NOVA IMS Information Management School

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Deep Learning Project

Plant disease detection using Convolutional Neural Network

Work done by: Group 20

Gaurav Luitel, m20210979

Lorena Hahn, m20211302

Mohamed Shamsudeen, m20210707

April, 2022

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação

Universidade Nova de Lisboa

Introduction

This Project aims to build a deep learning model able to classify whether a tomato plant is healthy or diseased based on its leaf image classification using Convolution Neural Network (CNN). Plant Disease Detection is one of the mind-boggling issue that exits when we talk about using Technology in Agriculture. Numerous researches has been done to detect weather a plant is healthy or diseased using Deep Neural Network and new techniques are still being discovered. With the inspiration of successful outcome of deep learning models based on CNN in the field of computer visions like traffic detection, medical image recognition, face recognition etc, in recent years this techniques are also been implemented in real agricultural practices. (Wang, 2021)

The dataset used for this Project has been taken from the Plant- Village Dataset [link]. This dataset consists of 54,305 leaves images of 14 different plant species collected under very controlled environmental conditions. Taking the entire dataset would cost a lot of training time. Considering simplicity and easy to train we decided to chosen 2 classes of “Tomato” plant namely, “Tomato healthy” with 1,591 images and “Tomato target spot” with 1,404 images to build our CNN model.

Approach

Building Convolution Neural Network (CNN)

Our approach is building a Convolutional Neural Network (CNN). CNN are a category of Neural Networks that have proven very effective in areas as classification and image recognition.

In the first step we build a Convolutional Layer. By doing the Convolution, features from images will be extracted. Convolution preserves the spatial relationship between the pixels by suing small squares of input data and learning image features. The input for the Layer is a (32,(3,3)) and therefore 32 is the depth of the first Convolutional Layer. It also shows the number of how many filters are applied for the original input image. The producing output is called the Feature Map. We are also activating the Rectified Linear Unit which is a nonlinear operation. With this operation we are replacing all the negative pixel values in the feature map by zero. With Rectified Linear Unit (ReLU) we are introducing non-linearity in our CNN.

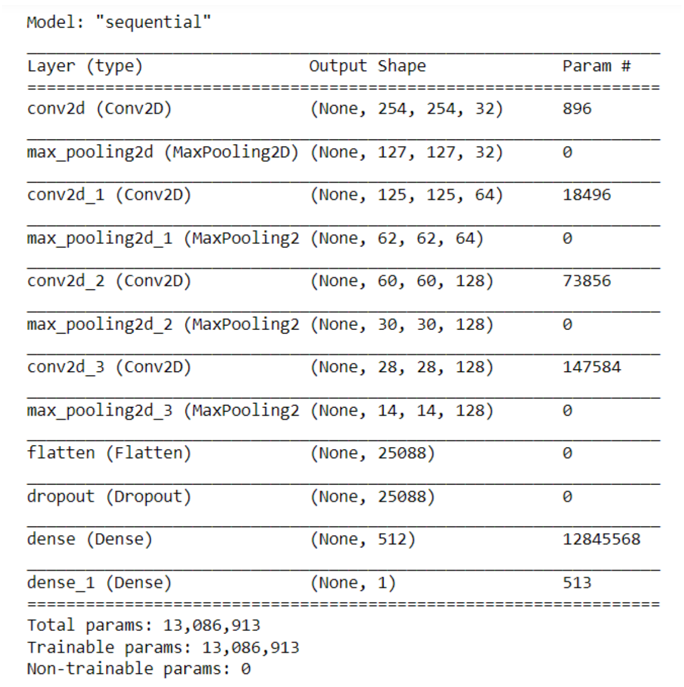
The next step is Pooling, where the dimensionality of each feature map is with retaining the most important information. We are using Max Pooling with defining spatial 2×2 neighborhood and taking the largest element from the rectified feature map in that window. Then we are sliding the 2×2 window by 2 cells and are taking the maximum value in each region. The dimensionality of our feature map will be reduced by this operation.

After that we are adding a second convolutional layer and a MaxPooling layer in it. The second convolutional layer is obtained for the first pooling layer. This means that the first pooling layer acts as a feature map for obtaining the second convolutional layer. This time we are using 64 filters. The output of the second Pooling layer acts as an input to the third Convolutional layer. Then we are adding two more Convolutional layers both with 128 filters and MaxPooling layers alternatively acting as third and a fourth layer of our CNN network. ReLU is applied individually on all of these feature maps.

Together this layer extract useful features from the images, reduce the dimension and introduces non-linearity in our network while aiming to make the feature equivariant to scale and translation. (Arc, 2018) An interesting thing we noticed it that as we go deeper into the network filter begins to have a larger receptive field. That also means that they are able to consider information from a larger area of the original input volume. The next step is the flattening. Flattening translates all multi-dimensional input tensors into a single dimension resulting into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image.

Furthermore, we added a Dropout layer right before fully connected layer, and we are ending the network with a single unit and a sigmoid activation. Dropout prevents overfitting on the training data. Dropout randomly deactivates some neurons of a layer, thus canceling their contribution to the output.

Here we can see how the dimension of the feature maps change with every successive layer.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 254, 254, 32)      896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 127, 127, 32)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 125, 125, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 60, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dropout (Dropout)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 512)               12845568
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 13,086,913
Trainable params: 13,086,913
Non-trainable params: 0
```

Figure 1: Model Summary

The next step is to compile the CNN. Because we run the network with a single sigmoid unit, we are using binary cross entropy as the loss.

Creating Train, Validation and Test Set

The dataset we were using was not available with separate train, validation and test set. The concept of not using the same data set for training, validating and testing also evident in every Neural Network Model similar to traditional Machine Learning techniques. To solve this issue we have used “splitfolder” library (an inbuilt python library) to split our principle image folder into separate train, validation and test folder. The splitting criterion was set to 80% for training, 10% for validating and 10% for testing purpose. After splitting we have got 2395 samples for training the model, 299 samples for validating the model and 301 samples for testing the final model

Data augmentation

The next step is Data augmentation. Since the amount of dataset we have used to built a CNN model is not significantly large there is a huge chance of resulting overly generalized model. To overcome this data augmentation technique is used in CNN. In this technique training examples are increased by producing very likely and close looking images by applying several transformations like rotation, width shift, height shift, shear range, zoom range and horizontal flip. Data augmentation acts as a regularizer and helps reduce overfitting when training machine learning model (Wang, 2021).Our goal for data augmentation is that our model will never see the exact same picture twice. We have used “*ImageDataGenerator*” which read images by performing several random transformations which is available in **keras**. Sample of Augmented images are shown in figure below:

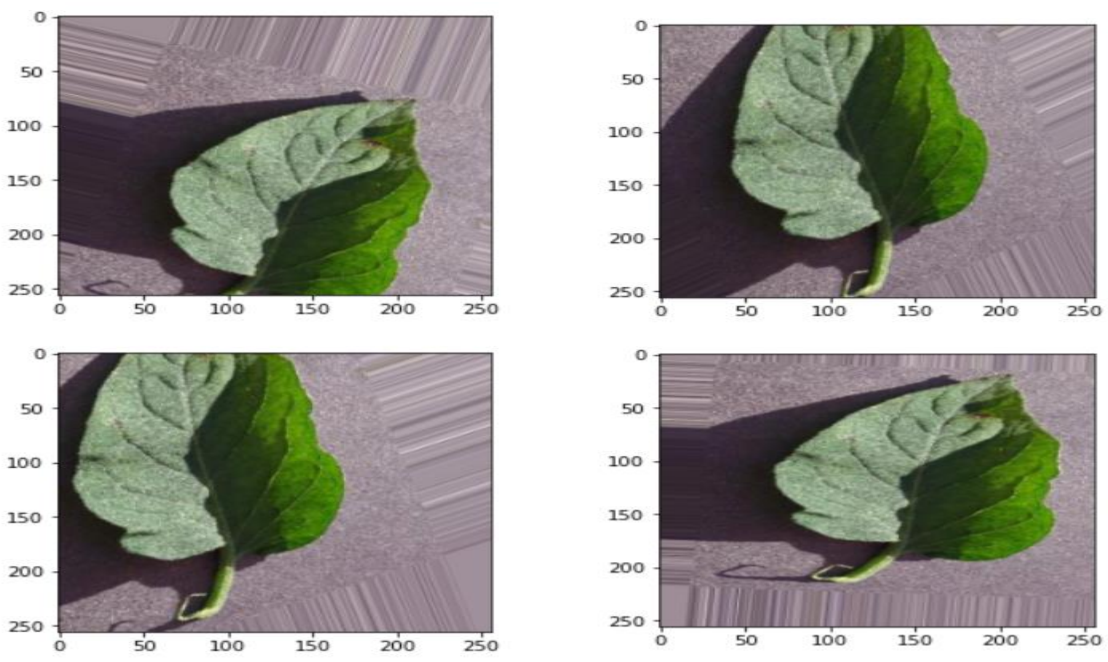


Figure 2: Augmented Images after aplying transformation

Preparation for Training

Only training dataset was subjected to augmentation where as test data was just rescaled around the value [0,1] as a part of preprocessing before feeding into the model. Training, validation and test samples were created by yielding batches of images from the subdirectories in the main data folder using "*flow_from_directory"* provided in Keras. The two classes in our dataset are "*Tomato__healthy*" and "*Tomato__Target_Spot*". 2395 images belonged to training set, 299 under validation and 301 under test set as mentioned earlier in section 2.2. Batch size was set to default size of 32. The class indices are 0 for "*Tomato__Target_Spot*" and 1 for "*Tomato__healthy*". Step per epoch was calculated by dividing total samples by the batch size and found to be 74 for training examples.

Prior to the training process two callbacks are added. One was "*EarlyStopping*", where validation loss was monitored and if it didn't improved the score for 10 consecutive epochs the training would automatically stop. Model was configured for training by setting **Adam** optimizer and learning rate of 0.001. After setting callback model was trained using "*fit_generator*" module for 100 epochs on a batch size of 32. Since we did not set validation step explicitly, it was automatically set to 10 by dividing total samples by batch size.

Model Result and Error Analysis

After successful training up to 24 epochs, training process was stopped because of early stopping criteria set in call back into the model parameter. Training accuracy had better consistency over validation accuracy, obtaining nearly to unity in every epoch, whereas validation accuracy bounced back and forth throughout each epoch. Huge sign of overfitting has been observed regarding validation set, which signifies that how the accuracy depends on the number of epochs in order to detect potential overfitting problem. Trend of training and validation accuracy over number of epoch is represented in the figure below.

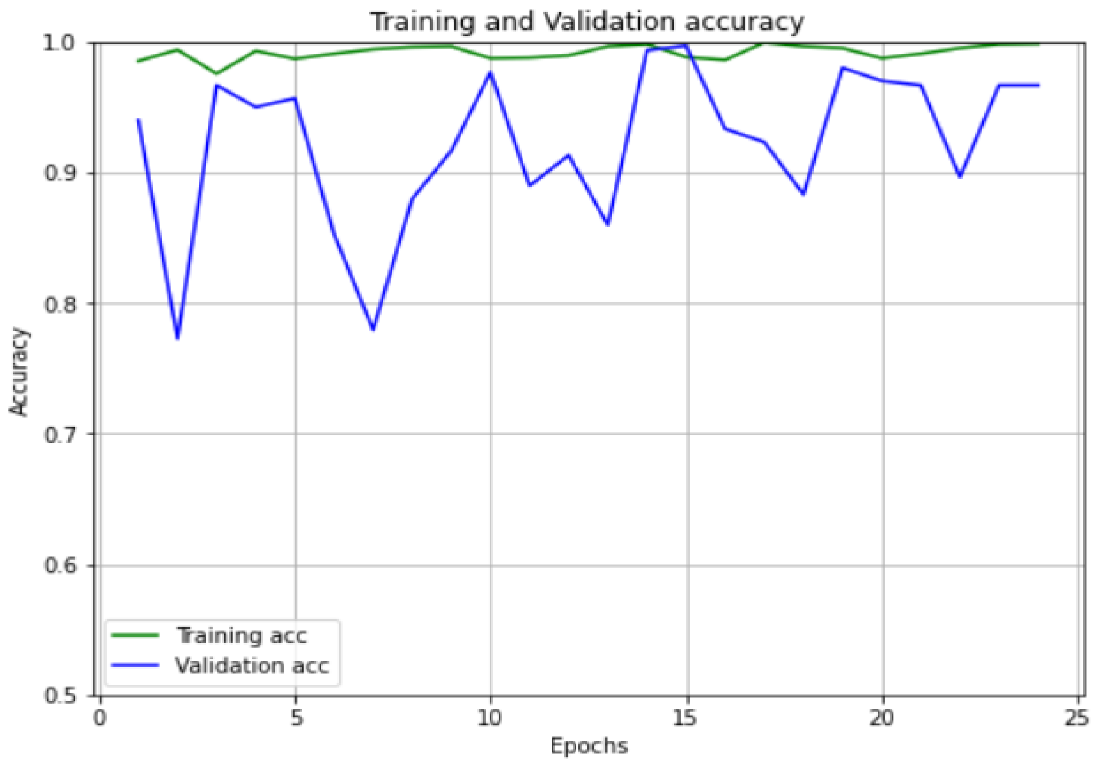


Figure 3 : Training and Validation accuray

Validation accuracy is usually less than training accuracy because training data is something with which the model is already familiar with. The trend shows that even though it oscillates in such an up and down manner, the patter seems converging to top in certain way. One major drawback is we did not take into consideration very large number of epochs without using stopping criteria, which could have resulted in better convergence of validation accuracy. If we see the loss trend, without any doubt we can observe exact opposite nature compared to accuracy score. The surprising result we have obtained is on the test set. The test accuracy we have obtained is **0.99** which is significantly high score. The training and Validation loss is shown in figure below.

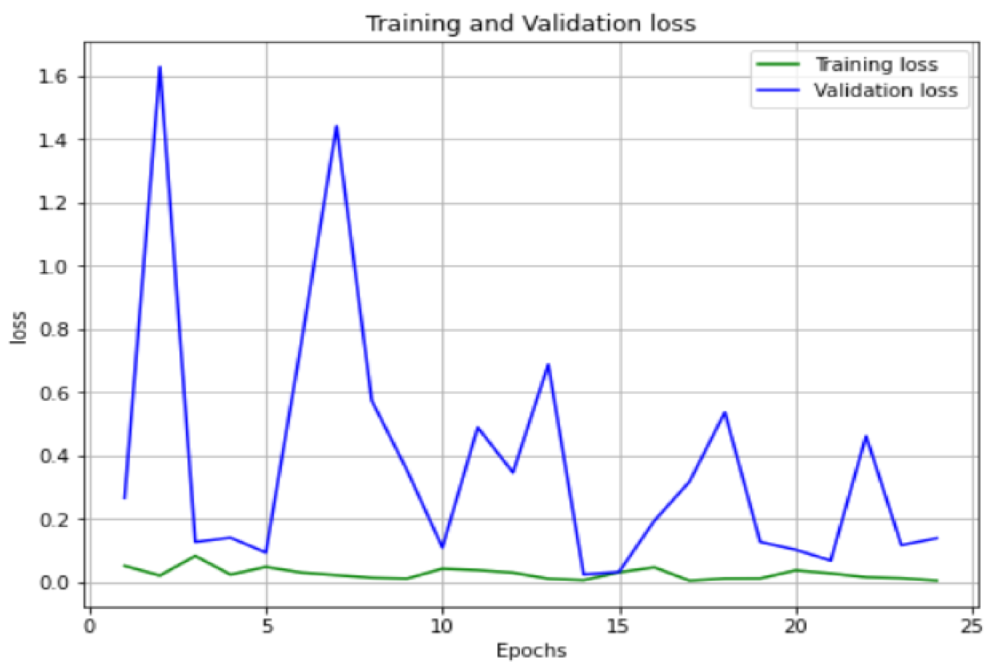


Figure 4 : Training and Validation Loss

Considering test accuracy our model can classify diseased or healthy leaf of Tomato plant with almost 100 percent accuracy. Improvement on validation accuracy can yield much better performance of overall model performance. Error might be potentially because of several reasons. The number of dataset we have trained is still not significantly higher. Since we have tried to built very simple model to get familiar with CNN we haven't taken into consideration many things. Having less training sample on hand we can increase the number of convolutional layer with more number of filters which eventually helps extracting large number of useful features for classifying images. On the other hand varying learning rate could also led to predict under how many epochs accuracy would get stable score.

Conclusion

In this way we have constructed a very simplified version of CNN model with 4 convolutional layers and Maxpooling layers connected with a dense layer having 512 neurons and a single output layer. Considering dropout of 0.2 and implementing data augmentation technique we trained the model taking Adam optimizer with learning rate of 0.001 for 100 epochs on batch size of 32. Model stops at 24 epochs as per stopping criteria achieving training accuracy of 0.9983 and overfitting validation set. Test accuracy of 0.99 was obtained by this model.

Reference

Arc. (2018, 12 25). *https://towardsdatascience.com/convolutional-neural-network.* Retrieved from towardsdatascience.com.

Khoshgoftaar, C. S. (2019, september). A Survery on Image Data Augmentation for Deep learning. *SpringerOpen* .

Sklearn. (2016). *https://contrib.scikit-learn.org/category_encoders/targetencoder.html*. Retrieved from scikit-learn.org.

Wang, J. L. (2021). Plant diseases and pets detection based on deep learning: a review. *BMC* .