Plant Pathology Image Classification

Bi Nguyen

University of California - Davis

ECS 171: Machine Learning

Professor Setareh Rafatirad

5 December 2021

Plant Pathology Image Classification

Introduction

With the world population growing exponentially, food scarcity remains a major threat in many countries. It is expected by 2050, the world population is projected to reach nearly over 10 billion people with a growing food demand between 59% and 98% (Elferink & Schierhorn, 2016). Agricultural farming contributes to 70% to 80% of the world's food, respectively to any family farm size (Ritchie, 2021). Not only that, but these farmers rely heavily on their crops for survival. So it is important to help farmers be successful in producing healthy crops for the sake of their livelihood as well as everyone's.

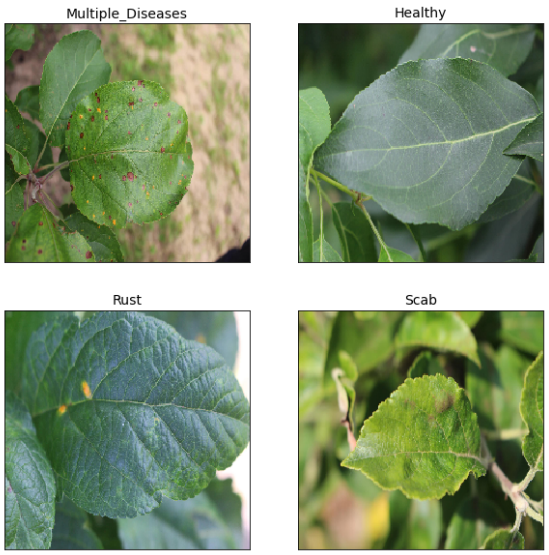
One of the many issues of crop failure is plant pathogens. These diseases drive food production down and increase the cost of treatment, and manually identifying the different underlining diseases can be time-consuming and costly. Therefore, it is crucial to identify the correct disease to apply the proper chemicals that help the crops and provide nutrients to the soil ("Chemicals in the Farm Ecosystem," 2019). With the rapid emergence of smartphones with different image capturing capabilities and recent developments in computer vision, we can help farmers reduce the misuse of chemicals that negatively affects the crops and environment by using deep neural network specifically convolutional neural network for image classification.

Method

Data Description

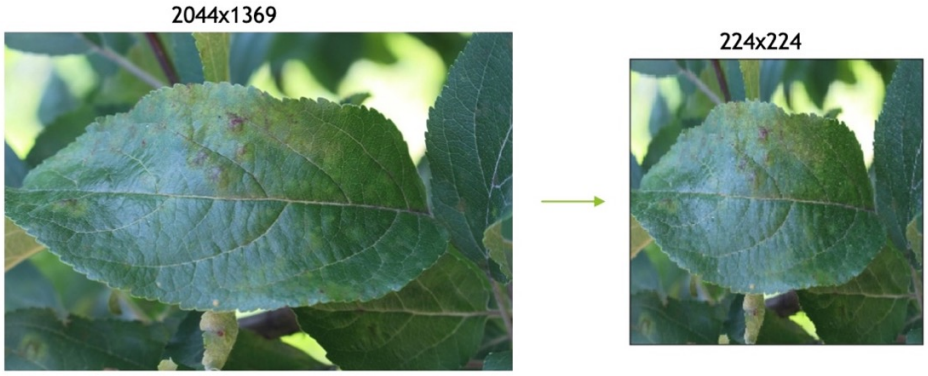
The dataset contains 3,240 images with 4 labels assigned to them. Figure 1 shows the images of the different classes assigned to them. Most of the images came with the same resolution of 2044 x 1369 pixels while some came at the inverse resolution of 1369 x 2044.

Figure 1
Class Labels



In order to apply any machine learning model to these images, I had to rescale them to different sizes. During the initial trials with different resolutions, I found that 224 x 224 pixels were the optimal size because it still captures the important details of the plant leaves while being computationally feasible. Figure 2 shows the rescaling of the original image to 224 x 224 pixels. You can see still see the marker that identifies this leave has multiply diseases.

Figure 2
Reduce Resolution



Note that of those images, half of them are divided as training set and the other as test set. That means only about 1,600 images are labeled and used to train the deep neural network. So how can we test the accuracy of our model performance? By splitting 20% of the training samples into a validation. Since our sample size decreased even more, this will possibly affect the performance of our model. Ideally, in any neural network, the more samples we have, the better the model will be at making predictions. Think of it this way, if we were to look at different images of different cat breeds, it is likely that we won't remember them after only seeing a few images. Therefore, if we were to look at many pictures of the same label, then we will be better at identifying the pattern for a particular label. This same idea applies to machine learning models.

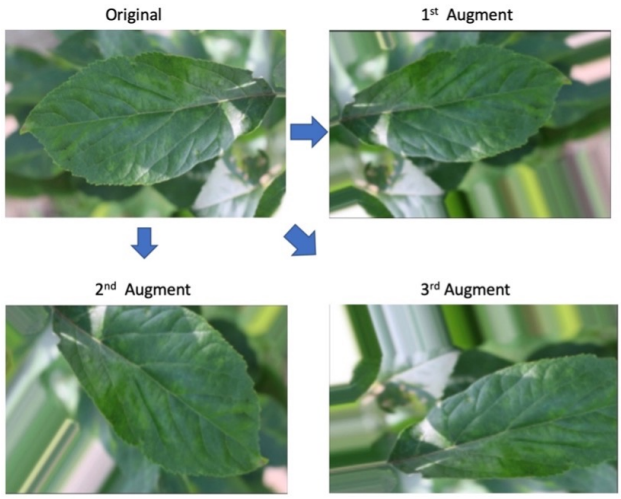
As a result of subsetting the training data, the features in the training set contains 77 images of plants with multiple diseases, 405 images of healthy plants, 511 images of plants with rust, and 463 images of plants with scab while the validation consists of 14 images of plants with multiple diseases, 111 images of healthy and rust plants and 129 images of plants with scab. Notice that the distribution of plants with multiple diseases is smaller compared to the other 3 labels. This means that our model might not be good at classifying plants with multiple diseases because it doesn't nearly have enough data to recognize patterns that can describe this class label.

Approach

If we take a look at Figure 1 again, notice that there are different variations in how the images have been taken. Each picture can have a different orientation, height, and width, whether it's zoomed in or not, blurry background, and level of brightness. All of these distinctions and combinations need to be addressed when building our model. We

want to address this issue because due to the lack of data, accounting for different ways of how the images are taken can help generalize our model and prevent overfitting. Figure 3 display an image with different sets of variations.

Figure 3
Image Augmentation



From the above images, you can see that these transformations can add additional data to our model so that it can generalize different images and hopefully improve the model's performance. Note that the images are augmented randomly with the following settings:

- Rotation Range: 45°
- Brightness Range: [0.2, 0.5]
- Height Shift Range: 0.2
- Width Shift Range: 0.2
- Zoom Range: 0.2
- Horizontal Flip, and Vertical Flip
- Fill mode: Nearest

I choose these settings because it resembles how some of the images are taken in the dataset.

Building the Model

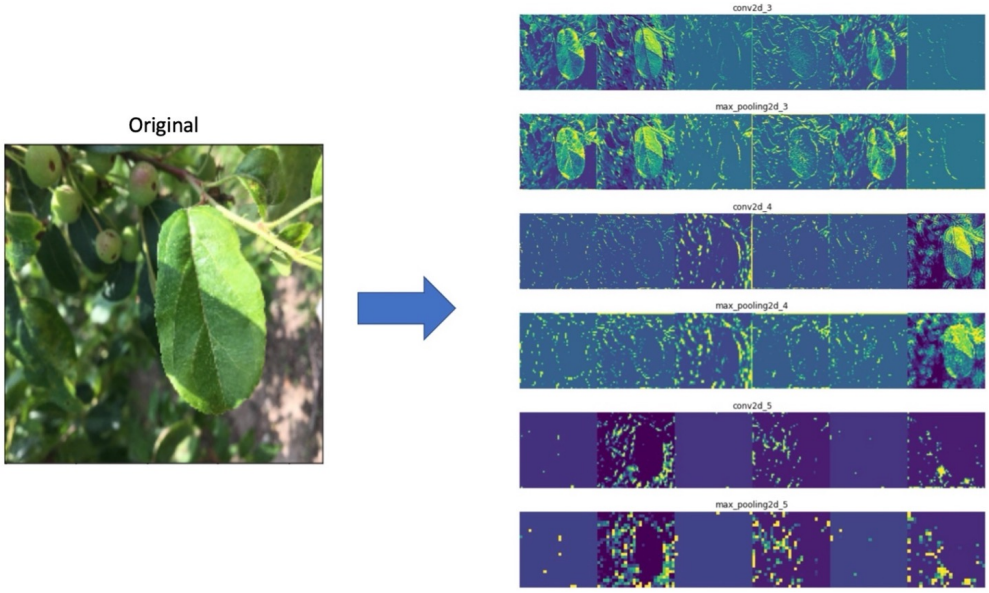
I used a Convolutional Neural Network (CNN) for image classification compared to a Multilayer Perceptron Model (MLP) because of its ability to be efficient, account for the spatial locality, and it's translational invariant. What I mean is that, if an MLP model is given a 224 x 224 pixels input with 3 color channels then that means there are over 150,000 inputs that have to be trained. That is computationally expensive and can cause overfitting due to a large number of variables. Furthermore, MLP does not account for the different positions that an object might be in. It will assume the object's location and try

to adjust its weights to place it there (Stewart, 2019). CNN on the other hand takes all these factors into account through the use of filtering and pooling.

Our model consists of 3 convolutional layers, and for each layer, there is a 3x3 filtering window and a max-pooling of size 2. These settings are chosen because of the input size of our image. If we had computational power, then we can improve the resolution to over 500 pixels and try different settings for each layer. Nevertheless, our model takes in an image that is 224 x 224 pixels scaled down to [0, 1] for ease of computation. Then it goes through the first convolutional layer where it extracts certain feature that makes up the plant leave such as the edges and corners of the image. This feature extraction is done by the filtering and max-pooling window. Essentially, the filtering layer calculates the dot product of its weight (a 3x3 matrix with randomly assigned weights) with the top-left window of the input matrix and continues to do so until the window has slid through the entire matrix. This output will then be used as input for the max-pooling layer where it extracts the pixels that are most concentrated. Similarly for the next 2 convolutional layers, the output of the previous layer is used as input for the next until we reach the end where we have to flatten the matrix to create a 1D feature vector to connect to the fully connected layer to classify the class labels for the plants. Figure 4 shows how the image gets processed as it goes through the different stages of the neural network.

Figure 4

Stages of the Convolutional Layers



As you can see from the image on the right, initially the first layer still keeps the details that make up the plant. But, as you go deeper into the 2nd and 3rd layers, you can see how the image is an abstraction of what it used to be. This is the model trying to perform feature extraction to determine the label for this image.

The model that build the images above consists of the following settings:

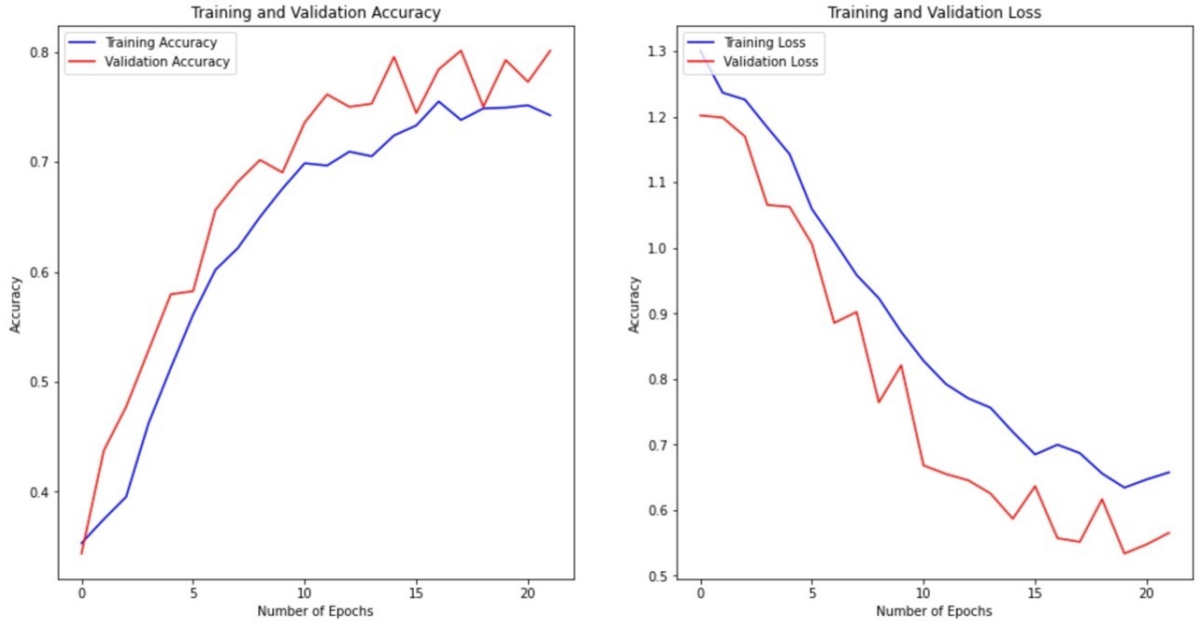
- 30 Training Iterations (Epochs)
- Dropout Layer of 0.02 for regularization to avoid additional overfitting

- Adam optimizer with learning rate of 0.0001
    - I chose this optimizer because it already incorporate mometum as part of gradient descent.
- Relu activation for all the convolutional layers
- Padding on the images
- Accuracy as a metric of performance
- Batch size of 32

Model Performance

As a result of our data pre-processing and analysis, we obtain an 80% accuracy rate. This is relatively decent considering that we have a small-sized data set. Figure 5 shows the performance of the model through each epoch.

Figure 5
Model Performance



As you can see from the images above, our model performs quite poorly in the initial stages of training. This is to be expected since the weights are all randomized in the beginning. But as the model updates its weights through backpropagation, it's learning the pattern that predicts the labels.

Let us take a look at how well our model performs on newly augmented data. Table 1 shows a classification report that captures the accuracy, precision, recall, and F1-score of the 4 labels on the validation set. Note that, during training, the model does not on the validation data, it was only used to evaluate the loss and accuracy of the model.

Table 1
Classification Report

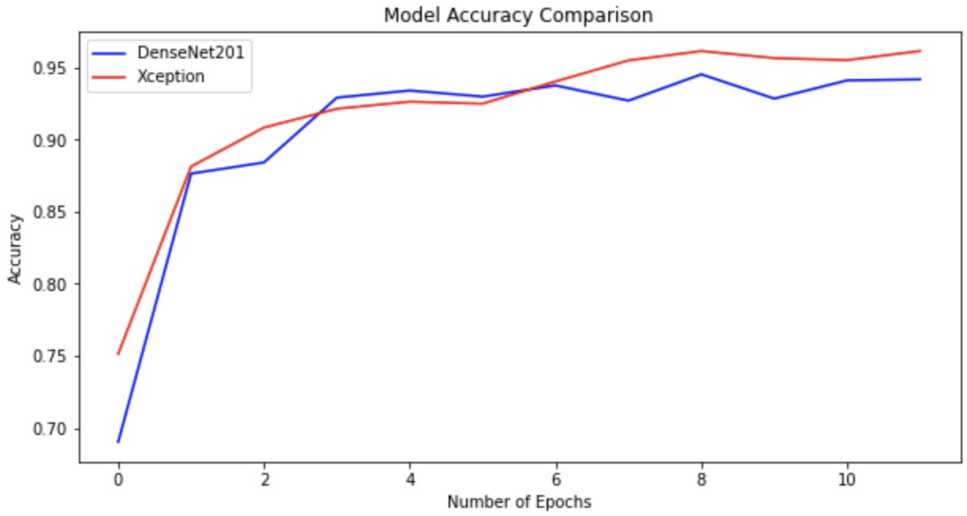
	Precision	**Recall**	**F1-Score**	**Support**
Healthy	0.74	0.61	0.67	111
Multiple Diseases	0.00	0.00	0.00	14
Rust	0.88	0.96	0.92	111
Scab	0.71	0.84	0.77	129
Accuracy			0.78	365
Macro Average	0.58	0.60	0.59	365
Weighted Average	0.74	0.78	0.76	365

Overall, our model does a great job at detecting plants with rust, however, as I mentioned earlier in the report about plants with multiple diseases, our model clearly can't find the pattern for this class label due to the lack of data on it.

To further improve the accuracy of the model, I decided to follow some of the steps that a 2016 research group did to obtain a 99.37% accuracy rate on a similar topic of image recognition of 26 plant diseases across 16 different plant species with over 55,000 samples in a controlled environment (Mohanty et al., 2016). They utilized transfer learning of pre-trained models such as GoogleNet and AlexNet to extend their version of CNN to classify the data. Pre-trained models like GoolgeNet and AlexNet have trained on over a million different images with 1000 class labels. What this means is that we can use the weights from these pre-trained models as a starting point for our model to hopefully improve accuracy. There are currently 18 pre-trained models on Keras API. So to see which model will be the most useful in helping us with our classification problem, we will calculate the performance of the pre-trained models by letting these models classify our data set. The weights of the model with the highest accuracy rate will be used as a starting point for the extended model.

After evaluating the 18 pre-trained models, Xception and DenseNet201 proved to be optimal for this problem. To utilize these models as feature extractors, we need to freeze all the layers when adding additional new layers to the model and then train it. The reason for this is because the new layers have randomized weights and therefore we don't want it to update the weights of the base model (yet). After training, we can unfreeze all the base model's layers and recompile our model so that the inner layers can take account to the new layer's weights. Figure 6 shows the performance of the two models.

Figure 6
Base Model Comparison



As you can see, DenseNet201 and Xception obtain similar results of achieving over 93% accuracy. This is a 14% improvement from the original model which is a big leap in model performance.

Summary

With the recent advances in image recognition and high-quality image capturing mobile devices, we can use different machine learning methods to help farmers detect various diseases across different plant species. This is just the initial stage of the project. There are more steps that could have been taken to further improve the performance of these models. For example, we could have generated more samples using Generative Adversarial Networks, create a segmentation model to remove the background noise and tune the parameters to achieve an even greater result.

After all, our sample size was relatively small, yet we were able to achieve a 94% accuracy after performing various steps such as data preprocessing where we resize, reshape and augment the images to provide more samples and better input for our model and achieving an 80% accuracy. Then we extended our model by embedding a pre-trained model on the top layer so that it can detect important features where we were able to achieve such high accuracy.

Resources

The data and code used in this project are available at the following location:
- Data: https://www.kaggle.com/c/plant-pathology-2020-fgvc7/
- Code: https://github.com/bitngu/Image-Recognition-Plant-Pathology

References

Chemicals in the farm ecosystem. (2019). https://www2.kenyon.edu/projects/farmschool/nature/chem.htm

Elferink, M., & Schierhorn, F. (2016, April 7). *Global demand for food is rising. can we meet it?* https://hbr.org/2016/04/global-demand-for-food-is-rising-can-we-meet-it

Mohanty, S. P., Hughes, D. P., & Salathé, M. (2016, September 22). *Using deep learning for image-based plant disease detection*. https://doi.org/10.3389/fpls.2016.01419

Ritchie, H. (2021, August 21). *Smallholders produce one-third of the world's food, less than half of what many headlines claim*. https://ourworldindata.org/smallholder-food-production

Stewart, M. (2019, February 26). *Simple introduction to convolutional neural networks*. https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac