# CSE-573 Project 2- Neural Networks

**Venugopal Shah**
Masters Student
Department of Computer Science
University at Buffalo
*vshah3@buffalo.edu*
*Person number: 50291126*

## Abstract

There are 3 tasks in this project. The first one is to build and implement a single hidden layer neural network from scratch for a multi-class problem. The second task is to use Keras library to implement a multi hidden layered neural network. The third task is to again use Keras to implement a Convolutional Neural Network (CNN). MNIST fashion dataset was used on all three tasks.

## 1 Introduction

### 1.1 Single Hidden Layer Neural Network

A single hidden layer neural network consists of 3 layers: input, hidden and output. The input layer has all the values from the input, in our case image features. In the hidden layer is where most of the calculations happens, every Perceptron unit takes an input from the input layer, multiplies and add it to initially random values. This initial output is not ready yet to exit the perceptron, it has to be activated by a function, in this case a Sigmoid function.

The last and third layer is the output layer, it takes all the previous layer Perceptron as input and multiplies and add their outputs to initially random values. It then gets activated by a Softmax function.

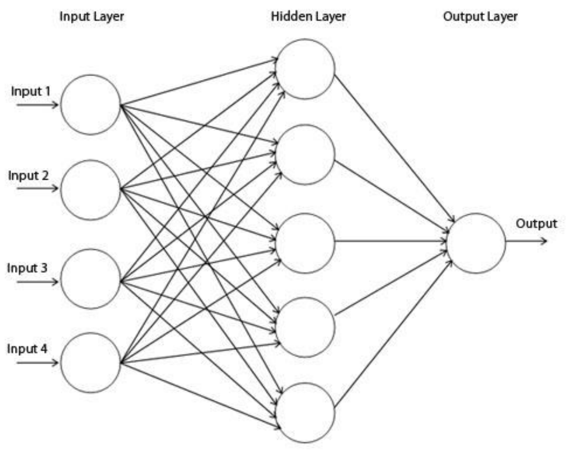
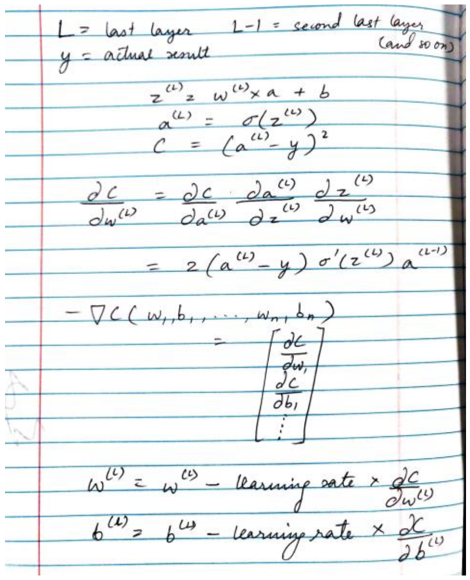


**Fig.1** Single Hidden Layer Neural Network

## 1.2 Backpropagation

The aim of Backpropagation is for calculating the gradients efficiently. We always start from the output layer and propagate backwards, updating weights and biases for each layer. We adjust the weights and biases throughout the network, so that we get the desired output in the output layer. We can only change the weights and biases, but activations are direct calculations of those weights and biases, which means we indirectly can adjust every part of the neural network, to get the desired output — except for the input layer, since that is the dataset that you input.



L = last layer  L-1 = second last layer (and so on)

y = actual result

$$z^{(L)} = w^{(L)} \times a + b$$
$$a^{(L)} = \sigma(z^{(L)})$$
$$C = (a^{(L)} - y)^2$$

$$\frac{\partial C}{\partial w^{(L)}} = \frac{\partial C}{\partial a^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial z^{(L)}}{\partial w^{(L)}}$$

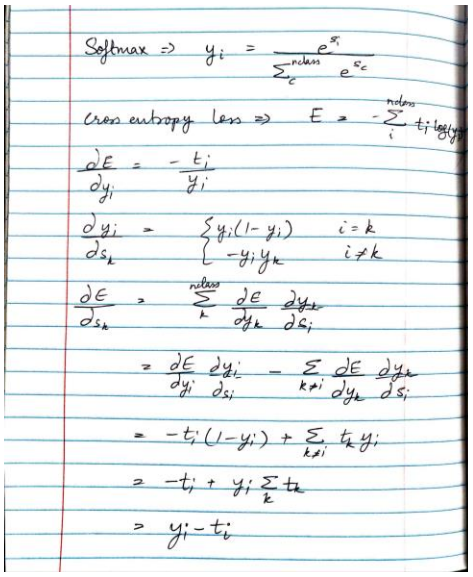
$$= 2(a^{(L)} - y)\,\sigma'(z^{(L)})\,a^{(L-1)}$$

$$-\nabla C(w_1, b_1, \ldots, w_n, b_n) = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial b_1} \\ \vdots \end{bmatrix}$$

$$w^{(L)} = w^{(L)} - \text{learning rate} \times \frac{\partial C}{\partial w^{(L)}}$$
$$b^{(L)} = b^{(L)} - \text{learning rate} \times \frac{\partial C}{\partial b^{(L)}}$$

**Fig.2** Backpropagation for a Mean Squared Error Loss

When a classification task has more than two classes, it is standard to use a softmax output layer. The softmax function provides a way of predicting a discrete probability distribution over the classes. We use the cross-entropy error function.



Softmax $\Rightarrow$ $y_i = \frac{e^{s_i}}{\sum_c^{nclass} e^{s_c}}$

Cross entropy loss $\Rightarrow$ $E = -\sum_i^{nclass} t_i \log(y_i)$

$$\frac{\partial E}{\partial y_i} = -\frac{t_i}{y_i}$$

$$\frac{\partial y_i}{\partial s_k} = \begin{cases} y_i(1-y_i) & i = k \\ -y_i y_k & i \neq k \end{cases}$$

$$\frac{\partial E}{\partial s_k} = \sum_k^{nclass} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial s_i}$$

$$= \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial s_i} - \sum_{k \neq i} \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial s_i}$$

$$= -t_i(1-y_i) + \sum_{k \neq i} t_k y_i$$

$$= -t_i + y_i \sum_k t_k$$

$$= y_i - t_i$$

**Fig.3** Backpropagation with Cross entropy loss function and Softmax activation

## 1.3 Multi Hidden Layered Neural Network

Artificial neural networks have two main hyperparameters that control the architecture or topology of the network: the number of layers and the number of nodes in each hidden layer. A Multilayer Perceptron can be used to represent convex regions. This means that in effect, they can learn to draw shapes around examples in some high-dimensional space that can separate and classify them, overcoming the limitation of linear separability.

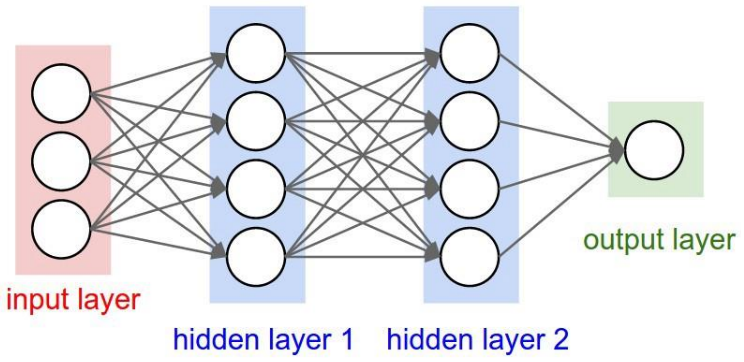


**Fig.4** Feedforward and Backpropogation have added steps over a single hidden layered neural network

## 1.4 Convolutional Neural Networks

Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms.

CNNs have two components:

- The Hidden layers/Feature extraction part: In this part, the network will perform a series of convolutions and pooling operations during which the features are detected. If you had a picture of a zebra, this is the part where the network would recognise its stripes, two ears, and four legs.

- The Classification part: Here, the fully connected layers will serve as a classifier on top of these extracted features. They will assign a probability for the object on the image being what the algorithm predicts it is.

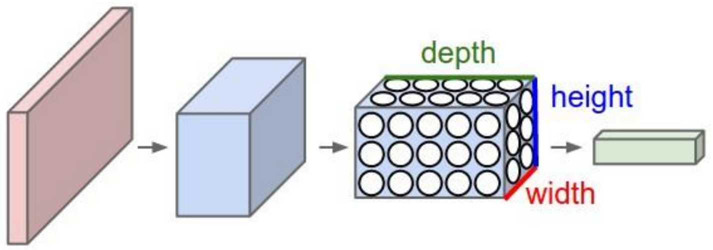


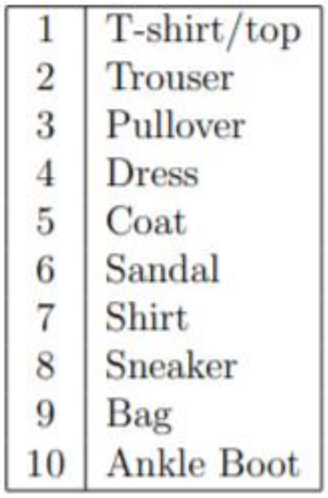
**Fig.5** Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data

## 2 Setup

### 2.1 Pre-processing data

First let us learn more about the dataset:

- The Fashion-MNIST is a dataset of Zalando’s article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes.
- Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255.
- The training and test data sets have 785 columns. The first column consists of the class labels (see above), and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image.
- We also normalize the dataset by dividing pixel values by 255, except for the results. This is to ensure all the values lie within a range which makes our calculations avoid divide by zero errors.



| | |
|---|---|
| 1 | T-shirt/top |
| 2 | Trouser |
| 3 | Pullover |
| 4 | Dress |
| 5 | Coat |
| 6 | Sandal |
| 7 | Shirt |
| 8 | Sneaker |
| 9 | Bag |
| 10 | Ankle Boot |

**Fig.6** Labels for Fashion MNIST dataset

### 2.2 Model

Let us now see how our Neural Network model from scratch works:

- We train our model using the training dataset. We will keep on changing our hyperparameters to see what works best with the validation set.
- The experiments on the validation set was tried for different epochs and different learning rates, noting the training and validation loss and also the training and validation accuracies.

  Different learning rates experimented with:

  `[0.1, 1]`

  Different epochs experimented with:

  `[100, 200, 400, 600]`

  Different depths experimented with:

  `[128, 200, 400]`

- The hyper-parameters for the best results was then noted and used for running the test set.
- The results on the test set were then used to construct a confusion matrix, which was then further used to calculate the precision and recall.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

**Fig.7** Formulas used to calculate Accuracy, Precision and Recall

- Similarly, we worked on multi-layer neural network and also the CNN, trying out various hyperparameters and then deciding what is best for the test set.

Now let us move on to the results where we will analyze and infer from our findings.

## 3 Results- One Hidden Layer Neural Network

### 3.1 Learning Rate

Learning rate is a very vital hyper-parameter that has to be chosen really carefully. A learning rate that is too large can cause the model to converge too quickly to a suboptimal solution, whereas a learning rate that is too small can cause the process to get stuck.

**NOTE:** For each of these tests, the epochs were set at 200

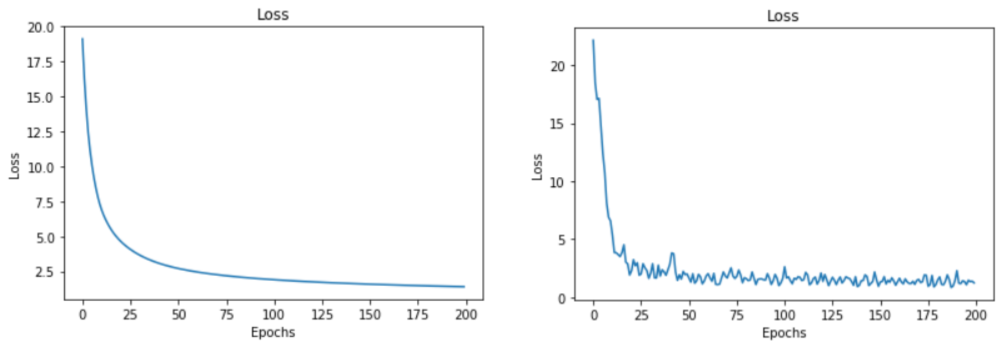
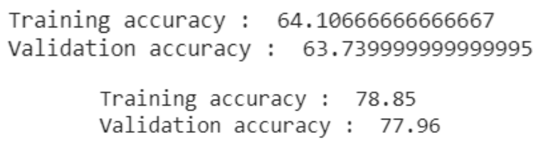


**Fig.8** The left figure shows loss when learning rate is 0.1 and the right figure shows loss when learning rate is 1. For our model coded from scratch, we can clearly see that a higher learning rate is yielding better results.



```
Training accuracy :  64.10666666666667
Validation accuracy :  63.739999999999995
```

```
Training accuracy :  78.85
Validation accuracy :  77.96
```

**Fig.9** Increasing learning rate increased training and validation accuracies significantly. So now we moved on to experiment with the number of epochs to see how that affected our accuracies.

## 3.2 Epoch

As the number of epochs increases, more number of times the weights are changed in the neural network and the curve goes from under-fitting to optimal to overfitting curve. As you can see from following graph, the number of epochs increase and reduce the loss. But again, a very high number of epoch can lead to overfitting.

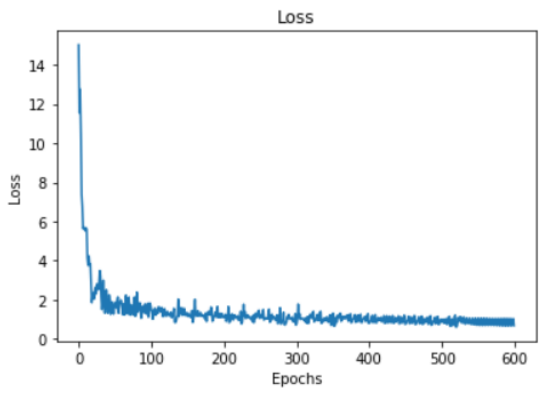
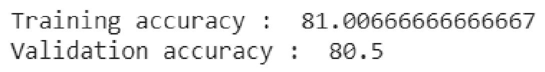


**Fig.10** With increasing Epoch the training loss goes down before overfitting happens



```
Training accuracy :  81.00666666666667
Validation accuracy :  80.5
```

**Fig.11** Both training and validation accuracies increase with increasing epoch. These accuracies were noted with learning rate as 1 and depth of layer as 200.

## 3.3 Depth of layer

The depth of the hidden layer also plays an important role in determining our results. But there is no fixed answer on how to choose the number of nodes in your hidden layer. After some hit and trial on different values, we got the following results.

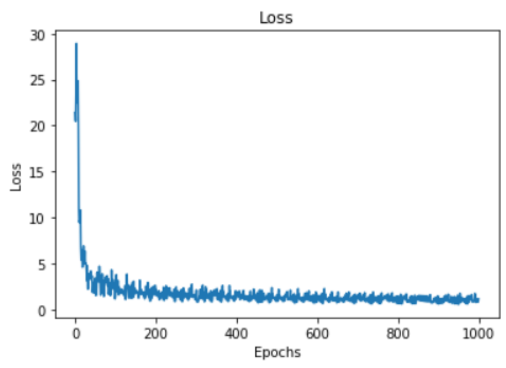
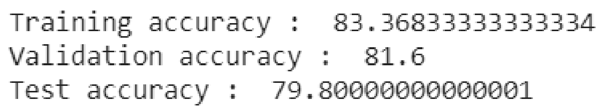


**Fig.12** Running till 1000 epochs with more number of nodes does give us better results but only marginally and takes a lot of computation time.

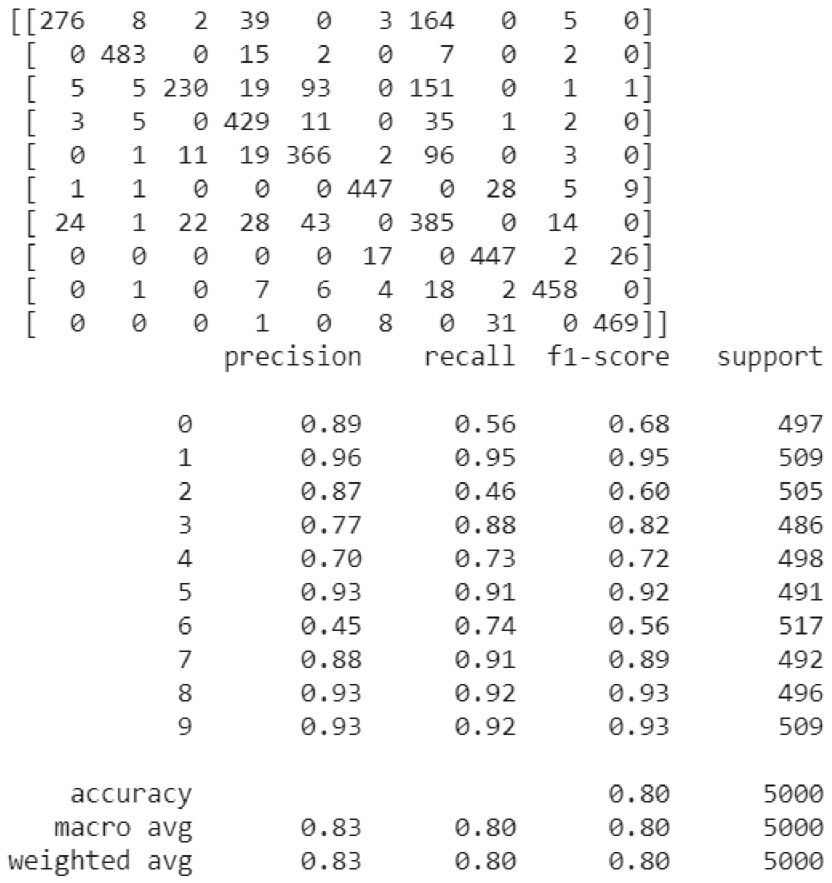


```
Training accuracy :  83.36833333333334
Validation accuracy :  81.6
Test accuracy :  79.80000000000001
```

**Fig.13** With the best results on validation accuracy, we ran for the test set and got the above outputs.

### 3.4 Test

Confusion matrix and Inferences:

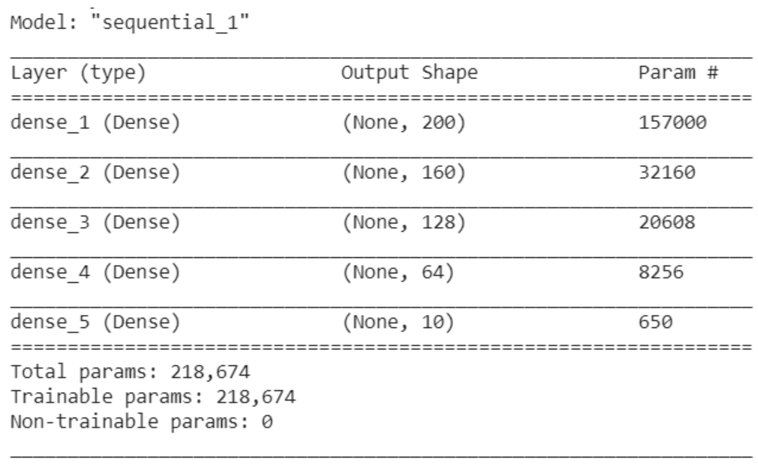


```
[[276   8   2  39   0   3 164   0   5   0]
 [  0 483   0  15   2   0   7   0   2   0]
 [  5   5 230  19  93   0 151   0   1   1]
 [  3   5   0 429  11   0  35   1   2   0]
 [  0   1  11  19 366   2  96   0   3   0]
 [  1   1   0   0   0 447   0  28   5   9]
 [ 24   1  22  28  43   0 385   0  14   0]
 [  0   0   0   0   0  17   0 447   2  26]
 [  0   1   0   7   6   4  18   2 458   0]
 [  0   0   0   1   0   8   0  31   0 469]]
              precision    recall  f1-score   support

           0       0.89      0.56      0.68       497
           1       0.96      0.95      0.95       509
           2       0.87      0.46      0.60       505
           3       0.77      0.88      0.82       486
           4       0.70      0.73      0.72       498
           5       0.93      0.91      0.92       491
           6       0.45      0.74      0.56       517
           7       0.88      0.91      0.89       492
           8       0.93      0.92      0.93       496
           9       0.93      0.92      0.93       509

    accuracy                           0.80      5000
   macro avg       0.83      0.80      0.80      5000
weighted avg       0.83      0.80      0.80      5000
```

**Fig.14** A visualization of the final predictions on the test data

## 4 Results- Multi Hidden Layer Neural Network

Here is the summary of the tests run on Multi hidden layer neural network using Keras.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 200)               157000
_________________________________________________________________
dense_2 (Dense)              (None, 160)               32160
_________________________________________________________________
dense_3 (Dense)              (None, 128)               20608
_________________________________________________________________
dense_4 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_5 (Dense)              (None, 10)                650
=================================================================
Total params: 218,674
Trainable params: 218,674
Non-trainable params: 0
_________________________________________________________________
```

**Fig.15** A visualization of the final predictions on the test data

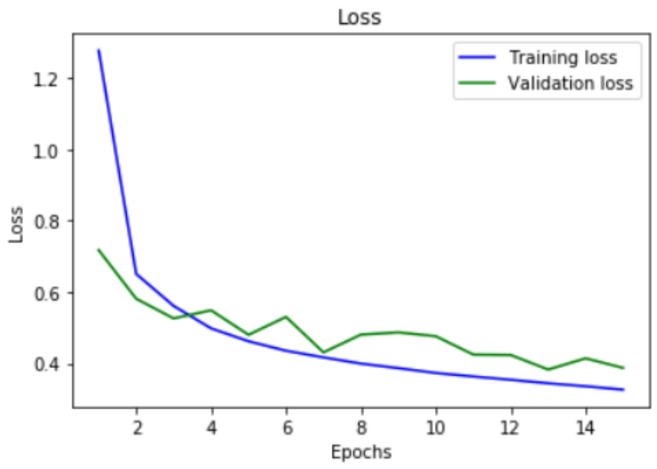


**Fig.16** The training loss decreased steadily but the validation loss suffered from fluctuations hinting at overfitting

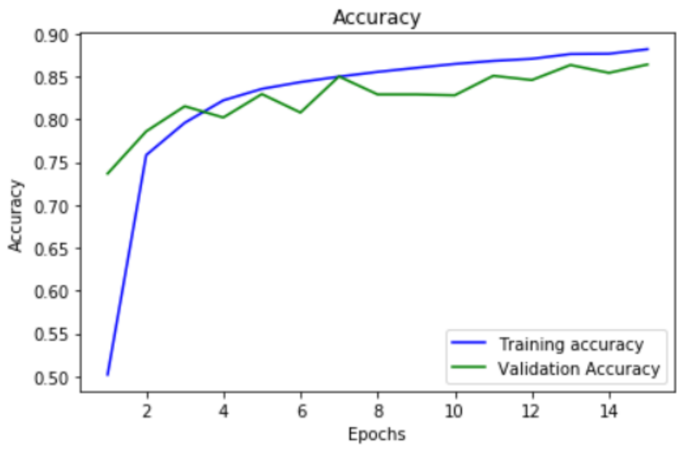
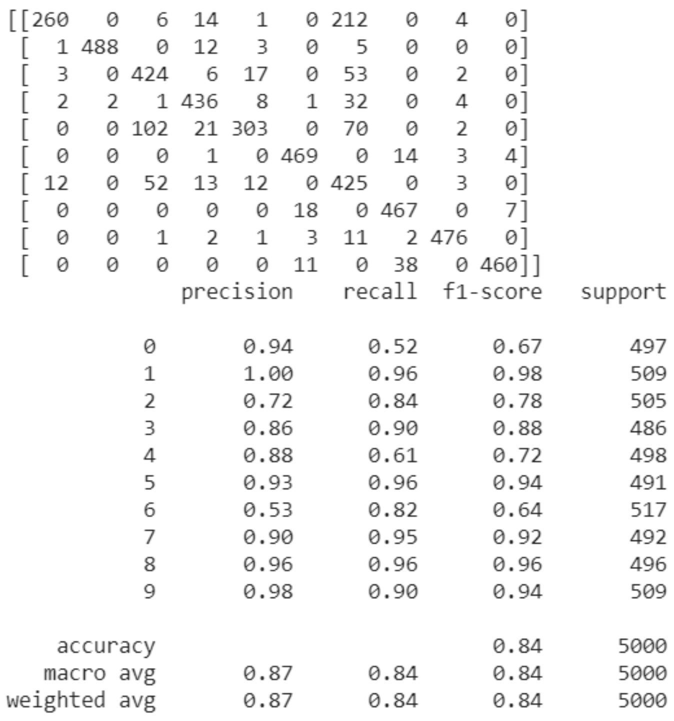


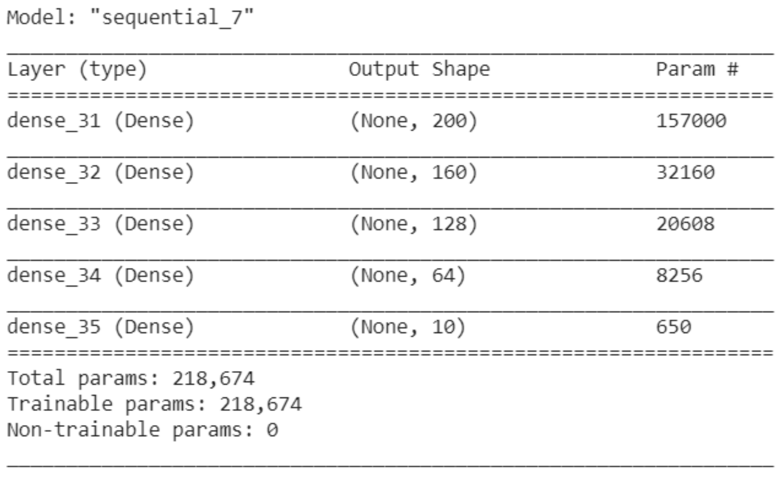
**Fig.17** Accuracy increased for both training and validation although stagnancy was observed for validation data



```
[[260   0   6  14   1   0 212   0   4   0]
 [  1 488   0  12   3   0   5   0   0   0]
 [  3   0 424   6  17   0  53   0   2   0]
 [  2   2   1 436   8   1  32   0   4   0]
 [  0   0 102  21 303   0  70   0   2   0]
 [  0   0   0   1   0 469   0  14   3   4]
 [ 12   0  52  13  12   0 425   0   3   0]
 [  0   0   0   0   0  18   0 467   0   7]
 [  0   0   1   2   1   3  11   2 476   0]
 [  0   0   0   0   0  11   0  38   0 460]]
              precision    recall  f1-score   support

           0       0.94      0.52      0.67       497
           1       1.00      0.96      0.98       509
           2       0.72      0.84      0.78       505
           3       0.86      0.90      0.88       486
           4       0.88      0.61      0.72       498
           5       0.93      0.96      0.94       491
           6       0.53      0.82      0.64       517
           7       0.90      0.95      0.92       492
           8       0.96      0.96      0.96       496
           9       0.98      0.90      0.94       509

    accuracy                           0.84      5000
   macro avg       0.87      0.84      0.84      5000
weighted avg       0.87      0.84      0.84      5000
```

**Fig.18** Confusion matrix and Precision, Recall calculations for test evaluation

## 5 Results- Convolutional Neural Network



```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_31 (Dense)             (None, 200)               157000
_________________________________________________________________
dense_32 (Dense)             (None, 160)               32160
_________________________________________________________________
dense_33 (Dense)             (None, 128)               20608
_________________________________________________________________
dense_34 (Dense)             (None, 64)                8256
_________________________________________________________________
dense_35 (Dense)             (None, 10)                650
=================================================================
Total params: 218,674
Trainable params: 218,674
Non-trainable params: 0
_________________________________________________________________
```

**Fig.19** The model structure for the convolutional neural network

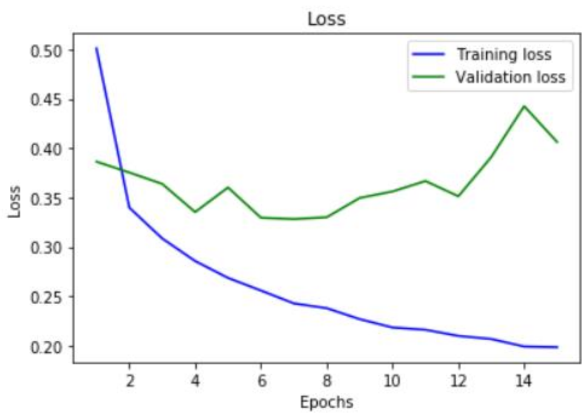


**Fig.20** The effect of overfitting was clearly visible on the validation set

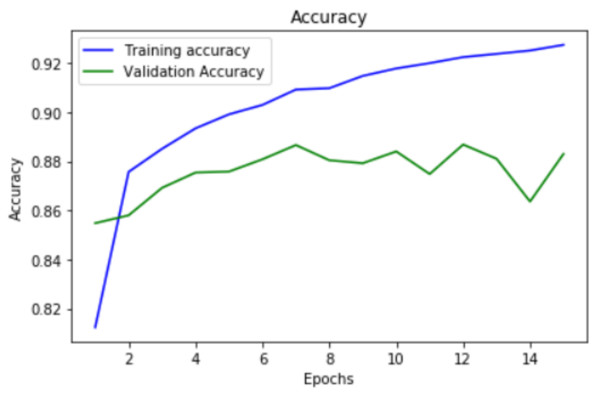
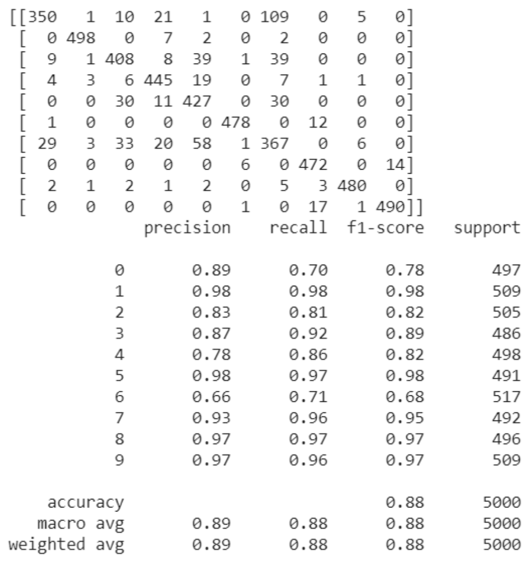


**Fig.21** The accuracy of training data continued to increase but the validation set suffered from overfitting



```
[[350   1  10  21   1   0 109   0   5   0]
 [  0 498   0   7   2   0   2   0   0   0]
 [  9   1 408   8  39   1  39   0   0   0]
 [  4   3   6 445  19   0   7   1   1   0]
 [  0   0  30  11 427   0  30   0   0   0]
 [  1   0   0   0   0 478   0  12   0   0]
 [ 29   3  33  20  58   1 367   0   6   0]
 [  0   0   0   0   0   6   0 472   0  14]
 [  2   1   2   1   2   0   5   3 480   0]
 [  0   0   0   0   0   1   0  17   1 490]]
              precision    recall  f1-score   support

           0       0.89      0.70      0.78       497
           1       0.98      0.98      0.98       509
           2       0.83      0.81      0.82       505
           3       0.87      0.92      0.89       486
           4       0.78      0.86      0.82       498
           5       0.98      0.97      0.98       491
           6       0.66      0.71      0.68       517
           7       0.93      0.96      0.95       492
           8       0.97      0.97      0.97       496
           9       0.97      0.96      0.97       509

    accuracy                           0.88      5000
   macro avg       0.89      0.88      0.88      5000
weighted avg       0.89      0.88      0.88      5000
```

**Fig.22** Confusion matrix and Precision, Recall calculations for test evaluation

## SUMMARY

| | Single Hidden Layer Neural Network | Multi Hidden Layer Neural Network | Convolutional Neural Network |
|---|---|---|---|
| Accuracy | 79.8% | 84% | 89% |
| Precision | 83% | 87% | 89% |
| Recall | 80% | 84% | 88% |

## References

https://keras.io/callbacks/#create-a-callbackhttps://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389

https://medium.com/@martinpella/logistic-regression-from-scratch-in-python-124c5636b8ac

https://mlfromscratch.com/neural-networks-explained/#/

https://towardsdatascience.com/how-to-build-your-own-neural-network-from-scratch-in-python-68998a08e4f6

Prof. Srihari's lectures, slides and assistance from the TAa was also essential in completing this project.