Implementation of Neural Network and Convolutional Neural Network using Fashion-MNIST clothing images

Ankita Das
adas4@buffalo.edu

Abstract

This paper presents the development and the simulation of a machine learning model to implement neural network and convolutional neural network for the task of classification. For obtaining a reasonable model, here we are using the Fashion-MNIST dataset of Zalando's article images, consisting of a training set of 60,000 examples and a test set of 10,000 examples. The first part of this work has been dedicated to pre-process the data to optimize the classifier. Next, we need to examine the dataset what it contains and for the implementation of the model, the train dataset has been partitioned in the following fashion: 80% for training phase, and 20% for the validation phase. Here the implementation of the model has been divided into three parts: building a Neural Network with one hidden layer, building multi-layer Neural Network, and building Convolutional Neural Network (CNN). The hyper-parameter tuning has been done to estimate the best model parameter in each case and the statistics is as follows: for Single Layer Neural Network it's 69% of accuracy, for Multi-Layer Neural Network it's 85.80% accuracy and Convolutional Neural Network gives an accuracy of 85.92%

1. Introduction

Images of clothes are great factors in driving the clothing industry today and trends found in the current fashion styles drive the online clothing economy. This study attempts to create a clothing classification tool with the use of neural networks.

This paper presents three different approaches, namely Neural Network with one hidden layer, multi-layer Neural Network, Convolutional Neural Network (CNN) for the task of classification. For training and testing of our classifiers, we will use the Fashion-MNIST dataset

2. Dataset Definition

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. 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.

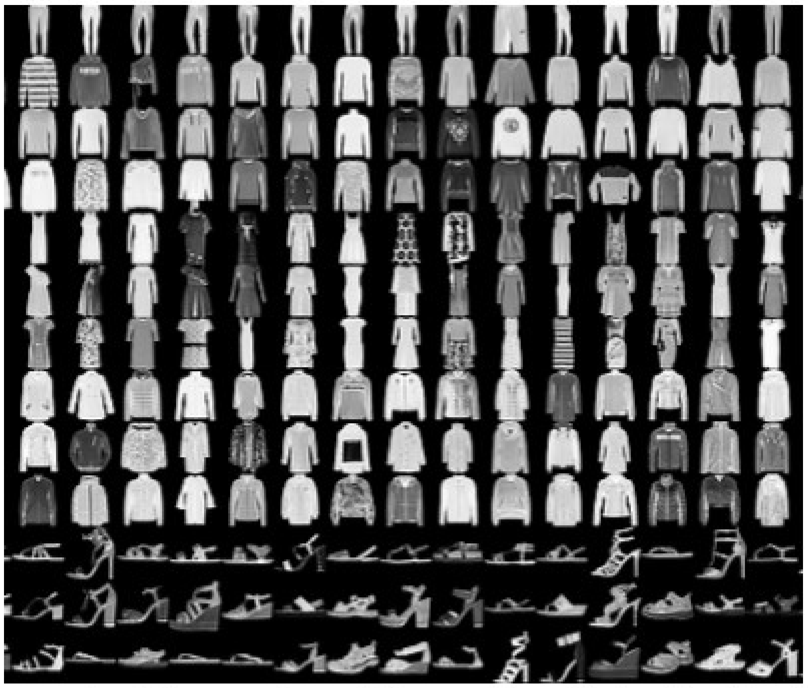


Figure 1: Example of how the data looks like.

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 are 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

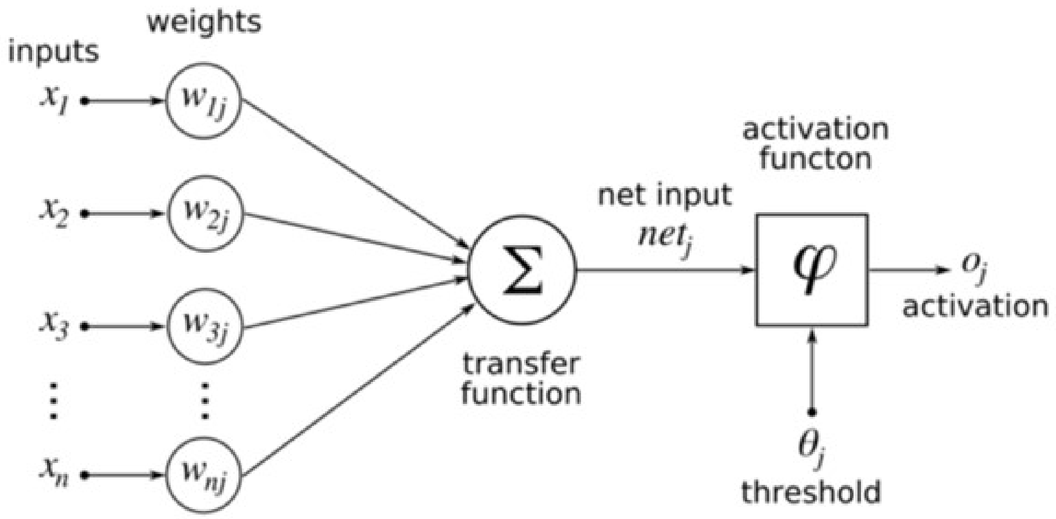
3. Pre-Processing

We are using Jupyter notebook to work on this dataset. We will first go with importing the necessary libraries and import our dataset to Jupyter notebook. We can visualize the data set by fetching randomly 4 examples from training data. Visualization of data is an imperative aspect of data science. It helps to understand data and also to explain the data to another person.

We do have dataset partitioned into training and test already. We shall partition the training dataset into 80%, 20% fashion to have validation set. The training set contains a known output and the model learns on this data to be generalized to other data later on. We have the test dataset to test our model's prediction on this subset and the introduction of the validation dataset allows us to evaluate the model on different data than it was trained on and select the best model architecture, while still holding out a subset of the data for the final evaluation at the end of our model development.

4. Model Architecture

- **Single hidden layer neural network**: A single hidden layer neural network consists of 3 layers: input, hidden and output. The input layer has all the values form the input, in the hidden layer, where most of the calculations happens, every Perceptron unit takes an input from the input layer, multiplies and add it to initially random values.



The last and third layer is the output layer, it takes all the previous layer Perceptrons as input and multiplies and add their outputs to initially random values. then gets activated by a Sigmoid function.

To train the network we rely on gradient descent and backpropagation of the gradients. Here, the output values are compared with the correct answer to compute the value of some predefined error-function. The error compared to the expected final output is then fed back through the network. Using this information, the algorithm adjusts the initially random weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has learned a certain target function. To adjust weights properly, one applies a general method for non-linear optimization that is called gradient descent. For this, the network calculates the derivative of the error function with respect to the network weights and changes the weights such that the error decreases (thus going downhill on the surface of the error function). For this reason, back-propagation can only be applied on networks with differentiable activation functions.

Mathematically, neural network has parameters $(W,b)=(W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]})$, where we write W to denote the parameter (or weight) and b is the bias. We can write the activation function as follows:

$Z^{[1]} = W^{[1]}x + b^{[1]}$

Activation function for the hidden layer:

$a^{[1]} = \sigma(Z^{[1]})$

$Z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$

Activation function for the output layer:

$a^{[2]} = \text{softmax}(Z^{[2]})$

Cost Function:

$L(a^{[2]},y) = -\sum y \log a^{[2]}$

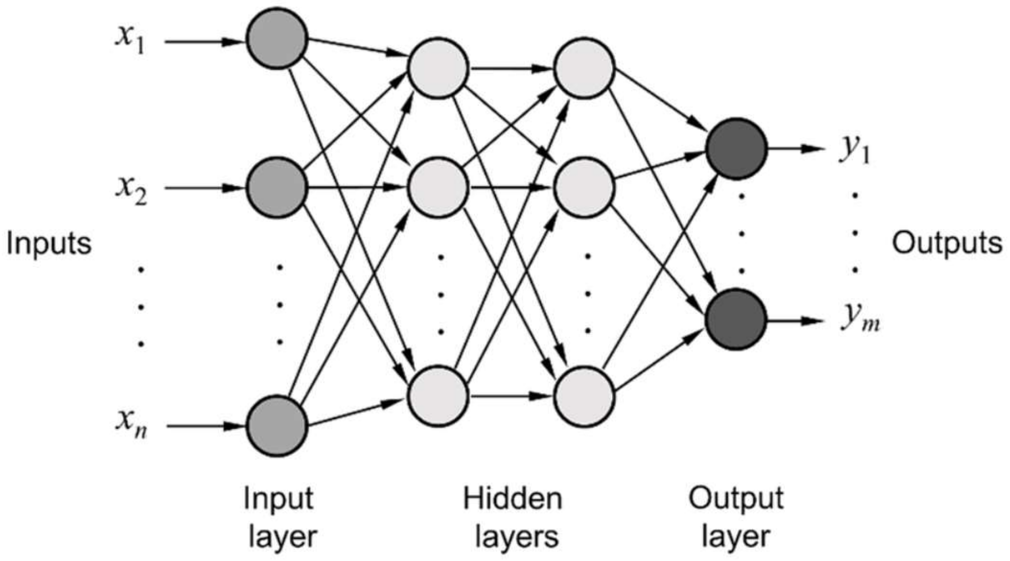
Derivatives of weights:

$\Delta w^{[1]} = (a^{[2]} - y)\, a^{[1]}$

$\Delta w^{[1]} = (a^{[2]} - y)\, w^{[2]}\, a^{[1]}(1-a^{[1]})x$

In single layer neural network we have considered only one hyperparameter, i.e number of hidden nodes and we are tuning this model using the same in 4 stages using 10, 20, 30 and 40 hidden nodes.

- **Multi-Layer Neural Network:** A multi-layer neural network contains more than one layer of artificial neurons or nodes. They differ widely in design. It is important to note that while single-layer neural networks were useful early in the evolution of AI, the vast majority of networks used today have a multi-layer model.



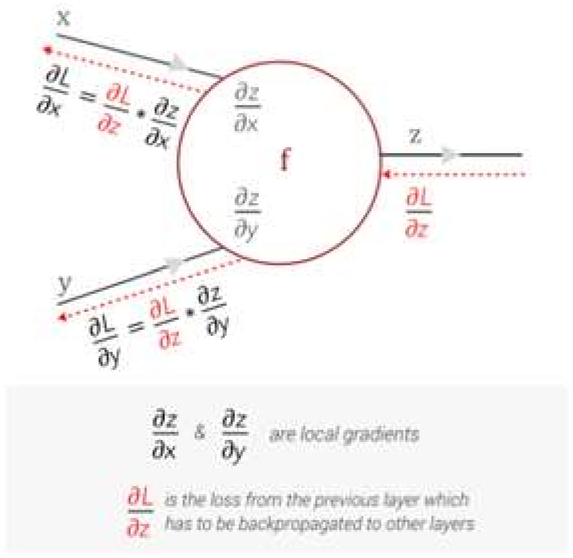
We will be using a Sequential model which is a linear stack of layers. It can be first initialized and then we add layers using add method.

- **Convolutional Neural Network (CNN):** A Convolutional Neural Network (ConvNet/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. CNN is useful for the following reason:

- Sharing of weights ends up reducing the overall number of trainable weights hence introducing sparsity

- After several convolutional and pooling layers, the image size (feature map size) is reduced and more complex features are extracted.

The usage of CNNs are motivated by the fact that they can capture / are able to learn relevant features from an image at different levels similar to a human brain.


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial y}$
$\frac{\partial z}{\partial x}$ & $\frac{\partial z}{\partial y}$ are local gradients
$\frac{\partial L}{\partial z}$ is the loss from the previous layer which has to be backpropagated to other layers

Computational Graph – Convolution Operation

We will be using a Sequential model which is a linear stack of layers. It can be first initialized and then we add layers using add method.

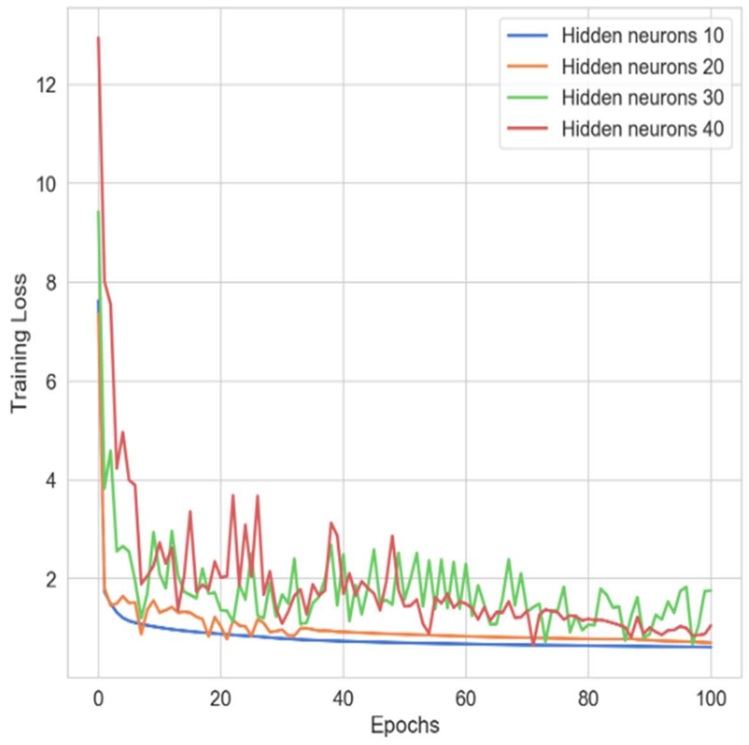
5. Results:

The results from this implementation is as follows:

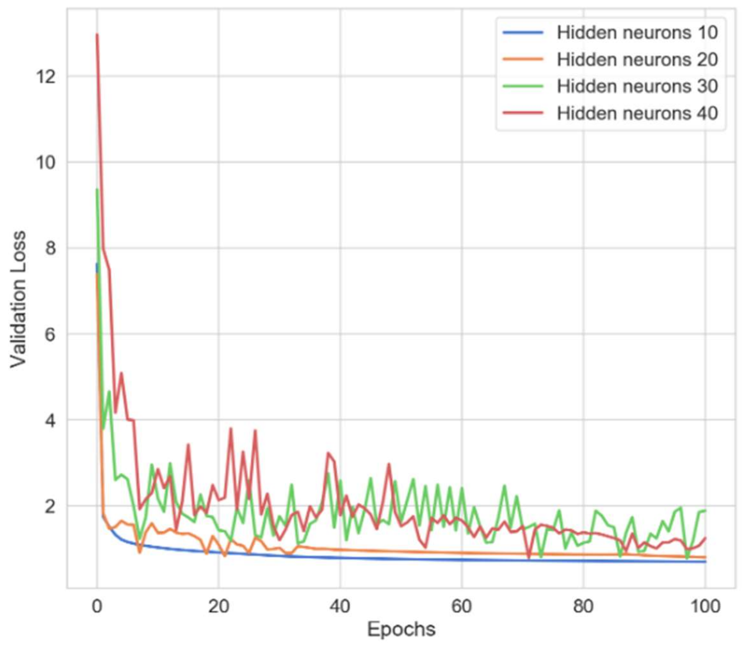
- **Single Layer Neural Network:** Here we have used tanh as the primary activation function and in the final output layer we use softmax activation function. Along with this We are using vanilla gradient descent as our optimizer and categorical cross entropy as the loss function.

Hyperparameter Tuning: We have considered one hyperparameter as number of hidden nodes values as 10, 20, 30, 40. Apart from these we have taken learning rate and number of epochs as hyperparameter also. In this case, our learning rate is set to 1e-4 and epochs set to 1000. To have the best estimated model we are updating the values of w1, b1, w2, b2 with respect to training and validation dataset.

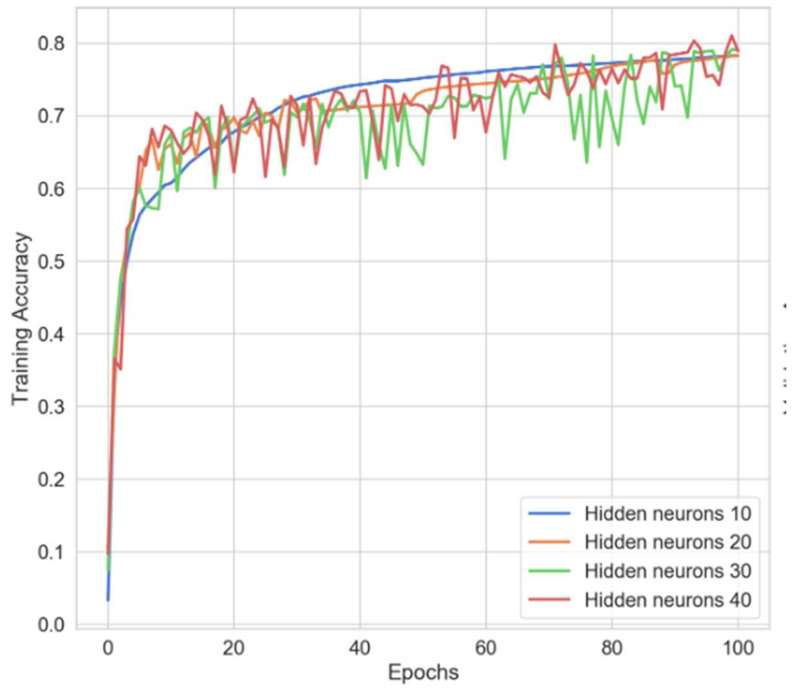
Recorded training loss with different hidden nodes is as follows:



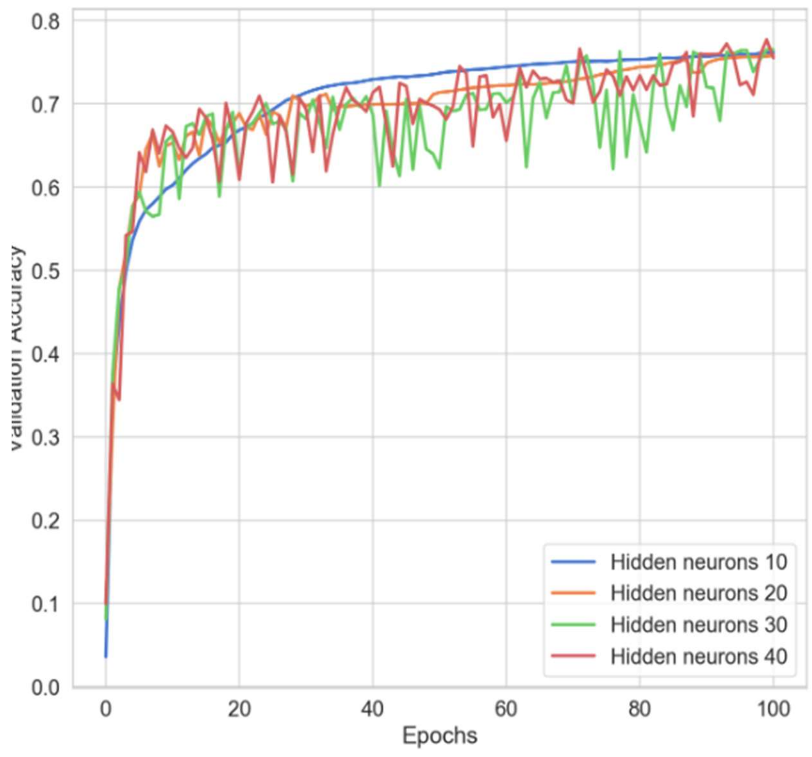
Recorded Validation loss with different hidden nodes is as follows:



Recorded Training Accuracy with hidden nodes is as follows:



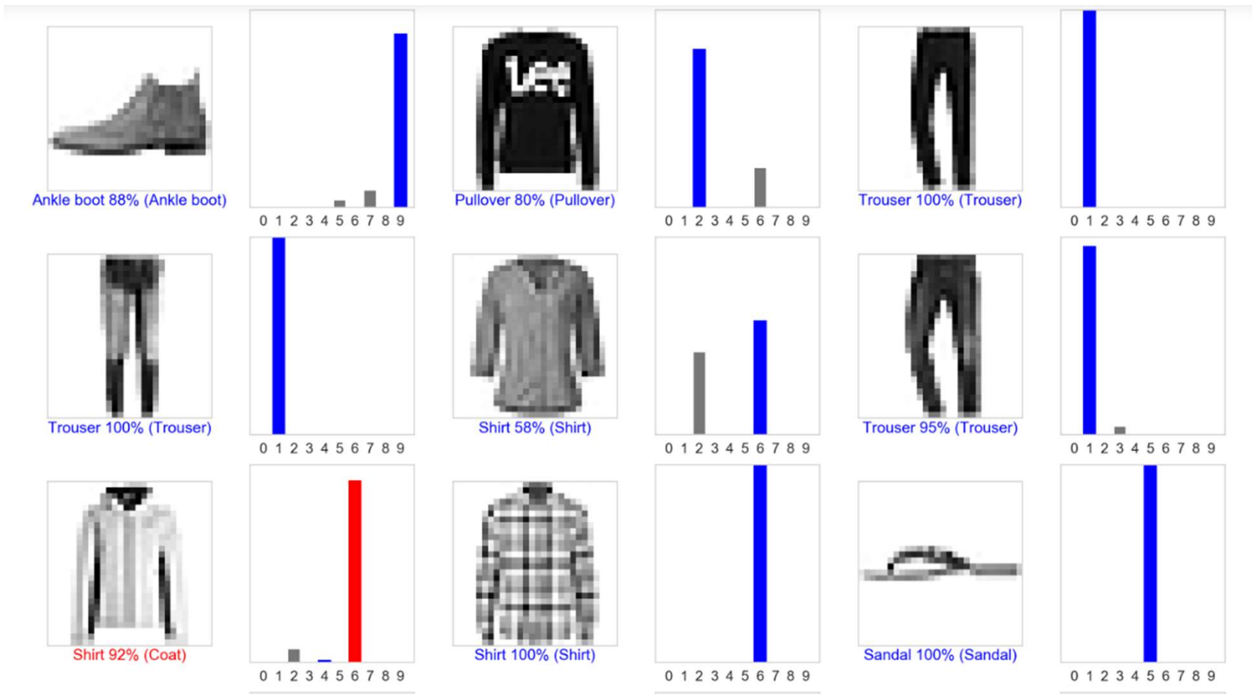
Recorded Validation Accuracy with different nodes is as follows:



We observed from the experiment that with the increase of the hidden nodes, performance is also increased. Hence, we are considering number of hidden nodes as 40 and its associated w1, b1, w2, b2 parameters which gives us an accuracy of 69%. The accuracy measure is as follows:

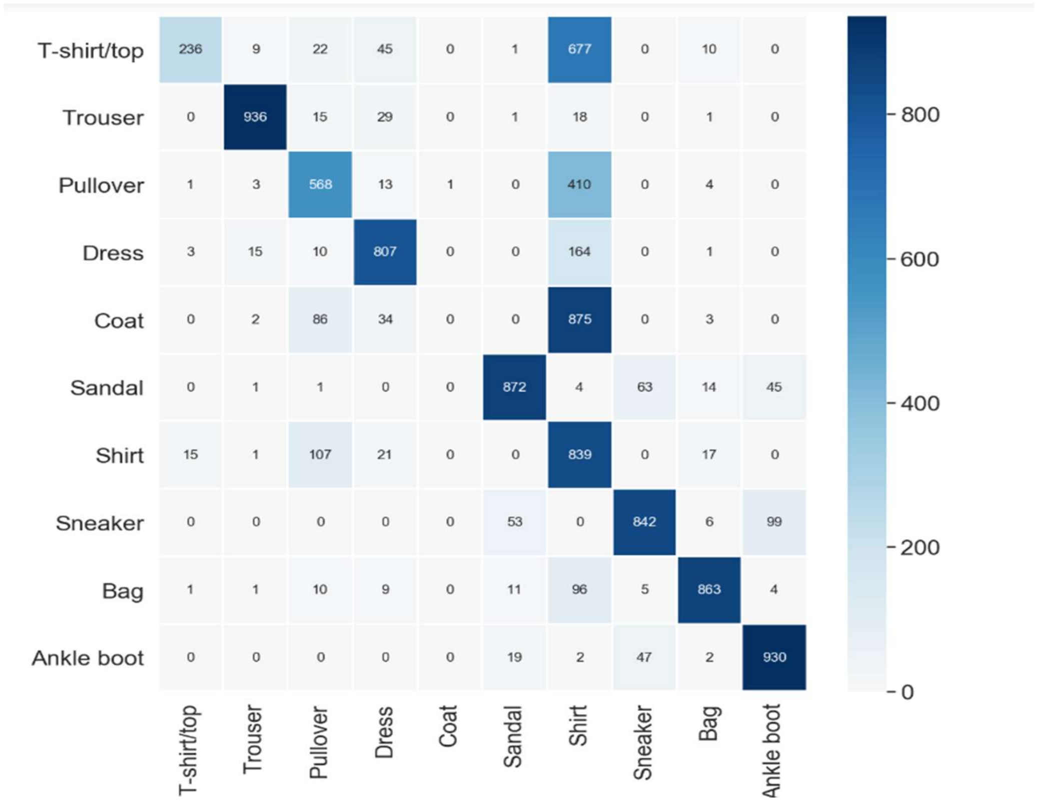
Accuracy	
Training Data	78.89%
Validation Data	75.55%
Test Data	69.00%

We plot the sample test images and check the full set of 10 class predictions as below:



Confusion Matrix: To examine the accuracy of test dataset, here we have introduced the concept of confusion matrix method of metrics class. Confusion Matrix is a performance measurement for machine learning classification problem where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

In our case the confusion matrix is as follows:



	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	236	9	22	45	0	1	677	0	10	0
Trouser	0	936	15	29	0	1	18	0	1	0
Pullover	1	3	568	13	1	0	410	0	4	0
Dress	3	15	10	807	0	0	164	0	1	0
Coat	0	2	86	34	0	0	875	0	3	0
Sandal	0	1	1	0	0	872	4	63	14	45
Shirt	15	1	107	21	0	0	839	0	17	0
Sneaker	0	0	0	0	0	53	0	842	6	99
Bag	1	1	10	9	0	11	96	5	863	4
Ankle boot	0	0	0	0	0	19	2	47	2	930

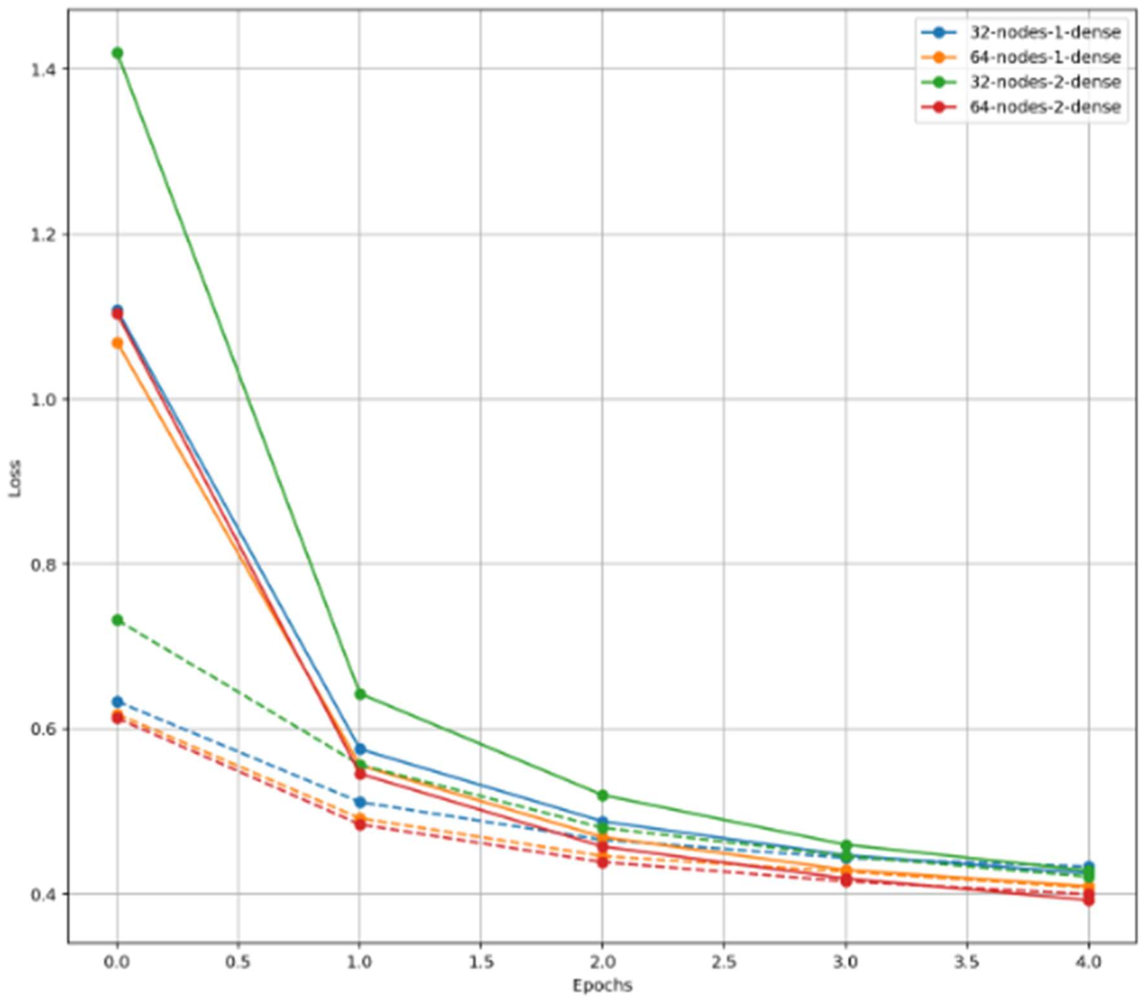
We will use the Confusion Matrix to calculate the value of Accuracy, Precision, Recall and F1- score.

```
              precision    recall  f1-score   support

           0       0.92      0.24      0.38      1000
           1       0.97      0.94      0.95      1000
           2       0.69      0.57      0.62      1000
           3       0.84      0.81      0.82      1000
           4       0.00      0.00      0.00      1000
           5       0.91      0.87      0.89      1000
           6       0.27      0.84      0.41      1000
           7       0.88      0.84      0.86      1000
           8       0.94      0.86      0.90      1000
           9       0.86      0.93      0.90      1000

    accuracy                           0.69     10000
   macro avg       0.73      0.69      0.67     10000
weighted avg       0.73      0.69      0.67     10000
```

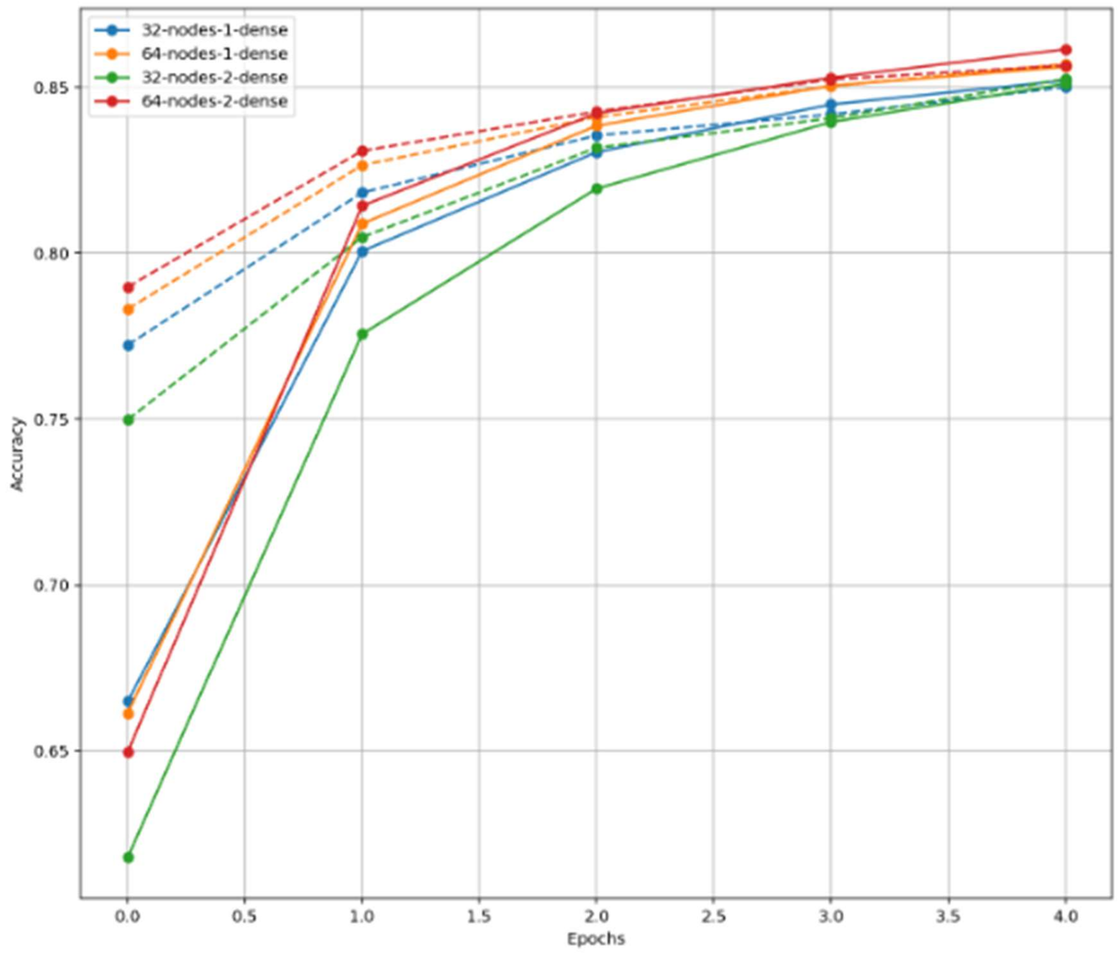
- **Multi - Layer Neural Network:** Here we have used Relu as the primary activation function in every hidden layer and in the final output layer we used softmax activation function. We have considered adam as our optimizer and categorical cross entropy as the loss function.

Hyperparameter Tuning: Here we have considered two hyperparameter, the number of dense layers (1-layer, 2-layer) and the number of hidden nodes in each layer (32 and 64). Apart from these we have also considered three hyper-parameters - learning rate, batch size and number of epochs.

Recorded Training and Validation loss with all the hyperparameters is as follows:



Recorded Training and Validation Accuracy with all the hyperparameters is as follows:

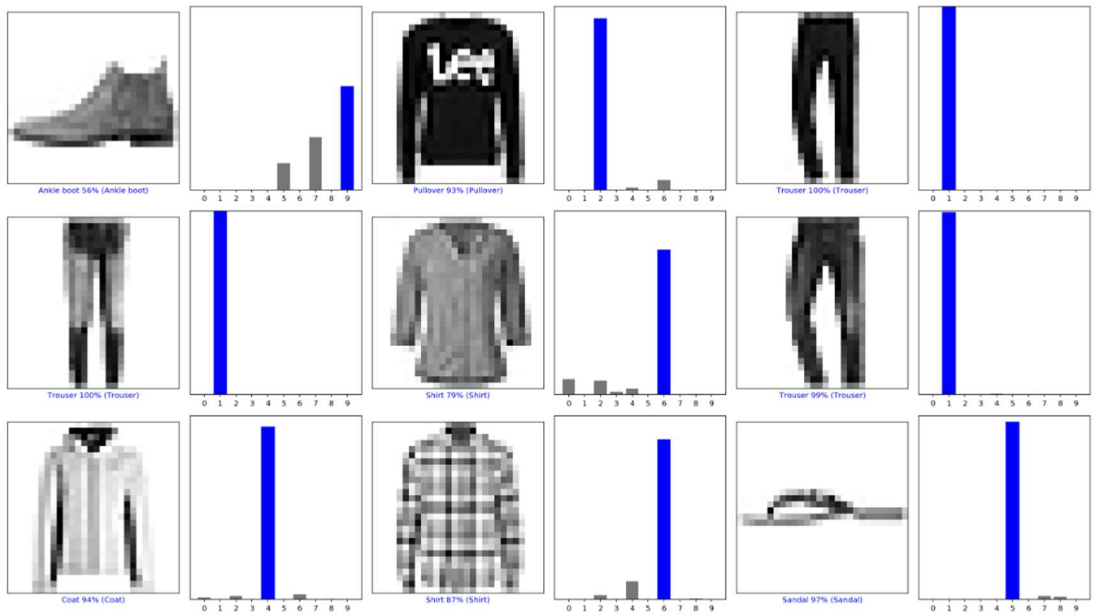


Now we estimate that the best model parameters which we get for 64 nodes in 2 dense layers and we predict the test data for this parameter to get an **accuracy of 84.80%**

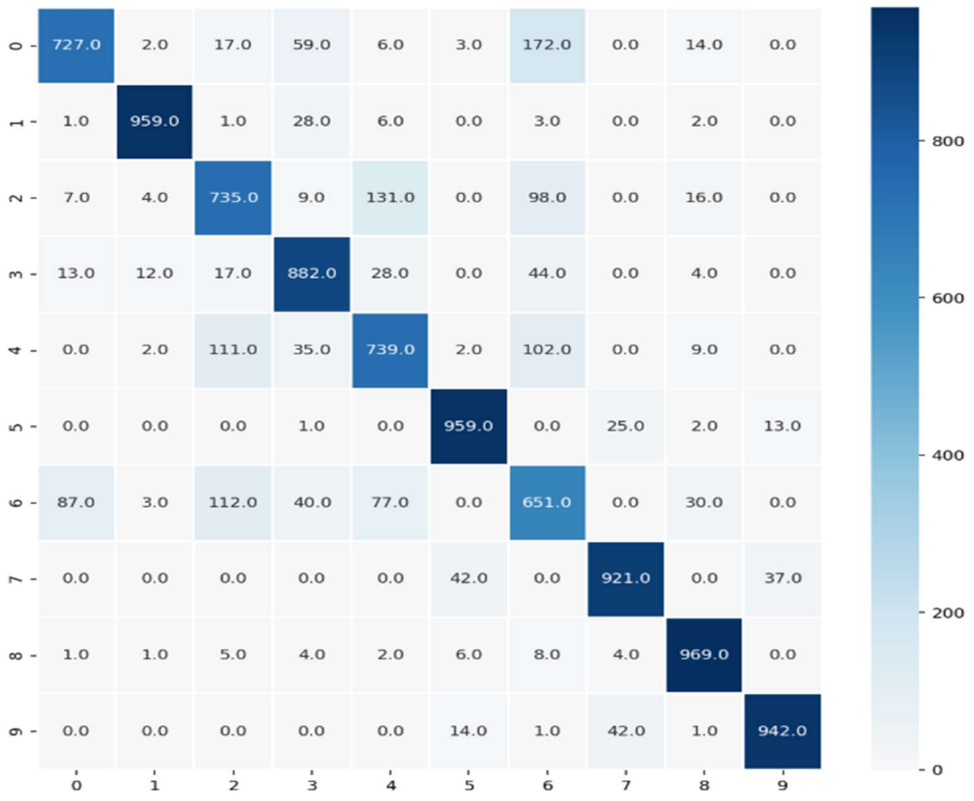
The overall performance is as follows:

Accuracy	
Training Data	86.00%
Validation Data	85.00%
Test Data	85.80%

We plot the sample test images and check the full set of 10 class predictions as below:

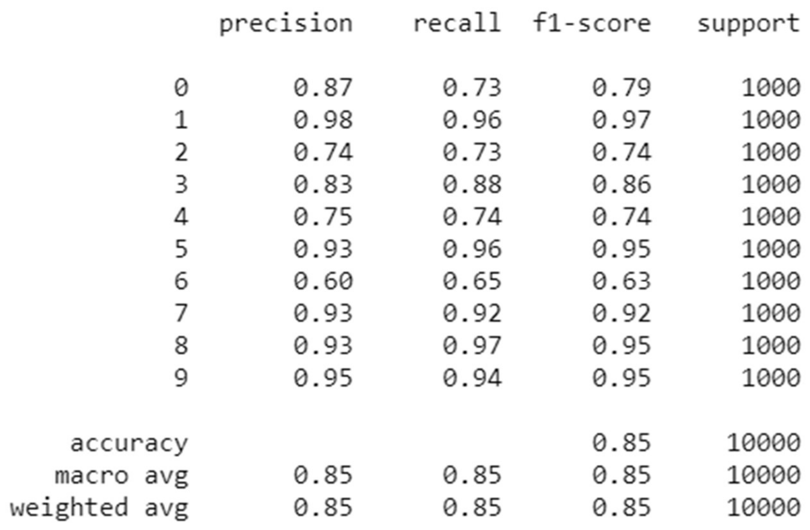


The confusion matrix in our case is as follows:



	0	1	2	3	4	5	6	7	8	9
0	727.0	2.0	17.0	59.0	6.0	3.0	172.0	0.0	14.0	0.0
1	1.0	959.0	1.0	28.0	6.0	0.0	3.0	0.0	2.0	0.0
2	7.0	4.0	735.0	9.0	131.0	0.0	98.0	0.0	16.0	0.0
3	13.0	12.0	17.0	882.0	28.0	0.0	44.0	0.0	4.0	0.0
4	0.0	2.0	111.0	35.0	739.0	2.0	102.0	0.0	9.0	0.0
5	0.0	0.0	0.0	1.0	0.0	959.0	0.0	25.0	2.0	13.0
6	87.0	3.0	112.0	40.0	77.0	0.0	651.0	0.0	30.0	0.0
7	0.0	0.0	0.0	0.0	0.0	42.0	0.0	921.0	0.0	37.0
8	1.0	1.0	5.0	4.0	2.0	6.0	8.0	4.0	969.0	0.0
9	0.0	0.0	0.0	0.0	0.0	14.0	1.0	42.0	1.0	942.0

Now we will use the Confusion Matrix to calculate the value of Accuracy, Precision, Recall and F1 score.



```
              precision    recall  f1-score   support

           0       0.87      0.73      0.79      1000
           1       0.98      0.96      0.97      1000
           2       0.74      0.73      0.74      1000
           3       0.83      0.88      0.86      1000
           4       0.75      0.74      0.74      1000
           5       0.93      0.96      0.95      1000
           6       0.60      0.65      0.63      1000
           7       0.93      0.92      0.92      1000
           8       0.93      0.97      0.95      1000
           9       0.95      0.94      0.95      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```

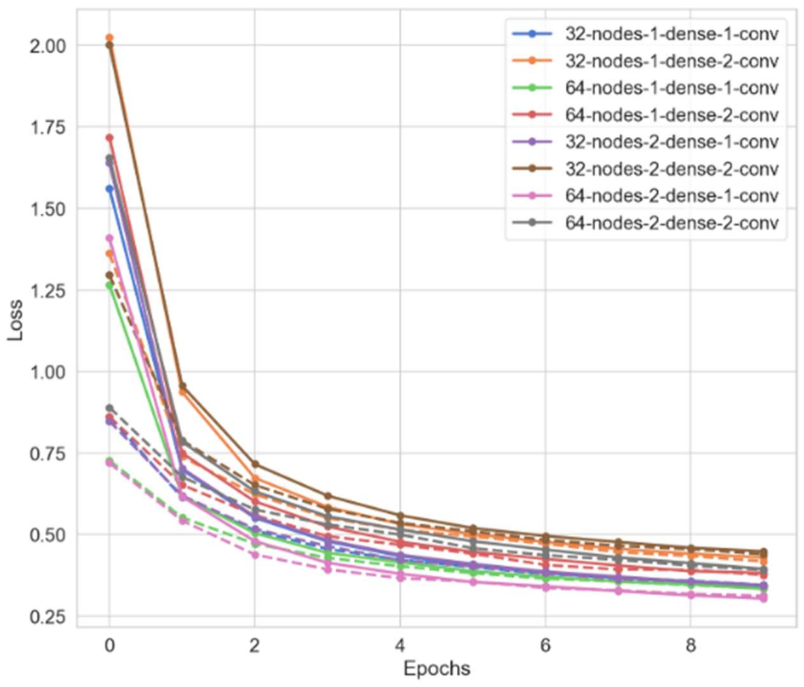
- **Convolutional Neural Network:** Here we have considered kernel size as (3,3) and max pooling layer divides the whole kernels height and width by 2 i.e. (2, 2).

We have used relu as the primary activation function in every hidden layer and in the final output layer we use softmax activation function. We have used adam as the optimizer and categorical cross entropy as the loss function and we evaluate the model on the accuracy metric.

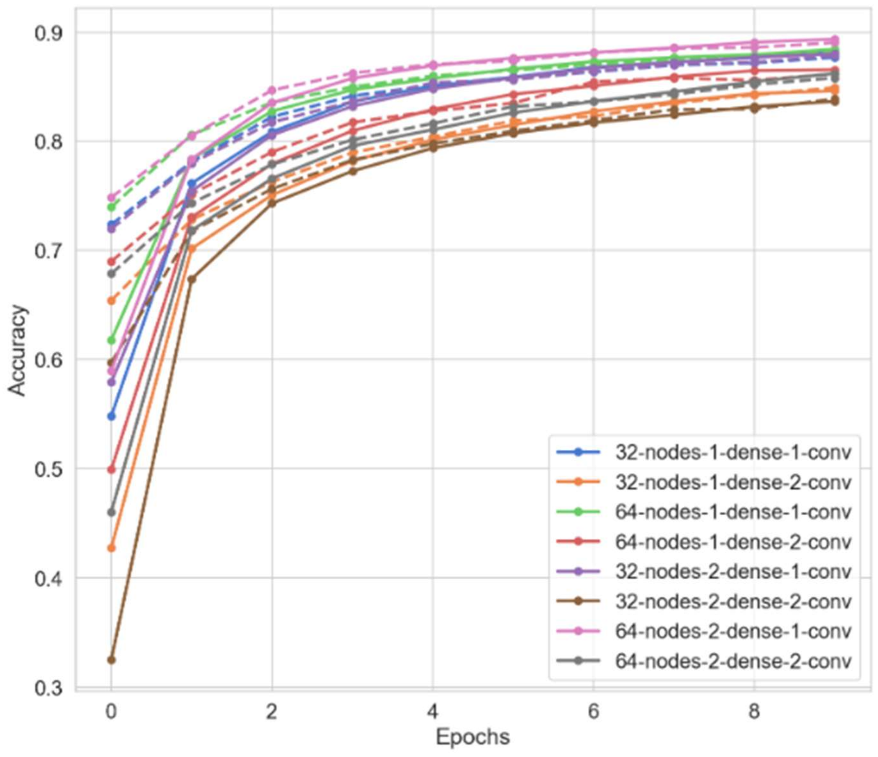
Hyperparameter Tuning: Three hyperparameters has been taken here; the number of dense layers (2-layer, 3-layer), the number of convolutional layer(1- layer, 2 layer) and the number of hidden nodes in each layer (32-nodes, 64-nodes) of convolutional neural networks.

Apart from these we have also considered learning rate, batch size and number of epochs as hyperparameters.

Recorded the Training and Validation loss is as follows:



Recorded Training and Validation Accuracy is as follows:

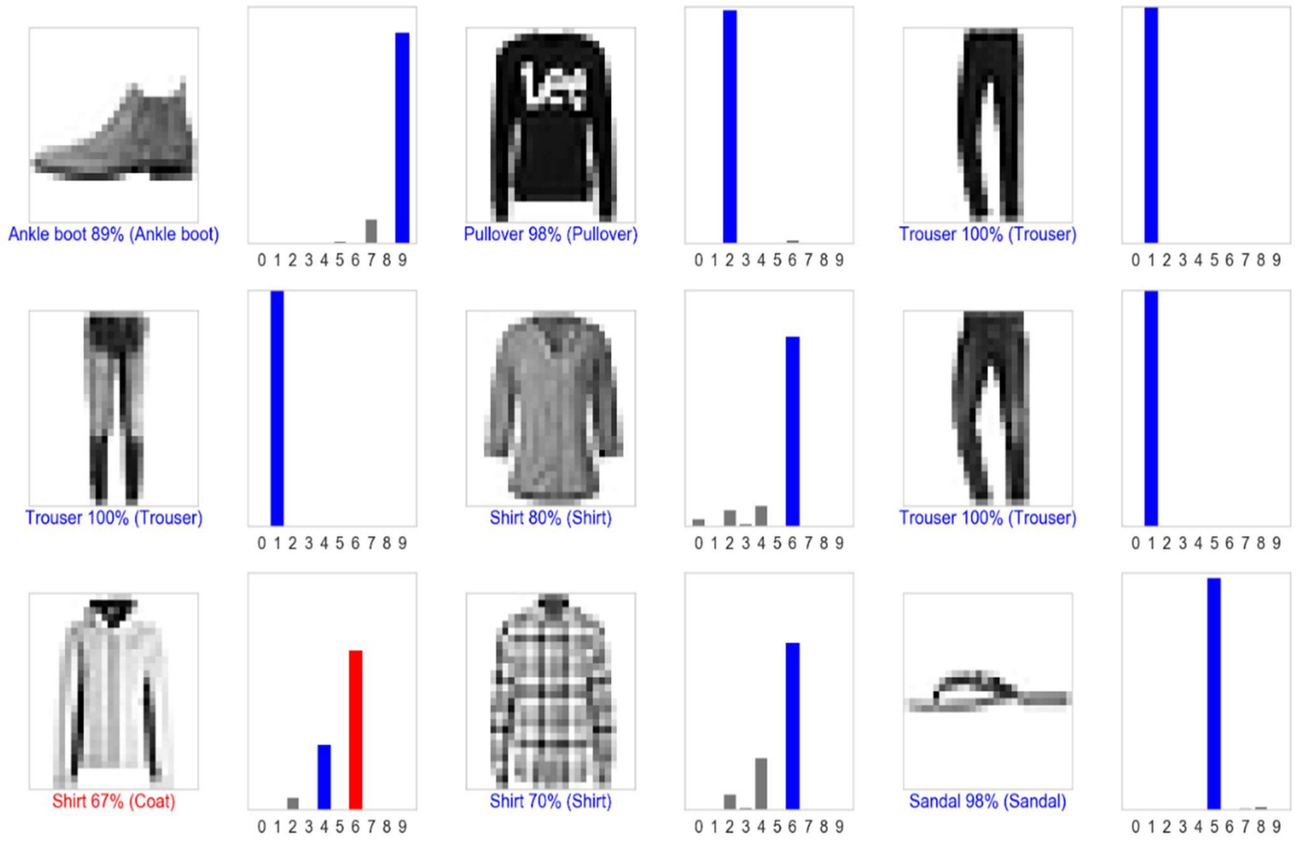


By observing the results, we concluded that best model parameters are 64 nodes in 2 dense layers and we predict the test data for this parameter to get an **accuracy of 84.84%**

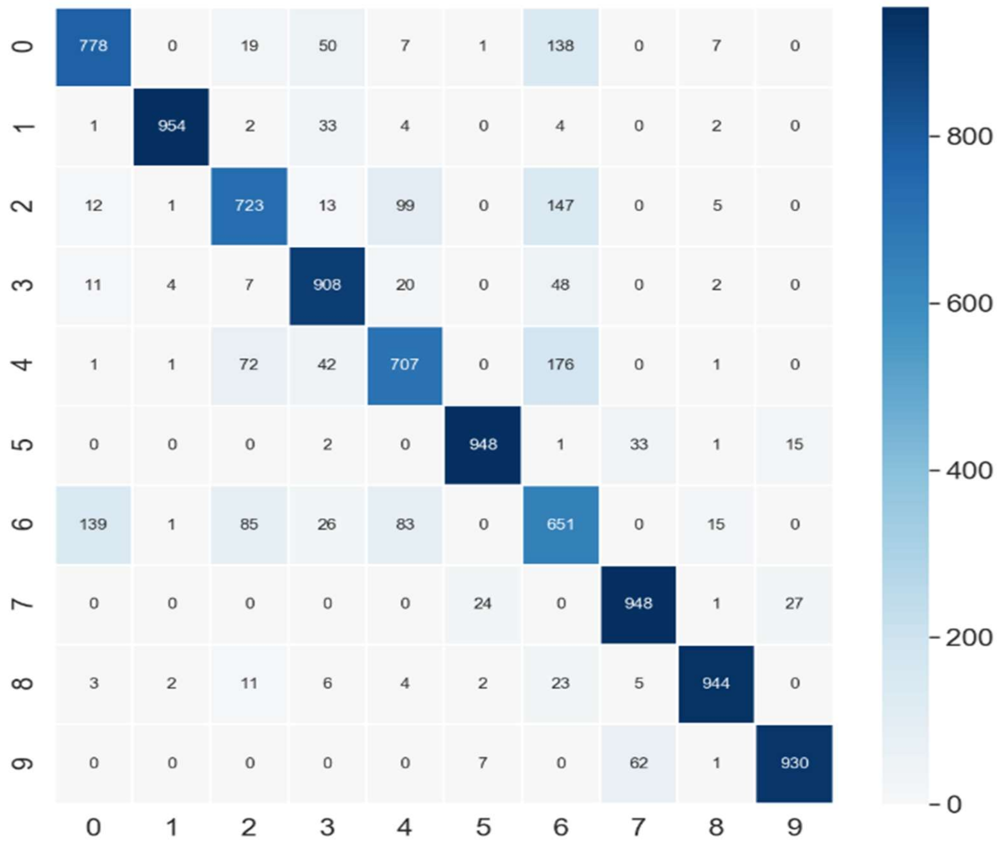
The overall performance is as follows:

Accuracy	
Training Data	86.00%
Validation Data	85.00%
Test Data	85.00%

We plot the sample test images and check the full set of 10 class predictions as below:

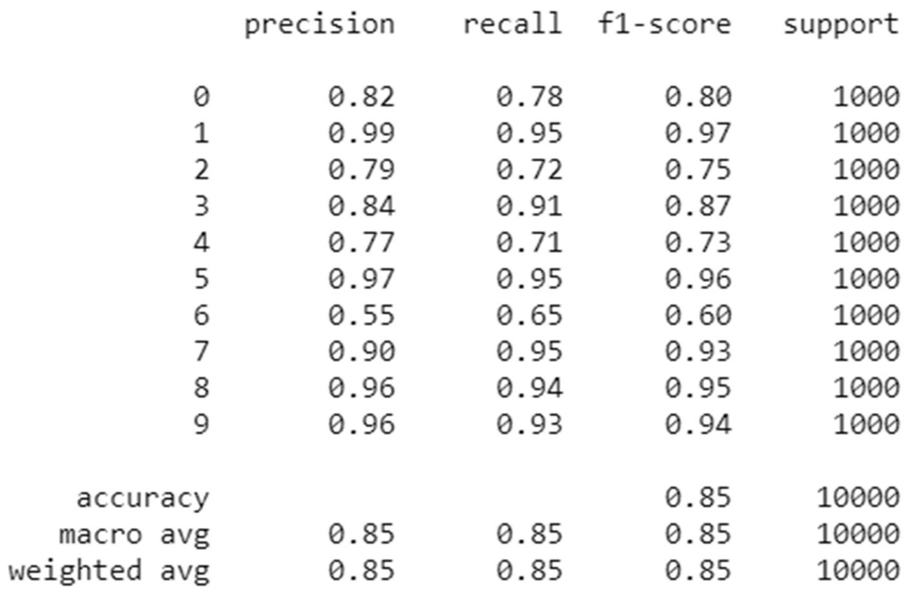


The confusion matrix in this case is as follows:



	0	1	2	3	4	5	6	7	8	9
0	778	0	19	50	7	1	138	0	7	0
1	1	954	2	33	4	0	4	0	2	0
2	12	1	723	13	99	0	147	0	5	0
3	11	4	7	908	20	0	48	0	2	0
4	1	1	72	42	707	0	176	0	1	0
5	0	0	0	2	0	948	1	33	1	15
6	139	1	85	26	83	0	651	0	15	0
7	0	0	0	0	0	24	0	948	1	27
8	3	2	11	6	4	2	23	5	944	0
9	0	0	0	0	0	7	0	62	1	930

Now we will use the Confusion Matrix to calculate the value of Accuracy, Precision, Recall and F1 score.



```
              precision    recall  f1-score   support

           0       0.82      0.78      0.80      1000
           1       0.99      0.95      0.97      1000
           2       0.79      0.72      0.75      1000
           3       0.84      0.91      0.87      1000
           4       0.77      0.71      0.73      1000
           5       0.97      0.95      0.96      1000
           6       0.55      0.65      0.60      1000
           7       0.90      0.95      0.93      1000
           8       0.96      0.94      0.95      1000
           9       0.96      0.93      0.94      1000

    accuracy                           0.85     10000
   macro avg       0.85      0.85      0.85     10000
weighted avg       0.85      0.85      0.85     10000
```

Conclusion

This paper presents the development and the simulation of a neural network and convolutional neural network machine learning model using Fashion-MNIST, a fashion product images dataset. The presented ML algorithm exhibited high performance on recognizing an image and identify it as one of the ten designated classes. Consequently, the statistical measures on the classification problem were also satisfactory.

Acknowledgments

We are extremely grateful to Professor Sargur Srihari and Mihir Chauhan for teaching all the necessary concepts related to Logistic Regression and helping in this project throughout.

References

- https://www.nicolamanzini.com/single-hidden-layer-neural-network/
- https://www.kaggle.com/gpreda/cnn-with-tensorflow-keras-for-fashion-mnist
- https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
- https://github.com/justanothergirlwhocodes/TensorFlowNNs/blob/master/NN_MNIST_depth.ipynb
- https://medium.com/tensorflow/hello-deep-learning-fashion-mnist-with-keras-50fcff8cd74a
- https://www.apress.com/gp/book/9781484236727