

**Machine Learning Mini-Project Report**  
**MTCS - 204(P) Fashion MNIST Problem**

**Group:**

**Saurav Rai (17558)**

**Saichand A V R P (17552)**

**Akhilesh Pandey (17551)**

**AIM :**

Variations of Neural Network Models for image classification: fashion-MNIST.

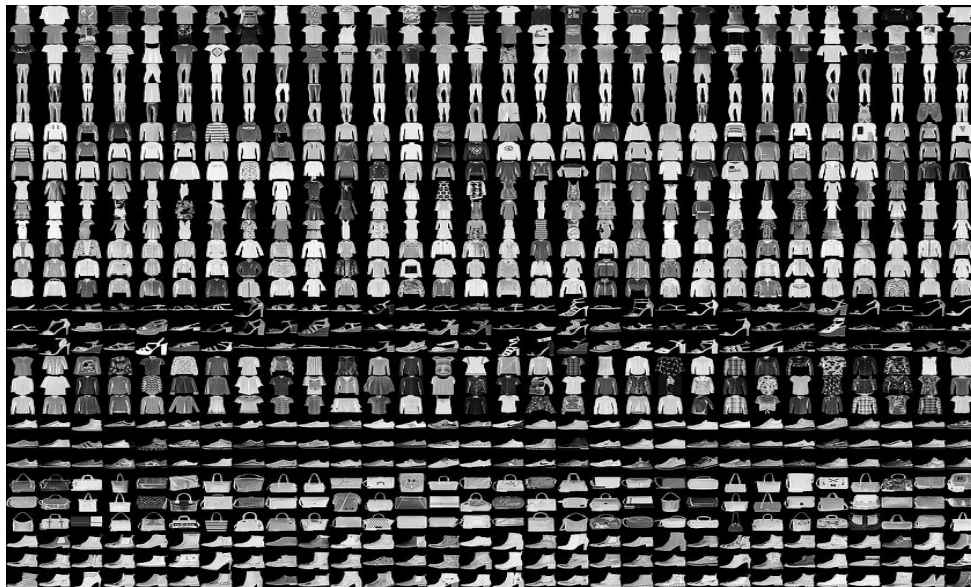
**Experimental Procedure :**

**Platforms Used :**

1. ( Saurav Rai ) : **Python3**( backend tensorflow ).
2. ( Saichand ) : **IPython**(anaconda3), **Keras, Python** ( backend Tensorflow).
3. ( Akhilesh Pandey ) : **Python3, Pytorch**

**Dataset Description :**

Fashion-MNIST is a dataset of Zalando's article images - consisting of training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28x28 gray scale image, associated with a label from 10 classes. An example of how the data looks:



*Figure reproduced from zalando's github page<sup>1</sup>.*

Each class takes three-rows in the data visualized above. 7-step conversion process is used to generate the Fashion-MNIST dataset<sup>2</sup>. Fashion-MNIST poses a more challenging classification task than the simple MNIST digits data.

<sup>1</sup> <https://github.com/zalando-research/fashion-mnist>; <sup>2</sup> [1708.07747v2\[cs.LG\], 15 Sep, 2017 arxiv.](https://arxiv.org/abs/1708.07747v2)

## Models Used : ( Work done by Saichand (17552))

S.No.	Classifier	Activation	Optimizer
1	3 ConvLayers with maxpooling and 2 FC, 241546 parameters ( <b>keras</b> )	Relu,softmax	Adam
	3 ConvLayers with maxpooling and 2 FC, 241546 parameters ( <b>keras</b> )	Relu, tanh	Adagrad
	3 ConvLayers with maxpooling and 2 FC, 241546 parameters ( <b>keras</b> )	Tanh, softmax	Adamax
	3 ConvLayers with maxpooling and 2 FC, 241546 parameters ( <b>keras</b> )	Sigmoid, relu	Adam
2	MLP with one hidden layer(256), ( <b>tensorflow API</b> )	Relu, sigmoid	Adagrad
	MLP with one hidden layer(256), ( <b>tensorflow API</b> )	Sigmoid, Relu	Adam
	MLP with one hidden layer(256), ( <b>tensorflow API</b> )	Sigmoid, Relu	Gradient Descent
3	Logistic Regression ( <b>python</b> )	Softmax	LBFGS
	K-Class Logistic Regression	Softmax	liblinear

We have used 3 distinct models, with varied Activation functions and Optimization techniques. 3 Models used are:

√ **Convolutional Neural Network (CNN)**: Algorithm has 3 convolution layers with different activation functions and each followed by a maxpooling phase with pool\_size (2, 2), which gives translation invariance. A dropout phase is used after every maxpool phase for regularization. The algorithm uses around 2.5 lakh parameters. 1.5 lakh parameters are given as input for the first Fully Connected layer and 1290 parameters to the second Fully Connected layer.

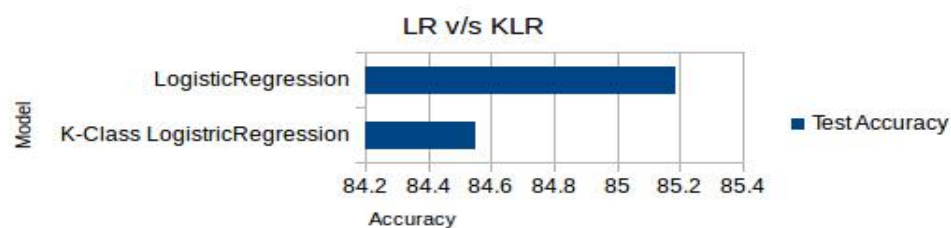
√ **Multi Layer Perceptron (MLP)**: Algorithm used is a simple multi layer perceptron model with one hidden layer. Initial parameters to input node are 784. Hidden layer size is 256. The output layer has nodes equal to the number of labels(10). Weights are initialized using *random\_normal* function of tensorflow package. *Softmax\_cross\_entropy\_with\_logits* is the cost function from the neural network package of tensorflow.

√ **Logistic Regression (LR)**: Algorithm is implemented using the linear\_model library of python-sklearn, that has LogisticRegression model. We use 'lbfgs' solver as the optimizer for the model, and softmax function for predicting probabilities. We also implemented the k-class LR model with *multinomial = 'ovr'*, which fits a binary problem for each label. The default optimizer used for this model is 'liblinear' which is an open

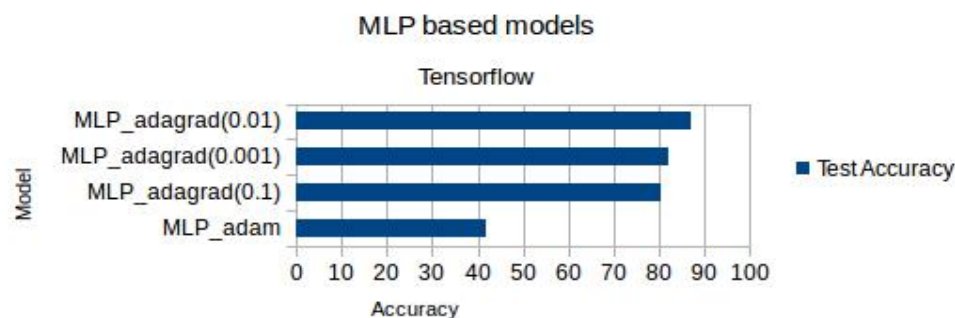
source library for large-scale linear classification.

## Experiments Conducted:

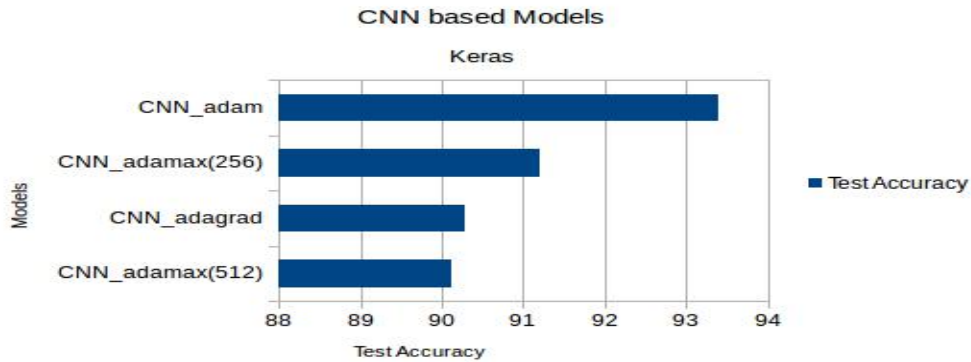
- ❖ At first, we dealt with the fashion-mnist image classification problem using Simple **Logistic Regression(LR)** to classify images, which are saved as a features in a .csv file for train and test data. A fixed train set (60,000) and a fixed test set (10,000) are used for the model as given by the creators of the fashion-MNIST (zalando). The accuracies observed in this model are not satisfactory. We observe that LR works better than k-class LR, which uses ovr(one versus rest) policy. But LR gives only **85.19** and KLR(K-Class LR) gives **84.55**.



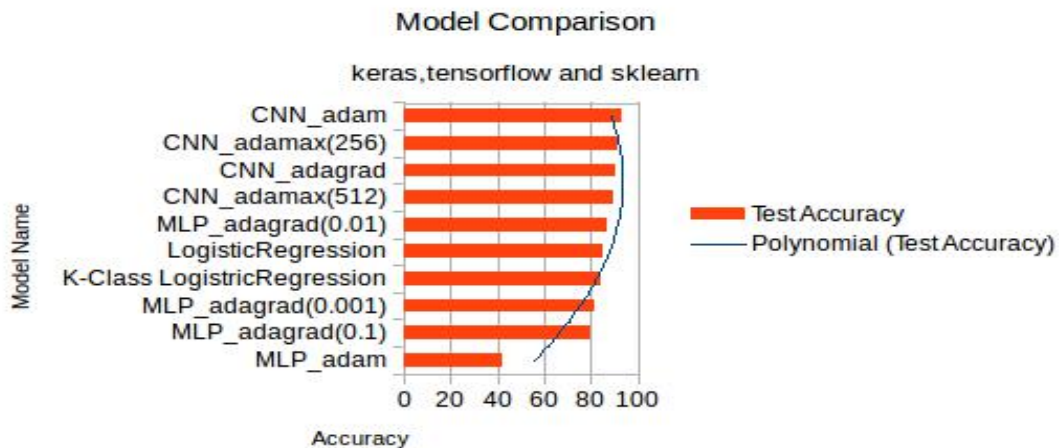
- ❖ Next, we tried to approach the problem using Simple Multi-layer Perceptron Model. We varied the optimizers and also activation functions at different layers. We noted down the accuracies for all the models and projecting the best few models that stood out in the analysis. The test accuracy has gone up till 87.01 for Adagrad optimizer with a learning rate of 0.01. But, this doesn't seem to impress because it takes enough time about half an hour, but improves by 3 percent. In the figure, by accuracy we mean the test accuracy. We even tried the Gradient Descent Technique, due to its inability of oscillation, it converged to local optima and the test accuracy was only 35.



- ❖ Then we used CNN using keras with tensorflow backend, which showed good improvement in the accuracy. The maximum test accuracy we touched was 93.4, with adamax optimizer. We say that CNN based model performed better, based on the model and also the time taken to execute was only 15 minutes compared to double the time taken for MLP and even more for LR models.



The Comparison for all the three different models shows that CNN based models outperform other models, due to the ability of multi-layer processing. The hyper parameters of the models also have been tuned to see the accuracy improve in case of both MLP and CNN models.



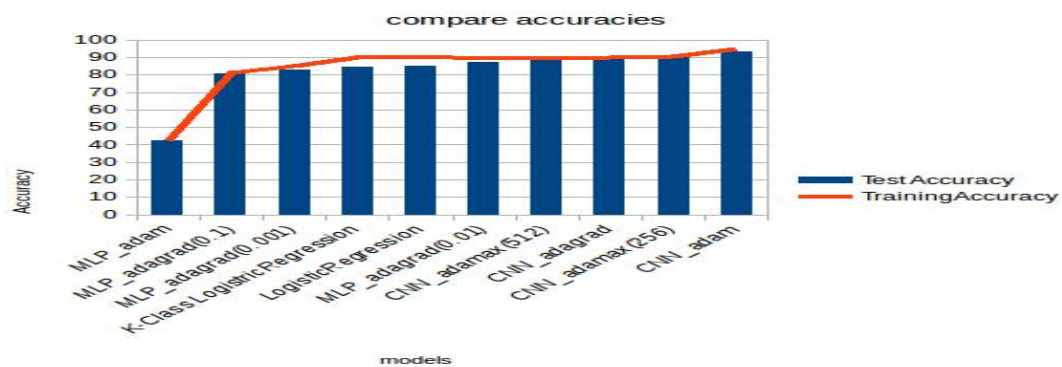
The figures 256 and 512 in CNN\_adamax model, correspond to the batch size. ROC curves for the models have been plotted and the training and test accuracies are noted down. In the above figure, the polynomial trend line shows the shift in the test accuracy.

## Observations:

The observations on the experimental models are as follows:

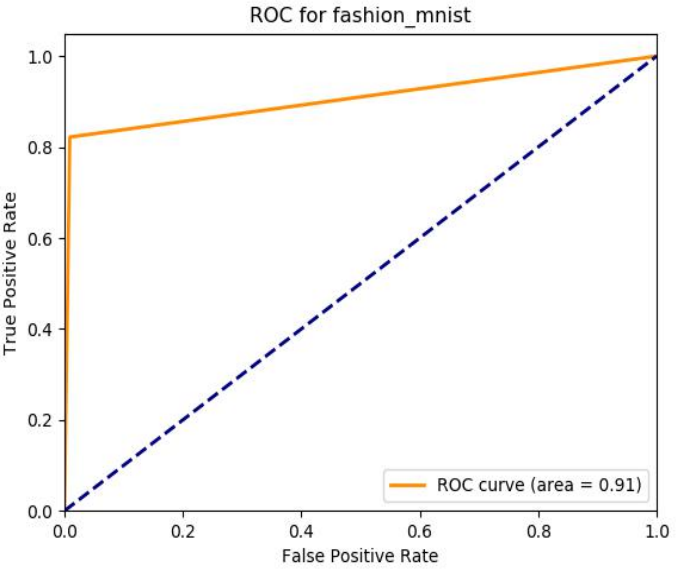
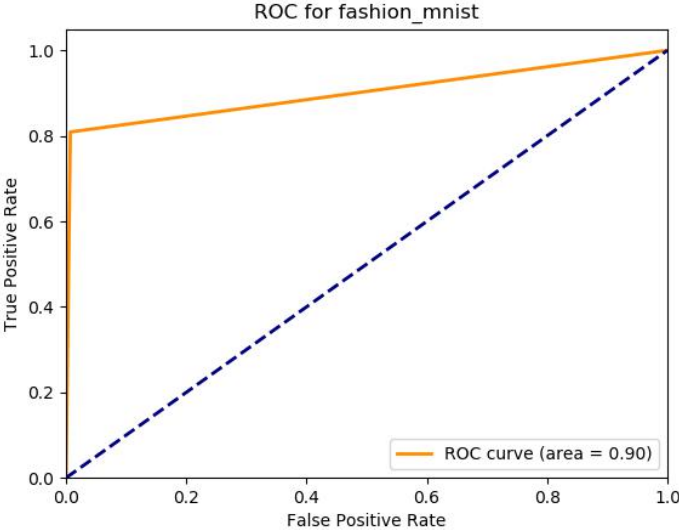
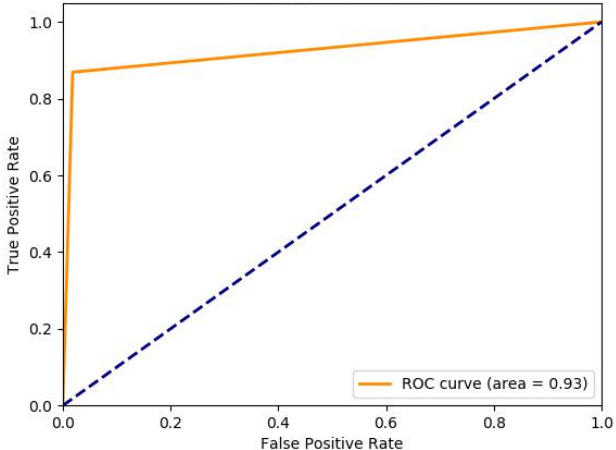
The training and test accuracies for most of the models seem to be close, and in fact for MLP and LR models, the test accuracy is less than the training accuracy. This indicates that the models when trained on themselves(training set) are better than when ran on the test data. Hence, the generalization error is more. Also, the training accuracies are all in the range of 85-95. Hence the models are not overfitting the training data. For different models we have plotted the ROC curves, given after the comparison of

accuracies. The ROC curves also show that the area under the curve is maximum for CNN based model, with adam optimizer.

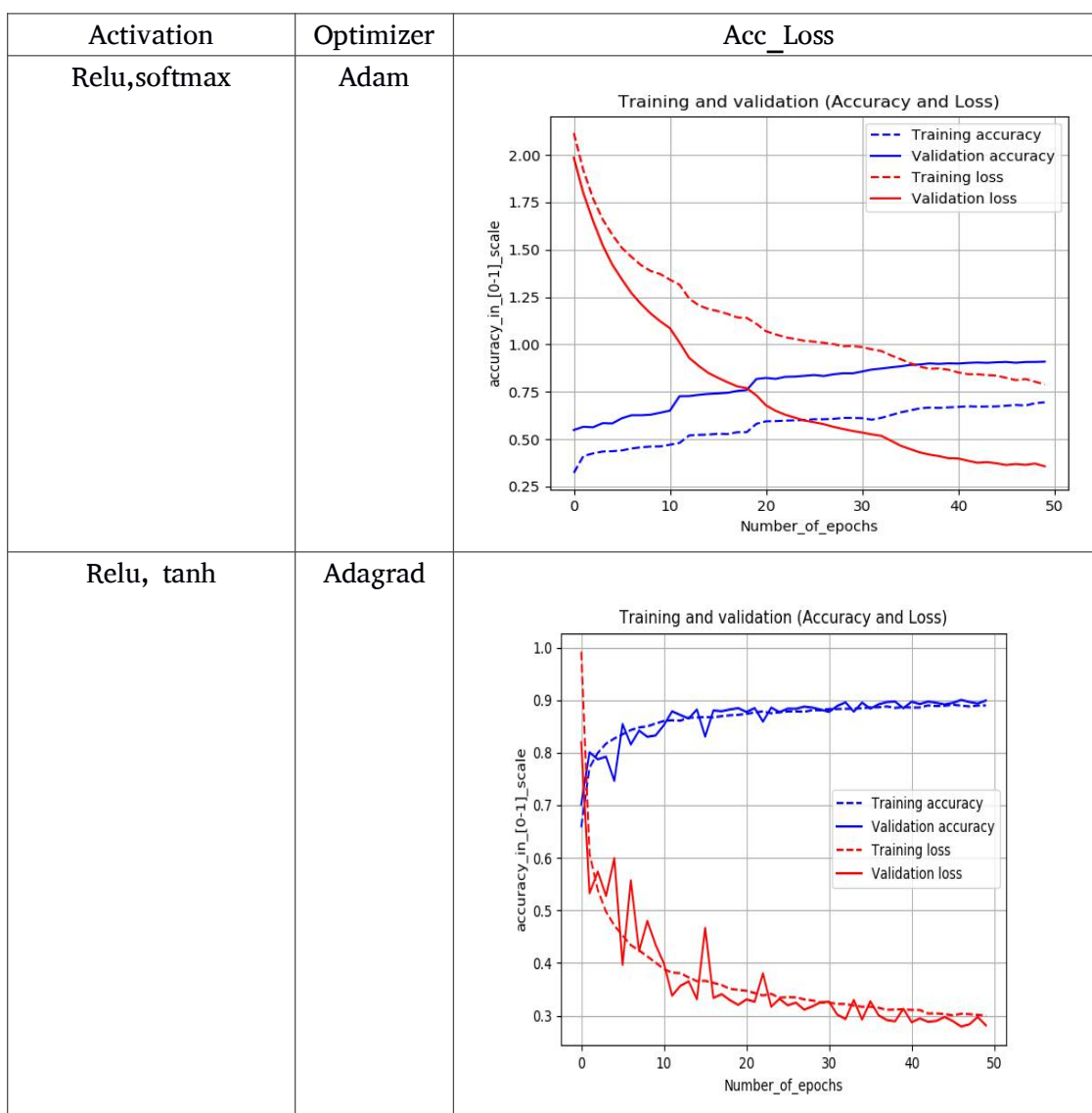
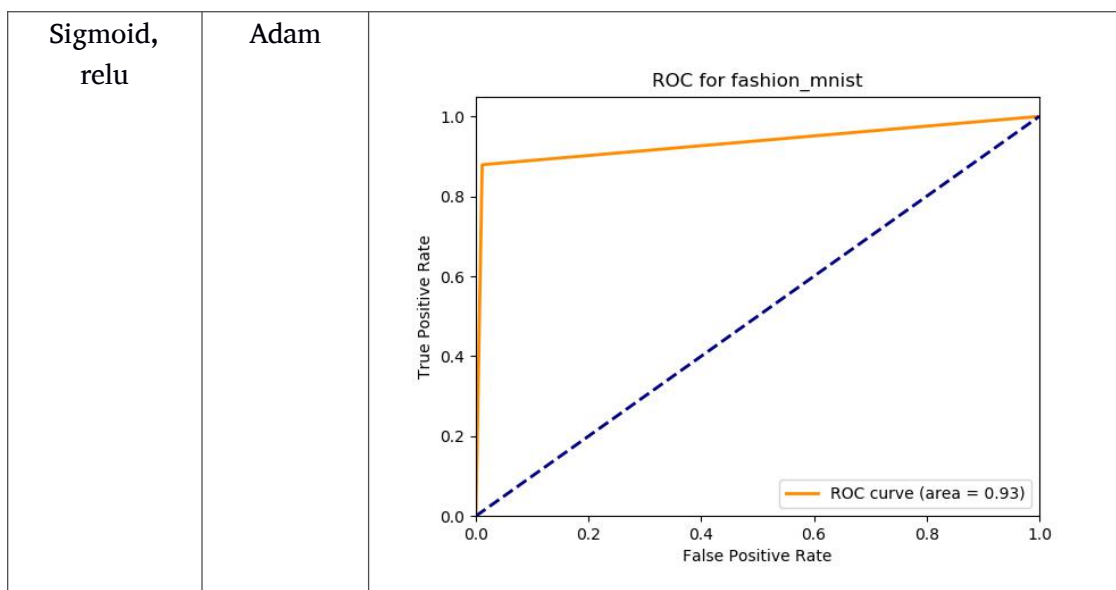


**ROC and Accuracy, Losses for CNN Models :**

Activation	Optimizer	ROC
Relu,softmax	Adam	
Relu, tanh	Adagrad	

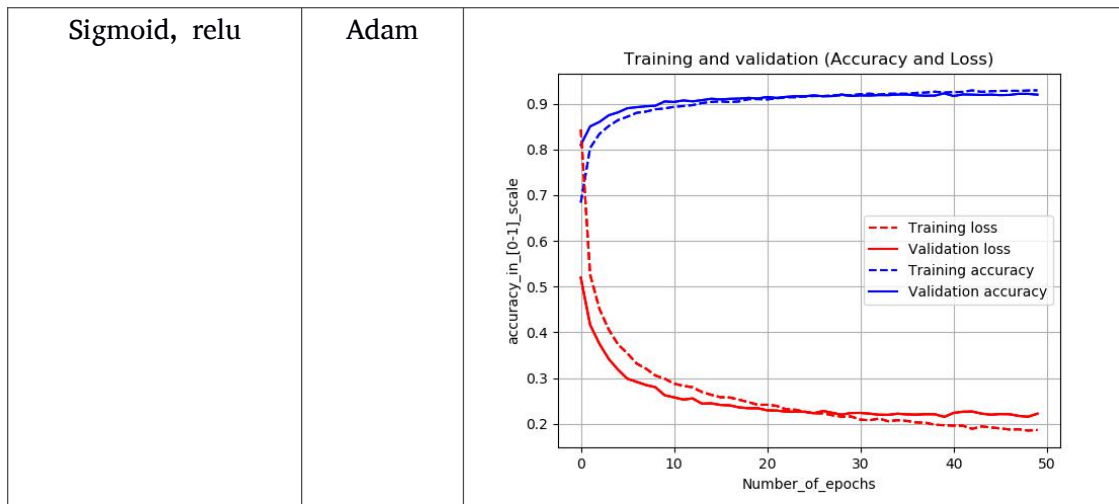
Relu,tanh	Adagrad (0.01)	<p>ROC for fashion_mnist</p>  <p>ROC curve (area = 0.91)</p>
Tanh, softmax	Adamax (256)	<p>ROC for fashion_mnist</p>  <p>ROC curve (area = 0.90)</p>
Tanh, softmax	Adamax (512)	<p>ROC for fashion_mnist</p>  <p>ROC curve (area = 0.93)</p>





Relu,tanh	Adagrad (lr = 0.01)	<p>Training and validation (Accuracy and Loss)</p>
Tanh, softmax	Adamax (256)	<p>Training and validation (Accuracy and Loss)</p>
Tanh, softmax	Adamax (512)	<p>Training and validation (Accuracy and Loss)</p>





## Inferences:

From our analysis, we infer that CNN boosts the accuracy, with less computation time, compared to other classifiers because of the factor of multi-layer processing. Also the validation phase of the CNN models help in boosting the accuracy. Because we split the training and validation data as 80 ad 20 percent, from the training data set. The area under the curve for CNN with adam and adamax optimizers outstands at 93. We haven't experimented thoroughly with variations of Batch\_size for the convolutional neural networks. Also, there is no much work done in MLP also. May be we can increase the hidden layers and improve the accuracies. The changes in the learning rates for CNN, MLP have showed some intuitive understandings as to, if lr is low (0.001), it takes more time and also learns very slow. If lr is too high (0.1, 0.2..), it halts much early, but drastic changes lead to more bias and reduction in accuracy can be observed. Hence, we found that a good learning rate for CNN with adagrad and adam is 0.01. But, strangely, for CNN with adamax optimizer, we see that learning rate of 0.2 gave 93 percent test accuracy.

## Conclusion:

In this mini-project, We have done an image classification for the fashion-MNIST dataset. We have used various classifiers such as CNN, MLP and LR. We have further studied the accuracies and concluded that with our minimal experiments, we observe that CNN performs better. We can extend the work to improve the performance by tuning the batch\_size for CNN, number of hidden layers for MLP and better novel solver for Logistic Regression.

Models used: ( Work done by Akhilesh Pandey (17551))

CNN

Sl. No	Conv. Layres (CL)	Model	Pool	Optimizers	Batch_norm (yes/no)	Batch size	Iters	Acc
1.	2 CL, 1 FC,	Relu	Max	Adam	No	100	7000	83.42
2.	2 CL, 1 FC,	Relu	Avg	Adam	Yes	256	4000	88.45
3.	2 CL, 1 FC,	Relu	Avg	Adam	Yes	256	5500	88.62
4.	2 CL, 1 FC,	Relu	Avg	Adam	Yes	256	4500	89.32
5.	2 CL, 1 FC,	Relu	Avg	RMSprop	Yes	300	4000	88.35
6.	2 CL, 1 FC,	Relu	Max	RMSprop	No	100	7000	83.56

7.	2 CL, 1 FC,	Relu	Avg	RMSprop	Yes	300	8000	88.31
8.	2 CL, 1 FC,	Relu	Avg	RMSprop	Yes	100	3000	87.21
9.	2 CL, 1 FC,	Relu	Avg	Adamax	Yes	100	3000	85.22
10.	2 CL, 1 FC,	Relu	Avg	Adamax	Yes	512	4000	89.07

#### Experiment Conducted and Observations Made:

We want to build a model that will classify digits into one of the ten classes of clothes ,shoes etc.

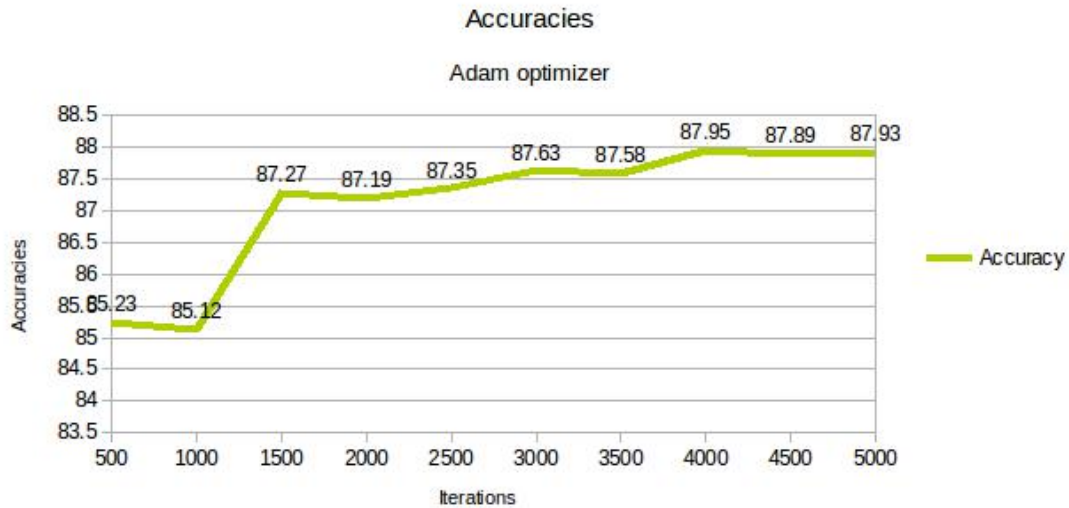
I preferred CNN with 2 convolution layer and one fully connected layer. The activation function I've used is Relu. Relu is defined as  $\text{Relu}(x) = x$  if  $x \geq 0$  and  $\text{Relu}(x) = 0$  otherwise.

I used two different CNN models with slightly different architecture – one without batch normalization and other with batch normalization.

Both of these models were run while varying the hyper parameters like batch\_size, number of iterations, optimizers, etc. For each optimizer, the model was run with varying batch\_size and number of iterations.

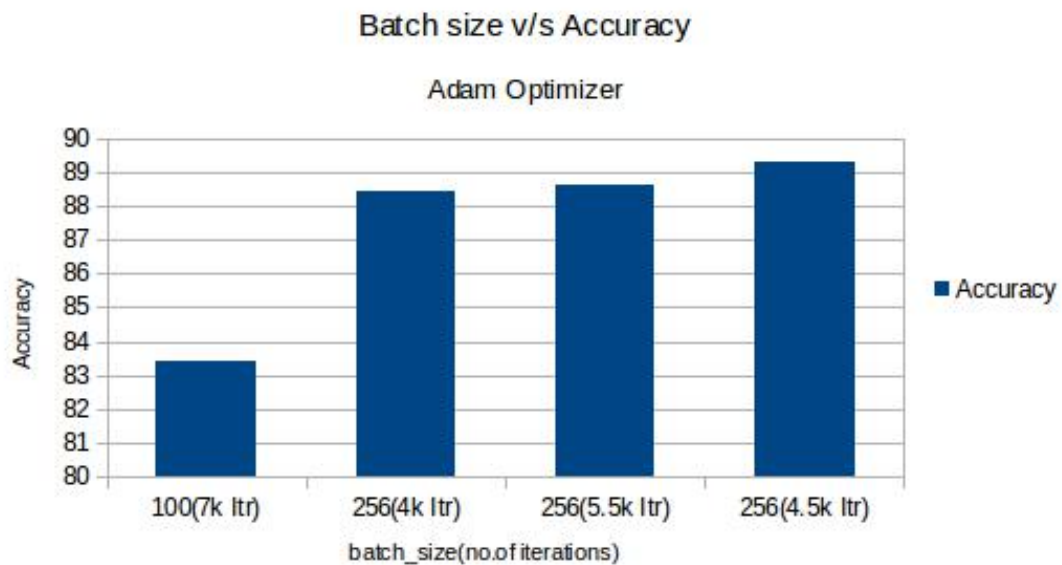
In the graph given below yes/no refers to whether batch normalization was used or not.

For example, we can observe the improvement in the accuracy for the model using Adam Optimizer, with batch\_size of 100 and running through 5.5 K iterations. From the figure shown below, we can observe that the accuracy initially improves faster than it did in the later iterations.

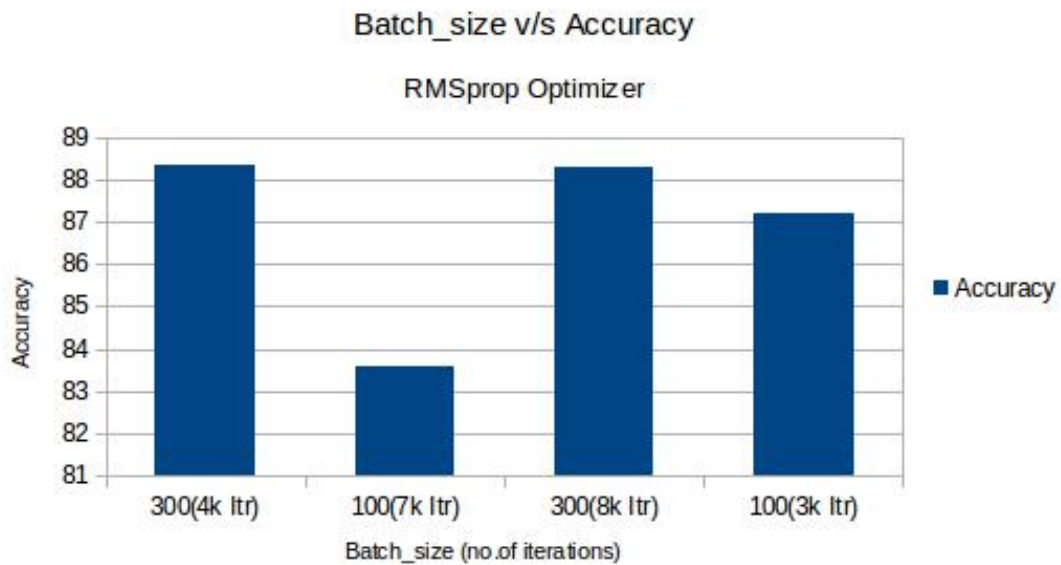


In this experiment we experimented with various optimizers like Adam, Adamax, RMSprop. Observation regarding each of these optimizer is given below.

#### ADAM Optimizer:



From the chart given above we can observe that as the batch size is increasing the accuracy too is increasing. This can be seen from the fact that as the batch size increases from 100 to 256, the accuracy increases from 83 to 88. However, this is not the case with number of iterations. We see from the figure that even though the number of iterations increased from 4k to 5.5 k, there is very less increase in the accuracy. Also we observe that when the iteration increases from 4.5k to 5.5k, there is a decrease in accuracy.

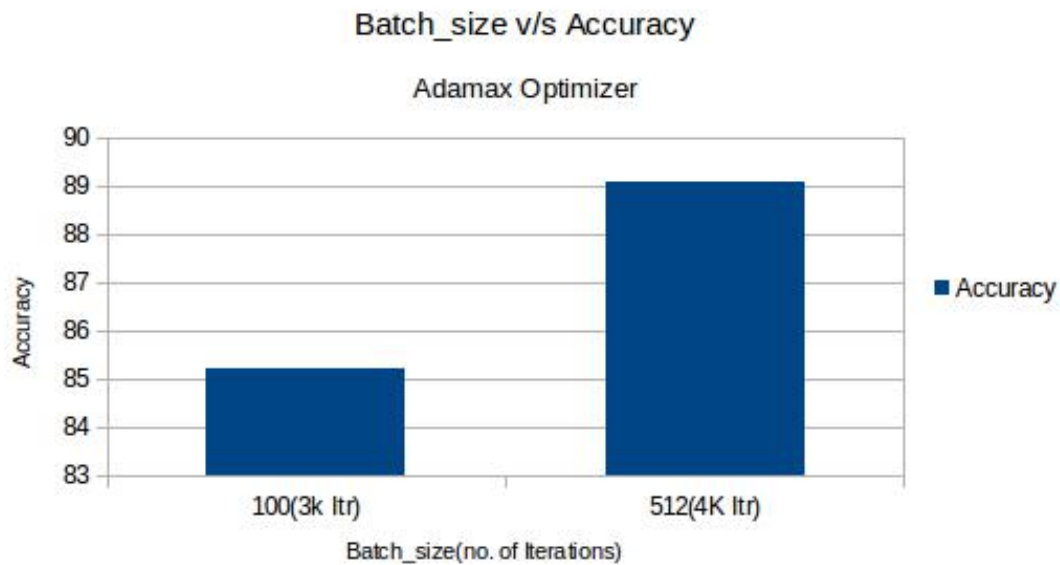


### **RMSprop Optimizer:**

From the chart given below we can observe that as the batch size is increasing the accuracy too is increasing. This can be seen from the fact that as the batch size increases from 100 to 300, the accuracy increases from 83 to 88. However, this is not the case with number of iterations. We see from the figure that keeping the batch\_size constant, even though the number of iterations increased from 4k to 8 k, there is slight decrease in the accuracy.

### **Adamax Optimizer:**

From the chart given below we can observe that as the batch size is increasing the accuracy too is increasing. This can be seen from the fact that as the batch size increases from 100 to 512, the accuracy increases from 85 to 89. Similar is the case with number of iterations. We see from the figure that when the number of iterations increased from 3k to 4k, there is great increase in the accuracy.



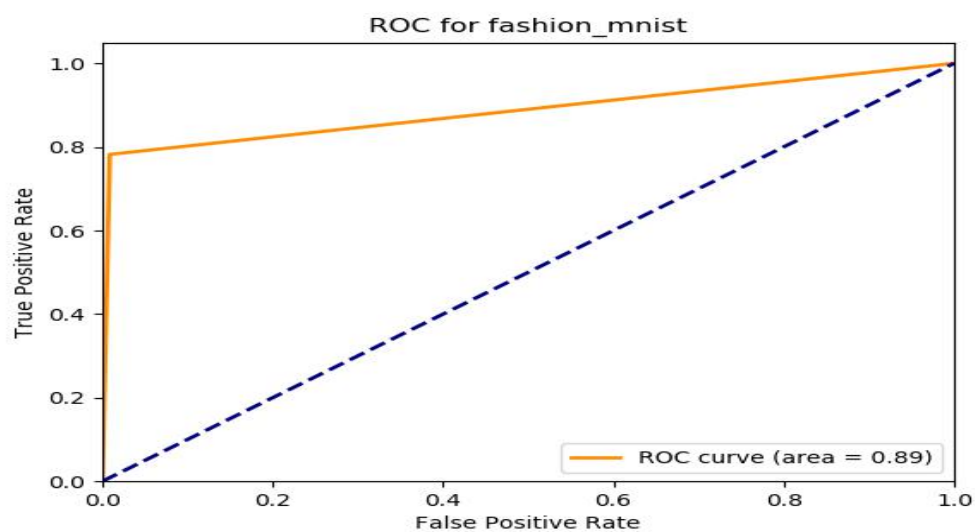
From the above graphs few things can be observed.

- ❖ Batch normalization gives better result compared to without batch normalization.
- ❖ As the batch size increases, the accuracy also increases.
- ❖ RMSprop does a better job in terms of accuracy.

#### ROC Plots for Different Models:

##### **Adam:**

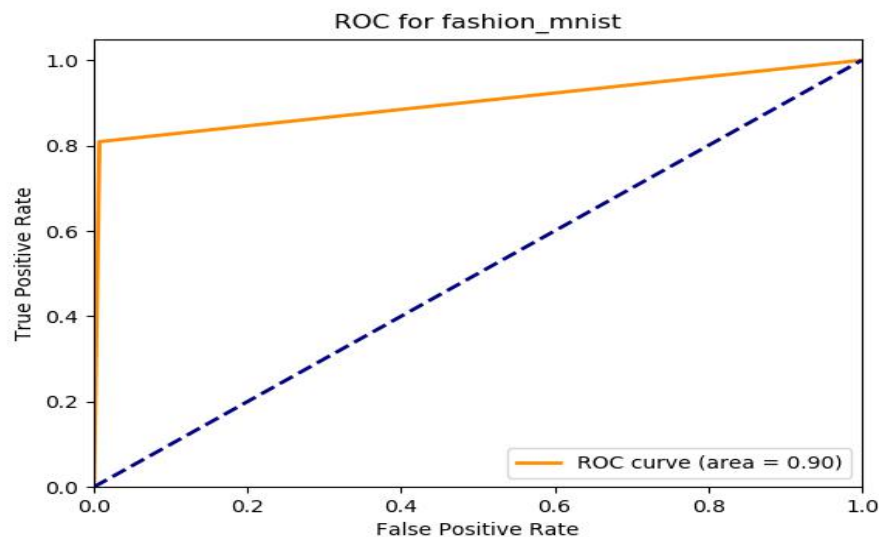
As we can see from the ROC curve that the area under the curve for the optimizer Adam is only 0.89, that indicates that the true positive rate is not up to the mark, because of which the area reduced.





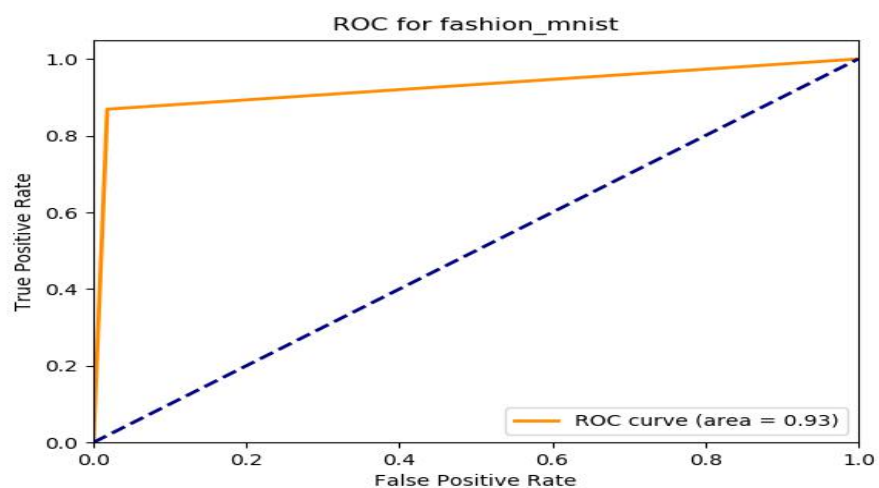
### Adamax:

In this model, we observe from the graph that the area under the curve has increased compared to the previous, as we note that the True positive rate increased from around 79 percent to 81.



### RMSprop:

In this final model, we conclude that it has maximum area under the curve with true positive rate at 84.



### Conclusion:

From above experiment and observations we can conclude the following:

- ❖ Accuracy is dependent on batch\_size, i.e. when batch\_size increases the accuracy too increases.
- ❖ With increase in number of iterations the accuracy tends to oscillate. Hence proper choice of number of iterations to run through is important.
- ❖ Apart from these, when the batch\_size is increased the model takes more time to train which is natural.

## **IMPLEMENTATION**

### **OF SELF NORMALIZING NEURAL NETWORK ON FASHION MNIST**

**( Work done by Saurav Rai (17558))**

#### **ABOUT SNN**

- **FNN(feed forward network) that perform well are typically shallow and, therefore cannot exploit many levels of abstract representations. Self-normalizing neural networks (SNNs) enable high-level abstract representations.**
- **While batch normalization requires explicit normalization, neuron activations of SNNs automatically converge towards zero mean and unit variance.**
- **The activation function of SNNs are "scaled exponential linear units" (SELUs), which induce self-normalizing properties.**
- **The activations close to zero mean and unit variance that are propagated through many network layers will converge towards zero mean and unit variance -- even under the presence of noise and perturbations.**

- This convergence property of SNNs allows to
  - (1) train deep networks with many layers,
  - (2) employ strong regularization, and
  - (3) to make learning highly robust.
- I have implemented SNN in python3 using Tensorflow (backend) .

### 3. **MODELS USED :**

#### MODEL :

Multilayer\_perceptron with 4 hidden layer ,SELU activation function , Adagrad Optimizer.

#### COST FUNCTION :

- Cross Entropy function

```
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
```

#### OPTIMIZER :

- AdagradOptimizer

```
tf.train.AdagradOptimizer(learning_rate=learning_rate).minimize(cost)
```

## ACTIVATION FUNCTION :

SELU activation function which is defined as

$$\text{selu}(x) = \lambda * x \text{ if } x > 0$$

$$= \lambda * \alpha * \exp(x) - \alpha \text{ if } x \leq 0$$

Here  $\alpha$  and  $\lambda$  are solved for in the equations resulting from find a fixed point  $\mu, v = g(\mu, v)$ .

## 5 .OBSERVATIONS:

### Parameters

Learning\_rate = 0.05

training\_epochs = 10

batch\_size = 100

display\_step = 1

### Networks parameters

n\_hidden\_1 = 784 #This is the 1st layer number of features

n\_hidden\_2 = 784 #This is the 2nd layer number of features

n\_input = 784 #FMNIST data input (image shape 28 \* 28)

**n\_classes = 10 #FMNIST total classes (0-9 digits)**

**Layer 1: selu and drop\_out selu or relu and drop\_out selu**

**Layer 2 : selu and drop\_out selu and drop\_out selu**

**• FOLLOWING IS THE OBSERVATIONS FROM SEVERAL TESTS.**

Sl.no	Model used	No of epochs	Optimizer	Batch_size	Accuracy%	Time(secs)
1	Multilayer Perceptron (SELU)	10	Adagrad Optimizer	50	89.59	139.51
2	Multilayer Perceptron (SELU)	20	Adagrad Optimizer	100	89.61	122.17
3	Multilayer Perceptron	10	Adadelat Optimizer	50	87.49	150.63

	(SELU)					
4	Multilayer Perceptron (SELU)	20	Adadelat Optimizer	100	88.58	303.89
5	<b>Multilayer Perceptron (RELU)</b>	<b>10</b>	<b>Adagrad Optimizer</b>	<b>50</b>	<b>81.21</b>	<b>141.21</b>
6	Multilayer Perceptron (SELU)	20	Proximal Adagrad Optimizer	50	89.56	345.56
7	<b>Multilayer Perceptron (SELU)</b>	<b>20</b>	<b>Proximal Adagrad Optimizer</b>	<b>50</b>	<b>90.33</b>	561.68
8	Logistic	100	GDM	-	84.12	135.21
9	K-logistic	100	GDM	-	81.23	160.12



10	Softmax	100	GDM	-	83.12	124.45
11	Softmax	200	GDM	-	85.11	245.12

### ***Observation :***

#### **1 . The Highest Accuracy :**

- **The optimizer chosen : Proximal Adagrad Optimizer , It allows the learning rate to adapt based on parameters.**
- **Batch size was taken as 50.**
- **The no of iterations was taken as 20.**
- **The activation function is SELUs which induce self-normalizing properties.**
- **The overall time taken was 561.68 seconds and**
- **The accuracy obtained is 90.33 % which is the highest accuracy among various combinations for self normalizing neural network.**

#### **2 . Using Logistic and K – Logistic regression :**

- **Logistic hypothesis model for this dataset.**

- Using 100 iterations and Gradient Descent optimizer
- Time taken : 135.21 seconds . The accuracy obtained : 84.12 %.
- Implemented K - logistic hypothesis model for this dataset , using 100 iterations and Gradient Descent optimizer. The time taken is 160.12 seconds and the accuracy obtained : 81.23 %.

### **3 . Using Softmax regression :**

- Implemented using Softmax hypothesis model for the same dataset.
- Using Gradient Descent Optimizer and 100 and 200 iterations.
- The accuracy obtained : 83.12 % and 85.11 % respectively.

### **4. Using Multilayer Perceptron :**

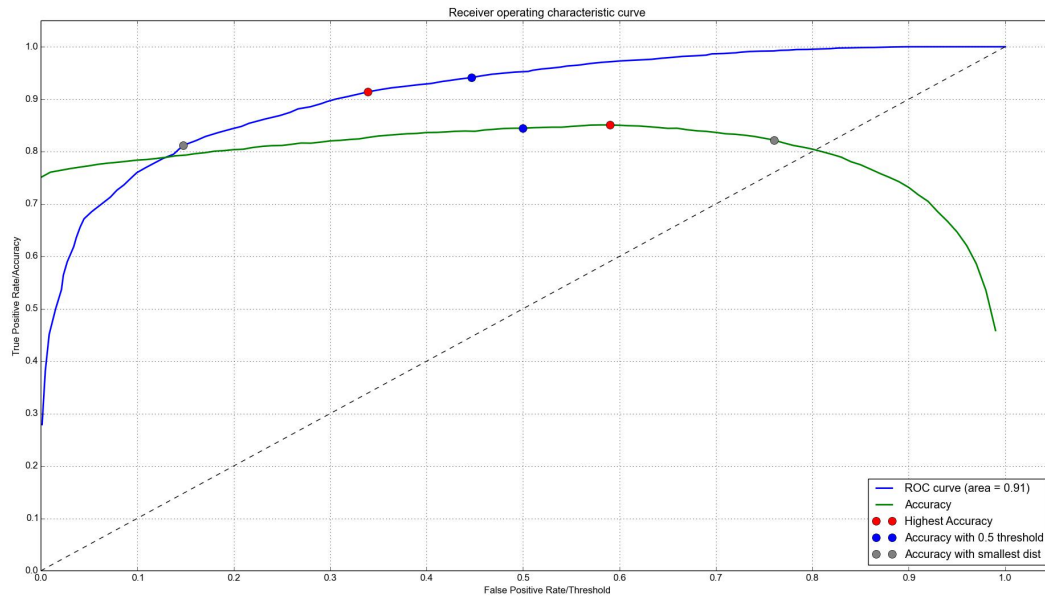
- Implemented using Multilayer Perceptron model and SELU activation function for the same dataset.
- Trained with different number of iterations and batch size with Adagrad and Adadelta optimizers .
- The average accuracy obtained is around 87 %.

## **5. Using ROC Curve :**

- After training, the network showed 94.4% accuracy. As the classes do not contain an equal amount of data, hence we can look at the ROC curve.

- I plotted here the accuracy of the network vs. the threshold. So the x and y axes have to meanings here: for the blue curve, the x and y axes are the false positive rate and true positive rate, respectively. For the green curve, the x axis is the threshold and the y axis the accuracy of the network.

- The blue points indicate the threshold/accuracy (FPR/TPR) if the threshold is chosen to be  $T1=0.5$ . I also marked the point, which labels the highest accuracy, namely 90.33% (red). The grey point is the one you get, when you minimize the distance between the ROC curve and the point (0,1). It corresponds to an accuracy of 81.21%.



## 6 . INFERENCE:

### Reason for High accuracy :

- The activation function chosen is SELU which induce self-normalizing properties by itself so the weight difference is always near 0.

- The optimizer Proximal Adagrad Optimizer allows the learning rate to adapt based on parameters.

It performs larger updates for infrequent parameters and smaller updates for frequent one. Because of this it is well suited for our Fmnist dataset.

**Another advantage is that it basically eliminates the need to tune the learning rate.**

**Reason for low accuracy :**

- **The activation function RELU does not have self-normalizing properties by itself . So it does not give a good accuracy for our model.**

**7. CONCLUSIONS :**

- **SNN's goal is to create neural networks in which, if the input of any layer is normally distributed, the output will automatically also be normally distributed.**
- **This is amazing because normalizing the output of layers is known to be a very efficient way to improve the performance of neural networks, but the current ways to do it (eg BatchNorm) basically involve weird hacks, while in SNN the normalization is an intrinsic part of the mathematics of the neural net.**
- **The implementation of SNNs is relatively simple.**

- **SNNs do have the property of keeping their output normalized even after several iterations of training is quite complicated.**
- **Future work will be to implement the model using more hidden layers and with other datasets or noisier dataset or strings that require segmentation.**

**SAIRAM**