EcoEye: Classification on Plant Pathology Image Dataset Using Deep Learning

Adam Zhao
Stanford University
adamzhao@stanford.edu

Nomin-Erdene Bayarsaikhan
Stanford University
nomin@stanford.edu

Abstract

Without proper early detection and identification of plant pathogens, leaf disease can pose significant challenges to agricultural yields. Instead of visual inspection done by farmers, which is subjective and requires expertise, we propose automated plant disease identification method using deep learning. Among different experiments, training a ResNet-50 architecture delivered the best result with validation accuracy of 96.5% and test accuracy of 65.6%.

1. Introduction

In the new age of agriculture at scale, it is increasingly difficult for farmers to manually inspect every patch of their crop fields and prescribe a proper treatment for the various pests present. If the presence of pests on crops and leaves is not detected early or properly, it could cause major agricultural and economic losses: the potential yield losses caused by plant pathogens can be up to 16% globally [2]. Modern agriculture especially facilitates the rapid spread of pathogens, so identification—one of the major plant pathology management—is imperative to mitigate this challenge.

There is a diverse array of plant pathology techniques, from traditional diagnostic methods to molecular and remote sensing methods. Traditionally, visual inspection has been done by farmers and experts who have experience in identifying plant diseases. This method, while valuable, is often subjective and time-consuming. Advances in molecular biology, on the other hand, have enabled rapid and highly sensitive tools of plant pathology detection. Remote sensing technologies have also proven to be powerful tools for disease monitoring [7].

In this project, we aim to make the process of plant disease identification scalable, inexpensive and reliable for farmers who diagnose diseases visually and who do not necessarily have access to perform molecular techniques, such as polymerase chain reaction. With the use of deep convolutional network, we investigate the feasibility of classifying plant species type and whether the leaf shows any indication of disease infection. In our project, the input is an image of a leaf and the output is the predicted plant pathology category as well as the plant type.

2. Related work

Recent and relevant uses of deep learning architectures for agriculture and plant disease identification specifically are discussed in this section. Aakanksha Rastogi et al. [1] propose a method with two phases: in the first one, the plant is recognized, and in the second phase, they classify the disease present in the leaf. Despite being simple, this approach cleverly introduces a grading scale, indicating the risk level of plant leaves, based on percentage infection.

More recent work moved towards using deep convolutional networks for better performance, as deep learning methods have emerged as revolutionary in the fields of object detection [17], speech recognition [16], drug discovery [5], and natural language processing [9]. Arunabha M. Roy and Jayabrata Bhaduri [3] uses improved and optimized YOLOV4 algorithm, which generates bounding boxes and corresponding confidence scores to detect the location of the disease on leaves. The article by by Murk Chohan et al. [11] also experiments with architectures like VGG and Inception V3 to classify plant pathology. In the paper, Soybean Plant Disease Identification Using Convolutional Neural Network [13], Wallelign et al. investigate the feasibility of CNN for plant disease classification for leaf images taken under the natural environment. They use an architecture consisting of three convolutional layers, each followed by a maxpooling layer, with the final layer being fully connected layer and ReLU being applied at the output of each layers. The proposed method is applied on 12,673 leaf images of four classes. All the input images are 128x128 pixel and the authors experimented on grayscale and segmented version of the original dataset to test the effects of different lighting and background conditions. The model achieved a classification accuracy of 99.32%.

CNN's ability to extract important features in the natural environment is further demonstrated by V V Srinidhi1 et al. [15]. They use deep convolutional neural network architectures to detect apple plant diseases and classify them into four categories: healthy, stab, rust and multi-

ple diseases. Models using EfficientNetB7 and DenseNet achieved accuracy of 99.8% and 99.75% respectively. The article concludes that in contrast to AlexNet, VGG and GoogleNet, heavy models with a large number of parameters and pooling layers, deep CNN models like EfficientNet and DenseNet can be used even with small training data.

Authors in the paper [12] propose a somewhat state-of-the-art method to augment the deep learning-based plant disease classification with attention mechanism. The proposed method is to apply convolutional block attention module (CBAM) to the output feature map of the CNNs. The result of 86.89% accuracy with EfficientNetB0+CBAM indicates the addition of attention mechanism does not improve the classification for plant pathology.

3. Method

3.1. Different Methods

3.1.1 Baseline methods

We experimented with a few baselines. First, we fed the features extracted using ResNet-50 into a Logistic Regression, which then outputted a predicted class. The parameters of ResNet-50 were frozen for this purpose, so the model was optimizing the typical logistic loss, while taking in extracted features from ResNet-50 as input. We used all of the default options offered by the LogisticRegression class in the sklearn library.

Another baseline was a simple convolutional neural network: it uses the

(Conv $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ MaxPool) x N

strategy, upsampling the channels while downsampling the height and width, and then feeds through two fully connected layers.

We also ran the model proposed by user Mohamadreza Momeni which used convolutions with simple residual connections [10]. We made slight modifications both to account for our image sizes being different, and some other small changes that we saw fit (changing the pooling kernel size, for example).

Lastly, we used a convolutional neural network that incorporates residual blocks. The architecture consists of many of these residual blocks, followed by fully connected layers for classification.

A residual block consists of a few convolutional layers followed by batch normalization and ReLU activation. The addition of the input, (or the down or upsampled version of it), to the output of the convolutional layers creates a "short connection" that addresses the vanishing gradient problem in deep networks.

3.1.2 Proposed method

We propose a transfer learning method to classify plant pathology by utilizing a pre-trained ResNet-50 model. ResNet-50 is a convolutional neural network that is 50 layers deep, and similar to our baseline model, the network stacks residual blocks but is much deeper and contains a bottleneck design [8]. Compared to the baseline model, ResNet-50 is designed for very deep networks, making it more suitable for very large and complex datasets.

In addition to ResNet-50, we include two fully connected layers afterwards in order to process the extracted features and to reduce the dimensionality into that of the number of classes that we have. Due to the size of our dataset, we chose to unfreeze all the layers.

3.2. Model Details

3.2.1 Logistic Regression

Following feature extraction from ResNet-50, we used sklearn's LogisticRegression class for our multiclass classification problem. Sklearn uses the cross-entropy loss for its multiclass regression, which is defined as

$$Loss = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(p(y_i)) + (1 - y_i)\log(1 - p(y_i))$$

where N is the size of the dataset. All of the default options provided by sklearn were used, including L_2 regularization, resulting in a true loss term of

$$Loss = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p(y_i))+(1-y_i)\log(1-p(y_i))]+\lambda\|W\|_2^2,$$

where $\lambda = 1$ and $\|W\|_2^2$ represents the squared L_2 norm of the weights of the model.

3.2.2 Simple Convolution

For a convolutional layer, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{out}, H_{out}, W_{out})$ is described by the following equation: $out(N_i, C_{outj})$

$$= b_{(C_{outj})} + \sum_{k=0}^{C_{in}-1} weight(C_{outj}, k) \star input(N_i, k)$$

where $\star$ is the 2D cross-correlation operator, N is the batch size, C is the number of channels, H is the height, and W is the width. The new height and new width can be calculated by the following equations:

$$H_{out} = \lfloor \frac{H_{in} + 2 \cdot padding[0]}{stride[0]} + 1 \rfloor$$

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} input(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

Figure 1: MaxPool 2D equation

and

$$W_{out} = \lfloor \frac{W_{in} + 2 \cdot padding[1]}{stride[1]} + 1 \rfloor$$

For our simple convolution, all convolutional layers were kernel size 3x3, stride 1, and using $'same'$ padding, which adjusts the padding to keep the same height and width after the layer. Each convolution is followed by a BatchNorm2d, which operates following the equation

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta$$

where γ and β are learnable vectors. This allows us to re-center the model's outputs at mean 0 and standard deviation 1, which aids in model training.

We use a Max-Pooling layer of size 2x2 and stride 2 after every layer in order to reduce the spatial dimension. The Max Pooling function's output from a layer with input size (N, C, H, W) and output (N, C, H_{out}, W_{out}) and kernel size (kH, kW) is here 1. We chose the ReLU activation function, which is defined as

$$ReLU(x) = max(0, x)$$

where the max is applied elementwise.

Finally, we use two fully connected layers each seperated by the ReLU function.

Our simple architecture is (Conv $\rightarrow$ BatchNorm $\rightarrow$ ReLU $\rightarrow$ MaxPool) x 4 $\rightarrow$ FC. The convolutions upsample the channels up to 512, while our pooling downsamples height and width down to 8.

3.2.3 Simple Residual Model

Our simple residual model used many of the same elements of the simple convolution model. However, the backbone is now a residual unit which consists of two (Conv $\rightarrow$ BatchNorm $\rightarrow$ ReLU) blocks, followed by the addition of the (downsampled or upsampled, if necessary) input. Mathematically, it would be similar to have some function

$$y = f(x) + x,$$

and here is a diagram of the block 2.

Our residual model consists of 4 sequential blocks, each of which contain three residual blocks and then a Max-Pooling layer. The channels are increased to 512 and the height and width down to 8, and then that is fed into two fully connected layers with ReLU activation.

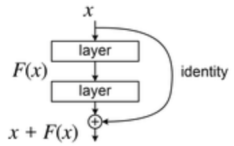


Figure 2: Residual Block

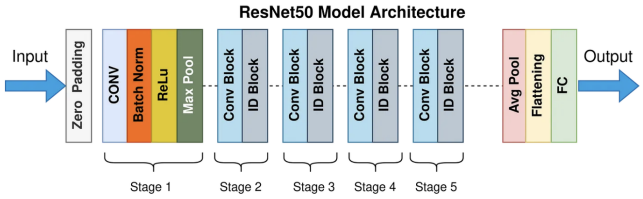


Figure 3: ResNet Architecture

3.2.4 Resnet Models

Our Resnet models all used ResNet-50 as a backbone. We will not get too lost in the details of ResNet-50's architecture, but at its core it consists of several convolutions layers armed with residual connections which allow it to avoid vanishing gradient problems and train very effectively. Because our image size is not the same as the 224x224 expected by ResNet-50, we first introduce a convolutional layer to turn our image into that size. Then, we piggyback off the powerful feature extraction capabilities and add two fully connected layers directly after the ResNet-50 call, separated by the ReLU activation function.

This network also uses the Cross-Entropy loss as defined in Section 3.2.1.

We experimented with two models: one where we froze the parameters of ResNet-50, and one where we unfroze them.

3.3. Training Details

3.3.1 Logistic Regression

For our logistic regression, we used sklearn's LogisticRegression Class. First we applied ResNet-50 on all of our training examples to extract features, and then used sklearn's API in order to train our model. It was extremely simple and straightforward compared to the others.

3.3.2 Deep Models

All of the other proposed models, including the naive convolution, simple residual convolution, as well as the ResNet-50 architectures, were trained using PyTorch. First, we converted our image directories into PyTorch dataloaders using built-in functions. With the 128x128x3 images, we found that we could comfortably use a batch size of 64

while maintaining reasonable training speed and avoiding memory issues.

Our optimizer of choice was Adam, using an initial learning rate of 0.001, and a weight decay of 0.0001. However, we saw issues with our model training as the epochs went on, and as such decided to use a different strategy to control our learning rate and improve results.

For our learning rate, we used PyTorch's OneCycleLR learning rate scheduler, which anneals the learning rate from an initial learning rate to some maximum learning rate and then from that maximum learning rate to some minimum learning rate much lower than the initial learning rate. The exact policy is described in the paper Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates [14]. In our case, that means that our learning rate would start at 0.001, increase to 0.01 (which we set as an upper bound), and then decrease to something much smaller than that. We saw greater success with this learning rate scheduling strategy compared to any particular learning rate that we tried, likely due to it preventing the model from stepping too far away from optimum towards the end of the training process.

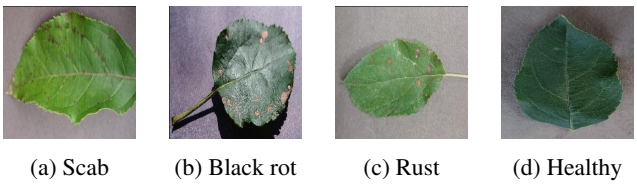
We trained all of our models for 10 epochs. We did this not only because training them for longer would take too much time, but also because our models seemed to converge within those 10 epochs in most cases, likely due to our use of the OneCycleLR scheduler.

4. Dataset

The dataset used for this project [4] is recreated from PlantVillage [6] database using augmentation. It consists of healthy and unhealthy leaf images divided into 38 categories by species and disease. The resolution of the images is 256 x 256, but to prepare the dataset for faster training, the images are re-sized to 128 x 128 pixels at the expense of some visual detail. We also attempted to use a 64 x 64 pixel image, however the amount of detail lost was deemed to be too great. The dataset was pre-split into training, validation, and test sets using each with 70295, 17572 and 38 images in them respectively. In order to make the test set contain more data, we used roughly a 60/20/20 split after removing mislabeled data from the dataset. This resulted in 56251 training, 17572 validation and 14044 test images. Examples of the training dataset is demonstrated in Figure 4 for apple plants in four states: healthy, scabby, rotten and rusty.

A diagram detailing the number of training examples for each class is included here Figure. 5. Our dataset was relatively well balanced, to the point where we did not feel the need to employ reweighted Cross-Entropy loss or specialized methods to sample batches.

Because the dataset already contained several augmentations (such as rotation, brightened or darkened, etc), we did not employ any augmentations of our own.



(a) Scab (b) Black rot (c) Rust (d) Healthy

Figure 4: Examples of different phenotypes of apple plants

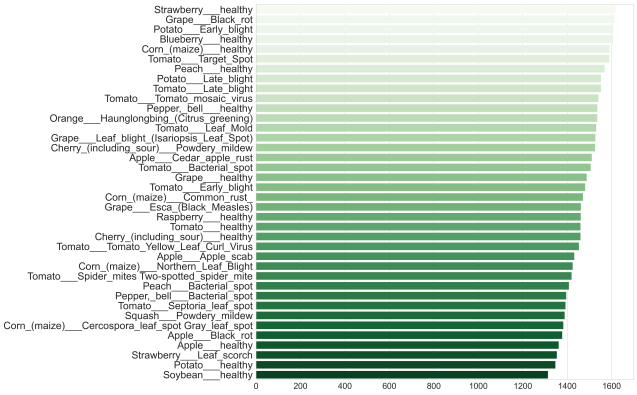


Figure 5: Training dataset distribution across classes

5. Experiments

Our process for selecting hyperparameters was relatively hands off. Because we had extensive use of batch norm layers both in our own models and in ResNet-50, it was in our best interest to have the batch size as large as possible for the best batch normalization performance. That ended up being a batch size of 64 for the 128 x 128 pixel images. We employed the use of a learning rate scheduler, setting an initial learning rate of 0.001 (which is within standard range) and a maximum learning rate of 0.01 (also within standard range). We utilized the Adam optimizer with a weight decay factor 0.0001, which takes into account the complexity our model would need while also somewhat encouraging the weight values to be small. We chose to use 10 epochs initially as a starting point and adjust as needed, however we saw convergent behavior consistently within 10 epochs, likely due to the scheduler that we used.

6. Results

6.1. Quantitative results

We ran the different models for 10 epochs and obtained statistics after each epoch. The results are shown in Table 1 in forms of final training loss and accuracy values. Table 2 also shows the precision and recall values that can be used to compare the different models we considered. To better understand how these models generalize to new and unseen

data, we also obtained test accuracy values, shown in Table 3.

	Val Loss	Val Accuracy
LogReg	0.271	0.941
Kaggle	0.193	0.941
Raw CNN	0.047	0.985
Residual Blocks	0.051	0.984
Resnet Frozen	0.751	0.773
Resnet Unfrozen	0.112	0.965

Table 1: Loss and Accuracy from Final Epoch

	Precision	Recall
LogReg	0.941	0.941
Kaggle	0.941	0.941
Raw CNN	0.985	0.985
Residual Blocks	0.984	0.984
Resnet Frozen	0.771	0.772
Resnet Unfrozen	0.965	0.965

Table 2: Precision and Recall from Final Epoch

	Test Accuracy
LogReg	20.3%
Kaggle	59.7%
Raw CNN	50.6%
Residual Blocks	47.6%
Resnet Frozen	29.4%
Resnet Unfrozen	65.6%

Table 3: Test Set Performance

Based on these results, we can see that our proposed method of using ResNet-50 architecture and training all the layers achieved the highest performance on the test set. The loss curve, precision-recall curve as well as the confusion matrix of our proposed method are shown for visualization purposes of the results.

The loss curve (Fig 6) shows a consistent fall in the training loss value as well as for validation loss value that starts from a relatively large value.

The precision-recall curve (Fig. 7) shows the trade-off between precision and recall for different threshold. For all 38 classes, the curves have similar characteristics: the classifier maintains both a high precision and high recall across the graph at most thresholds. This means that the model is returning accurate results as well as a majority of all positive results.

Fig. 8 is the confusion matrix for our proposed method. High values on the diagonal imply that the model performs well on the dataset. For example, healthy strawberry (strawberry_healthy) has a diagonal value of 477, indicating that the model correctly predicted this class 477 times. Since the off-diagonals have relatively lower values, we can see that the model does not misclassify as much.

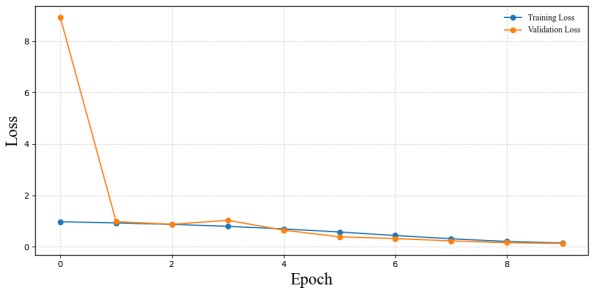


Figure 6: Loss Curve

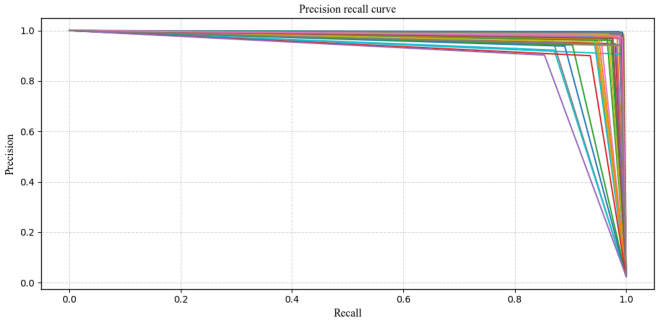


Figure 7: Precision-Recall Curve

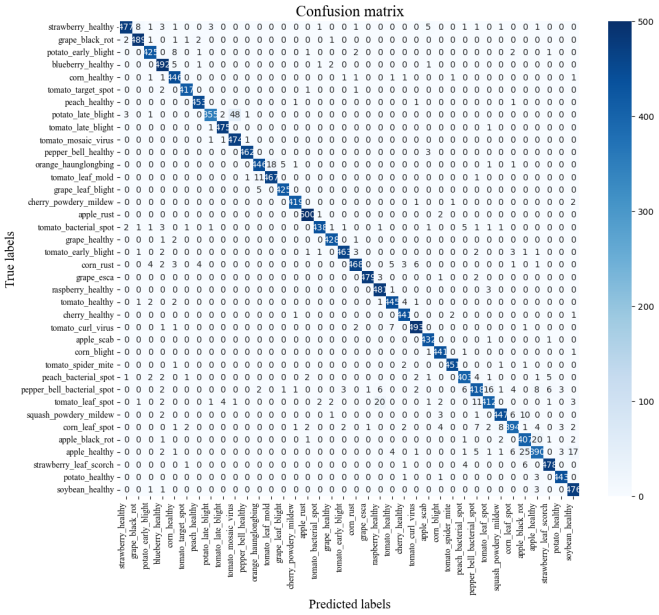


Figure 8: Confusion Matrix

(a) Grape leaf with esca

(b) Grape leaf with rot

Figure 9: Failure case 1: Same plant type with different diseases

(a) Healthy apple leaf

(b) Healthy cherry leaf

Figure 10: Failure case 2: Healthy plants of different types

6.2. Qualitative results

In this section, we look at failure modes to understand why the model mislabeled data. Inspecting the misclassified instances, we found out that there were mainly two failure cases. The first one was that the plant was correctly identified but the disease was mislabeled, as seen in Figure 9. The second case was when the disease condition type matched but the plant type was misclassified as shown in Figure 10. Such failures could be due to the fact that there are low inter-class variation: images from the same sub-class share very similar visual characteristics.

7. Conclusion & Future Work

In this paper, we experimented with multiple classification architectures to predict plant diseases based on images of plant leaves. Our final model, training all the layers of ResNet-50 architecture on our dataset, performed the best (in terms of test accuracy as it shows model's ability to generalize to new and unseen data): validation set accuracy of 96.5% and test set accuracy of 65.6%.

If we had more team members, time and compute, we would have liked to train our model on the original dataset without having the need to down sample them. We would also like to explore other architectures and models such as transformers, which might improve model performance on our dataset.

Nevertheless, we learned a lot from this project, from transfer learning to visualization of our results. This work provides an efficient method for detecting different plant diseases, especially for plant owners who do not necessarily have expertise in plant pathology. With further advancement in the field of computer vision, this system can be integrated to drones for live and automated agricultural detection processes.

8. Contributions & Acknowledgement

Both members had balanced contribution to the project in terms of coding, training the models, writing the report and conducting literature review.

References

[1] S. S. Aakanksha Rastogi, Ritika Arora. Leaf disease detection and grading using computer vision technology fuzzy logic. *IEEE*, 2015.

[2] G. B. G. B. Andrea Ficke, Christina Cowger. Understanding yield loss and pathogen biology to improve disease management: Septoria nodorum blotch - a case study in wheat. *APS Publications*, 2018. doi: 10.1094/PDIS-09-17-1375-FE.

[3] J. B. Arunabha M. Roy. A deep learning enabled multi-class plant disease detection model based on computer vision. *MDPI AI*, 2021.

[4] S. Bhattarai. New plant diseases dataset. `https://www.kaggle.com/datasets/vipoooool/new-plant-diseases-dataset/data`, 2019.

[5] O. E. Y. W. M. O. Chen, Hongming and T. Blaschke. The rise of deep learning in drug discovery. *Drug discovery today*, 2018.

[6] M. S. David. P. Hughes. An open access repository of images on plant health to enable the development of mobile disease diagnostics. *Frontiers in Plant Science*, 2016.

[7] C. A. R. et al. Advancing disease management in agriculture: A review of plant pathology techniques. *Plant Science Archives*, 2024. doi: 10.5147/PSA.2024.9.1.16.

[8] S. R. J. S. Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. 2015.

[9] H. Li. Deep learning for natural language processing: advantages and challenges. *National Science Review*, 2018.

[10] M. Momeni. Plant diseases detection pytorch. `https://www.kaggle.com/code/imtkaggleteam/plant-diseases-detection-pytorch`.

[11] R. C. S. H. K. M. S. M. Murk Chohan, Adil Khan. Plant disease detection using deep learning. *IJRTE*, 2020.

[12] M. S. Pendar Alirezazadeh and F. Stolzenburg. Improving deep learning-based plant disease classification with attention mechanism. *Gesunde Pflanzen*, 2023.

[13] C. B. Serawork Wallelign, Mihai Polceanu. Soybean plant disease identification using convolutional neural network. *FLAIRS-31*, 2018.

[14] L. N. Smith and N. Topin. Super-convergence: Very fast training of neural networks using large learning rates, 2018.

[15] K. D. V V Srinidhi, Apoorva Sahay. Plant pathology disease detection in apple leaves using deep convolutional neural networks. *IEEE*, 2021.

[16] J. G. J. P. A. E.-D. M. W. J. Zhang, Zixing and B. Schuller. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Transactions on Intelligent Systems and Technology*, 2018.

[17] P. Z. S.-t. X. Zhao, Zhong-Qiu and X. Wu. Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 2019.