Classification of Fashion MNIST dataset

Abhishek S. Dalvi
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
adalvi@buffalo.edu

Abstract

The purpose of this project is to categorically classify fashion MNIST dataset using a neural network with one hidden layer, a neural network with multiple hidden layers made by using keras library, and a convolutional neural network made by using keras library.

1 Introduction

Several ways have been proposed to classify images like neural networks, counting number of pixels, CNN, etc. This project will compare the results obtained from few of this models i.e a neural network with one hidden layer, a neural network with multiple hidden layers and a Convolutional Neural Network.

2 Dataset

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. We intend Fashion-MNIST to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms. It shares the same image size and structure of training and testing splits.

Each training and test example is assigned to one of the following labels:

Table 1: Labels

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

3 Pre-Processing

3.1 Data Partitioning

The whole dataset is partitioned into three parts, Testing, Validation and Testing randomly in the ratio of 70%, 10% and 20% respectively. The partioning is done using sk-learns train-test-

split function.

3.2 One hot Encoding

One hot encoding is a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. it is done manually (in 1st network) and also done by using to_categorical function.

3.3 Normalization

A standard scalar normalization is applied on the training data i.e dividing each value by 256(the max value) to standardize the values between 0 to 1.

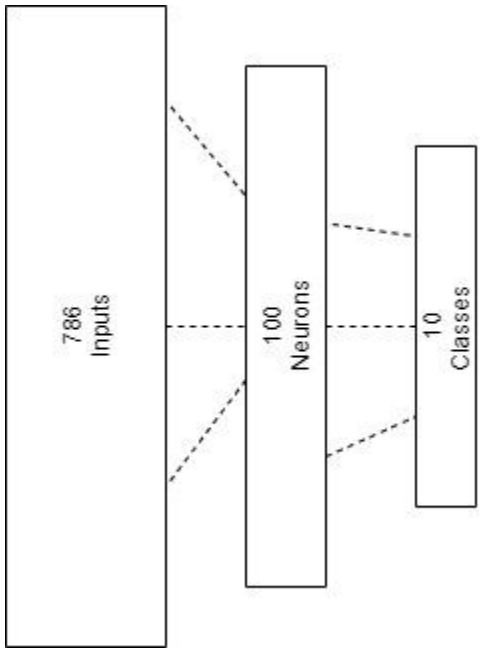
4 Architecture

4.1 Neural Network with one hidden layer

A basic neural network with one hidden layer is made for classification which has 100 neurons in the hidden layer.

Sigmoid activation is used for the first layer and soft-max is used for the second layer.

Figure 1: Neural Network architecture with one hidden layer

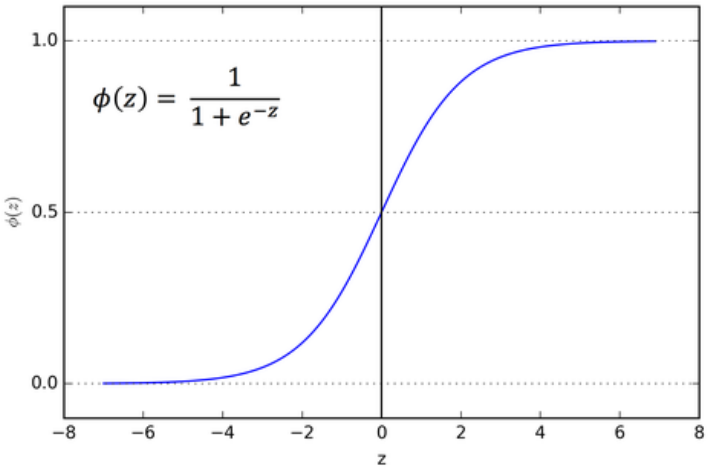


Loss function used is entropy loss which is one of the best loss function for classification. The learning rate is kept as 0.1 which was found to be ideal after several attempts.

Sigmoid activation function:

$$y = \frac{1}{1 + e^{-x}}.$$

Figure 2: Sigmoid activation function



The main reason why we use sigmoid function is because it exists between **(0 to 1).** Therefore, it is especially used for models where we have to **predict the probability** as an output. Since probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice. The function is **differentiable**, that means, we can find the slope of the sigmoid curve at any two points.

Softmax activation function:

In mathematics, the softmax function, also known as softargmax[or normalized exponential function, is a function that takes as input a vector of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K$$

To derive the loss function for the softmax function we start out from the likelihood function that a given set of parameters θθ of the model can result in prediction of the correct class of each input sample, as in the derivation for the logistic loss function. The maximization of this likelihood can be written as:

$$\underset{\theta}{\operatorname{argmax}}\, \mathcal{L}(\theta|\mathbf{t}, \mathbf{z})$$

The likelihood L(θ|t,z)L(θ|t,z) can be rewritten as the joint probability of generating tt and zz given the parameters θθ: P(t,z|θ)P(t,z|θ). Which can be written as a conditional distribution:

$$P(\mathbf{t}, \mathbf{z}|\theta) = P(\mathbf{t}|\mathbf{z}, \theta)P(\mathbf{z}|\theta)$$

Since we are not interested in the probability of zz we can reduce this to: L(θ|t,z)=P(t|z,θ)L(θ|t,z)=P(t|z,θ). Which can be written as P(t|z)P(t|z) for fixed θθ. Since each titi is dependent on the full zz, and only 1 class can be activated in the tt we can write

$$P(\mathbf{t}|\mathbf{z}) = \prod_{i=c}^{C} P(t_c|\mathbf{z})^{t_c} = \prod_{i=c}^{C} \varsigma(\mathbf{z})_c^{t_c} = \prod_{i=c}^{C} y_c^{t_c}$$

As was noted during the derivation of the loss function of the logistic function, maximizing this likelihood can also be done by minimizing the negative log-likelihood:

$$-\log \mathcal{L}(\theta|\mathbf{t}, \mathbf{z}) = \xi(\mathbf{t}, \mathbf{z}) = -\log \prod_{i=c}^{C} y_c^{t_c} = \sum_{i=c}^{C} t_c \cdot \log(y_c)$$

Which is the cross-entropy error function ξξ. Note that for a 2 class system output t2=1−t1t2=1−t1 and this results in the same error function as for logistic regression:

$$\xi(\mathbf{t}, \mathbf{y}) = -t_c \log(y_c) - (1 - t_c) \log(1 - y_c).$$

The cross-entropy error function over a batch of multiple samples of size nn can be calculated as:

$$\xi(T, Y) = \sum_{i=1}^{n} \xi(\mathbf{t}_i, \mathbf{y}_i) = -\sum_{i=1}^{n} \sum_{i=c}^{C} t_{ic} \cdot \log(y_{ic})$$

Derivative of the cross-entropy loss function for the softmax function

$$\begin{aligned}
\frac{\partial \xi}{\partial z_i} &= -\sum_{j=1}^{C} \frac{\partial t_j \log(y_j)}{\partial z_i} = -\sum_{j=1}^{C} t_j \frac{\partial \log(y_j)}{\partial z_i} = -\sum_{j=1}^{C} t_j \frac{1}{y_j} \frac{\partial y_j}{\partial z_i} \\
&= -\frac{t_i}{y_i} \frac{\partial y_i}{\partial z_i} - \sum_{j \neq i}^{C} \frac{t_j}{y_j} \frac{\partial y_j}{\partial z_i} = -\frac{t_i}{y_i} y_i(1 - y_i) - \sum_{j \neq i}^{C} \frac{t_j}{y_j}(-y_j y_i) \\
&= -t_i + t_i y_i + \sum_{j \neq i}^{C} t_j y_i = -t_i + \sum_{j=1}^{C} t_j y_i = -t_i + y_i \sum_{j=1}^{C} t_j \\
&= y_i - t_i
\end{aligned}$$

4.2 Neural Network with multiple hidden layers

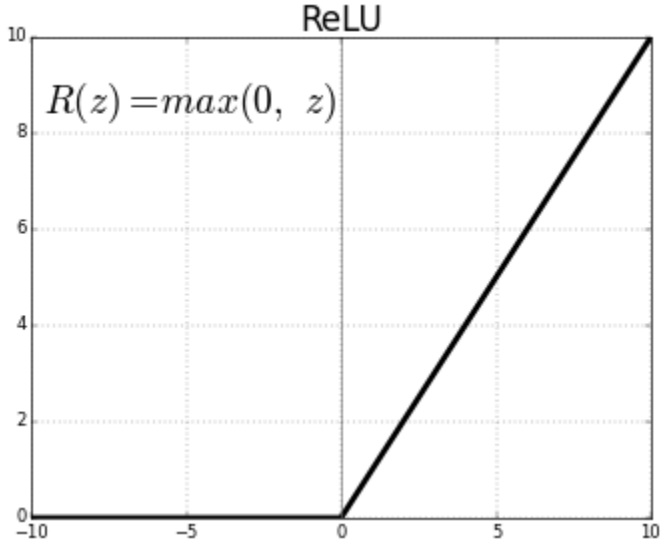
A Neural Network with multiple hidden layers is made by using Keras library which is built on tensorflow. The activation function used in this arctiruec are Relu and softmax. Relu is used instead of sigmoid as there are several limitations to the sigmoid activation.

One such important limitation is the **vanishing gradient problem** in deep neural networks. While backpropagating in neural netwrok we see that the derivative of sigmoid fucntion is needed. We see that the first derivative itself is a really small value, as sigmoid is a squashing function. This derivative is used for calculating the gradient, which is further used for updating the weights as we back propagate. At one point, the gradient itself becomes so small that the weight updation is negligible. Thus, in really deep neural network we observe that the weights in the starting layers are not updated. This is the vanishing gradient problem.

We also see that the sigmoid function is not sensitive to the values, i.e a very large value and a relatively large value will have almost the same values, which shouldn't be the case as the very large value should receive more activation.

The solution is to use the rectified linear activation function, or ReLU for short.

Figure 3: ReLu activation function

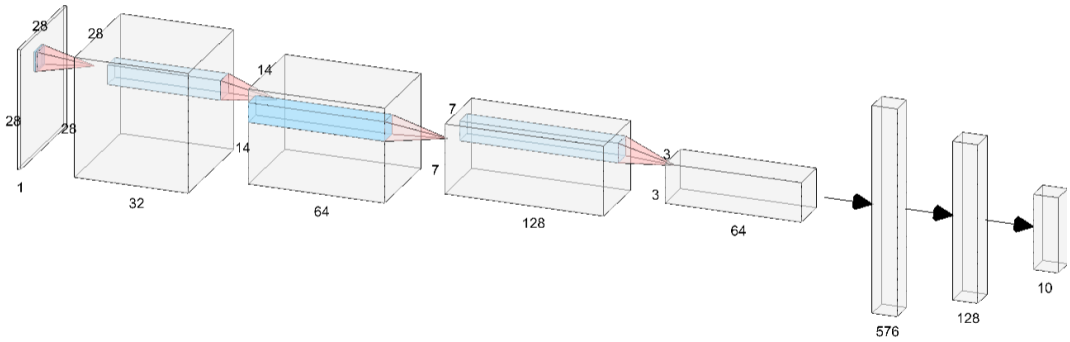


4.3 Convolutional Neural Network

The Convolutional Neural Network is using 2-D convolutional and in each layer, ReLu activation is used. No max-pooling is used, instead a stride of 2 is applied to the filter, hence decreasing the dimensions of the layer which also decreases the number of parameter. Using strides of two can be a good alternative as max pooling leads to loss of data. The CNN is flattened at a point in the network and then this flattened vector is passed to a fully connected neural network for classification where in the end it goes through softmax activation for classification.

The diagram below represents the CNN used in the project with the dimensions of the layers.

Figure 4: CNN architecture used in project



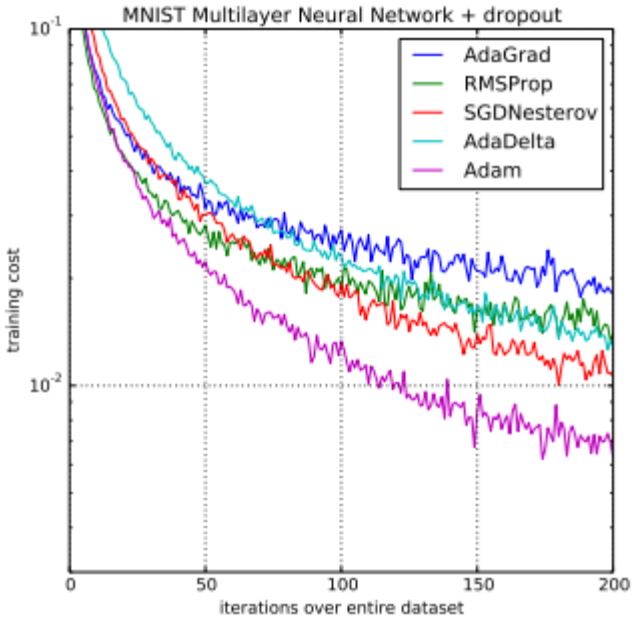
We see that when we apply a padding (using keras padding=”same”) and a stride of two, the diemnsions are reduced to half. This can also be calculated using the below formula

$$D_2=(D_1-F+2P)/S+1$$

Adam Optimizer is used in both CNN and neural network with multiple hidden layers. The Adam optimization algorithm is an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision and natural language processing.

The method computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients.

Figure 5: Adam optimizer comparison graph



The graph is taken from “Comparison of Adam to Other Optimization Algorithms Training a Multilayer Perceptron Taken from Adam: A Method for Stochastic Optimization, 2015”. We see that Adam Optimizer produces the best result among the other optimizers.
Hence we use the Adam optimizer for training the dataset.

5 Results

5.1 Neural Network with one hidden layer

The network is trained with learning rate of 0.1 over 100 epochs with a normal gradient descent.

Figure 6: Training vs validation loss graph for NN with single hidden layer

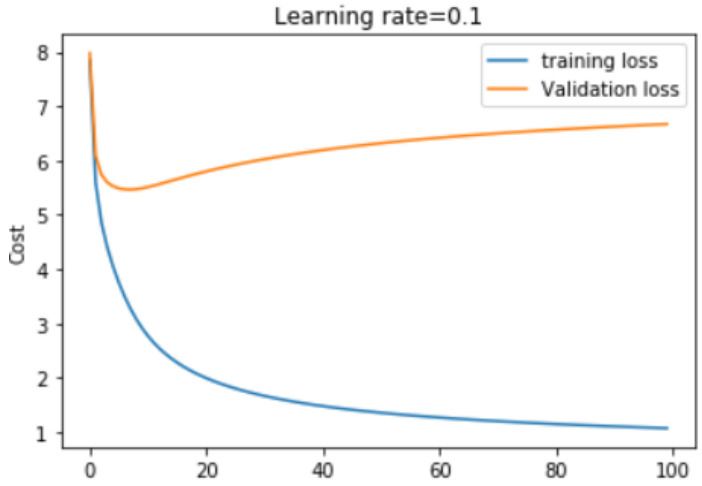


Figure 7: Confusion matrix for NN with single hidden layer

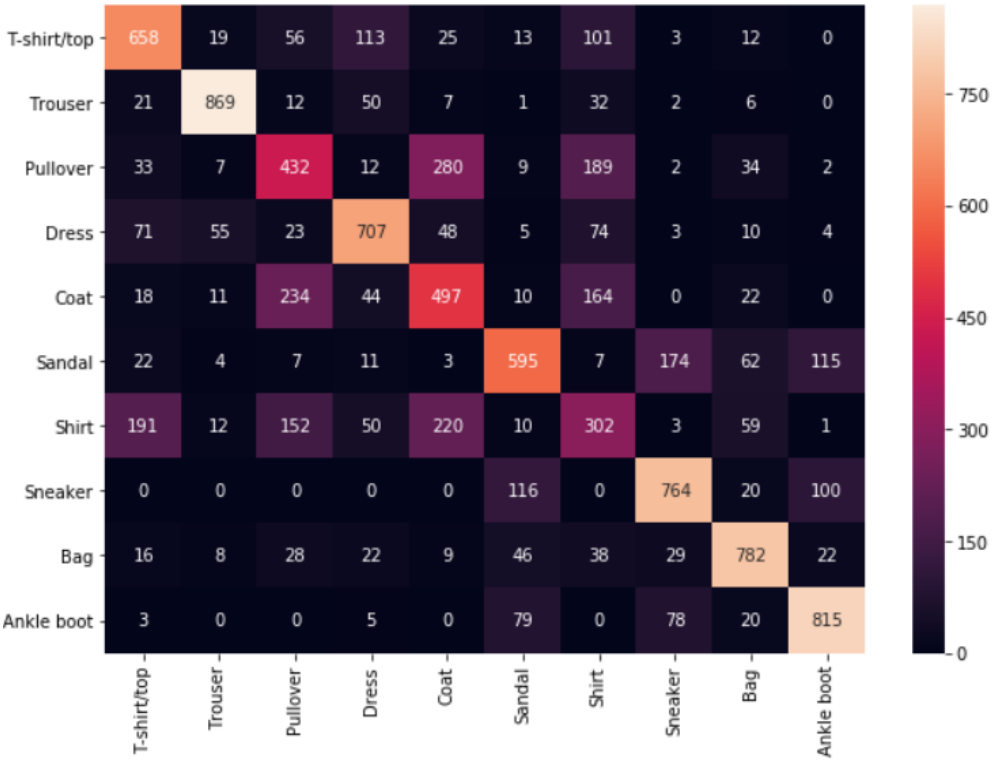
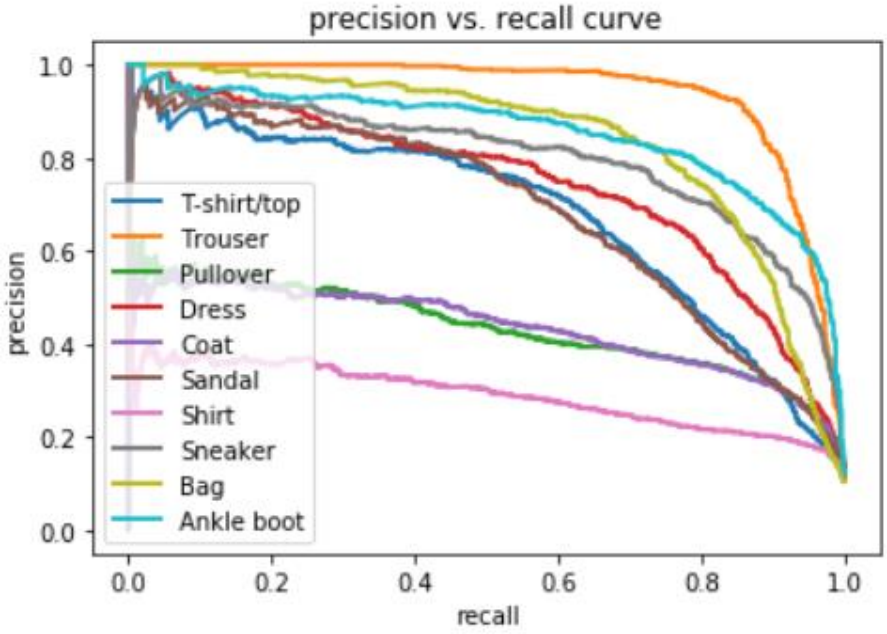


Figure 8: Precision vs Recall curve for NN with single hidden layer



From the confusion matrix and the precision and recall graph we see that the network has a difficulty classifying most of the classes(trousers classification is better).

ACCUARACY= 64.21 %

The accuracy obtained is very low, a decent model will have accuracy atleast of 80%, hence we try to built a better model.

5.2 Neural Network with multiple hidden layers

Figure 9: Train vs Validation loss for NN with multiple hidden layers

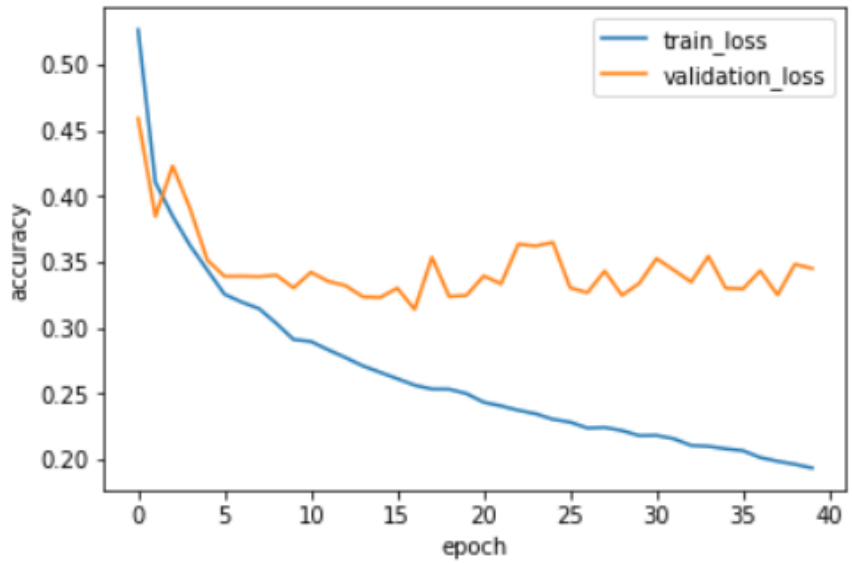


Figure 10: Confusion matrix for NN with multiple hidden layers

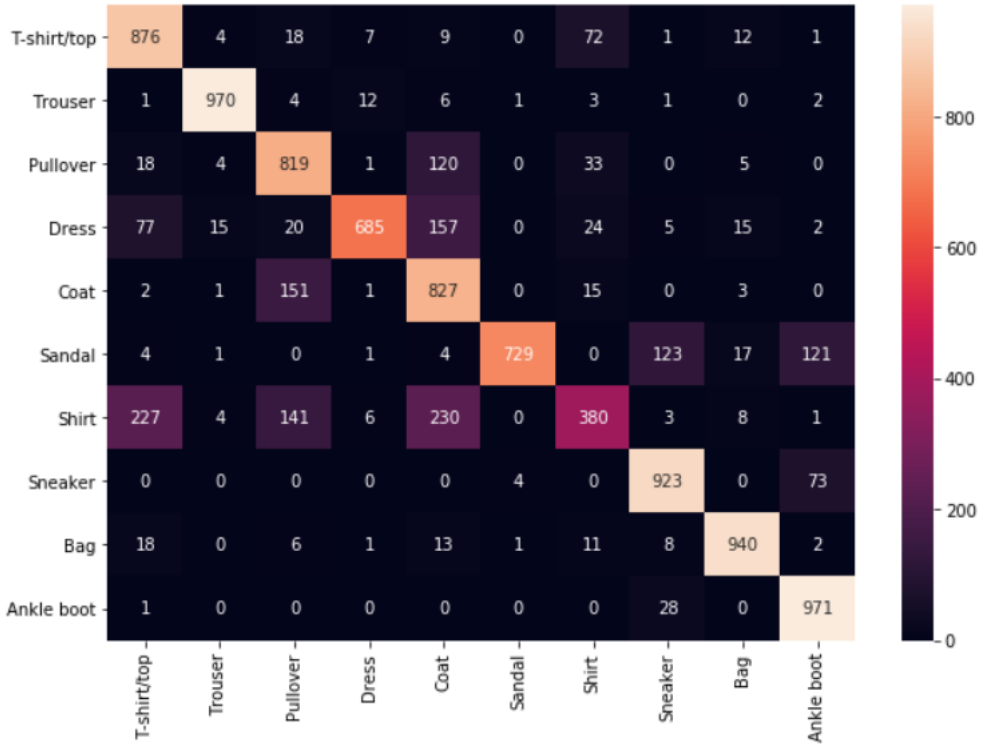
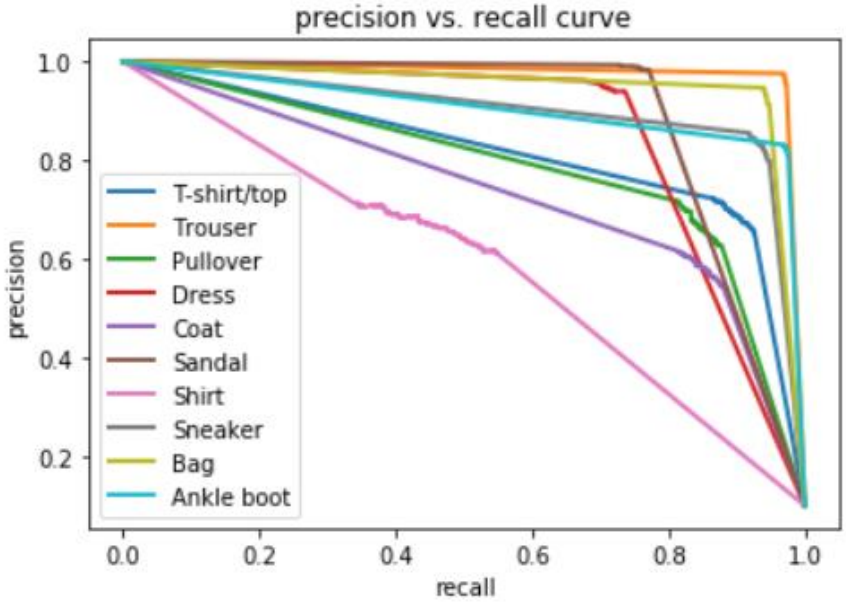


Figure 11: Precision vs Recall curve for NN with multiple hidden layers



ACCUARACY= 81.2 %

We get a better accuracy than the previous model but it can be improved using CNN.

5.1 Convolutional Neural Networks

CNN is trained over 25 epochs having a batch size of 10.

Figure 12: Train vs validation loss for CNN

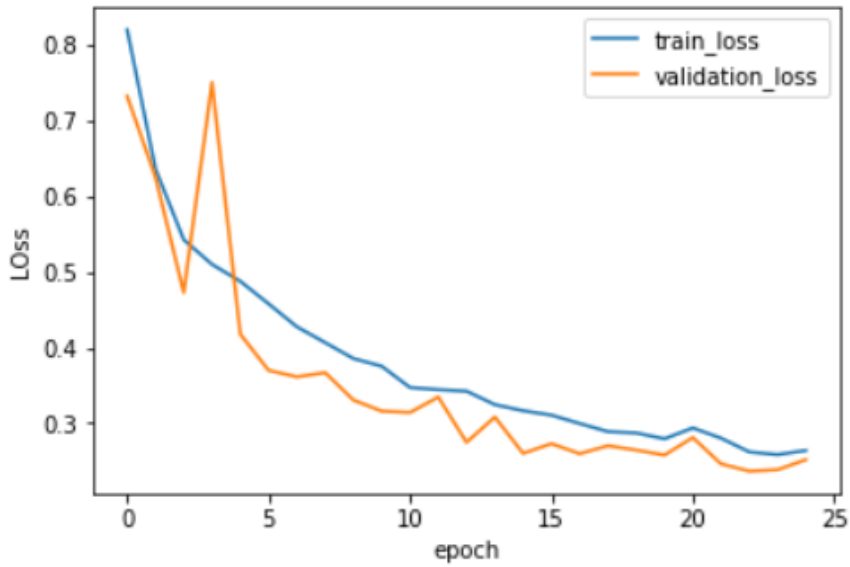


Figure 13: Confusion matrix for CNN

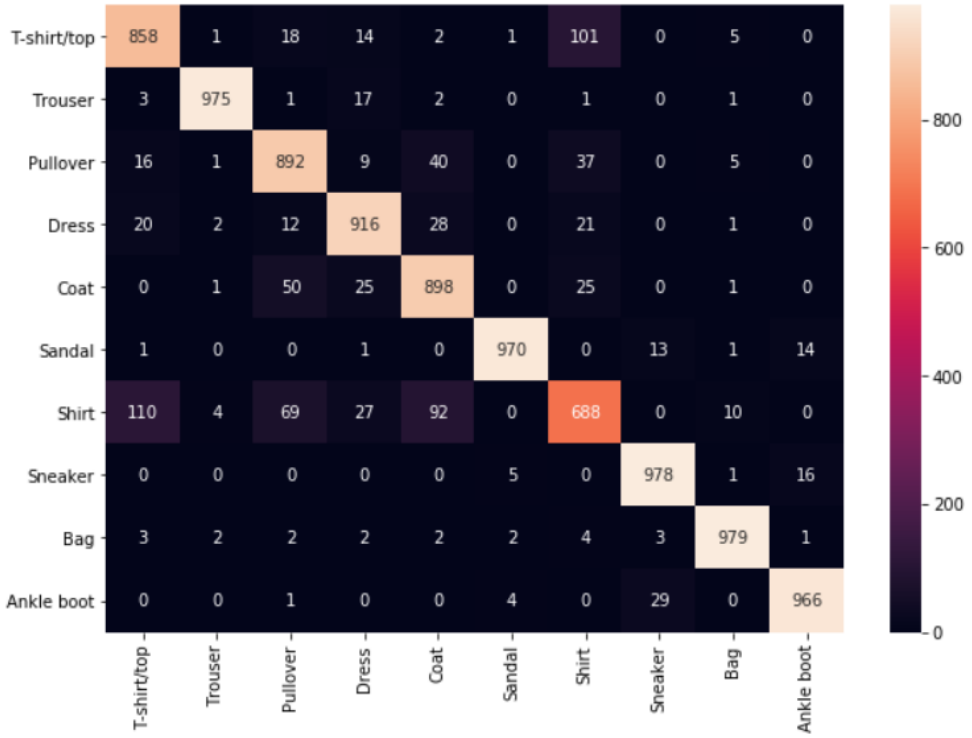
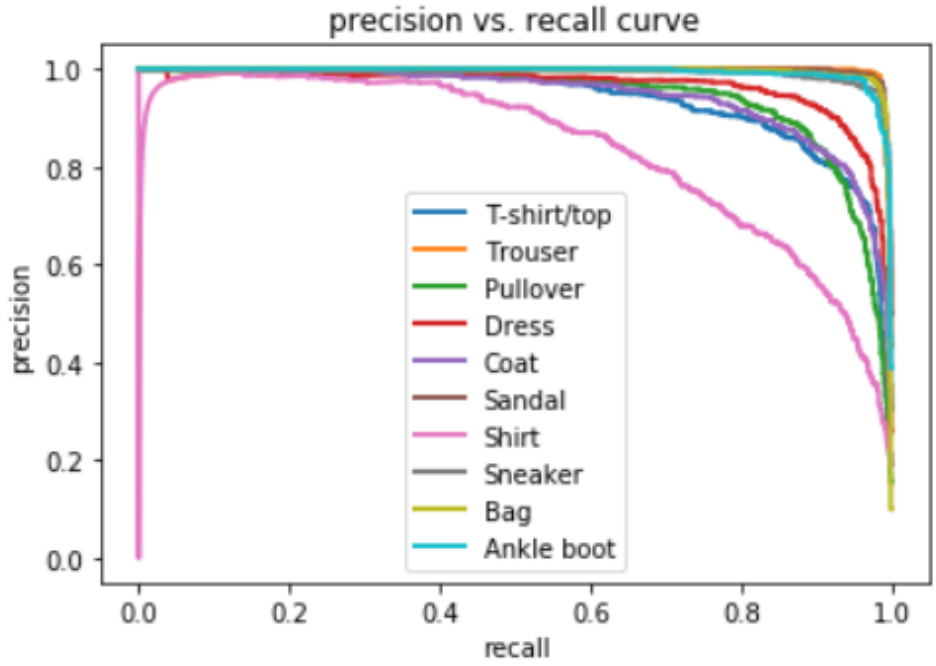


Figure 14: Precision vs Recall curve for CNN



ACCUARACY= 91.2 %

We observe a better accuracy than the previous two models for CNN.

6 Conclusion

It can be concluded that CNNs definitely perform better than a fully connected neural network which is seen from the accuracies. We observe from the precision recall graphs that shirts are the hardest to classify and they are best classified by CNNs (which can be seen in the confusion matrix). Trousers are relatively easy to classify hence they are classified corrected by a neural network even with one hidden layer. In conclusion we can say that Convolutional Neural networks works better to neural networks.