# Project 2: Neural Networks

**Snehashish Mandal**
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
snehashi@buffalo.edu

## Abstract

The goal of this project is to implement neural network and convolutional neural network for the task of classification. It is for a 10-class problem. The classification task is recognizing an image and identify it as one of ten classes. The classifiers have been trained using Fashion-MNIST clothing images, which are 28X28 pixels in size. There are 60000 such images are available for training the model and 10000 images are available for validating and testing the model. The first classifier used is a neural network with one hidden layer, second one is a multi-layer neural network and finally a convolutional neural network has been implemented. For training the first model, stochastic gradient descent with backward propagation method is used to update the weights. The activation function used for the hidden layer is sigmoid whereas for the output layer, softmax has been used as the activation function. The model has been validated using a validation set data by comparing the training loss and validation loss of the model. Then we use the trained model to predict the output of the training dataset.

## 1 Introduction

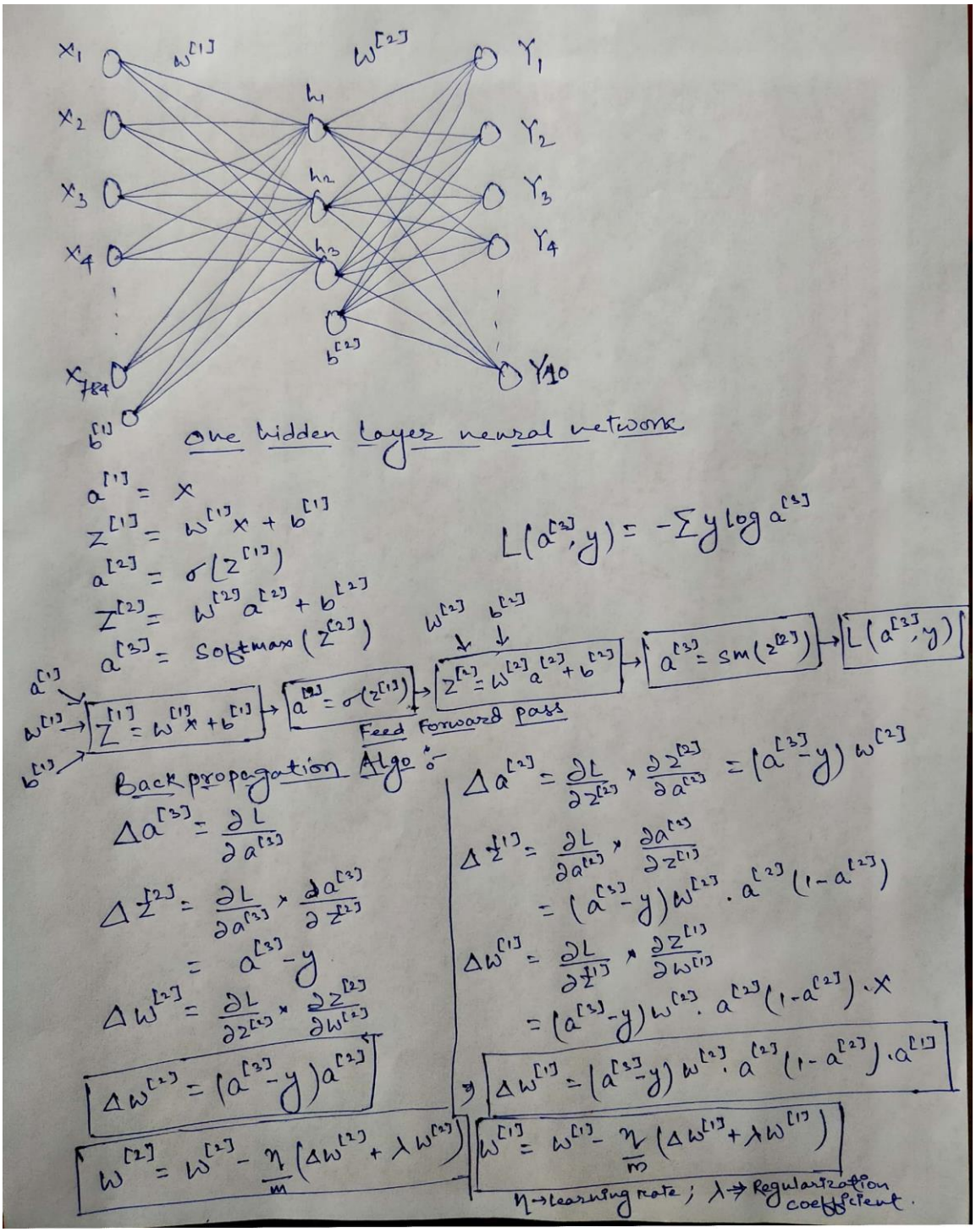
Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labeling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated. Neural networks help us cluster and classify. [1]
In this project we need to identify an image and classify it as one of the ten classes. The model has been trained using the Fashion-MNIST clothing images. There are three tasks performed in this project:

- Implemented a Neural Network with one hidden layer, trained and tested on Fashion-MNIST images from scratch using numpy.
- Implemented a multi-layer Neural Network with open-source neural-network library, Keras on Fashion-MNIST dataset.
- Implemented Convolutional Neural Network (CNN) with open-source neural-network library, Keras on Fashion-MNIST dataset.

### 1.1 Neural Networks

In the first part of the project, a neural network with one hidden layer has been implemented using numpy. The input of the model is a 28X28 pixels of image, i.e, 784 features. The output of the model are ten classes from 1 to 10 which corresponds to 10 different clothing images. The number of hidden layers is a hyper-parameter and it needs to be tuned. The number of hidden layers used in this model is 75. Multiple number (25, 50, 75, 100) of hidden layers node has been used to tune the model and by comparing the accuracy and loss,

75 is set as the number of the hidden layer nodes. The algorithm used to train the model is Stochastic Gradient Descent. In normal gradient descent method, we train the model using all the examples in a single epoch and then update the weights, whereas in stochastic gradient descent we divide the whole dataset randomly into mini-batches in every epoch and then after every mini-batch we update the weights. The batch size used in this project is 200 and the number of epochs used is 30. Also, while updating the weights, L2 regularization has been used with lambda = .001. The back-propagation algorithm is explained below:



### 1.2 Convolutional Neural Networks

A Convolutional Neural Network (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 CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNN have the ability to learn these filters/characteristics. A CNN is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights, hence introducing sparsity. In other words, the network can be trained to understand the sophistication of the image better.

The role of the CNN is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets. [2]

The following architecture has been used for the model:

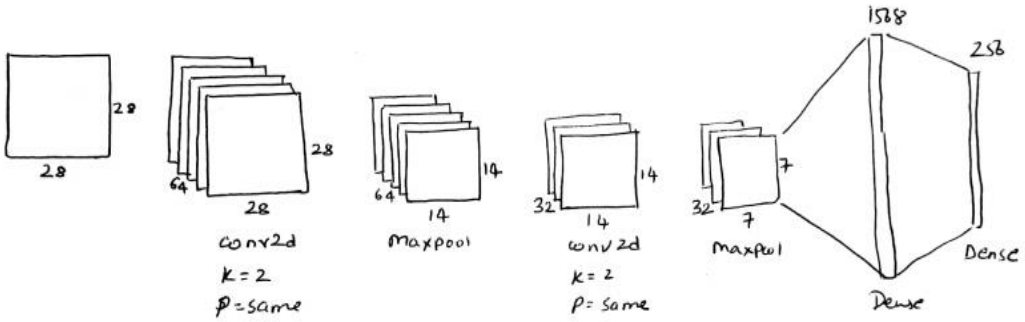



*Figure 1. Model used for CNN*

## 2 Dataset

For training and testing of our classifiers, the Fashion-MNIST dataset has been used. 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, and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image. [3]

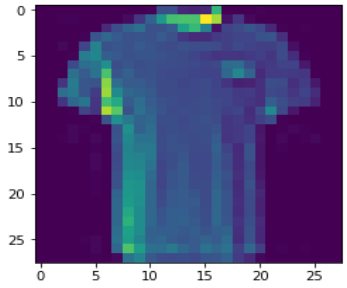



*Figure 2. Image from Fashion-MNIST dataset*

The labels are:

1 T-shirt/top

2 Trouser

3 Pullover

4 Dress

5 Coat

6 Sandal

7 Shirt

8 Sneaker

9 Bag

10 Ankle Boot

## 3 Preprocessing

The preprocessing step includes data partitioning, training of the model using stochastic gradient descent with back propagation algorithm, tuning the hyper-parameters and finally testing the model using the test dataset.

### 3.1 Data Splitting

The Fashion-MNIST dataset consists of 60000 examples of training data and 10000 examples of testing data. The testing dataset has further been split into two equal parts, validation set and testing set respectively. The model has been trained using the training dataset of 60k examples.

## 3.2 Tuning the hyper parameters

Hit and trial method has been used to tune the hyper parameters learning rate, number of epochs, batch size, regularization coefficient and number of hidden layer nodes. Validation of the model has been done using the validation dataset and by looking at the loss and accuracy of the model on the training and validation set, the optimal values of the hyper-parameters has been set and then the model has been tested on the testing dataset.

## 3.3 Testing the model

For each classifier, evaluation on the test dataset has been done using the classification accuracy.

$$\text{Accuracy} = \frac{Ncorrect}{N}$$

Where, Ncorrect = number of corrected classified data samples

N = the total number of samples of the validation/testing set.

The accuracy for the single layer neural network is 72.96%, for multi-layer neural network is 87.94% and for convolutional neural network is 89.66%.

## 4 Results

```
Epoch 1: 1644 / 5000
Epoch 2: 2361 / 5000
Epoch 3: 2648 / 5000
Epoch 4: 2827 / 5000
Epoch 5: 2947 / 5000
Epoch 6: 3057 / 5000
Epoch 7: 3127 / 5000
Epoch 8: 3199 / 5000
Epoch 9: 3232 / 5000
Epoch 10: 3284 / 5000
Epoch 11: 3318 / 5000
Epoch 12: 3327 / 5000
Epoch 13: 3336 / 5000
Epoch 14: 3322 / 5000
Epoch 15: 3353 / 5000
Epoch 16: 3393 / 5000
Epoch 17: 3406 / 5000
Epoch 18: 3423 / 5000
Epoch 19: 3437 / 5000
Epoch 20: 3455 / 5000
Epoch 21: 3474 / 5000
Epoch 22: 3487 / 5000
Epoch 23: 3500 / 5000
Epoch 24: 3507 / 5000
Epoch 25: 3517 / 5000
Epoch 26: 3529 / 5000
Epoch 27: 3561 / 5000
Epoch 28: 3569 / 5000
Epoch 29: 3560 / 5000
Epoch 30: 3569 / 5000
Testing Accuracy: 72.96
```

*Figure 3. Results for 1 hidden layer neural network*

```
y: 0.8584
Epoch 21/30
60000/60000 [==============================] - 5s 79us/step - loss: 0.2741 - accuracy: 0.8966 - val_loss: 0.3965 - val_accurac
y: 0.8640
Epoch 22/30
60000/60000 [==============================] - 3s 52us/step - loss: 0.2708 - accuracy: 0.8982 - val_loss: 0.4012 - val_accurac
y: 0.8610
Epoch 23/30
60000/60000 [==============================] - 3s 50us/step - loss: 0.2725 - accuracy: 0.8970 - val_loss: 0.4033 - val_accurac
y: 0.8634
Epoch 24/30
60000/60000 [==============================] - 4s 60us/step - loss: 0.2649 - accuracy: 0.8991 - val_loss: 0.4238 - val_accurac
y: 0.8600
Epoch 25/30
60000/60000 [==============================] - 4s 74us/step - loss: 0.2627 - accuracy: 0.9011 - val_loss: 0.4038 - val_accurac
y: 0.8696
Epoch 26/30
60000/60000 [==============================] - 4s 68us/step - loss: 0.2516 - accuracy: 0.9043 - val_loss: 0.3993 - val_accurac
y: 0.8712
Epoch 27/30
60000/60000 [==============================] - 4s 70us/step - loss: 0.2512 - accuracy: 0.9048 - val_loss: 0.3973 - val_accurac
y: 0.8740
Epoch 28/30
60000/60000 [==============================] - 4s 68us/step - loss: 0.2539 - accuracy: 0.9037 - val_loss: 0.4311 - val_accurac
y: 0.8652
Epoch 29/30
60000/60000 [==============================] - 5s 78us/step - loss: 0.2476 - accuracy: 0.9061 - val_loss: 0.4103 - val_accurac
y: 0.8674
Epoch 30/30
60000/60000 [==============================] - 4s 64us/step - loss: 0.2457 - accuracy: 0.9070 - val_loss: 0.3998 - val_accurac
y: 0.8692
5000/5000 [==============================] - 0s 95us/step
Testing Accuracy: 87.94
```

*Figure 4. Last 10 epoch results for multi-layer neural network*

```
Train on 60000 samples, validate on 5000 samples
Epoch 1/3
60000/60000 [==============================] - 51s 850us/step - loss: 1.0122 - accuracy: 0.8242 - val_loss: 0.3559 - val_accura
cy: 0.8680
Epoch 2/3
60000/60000 [==============================] - 51s 849us/step - loss: 0.2912 - accuracy: 0.8946 - val_loss: 0.3028 - val_accura
cy: 0.8904
Epoch 3/3
60000/60000 [==============================] - 51s 854us/step - loss: 0.2439 - accuracy: 0.9108 - val_loss: 0.2911 - val_accura
cy: 0.9020
5000/5000 [==============================] - 1s 293us/step
Testing Accuracy: 89.66
```

*Figure 5. Results for CNN*

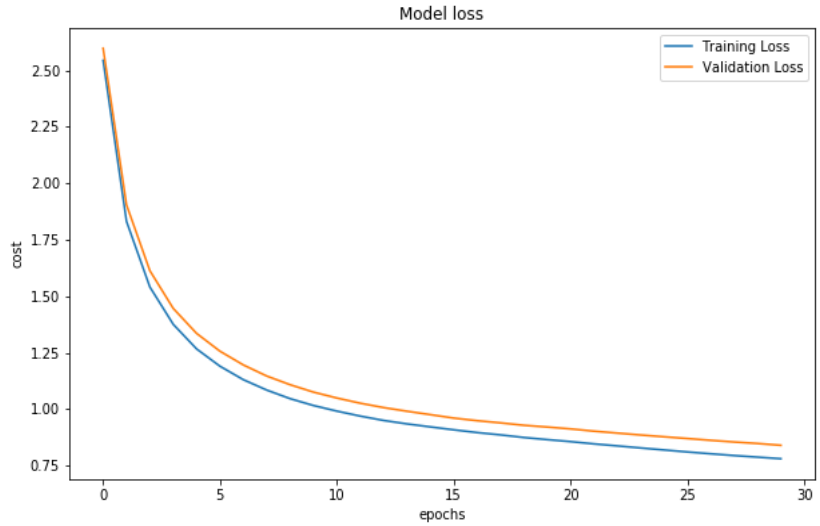



*Figure 6. Loss vs epochs for 1 hidden layer NN*

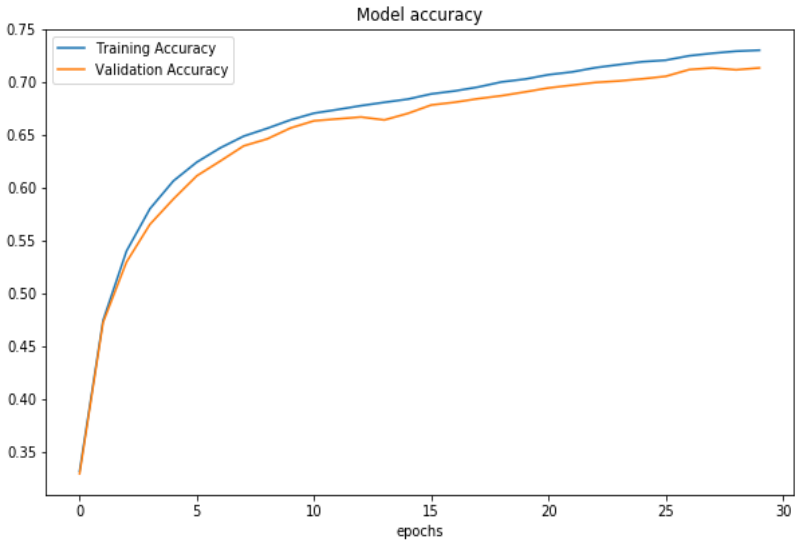



*Figure 7. Accuracy vs epochs for 1 hidden layer NN*

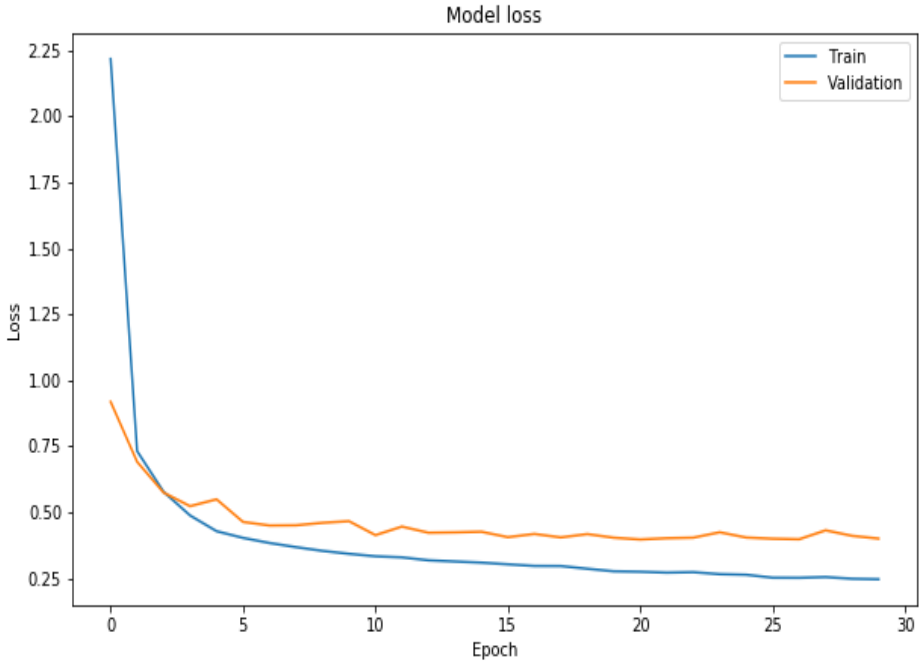



*Figure 8. Loss vs epochs for multi-layer NN*

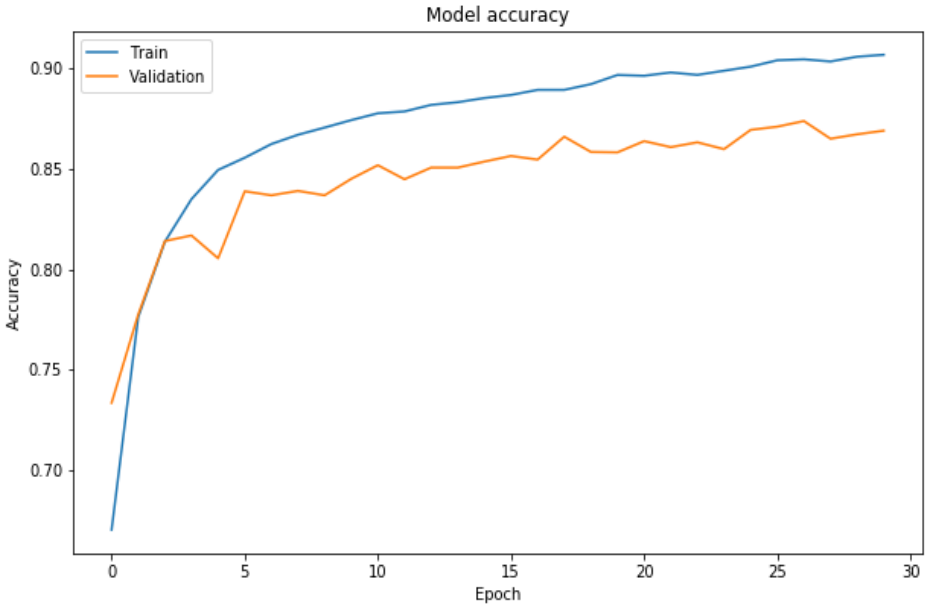



*Figure 9. Accuracy vs epochs for multi-layer NN*

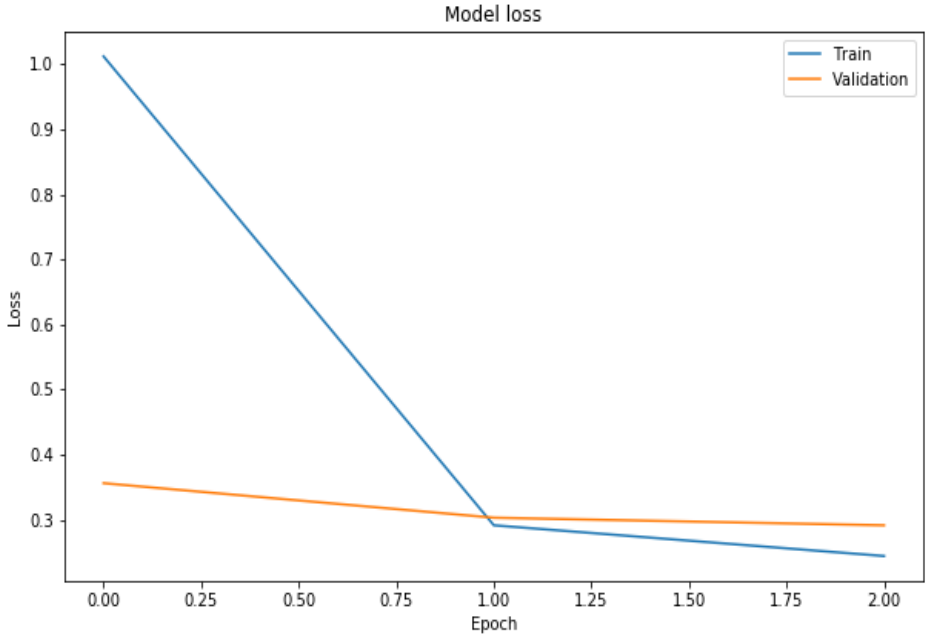



*Figure 10. Loss vs epochs for CNN*

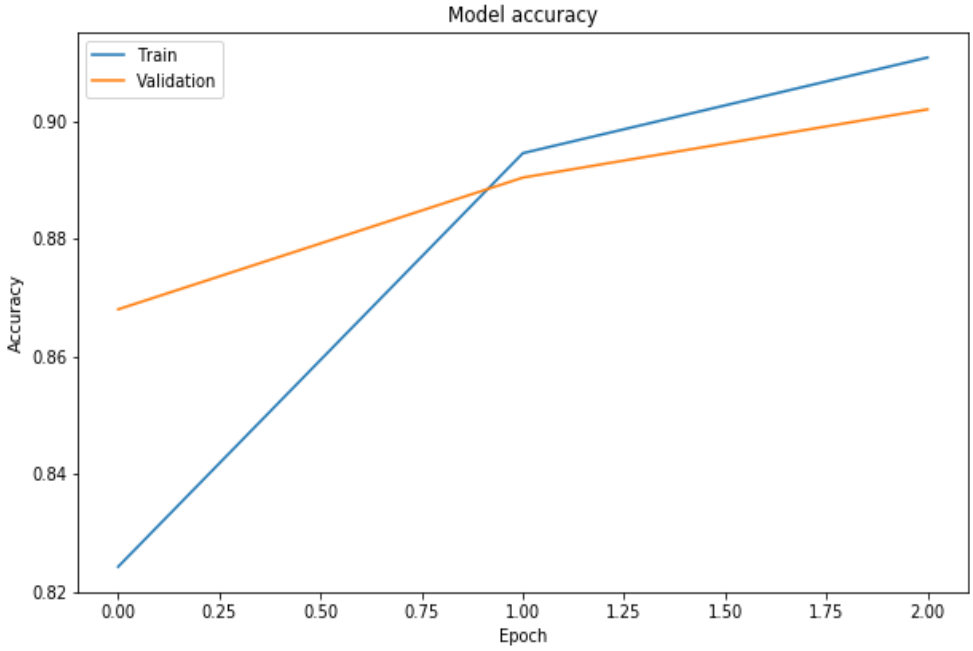



*Figure 11. Accuracy vs epochs for CNN*

```
Confusion Matrix for 3 Layer Network

[[353   7   9  34   9   2  55   0  17   0]
 [ 11 462   3  25   8   0   5   0   0   0]
 [ 10   4 293   4  91   2  78   0  14   0]
 [ 37  22   9 404  23   0  18   0   4   0]
 [  4   5  97  25 303   0  55   0   8   0]
 [  2   1   0   1   2 361   4  50  22  58]
 [ 92   0  64  33  82   2 190   0  29   0]
 [  0   0   0   0   0  33   0 420  10  45]
 [ 15   0   5   6   2  12  16   5 436   2]
 [  0   0   0   0   0  20   1  39   4 426]]
```

*Figure 12. Confusion matrix for 1 hidden layer NN*

```
Confusion Matrix for Multi Layer Network

[[420   0  11  15   0   0  34   0   6   0]
 [  0 495   0  12   4   0   3   0   0   0]
 [  4   1 425   0  39   0  27   0   0   0]
 [ 12   8   7 465  14   0   8   0   2   1]
 [  0   0  50  23 404   0  20   0   0   0]
 [  0   0   0   0   0 478   0  10   2  11]
 [ 91   0  59  17  47   0 271   0   7   0]
 [  0   0   0   0   0   9   0 468   0  31]
 [  1   1   0   2   0   0   2   2 490   1]
 [  0   0   0   0   0   3   0   6   0 481]]
```

*Figure 13. Confusion matrix for multi-layer NN*

```
Confusion Matrix for CNN

[[409   2   4   8   1   1  80   0   2   0]
 [  0 494   0  11   0   0   2   0   1   0]
 [ 10   0 411   5  20   0  46   0   0   0]
 [  5   1   4 475  11   0  26   0   1   0]
 [  1   1  43  19 393   0  43   0   0   0]
 [  1   0   0   1   0 479   0   7   0   7]
 [ 35   1  19  19  30   0 374   0   1   0]
 [  0   0   0   0   0   2   0 482   0  11]
 [  1   1   0   5   1   1   5   0 472   0]
 [  0   0   0   0   0   4   0  17   0 494]]
```

*Figure 14. Confusion matrix for CNN*

Classification Report for 3 Layer Network

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| T-shirt/top | 0.67 | 0.73 | 0.70 | 486 |
| Trouser | 0.92 | 0.90 | 0.91 | 514 |
| Pullover | 0.61 | 0.59 | 0.60 | 496 |
| Dress | 0.76 | 0.78 | 0.77 | 517 |
| Coat | 0.58 | 0.61 | 0.60 | 497 |
| Sandal | 0.84 | 0.72 | 0.77 | 501 |
| Shirt | 0.45 | 0.39 | 0.42 | 492 |
| Sneaker | 0.82 | 0.83 | 0.82 | 508 |
| Bag | 0.80 | 0.87 | 0.84 | 499 |
| Ankle Boot | 0.80 | 0.87 | 0.83 | 490 |
| avg / total | 0.73 | 0.73 | 0.73 | 5000 |

*Figure 15. Classification Report for 1 hidden layer NN*

Classification Report for Multi Layer Network

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| T-shirt/top | 0.80 | 0.86 | 0.83 | 486 |
| Trouser | 0.98 | 0.96 | 0.97 | 514 |
| Pullover | 0.77 | 0.86 | 0.81 | 496 |
| Dress | 0.87 | 0.90 | 0.88 | 517 |
| Coat | 0.80 | 0.81 | 0.80 | 497 |
| Sandal | 0.98 | 0.95 | 0.96 | 501 |
| Shirt | 0.74 | 0.55 | 0.63 | 492 |
| Sneaker | 0.96 | 0.92 | 0.94 | 508 |
| Bag | 0.97 | 0.98 | 0.97 | 499 |
| Ankle Boot | 0.92 | 0.98 | 0.95 | 490 |
| avg / total | 0.88 | 0.88 | 0.88 | 5000 |

*Figure 16. Classification Report for multi-layer NN*

Classification Report for CNN

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| T-shirt/top | 0.89 | 0.81 | 0.84 | 507 |
| Trouser | 0.99 | 0.97 | 0.98 | 508 |
| Pullover | 0.85 | 0.84 | 0.84 | 492 |
| Dress | 0.87 | 0.91 | 0.89 | 523 |
| Coat | 0.86 | 0.79 | 0.82 | 500 |
| Sandal | 0.98 | 0.97 | 0.98 | 495 |
| Shirt | 0.65 | 0.78 | 0.71 | 479 |
| Sneaker | 0.95 | 0.97 | 0.96 | 495 |
| Bag | 0.99 | 0.97 | 0.98 | 486 |
| Ankle Boot | 0.96 | 0.96 | 0.96 | 515 |
| avg / total | 0.90 | 0.90 | 0.90 | 5000 |

*Figure 17. Classification Report for CNN*

```
Summary for Multi Layer Network

Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_13 (Dense)             (None, 100)               78500
_________________________________________________________________
dense_14 (Dense)             (None, 50)                5050
_________________________________________________________________
dense_15 (Dense)             (None, 25)                1275
_________________________________________________________________
dense_16 (Dense)             (None, 10)                260
=================================================================
Total params: 85,085
Trainable params: 85,085
Non-trainable params: 0
_________________________________________________________________
```

*Figure 18. Summary for multi-layer NN*

```
Summary for CNN

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, 256)               401664
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 412,778
Trainable params: 412,778
Non-trainable params: 0
_________________________________________________________________
```

*Figure 19. Summary for CNN*

## 5 Conclusion

In this project, at first, I built a single hidden layer neural network, then using keras library I built a multi-layer neural network. At last I built a CNN. Via observations and multiple runs, I came to the conclusion that a CNN achieves more accuracy than others and that too with less epochs as compared to a multi-layer NN and a single layer NN.

The accuracy of one hidden layer network is around 73% in 30 epochs, multi-layer network is around 88% in 30 epochs and accuracy of CNN is about 90% in 3 epochs.

## References

[1] Michael Nielsen. http://neuralnetworksanddeeplearning.com/

[2] Sumit Saha. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[3] Project description PDF.