---

# IMPLEMENTATION OF NEURAL NETWORKS USING

# 1. USING PYTHON
# 2. USING TENSORFLOW LIBRARY
# 3. USING KERAS ADVANCED LIBRARY

**Rugved Hattekar**

**#50320920**

MASTER'S ROBOTICS AND AUTONOMOUS SYSTEMS

University at Buffalo

Buffalo, NY, 14226

**ABSTRACT:**

Neural Networks are basically set of algorithms which are inspired from human brain and are designed to recognize the patterns. The patterns are recognized using the given data on which the statistical models are implemented. Neural networks use classification algorithms to recognize the pattern in terms of logic or logic 1. The Neural Network a can be referred as modest extension of logistic regression on a multi-class problem. Neural Networks basically help to group unlabeled data according to the similarities between them. . In this report, we have discussed about the methodology of neural networks with 1 and n hidden layers, and at last we have also discussed the implementation with CNN.

**Introduction:**

Neural Networks are basically used to differentiate the large data from one another. A common practice in it is giving the large data from an image as an input and assign the weights and biases to it and classify the given data according to the values. In this report we are using 2D CNN and implementing it using Keras and TensorFlow, both of them are highly popular for implementing MNIST Data sets of image classification. The given MNIST dataset is of fashion images and asks to train the data of 50000 instances and test it on remaining 10000 instances of 28X28 image each. Through prediction the algorithm should detect the given class of the problem. MNIST data set is very known for implementing machine learning algorithms with higher accuracy. This dataset contains 60,000 greyscale which are usually classified into 10 categories. Usually images are in a low quality at 28*28 pixel with a scale between 0:255.

**Dataset:**

For training and testing of our classifiers, we will use 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. Each training and test example is assigned to one of the labels as shown in table.

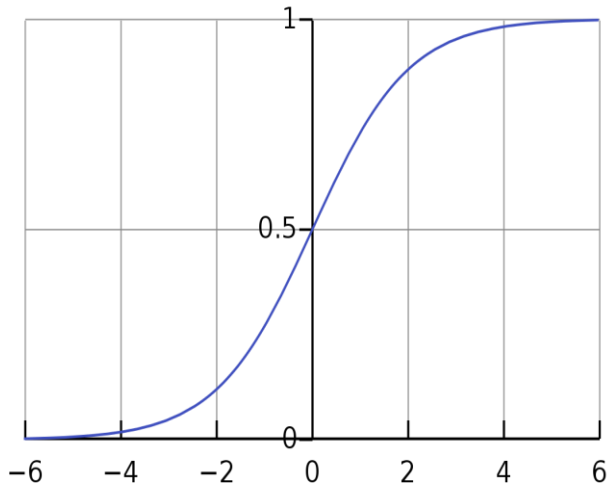
1. T-shirt/top
2. Trouser
3. Pullover
4. Dress
5. Coat
6. Sandal
7. Shirt
8. Sneaker
9. Bag
10. Ankle Boot

**Neural Networks Architecture:**

**Sigmoid Function:**

The Neural Network architecture is very important step before implementing the algorithms. In this problem we are going to implement the Cross Entropy Loss function with sigmoid function. The sigmoid function is given any input value in the range of 0 and 1. The sigmoid is a step like function. For certain problems the number 0.5 is considered as threshold and if the number is less than threshold it is assigned as 0 else 1.

$$f(x) = \frac{L}{1+e^{-k(x-x_0)}}$$




The graph of sigmoid function

**The Cross Entropy Loss:**

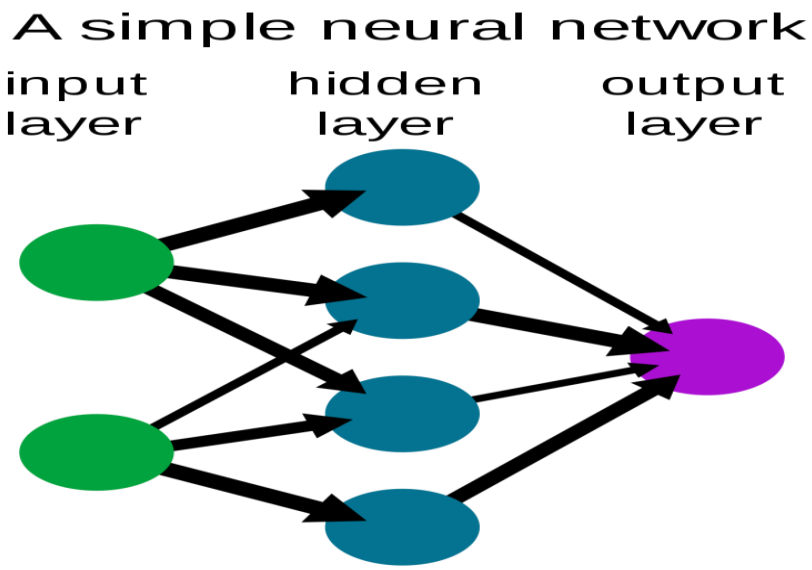
The cross entropy loss is used to measure the performance of the system. The cross entropy loss diverges when the predicted value is close to the actual value of the system. The cross-entropy loss function is the extention of loss function used in the Logistic Regression problems. The Cross entropy function is given below. The Cross entropy function summation operator is used when multiclass problem is considered.

Cross Entropy function:- $-(y\log(p)+(1-y)\log(1-p))$

Cros Entropy function when multiple classes are involved $-\sum_{c=1}^{M} y_{o,c}\log(p_{o,c})$

Where, p = $\mathbf{1/1+e^{-z}}$

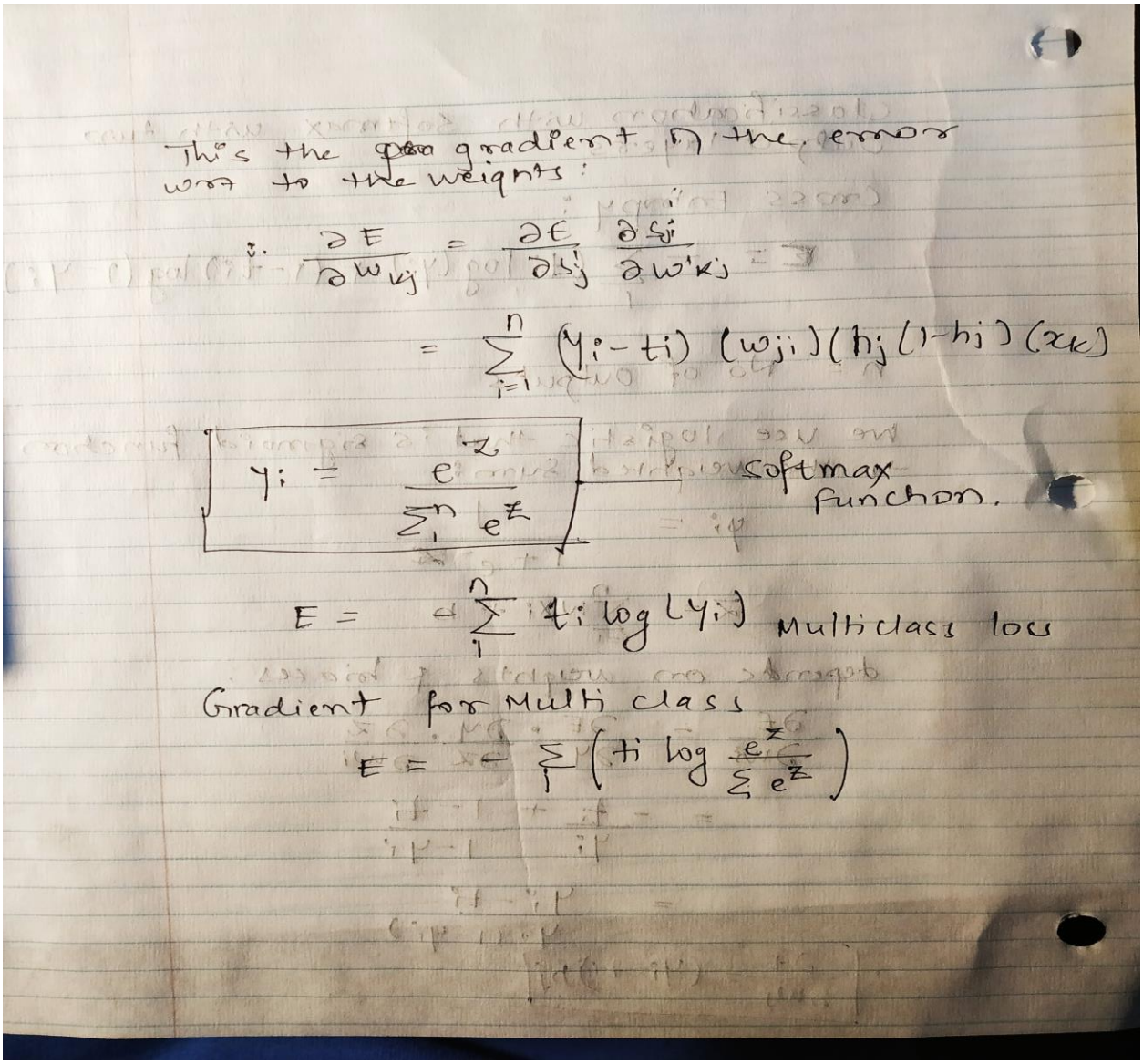
**Neural Network:**

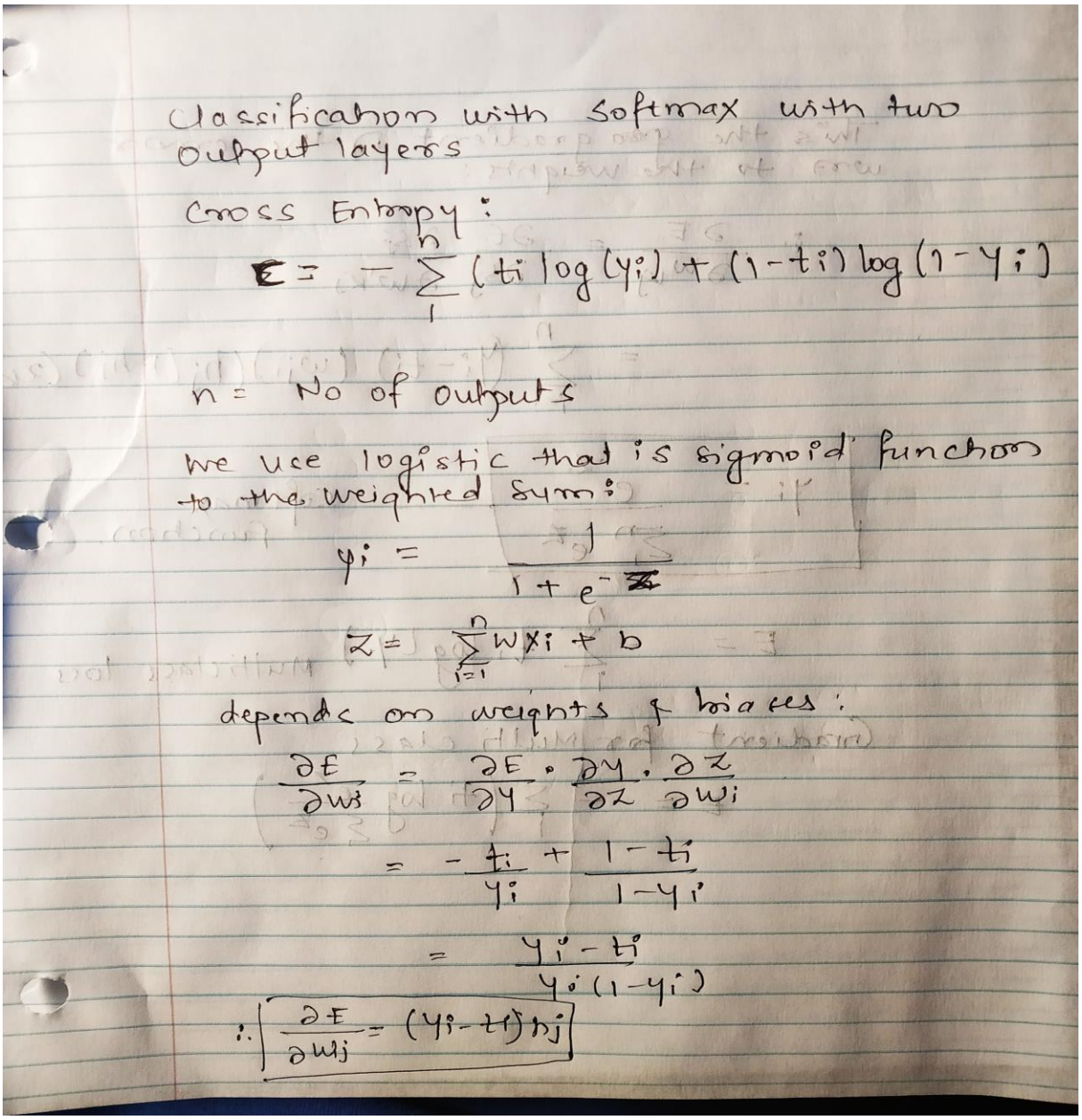



Neural networks are usually designed after taking an inspiration from human brain. They are set of algorithms with some sensory data through a kind of machine perception, altering perception. Usually they recognize every data in numbers, it doesn't matter what it is e.g.-; images, audio etc. Neural networks also known as neural nets are used for classifying and clustering. They usually group unlabeled data according to similarities among the example inputs, and they classify data when they have a labeled dataset to train on. They consist of weights and layers in which they usually store the data and pass on.

**Softmax Function:**

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 again use the cross-entropy error to derive the softmax



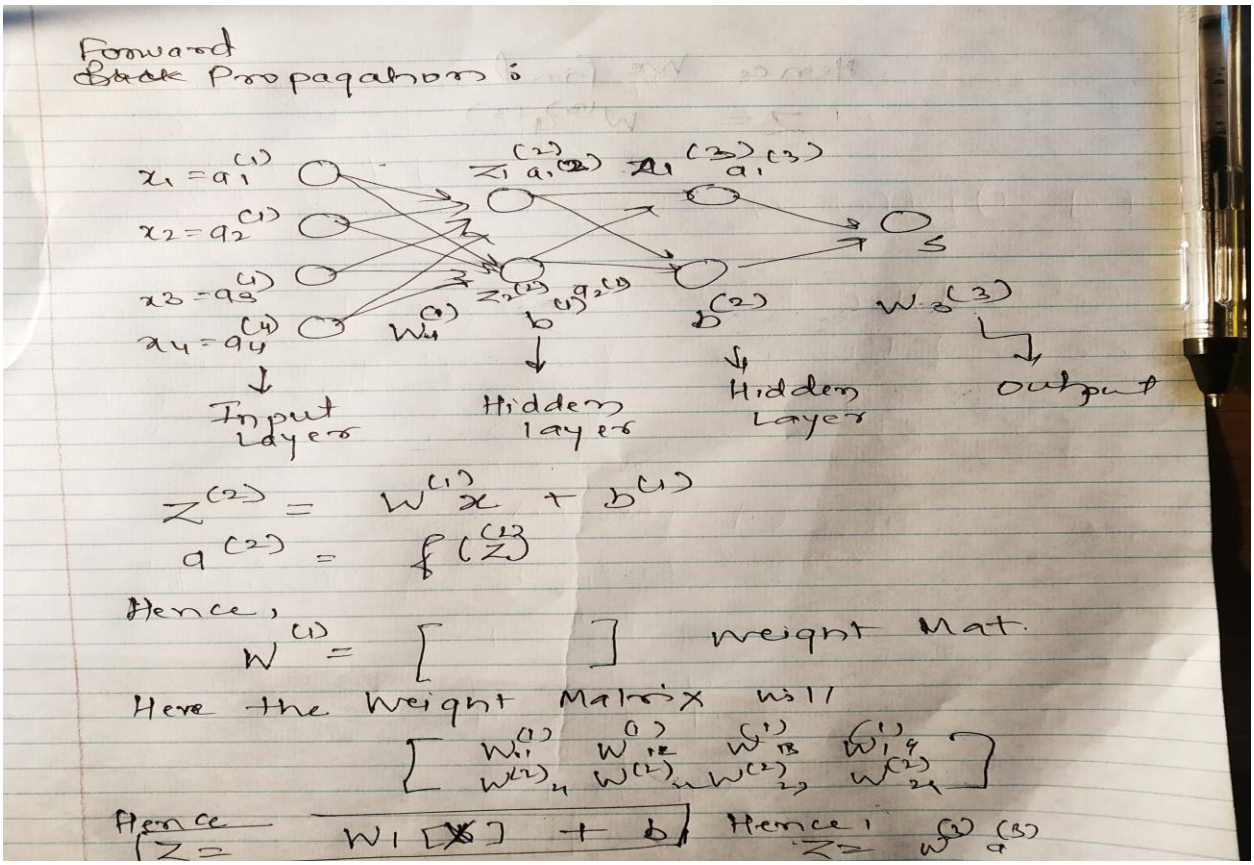





**Layers and Weights:**

A neural network consists of artificial neurons or processing elements just like the human brain consists of neurons and is organized in three interconnected layers: input, hidden that may include more than one layer, and output. Hence layers are the intermediator points through which we pass the information from one side to the other. Whereas, weights could be explained as data is broken into small input pieces. These input pieces are mixed in “right proportion” to get desired output. It is similar to chef mixing the ingredient in “right proportion” to prepare a food. “Weight” captures the “proportion” in which input pieces are mixed together.Which means weights are the values that are defined in the neural nets with which we usually carry out our operations.

Forward Propagation:



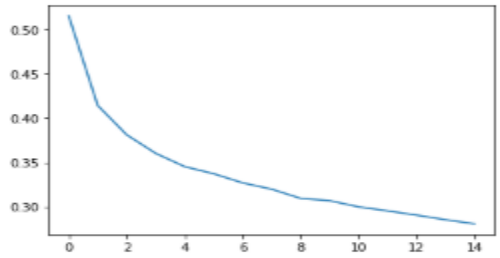

**Back Propagation:**

Back-propagation is one of the most important essence of neural net training. It is the practice of fine-tuning of the weights in a neural net based on the loss rate obtained through the previous epoch or iteration. It is basically doing the fine tuning to the forward prop model.

:

**Results:**

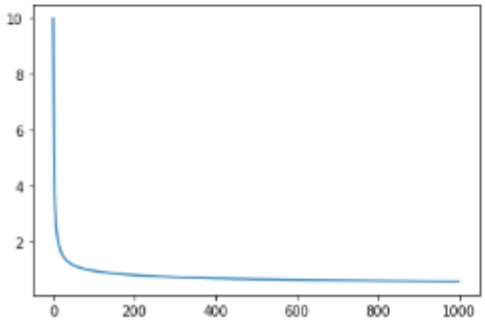
```
60000/60000 [==============================] - 4s 73us/sample - loss: 0.2853 - acc: 0.8929
Epoch 15/15
60000/60000 [==============================] - 4s 75us/sample - loss: 0.2807 - acc: 0.8962
(10000,)
(10000, 28, 28)
[[820  13  24  32   6   1  92   0  12   0]
 [  0 981   1  11   5   0   0   0   2   0]
 [ 11   2 876   5  75   0  29   0   2   0]
 [ 15  29  22 851  54   0  25   0   4   0]
 [  0   2 140  14 829   0  12   0   3   0]
 [  0   0   0   1   0 946   0  37   1  15]
 [116  15 159  24 117   0 553   0  16   0]
 [  0   0   0   0   0   5   0 978   0  17]
 [  2   0   5   3   6   2   1   3 978   0]
 [  0   0   0   1   0   8   1  50   0 940]]
              precision    recall  f1-score   support

           0       0.85      0.82      0.84      1000
           1       0.94      0.98      0.96      1000
           2       0.71      0.88      0.79      1000
           3       0.90      0.85      0.88      1000
           4       0.76      0.83      0.79      1000
           5       0.98      0.95      0.96      1000
           6       0.78      0.55      0.65      1000
           7       0.92      0.98      0.95      1000
           8       0.96      0.98      0.97      1000
           9       0.97      0.94      0.95      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.87     10000
weighted avg       0.88      0.88      0.87     10000
```




Results Using TensorFlow

The Maximum accuracy of 88% with min cost function of 0.28 and 15 epochs.

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
Epoch 0 cost:  9.990735441552415
Epoch 100 cost:  0.9483808670685154
Epoch 200 cost:  0.7967245511928275
Epoch 300 cost:  0.7241546753117123
Epoch 400 cost:  0.679970144201329
Epoch 500 cost:  0.6493260322568961
Epoch 600 cost:  0.6261850200327886
Epoch 700 cost:  0.6077483856736959
Epoch 800 cost:  0.5924867018550029
Epoch 900 cost:  0.5794712481653643
Final cost: 0.5682830176778694
```

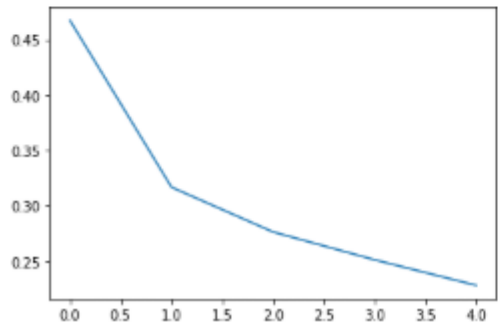



```
[[781   9  20  37   0   1 170   0   6   1]
 [  8 929   2  25   4   1   5   0   3   0]
 [ 25  14 653  17 123   0 134   0  12   2]
 [ 75  35  12 816  35   0  57   0  10   0]
 [ 11   8 159  42 700   1 151   0   6   0]
```

Results Python Code From Scratch.

The maximum accuracy of 89% is achieved using Python Code from Scratch. The lowest cost function value is 0.568 for 1000 epoch and 0.01 learning rate.

```
Epoch 5/5
60000/60000 [==============================] - 32s 541us/step - loss: 0.2280 - acc: 0.9171
10000/10000 [==============================] - 2s 209us/step
Test loss 0.2632779253363609
Test accuracy 0.9047
[[861   3  22   8   6   1  95   0   4   0]
 [  3 983   0  12   1   0   1   0   0   0]
 [ 14   2 851   8  52   0  73   0   0   0]
 [ 25  10  10 901  20   0  34   0   0   0]
 [  1   1  28  35 880   1  53   0   1   0]
 [  0   0   0   0   0 976   0  14   1   9]
 [134   2  61  21  78   0 699   0   5   0]
 [  0   0   0   0   0  14   0 963   0  23]
 [  2   1   6   3   4   3   9   3 969   0]
 [  0   0   0   0   0   4   1  31   0 964]]
              precision    recall  f1-score   support

           0       0.83      0.86      0.84      1000
           1       0.98      0.98      0.98      1000
           2       0.87      0.85      0.86      1000
           3       0.91      0.90      0.91      1000
           4       0.85      0.88      0.86      1000
           5       0.98      0.98      0.98      1000
           6       0.72      0.70      0.71      1000
           7       0.95      0.96      0.96      1000
           8       0.99      0.97      0.98      1000
           9       0.97      0.96      0.97      1000

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```




Results using keras.

Note: Due to less computing power I had to use less epochs. But The accuracy is still considerably good.

**Conclusions:**

Thus, we were able to see how Machine Learning algorithms can be put into fashion industry. I tried out tuning the hyper parameters with various values and found out that the best result of training accuracy for neural nets with 1 layer was 88 percent. For n layer neural nets, it was 89 percent and for CNN it was 91 percent. These all were done for 100, 100 and 50 epochs(iterations) respectively. This kind of result can be obtained even if you increase your learning rate, hence by increasing your iterations you can increase your accuracy. Hence from this project we can conclude that CNN is one of the most powerful tool for image classification.