# Classification of fashion-MNIST dataset using different types of neural networks

**Sankeerth Tella**
50317364
stella3@buffalo.edu

## Abstract

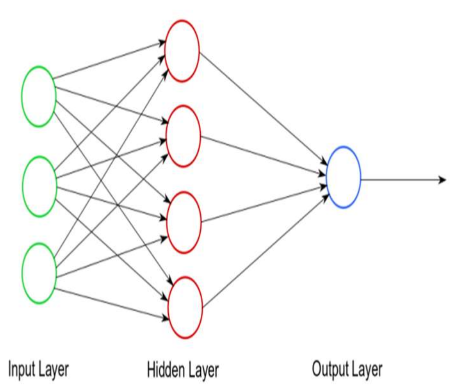
In this project, we perform classification of fashion-MNIST dataset using different types of neural networks and classified the images into different categories of clothing. The goal of this project is to build three different neural networks, namely one hidden layer neural network in python from scratch, multi-layer neural network with keras, convolutional neural network.

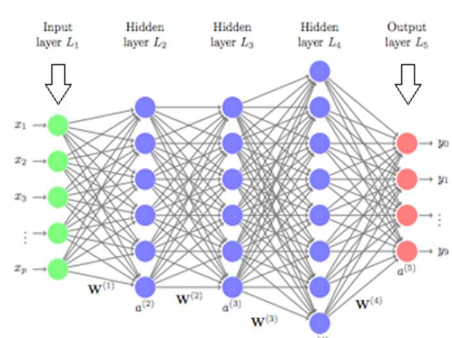
## 1 INTRODUCTION

Image classification is one of the most fundamental problems in Machine Learning. It is the core foundation for bigger problems such as Computer Vision, Face Recognition System, or Self-driving car. There are many classification models that can be used for this task; however, it is important to fully understand the concepts of each model, and how they perform on dataset. The given Fashion-MNIST data is a dataset consisting of 70,000 28x28 grayscale images of 10 different class labels. The training set has 60,000 images, and the test set has 10,000 images.

### 1.1 NEURAL NETWORKS

In simple words neural networks works like how brain performs various actions using neurons. A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. The number of layers present in between input and output layer are called hidden layers. Based upon the number of hidden layers neural networks are classified into two categories. Firstly, a simple neural network is nothing, but which has only one hidden layer which we have implemented in our part 1. Secondly, a deep neural network which has multiple hidden layers which we have implemented in part 2 and part3.




Simple neural network Deep neural network

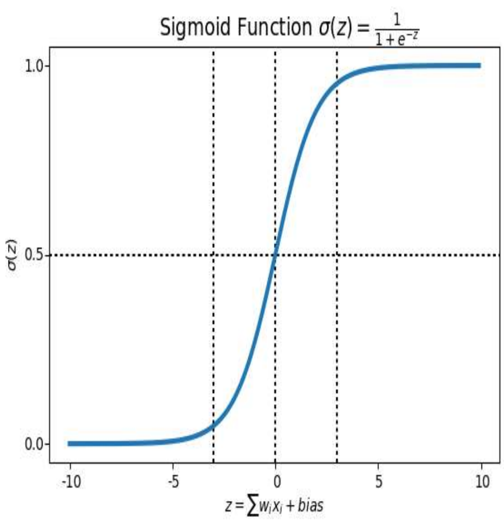
## 1.2 ACTIVATION FUNCTIONS

It's just a function that you use to get the output of node. It is also known as Transfer function. It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

Below is the list of various activation functions.

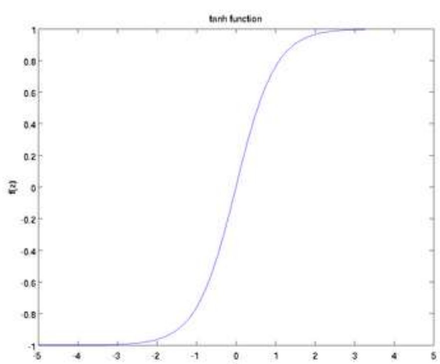
### 1.2.1 SIGMOID FUNCTION

To compress data points between -1 to +1 we use sigmoid function. The sigmoid function graph is shown below and it is ranged from -1 to +1. It is an S-shaped curve.



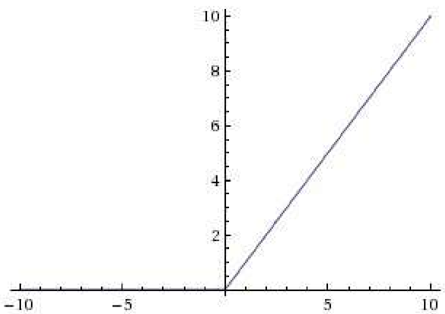

### 1.2.2 Tanh FUNCTION

Another activation function that is used is the tanh function.




### 1.2.3 ReLU FUNCTION

ReLU is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations. That is a good point to consider when we are designing deep neural nets.

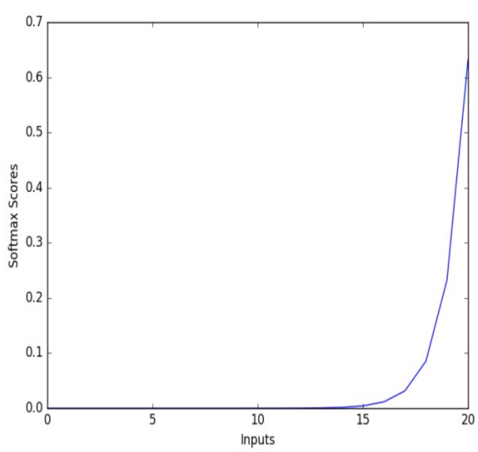



### 1.2.4 Softmax function

Softmax function calculates the probabilities distribution of the event over ‘n’ different events. In general way of saying, this function will calculate the probabilities of each target class over all possible target classes. Later the calculated probabilities will be helpful for determining the target class for the given inputs.

The main advantage of using Softmax is the output probabilities range. The range will 0 to 1, and the sum of all the probabilities will be equal to 1. If the softmax function used for multi-classification model it returns the probabilities of each class and the target class will have the high probability.

The formula computes the exponential(e-power) of the given input value and the sum of exponential values of all the values in the inputs. Then the ratio of the exponential of the input value and the sum of exponential values is the output of the softmax function.




## 1.3 Cross entropy loss function

Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label. So predicting a probability of .012 when the actual observation label is 1 would be bad and result in a high loss value. A perfect model would have a log loss of 0.

## 1.4 Hyper parameters

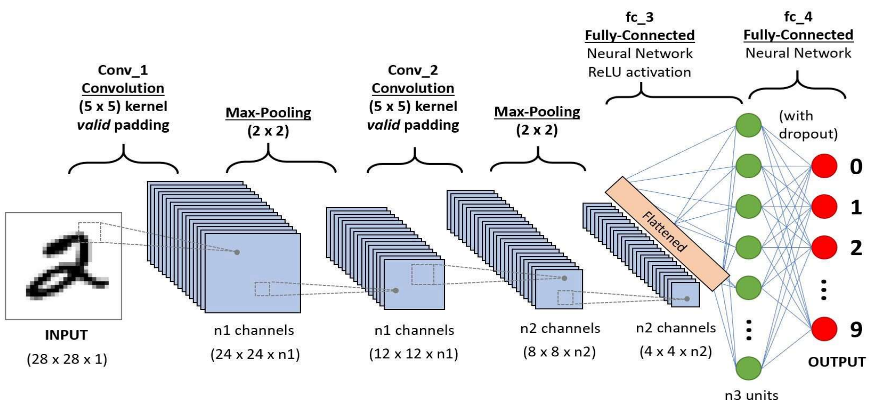
In our models we have developed, there are hyper parameters namely number of hidden layers, number of nodes in each layer, learning rate. By changing these values we improve our model.

## 1.5 Regularization

Regularization is technique for combating overfitting and improves training. There are various regularization techniques. We used early stopping technique which is very optimal regularization technique.

### 1.6 Convolutional neural network

A convolutional neural network (ConvyNet/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. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.




## 2 DATASET

For training and testing of our classifiers, we used the Fashion-MNIST 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.

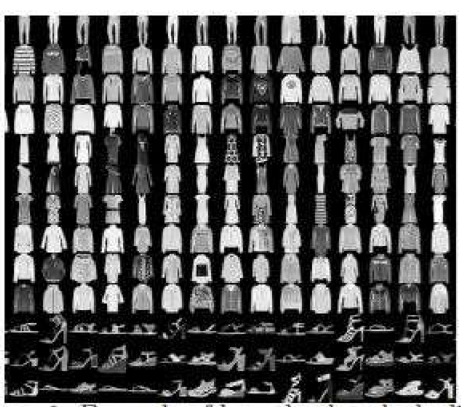


Figure 1: Example of how the data looks like.

Each training and test example is assigned to one of the labels as shown in table 1.

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

Table 1: Labels for Fashion-MNIST dataset

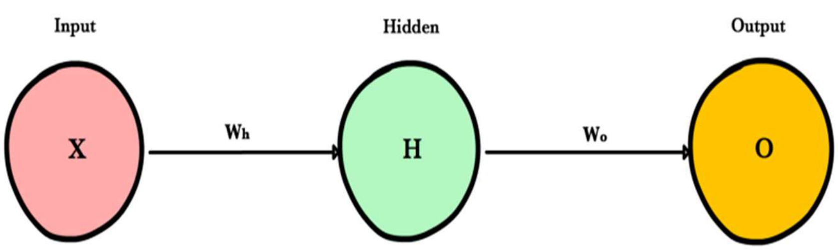
# 3 PREPROCESSING

In our given dataset, we have gray scale images whose values are in range of o to 255. So, I need to apply normalization technique. Generally, normalization is changing the range of values of data without distorting the data. In this case the data can be normalized by dividing the data with 255 so that whole data can be normalized between 0 to 1.

# 4 ARCHITECTURE

## 4.1 Single layer neural network

After preprocessing the data, we apply forward propagation. As it is a simple neural network it has only one hidden layer and one input layer and one output layer.




### 4.1.1 Forward propagation

Forward propagation is how neural networks makes predictions. Input data is “forward propagated” through the network layer by layer to the final layer which outputs a prediction. For the neural network above, a single pass of forward propagation translates mathematically to:

$$\text{Prediction}=A(A(XW_h)W_o)$$

Where A is an activation function like ReLU, X is the input and Wh and Wo are weights.

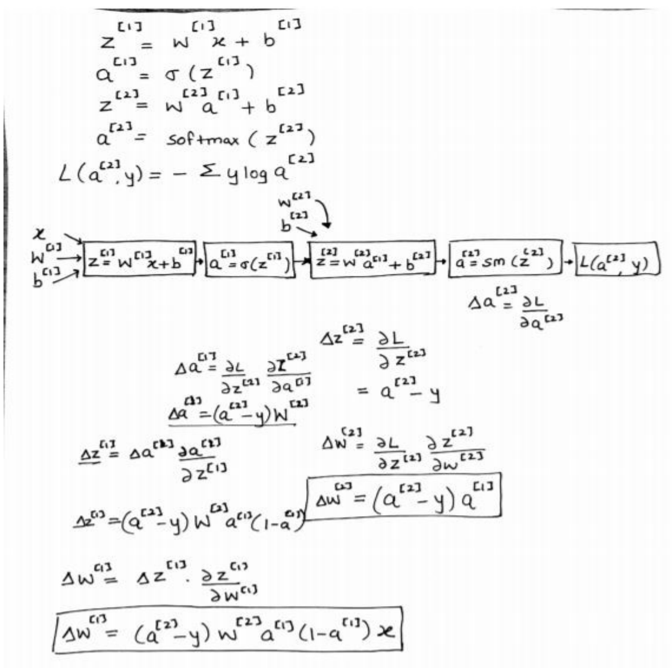
After forward propagation, we need to find cost and then we need to update bias and weights. For this we need to use back propagation.

### 4.1.2 Backward propagation

At this point we have one forward pass done, and we can compute how bad our neural network is using the negative log likelihood function. It's time to change our parameters so that on the next forward pass the neural network does better. The backpropagation step involves the propagation of the neural network's error back through the network. Based on this error the neural network's weights can be updated so that they become better at minimizing the error. This is the more math heavy part of a neural network.

#### 4.1.2.1 Gradient descent for neural networks:

Applying gradient descent to our neural network is somewhat more involved in terms of the calculus required but the basic principles are the same. We have a loss function defined and the parameters of this function are the weights and biases of our neural network. So we need to update the weights of our neural network such that the value of our loss function is minimized.




After adjusting my hyper parameters and number of epochs I was able to build my model and attained accuracy for test data of 71.7%.

## 4.2 Multilayer neural network with keras

In this multilayer neural network with keras, first we need to install keras and tensorflow for faster computations that runs on CPU, GPU. After preprocessing the data I used two hidden layers with 128 nodes in first hidden layer and 64 hidden nodes in the second hidden layer. I used relu activation function for first hidden layer. For second layer I used sigmoid activation function. For output layer I used softmax function.

For regularization I used early stopping where I used patience value of 5. According to it if it encounters 5 serial increase in cost function then it automatically stops. I had my early stopping after completion of 21 epochs. Using this two hidden layers I achieved accuracy of

93.7% and validation accuracy of 88.97%.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

## 4.3 convolutional neural network

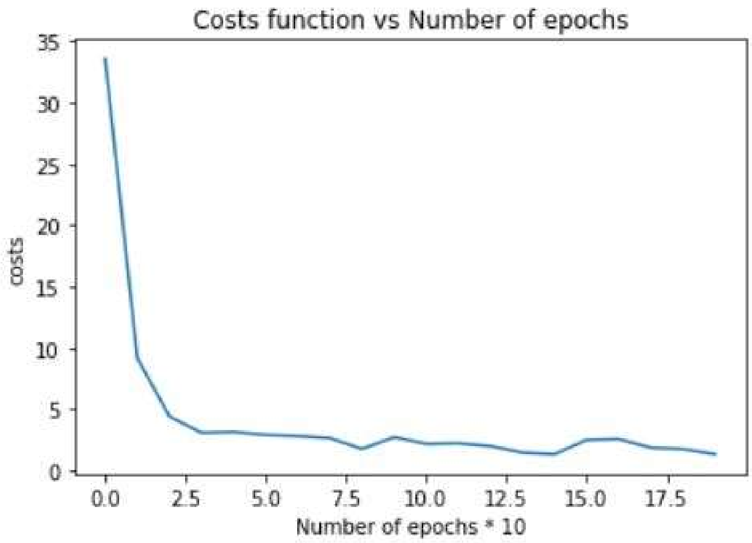
After preprocessing the data, in this convolutional neural network I used one convolutional layer with 64 neurons and used relu activation function for this first layer. The next layer is max pooling layer with pool_size=2. In the second convolutional layer I used 32 neurons and used sigmoid activation function for this layer. The next layer is max pooling layer with pool size=2. Later I flatten the image and then give a dense fully connected layer with 128 neurons with activation function of relu. Finally, the output layer consists of 10 neurons with an activation of SoftMax. I used early stopping on val_loss with patience of 5. The training stopped after 9 epochs. I got an accuracy of 93.34% and a validation accuracy of 90.57%

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 64)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        8224
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               200832
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 210,666
Trainable params: 210,666
Non-trainable params: 0
_________________________________________________________________
```

# 5 RESULTS

## 5.1 Single layer neural network

### 5.1.1 Graph between cost and number of epochs




### 5.1.2 Test Accuracy

```
acc=cal_acc(actual_y_test,pred,test_samples)
print("Test Accuracy :", acc*100)

Test Accuracy : 71.7
```

### 5.1.3 Training Accuracy

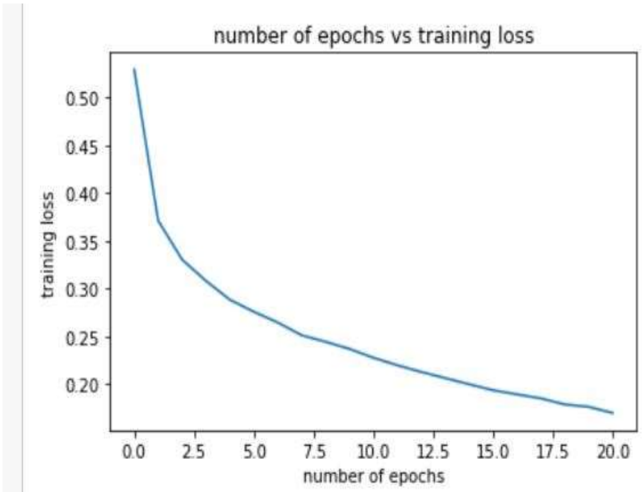
```
In [62]: print('Training Accuracy :',accuracy[-1]*100)

         Training Accuracy : 77.30166666666666
```

### 5.1.4 Confusion matrix

```
: array([[641,   7,   0,   8,  32,   4, 290,   1,  17,   0],
         [ 12, 935,   0,  14,  23,   2,   9,   1,   4,   0],
         [ 21,  11,  48,   3, 415,   2, 486,   0,  13,   1],
         [ 68,  77,   2, 506, 132,   1, 193,   0,  21,   0],
         [  3,   4,   0,   5, 841,   2, 137,   0,   8,   0],
         [  4,   1,   0,   2,   2, 853,  10,  73,  14,  41],
         [ 85,   4,   8,   7, 175,   5, 686,   0,  29,   1],
         [  1,   0,   0,   0,   1,  65,   0, 871,   4,  58],
         [ 12,   2,   0,   5,  21,  18,  56,   4, 878,   4],
         [  1,   0,   0,   0,   1,  24,   1,  56,   6, 911]], dtype=int64)
```

## 5.2 Multilayer neural networks with keras

### 5.2.1 Graph between number of epochs and training loss




### 5.2.2 Accuracy

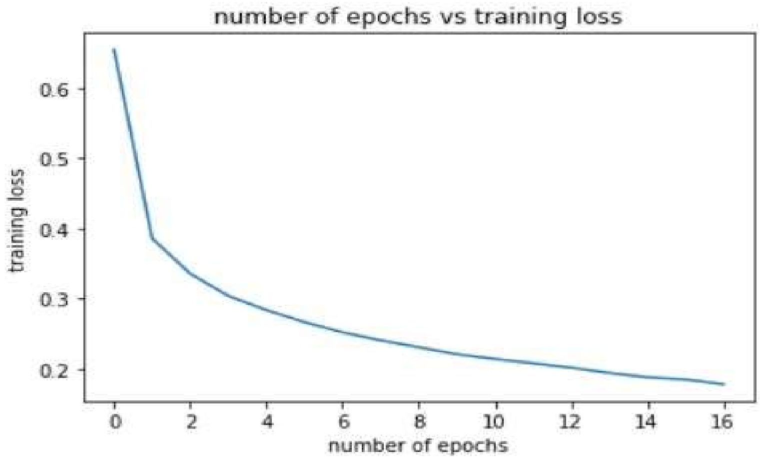
```
Epoch 18/100
60000/60000 [==============================] - 14s 228us/step - loss: 0.1850 - accuracy: 0.9297 - val_loss: 0.3337 - val_accura
cy: 0.8864
Epoch 19/100
60000/60000 [==============================] - 12s 194us/step - loss: 0.1787 - accuracy: 0.9339 - val_loss: 0.3226 - val_accura
cy: 0.8927
Epoch 20/100
60000/60000 [==============================] - 11s 191us/step - loss: 0.1762 - accuracy: 0.9341 - val_loss: 0.3346 - val_accura
cy: 0.8914
Epoch 21/100
60000/60000 [==============================] - 12s 194us/step - loss: 0.1697 - accuracy: 0.9370 - val_loss: 0.3374 - val_accura
cy: 0.8897
Restoring model weights from the end of the best epoch
Epoch 00021: early stopping
```

### 5.2.3 Confusion matrix

```
In [51]: confuson_matrix = confusion_matrix(y_test.argmax(axis=1), y_pred.argmax(axis=1))
         confuson_matrix

Out[51]: array([[849,   1,  10,  16,   3,   0, 116,   0,   5,   0],
                [  7, 978,   0,  10,   3,   0,   2,   0,   0,   0],
                [ 63,   2, 750,   9, 115,   0,  60,   0,   1,   0],
                [ 86,   7,   6, 840,  35,   0,  22,   0,   4,   0],
                [ 47,   1,  54,  13, 828,   0,  55,   0,   2,   0],
                [  4,   0,   0,   1,   0, 962,   0,  18,   2,  13],
                [159,   1,  58,  15,  51,   0, 709,   0,   7,   0],
                [  2,   0,   0,   0,   0,  10,   0, 971,   0,  17],
                [ 13,   0,   1,   4,   2,   2,   4,   3, 971,   0],
                [  1,   0,   0,   0,   0,   9,   1,  50,   0, 939]], dtype=int64)
```

## 5.3 Convolutional neural networks

### 5.3.1 Graph between number of epochs and training loss




### 5.3.2 Accuracy

```
Epoch 15/50
60000/60000 [==============================] - 42s 696us/step - loss: 0.1886 - accuracy: 0.9299 - val_loss: 0.2816 - val_accura
cy: 0.9038
Epoch 16/50
60000/60000 [==============================] - 43s 715us/step - loss: 0.1854 - accuracy: 0.9300 - val_loss: 0.2761 - val_accura
cy: 0.9038
Epoch 17/50
60000/60000 [==============================] - 41s 692us/step - loss: 0.1787 - accuracy: 0.9334 - val_loss: 0.2826 - val_accura
cy: 0.9057
Restoring model weights from the end of the best epoch
Epoch 00017: early stopping
```

### 5.3.3 Confusion matrix

```
array([[924,   0,  13,   9,   2,   0,  45,   0,   7,   0],
       [ 12, 971,   0,  14,   0,   0,   1,   0,   2,   0],
       [ 52,   0, 853,   5,  47,   0,  43,   0,   0,   0],
       [ 54,   4,  10, 896,  18,   0,  16,   0,   2,   0],
       [ 39,   0,  39,  33, 843,   0,  45,   0,   1,   0],
       [  1,   0,   0,   0,   0, 977,   0,  19,   1,   2],
       [199,   1,  51,  18,  64,   0, 658,   0,   9,   0],
       [  4,   0,   0,   0,   0,   4,   0, 985,   0,   7],
       [ 11,   1,   1,   1,   0,   2,   0,   3, 981,   0],
       [  3,   0,   0,   0,   0,   8,   0,  67,   1, 921]], dtype=int64)
```

## 6 CONCLUSION

I was successfully able to train my one hidden layer neural network from scratch with training accuracy of 77.3% and testing accuracy of 71.7%.
I achieved a validation accuracy of 88.97% using multilayer neural network with keras.
Using convolutional neural networks I achieved a validation accuracy of 90.57%
From the above results we can conclude that accuracy increases as we from one hidden layer neural network from scratch to multilayer neural network with keras to convolutional neural network while dealing with multi class problems.

### References

1) https://adventuresinmachinelearning.com/neural-networks-tutorial/#what-are-anns

2) https://www.investopedia.com/terms/n/neuralnetwork.asp

3) https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/relu_layer.html

4) https://www.kdnuggets.com/2017/10/neural-network-foundations-explained-gradient-descent.html

5) https://orbograph.com/deep-learning-how-will-it-change-healthcare/

6) https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

7) https://dataaspirant.com/2017/03/07/difference-between-softmax-function-and-sigmoid-function/

8)https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

9)http://cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf

10)http://www.cristiandima.com/neural-networks-from-scratch-in-python/

11)https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html