Multiclass Classification Using Neural Networks and Convolutional Neural Networks

Upasana Ghosh
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
upasanag@buffalo.edu

Abstract:

The purpose of this project is to design a Neural Network and a Convolutional Neural Network to perform classification on a multi class Classification Problem. We are provided with the Fashion-MNIST dataset. Fashion-MNIST is a dataset of Zalando's article images, consisting of a total 70,000 examples. Each example is a 28x28 grayscale image, associated with a label from 10 classes. We would be classifying the images into classes (0-9): T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag and Ankle Boot using a Feed Forward Backpropagation network and a Convolutional Neural Network. For this, the label field or the output in the original dataset have been preprocessed using one hot labelling. Three approaches have been applied to perform classification on the same dataset. The initial approach involves building a basic Neural Network from scratch with one hidden layer to be trained and tested on the dataset. The second approach leverages the functionalities provided by Keras, to build a Multi-layer Neural Network and the third approach involves using the Keras functionalities again to build a Convolutional Neural Network (CNN). The graph that shows the cost and accuracy trends with epoch has been leveraged to verify if the tuning of the hyperparameters are correct. Finally, the accuracy, recall and precision on the testing data set has been calculated to validate if the model can predict the correct outputs for unknown datasets in the future.

Introduction:

A Classification task is a task in which we divide the input data into discrete categories or separate classes. There are mainly two types of classification tasks:

- ***Binary classification***: where we map our data set broadly into two discrete categories or classes.
- ***Multi-class classification***: where we map the data in our data set into multiple discrete classes or categories

Here we are presented with a Multi-class Classification task and we would be using Neural Network models as our classifiers to solve the problem. The simplest definition of a neural network, more often referred to as an **'Artificial' Neural Network (ANN)**, is provided by the inventor of one of the first neurocomputers, Dr. Robert Hecht-Nielsen. He defines a neural network as:

"...a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

From "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989

ANNs are basically processing units that are loosely modeled after the neuronal structure of a human brain but on a much smaller scale. Neural networks are typically organized in layers made up of several interconnected 'nodes' which contain an 'activation function'. Patterns are presented to the network through the 'input layer', which communicates to one or more 'hidden layers' where the actual processing is done via a system of weighted connections. The hidden layers then link to an 'output layer' which provides the proper classification for the data. This type of a neural network, where the signals flow in only one direction: from input, through successive hidden layers, to the output is known as a feedforward network. Here I have implemented a feedforward backpropagation network to solve the given classification problem. A feedforward backpropagation network is a feedforward neural network that has been trained using the backpropagation training algorithm. The backpropagation training algorithm subtracts the training output from the target (desired answer) to obtain the error signal. It then goes back to adjust the weights and biases in the input and hidden layers to reduce the error.

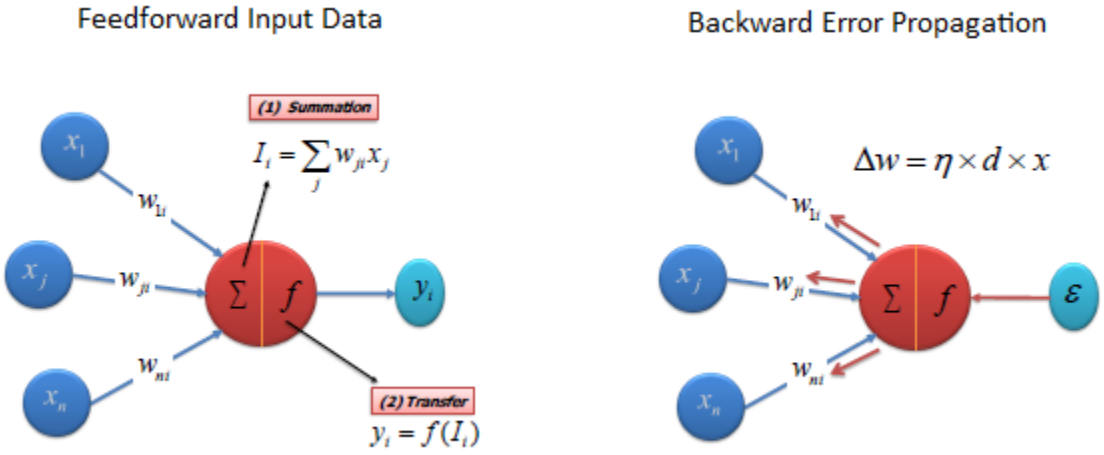


Fig 1: Feed Forward – Backward Propagation Neural Network

Three approaches have been applied to solve the given classification problem. The initial approach involves building a basic **Feed Forward Backward Propagation Neural Network** from scratch with one hidden layer to be trained and tested on the dataset. The second approach leverages the functionalities provided by Keras, an open-source neural network library, to build a **Multi-layer Neural Network** and the third approach involves using the Keras functionalities again to solve the problem by building a **Convolutional Neural Network (CNN)**. A Convolutional Neural Network, also known as CNN, is a class of neural networks that specializes in processing data that has a grid-like topology, such as an image. A CNN typically consists of the following layers:

- *Convolutional layer* – This layer extract features from the input image by performing a dot product between a set of learnable parameters known as the kernel and a restricted portion of the image or the receptive field.

- *Pooling layer* – This layer replaces the output of the network at certain locations by deriving a summary statistic of the nearby outputs. This helps in reducing the spatial size of the representation, which decreases the required amount of computation and weights.

- *Fully Connected layer* - The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset using a softmax activation function.

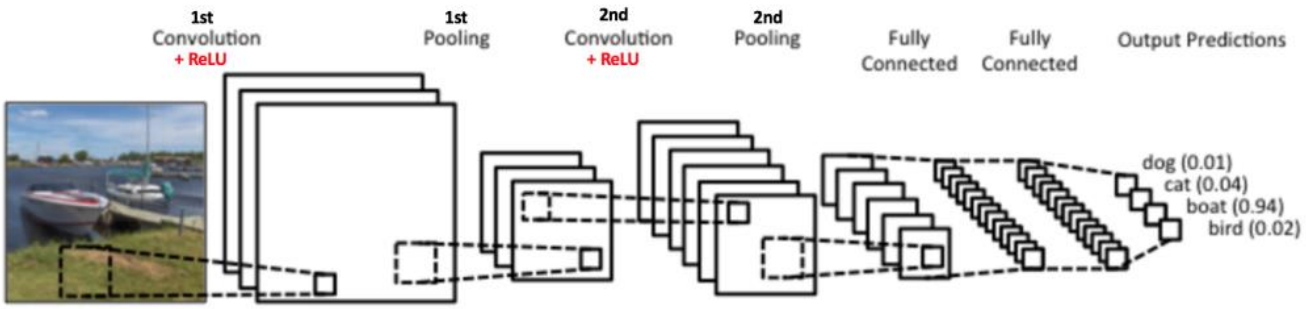


Fig 2: Convolutional Neural Network

To train and evaluate the Neural Network model that I'm designing here for the Fashion-MNIST dataset, I have divided the dataset into 'Training' and 'Testing' datasets and have used the confusion matrix to calculate the accuracy, precision and recall for each epoch or iteration. I have also used the following graphs to evaluate the model further:

- Training Cost VS Epoch
- Training Accuracy VS Epoch

Dataset:

We are using the Fashion-MNIST dataset for training and testing our classifiers. 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 the test data sets have 785 columns. The first column consists of the class labels (Fig 3 below) and represents the article of clothing. The rest of the columns contain the pixel-values of the associated image. We would be assigning each training and test example to one of the labels

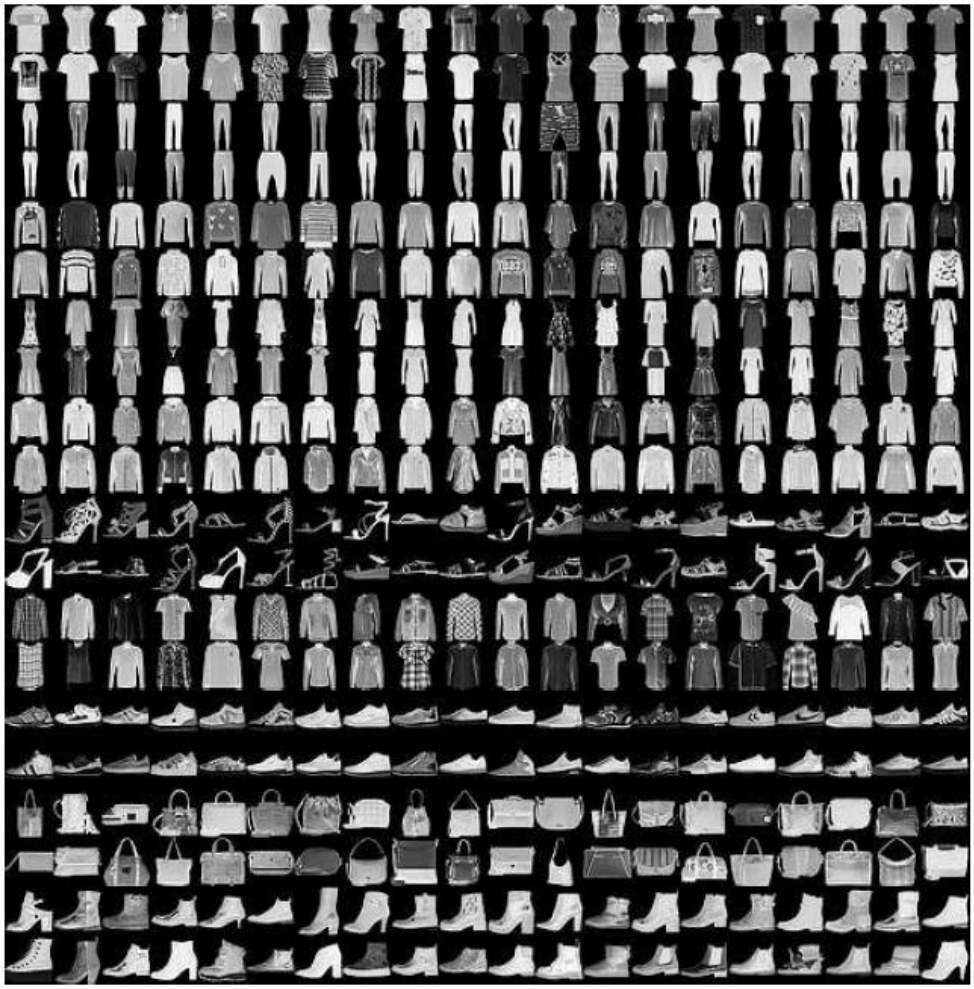
Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Fig 3: Example of MNIST Fashion Dataset

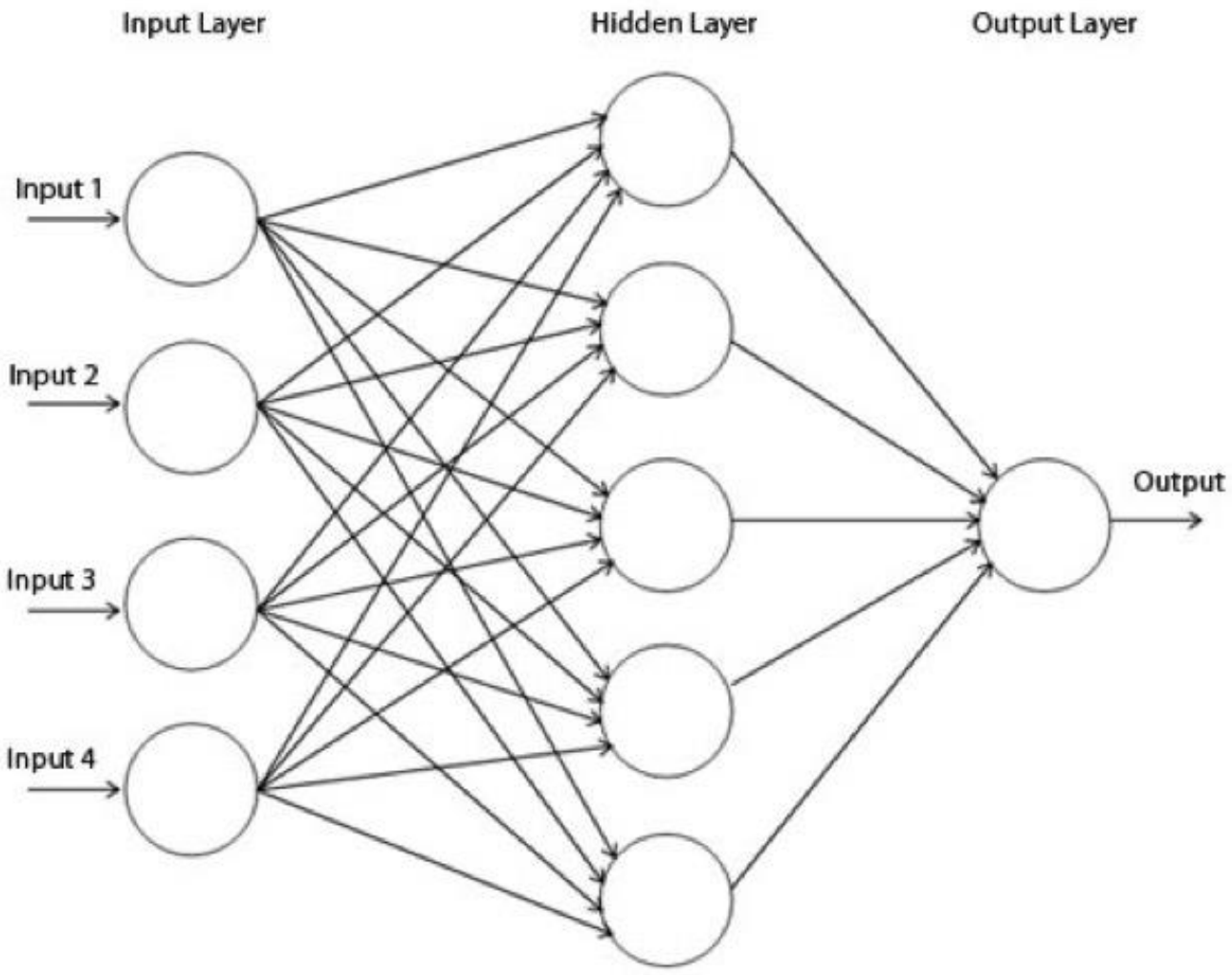
Preprocessing:

The original MNIST Fashion dataset is downloaded and processed into a Numpy array using the mnist reader. The dataset consists of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 grayscale image having pixel-values ranging from 0 to 255. The input data for the training dataset has been normalized by dividing each input by 255 to avoid weight parameter explosions and to properly tune our hyperparameters with each iteration. The output or the label fields for the training and testing data sets has also been preprocessed using the one hot labelling scheme for the ease of mapping while calculating the loss and accuracy with our predicted output and the original output.

Architecture:

Approach 1: Implementing Feed Forward Backward Propagation Neural Network

The Multiclass Classification problem presented here is solved using a Feed Forward Backward Propagation Neural Network. A generalized model of a Feed Forward Backward Propagation Neural Network Classifier can be shown below:



For each of the hidden layers, we leverage the **Sigmoid function** as our Activation function to scale the input values to a probabilistic discrete measure ranging between 0 and 1. In the final layer or the output layer we use the **Softmax function** to predict a probabilistic value that can be mapped to a class (0 – 9) for the data or image passed as input.

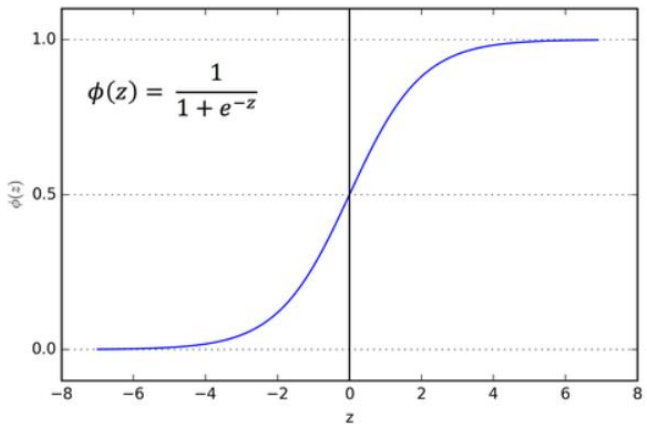


Fig: Sigmoid Function

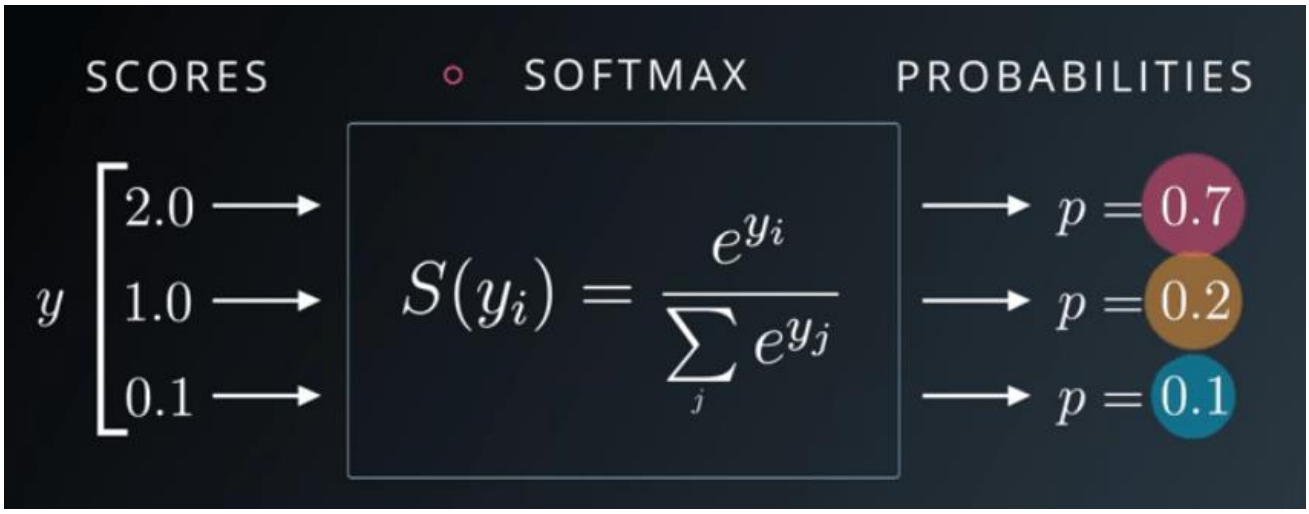


Fig: Softmax function

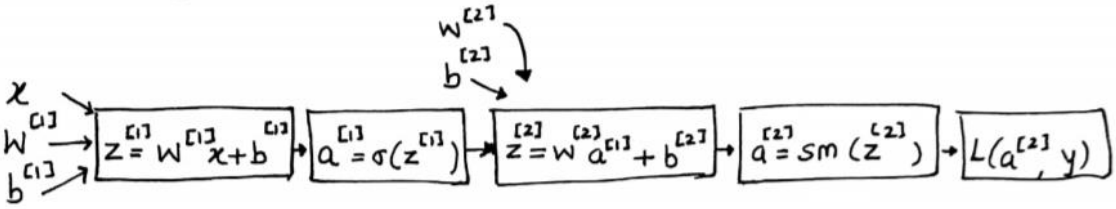
For this project, we have simplified our neural network model to contain a single hidden layer. We need to initialize the weight vector W with either some random values or as a zero vector. In this case the weight vector W has been initialized with random values. Also, the bias has been initialized with a random value. By adjusting the learning rate, we tune these hyperparameters for each epoch. The genesis function $Z_j = (W_i^T X_i) + b_i$ is calculated for each input value X_i, where i = 1, 2, 3 ….. 784 for each epoch and is passed through the activation function of the next layer. The activation function for the hidden layer is a Sigmoid function for our case:

$$\sigma(Z) = \frac{1}{1+e^{-z}},$$

The Gradient Descent for neural network has been used to train the model using a group of hyperparameters. In the output layer of the network, the output of each hidden node is passed through a Softmax function that acts as a predictor function predicting a probabilistic value for each of the ten output classes. The highest probability value among the ten values gives the output class of the input image. Considering the total epoch as 500 for this approach, for each epoch we calculate the following parameters while training the model:

Forward Propagation:

$$z^{[1]} = W^{[1]} x + b^{[1]}$$
$$a^{[1]} = \sigma(z^{[1]})$$
$$z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}$$
$$a^{[2]} = \text{softmax}(z^{[2]})$$
$$L(a^{[2]}, y) = -\sum y \log a^{[2]}$$



Backward Propagation:

1. $\Delta a_2 = \frac{\partial L}{\partial a_2}$
2. $\Delta z_2 = \frac{\partial L}{\partial z_2} = a_2 - y$
3. $\Delta a_1 = \frac{\partial L}{\partial z_2} \times \frac{\partial z_2}{\partial a_1} = (a_2 - y)W_2$
4. $\Delta z_1 = \Delta a_1 \frac{\partial a_1}{\partial z_1} = (a_2 - y)W_2\, a_1(1 - a_1)$

5. $\Delta W_2 = (a_2 - y)a_1$
6. $\Delta W_1 = (a_2 - y)W_2\, a_1(1 - a_1)x$

After every forward propagation, we calculate the cost using the following loss function, where L is the Cross-Entropy Function (i.e., Loss Function) and do a backward propagation to tune our hyperparameters accordingly:

$$L = \frac{-(ylogp + (1-y)\log(1-p)}{m}$$

We have used a confusion matrix to evaluate the model. The Confusion Matrix gives us the count of the True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) in the predicted output. We can define the TP, TN, FP and FN as follows:

		PREDICTED	
		Positive	Negative
ACTUAL	Positive	TP	FN
	Negative	FP	TN

Using these values, we calculate the Accuracy, Precision and Recall for the training, validation and testing data sets with the help of the following formulae:

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

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

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

The trend in the accuracy, precision and recall values show us whether we are tuning the hyperparameters correctly and finally help us judge if our model is trained enough to predict the correct output for an unknown dataset in the future.

Approach 2: Implementing Multilayer Neural Network using Keras

In the second approach, we have used the functionalities provided by the Keras library to implement a Multilayer Neural Network. The model type that we are using here to define our model is Sequential. Sequential is one of the simplest and easiest model types provided by Keras to build Neural Network models layer by layer. We use the Flatten ( ) function to flatten the input images to a shape 28X28 and then add two hidden layers using the Dense ( ) function. For each of the hidden layers we have defined 128 nodes and have set the activation function to Sigmoid function. For the output layer or the last layer of our Neural network, we have defined 10 nodes for classifying the input data to the ten different classes and for this layer we have used the Softmax function as our activation function. The following figure gives the summary of our model:

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_16 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_40 (Dense)             (None, 128)               100480
_________________________________________________________________
dense_41 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_42 (Dense)             (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Stochastic Gradient Descent for Neural Networks has been used as the optimizer function for compiling our model by calculating the loss using the Sparse Categorical Cross Entropy and the metrics as accuracy. The model fitting has been done using the fit( ) method by training the model using the training data and taking the epoch as 200, the input batch size as 100 and the test data set for the validation. The accuracy and the loss have been calculated for the test dataset to evaluate the performance of the model. Also, we have the accuracy v/s epoch and the loss v/s epoch graph for the test and the training data set depicting the performance of our model.

Approach 3: Implementing Convolutional Neural Network (CNN) using Keras

In the third approach, we use a Convolutional Neural Network (CNN) to perform classification on the given MNIST Fashion dataset. To implement this Neural Network Model, we have used the functions provided by the high-level Neural Network Keras library. We have used the to_categorical( ) function for the one hot labelling of the training and test output data. The Sequential model type is used to define our model. We have added two hidden layers of the convolutional neural network using the Conv2D function, taking the nodes as 64 and 32 respectively, the kernel size as 2X2 and the activation function as Sigmoid function for both the layers. We have used the Dense function to define out output layer with the Softmax function as the activation function for this layer. The following figure gives the summary of our model:

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_20 (Flatten)         (None, 784)               0
_________________________________________________________________
dense_50 (Dense)             (None, 128)               100480
_________________________________________________________________
dense_51 (Dense)             (None, 128)               16512
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1290
=================================================================
Total params: 118,282
Trainable params: 118,282
Non-trainable params: 0
```

Stochastic Gradient Descent for Neural Networks has been used as the optimizer function for compiling the model by calculating the loss using the Sparse Categorical Cross Entropy and the metrics as accuracy. The model fitting has been done using the fit( ) method by training the model using the training data and taking the epoch as 100. The test data set has been used for the validation. The accuracy and the loss have been calculated for the test dataset to evaluate the performance of the model. Also, we have the accuracy v/s epoch and the loss v/s epoch graph for the test and the training data set depicting the performance of our model.

Results:

Approach 1: Implementing Feed Forward Backward Propagation Neural Network

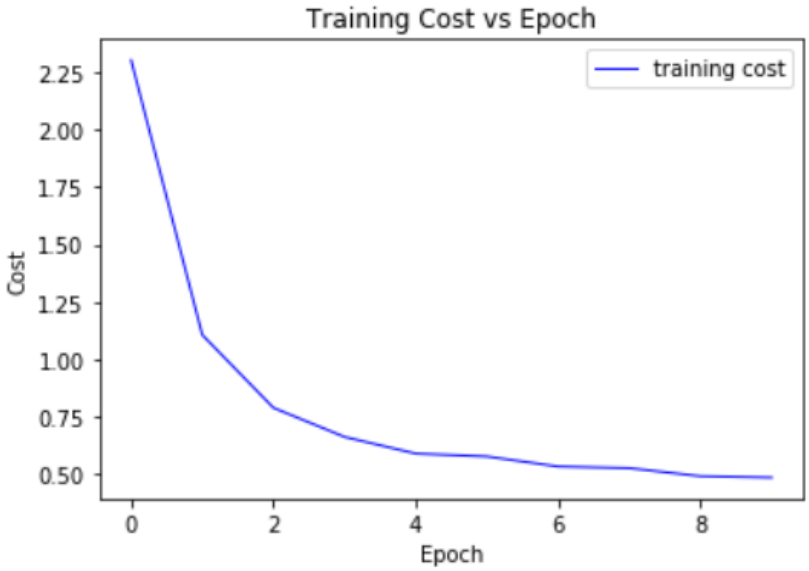
- The trend in the calculated cost and accuracy per epoch for the training dataset calculated over 500 epochs is as follows (Data collected after every 50th epoch):

```
Epoch 0 cost:  2.302420657485193 accuracy:  0.9
Epoch 50 cost:  1.1083824249062542 accuracy:  0.925885
Epoch 100 cost:  0.7880833261001491 accuracy:  0.9475033333333334
Epoch 150 cost:  0.6642203634450724 accuracy:  0.952245
Epoch 200 cost:  0.5941118077898931 accuracy:  0.958595
Epoch 250 cost:  0.5862622705637829 accuracy:  0.9580466666666667
Epoch 300 cost:  0.5348621547740406 accuracy:  0.961375
Epoch 350 cost:  0.5070168247050378 accuracy:  0.96437
Epoch 400 cost:  0.4930972624447583 accuracy:  0.966425
Epoch 450 cost:  0.4801593717512722 accuracy:  0.96713
```

- The accuracy, precision and recall calculated on the Testing dataset:

```
 CALCULATED VALUE OF ACCURACY, PRECISION AND RECALL ON TEST DATA FOR FASHION-MNIST DATASET
-------------------------------------------------------------------------------------------
ACCURACY VALUE: 0.96543
PRECISION VALUE: 0.9860111111111111
RECALL VALUE: 0.9758299519458098
```

- The Training Cost v/s Epoch graph for the training data:



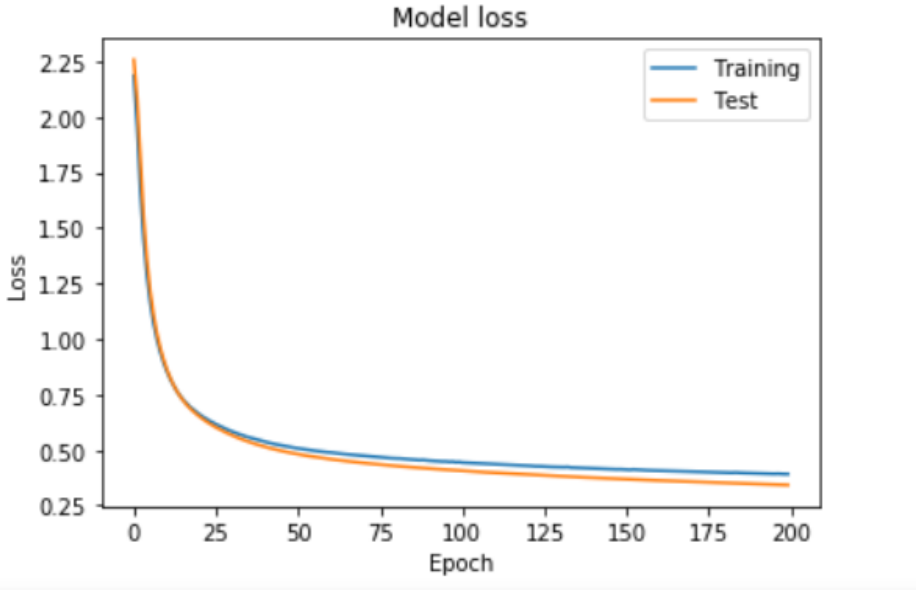
- The Training Accuracy v/s Epoch graph for the training data:

Approach 2: Implementing Multilayer Neural Network using Keras

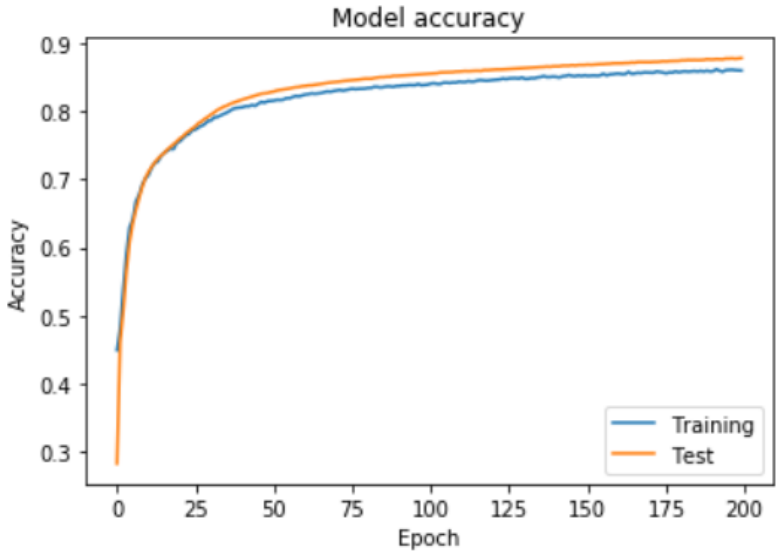
- The loss and accuracy calculated for the Testing dataset:

```
 CALCULATED VALUE OF ACCURACY AND LOSS ON TEST DATA FOR MULTI LAYER NEURAL NETWORK
-------------------------------------------------------------------------------------------
TEST LOSS:0.3917805345058441
TEST ACCURACY: 0.8593000173568726
```

- The Loss v/s Epoch graph for the training and test data:



- The Accuracy v/s Epoch graph for the training and test data:

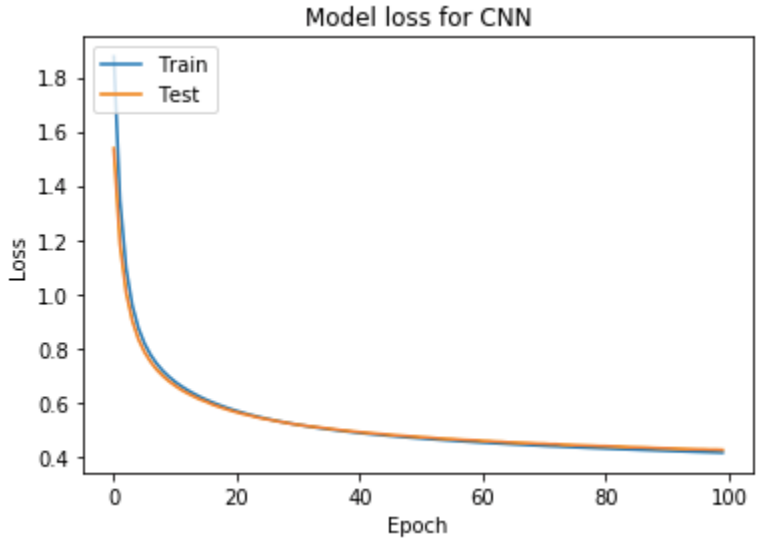


Approach 3: Implementing Convolutional Neural Network (CNN) using Keras

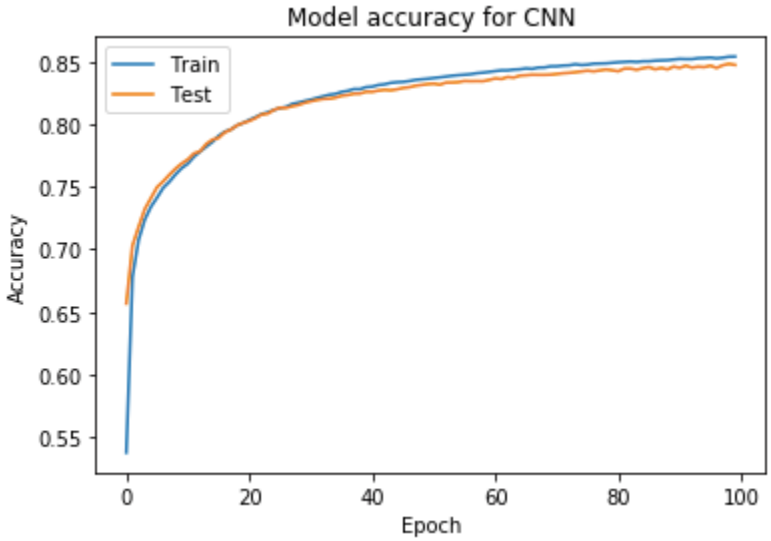
- The loss and accuracy calculated for the Testing dataset:

```
 CALCULATED VALUE OF ACCURACY AND LOSS ON TEST DATA FOR CONVOLUTIONAL NEURAL NETWORK
-------------------------------------------------------------------------------------------
CNN MODEL TEST_DATA LOSS:0.34554590249061584
CNN MODEL TEST_DATA ACCURACY: 0.8792999982833862
```

- The Loss v/s Epoch graph for the training and test data:



- The Accuracy v/s Epoch graph for the training and test data:



Conclusion:

This project to design a single layered Neural Network, a multi layered Neural Network and a Convolutional Neural Network to perform classification on a multi class Classification Problem provides us with an insight about how different Neural Network models can be used for modelling classifiers to solve classification problems involving real world data sets. This also gives us an opportunity to compare the performance of the different models when applied to the same dataset. From the graphs in our result, we can conclude the following:

- The Accuracy v/s Epoch graph shows that with the increase in the number of epochs, the difference between the predicted output and the actual output decreases and after a certain iteration the predicted output becomes almost same. Hence, it can be said that if our model is well trained, it is capable of predicting the correct output for unknown data sets.

- The cost v/s epoch graph shows that as we tune the hyperparameters correctly with the increasing iterations, the cost becomes lower and the cost of training data set starts resembling the cost of the test data set which means that the loss reduces as we tune our hyperparameters correctly.

- The accuracy calculated over the testing data set shows how accurately our model can predict the output for a set of unknown data set in the future.

Acknowledgement:

1. An Introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani
2. http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html
3. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
4. https://towardsdatascience.com/convolutional-neural-network-for-image-classification-with-implementation-on-python-using-pytorch-7b88342c9ca9
5. https://blogs.oracle.com/datascience/convolutional-neural-networks%2c-explained
6. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/
7. https://victorzhou.com/blog/keras-neural-network-tutorial/
8. https://mlfromscratch.com/neural-networks-explained/#/
9. https://dev.to/nexttech/introduction-to-multilayer-neural-networks-with-tensorflow-s-keras-api-39k6