Artificial Neural Network

Term Project Report

Mehmet Kapson 040150111

Berk Tanrıkulu 040140085

Fatma Rumeysa Önal 040150153

Asel Menekşe 040160151

Spring 2020

MUSHROOM CLASSIFICATION FOR DETECTION OF POISON

Model Estimation

In our project, we aimed to classify the mushrooms according to their cap features, gill features, population etc. in order to detect if the mushroom is poisonous or edible. To be able to do that classification, we have used several libraries such as Tensorflow, Sklearn, Pandas, Keras etc. We've used a .csv file as a data source 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%. We've also used one hidden layer, 22 features, and 30 neurons to train our model.

- **Training Set**

The training set is taken from the UCI Repository[1]. In the data file, there were 23 columns which represents the species, 8124 rows as the sample, and two classes as edible and poisonous represented with 'e' and 'p' letter as it can be seen from the figure of a small part of our training set below.

class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing
p	x	s	n	t	p	f	c
e	x	s	y	t	a	f	c
e	b	s	w	t	l	f	c
p	x	y	w	t	p	f	c
e	x	s	g	f	n	f	w
e	x	y	y	t	a	f	c
e	b	s	w	t	a	f	c
e	b	y	w	t	l	f	c
p	x	y	w	t	p	f	c

This dataset includes descriptions of hypothetical samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota Family (pp. 500-525)[2]. Each species is identified as definitely edible, definitely poisonous, or of unknown edibility and not recommended.

- **Activation Functions**

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 one hidden layer, and two layers in total. Later on it will be explained how the replacement of one of the activation function affects the model generally.

- **Optimizer**

We've used Stochastic gradient descent (SGD) as 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 [3].

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

Later on, Adam optimizer have been used instead of the SGD optimizer to understand how the optimizer type affects the model outcomes.

- **Learning Rate**

The learning rate is one of the most important parameter 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've set the learning rate to 10^{-3} which was adequate for our model estimation.

- **Batch Size**

Batch size is the number of training samples utilized in one cycle of the iteration. A training dataset can be divided into one or more batches:

- Batch Gradient Descent: Batch Size = Size of Training Set
- Stochastic Gradient Descent: Batch Size = 1
- Mini-Batch Gradient Descent: 1 < Batch Size < Size of Training Set

We've set the batch size to 16 which is less than the size of the training set, so that it is a type of Mini-Batch Gradient Descent. Therefore, it requires less memory since we train the network using fewer samples which also ensures a faster process.

- **Number of Epochs**

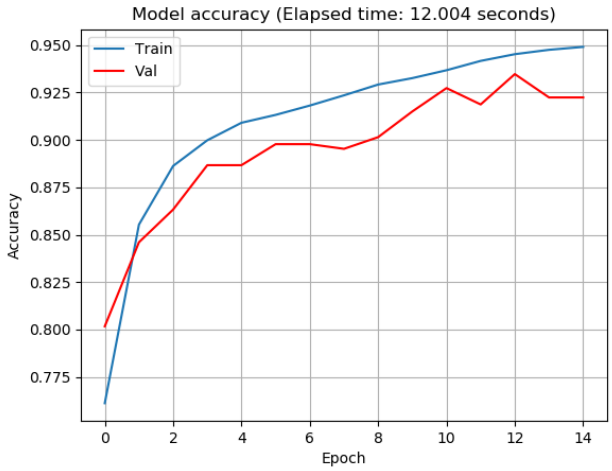
Number of epochs is also an important parameter that defines the number times that the learning algorithm will work through the entire training dataset. And it has been arranged to 15 by looking at the validation and training losses. If the validation loss increases, that means overfitting occurs. So that, we've set the number of epochs as high as possible and avoided from the overfitting.

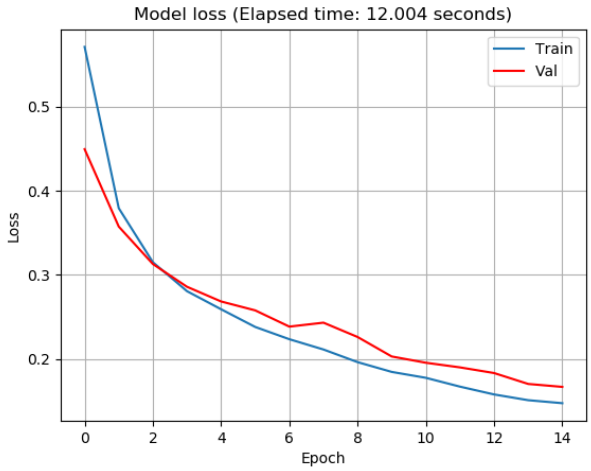
- **Model Accuracy & Model Loss**

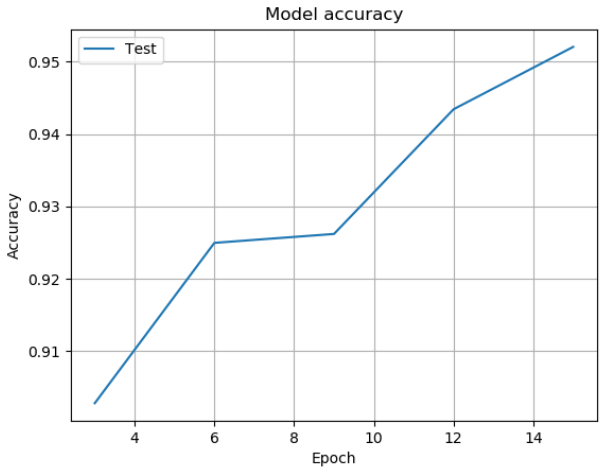
And as a result of the 15 epochs, 2 layers and the accuracy has reached to 95% approximately, while the loss decreases to 14% in 12.004 seconds which is the elapsed time, and they can be seen from the figure below.

```
6499/6499 [==============================] - 1s 167us/step - loss: 0.5713 - acc: 0.7612 - val_loss: 0.4496 - val_acc: 0.8017
Epoch 2/15
6499/6499 [==============================] - 1s 104us/step - loss: 0.3795 - acc: 0.8554 - val_loss: 0.3576 - val_acc: 0.8461
Epoch 3/15
6499/6499 [==============================] - 1s 107us/step - loss: 0.3152 - acc: 0.8863 - val_loss: 0.3130 - val_acc: 0.8633
Epoch 4/15
6499/6499 [==============================] - 1s 105us/step - loss: 0.2808 - acc: 0.8998 - val_loss: 0.2860 - val_acc: 0.8867
Epoch 5/15
6499/6499 [==============================] - 1s 109us/step - loss: 0.2592 - acc: 0.9091 - val_loss: 0.2685 - val_acc: 0.8867
Epoch 6/15
6499/6499 [==============================] - 1s 119us/step - loss: 0.2382 - acc: 0.9132 - val_loss: 0.2579 - val_acc: 0.8978
Epoch 7/15
6499/6499 [==============================] - 1s 115us/step - loss: 0.2236 - acc: 0.9181 - val_loss: 0.2386 - val_acc: 0.8978
Epoch 8/15
6499/6499 [==============================] - 1s 118us/step - loss: 0.2114 - acc: 0.9235 - val_loss: 0.2433 - val_acc: 0.8953
Epoch 9/15
6499/6499 [==============================] - 1s 121us/step - loss: 0.1963 - acc: 0.9292 - val_loss: 0.2262 - val_acc: 0.9015
Epoch 10/15
6499/6499 [==============================] - 1s 107us/step - loss: 0.1847 - acc: 0.9326 - val_loss: 0.2031 - val_acc: 0.9150
Epoch 11/15
6499/6499 [==============================] - 1s 116us/step - loss: 0.1777 - acc: 0.9368 - val_loss: 0.1956 - val_acc: 0.9273
Epoch 12/15
6499/6499 [==============================] - 1s 158us/step - loss: 0.1672 - acc: 0.9417 - val_loss: 0.1901 - val_acc: 0.9187
Epoch 13/15
6499/6499 [==============================] - 1s 133us/step - loss: 0.1579 - acc: 0.9452 - val_loss: 0.1834 - val_acc: 0.9347
Epoch 14/15
6499/6499 [==============================] - 1s 127us/step - loss: 0.1510 - acc: 0.9475 - val_loss: 0.1704 - val_acc: 0.9224
Epoch 15/15
6499/6499 [==============================] - 1s 109us/step - loss: 0.1475 - acc: 0.9491 - val_loss: 0.1669 - val_acc: 0.9224
Elapsed time for training: 12.004 seconds
```

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.

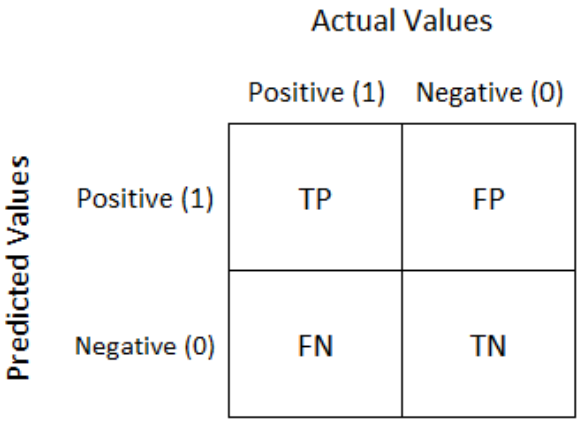






- **Confusion Matrix**

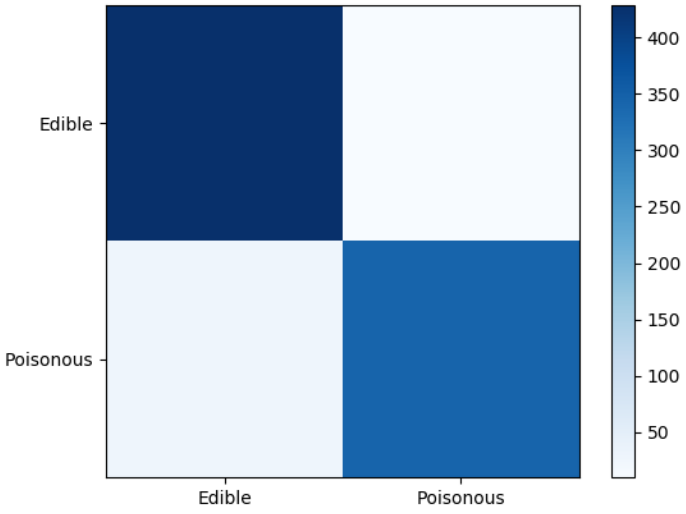
Confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. It allows the visualization of the performance of an algorithm. Confusion matrix has been computed to evaluate the accuracy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



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

```
813/813 [==============================] - 0s 101us/step
Epoch 3 test accuracy: 0.90283
813/813 [==============================] - 0s 125us/step
Epoch 6 test accuracy: 0.92497
813/813 [==============================] - 0s 122us/step
Epoch 9 test accuracy: 0.92620
813/813 [==============================] - 0s 129us/step
Epoch 12 test accuracy: 0.94342
813/813 [==============================] - 0s 145us/step
Epoch 15 test accuracy: 0.95203
[[429  10]
 [ 29 345]]
```

In our model, accuracy has been evaluated as 95% and the confusion matrix has been predicted as it can be seen above. Actually, accuracy could be increased by incrementing the number of epochs yet the elapsed time also increases which is unavoidable. In the figure below, the confusion matrix's both colormap and matrix form, and accuracy that is calculated can be seen clearly.



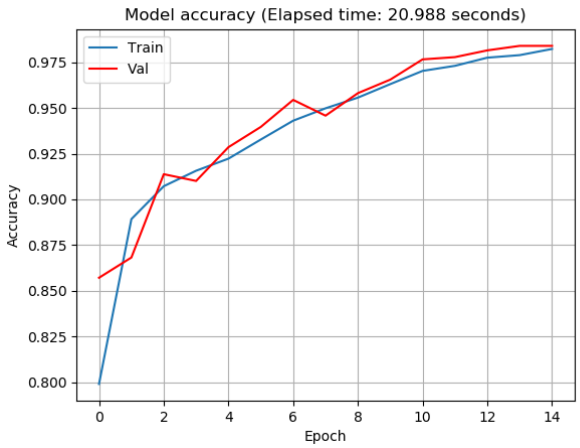
$$Confusion\ Matrix = \begin{bmatrix} 429 & 10 \\ 29 & 345 \end{bmatrix}$$

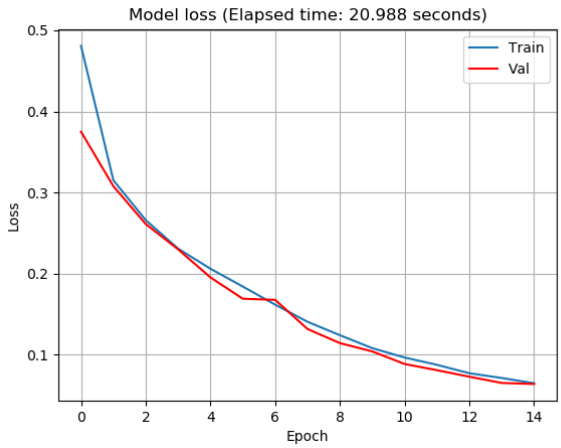
$$Accuracy = \frac{429 + 345}{813} = 0{,}9520$$

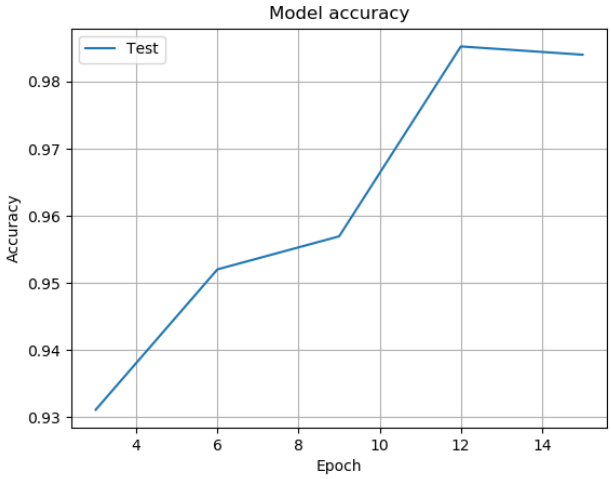
Comparison According to Some Parameters

- **Batch Size**

Batch size has been reduced to 8 to see how the accuracy changes. And as result of the decrement, the train and test accuracies has scaled up to 98.23% and 98.401%, respectively. However, the elapsed time for training has increased to 20.998 seconds as it can be seen from the figures below.



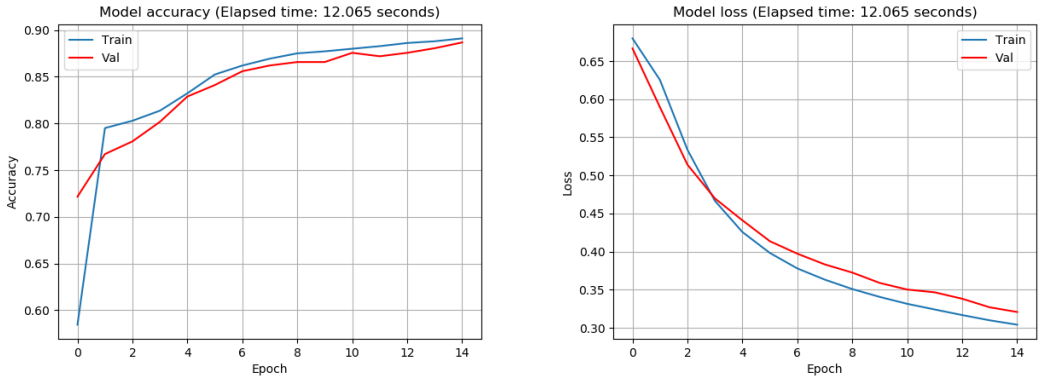


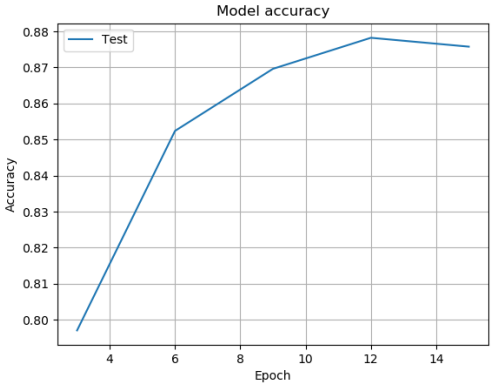


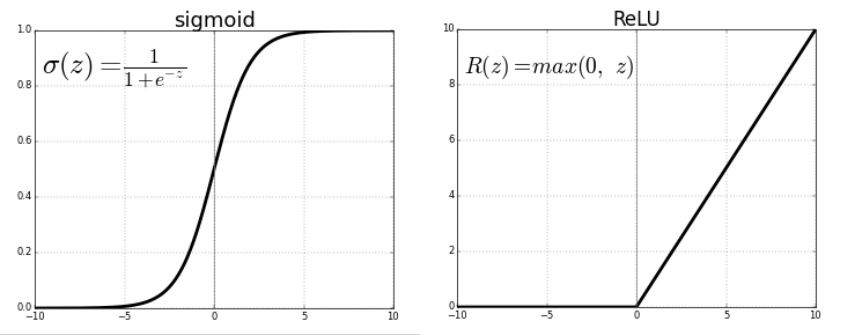
Actually, in researches about the effect of batch size, there are different results. In a research worked on two different datasets using 1024 and 16 batch sizes for the two datasets, the best accuracies has been obtained for 1024 batch sizes but, according to another research, the batch size does not affect the performance of the network yet, it decreases elapsed time to convergence for larger batch sizes [4].

- **Activation Function**

In our main training code there were two different functions as Relu and Sigmoid that is used for the activation function as mentioned before. And to compare the accuracy by changing the function, both of the 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 [5]. The graphs of accuracy history of training and testing and history of loss functions are seen in the figure below.







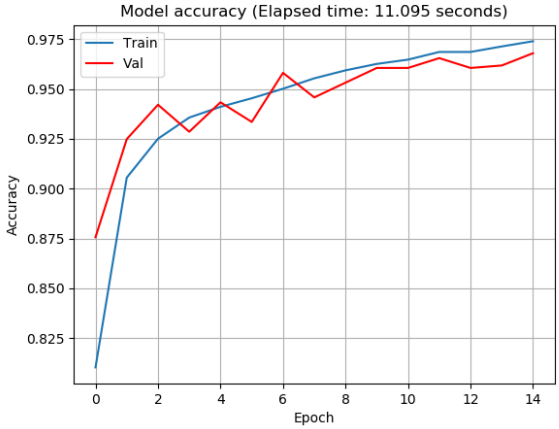
The graphs of sigmoid and relu functions are shown in the figure below. The Relu function has two basic advantages over the sigmoid function. First, its gradient is constant, and the probability of vanishing gradient is lower. As seen in the figure above, absolute values of gradient of the Sigmoid function 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.

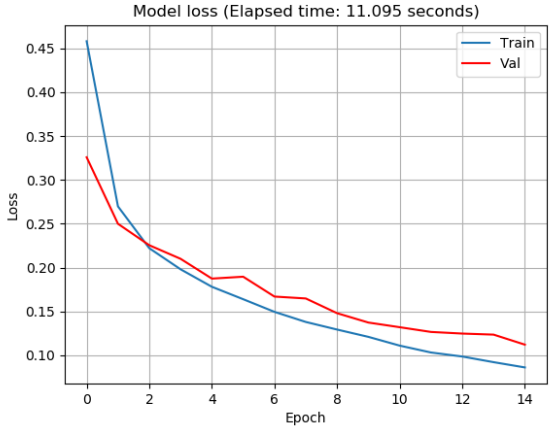
- **Optimizer**

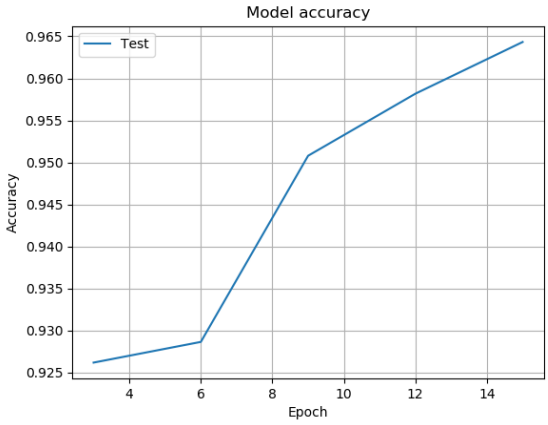
As a comparison option we've tried to use Adam Optimizer instead of SGD Optimizer which has a different working principle. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based in training data.

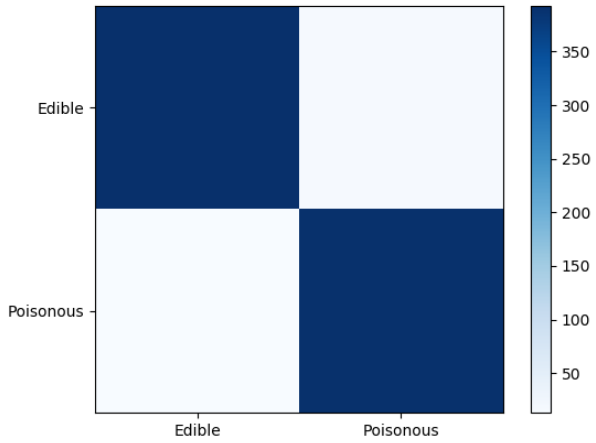
SGD Optimizer maintains a single learning rate for all weight updates and the learning rate does not change during training whereas the learning rate is maintained for each network weight and separately adapted as learning unfolds in Adam Optimizer. So that it is much more effective to estimate less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

As expected it has higher model accuracy as 97% and lower model loss as 8.6%, also the confusion matrix is distributed more accurate which can be also seen below.









$$Confusion\ Matrix = \begin{bmatrix} 393 & 16 \\ 13 & 391 \end{bmatrix}$$

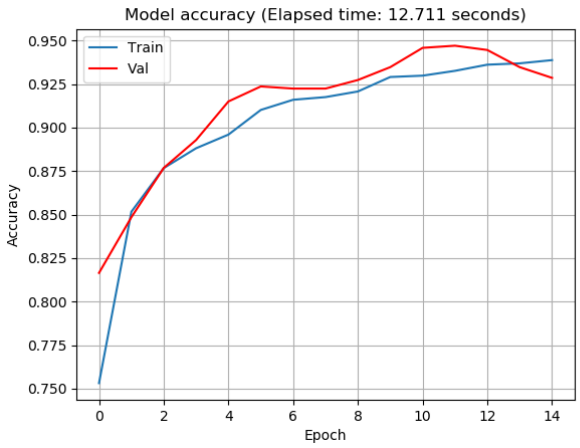
$$Accuracy = \frac{393 + 391}{813} = 0{,}9643$$

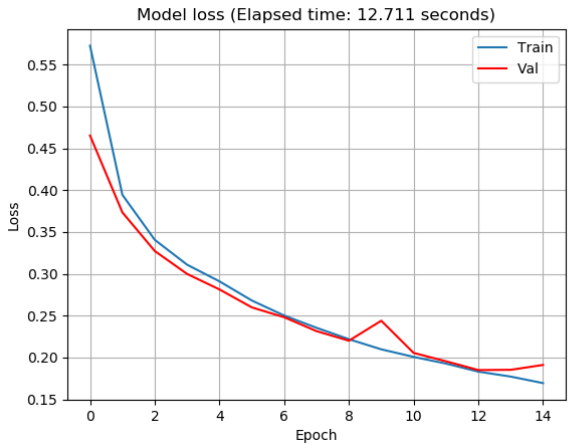
- **Number of Neurons**

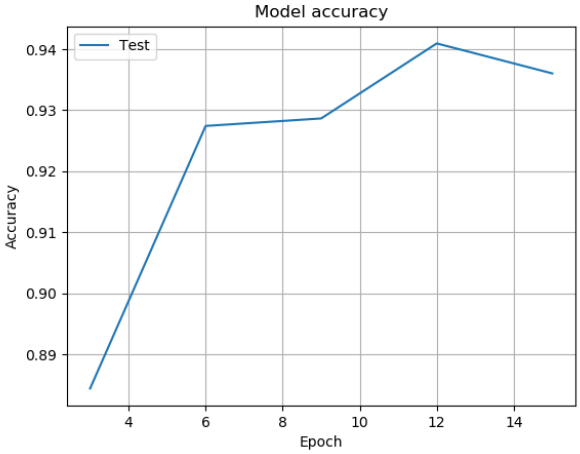
Neuron, also called a node or Perceptron, is a computational unit that has one or more weighted input connections, a transfer function for combining the inputs, and an output connection. Neurons are then organized into layers to comprise a network.

As the number of neurons increases, model gets more adaptive, so it can learn smaller details. But this doesn't mean that our classifier is then better on the 'next dataset'. 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, number of neurons have been decreased to 10 from 30 to see how the accuracy, loss and the execution time will change.

Consequently, the accuracy was almost the same with 93.6% whereas the model loss was a bit higher than our estimation as 19.12%, and the elapsed time was also a little less than our actual models' elapsed time which is 12.711 seconds. And all the process can be seen from the given figures below.

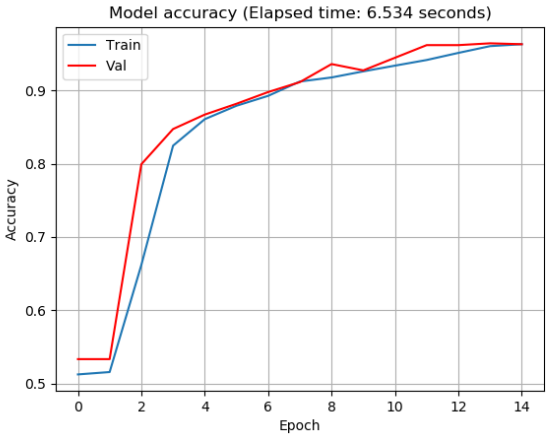


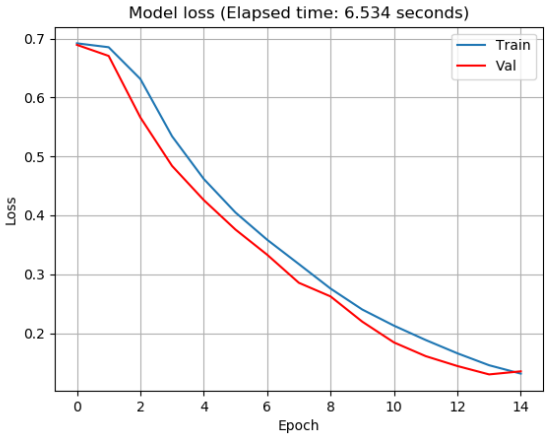


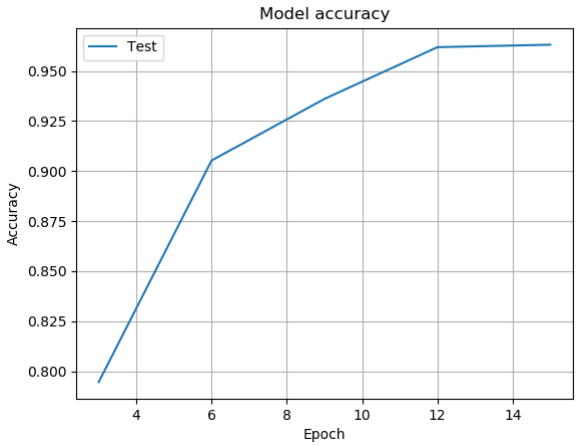


- **Number of Layers**

In artificial neural networks, hidden layers are required if and only if the data must be separated non-linearly. In our model, we've used only one hidden layer so that used two layers in total. To see the effect of the number of layers, we've increased the total layer number to three. As a result, accuracy have been increased very slightly to 96% and the model loss has decreased a bit to 13% too. Besides, the elapsed time has the most obvious decrease to 6.5 seconds, since hidden layer affects the time reasonably. To see the results in a more clear way the below figures can be examined.



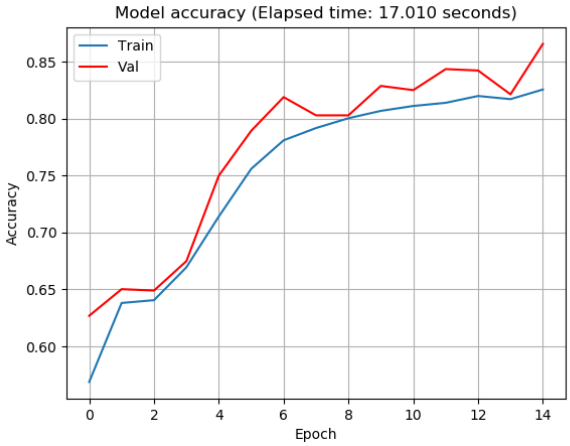


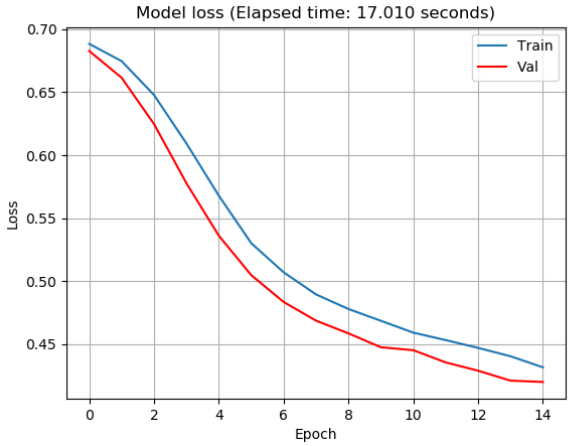


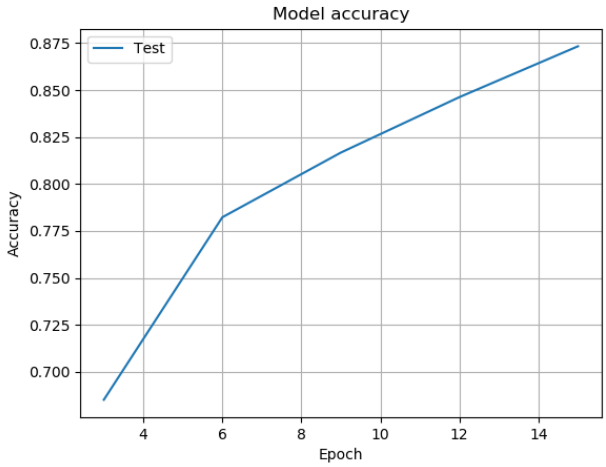
- **Number of Features**

Number of features of the model must match the input, so that we've used 22 features. And our goal is to find a trade-off between the accuracy and the execution time (computational cost). Since the execution time is directly connected to the number of features that is used, we should optimize the result by having the highest accuracy with the lowest execution time.

Hence, number of feature arrangement is important to reach to this point of equilibrium. In our model, we've already used the maximum feature number that is possible therefore, to show how much the accuracy goes down with less feature, we've set the number of features to 5. Eventually, we've seen that the result came out as we've expected as 85% accuracy, 17 seconds elapsed time which can be seen below.

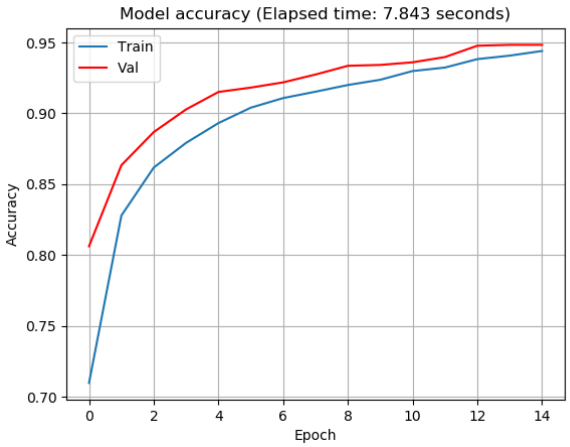


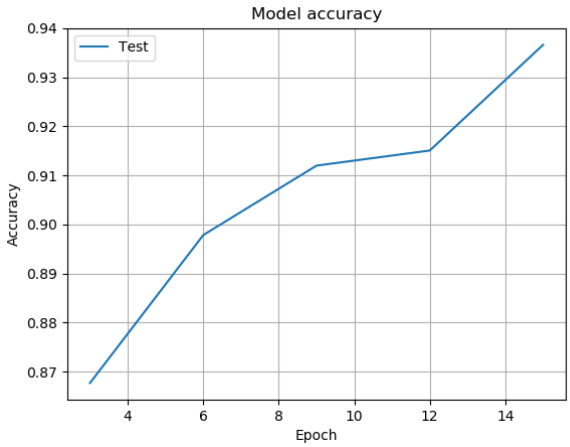




- **Test Size**

To see the effect of test size, total of the percentage of validation and test has increased to 40. The results are seen in the figures below.



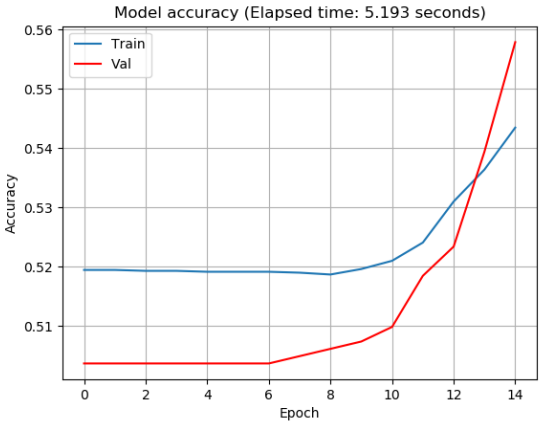


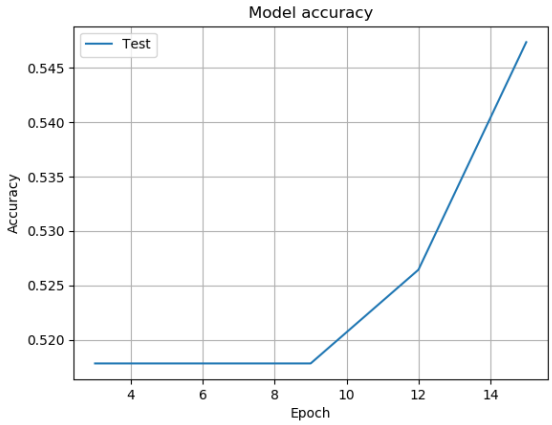
Train and test accuracies has scaled down to 94.4% and 93.662% respectively.

The effect of the test size changes according to properties of the dataset. Therefore, it cannot be said that smaller test size results in higher accuracy. Percentage of the test and training sizes can affect accuracy of the model differently with respect to size of the dataset.

- **Learning Rate**

The learning rate has decreased to 10^{-5} and increased to $3x10^{-1}$ and observed the effect of the learning rate. When the learning rate is 10^{-5}, training and test accuracies are 54.35% and 54.736% respectively so the accuracies has decreased extremely. The graphs of the test and train histories are shown in the figure below.





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.

When the learning rate is $3x10^{-1}$, training and test accuracies are 93.26% and 92.989% respectively. It is seen that increase in the learning rate has caused decrease in the accuracies.

REFERENCES

[1] "UCI Machine Learning Repository: Mushroom Data Set", Archive.ics.uci.edu, 2020. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Mushroom. [Accessed: 05- May- 2020].
[2] The Audubon Society Field Guide to North American Mushrooms (1981). G. H. Lincoff (Pres.), New York: Alfred A. Knopf
[3] "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].
[4] 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.
[5]"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].