Image Classification Using Neural Networks On MNIST Dataset

Prudhveer Reddy Kankar
50320121
prudhvee@*buffalo.edu*

Abstract

In this project we implement neural network and convolutional neural network for the task of classification. A classification task will be that of recognizing an image and identify it as one of ten classes i.e T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle Boot. We train the classifiers using Fashion-MNIST clothing images. This project report provides a brief explanation of how to use a to perform the below three tasks :

1. Building a Neural Network with one hidden layer from scratch in python which is to be trained and tested on Fashion-MNIST dataset.
2. Building a multi-layer Neural Network with open-source neural-network library, Keras on Fashion- MNIST dataset.
3. Building Convolutional Neural Network (CNN) with open-source neural-network library, Keras on Fashion-MNIST dataset.

1 Introduction

There are basically two types of problems i.e.. Regression Problem and Classification Problem. Regression problems basically deals with predicting continuous valued outputs , where as classification problems deal with discrete valued outputs (0,1,2 etc). We use neural networks in this project to solve classification problems.

1.1 Neural Networks

The whole purpose of Deep Learning is to mimic how human brains work. The neurons form the basis of neural networks. Neural networks try to recreate and copy the functionalities of a neuron. Artificial Neural Networks systems learn to perform tasks by considering examples, generally without being programmed with task-specific rules.
Convolutional Neural Network are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers. There are four main steps in CNN. They are:

- Convolution
- Pooling
- Flattening
- Full Connection

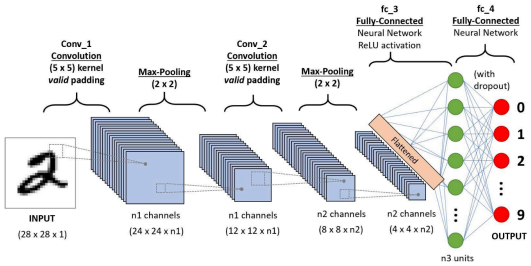


Figure 1: CNN Steps.

These steps are explained in brief in the 4.3 section of this report.

1.1.2 Gradient descent

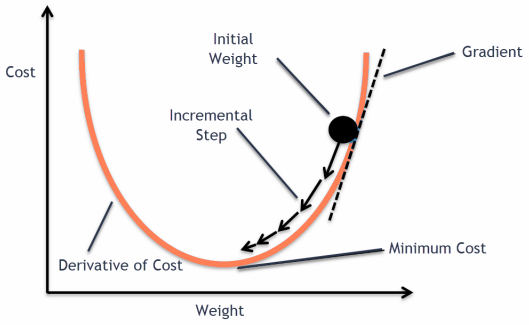


Figure 2:Gradient Descent Graph.

We use Gradient descent finding the minimum of a function. In our machine learning algorithm, we use gradient descent to continuously update the parameters of our model. The formula is given by

$$\text{repeat until convergence } \{ \quad \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad \}$$

Where j can be 0 and 1. Alpha is the learning rate. We have the select alpha in such a way that it has to find out the global minimum without overshooting or taking more that average time. In our problem we update weights and bias.
It is referred to as “categorical_crossentropy” while coding the loss

1.1.3 Activation Function : Rectifier (ReLu)

In computational networks, the activation function of a node defines the output of that node for set of inputs. The function of rectifier is given by

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \Rightarrow 0 \end{cases}$$

In simple words we can say that it removes the negative part of the function. The graph is given as follows:

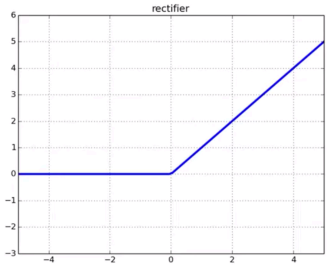


Figure 3:Rectifier function output Graph.

1.1.4 Activation Function : Sigmoid

We apply a sigmoid function to make sure that the predictions made by the algorithm stays between 0 or 1.The sigmoid function and the output graph is given in the below figure.

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

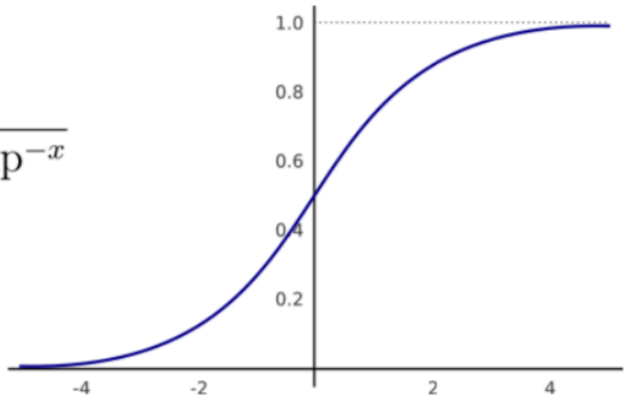


Figure 4: Sigmoid function and Logistic Regression output value graph.

As we can see from the above graph the output range of the predictions is limited to values between 0 and 1. This is the example of two class problem. We then set the threshold values (predicted value>=0.5 or predicted value < 0.5) to predict the to which class the prediction belongs to.

1.1.5 Confusion Matrix:

A confusion matrix is also known as an error matrix. It is a visualization table that allows us to compute the performance of an algorithm.

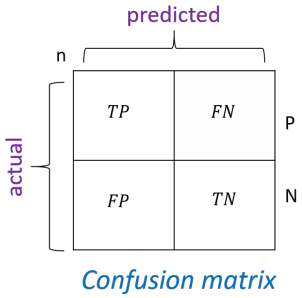


Figure 4: Confusion Matrix

Where TP = True Positives, TN = True Negatives, FP = False Positives, and FN = False Negatives.We generate a confusion matrix to the test data and compute accuracy, precision and recall by using the below formulas.

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

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

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

2 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

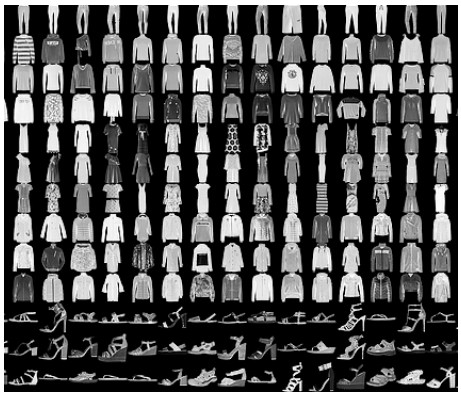


Figure 5: Example of how the data looks like.

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.

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

Reading:
I simply loaded the Fashion MNIST dataset using fashion mnist reader notebook present inside the scripts folder.

Partitioning:
The entire data was partitioned into two parts: training and test.

Normalization:
The data dimensions were normalized so that they are of approximately the same scale. The data values were converted into float32 type and then divided by value 255 so that the values lie between 0 and 1 and it becomes easy for computation.

4 Architecture

4.1 Building a Neural Network with one hidden layer from scratch in python which is to be trained and tested on Fashion-MNIST dataset.

Step 1. Preprocessing the Data:
The dataset given has to be preprocessed as explained in section 3. The data has to be divided so that we get test data and train data on which the testing and training operations can be performed.
We use pandas get.dummys function to convert the output values class vectors (integers) into 2 dimensional binary class matrix with 10 classes.

Step 2. Building Forward Pass and Softmax:
A neural network is a function that maps inputs to outputs. One single pass or computation of this function is known as a forward pass. The network takes some input from the input layer, passes it through the hidden layer and arrives at the output. We are gonna work on a NN with one hidden layer, similar to the figure shown below.

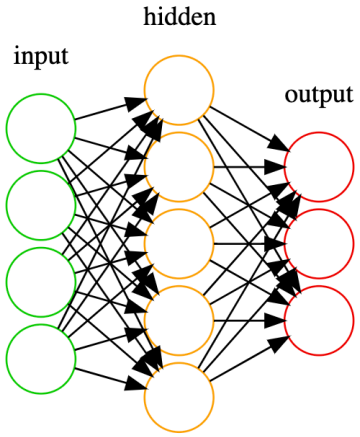


Figure 6: ANN.

We have 60000 samples and 784 features with form a 60000x784 matrix. The hidden layer has 512 and finally the output layer has 10 nodes i.e. 10 classifications.
We basically take the dot product of weight matrix and multiply with the number of features in each sample. We also consider a bias term which allows to shift the function up or down and thus produce a better model for our data.This value plus the bias term is sent into a sigmoid function which is an activation function as shown below.

$$A = \sigma(Z) = \sigma(XW + b)$$

Now that we have computed the hidden layer it's time to move forward in the network and arrive at the final layer, the output layer. The calculation here are very similar to the previous one except that instead of input data we consider the output from the hidden layer and we use softmax as activation function. The softmax function produces a probability distribution. It assigns higher probabilities to the higher numbers and lower probabilities to the lower numbers.

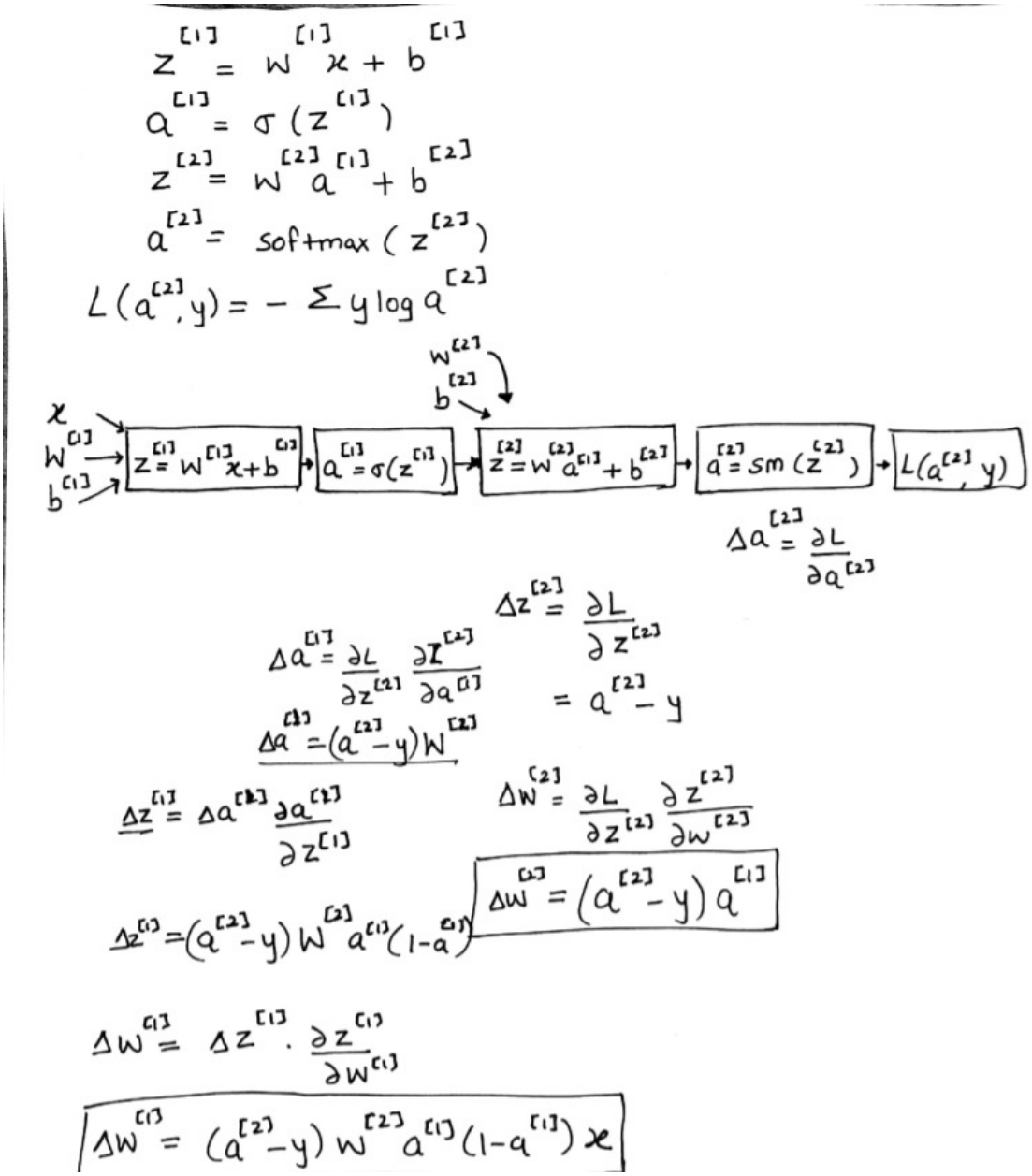
Step 3. Softmax activation:
To compute the values associated with the output nodes we will first use the same linear combination we used in the previous step. Given the hidden layer activation values A, a weights matrix W2 describing the connections between the hidden layer and the output layer, and a bias vector b2 we can compute a new set of Z2 such that

$$Z^{(2)} = AW^{(2)} + b^{(2)}$$

Step 4. Building Back-propagation and Gradient Descent:

Only forward passes produce very bad results. To overcome this, we use the concept of back propagation. We use an error function which calculates the difference between output produced after each pass of front propagation and original given output. The back-propagation 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. We use the concept of Gradient Descent explained in 1.1.2 for the purpose of updating the values.

The entire set of formulas can be summarized from the below figure.



4.2 Building a multi-layer Neural Network with open-source neural-network library, Keras on Fashion- MNIST dataset.

The architecture of multi-layer Neural Network algorithm used is this project is explained briefly in a stepwise manner:

Step 1. Preprocessing the Data:
The dataset given has to be preprocessed as explained in section 3. The data has to be divided so that we get test data and train data on which the testing and training operations can be performed.
We use to_categorical function to convert y_train and y_test class vectors (integers) to binary class matrix with 10 classes.

Step 2. Installing and Importing Tensorflow and Keras:
Tensorflow is an open source numerical computations library that runs very fast computations , which run on CPU and GPU. Originally developed by google and now under Apache 2.0 license.
Keras wraps the tensor flow library which is used to build deep learning models in a very few lines of code. It is based on tensor flow and theano libraries.
We use both these libraries in this project.

Step 3. Initializing and adding hidden layers and tuning the HyperParameters:
We import sequential library form keras and initialize our classifier which is basically our neural network. Sequential class means that our neural network contains sequence of layers.
The hyper parameters I.e.. number of nodes in each hidden layer was decided after a series of trials and tests.
We now use the sequential classifier and use the add function to add our first input layer. In the first input layer, we use the dense function and specify the number of output dimensions, activation functions and also the number of input dimensions. In this project we have used 512 output dimensions, the rectifier activation function for activation and 784 input nodes as given in the dataset.
Similarly we then add the second hidden layer with 256 output dimensions and the rectifier activation function. We do not mention the input dimensions as it was already previously defined.
Finally we add the output layer with 10 outputs and use the sigmoid activation function.

Step 5. Compiling the Neural Network:
Finally adding all the layers we compile the classifier, We use the gradient descent algorithm for calculating the losses. In this project we are using accuracy metric to calculate the correctness.

Step 4. Fitting the ANN to the Training set and tuning the Epochs:
Finally we fit the classifier which is our neural network to the training and testing data. We define the batch size and the number of epochs. It is optional to define the batch size. For example if we define the batch size as 10, then the weights get updated after every 10 observations or else if its not defined , the weights get updated after each observation. First argument in fit method has the input arguments (training sets), then we define the batch size, number of epochs and then the testing or the validation set. An epoch is defined as one iteration through the dataset. Each epoch includes a forward propagation and then a backward propagation.
Forward Propagation goes from left to right , the neurons are activated in a way that the impact of each input activation is limited by weights. Propagate the citations until we get the results. We have tuned the hyper parameter epochs after a series of trial and test method and
Then we compare the predicted result to the actual result and measure the generated error.
We then backprogate from right to left and simultaneously update the weights according to the generated error. The learning rate decided how much we update weights.

4.3 Building Convolutional Neural Network (CNN) with open-source neural-network library, Keras on Fashion-MNIST dataset.

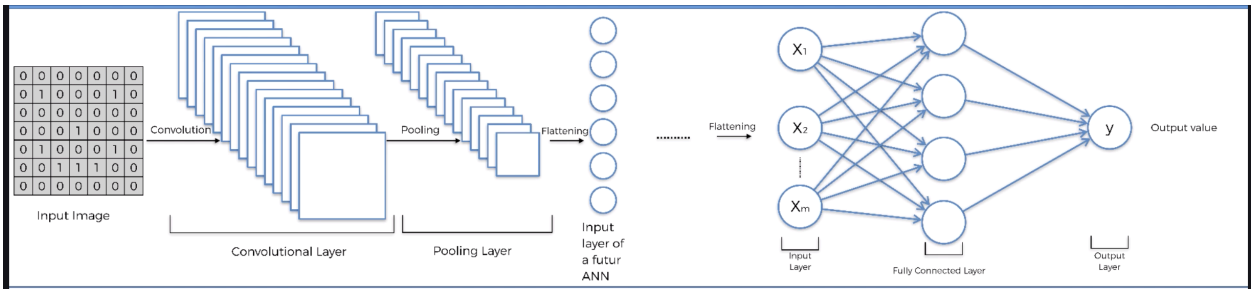


Figure 7: Steps in CNN.

Step 1. Preprocessing the Data:
The dataset given has to be preprocessed as explained in section 3. The data has to be divided so that we get test data and train data on which the testing and training operations can be performed.
We use to_categorical function to convert y_train and y_test class vectors (integers) to binary class matrix with 10 classes.

Step 2. Installing and Importing Tensorflow and Keras:
Tensorflow is an open source numerical computations library that runs very fast computations , which run on CPU and GPU. Originally developed by google and now under Apache 2.0 license.
Keras wraps the tensor flow library which is used to build deep learning models in a very few lines of code. It is based on tensor flow and theano libraries.
We use both these libraries in this project.

Step 3. Initializing CNN:
We import sequential library form keras and initialize our classifier which is basically our neural network. Sequential class means that our neural network contains sequence of layers.

Step 4. Convolution:
Convolution step consists of applying feature detectors on the input image. For each feature detector we apply on the input image, we get a feature map. The highest number in the feature map is where the feature detector can detect a particular feature in the input image. The consolation layer is composed of all the feature maps.
We using Conv2D keras method to do convolution. We have to pass the number of rows and columns in the feature detector and the number of filters as our first argument in the Conv2D method. We have used 42 feature detectors and 3x3 matrix for feature detectors.
Next is the input shape. We have to give the number of pixels and the number of layers as input. Since we are using a greyscale image, we define the number of layers as only 1. And last we using rectifier activation function.

Step 5. Pooling:
It is used to reduce the size of the feature maps. We can use max pooling or min pooling. The size of the original image is reduced by half when we use pooling. Applying pooling makes it less competitive intensive.
We use the MaxPooling2D function for pooling. We have to define the pool size which is 2x2. The pool size is nothing but the size of the matrix that strides over the feature detectors and determine if the maximum or minimum pooling.

Step 6. Flattening:
In this step we take all pooled feature maps and put it into a one single vector. This basically becomes the input layer. We use the flatten function for this. No need to input any parameters, keras automatically know that the previous layers has to be flattened.

Step 7. Full Connection and and tuning Hyper Parameters - hidden layers:
The hyper parameters I.e.. number of nodes in each hidden layer was decided after a series of trials and tests.
In this step we have to create the hidden layers which are similar to those in ANN. We add the hidden layer with 512 output dimensions and the rectifier activation function. We do not mention the input dimensions as it was already previously defined.
Finally we add the output layer with 10 outputs and use the sigmoid activation function.

Step 8. Compiling the Neural Network:

Finally adding all the layers we compile the classifier, We use the gradient descent algorithm for calculating the losses. In this project we are using accuracy metric to calculate the correctness.

Step 9. Fitting the ANN to the Training set and tuning the hyper parameters - Epochs:

Finally we fit the classifier which is our neural network to the training and testing data. We define the batch size and the number of epochs. It is optional to define the batch size. For example if we define the batch size as 10, then the weights get updated after every 10 observations or else if its not defined , the weights get updated after each observation. First argument in fit method has the input arguments (training sets), then we define the batch size, number of epochs and then the testing or the validation set. An epoch is defined as one iteration through the dataset. Each epoch includes a forward propagation and then a backward propagation.
Forward Propagation goes from left to right , the neurons are activated in a way that the impact of each input activation is limited by weights. Propagate the citations until we get the results. We have tuned the hyper parameter epochs after a series of trial and test method and
Then we compare the predicted result to the actual result and measure the generated error.
We then backprogate from right to left and simultaneously update the weights according to the generated error. The learning rate decided how much we update weights.

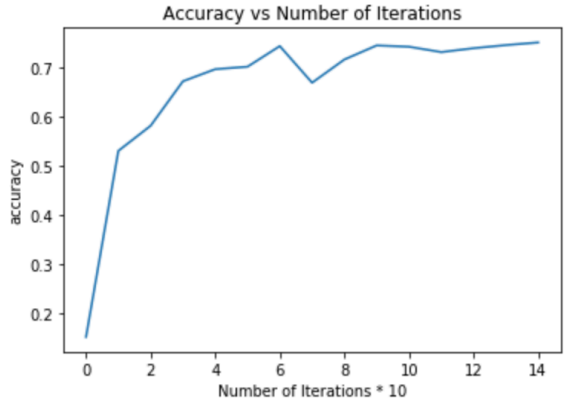
5 Results

5.1 Building a Neural Network with one hidden layer from scratch in python which is to be trained and tested on Fashion-MNIST dataset.

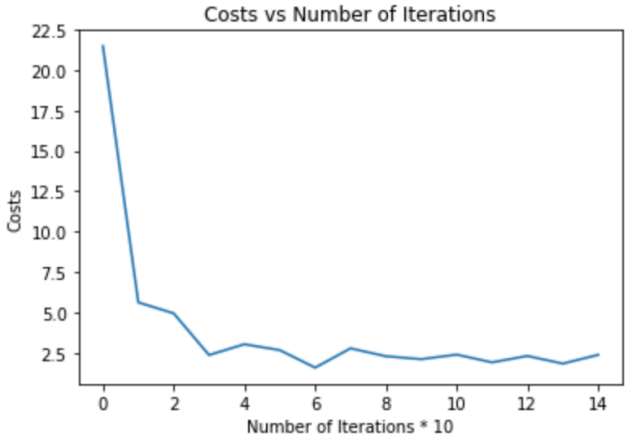
Loss and Training accuracy :

```
Loss function value:  21.463904830558807
Training Accuracy after  0  iterations :  15.110000000000001
Loss function value:  5.6293639272374785
Training Accuracy after  10  iterations :  52.99166666666667
Loss function value:  4.95418857956858
Training Accuracy after  20  iterations :  58.10333333333333
Loss function value:  2.3766386049177117
Training Accuracy after  30  iterations :  67.16333333333333
Loss function value:  3.0437540842813893
Training Accuracy after  40  iterations :  69.62
Loss function value:  2.678812526473803
Training Accuracy after  50  iterations :  70.125
Loss function value:  1.595063154733021
Training Accuracy after  60  iterations :  74.32333333333332
Loss function value:  2.788878642002924
Training Accuracy after  70  iterations :  66.86999999999999
Loss function value:  2.3014587872692807
Training Accuracy after  80  iterations :  71.595
Loss function value:  2.1202969379281935
Training Accuracy after  90  iterations :  74.46000000000001
Loss function value:  2.4009786421665282
Training Accuracy after  100  iterations :  74.19166666666666
Loss function value:  1.923610917677939
Training Accuracy after  110  iterations :  73.10166666666666
Loss function value:  2.318679758839456
Training Accuracy after  120  iterations :  73.89
Loss function value:  1.845769196387244
Training Accuracy after  130  iterations :  74.52499999999999
Loss function value:  2.3884188460175992
Training Accuracy after  140  iterations :  75.05833333333334
```

Accuracy vs Number of Iterations Graph:



Costs vs Number of Iterations Graph:



Confusion Matrix:

```
array([[880,  16,  19,  25,  18,   7,  11,   1,  23,   0],
       [  9, 933,   4,  25,  22,   1,   2,   1,   3,   0],
       [ 45,  12, 469,  12, 412,   4,  22,   0,  24,   0],
       [114,  43,   8, 671, 136,   6,   7,   2,  12,   1],
       [  5,   6,  30,   6, 925,   6,  10,   0,  12,   0],
       [  3,   0,   0,   2,   3, 838,   0,  71,  26,  57],
       [288,  17,  91,  19, 395,  11, 127,   1,  50,   1],
       [  0,   0,   0,   0,   0,  88,   0, 842,   8,  62],
       [ 25,   4,   6,  12,  22,  22,   3,  17, 886,   3],
       [  2,   0,   0,   1,   1,  41,   1,  66,   6, 882]])
```

Test Data Accuracy:

```
Number of test samples: 10000
Test Accuracy with the current Model: 74.53
```

5.2 Building a multi-layer Neural Network with open-source neural-network library, Keras on Fashion- MNIST dataset.

Accuracy :

```
val_accuracy: 0.8807
Epoch 22/30
60000/60000 [==============================] - 37s 610us/step - loss: 0.1980 - accuracy: 0.9260 - val_loss: 0.4929 -
val_accuracy: 0.8837
Epoch 23/30
60000/60000 [==============================] - 36s 602us/step - loss: 0.1949 - accuracy: 0.9270 - val_loss: 0.5300 -
val_accuracy: 0.8837
Epoch 24/30
60000/60000 [==============================] - 36s 604us/step - loss: 0.1966 - accuracy: 0.9269 - val_loss: 0.5012 -
val_accuracy: 0.8887
Epoch 25/30
60000/60000 [==============================] - 36s 604us/step - loss: 0.1892 - accuracy: 0.9293 - val_loss: 0.5364 -
val_accuracy: 0.8872
Epoch 26/30
60000/60000 [==============================] - 36s 602us/step - loss: 0.1912 - accuracy: 0.9287 - val_loss: 0.5675 -
val_accuracy: 0.8805
Epoch 27/30
60000/60000 [==============================] - 36s 604us/step - loss: 0.1910 - accuracy: 0.9291 - val_loss: 0.6069 -
val_accuracy: 0.8876
Epoch 28/30
60000/60000 [==============================] - 36s 605us/step - loss: 0.1824 - accuracy: 0.9315 - val_loss: 0.6471 -
val_accuracy: 0.8817
Epoch 29/30
60000/60000 [==============================] - 36s 594us/step - loss: 0.1858 - accuracy: 0.9324 - val_loss: 0.6339 -
val_accuracy: 0.8833
Epoch 30/30
60000/60000 [==============================] - 36s 607us/step - loss: 0.1830 - accuracy: 0.9322 - val_loss: 0.6199 -
val_accuracy: 0.8849
```

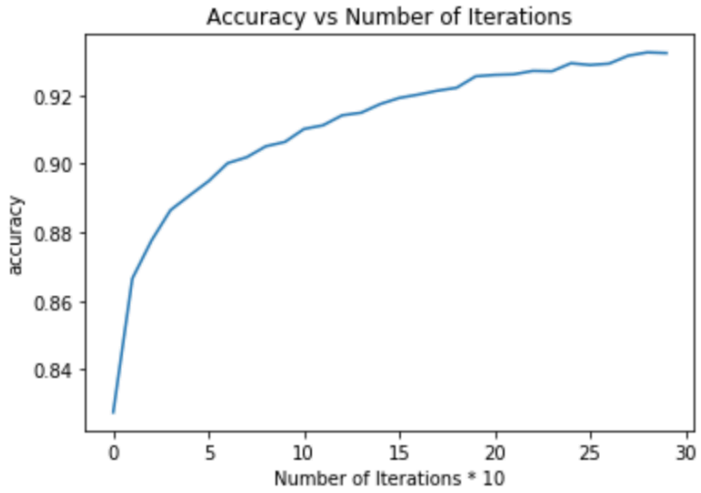
Summary:

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 27, 27, 32)        160
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_18 (Dense)             (None, 256)               1384704
_________________________________________________________________
dense_19 (Dense)             (None, 10)                2570
=================================================================
Total params: 1,387,434
Trainable params: 1,387,434
Non-trainable params: 0
_________________________________________________________________
```

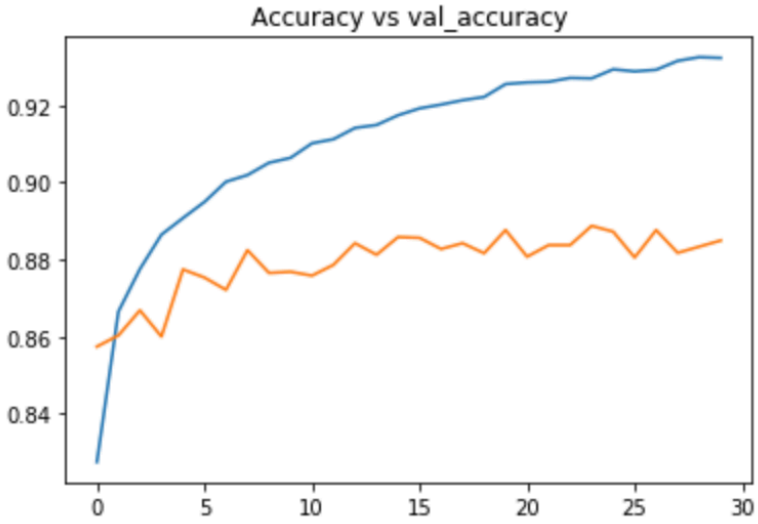
Confusion Matrix:

```
array([[826,   4,  12,  39,   1,   1, 112,   0,   5,   0],
       [  6, 973,   0,  14,   6,   0,   0,   0,   1,   0],
       [ 49,   1, 830,  14,  68,   0,  37,   0,   1,   0],
       [ 31,   5,   6, 940,   6,   0,  10,   0,   2,   0],
       [ 40,   0, 112,  61, 765,   0,  21,   0,   1,   0],
       [  1,   0,   0,   1,   0, 965,   0,  15,   0,  18],
       [174,   0,  95,  36,  93,   0, 600,   0,   2,   0],
       [  0,   0,   0,   0,   0,  10,   0, 973,   0,  17],
       [ 22,   1,   1,   4,   1,   3,   2,   3, 963,   0],
       [  1,   0,   0,   0,   0,   5,   1,  48,   0, 945]])
```

Accuracy vs Number of Iterations Graph :



Accuracy vs val_accuracy:



5.3 Building Convolutional Neural Network (CNN) with open-source neural-network library, Keras on Fashion-MNIST dataset.

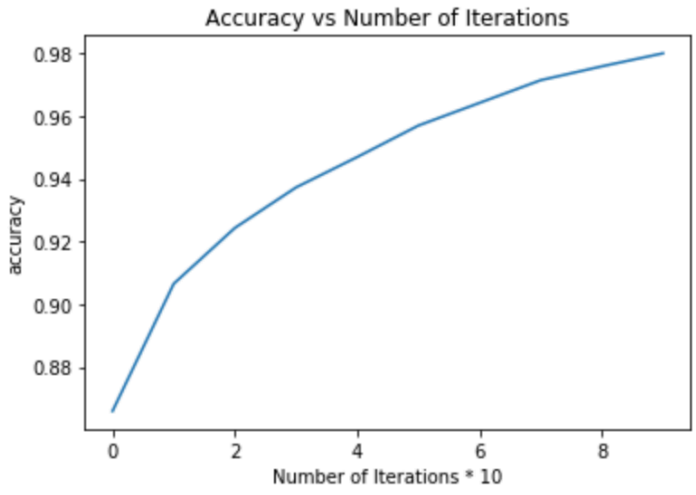
Summary:

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 27, 27, 32)        160
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_18 (Dense)             (None, 256)               1384704
_________________________________________________________________
dense_19 (Dense)             (None, 10)                2570
=================================================================
Total params: 1,387,434
Trainable params: 1,387,434
Non-trainable params: 0
_________________________________________________________________
```

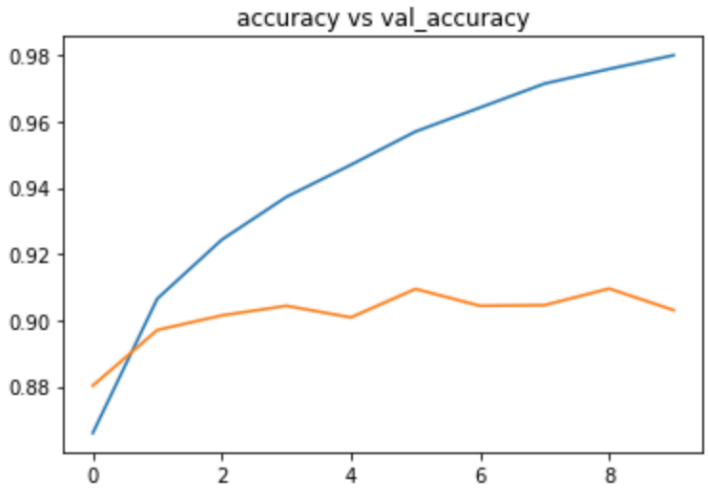
Accuracy :

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 107s 2ms/step - loss: 0.3721 - accuracy: 0.8660 - val_loss: 0.3207 - v
al_accuracy: 0.8803
Epoch 2/10
60000/60000 [==============================] - 108s 2ms/step - loss: 0.2540 - accuracy: 0.9066 - val_loss: 0.2738 - v
al_accuracy: 0.8971
Epoch 3/10
60000/60000 [==============================] - 111s 2ms/step - loss: 0.2048 - accuracy: 0.9244 - val_loss: 0.2785 - v
al_accuracy: 0.9015
Epoch 4/10
60000/60000 [==============================] - 112s 2ms/step - loss: 0.1702 - accuracy: 0.9373 - val_loss: 0.2788 - v
al_accuracy: 0.9044
Epoch 5/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.1397 - accuracy: 0.9470 - val_loss: 0.3047 - v
al_accuracy: 0.9009
Epoch 6/10
60000/60000 [==============================] - 105s 2ms/step - loss: 0.1149 - accuracy: 0.9570 - val_loss: 0.3034 - v
al_accuracy: 0.9095
Epoch 7/10
60000/60000 [==============================] - 109s 2ms/step - loss: 0.0947 - accuracy: 0.9642 - val_loss: 0.3485 - v
al_accuracy: 0.9044
Epoch 8/10
60000/60000 [==============================] - 107s 2ms/step - loss: 0.0767 - accuracy: 0.9715 - val_loss: 0.3682 - v
al_accuracy: 0.9046
Epoch 9/10
60000/60000 [==============================] - 117s 2ms/step - loss: 0.0653 - accuracy: 0.9759 - val_loss: 0.3892 - v
al_accuracy: 0.9096
Epoch 10/10
60000/60000 [==============================] - 116s 2ms/step - loss: 0.0542 - accuracy: 0.9800 - val_loss: 0.4575 - v
al_accuracy: 0.9031
```

Accuracy vs Number of Iterations Graph :



Accuracy vs val_accuracy:



Confusion Matrix:

```
array([[819,   0,   7,  23,   3,   1, 142,   0,   5,   0],
       [  1, 979,   0,  13,   2,   0,   4,   0,   1,   0],
       [ 23,   1, 824,  10,  65,   0,  76,   0,   1,   0],
       [ 19,   7,   8, 917,  14,   0,  30,   0,   5,   0],
       [ 12,   2,  40,  40, 849,   1,  54,   0,   2,   0],
       [  2,   0,   0,   0,   0, 989,   0,   1,   0,   8],
       [ 87,   3,  45,  16,  43,   0, 796,   0,   9,   1],
       [  3,   0,   0,   0,   0,  72,   0, 902,   0,  23],
       [  5,   0,   0,   5,   0,   4,   5,   0, 981,   0],
       [  1,   0,   0,   0,   0,   7,   0,  22,   3, 967]])
```

6 Conclusions

Successfully completed training, validation and testing on the given MNIST dataset. I have received the following values for evaluation metrics.

Artificial Single Layer Neural Network :
Accuracy : 75.05
Val Accuracy : 74.53

Artificial Multi Layer Neural Network with Keras:
Accuracy : 93.22
Val Accuracy : 88.49

Convolution Neural Network with Keras:
Accuracy : 98.00
Val Accuracy : 90.31

From the above values we can say that CNN works better than ANN.

7 References

[1] Neural networks from scratch in Python http://www.cristiandima.com/neural-networks-from-scratch-in-python/

[2] Keras https://keras.io/

[3] Confusion Matrix https://www.geeksforgeeks.org/confusion-matrix-machine-learning/

[4]Gradient Descent https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html

[5] Confusion Matrix https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html