Plant Disease Detection

Using Supervised Learning

With Deep Neural Network

Syed Mohammad Adil

Hong Shi

**Introduction:**

The Agriculture Sector of almost every country, especially developing countries, is a huge part of the Country’s GDP. Annually, about 20-40% of crops worldwide are lost to pests and diseases. A major factor for this is, it is nearly impossible to keep a check on the condition of each crop, especially when the crop field spans over acres of areas. This led to the need to automate this process so that crop disease can be detected at an early stage where it can be treated properly resulting in a much lower crop yield loss.

To tackle this issue, we decided to develop and train a deep learning model to differentiate between healthy and diseases plants. Through this model we can automate the process of differentiating between healthy and diseases plants and make the process of identifying diseased plants much more efficient. Furthermore, this would also help farmers to reduce cost on manual labour which they hire to detect crop diseases and look after them.

Both team members were involved through the entire project. The process followed by both:

1. Research on image classification.
2. Research on different neural networks to get the best one.
3. Research on resources for data gathering.
4. Deciding which library to use for model development.
5. Implementation of model.
6. Implementation of learning rate scheduler.
7. Making and preparation of Presentation.
8. Writing the Report.

**Related Work**

Previously, a few deep learning models have been trained to differentiate between healthy and diseased plants, however they either focus on just one plants or a few plants and a specific kind of disease. In our model we have developed a model that can first differentiate amongst 14 plants, and then different diseases for each plant.

**Dataset**

The data used by us was gathered from plant village. These images of the plants were taken in controlled environments. The original dataset that was gathered had a little imbalance between classes. The images between the classes ranged from 1600 to approximately 1900. To make the dataset balanced we made each class of the dataset to have 1600 images. Thus to to train the model we had a total 76000 images, containing 14 different plants and 38 classes. Each class was labeled with the plant and disease name. We split 80% of the original dataset into a training set, which contains 60800 images, and the rest 20% into a testing set which consists of 15200 images.

The reason for making the dataset equal was because data imbalance can lead to poor model performance, especially for classes that have fewer images. Furthermore, the model may overfit the majority class, resulting in incorrect predictions.

Finally, we got a training dataset containing 14 plants and 38 diseases, as shown below. There are 1600 images in each plant.

A close-up of a list

Description automatically generated with low confidence

The following images depict what are dataset actually looks like. The dataset contains images of the a single leaf of the plant. For labelling, we added the same class images into the same folder, and the folder name was used as the class label.A close-up of a leaf

Description automatically generated

**Data Preprocessing:**

For image preprocessing we use the ImageDataGenerator from python keras library it to:

1. rescale the image by dividing it by 255 to normalize it between 0 and 1. This was done to help with convergence during training.
2. Randomly rotate the images between the range of -30 and 30 degrees. This was done to increase the robustness and generalization of the model because of the variation in the dataset.

**Methodology:**

**Model Architecture:**

To develop our deep learning model we used the Python tensorflow and keras library. After researching and experimenting on various deep learning models, we decided to create a ResNet 50 model for our project, and to train it using supervised learning.

The ResNet 50 model is one of the best in image classification. It can achieve higher accuracy than conventional neural network models. The reason for that is that it fixed one major issue that most Convolution Neural Networks face, the infamous Vanishing Gradient Problem. This problem occurs when the gradients of the loss function become too small during backpropagation. What this does as that in deeper layers the gradient that updates the training weights of the model becomes negligible, which leads to extremely slow and in some cases stagnated learning of the model.

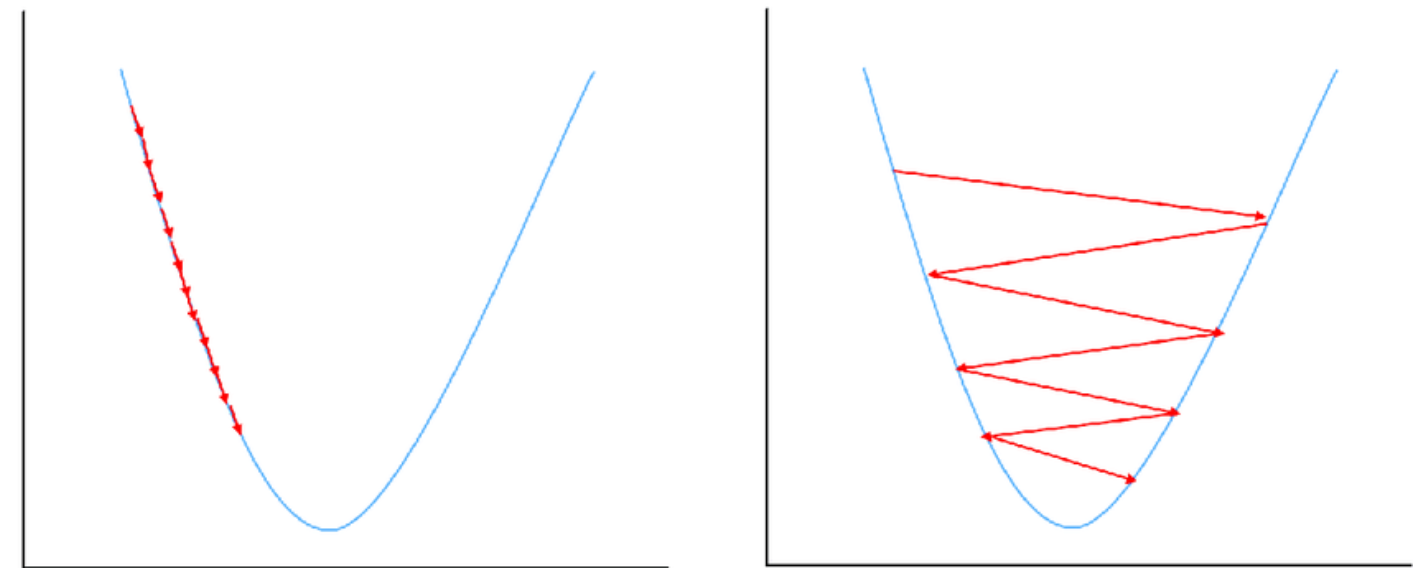
Residual Neural Networks addresses this issue by the use of mutliple residual blocks. A residual block is composed of two or more convolutional layers and a shortcut/skip connection. The basic idea is that the input for each residual block is added to the output of the block, thus the problem of diminishing gradient never occurs.

Our ResNet model contains 50 layers which start with the convolution layer (7x7) followed by batch normalization and ReLU activation function. The output is passed through a max-pooling layer (3x3) and followed by 8 residual blocks. Residual block filters increase gradually from 64 to 512. Residual blocks contain 2 convoluted layers, batch normalization, and ReLU activation. The output of the last residual block is passed through global average pooling and a dense layer. This is done with a SoftMax activation function that produces the output probabilities for each class. The following is the code to make our model.



**Training Process and Hyperparameter Tuning:**

To train the model we first developed a custom learning rate scheduler. The need for this arised when during training we realized that using a low learning rate led to very slow training and it was taking a lot of time, and the second was that when we increased the learning rate, the model accuracy was affected. This is because of high learning rate the model never converged. This can be depicted in the following diagrams.



The left images shows model convergence using a very small learning rate, while the right one depicts training with a high learning rate.

So, to address this issue, we created a learning scheduler which is an extension of the Cossine Annealing. The cosine annealing is a learning rate scheduler where the training starts with a comparatively high learning rate and it is gradually decreased. However, we extended this such that we divided the training process into two parts. In the first part the learning rate is gradually increased from the initial small value to a maximum value and then it is gradually decreased back to a minimum value in the second part.

For this we set the initial learning to be 0.001 which is the default learning rate of the Adam optimizer that we were using. The maximum learning rate was set to 0.01. The model was set to train for 10 epochs, and the epochs were divided in a 30/70 ratio. Thus for the first 3 epochs, the learnin rate was increased from 0.001 to 0.01, and then for the next 7 epochs it was then gradually decreased back to minimum value which was set to be ¼ of the max value, thus it was 0.0004.

The change in learning rate was set to be dependant on the number of epochs and was changed after each batch fed into the model. The batch size was set to be 32. That means at one time 32 images were fed into the model. This also increases training speed.

Due to the learning rate scheduler we were able to train the model and achieve high accuracy and made the training efficient aswell.

**Experiments and Results**

As mentioned above the model was trained for 10 epochs, with a warm-up period of 3 epochs followed by 7 epochs with increasing learning rates. Learning rates were 0.01 for maximum training and 0.0004 for minimum training. In training, 32 batches were used with 1900 steps per epoch. Training and validation datasets were split 80/20, with 80% being collected for training and 20% for validation.

Due to our ResNet model with skip connections and a smart learning rate scheduler, we were able to achieve 99.32% accuracy. This accuracy is the validation accuracy. The output of the result is shown below

A screenshot of a computer

Description automatically generated with medium confidence

As we can observe the learning rate was increased during the first 3 epochs till it reaches the max learning rate and then decreased for the rest of the epochs. The validation accuracy and the loss of the model is depicted below.

A graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of a graph of

Description automatically generated with low confidence

Since we achieved a high accuracy against the validation set it means that the model is performing well on unseen data aswell.

The loss function on validation data has also decreased, which is another testament to the fact that our model is performing well.

The following images depict how we used the test data to see if our model was predicting the correct plant and disease. The actual and predicted label were displayed along with the image to confirm the accuracy of our model.

A picture containing vegetable, leaf, text, plant

Description automatically generated

A close-up of a leaf

Description automatically generated with medium confidence

**Conclusion and Future Work**

Thus, we can conclude that deep neural networks can be useful in accurately identifying between different type of plants and different diseases. This work can be extended such that we can use deep neural networks to train on aerial images. These images can be taken through drones. The benefit of this will be that through the help of drones plant disease will be detected in a much faster and accurate way since a drone can cover land in minutes what would take days for many people to do on their own.

This could really revolutionize how the plant diseases are detected and taken care for and will really decrease the percentage of crop yield loss worldwide.