Artificial Neural Network

Term Project Report

Mehmet Kapson 040150111

Berk Tanrıkulu 040140085

Fatma Rumeysa Önal 040150153

Asel Menekşe 040160151

ELECTRONICS AND COMMUNICATION ENGINEERING
DEPARTMENT

Spring 2020

We are submitting the EHB 420E ARTIFICIAL NEURAL NETWORKS course, Term Project Report entitled as “TOMATO PLANT’S DISEASES CLASSIFICATION”. We hereby confirm that we have realized all stages of the Course Project work by ourselves and we have abided by the ethical rules with respect to academic and professional integrity.

Asel MENEKŞE 040160151	....
Fatma Rumeysa ÖNAL 040150153	....
Mehmet KAPSON 040150111	....
Berk TANRIKULU 040140085	....

TABLE OF CONTENTS

TOMATO PLANT’S DISEASES CLASSIFICATION

1. INTRODUCTION

Agriculture is one of the most important food sources for human beings. Besides, tomato production is one of the most common agricultural execution. According to FAO (the Food and Agriculture Organization of the United Nations) data, in 2018, approximately 182 million tons of tomato were produced in all around the world [1]. Furthermore, again according to FAO data, in 2018, both The USA and Turkey produced approximately 12 million tons of tomato [1]. For producing these amounts of tomato with efficiency, health of tomato plants is vital. If one of the tomato plants gets a disease for instance a virus and no one realizes this, this disease can spread to other plants. Therefore, some tomato plants rot until someone notices the disease. This type of situations not only decreases the efficiency but also increases product prices due to the damage of the production. For detecting diseased plants, machine learning and deep learning applications can be used. Nowadays, deep learning has huge usage for the detection [2].

2. PURPOSE OF PROJECT

In our project, we aimed to classify the tomatoes according to their leaf features in order to detect if the tomato is healthy or diseased with its disease type. It is essential for farmers and gardeners to identify disease type that plants have. If one confuses disease with one an another, s/he cannot apply the correct treatment. Hence, plants will be rot after a short time. Thus, it will be huge financial damage for producer. Besides this situation, early detection of any diseases is crucial for preventing the possible damage by using right treatment. With the help of this project by a pretty good accuracy, anyone can detect the actual disease of a tomato plant on early stages of the disease. Also, this classification can be used in an application of determination of the healthy seeds for food security which could be highly effective and affordable.

3. MODEL ESTIMATION

Since we aimed to classify the tomatoes according to their leaf features in order to detect if the tomato is diseased or not, to be able to do that classification, we have used several libraries such as Pytorch, Numpy etc and GPU to render high-resolution images concurrently quicker. We’ve processed images of leaves of tomatoes to train our model. Data has been separated into three parts as validation part which has a 10%, test part which also has the same as 10% and lastly train part that has the maximum as 80%. MLP is used with 3 hidden layers, and 1712 neurons to train our model.

3.1 Dataset

The dataset is taken from the Kaggle [3]. In the data file, there are 10 folders: 9 folders for 9 disease types of tomato leaf and 1 folder for healthy tomato leaf. Disease classes' names are bacterial spot, early blight, late blight, leaf mold, Septoria leaf spot and spider mites, target spot, mosaic virus, yellow leaf curl virus. There are 16.017 images in total as it can be seen from the figure of a small part of our dataset below.

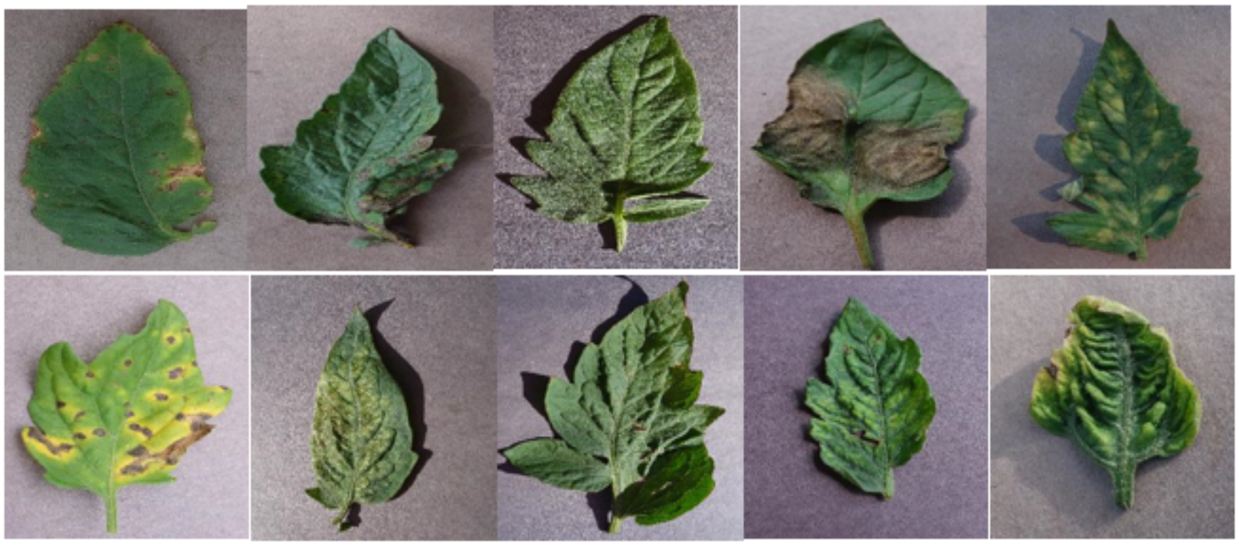


Figure 3.1.1: Samples from Plant Village dataset (For the first row, from left to right; bacterial spot, early blight, healthy, late blight, leaf mold; for the second row, from left to right; Septoria leaf spot, spider mites, target spot, mosaic virus, yellow leaf curl virus).

In the literature, Plant Village dataset was used with different approaches by researchers. Most of them utilized the power of deep learning models. For example, deep learning used for plant disease detection [4] and plant disease identification by fine-tuning deep learning models [5].

3.2 Image Processing

Since the dataset was in a form of images which was non-processable directly, it has been converted to pixel numbers by using image histogram. Image histograms can be considered as a graph or plot, which gives an overall idea about the intensity distribution, contrast, and brightness of an image. It is a plot with pixel values, ranging from 0 to 255, in X-axis and corresponding number of pixels in the image on Y-axis. Besides, in our project the image dimensions have been set to 100x100; moreover, colour space is changed RGB to HSV and then histograms have been constructed with use of required libraries.

Before Hu Moments operation, colour space of images is changed RGB to grayscale. Hu Moments are used for determining the shapes of the images which are defined as weighted average of image pixel intensities. Hu moments are a set of 7 numbers calculated using central moments that are invariant to image transformations [6]. So, let's have background information of the Hu Moments calculation first.

The moment is calculated by the given formula below as:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y)$$

where I(x, y) represents the intensity and, i and j are integers. These moments are often referred to as raw moments to distinguish them from central moments. Above moments depend on the intensity of pixels and their location in the image. So intuitively these moments are capturing some notion of shape [6].

Central moments are very similar to the raw image moments which are known as translation invariant, except that we subtract the centroid from the x and y in the moment formula:

$$\mu_{ij} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x, y) \text{ where } \bar{x} = \frac{M_{10}}{M_{00}} \text{ and } \bar{y} = \frac{M_{01}}{M_{00}}$$

To scale the moment invariant, we need normalized central moments which are both translation and scale invariant, that is calculated as shown below:

$$\eta_{ij} = \frac{\mu_{i,j}}{\mu_{00}^{(i+j)/2+1}}$$

Central moments are translation invariant however it is not enough for shape matching. So that, we need translation, scale and rotation invariant moments which is Hu Moments calculated by given formulas below:

$$h_0 = \eta_{20} + \eta_{02}$$
$$h_1 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2$$
$$h_2 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2$$
$$h_3 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2$$
$$h_4 = (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (3\eta_{21} - \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$
$$h_5 = (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2 + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03})]$$
$$h_6 = (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]$$

The first 6 moments have been proved to be invariant to translation, scale, rotation, and reflection. While the 7th moment's sign changes for image reflection. These hu moments are used as key elements in order to extract the shape of image [6].

In order to express the data in a more calculable and distinguishable way, images were converted into histograms which are expressed as pixel values in a vector space of 512x1 since there are 8 bins that is used.

The vector 512x1 obtained as a result of histogram operation and the 7x1 vector obtained as a result of Hu Moment are concatenated. Thus, 519x1 vector is acquired. This procedure is applied for each image.

3.3 Activation Function

Activation functions are mathematical equations that determine the output of a neural network. The function is attached to each neuron in the network, and determines whether it should be activated or not, based on the relevancy of each neuron's input for the model's prediction. In our project, Sigmoid and ReLU functions have been used as activation functions for the fully connected model which has three hidden layers. Later, it will be explained how the replacement of one of the activation functions affects the model generally.

3.4 Optimizer

We have used Adam optimizer which is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data. SGD Optimizer maintains a single learning rate for all weight updates and the learning rate does not change during training whereas the learning rate is maintained for each network weight and separately adapted as learning unfolds in Adam Optimizer. So that Adam is much more effective to estimate less error.

3.5 Learning Rate

The learning rate is one of the most important parameters that controls how much to change the model in response to the estimated error each time the model weights are updated. Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process. So that we have set the learning rate to 10^{-3} which was adequate for our model estimation.

3.6 Batch Size

Batch size is the number of training samples utilized in one cycle of the iteration and one of the most important hyper parameters to train a dataset. Using a larger batch size to train a model as allows computational speedups. However, it is known that too large batch size might lead to poor generalization which the reason of this issue is not known yet.

3.7 Number of Epochs

Number of epochs is also an important parameter that defines the number times that the learning algorithm will work through the entire training dataset. And it has been arranged to 11 by looking at the validation and training losses. If the validation loss increases, that means overfitting occurs.

3.8 Test Accuracy & Loss

And as a result of the 11 epochs and 3 hidden layers the accuracy has reached to 96% approximately, while the loss decreases to 0.3 in 15 minutes 22 seconds which is the elapsed time. Furthermore, to see how the model accuracy increases step by step and the model loss decreases in more detailed form, the below graphs can be examined.

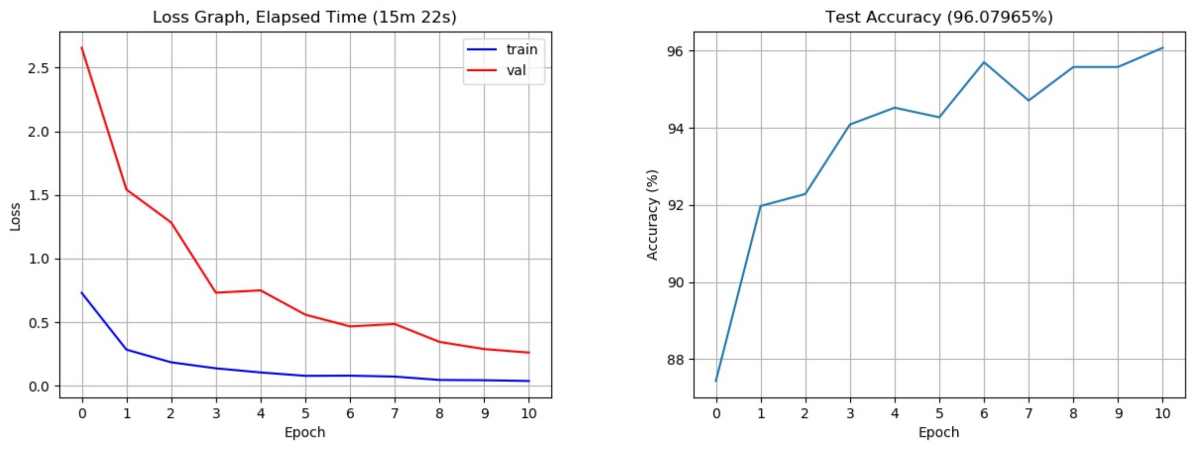


Figure 3.8.1: Test Accuracy Graph and Loss Graph for model with 3 hidden layers.

3.9 Confusion Matrix

Confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. Confusion matrix has been computed to evaluate the accuracy of the classification as given in the figure.

$$Accuracy = \frac{TP + TN}{Total}$$

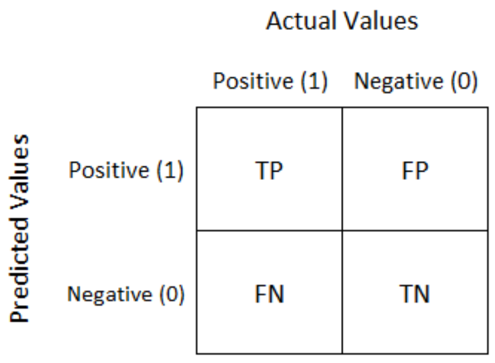
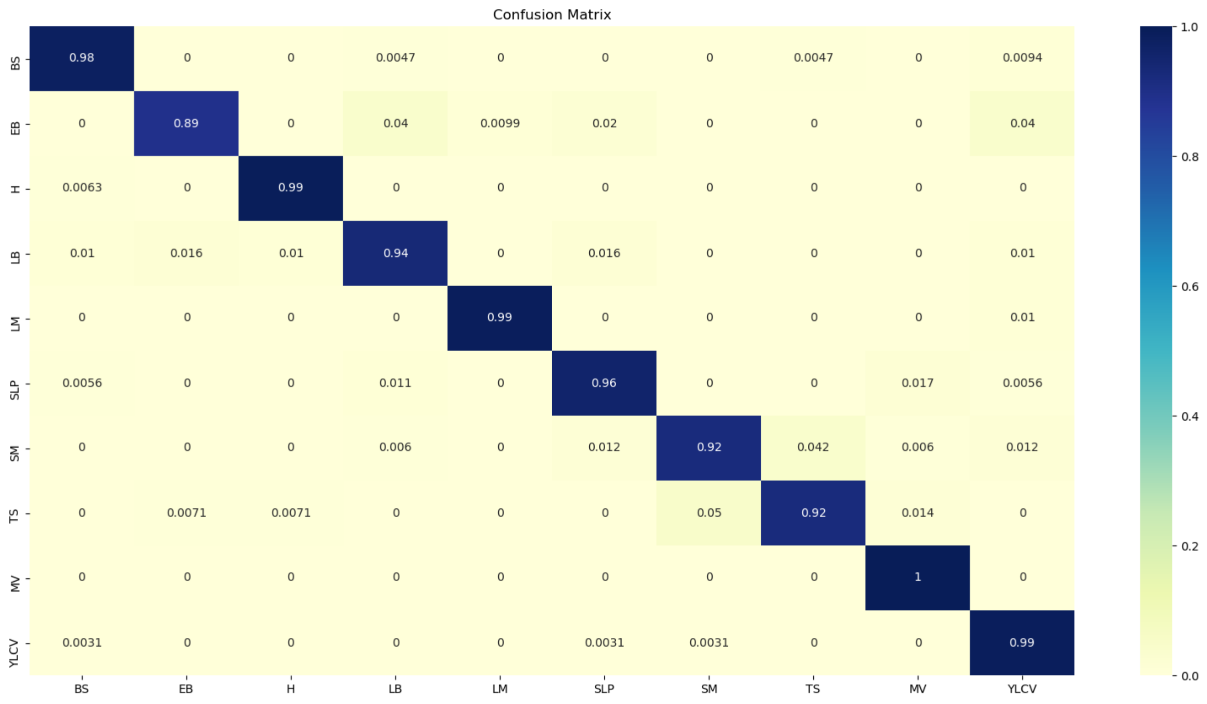


Figure 3.9.1: Accuracy formula from confusion matrix and confusion matrix example.

In our model, accuracy has been evaluated as 96.08% and the confusion matrix has been predicted. Actually, accuracy could be increased by incrementing the number of epochs, yet the elapsed time also increases which is unavoidable. In the figure below, colormap of the confusion matrix can be seen clearly.



	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.98	0	0	0.0047	0	0	0	0.0047	0	0.0094
EB	0	0.89	0	0.04	0.0099	0.02	0	0	0	0.04
H	0.0063	0	0.99	0	0	0	0	0	0	0
LB	0.01	0.016	0.01	0.94	0	0.016	0	0	0	0.01
LM	0	0	0	0	0.99	0	0	0	0	0.01
SLP	0.0056	0	0	0.011	0	0.96	0	0	0.017	0.0056
SM	0	0	0	0.006	0	0.012	0.92	0.042	0.006	0.012
TS	0	0.0071	0.0071	0	0	0	0.05	0.92	0.014	0
MV	0	0	0	0	0	0	0	0	1	0
YLCV	0.0031	0	0	0	0	0.0031	0.0031	0	0	0.99

Figure 3.9.2: Confusion matrix for model with 3 hidden layers.

In all of confusion matrices that report has, class names' first letters are used as x and y labels.

4. COMPARISON ACCORDING TO SOME PARAMETERS

4.1 Batch Size

Batch size has been increased to 64 to see how the accuracy changes. And as result of the increment, the test accuracy has scaled a bit up to 96.20%. Moreover, the elapsed time for training has increased to 18 minutes 53 seconds as it can be seen from the figures below. Increasing batch size is expected to decrease the training elapsed time. However, in this project, it increased elapsed time as 3 mins and 31 secs. When batch size is increased, more data is used as input at the same iteration. Thus, number of iterations for one epoch decreases, which is dependent for number of data in a dataset and batch size. This means that weights are fewer times updated. In this project, images are processed to extract features while taking the dataset. Different images which are amount of batch size are taken randomly and processed for each iteration. This situation might be the possible reason for the increase in elapsed time. Furthermore, test accuracy is nearly the same as default which is batch size 32.

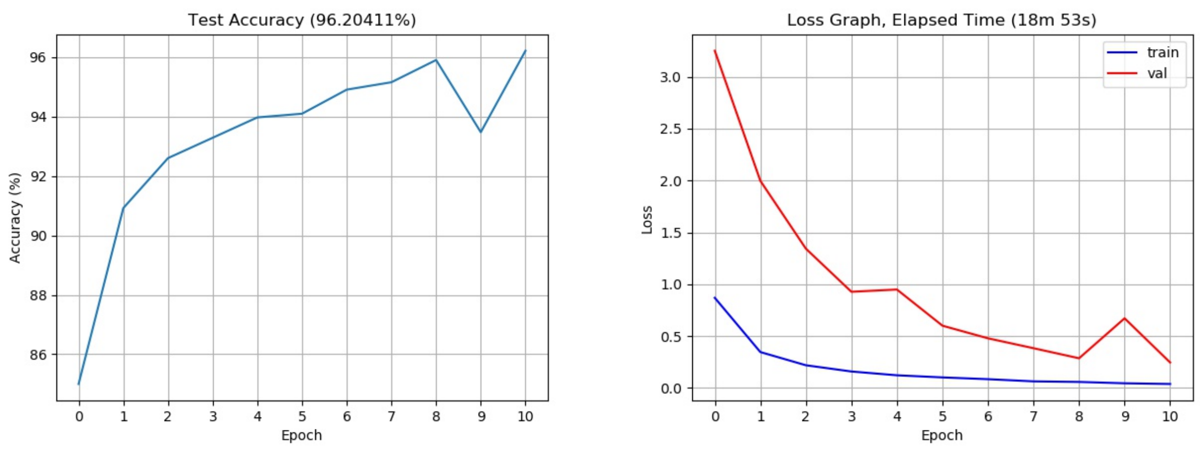
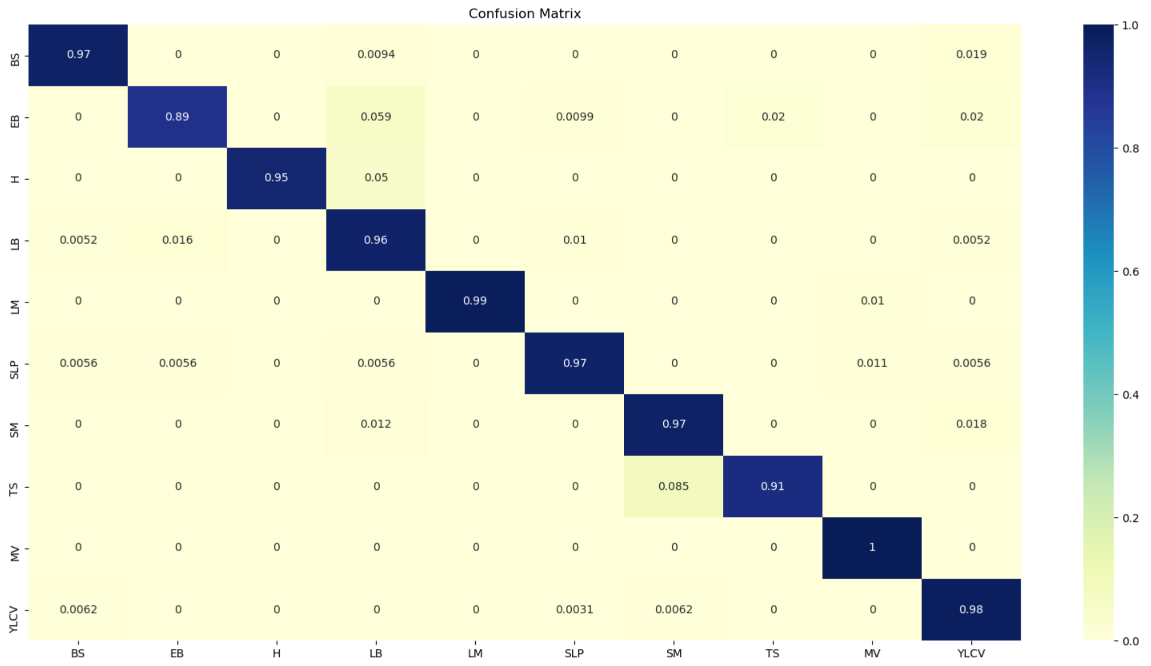


Figure 4.1.1: Test Accuracy Graph and Loss Graph for batch size increment to 64.



	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.97	0	0	0.0094	0	0	0	0	0	0.019
EB	0	0.89	0	0.059	0	0.0099	0	0.02	0	0.02
H	0	0	0.95	0.05	0	0	0	0	0	0
LB	0.0052	0.016	0	0.96	0	0.01	0	0	0	0.0052
LM	0	0	0	0	0.99	0	0	0	0.01	0
SLP	0.0056	0.0056	0	0.0056	0	0.97	0	0	0.011	0.0056
SM	0	0	0	0.012	0	0	0.97	0	0	0.018
TS	0	0	0	0	0	0	0.085	0.91	0	0
MV	0	0	0	0	0	0	0	0	1	0
YLCV	0.0062	0	0	0	0	0.0031	0.0062	0	0	0.98

Figure 4.1.2: Confusion matrix for batch size increment to 64.

Actually, in researches about the effect of batch size, there are different results. In a research worked on two different datasets using 1024 and 16 batch sizes for the two datasets, the best accuracies has been obtained for 1024 batch sizes but, according to another research, the batch size does not affect the performance of the network yet, it decreases elapsed time to convergence for larger batch sizes [7]. So, it can be said that batch size is an important parameter yet not effective alone as the other parameters.

4.2 Activation Function

In our main training code, there were two different functions as ReLU and Sigmoid that are used for the activation function as mentioned before. And to compare the accuracy by changing the function, all functions are set to Sigmoid. As a result, the accuracy has gone down since Sigmoid function has a back-propagation error unlike the ReLU function which is also faster than Sigmoid [8]. The graphs of accuracy history of training and testing and history of loss functions are seen in the figure below.

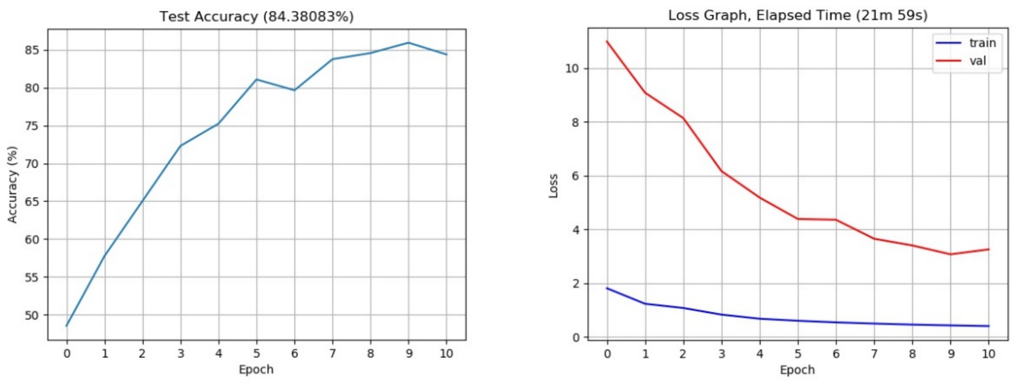


Figure 4.2.1: Test Accuracy Graph and Loss Graph for change of the activation function as Sigmoid.

The graphs of Sigmoid and ReLU functions are shown in the figure below. The ReLU function has two basic advantages over the Sigmoid function. First, its gradient is constant, and the probability of vanishing gradient is lower. As seen in the figure above, absolute values of gradient of the Sigmoid function at maximum and minimum values are exceedingly small, so its gradient becomes nearly zero. Second, when z<0, units activated by ReLU are not activated so, the computation process becomes more easily. This is called as sparsity.

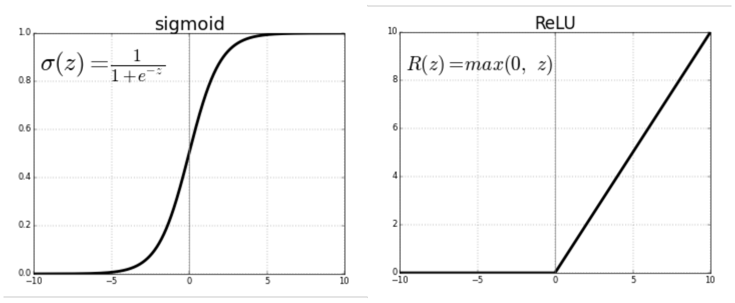


Figure 4.2.2: Sigmoid and ReLU [8].

4.3 Optimizer

As a comparison option we have tried to use SGD Optimizer instead of Adam Optimizer which has a different working principle. Stochastic Gradient Descent (SGD) is an optimizer which is an iterative method for optimizing an objective function with suitable smoothness properties. It can be regarded as a stochastic approximation of gradient descent optimization since it replaces the actual gradient (calculated from the entire dataset) by an estimate thereof (calculated from a randomly selected subset of the data). And as the loss function 'binary cross entropy' has been used which is given below [9].

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

SGD Optimizer maintains a single learning rate for all weight updates and the learning rate does not change during training whereas the learning rate is maintained for each network weight and separately adapted as learning unfolds in Adam Optimizer. So that it is less effective to estimate less error.

As expected, test accuracy has scaled down to 48.16% and model loss scaled up to 12, also the confusion matrix is distributed very badly which can be also seen below.

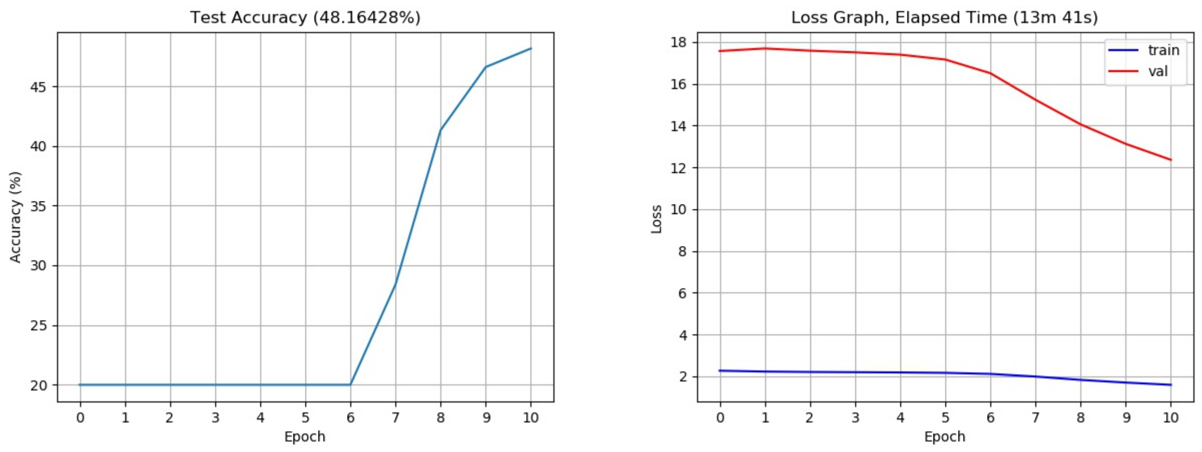


Figure 4.3.1: Test Accuracy Graph and Loss Graph for using SGD instead of Adam optimizer.

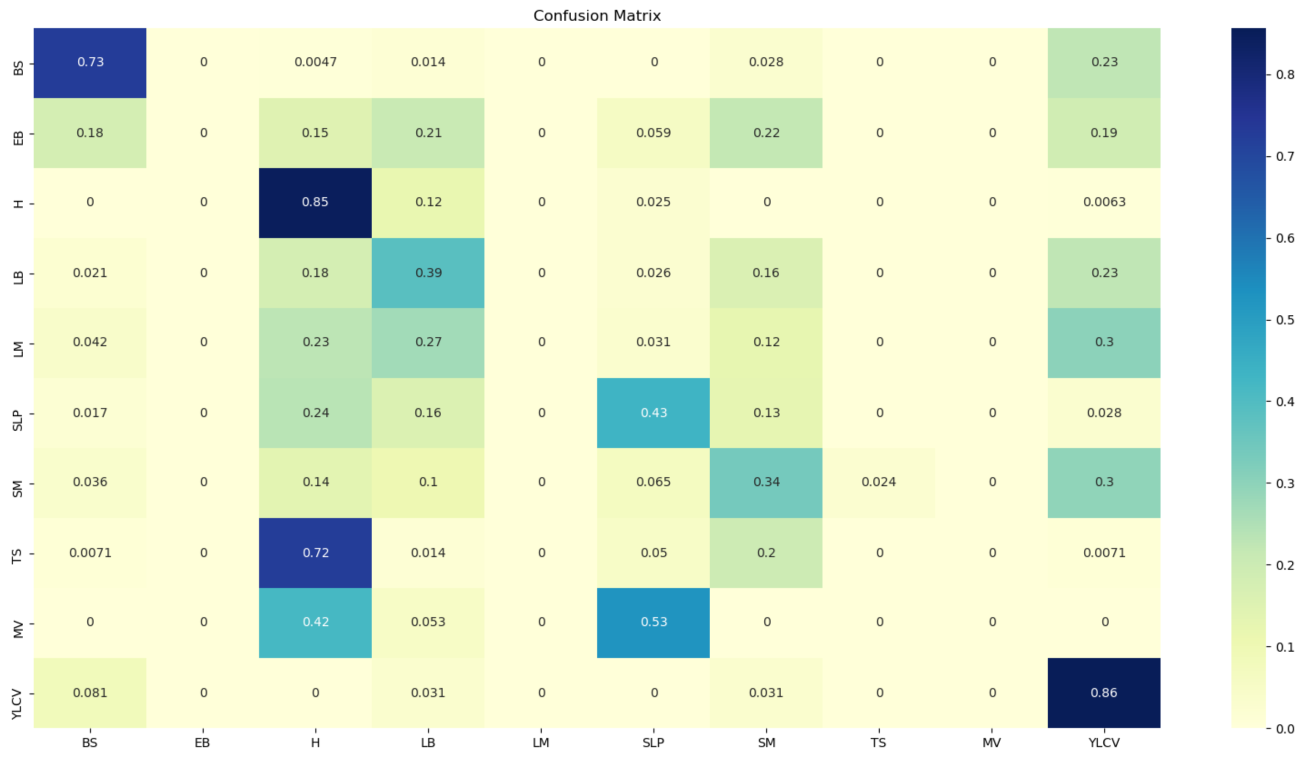


Figure 4.3.2: Confusion matrix for using SGD instead of Adam optimizer.

4.4 Number of Neurons

Neuron, also called a node or Perceptron, is a computational unit that has one or more weighted input connections, a transfer function for combining the inputs, and an output connection. Neurons are then organized into layers to comprise a network.

As the number of neurons increases, model gets more adaptive, so it can learn smaller details. However, the model gets more affected to over-fitting and so the generalization of our classification model can also decrease. Hence, to see this effect on our model, neuron numbers in all layers have been decreased from 1000, 512, 200 to 700, 350, 100, respectively to see how the accuracy, loss and the execution time will change.

Consequently, the accuracy slightly decreased to 95.46% whereas the model loss was a bit higher than our estimation as 0.5, and the elapsed time which is 13 minutes 31 seconds was also a little less than our actual models' elapsed time. These results are expected because computational unit number is less than our actual model. This difference causes to decrease the model's capacity of learning as well as the elapsed time. And all the process can be seen from the given figures below.

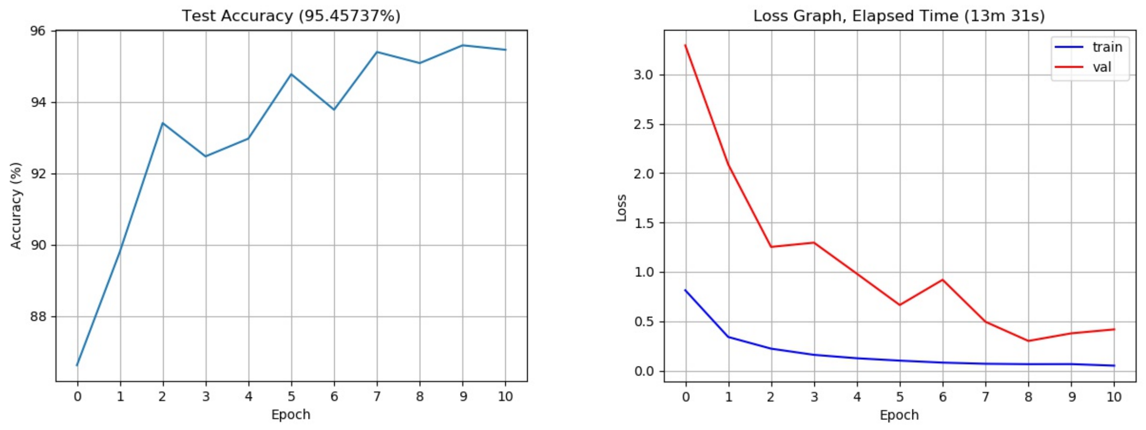
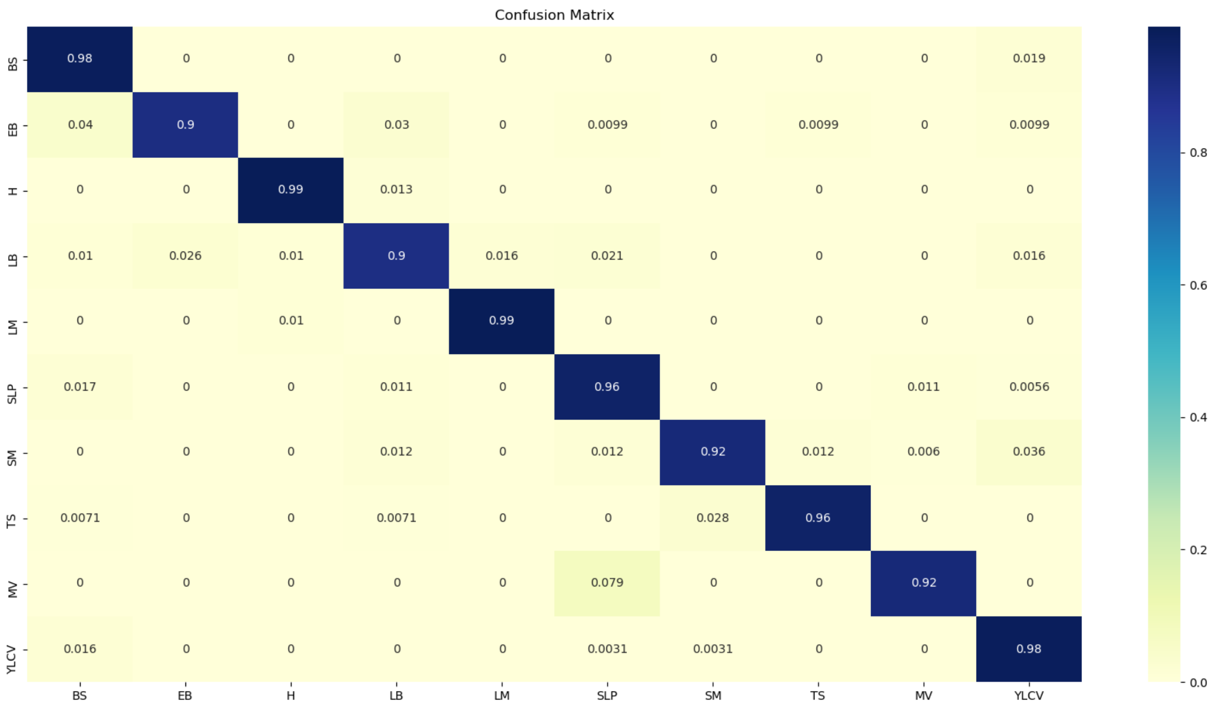


Figure 4.4.1: Test Accuracy Graph and Loss Graph for usage of less neurons.



	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.98	0	0	0	0	0	0	0	0	0.019
EB	0.04	0.9	0	0.03	0	0.0099	0	0.0099	0	0.0099
H	0	0	0.99	0.013	0	0	0	0	0	0
LB	0.01	0.026	0.01	0.9	0.016	0.021	0	0	0	0.016
LM	0	0	0.01	0	0.99	0	0	0	0	0
SLP	0.017	0	0	0.011	0	0.96	0	0	0.011	0.0056
SM	0	0	0	0.012	0	0.012	0.92	0.012	0.006	0.036
TS	0.0071	0	0	0.0071	0	0	0.028	0.96	0	0
MV	0	0	0	0	0	0.079	0	0	0.92	0
YLCV	0.016	0	0	0	0	0.0031	0.0031	0	0	0.98

Figure 4.4.2: Confusion matrix for usage of less neurons.

4.5 Number of Hidden Layers

In artificial neural networks, hidden layers are required if and only if the data must be separated non-linearly. In our model, we have used three hidden layers. To see the effect of the number of layers, we have decreased the hidden layer number to two. As a result, accuracy have been decreased very slightly to 95.77% and the model loss almost stayed the same. Besides, the elapsed time has the most obvious decrease to 13 minutes 56 seconds since hidden layer affects the time reasonably. Results are obtained as below because of the same reason as explained in number of neurons section. To see the results in a clearer way the below figures can be examined.

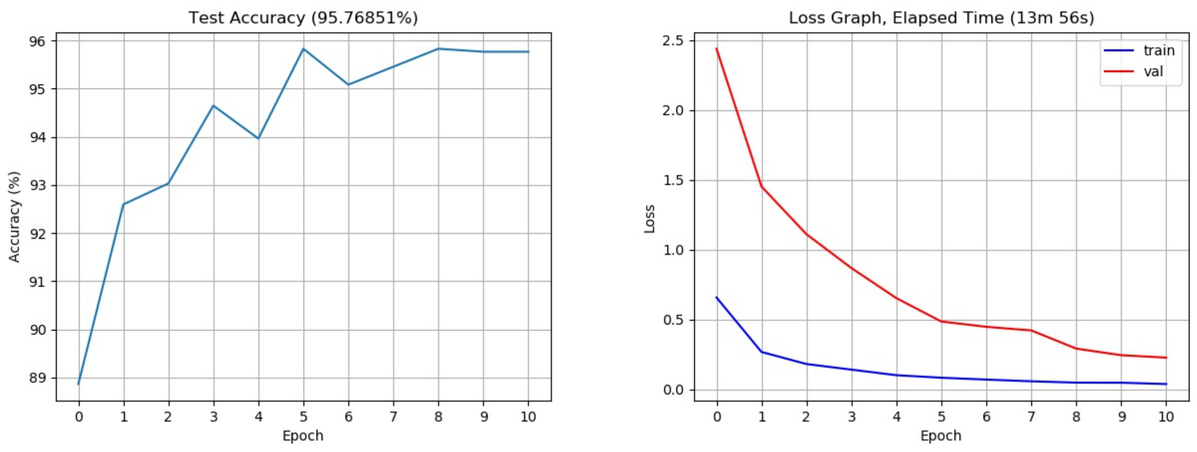


Figure 4.5.1: Test Accuracy Graph and Loss Graph for model with 2 hidden layers.

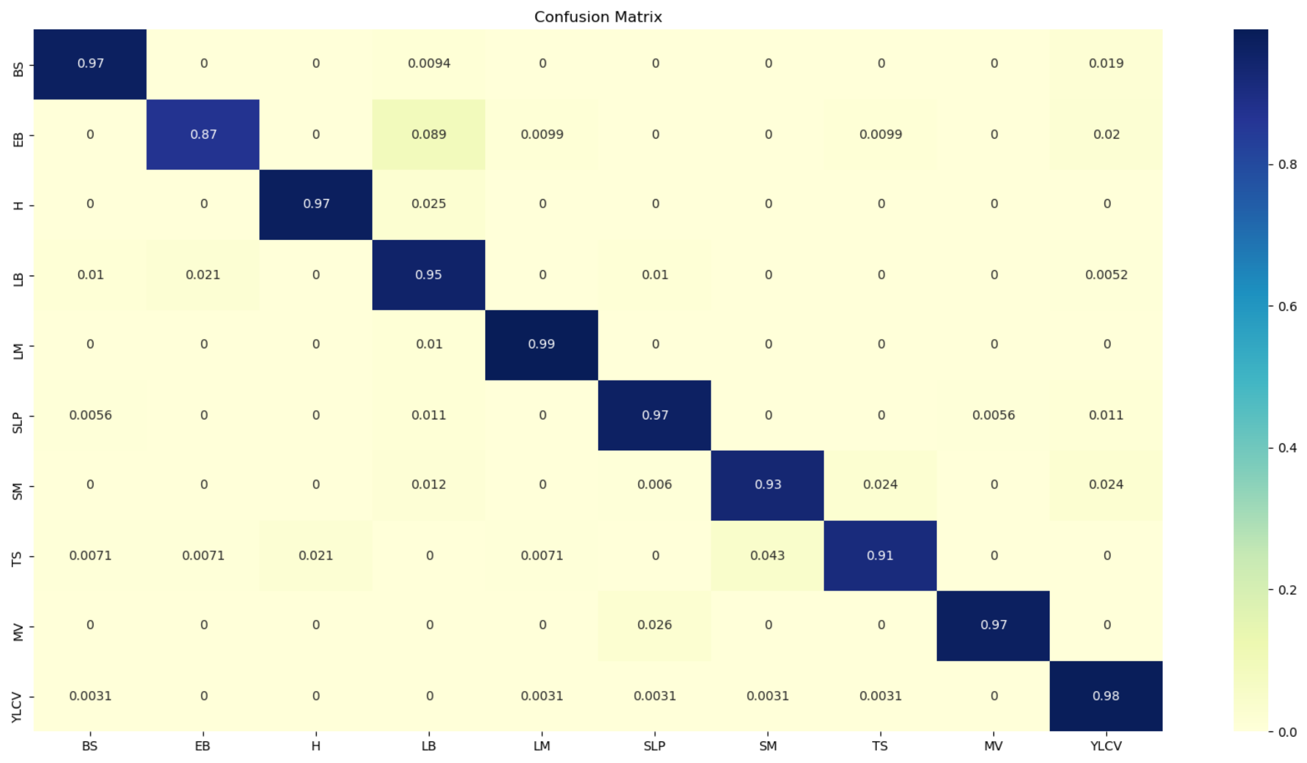


Figure 4.5.2: Confusion matrix for model with 2 hidden layers.

4.6 Learning Rate

The learning rate has decreased to 10^{-5} and observed the effect of the learning rate. When the learning rate is 10^{-5}, test accuracy is 73.55% so the accuracy has decreased extremely. As one can see that accuracy and loss graphs are more robust than before. The less learning rate means the less changes in weights of model for each iteration in learning phase. The graphs of the test and loss histories are shown in the figure below.

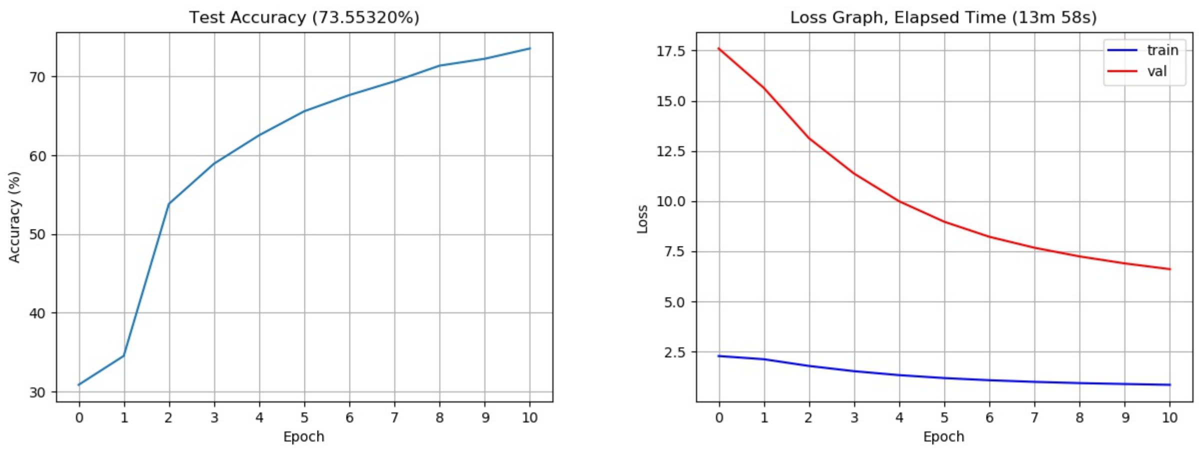
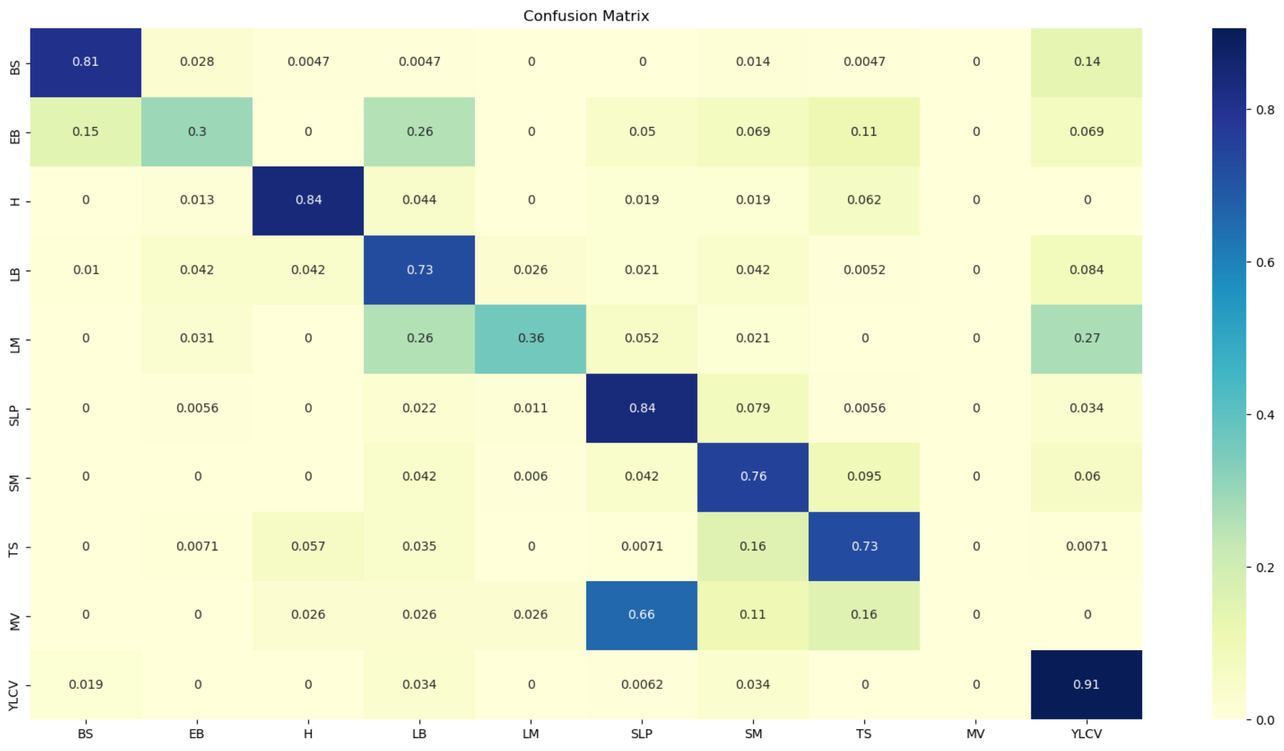


Figure 4.6.1: Test Accuracy Graph and Loss Graph for 10^{-5} learning rate.

As mentioned before, because of the learning rate has been scaled down, to reach the same accuracy level with the main model, it is required to increase the number of epochs therefore it causes increasing the training time.



	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.81	0.028	0.0047	0.0047	0	0	0.014	0.0047	0	0.14
EB	0.15	0.3	0	0.26	0	0.05	0.069	0.11	0	0.069
H	0	0.013	0.84	0.044	0	0.019	0.019	0.062	0	0
LB	0.01	0.042	0.042	0.73	0.026	0.021	0.042	0.0052	0	0.084
LM	0	0.031	0	0.26	0.36	0.052	0.021	0	0	0.27
SLP	0	0.0056	0	0.022	0.011	0.84	0.079	0.0056	0	0.034
SM	0	0	0	0.042	0.006	0.042	0.76	0.095	0	0.06
TS	0	0.0071	0.057	0.035	0	0.0071	0.16	0.73	0	0.0071
MV	0	0	0.026	0.026	0.026	0.66	0.11	0.16	0	0
YLCV	0.019	0	0	0.034	0	0.0062	0.034	0	0	0.91

Figure 4.6.2: Confusion matrix for 10^{-5} learning rate.

4.7 Comparison of Model Sizes on Memory

In the main model (3 hidden layers), parameters take up 4.33 MB in memory. When the number of neurons is decreased, model size decreases to 2.46 MB. Besides, model size is 3.95 MB for 2 hidden layers. These decreases are caused by reducing neuron numbers which means reducing the number of parameters.

4.8 Comparison of Algorithms

In our main training, MLP algorithm has been used. However, to see the effects of different machine learning algorithms which are supervised learning methods: KNN, Random Forest, Decision Tree and multi class SVM algorithms have also been implied. As a result, the following confusion matrices have been obtained. Also, these algorithms are explained briefly.

4.8.1 KNN

KNN (K - Nearest Neighbors) is one of the basic machine learning algorithm which is used for regression and classification. Only preprocessing and testing parts take time. Distance between input sample and the nearest k training samples to input sample is calculated and predictions are made by depending on similarities. k is the number of training instance which are compared with input sample and it was chosen as 5 in this project.

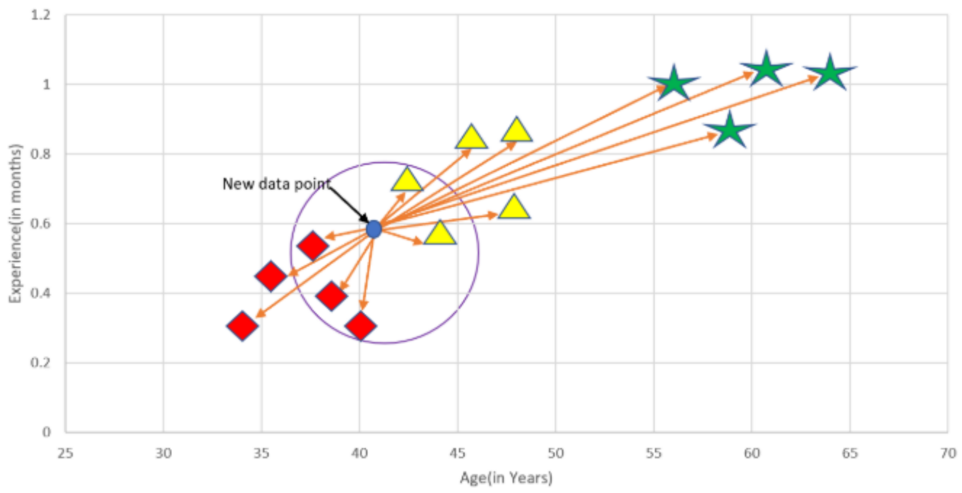
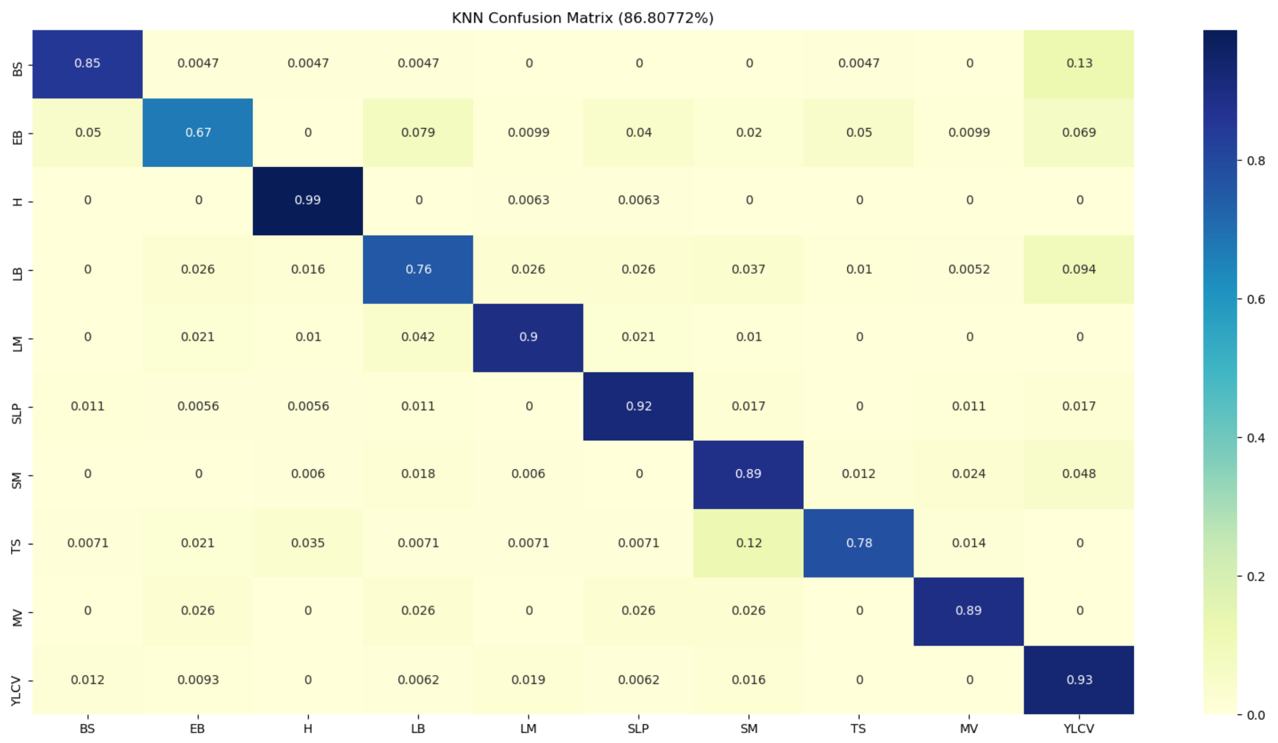


Figure 4.8.1.1: KNN Example [10].



	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.85	0.0047	0.0047	0.0047	0	0	0	0.0047	0	0.13
EB	0.05	0.67	0	0.079	0.0099	0.04	0.02	0.05	0.0099	0.069
H	0	0	0.99	0	0.0063	0.0063	0	0	0	0
LB	0	0.026	0.016	0.76	0.026	0.026	0.037	0.01	0.0052	0.094
LM	0	0.021	0.01	0.042	0.9	0.021	0.01	0	0	0
SLP	0.011	0.0056	0.0056	0.011	0	0.92	0.017	0	0.011	0.017
SM	0	0	0.006	0.018	0.006	0	0.89	0.012	0.024	0.048
TS	0.0071	0.021	0.035	0.0071	0.0071	0.0071	0.12	0.78	0.014	0
MV	0	0.026	0	0.026	0	0.026	0.026	0	0.89	0
YLCV	0.012	0.0093	0	0.0062	0.019	0.0062	0.016	0	0	0.93

Figure 4.8.1.2: Confusion matrix for KNN.

4.8.2 Decision Tree

Although Decision Tree is simple, it is efficient. A decision tree consists of internal nodes that correspond a test on a feature, each branch corresponds the test result, and leaf node corresponds a class label [11].

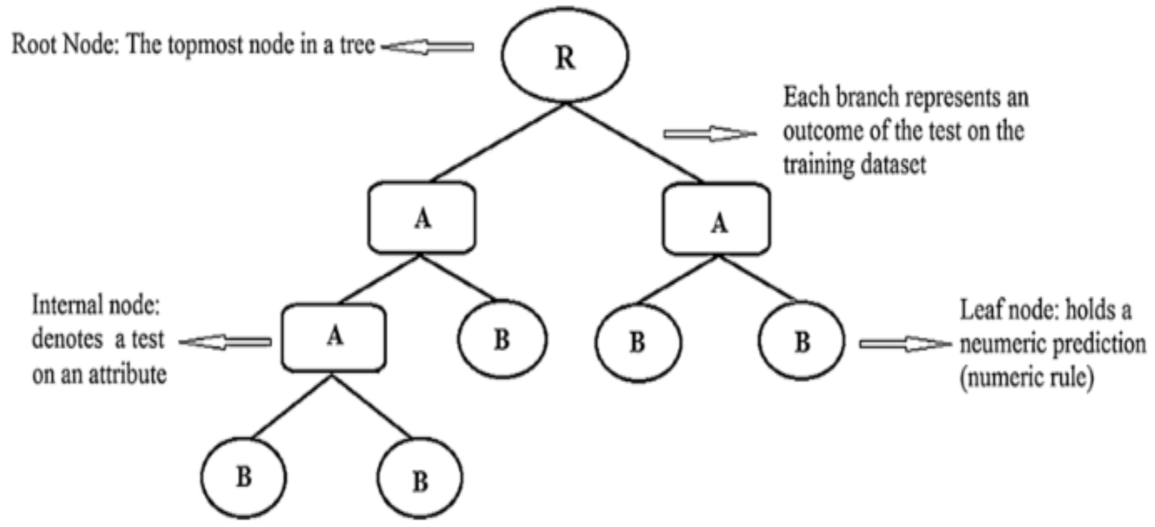
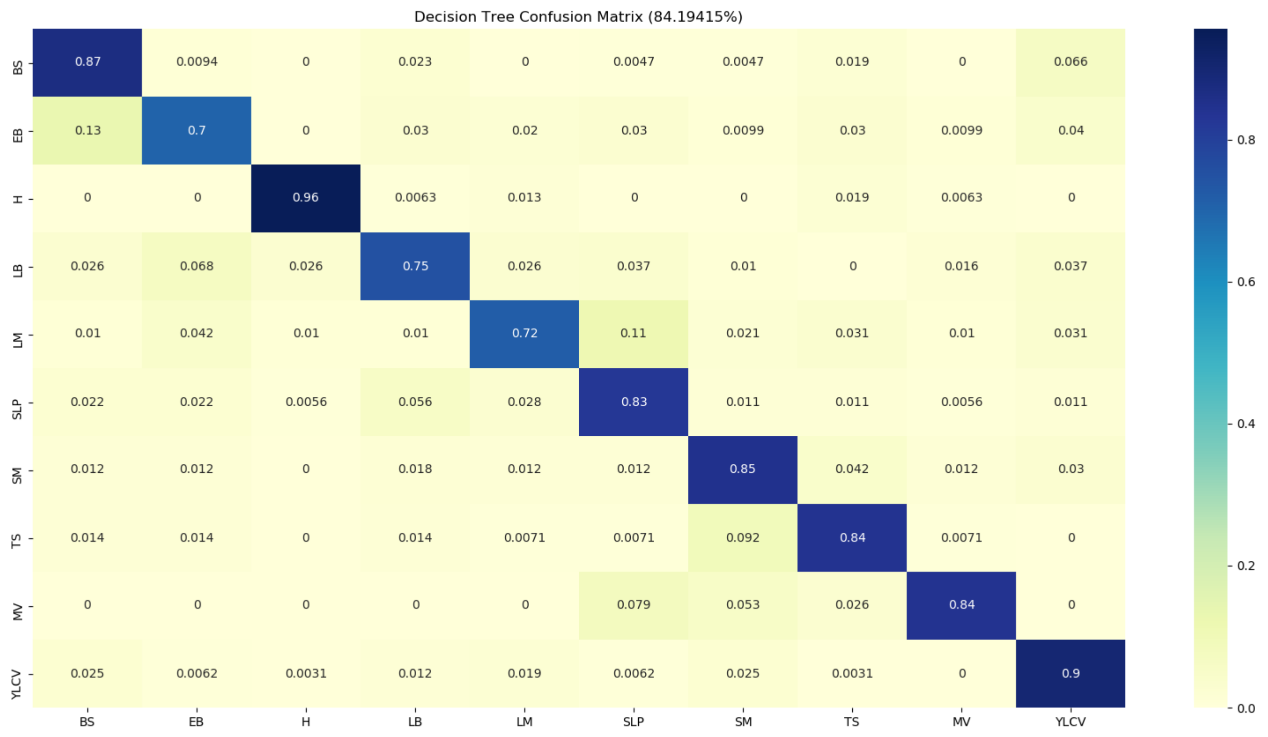

Root Node: The topmost node in a tree
R
Each branch represents an outcome of the test on the training dataset
A
A
Internal node: denotes a test on an attribute
A
B
B
B
Leaf node: holds a neumeric prediction (numeric rule)
B
B

Figure 4.8.2.1: Decision Tree example [11].



	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.87	0.0094	0	0.023	0	0.0047	0.0047	0.019	0	0.066
EB	0.13	0.7	0	0.03	0.02	0.03	0.0099	0.03	0.0099	0.04
H	0	0	0.96	0.0063	0.013	0	0	0.019	0.0063	0
LB	0.026	0.068	0.026	0.75	0.026	0.037	0.01	0	0.016	0.037
LM	0.01	0.042	0.01	0.01	0.72	0.11	0.021	0.031	0.01	0.031
SLP	0.022	0.022	0.0056	0.056	0.028	0.83	0.011	0.011	0.0056	0.011
SM	0.012	0.012	0	0.018	0.012	0.012	0.85	0.042	0.012	0.03
TS	0.014	0.014	0	0.014	0.0071	0.0071	0.092	0.84	0.0071	0
MV	0	0	0	0	0	0.079	0.053	0.026	0.84	0
YLCV	0.025	0.0062	0.0031	0.012	0.019	0.0062	0.025	0.0031	0	0.9

Figure 4.8.2.2: Confusion matrix for Decision Tree.

4.8.3 Random Forest

Random Forest contains n numbers of decision trees and makes predictions based on the majority of decision trees' outputs. n number was chosen 100 in this project.

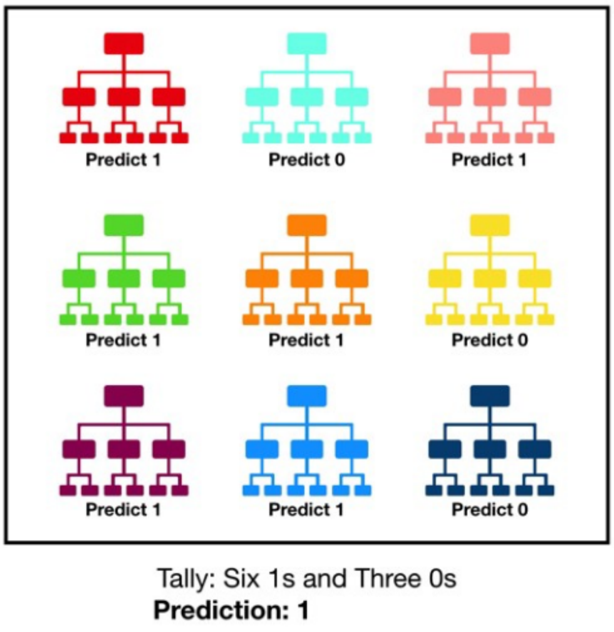
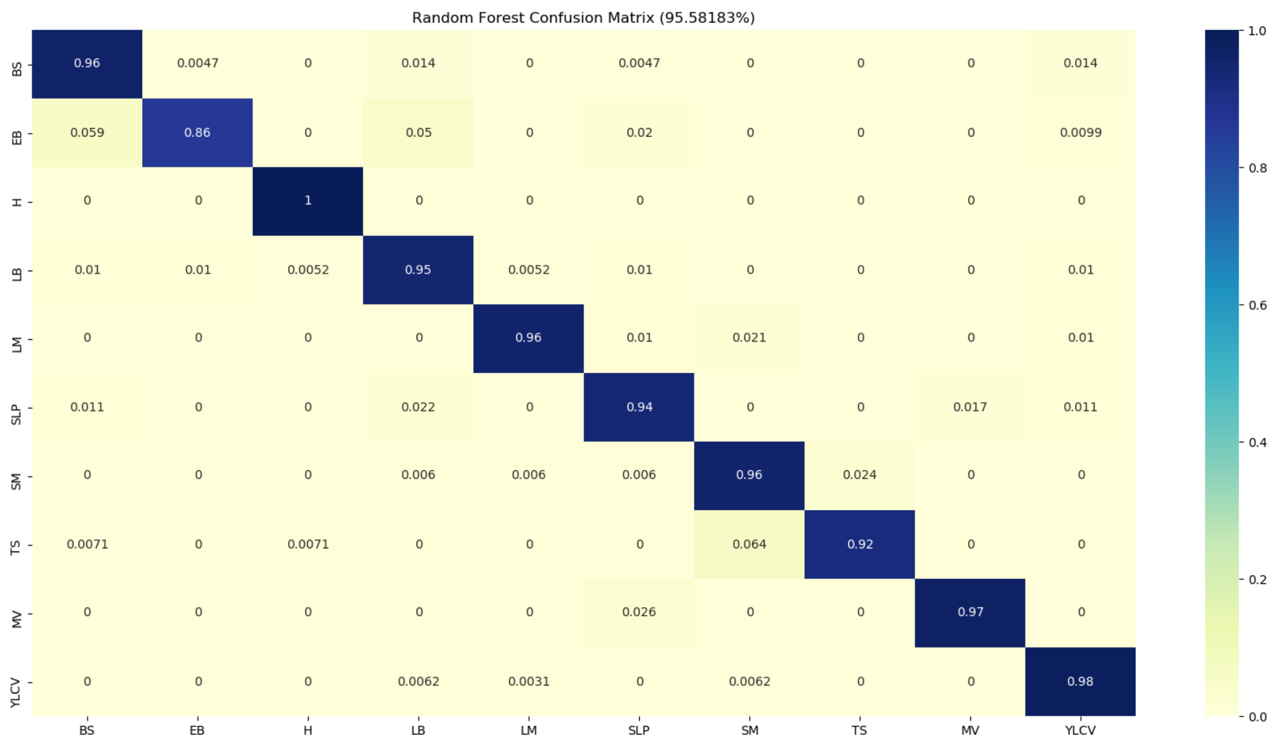


Figure 4.8.3.1: Random Forest example [12].


Random Forest Confusion Matrix (95.58183%)

	BS	EB	H	LB	LM	SLP	SM	TS	MV	YLCV
BS	0.96	0.0047	0	0.014	0	0.0047	0	0	0	0.014
EB	0.059	0.86	0	0.05	0	0.02	0	0	0	0.0099
H	0	0	1	0	0	0	0	0	0	0
LB	0.01	0.01	0.0052	0.95	0.0052	0.01	0	0	0	0.01
LM	0	0	0	0	0.96	0.01	0.021	0	0	0.01
SLP	0.011	0	0	0.022	0	0.94	0	0	0.017	0.011
SM	0	0	0	0.006	0.006	0.006	0.96	0.024	0	0
TS	0.0071	0	0.0071	0	0	0	0.064	0.92	0	0
MV	0	0	0	0	0	0.026	0	0	0.97	0
YLCV	0	0	0	0.0062	0.0031	0	0.0062	0	0	0.98

Figure 4.8.3.2: Confusion matrix for Random Forest.

4.8.4 Multi Class SVM

A Support Vector Machine (SVM) is aimed to find separating hyperplanes among all classes. Optimal hyperplane is defined as having the largest margin between classes. A margin is the distance between the hyperplane and class points nearest to it. The data points which are nearest to the separating hyperplane are the support vectors and if these vectors move, hyperplane also moves [13].

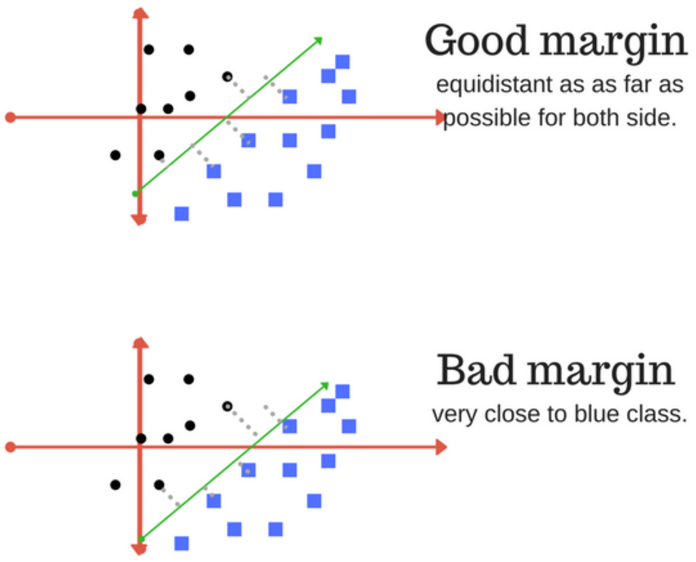

Good margin
equidistant as as far as possible for both side.
Bad margin
very close to blue class.

Figure 4.8.4.1: Multi Class SVM example [14].

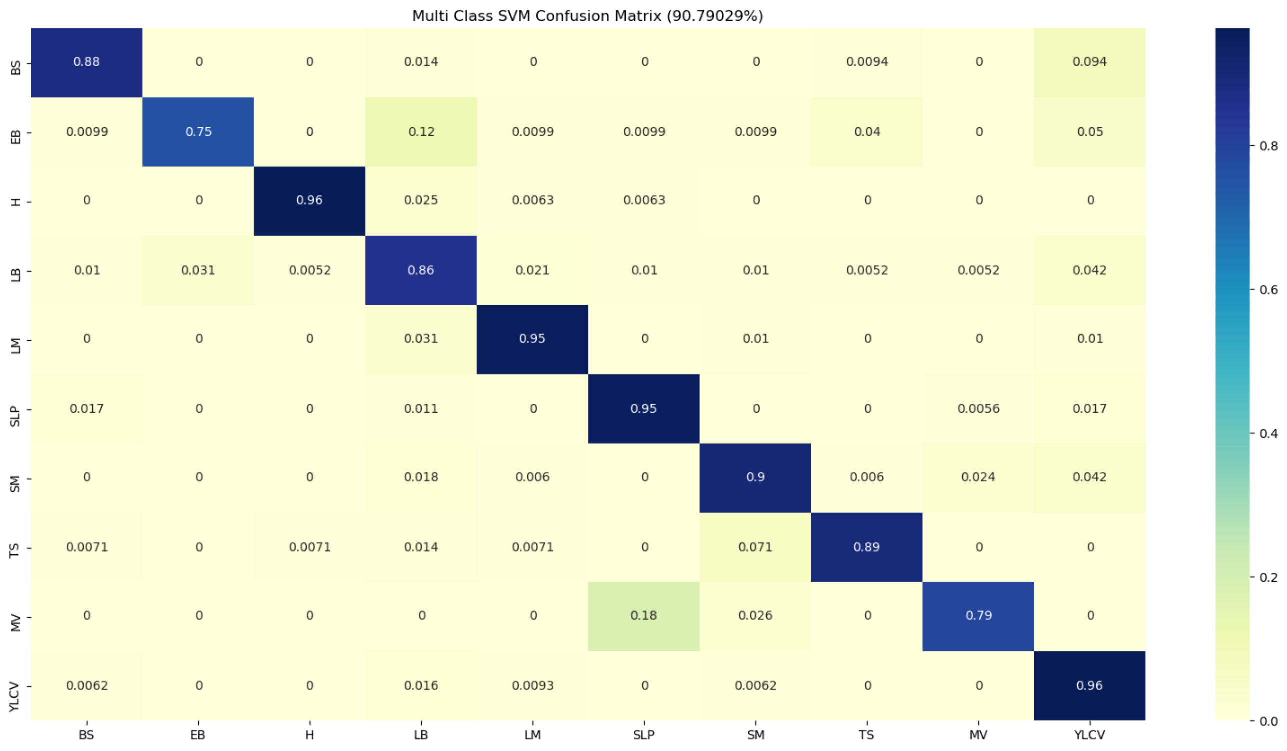


Figure 4.8.4.2: Confusion matrix for SVM.

Results of all machine learning algorithms and MLP are shown as bar plot in below.

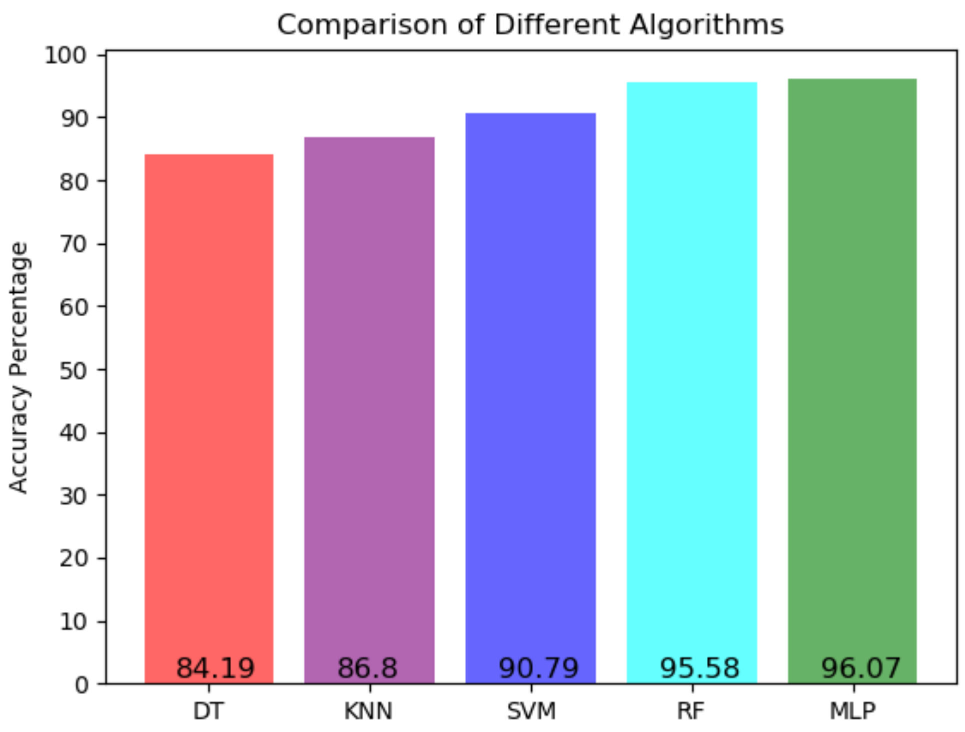


Figure 4.8.4.3: Accuracy comparison of different methods.

As it can be seen that MLP gave better result than other algorithms.

5. CONCLUSION

In conclusion, it is aimed to classify the tomato disease and its type by processing the images of tomato that is collected properly and compare the results by changing some parameters and algorithms. As a result, it can be said that the accuracy of the Multilayer Perceptron Algorithm has the best accuracy yet Random Forest also has a great accuracy percentage. However, Random Forest is suitable with only tabular data while Neural Network can be used for images, text and audio files. So that, Multilayer Perceptron has been preferred to train our data.

However, it is a well-known issue that making a general statement about the effects of the hyperparameters is difficult since the behaviour often varies from dataset to dataset and model to model. Therefore, the conclusions we make can only serve as signposts rather than general statements.

In future, an application for mobile devices can be developed in order to detect the disease state of a tomato by uploading an image of a tomato leaf simultaneously. Furthermore, it can be extended to all plants with use of proper data.

6. REFERENCES

[1] "FAOSTAT", Fao.org, 2020. [Online]. Available: http://www.fao.org/faostat/en/#home. [Accessed: 02- Jul- 2020].

[2] H. Durmuş, E. O. Güneş and M. Kırcı, "Disease detection on the leaves of the tomato plants by using deep learning," 2017 6th International Conference on Agro-Geoinformatics, Fairfax, VA, 2017, pp. 1-5, doi: 10.1109/Agro-Geoinformatics.2017.8047016.

[3] "PlantVillage Dataset", Kaggle.com, 2020. [Online]. Available: https://www.kaggle.com/emmarex/plantdisease. [Accessed: 02- Jul- 2020].

[4] S. Mohanty, D. Hughes and M. Salathé, "Using Deep Learning for Image-Based Plant Disease Detection", Frontiers in Plant Science, vol. 7, 2016. Available: 10.3389/fpls.2016.01419 [Accessed 2 July 2020].

[5] E. Too, L. Yujian, S. Njuki and L. Yingchun, "A comparative study of fine-tuning deep learning models for plant disease identification", Computers and Electronics in Agriculture, vol. 161, pp. 272-279, 2019. Available: 10.1016/j.compag.2018.03.032 [Accessed 2 July 2020].

[6] G. Started, R. Guide, O. Courses, C. (Old) and A. Consulting, "Shape Matching using Hu Moments (C++/Python)", Learnopencv.com, 2020. [Online]. Available: https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/. [Accessed: 03- Jul- 2020].

[7] I. Kandel and M. Castelli, "The effect of batch size on the generalizability of the convolutional neural networks on a histopathology dataset", ICT Express, 2020. Available: 10.1016/j.icte.2020.04.010.

[8] "Activation Functions: Sigmoid, ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep…", Medium, 2020. [Online]. Available: https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-andsoftmax-basics-for-neural-networks-and-deep-8d9c70eed91e. [Accessed: 15- May- 2020].

[9] "Understanding binary cross-entropy / log loss: a visual explanation", Medium, 2020. [Online]. Available: https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visualexplanation-a3ac6025181a. [Accessed: 10- May- 2020].

[10] "K-Nearest Neighbors(KNN)", *Medium*, 2020. [Online]. Available: https://medium.com/datadriveninvestor/k-nearest-neighbors-knn-7b4bd0128da7. [Accessed: 02- Jul- 2020].

[11] "Decision Trees—A simple way to visualize a decision", *Medium*, 2020. [Online]. Available: https://medium.com/greyatom/decision-trees-a-simple-way-to-visualize-a-decision-dc506a403aeb. [Accessed: 02- Jul- 2020].

[12] "Understanding Random Forest", *Medium*, 2020. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2. [Accessed: 02- Jul- 2020].

[13] "Build a Multi-Class Support Vector Machine in R", Medium, 2020. [Online]. Available: https://medium.com/@ODSC/build-a-multi-class-support-vector-machine-in-r-abcdd4b7dab6. [Accessed: 02- Jul- 2020].

[14] "Chapter 2 : SVM (Support Vector Machine) — Theory", Medium, 2020. [Online]. Available: https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72. [Accessed: 02- Jul- 2020]