Project ReLeaf

Final Report

Presented to
Professor Mark Stamp
CS 271

By

Jason Do

Justin Chang

Katrina Tran

December 2020

I. Introduction

The goal for our project was to be able to classify plant diseases through images of their leaves. This novel application of computer vision is still in its infancy and we tested various machine learning techniques to test their effectiveness on this problem. We chose one of the few popular datasets to work with called PlantVillage and we applied three main algorithms to this problem: Convolutional Neural Networks (CNN), k-Nearest Neighbors (KNN), and Support Vector Machines (SVM).

II. Background

Crop disease has been a continuous problem since agriculture has existed. The diseases affect the cosmetic appearances of the crops, nutrition, and overall yield. The presence of these diseases leads to significant negative economic and ecological ramifications. Methods of treating diseased crops have improved with the invention of pesticides, fertilizer, and other chemicals to boost the health of the crops. However, the improvements in disease detection have largely remained stagnant. Thus, there is a need to detect plant diseases in an accurate and timely manner.

Existing methods for detecting crop pathogens involve scouting for samples in fields and sending them back to labs for more scrutinous testing. This slow process results in a large waste of product and time while the demand for these experts far outweighs the supply. Computer vision (CV) and machine learning techniques are currently being applied to solve this problem. Models are created to accept pictures as input and output a classification of what type of disease a plant is infected with. This approach is novel and researchers are continuously discovering many methods to apply CV to crop disease classification. Accuracy and speed of the model are two metrics that researchers are striving to achieve with their prototypical models. Hence, there is a need for a robust automated solution that does not rely on the limited amount of plant pathology experts.

III. Related Works

A. Plant Disease Diagnosis and Classification by Computer Vision using Statistical Texture Feature Extraction Technique and K Nearest Neighbor Classification [13]

This publication discussed the results of using K-NN as a classification algorithm for images. Popular image features to extract for image classification were also mentioned. We drew inspiration from this article to

use more simple algorithms in hope of achieving high results with little computational overhead. While this article did cover feature extraction using a statistical PCA approach, we did not cover this in our project.

B. The Plant Pathology 2020 challenge dataset to classify foliar disease of apples [14]

This work was the first and most inspirational of the articles we read. The paper functioned as a detailed introduction to the problem of plant disease classification. Detailed background info about the problem was given as well as a reason why a solution was needed. It mentioned the faults of using simple datasets and called for more datasets representative of real world scenarios. While we did not use the more realistic and unrefined images in the plant pathology dataset, it did inspire us to augment our data from PlantVillage.

IV. Dataset: Plant Village

The Plant Village dataset is a set of images collected by a project from Plant Village. It contains images of leaves of different species that are healthy and diseased. All the leaves from this dataset are photographed in a similar environment. We got the raw color dataset from SP Mohanty's GitHub repository Plant Village Dataset [1]. This dataset contains 54, 305 images in total. It has 38 classes - this includes the diseased plant leaves and the healthy version of that species. In some of our models, we used a subset of that dataset on kaggle by Tairu Oluwafemi Emmanuel which contains a total of 20, 638 images in this dataset [2]. The leaves included in this subset are pepper bell bacterial, potato early blight, potato late blight, tomato target spot, tomato mosaic virus, tomato yellow leaf curl virus, tomato arterial spot, tomato early blight, tomato late blight, tomato leaf mold, tomato septoria leaf spot, tomato spider mites, and all of their healthy counterparts.

V. CNNs built from ground up

CNNs are a popular choice for image recognition problems. It involves applying convolutional filters to parts of the images to train the model to recognize features of certain classes. After surveying related works that use CNNs for classifying diseased plant leaves, we noticed that many of the networks require hardware that we do not have access to. We attempted to make a model that has faster inference time and less hardware requirements, without sacrificing too much accuracy.

A. Data Augmentation

There were two sources of augmented data upon which we used. One source came from a subset of the PlantVillage dataset that contained images from each category that were shifted, cropped, and rotated [3]. Another data source came from a subset of the PlantVillage dataset that did not have any inherent augmentation [2]. We applied the augmentation to this dataset manually using a keras library called ImageDataGenerator.

B. Architecture

There were many architecture variations we considered for this project. We run two main variations of our CNN model:

1. CNN model trained from scratch:

   The first CNN model trained was one without any pretrained weights [6]. We used a subset of 200 images from each of the 15 classes of the PlantVillage dataset [2]. Images were converted to an array of integers using the Keras library. All of the images were resized so they have the same pixel dimensions. Ground truth labels were created using the folder title as the label. Labels were converted to one hot vectors using LabelBinarizer from sklearn. The data was then split into training, validation, and test sets using a 6:2:2 split. We called ImageDataGenerator to augment the images before feeding them into our model to allow for more regularization during training. The model was created using a mixture of 2D convolutions, batch normalization, max pooling, and drop out layers. Near the last layers, a flatten layer was added along with more dense layers, and finally a softmax activation layer for the final classification output. Adam was used to allow our learning rate to be dynamic to allow for faster and more accurate convergence. We used the categorical_crossentropy loss function provided by keras. We trained over 25 epochs.

   Results:

   The final training accuracy was around 86%. The validation accuracy varied a lot during training but the later epochs averaged around 65%. The test accuracy was 64%. The run time for this model was extremely slow on my machine. It took around 3.6 hours to train this model. The confusion matrix [12] and accuracy over epochs is provided below in the graphs section.

2. More simple CNN model using more data:

   The approach we took in this model was to use fewer layers and more data [9]. The accuracy from the previous model was lacking and we were looking for ways to boost the accuracy. We included much more training data from a subset of PlantVillage that contained all 38 classes. We also decided to use the full set of augmented images in each category. We removed the drop out layers and only left in the first batch normalization layer. We also removed one convolutional layer and max pooling layer pair.

   Results:

   The data took a little under 2 hours to train for each epoch. We decided to train only using two epochs due to fast convergence and the long duration of each epoch. The training accuracy for the second epoch was around 92% and the validation accuracy was around 91%. The test accuracy was 95% was generated from testing on 50 raw images of each class from the PlantVillage dataset. The results were very impressive considering the little amount of epochs that were trained. The large amount of training data did increase our training time, despite our more simple model. However, the accuracy greatly improved from last time. This demonstrates how important the quantity of data is for CNNs to have accurate results. One explanation for the high test accuracy might have been due to some augmented images being too similar to the raw unedited images being used in the test set.

VI. Transfer Learning Models

Despite the impressive results, the runtime for the models trained from the start were too long. In an attempt to reduce training time, while maintaining high accuracy was our goal. Transfer learning was a method that seems promising towards reducing training time by freezing multiple layers, as well as using weights already fine tuned for general image recognition.

A. Inception model architecture

The Inception v3 model was easily importable through keras and was known for being more efficient than the classical VGG16 architecture. The Inception v3 model achieves computational efficiency through 5 properties: factorized convolutions, smaller convolutions, asymmetric convolutions, classifiers, and grid size reduction. We decided to use this model as our base to add onto for classifying plant diseases through leaves.

B. ImageNet

The model was initialized with weights that were generated by training Inception v3 on the ImageNet dataset. ImageNet is a very large database consisting of over 1.2 million images with 1,000 different object classes. We thought that the Inception v3 model trained on this dataset will generalize well to classify plant diseases.

C. Architectures

1. Softmax Layer Addition

The Inception v3 model with the pretrained ImageNet weights was first imported [10]. Then a dense softmax activation layer was added onto the end of the model. This was an experiment to discover if only fine tuning the classification layer would give us desirable results compared to non transfer learned architectures. Using the subset of PlantVillage [1], we trained our model for 10 epochs.

Results:

90% training accuracy was reached, however the validation accuracy was only around 40%. Test accuracy also was around 40%. The inclusion of a final classification later to train over did not seem to provide high accuracy. However, there was a noticeable increase in training time due to the low amount of parameters that needed to be updated. Training time only took around 1 hour.

2. Addition of more layers

It seemed clear to us that we needed to add more layers to train to fine tune our network to fit our training data better [11]. A global average pooling layer and two dense layers with ReLU activation functions were added. Finally, the softmax classification layer was added for multiclass classification.

Results:

The model was trained over 12 epochs and achieved a training accuracy of 99% and a validation accuracy of 75%. This was already a noticeable improvement over our previous model. The model also only took 30 minutes to run. We used Adam as our optimizer in this case, instead of rmsprop like the last model. A test accuracy of 76% was achieved. In such

a short amount of time, this result was still remarkable. Plots for the results of these experiments are in the graphs section.

VII. k-Nearest Neighbors

The problem we are working to solve is to be able to identify diseased leaves based on their images. The k-Nearest Neighbors algorithm classifies data into classes based on its neighbors. We can use this algorithm to take in the image data and classify them.

A. Raw pixel vs. Histogram

The k-Nearest Neighbors (KNN) model [4] that we used to start off with extracted data in two ways. One way was extracting the raw pixel data from the image. The other extraction was taking the image and creating a color histogram based on the image. Raw pixel data is not good to train KNN on because of how much it would change in different image orientations. Two images of the same leaf can have different raw pixel data if the image was just simply rotated.

Both of these inputs were used in the KNN model to compare. The color histogram input did a lot better than the raw pixel input in all cases. We did do some experimentation on the image sizing of the images before extracting the raw pixel input to test if it would change the accuracy score as well. The resizing of the image does change the score but there is a sweet spot to the resizing.

B. Fine Tuning Hyperparameters

The hyperparameters of KNN would be the number of neighbors for the model and the type of distance metric that the model relies on to figure out the closest neighbors. Since we had seen that using the histogram data would give better results when doing k-nearest neighbors, the fine tuning of the hyperparameters was done only on the histogram data. The scikit-learn library has a grid search tuning algorithm that will test the different hyperparameters combinations to see what gives the best result [5]. We limited the number of neighbors from 1 - 29 and the types of distance metrics to Euclidean, Cosine, and Manhattan [8]. The Euclidean distance metric measures distance between the two points by the Pythagorean theorem. The Cosine similarity distance metric projects the points as vectors and then looks at the angle between the two vectors. The Manhattan

distance metric calculates the sum of the horizontal and vertical distance between two points.

C. Results

The experiments with KNN started by comparing the results from the raw pixel data with the color histogram data. We used the subset of the PlantVillage data for this portion. The training size was 75% while test size 25%. The initial hyperparameters of this KNN was 1 neighbor and the Euclidean metric. The image for the raw pixel data input was resized to 32 x 32 pixels. This gave the KNN model with the raw pixel data input with an accuracy of 49.07% while the color histogram input an accuracy of 86.86%. Experimenting with different number of neighbors (k) gave the following results:

k	Raw pixel accuracy	Color histogram accuracy
1	49.07	86.86
3	48.08	86.90
5	48.16	86.22

As shown, the color histogram input did a lot better than the raw pixel input. Then, to check if the image resizing for the raw pixel data input would make a difference, the next experiment was done by changing the image sizes. The accuracy scores below are all based on the raw pixel input.

k	32 x 32	64 x 64
1	49.07	49.30
3	48.08	48.18
5	48.16	48.37

Based on this, it can be seen the accuracy score did do better when the image was resized a little bigger. However, the accuracy was still lacking and the color histogram input is still far better so the rest of the experiments were focused on that as the input.

Focusing on the color histogram input data for a little bit, we created some graphs based on each image's color histogram data. In one scatter plot, we focused on the data for potatoes. The PlantVillage dataset has data for healthy potatoes, potatoes with early blight, and potatoes with late blight. In

this graph, the diseased potatoes had more divergency. Images of leaves with early blight had more blue while images of leaves with late blight had more green. The healthy leaves were more of a middle ground between the two. Seeing a pattern between the data in each class shows promise for using KNN.

From that, we started more experiments with KNN. We applied the grid search tuning algorithm to figure out the best hyperparameter combination that would give the best accuracy score. Doing this found an accuracy score of 92.81% by using the Manhattan distance metric and 1 neighbor.

We also did some experiments with the test size and applied the grid search tuning algorithm. The 3 experiments pointed out that the most promising hyperparameters for the KNN were using the Manhattan distance metric and 1 neighbor. The results are shown below with the different test sizes and their accuracy scores:

Test Sizes	Runtime (s)	Accuracy
0.2	1354.22	93
0.25	1137.57	92.81
0.5	602.93	91.78

While the 20% test size did do the best, we thought that might be causing overfitting so we stuck with 25%.

After doing all of these experiments, we tried to do it on the larger PlantVillage dataset instead of the subset. With a training size of 75% and test size of 25% and using the Manhattan distance metric with 1 neighbor, the KNN model gave a raw pixel data accuracy score of 65.01% while the color histogram got an accuracy score of 90.70%. The KNN model on the subset data gave the color histogram 92.81%, which is better than the score from the bigger dataset. However, the raw pixel data input had a better accuracy score in the bigger dataset. It had a 59.19% accuracy before; the full PlantVillage dataset gave an increase to a 65.01% accuracy.

We were able to run the grid search tuning algorithm on the bigger dataset and found that the Manhattan distance metric with 1 neighbor was the best combination for the full PlantVillage dataset. The conclusion here is that the KNN did best with the color histogram input with the Manhattan distance metric and 1 neighbor using the data subset at 92.81% accuracy. We had expected the larger dataset to have a better accuracy, but it didn't. This may

be because the dataset is not evenly distributed. There are more images in some classes over others.

VIII. SVM image classification

Support vector machines (SVMs) aim to classify data by finding a separating hyperplane between classes and maximizing the margin for error. By using a kernel function to transform the data, SVMs are able to work in high dimensions with low resource cost. SVMs are very powerful, and they may be useful in solving the problem of classifying diseased plant leaves.

However, at base, SVMs are primarily used for binary class problems, not multi class ones. In order to allow SVMS to work with multiclass problems, we utilize a "one to one" strategy that breaks the multi class problem into multiple binary class problems. Essentially, there are multiple SVMs generating separating hyperplanes between each class versus every other class one at a time [15].

A. Dataset and Implementation

For the SVM classifier, we used the subset of the Plant Village repository [2]. This includes the families of peppers, potatoes, and tomatoes, each with a healthy leaf set and a variety of diseased leaves. For the SVM, we used the SciKit-Learn library's implementation of SVMs with a C of 100 and gamma of 0.1. We also used the RBF kernel and "one versus one" as the decision function. These parameters were picked after using a grid search to determine the best combination of settings [5].

B. Feature Extraction

Extracting appropriate features from images to use in an SVM can be tricky. Usually, these features are obtained from observing the RGB color data from each image. With that in mind, we extracted the mean RGB values of all pixels from each image, as well as the standard deviation. The same features were also extracted from the grayscale version of each image. We also found research on Haralick texture features [6]. These features describe the "texture" of an image, and are calculated using the gray scale version of each image. The features used in our experiment are entropy, correlation, contrast, and inverse differential moments. A vector of these values serve as the features used in our SVM classifier.

C. Results

First we ran the SVM through the entire dataset with a training and testing split of 80% to 20%. This resulted in an accuracy of 83.72%. The confusion matrix can be found in the graph section under VIII.A. This result is somewhat low, which led us to experiment applying SVMs to individual families.

When run solely on pepper images, the SVM scored an accuracy of 94.75%. On solely potatoes, the SVM scored an accuracy of 94.20%. Finally, on solely tomatoes, the SV scored an accuracy of 87.47%. These confusion matrices can be found in the graphs section under VIII.B, VIII.C, and VIIID. All of them were higher than the full data set's accuracy, with peppers and potatoes scoring the highest. There can be several reasons for this.

Firstly, SVMs might be better suited at classifying leaves from within single families. By including too many different plant family leaves during training, the model may learn to detect an average "leaf" from all families and lower its accuracy. Within the same family, leaves can be similar enough such that the SVM focuses on diseased spots instead.

Another possibility may be that SVMs perform better with fewer classes to distinguish. The tomato dataset had the most varied leaf classes, with 9 disease types and one healthy type. Having too many different classes could lead to the SVM not optimizing enough to detect specific diseases.

In conclusion, SVMs performed decently well on classifying the entire dataset, but performed much better when dealing strictly within specific plant families. This can still be useful when detecting diseases within specific species of plants, such as for farmers growing large plots of the same crop suffering from similar diseases.

IX. Future Works

One thing that might be interesting to experiment with is creating a more generalized data set. That is, instead of having separate plant families and diseases as classes, we create a healthy plant dataset and a diseased plant dataset. This can be useful in cases where there are a variety of plants growing in the same area and suffering from similar diseases. Instead of undergoing the time consuming process of dividing specific plant species, data gatherers can simply distinguish between diseased and healthy plants. This may cause some of our classifiers to lose effectiveness while others may perform better.

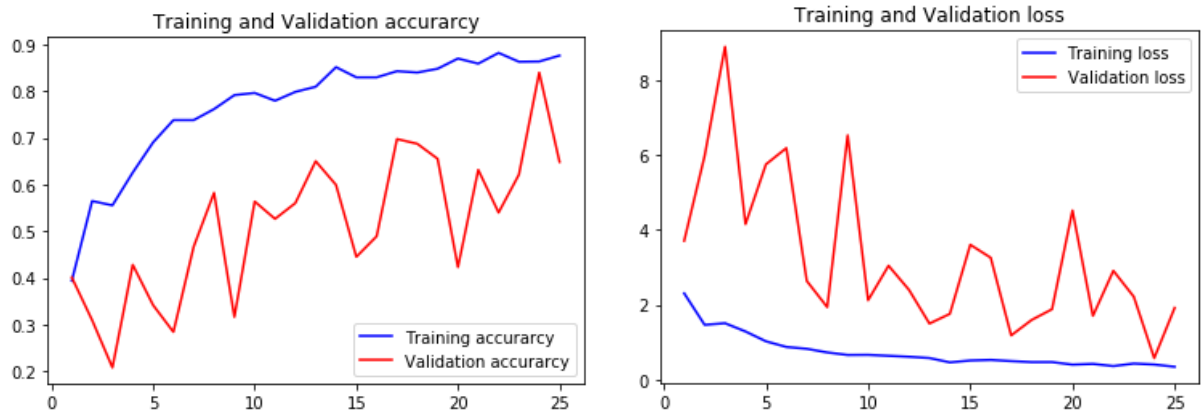
Another interesting experiment would be to create a meta classifier using SVMs as discussed in class. This SVM would take in scores and predictions made from various other classifiers as features and use those to make a prediction. This ensemble of different classifier predictions may prove extra effective in classifying diseases. However, running many different classifiers for each leaf image would cause the runtime to increase dramatically, and would require more resources than what we have access to.

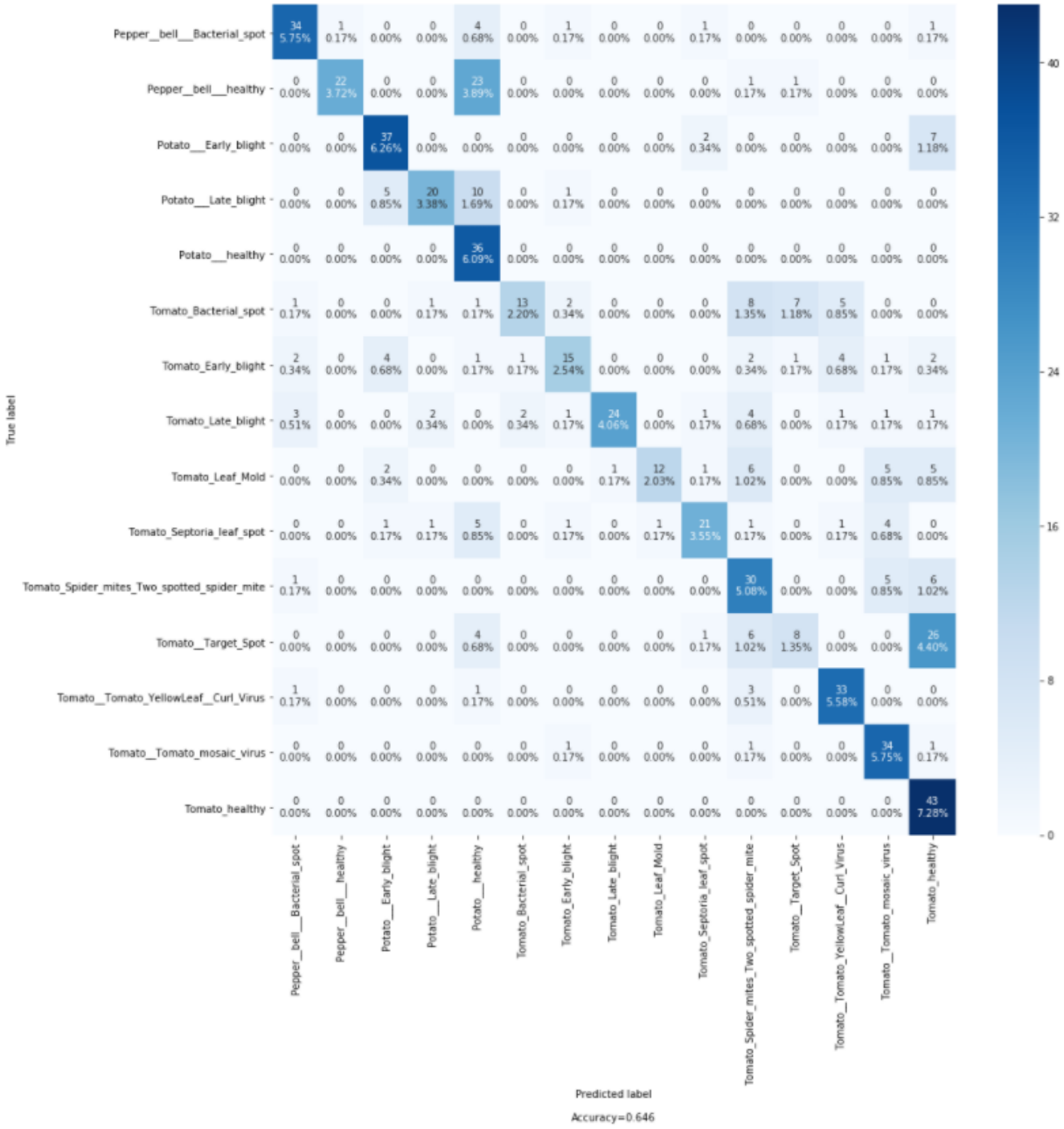
Extreme learning machines would be another interesting concept to apply here. They consist of a single layer Neural Network where the weights of each node are assigned at random each iteration. There is no current library in keras that supports the creation of ELMs, so we would have to implement one from scratch. They are claimed to have faster training times and better generalization than methods that use backpropagation to update weight. One of our limitations for this project was access to powerful hardware and this method could potentially be a great solution.

X. Graphs

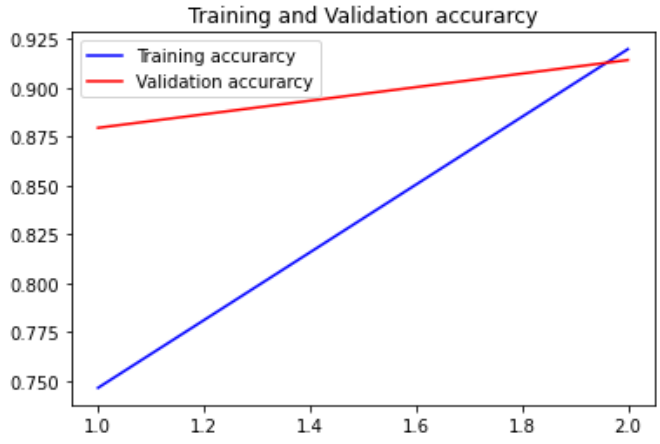
Graphs correspond to heading list numbers.

V.B.1



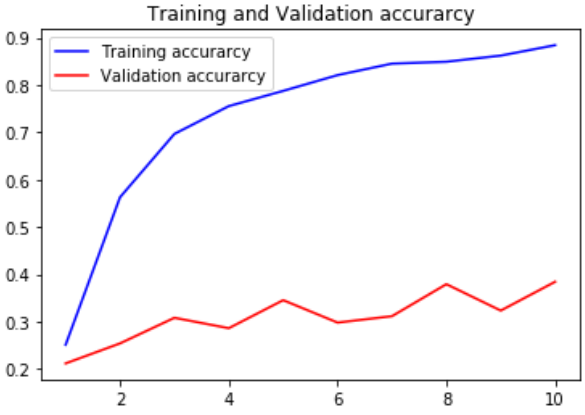

Pepper__bell___Bacterial_spot
Pepper__bell___healthy
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
True label
Predicted label
Accuracy=0.646
40
32
24
16
8
0

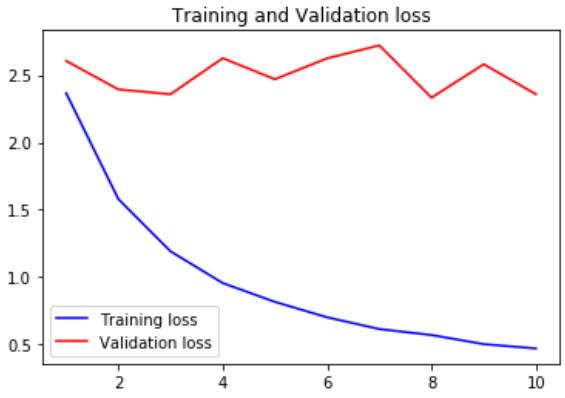
V.B.2.

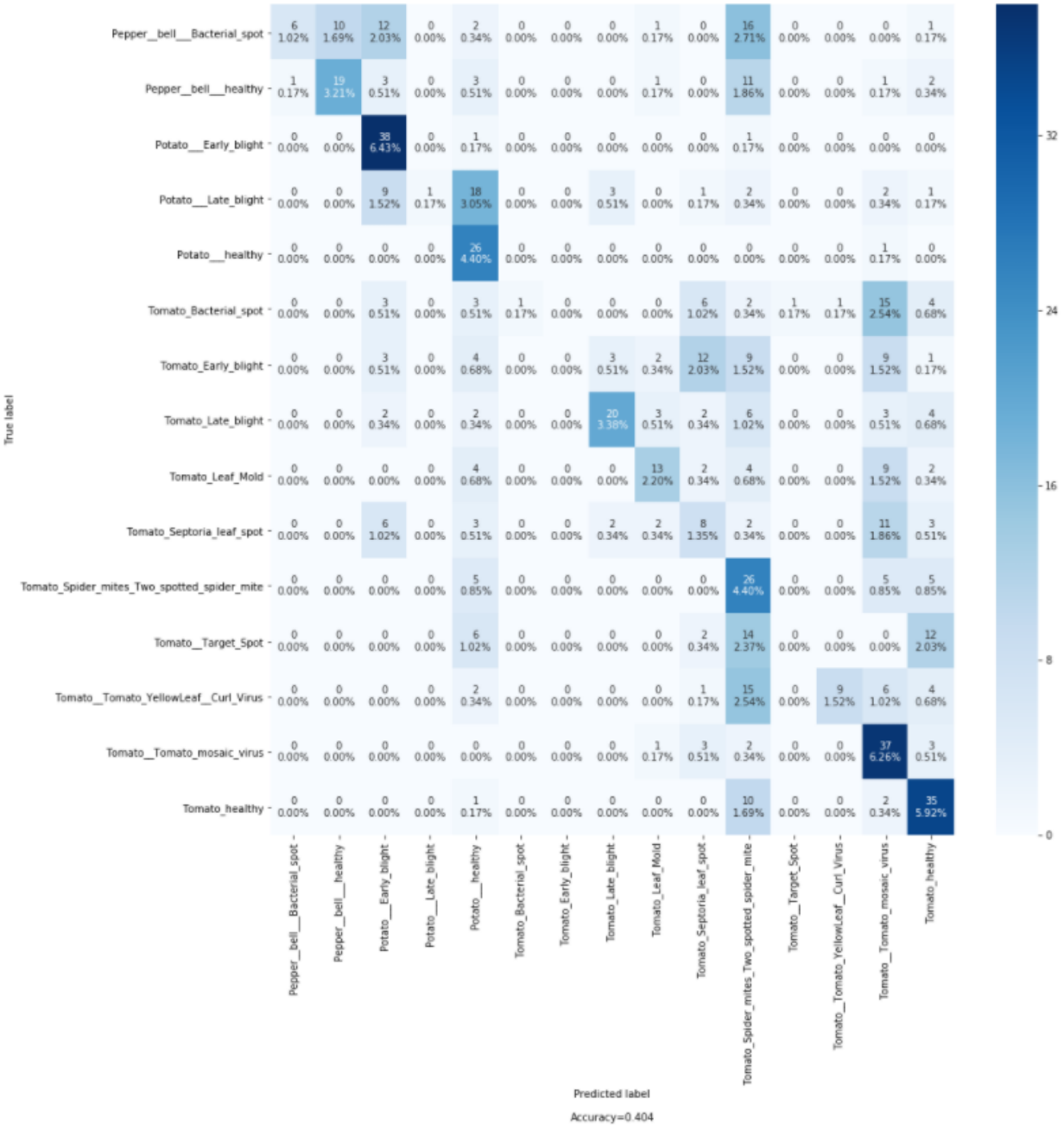



```
[INFO] Calculating model accuracy
19/19 [==============================] - 37s 2s/step - loss: 0.1725 - accuracy: 0.9516
Test Accuracy: 95.15789747238159
```

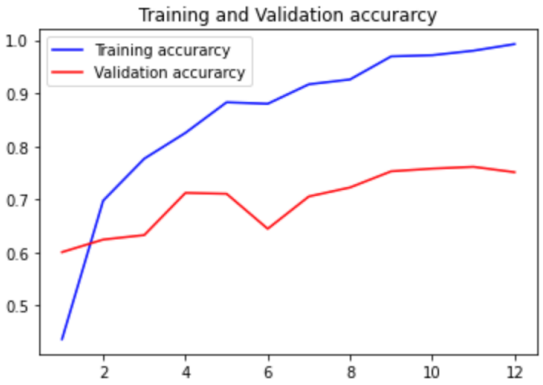
VI.C.1



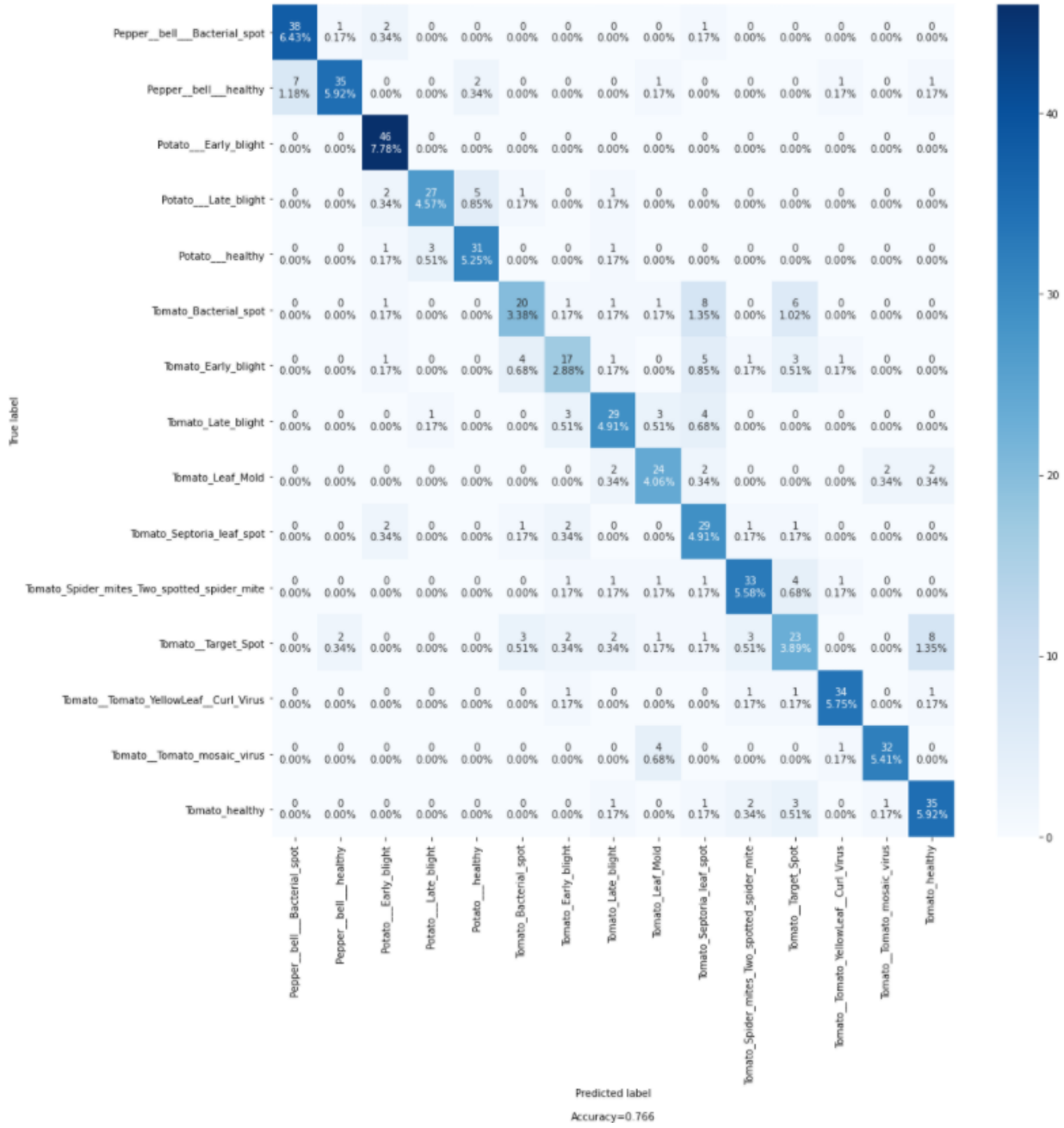



Pepper__bell___Bacterial_spot
Pepper__bell___healthy
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
True label
Predicted label
Accuracy=0.404
32
24
16
8
0

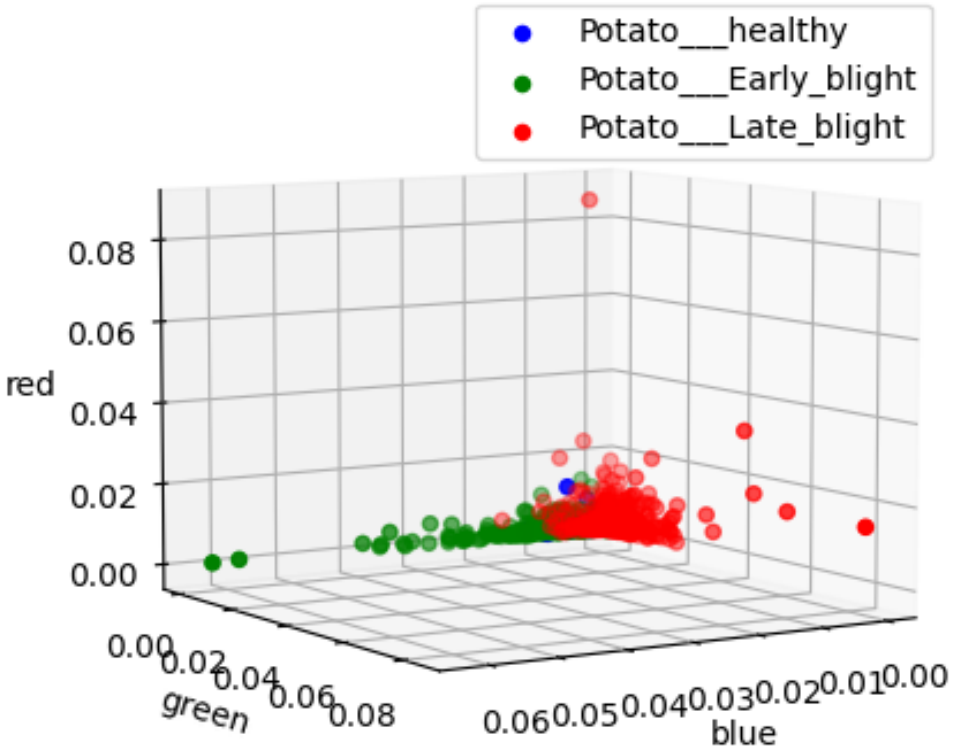
VI.C.2

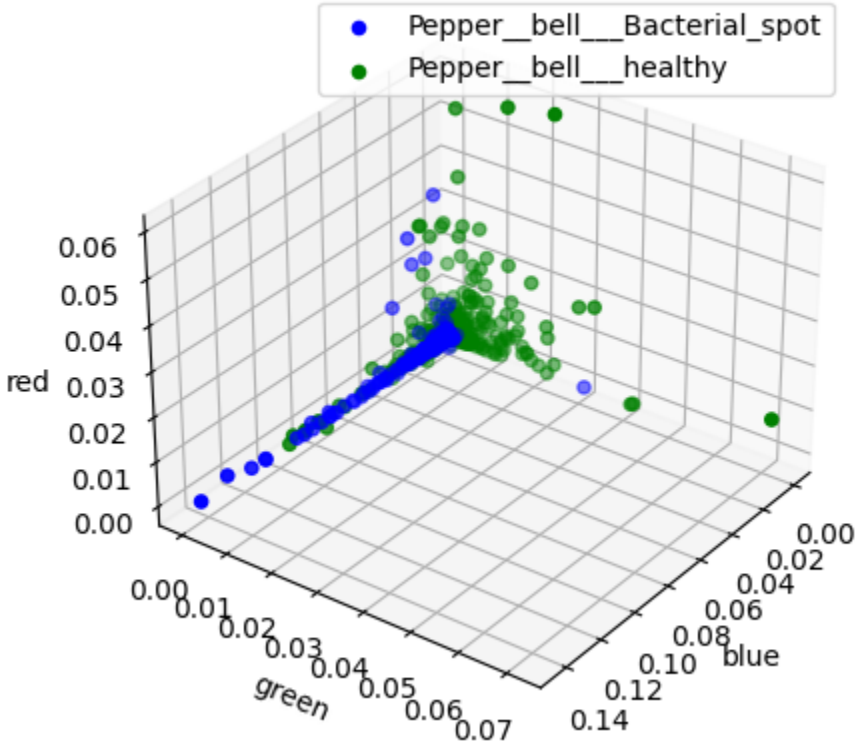

Training and Validation accurarcy
Training accurarcy
Validation accurarcy
1.0
0.9
0.8
0.7
0.6
0.5
2
4
6
8
10
12


Training and Validation loss
Training loss
Validation loss
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
2
4
6
8
10
12

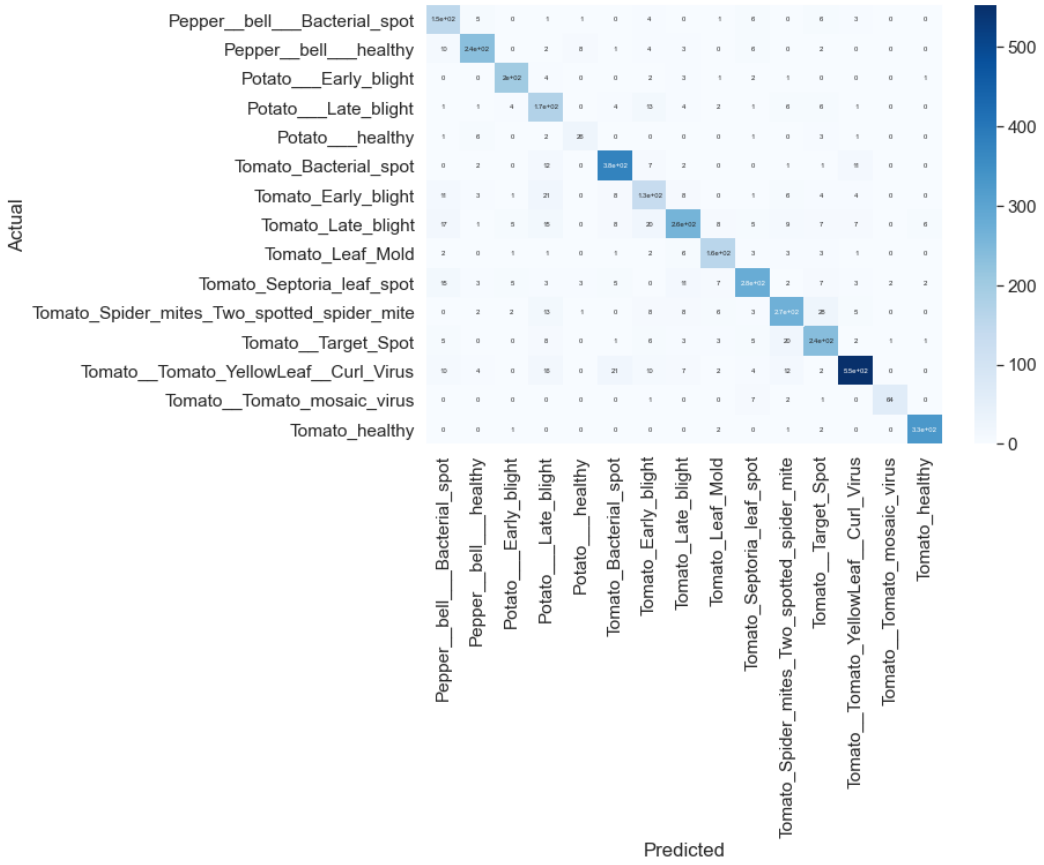

Pepper__bell___Bacterial_spot
Pepper__bell___healthy
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
True label
Predicted label
Accuracy=0.766
40
30
20
10
0

VII.C

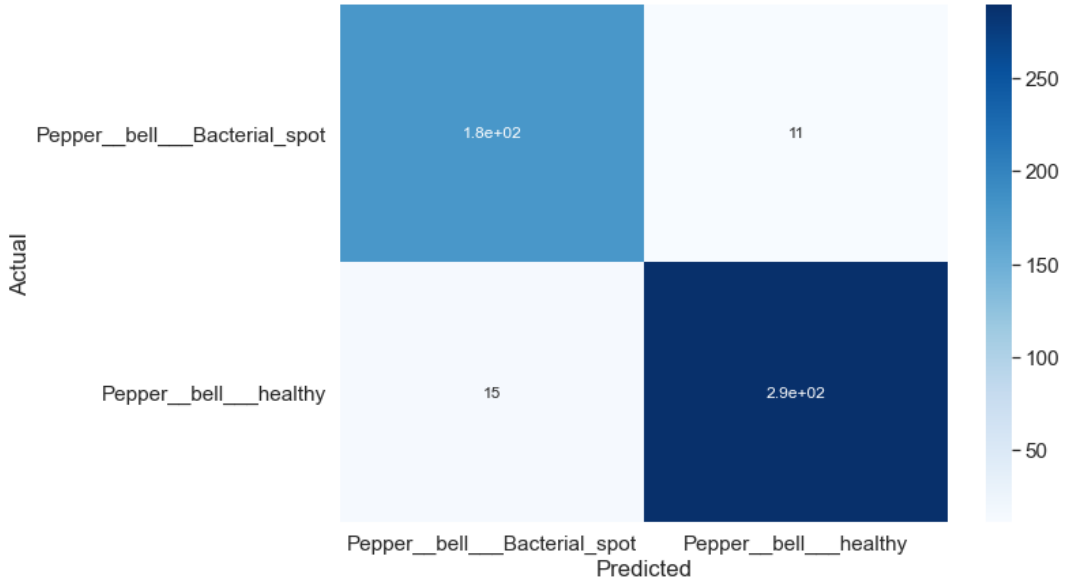

Potato___healthy
Potato___Early_blight
Potato___Late_blight
red
green
blue
0.08
0.06
0.04
0.02
0.00
0.00
0.02
0.04
0.06
0.08
0.06
0.05
0.04
0.03
0.02
0.01
0.00


Pepper__bell___Bacterial_spot
Pepper__bell___healthy
red
green
blue
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14

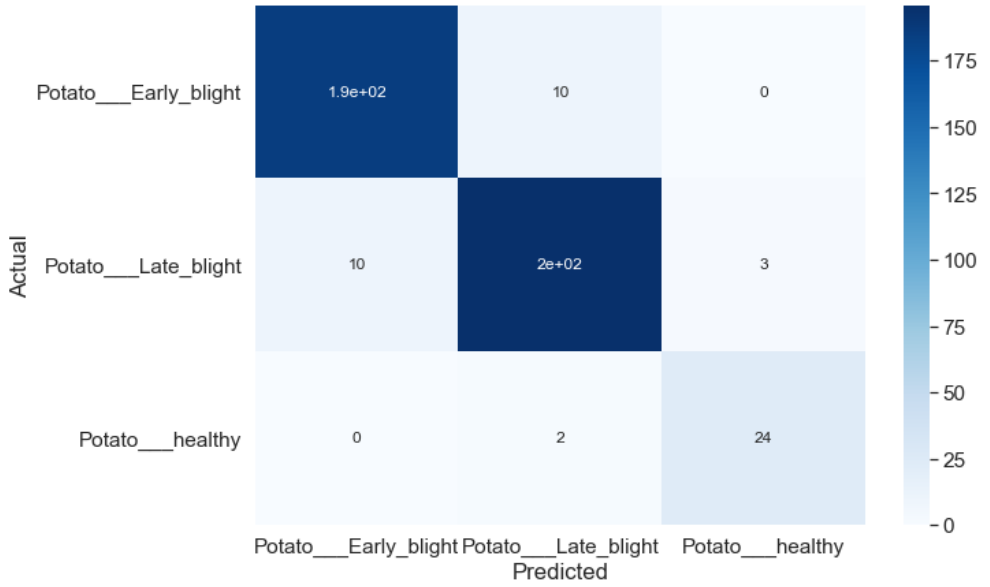
VIII.A


Pepper__bell___Bacterial_spot
Pepper__bell___healthy
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
Actual
Predicted
500
400
300
200
100
0

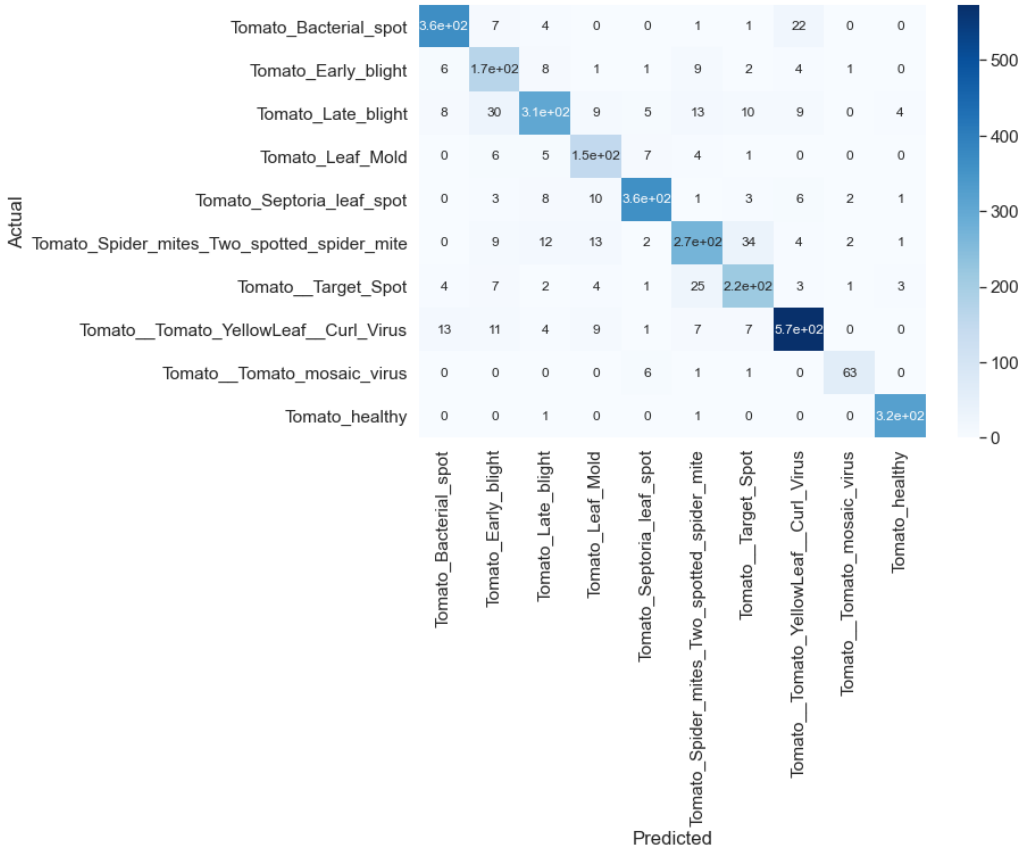
VIII.B


Pepper__bell___Bacterial_spot
Pepper__bell___healthy
1.8e+02
11
15
2.9e+02
Actual
Pepper__bell___Bacterial_spot
Pepper__bell___healthy
Predicted
250
200
150
100
50

VIII.C


Potato___Early_blight
Potato___Late_blight
Potato___healthy
Actual
1.9e+02
10
0
10
2e+02
3
0
2
24
175
150
125
100
75
50
25
0
Potato___Early_blight
Potato___Late_blight
Potato___healthy
Predicted

VIII.D


Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
Actual
3.6e+02 7 4 0 0 1 1 22 0 0
6 1.7e+02 8 1 1 9 2 4 1 0
8 30 3.1e+02 9 5 13 10 9 0 4
0 6 5 1.5e+02 7 4 1 0 0 0
0 3 8 10 3.6e+02 1 3 6 2 1
0 9 12 13 2 2.7e+02 34 4 2 1
4 7 2 4 1 25 2.2e+02 3 1 3
13 11 4 9 1 7 7 5.7e+02 0 0
0 0 0 0 6 1 1 0 63 0
0 0 1 0 0 1 0 0 0 3.2e+02
500
400
300
200
100
0
Tomato_Bacterial_spot
Tomato_Early_blight
Tomato_Late_blight
Tomato_Leaf_Mold
Tomato_Septoria_leaf_spot
Tomato_Spider_mites_Two_spotted_spider_mite
Tomato__Target_Spot
Tomato__Tomato_YellowLeaf__Curl_Virus
Tomato__Tomato_mosaic_virus
Tomato_healthy
Predicted

XI. References

1. Raw color dataset: https://github.com/spMohanty/PlantVillage-Dataset/tree/master/raw/color
2. Subset of kaggle dataset: https://www.kaggle.com/emmarex/plantdisease
3. Augmented subset of PlantVillage: https://www.kaggle.com/vipoooool/new-plant-diseases-dataset
4. Starting k-NN model that we followed: https://www.pyimagesearch.com/2016/08/08/k-nn-classifier-for-image-classification/
5. Fine-tuning the hyperparameters for k-NN: https://www.pyimagesearch.com/2016/08/15/how-to-tune-hyperparameters-with-python-and-scikit-learn/
6. Reference CNN model: https://www.kaggle.com/emmarex/plant-disease-detection-using-keras
7. Haralick texture features: http://murphylab.web.cmu.edu/publications/boland/boland_node26.html
8. Distance metrics: https://scikit-learn.org/stable/modules/metrics.html#metrics
9. Reference CNN model: https://www.kaggle.com/vanvalkenberg/cnn-for-plant-disease-detection-92-val-accuracy/notebook
10. Reference CNN model: https://towardsdatascience.com/how-to-train-your-model-dramatically-faster-9ad063f0f718
11. Reference CNN model: https://towardsdatascience.com/a-comprehensive-hands-on-guide-to-transfer-learning-with-real-world-applications-in-deep-learning-212bf3b2f27a
12. Confusion matrix function: https://github.com/DTrimarchi10/confusion_matrix/blob/master/cf_matrix.py
13. Plant Disease Diagnosis and Classification by Computer Vision using Statistical Texture Feature Extraction Technique and K Nearest Neighbor Classification: https://www.ijeat.org/wp-content/uploads/papers/v9i2/B3849129219.pdf
14. The Plant Pathology 2020 challenge dataset to classify foliar disease of apples: https://arxiv.org/pdf/2004.11958v1.pdf
15. Multi Class SVM Classification: https://www.baeldung.com/cs/svm-multiclass-classification