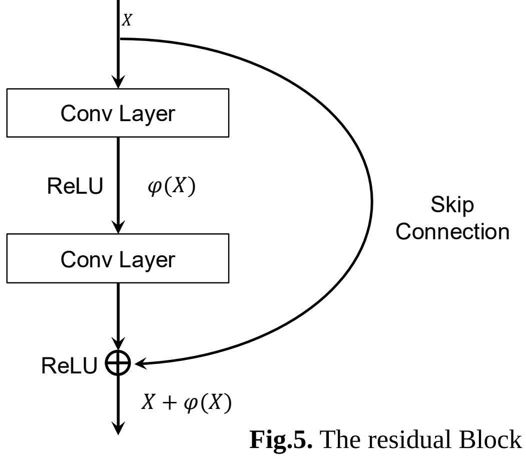

Fig.5. The residual Block

Skip Connection is one of the most important features that have contributed to the success of ResNet-18. The primary role of it is to facilitate the flow of information through the deep neural network, which helps overcome problems such as vanishing gradients. It facilitates the flow of gradients, improve the performance of the neural network and can help speed up the training process, as the network can access information more quickly which reduces the time it takes for the network to converge and reach its best performance.
The following table shows the model structure of the resNet-18.

Table 1. Layers of resNet-18

Stage	Filters	Kernel Size	Stride	Padding	Filters
Conv2d	64	7x7	2	3	64
BatchNorm + ReLU	64	-	-	-	-
MaxPool2d	64	3x3	2	0	-
Residual Block #1	64	3x3	1	1	64
	64	3x3	1	1	64
Residual Block #2	64	3x3	1	1	64
	64	3x3	1	1	64
Residual Block #3	128	3x3	2	1	128
	128	3x3	2	1	128
Residual Block #4	128	3x3	1	1	128
	128	3x3	1	1	128
Residual Block #5	256	3x3	2	1	256
	256	3x3	2	1	256
Residual Block #6	256	3x3	1	1	256
	256	3x3	1	1	256
Residual Block #7	512	3x3	2	1	512
	512	3x3	2	1	512
Residual Block #8	512	3x3	1	1	512
	512	3x3	1	1	512
Global Average Pooling	512	-	-	-	-
Fully Connected	38 nodes	-	-	-	-

We use ResNet-18 because it is very efficient in image classification. It is a very famous type from the ResNet (Residual Network) family of deep neural networks (deep learning) that revolutionized the field of computer vision. The main of it is its 18-layer depth and the architecture of it based on Residual Units which address the Vanishing Gradient problem in deep networks. Residual units help for the training of deeper networks more effectively, enabling the network to learn more complex patterns in the data.

ResNet-18 consists of a series of convolutional layers **[16]**, pooling layers, and the aforementioned residual units. Each residual unit contains a shortcut connection that links the input and output of the unit, allowing for better gradient flow and improving the network's ability to learn complex patterns. This unique architecture enables ResNet-18 to efficiently process images and extract important features from them **[17]**.

ResNet-18 has achieved remarkable performance in various computer vision tasks, such as image classification, object detection, and semantic segmentation. 1 This performance is attributed to the network's depth, the use of residual units, and its ability to extract important features from images. For example, ResNet-18 achieved high accuracy on the renowned ImageNet dataset, surpassing many other models at the time.

Data Augmentation and Transformation

Data augmentation increases the diversity of the dataset, helping the model generalize better. It is performed using PyTorch's transforms module to enhance the diversity of the dataset. The transformations applied include:

Resize ((256,256)): Resize all images to 256×256 pixels.
This part of the code is just for using different image sizes in future work.

Random Horizontal Flip ( ): Randomly flips images horizontally with a probability of 50%.

We use random flips and rotations to simulate different viewing angles.

Random Rotation(15): Rotates images randomly within a range of 15 degrees.

Color Jitter: Adjusts the brightness, contrast, saturation, and hue of the images.

Color jitter adds variation in lighting conditions.

To Tensor ( ): Converts the images into PyTorch tensors.

Normalize: Normalize the images with mean and standard deviation values typical of ImageNet datasets ([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]).

Dataset Loading and Splitting

The dataset is loaded from the specified directory (data_dir) using datasets.ImageFolder. The dataset is then split into training (80%) and validation (20%) sets using PyTorch's random_split method.

Data loader Initialization

PyTorch's Data Loader is used to handle batches of data. The training and validation dataloaders are created with the following parameters:

batch_size=64: Processes 64 images per batch.

We first use batch size as 1 which is faster in training and evaluation, but it makes low accuracy as shown in Fig. 6.

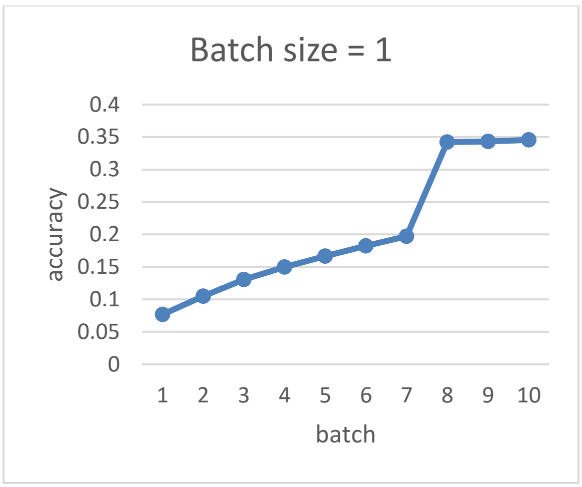

Fig. 6. Training dataset with batch size =1

Table 1. training dataset with batch size = 1 and learning rate = 0.001

	train_value	**val_value**
Epoch 1	0.0768	0.2641
Epoch 2	0.1051	0.2737
Epoch 3	0.1304	0.3063
Epoch 4	0.1497	0.284
Epoch 5	0.1666	0.3191
Epoch 6	0.1824	0.2944
Epoch 7	0.1974	0.2924
Epoch 8	0.3422	0.3032
Epoch 9	0.3432	0.3434
Epoch 10	0.3455	0.3217

So, we use different values of batch size like 8, 32, 64, 128, 256, 512 and 1024 but the higher the value of batch size, the longer the time it takes to finish the training. When we use batch size = 1024 it makes the computational more expensive so to balance data accuracy, calculation complexity and time taken we take batch size as 64 which is fairly decent value with different learning values.

We first use learning rate with batch size = 64 with different values [0.1, 0.2, 0.3, …, 0.9, 1] then [0.01, 0.02, 0.03, …, 0.09] then [0.001, 0.002, 0.003, …, 0.009] and we observe that the training is more stable with learning rate as 0.05 as in Table 3 and it achieved a remarkable accuracy as illustrated in Fig. 7:

Table 2. training dataset with batch size = 64 and learning rate = 0.05

	train_value	**val_value**
Epoch 1	0.8366	0.9126
Epoch 2	0.9021	0.8878
Epoch 3	0.9128	0.9207
Epoch 4	0.9169	0.9159
Epoch 5	0.9181	0.9299
Epoch 6	0.9227	0.9247
Epoch 7	0.9258	0.9126
Epoch 8	0.9494	0.9492
Epoch 9	0.9501	0.9461
Epoch 10	0.951	0.9511

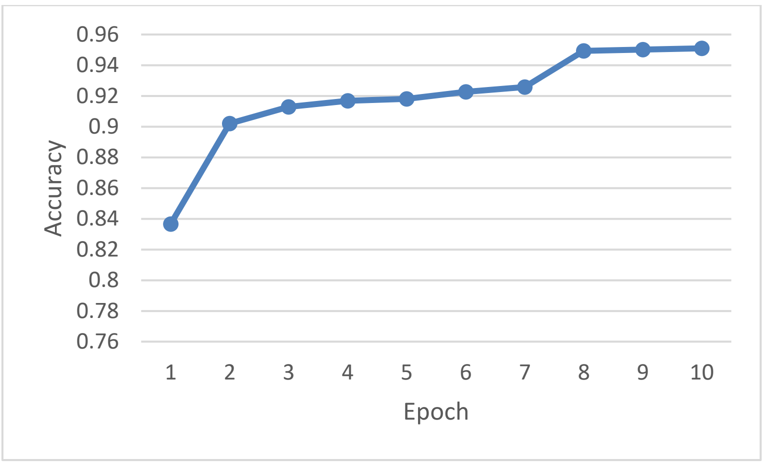

Fig. 7. training dataset with batch size= 64 and learning rate = 0.05

shuffle=True (training) and shuffle=False (validation): Ensures the training data is randomized, while validation data order remains fixed.

Num workers=8: Utilizes 8 threads for loading data.

The ResNet-18 architecture, pre-trained on ImageNet, is loaded using models. Resnet 18 (weights=models. ResNet18_Weights.DEFAULT). The key steps include: Freezing Pre-trained Layers: All layers are frozen to retain pre-trained weights, except the fully connected (FC) layer. Replacing the FC Layer: The FC layer is replaced with a custom classifier using nn.Linear, where the output size matches the number of classes in the dataset.

Loss Function and Optimizer

In this paper, we choose our loss function, cross-entropy loss (nn.Cross Entropy Loss), as the criterion for classification tasks.

$$L = -\sum_{i} y(i) \times \log\big(P(i)\big), \tag{4}$$

where:

$y(i)$: is the true distribution value for class i (the value is 1 if the image belongs to class i, 0 if it does not).

$P(i)$: is the probability calculated by the neural network for class i.

Optimizer: Stochastic Gradient Descent (SGD) optimizes only the parameters of the newly added FC layer. The optimizer uses a learning rate of 0.001 and a momentum of 0.9.

Learning Rate Scheduler: The learning rate is reduced by a factor of 0.1 every 10 epochs using StepLR.

Training and Evaluation

The model alternates between training and validation modes for a specified number of epochs (10). Key steps include:

Forward Pass: Computes the model predictions.

Loss Calculation: Calculates the loss between predictions and ground truth.

Backward Pass (Training Phase Only): Updates the model's weights using gradients.

Accuracy Calculation: Tracks the number of correct predictions.

Model Saving

The trained model can be preserved for later use by utilizing torch.save. The specified saving command enables the storage of the model's weights, allowing for the continuation of training on different GPUs in various locations.
The pre-trained ResNet-18 model leverages knowledge learned from ImageNet, reducing the amount of data required for training from scratch. Only the last FC layer is trained to adapt the model to the new dataset.

Stochastic Gradient Descent (SGD)

We employed SGD, or Stochastic Gradient Descent, here to enhance model performance. It is a fundamental algorithm in machine learning that optimizes model parameters to minimize error. By calculating gradients from small subsets of data, it updates model weights in the direction of the steepest descent. This iterative process continues until convergence is reached.
SGD is efficient with large datasets, explores diverse solutions, and reduces overfitting risk. For example, in image classification, it adjusts weights based on gradients calculated from batches of images, enhancing accuracy with each iteration. In the algorithm of Stochastic Gradient Descent, the weights are updated using the following equation,

$$\omega_{t+1} = \omega_t - \eta \cdot \nabla L(\omega_t), \tag{5}$$

where:

ω_t: the current weight at iteration of t.

η: the learning rate of the model.

$\nabla L(\omega_t)$: the gradient of the loss function with respect to ω_t.

We improved the previous equation by adding the momentum to accelerate the convergence of the model by calculating the velocity vector ν_t as follows:

$$\nu_t = \gamma \cdot \nu_{t-1} + \eta \cdot \nabla L(\omega_t). \tag{6}$$

Then, we update the weight using the velocity vector:

$$\omega_{t+1} = \omega_t - \nu_t , \tag{7}$$

where:

γ: the momentum coefficient. We put it equals to 0.9 in the code.

ν_{t-1}: the previous value of the velocity vector. The initial value $\nu_0 = 0$.

A smaller learning rate and momentum improve stability during optimization, preventing oscillations in the loss function.

Batch Processing

Using a batch size of 64 ensures efficient computation on GPUs, balancing between speed and memory usage. If batch size is set to 1, the training process mimics pure SGD, which might lead to slower convergence and lower accuracy.

Learning Rate Scheduler

The scheduler reduces the learning rate by 0.1 every 10 epochs. This helps the model converge smoothly as training progresses. During each epoch, the following metrics are calculated:

Loss: Measures the model's prediction error.

Accuracy: Percentage of correctly classified images. For example, the model achieves a training accuracy of 96% after 10 epochs.

Potential Improvements

To further enhance the model's performance, the following improvements can be applied:

Fine-tuning Pre-trained Layers: Instead of freezing all layers, unfreeze specific layers (e.g., deeper layers) and fine-tune them on the new dataset.

Use of Advanced Optimizers: Experiment with optimizers like Adam or AdamW for potentially faster convergence.

Data Augmentation: Introduce additional augmentations like CutMix or MixUp.

Hyperparameter Tuning: Adjust learning rate, batch size, and momentum.

Regularization: Apply techniques like dropout to prevent overfitting.

Discussion

There are several key limitations to consider. First and foremost, the model’s generalizability may be constrained by the specific dataset utilized, which might not fully represent the diversity of plant leaf diseases found across different regions. Furthermore, our dataset includes only 38 classes, which is a relatively small number compared to the variety of plant leaf diseases observed in the orchard. Additionally, relying exclusively on images for classification can present challenges; incorporating supplementary data modalities could significantly enrich our understanding of these diseases. Environmental factors, such as varying lighting conditions, may also affect the model's performance, so implementing normalization or domain adaptation strategies should be considered. Lastly, the current model is primarily focused on disease classification rather than estimating disease severity, which could offer valuable insights for effective disease management. By acknowledging these limitations, we can improve the transparency and reliability of the study, leading to more accurate interpretations of the results. Table provides a comparison of various models, including our own, for classifying plant leaf disease images.

Table 3: Compare between different models and our model

Model	Accuracy
YoloV3	86.2%
k-means clustering	82.3%
SVM	87.9%
MobilNetV2	94.3%
resNet18	95.1%

Conclusion

This implementation demonstrates a complete pipeline for transfer learning using ResNet-18. The code showcases the application of data augmentation, transfer learning, and efficient training practices to achieve high accuracy (95.1%). With further optimizations, the model can be adapted for real-world deployment. Fig. 7. shows the model's overall performance.

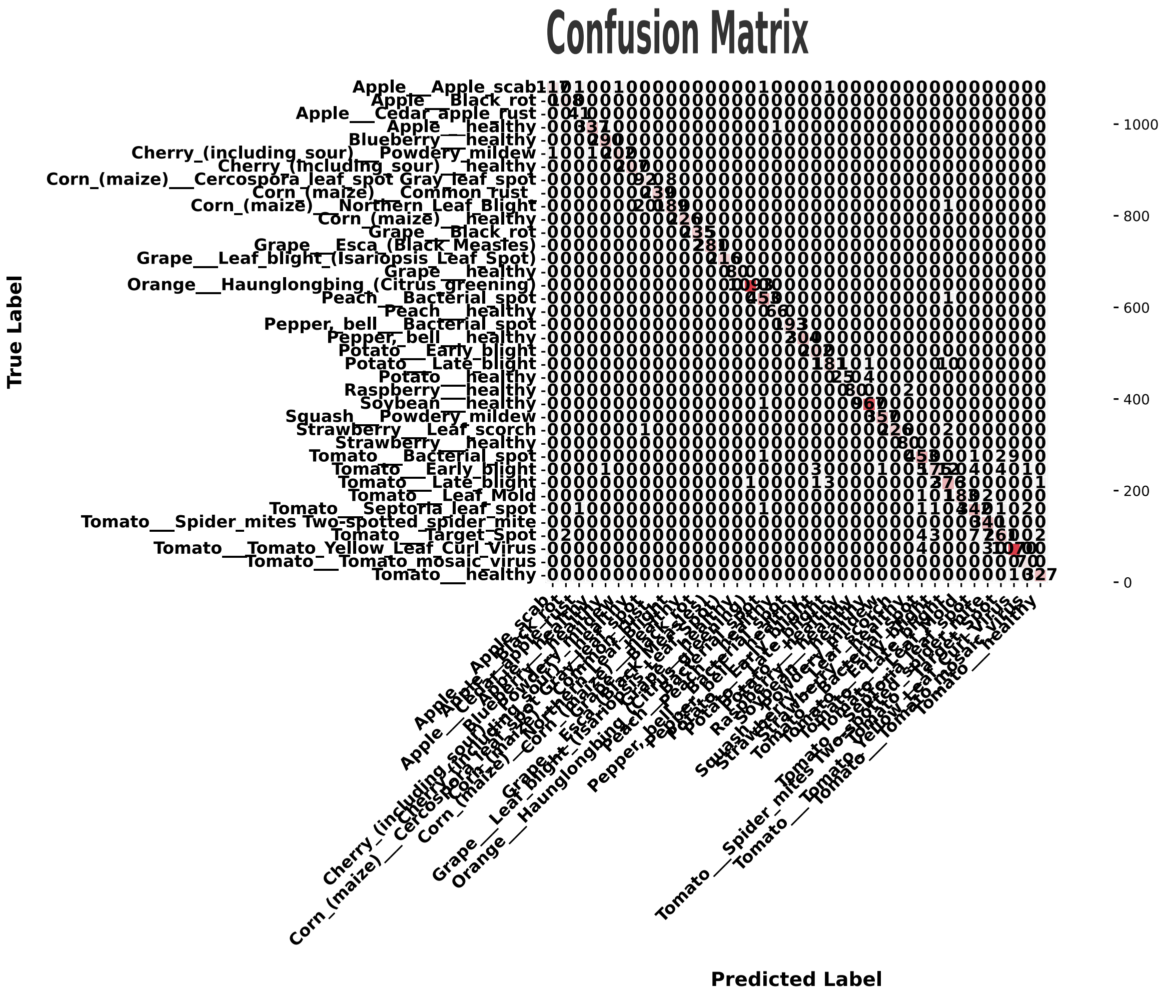

Fig. 8. Confusion matrix of the model

Conflict of Interest

None.

Declaration of Funding

No funding was received for this study

References

[1] Y. Zhou, C. Fu, Y. Zhai, J. Li, Z. Jin, Y. Xu, "Identification of rice leaf disease using improved shufflenet v2.," Computers, Materials & Continua, vol. 75, no. 2, 2023.

[2] W. Hao and S. Zhili, "Improved mosaic: algorithms for more complex images," Journal of Physics: Conference Series, vol. 1684, no. 1, 2020.

[3] R. Ding, Y. Qiao, X. Yang, H. Jiang, Y. Zhang, Z. Huang, D. Wang, H. Liu, "Improved resnet based apple leaf diseases identification," Ifac-Papersonline, vol. 55, no. 32, pp. 78-82, 2022.

[4] S. Kumar, S. Pal, V. P. Singh and P. Jaiswal, "Performance evaluation of resnet model for classification of tomato plant disease," Epidemiologic Methods, vol. 12, no. 1, 2023.

[5] K. M. He, X. Y. Zhang, S. Q. Ren and J. Sun. “Deep residual learning for image recognition”. Computer vision and pattern recognition, vol. 1, pp. 770-778, 2016.

[6] X. He, K. Fang, B. Qiao, X. Zhu and Y. Chen, "Watermelon disease detection based on deep learning," International Journal of Pattern Recognition and Artificial Intelligence, vol. 35, no. 5, 2021.

[7] D. Singh, N. Jain, P. Jain, P. Kayal and S. Kumawat and N. Batra, "Plantdoc: a dataset for visual plant disease detection," Proceedings of the 7th ACM IKDD CoDS and 25th COMAD, pp. 249-253, 2020.

[8] E. Elfatimi, R. Eryigit and L. Elfatimi, "Beans leaf diseases classification using mobilenet models," Ieee Access, vol. 10, pp. 9471-9482, 2022.

[9] K. P. Vakalapudi and N. S. Kumar. “High-precision multiclass classification of chilli leaf disease through customized effecientnetb4 from chilli leaf images”. Smart Agricultural Technology, vol. 5, ISSN. 2772-3755, 2023.

[10] S. Zhang, D. Wang and C. Yu. “Apple leaf disease recognition method based on siamese dilated inception network with less training samples”. Computers and Electronics in Agriculture, vol. 2013, ISSN 0168-1699, 2023.

[11] B. Ding, H. Qian and J. Zhou, "Activation functions and their characteristics in deep neural networks," 2018 Chinese Control And Decision Conference (CCDC), Shenyang, pp. 1836-1841, 2018.

[12] H. Gao, T. Zhen and Z. Li, "Detection of wheat unsound kernels based on improved resnet," Ieee Access, vol. 10, pp. 20092-20101, 2022.

[13] B. Du, W. Li and X. Qin. “A classification and recognition model for multiple fruit tree leaf diseases”. Environmental Research Communications. vol. 6, no. 10, 2024.

[14] M. T. Ahad, Y. Li and B. Song. “Comparison of CNN-based deep learning architectures for rice diseases classification”. Artificial Intelligence in Agriculture, vol. 9, pp 22-35, 2023.

[15] S. Amari. “Backpropagation and stochastic gradient descent method”. Neurocomputing, vol. 5, pp 185-196, 1993.

[16] Q. Wang, B. Wu, P. Zhu, P. Li, W. Zuo and Q. Hu, "Eca-net: efficient channel attention for deep convolutional neural networks," Computer Vision and Pattern Recognition, vol. 4, pp. 11534-11542, 2020.

[17] G. Ghiasi, T. Lin and Q. V. Le, "Nas-fpn: learning scalable feature pyramid architecture for object detection," Computer Vision and Pattern Recognitio, vol. 1, pp. 7036-7045, 2019.