# Convolution neural network using Tensor flow

Submitted by:  yalavarthi sri dharani [199804]

Masters in software engineering and management

Hoschule Heilbronn

Submitted to: M.Sc.  kateryna Sergieieva

## Table of contents

# ABSTRACT

In this paper I propose to you convolution neural network as a feasible method for quality acquire learning. Deep convolution network can be applied for common domain with success. I have build a simple convolution neural network with the proposed parameter and train it with MNIST Hand written numbers data images dataset by using different optimizers. And through Loss function I have tested the accuracy for epoch. Through the loss function we can identify the accuracy of different optimizers i.e., how accurately CNN can process an image using the high accurate resulted optimizer.

# INTRODUCTION:

Feature learning by using extreme neural networks has newly been applicable to a variety of computer eyesight and categorization complications, and manifested successfully in many domains. The classification precision over several benchmark vision data sets, similarly with an enormous number of trails per class, has been enhanced over the Deep learning approaches are depended on general presumptions that disregard the characteristics of the given real data. In connection to the procedures, deep learning techniques assist to move away from hand-crafted features, directly from the data. Convolutional neural networks are based on this scheme. Generally large feature learning networks such as convolutional neural nets have hundreds or thousands of parameters, which needs vast data sets for training. In real-world applications, not all the gathered big data are good sample sets of the target classes. So, the question whether features can be efficiently learned for classification of such data sets is prominent for such applications.

For building a convolution network there are several deep learning framework such as TensorFlow, Caffe ,Microsoft cognitive tool kit ,Pytorch, MX net ,Chainer, Keras etc. Out of these frameworks Pytorch and TensorFlow has gained a significant importance for this project I have adopted TensorFlow deep learning framework for building convolution neural network.

When compared to other deep learning frameworks TensorFlow is one of the best and have been adopted by many organizations (i e, airbus, twitter etc). Because of its extreme pliable system architecture. TensorFlow has a wide variety of proficiencies such as natural language processing, text summarization, and speech/image/hand writing recognition etc. TensorFlow can be extended to other models and data type.

To train and test this deep learning framework, we are using MNIST data sets which are a large data base of hand written digit that are commonly used for various image processing systems. TensorFlow has a wide range of optimizers, for this project I am using Adam, Adadelta, stochastic gradient descent for epochs and accuracies.

# INSTALLATION AND SETUP:

- First install python 2.7.6
- Install TensorFlow for Deep learning Framework
- install packages and Dependent libraries
- Download MNIST packages from following link: http://yann.lecun.com/exdb/mnist/
- In terminal run following command to run each optimizer:
    - For Adam optimizer: $python train_adam.py
    - For AdaDelta: $python train_adadelta.py
    - For SGD: $python train_sgd.py
- MNIST Dataset:
- Training Set:

    X : (55000, 784)

    Y : (55000, 10)

    - Test Set:

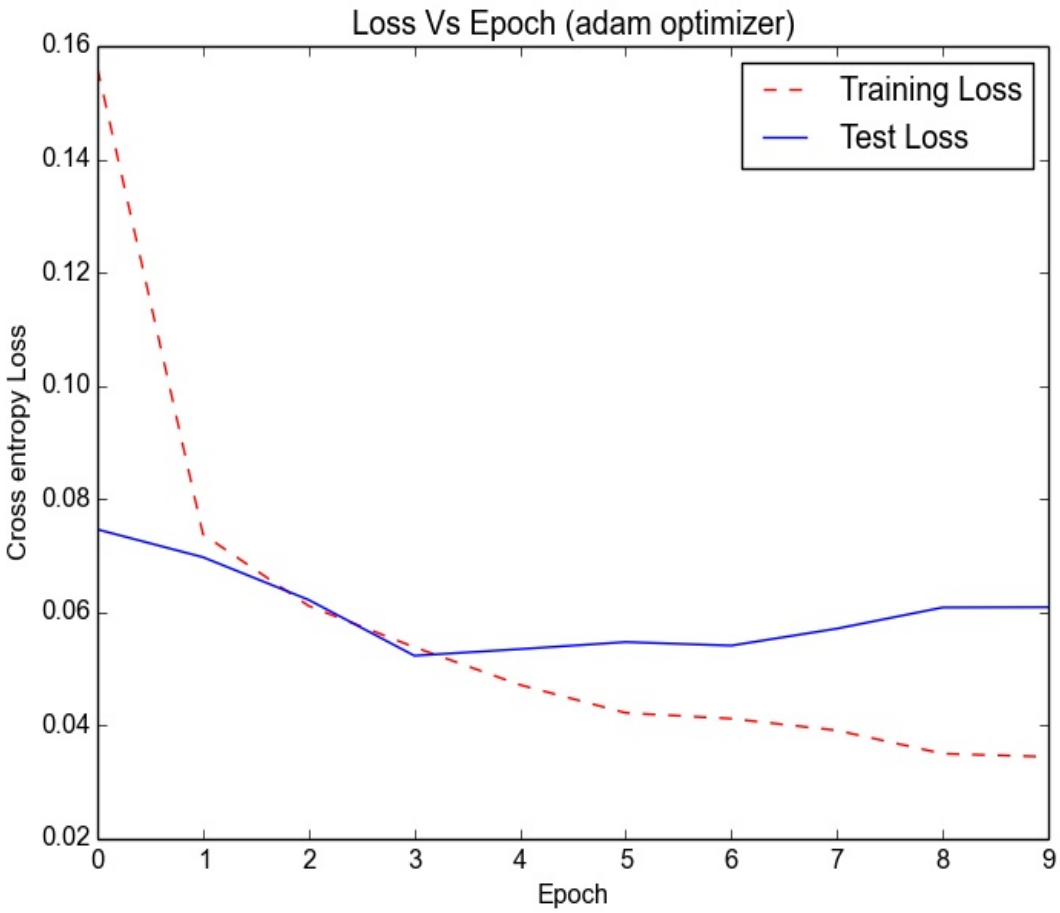
        X: (10000, 784)

        y : (10000, 10)

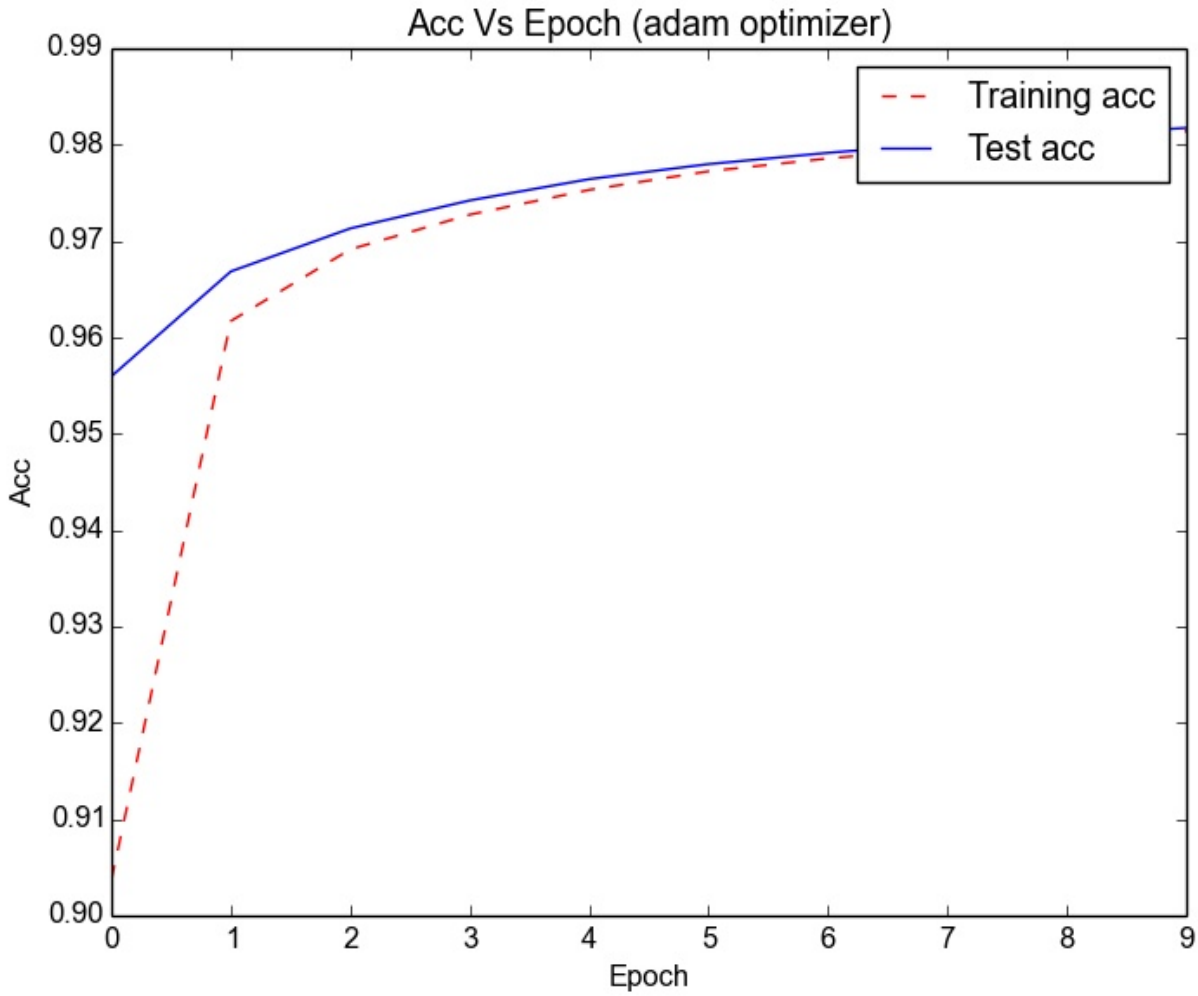
# Results

## 1. Adam optimizer

| Epoch | Train Loss | Test Loss |
|---|---|---|
| **0** | 0.15616 | 0.07465 |
| **1** | 0.07378 | 0.06974 |
| **2** | 0.06114 | 0.06221 |
| **3** | 0.05386 | 0.05235 |
| **4** | 0.04720 | 0.05352 |
| **5** | 0.04100 | 0.06423 |
| **6** | 0.04265 | 0.06521 |
| **7** | 0.04050 | 0.06590 |
| **8** | 0.03945 | 0.07108 |
| **9** | 0.03713 | 0.07005 |

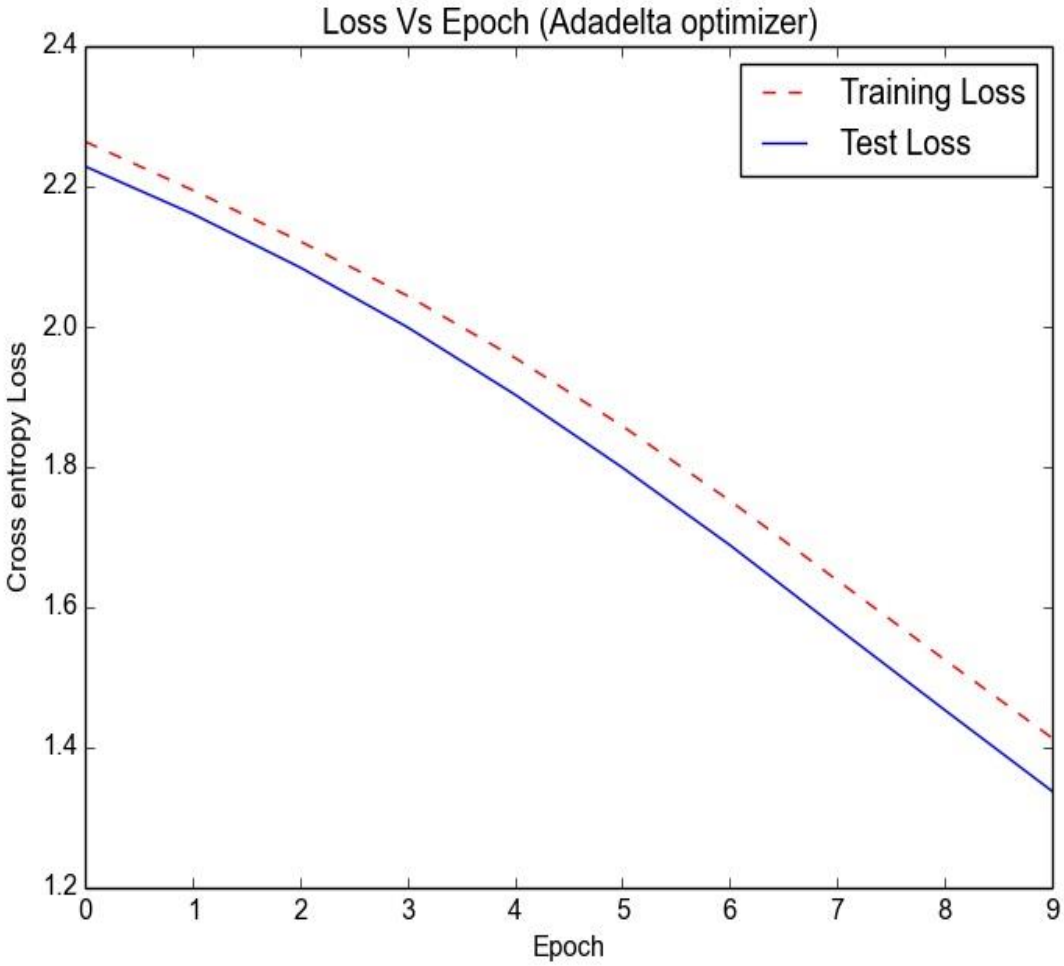

Loss Vs Epoch (adam optimizer)
Training Loss
Test Loss
Cross entropy Loss
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0
1
2
3
4
5
6
7
8
9
Epoch

| Epoch | Train Accuracy | Test Accuracy |
|---|---|---|
| 0 | 0.90369 | 0.95596 |
| 1 | 0.96170 | 0.96685 |
| 2 | 0.96914 | 0.97131 |
| 3 | 0.97274 | 0.97422 |
| 4 | 0.97532 | 0.97643 |
| 5 | 0.97557 | 0.97712 |
| 6 | 0.97604 | 0.97807 |
| 7 | 0.97630 | 0.97917 |
| 8 | 0.97648 | 0.97937 |
| 9 | 0.97669 | 0.97970 |

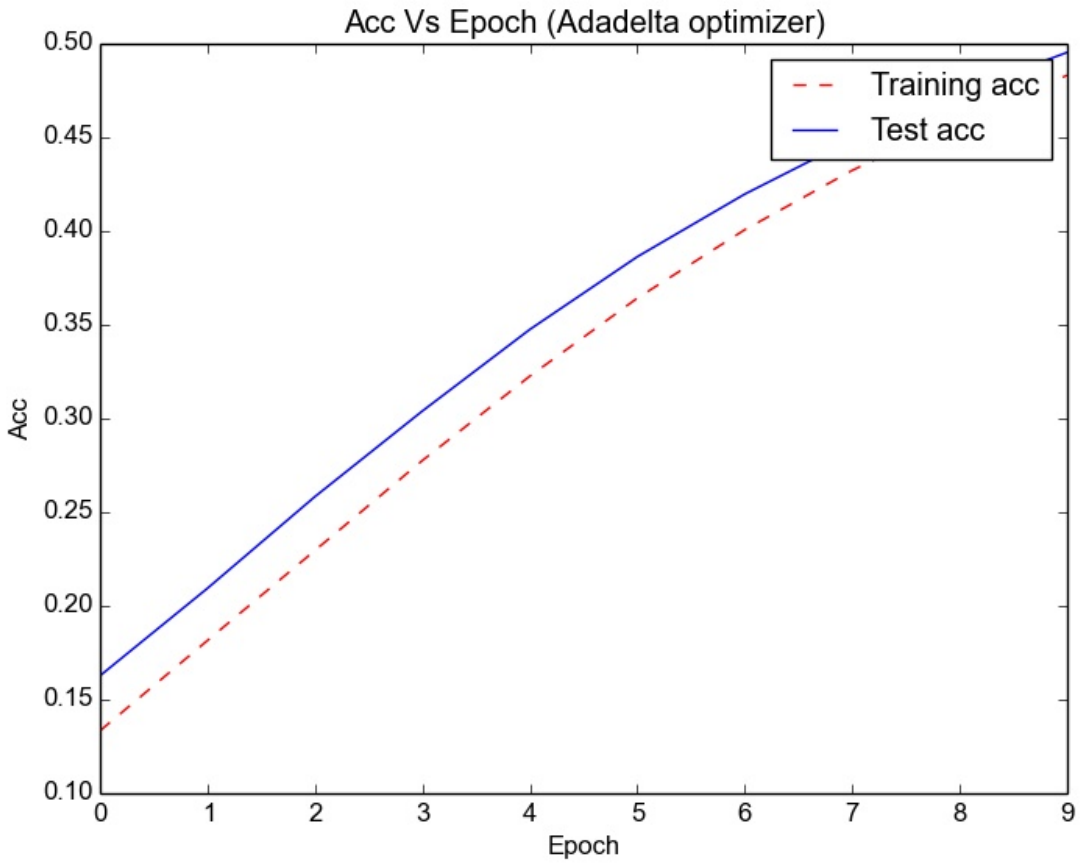

Acc Vs Epoch (adam optimizer)
Training acc
Test acc
Acc
Epoch
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.90
0
1
2
3
4
5
6
7
8
9

## 2. Adadelta Optimizer:

| Epoch | Train Loss | Test Loss |
|---|---|---|
| **0** | 2.26402 | 2.22873 |
| **1** | 0.13336 | 0.16273 |
| **2** | 2.19492 | 2.16103 |
| **3** | 2.04413 | 1.99939 |
| **4** | 1.95580 | 1.90332 |
| **5** | 1.85846 | 1.79923 |
| **6** | 1.75215 | 1.68875 |
| **7** | 1.63849 | 1.57054 |
| **8** | 1.52515 | 1.45352 |
| **9** | 1.41358 | 1.33771 |

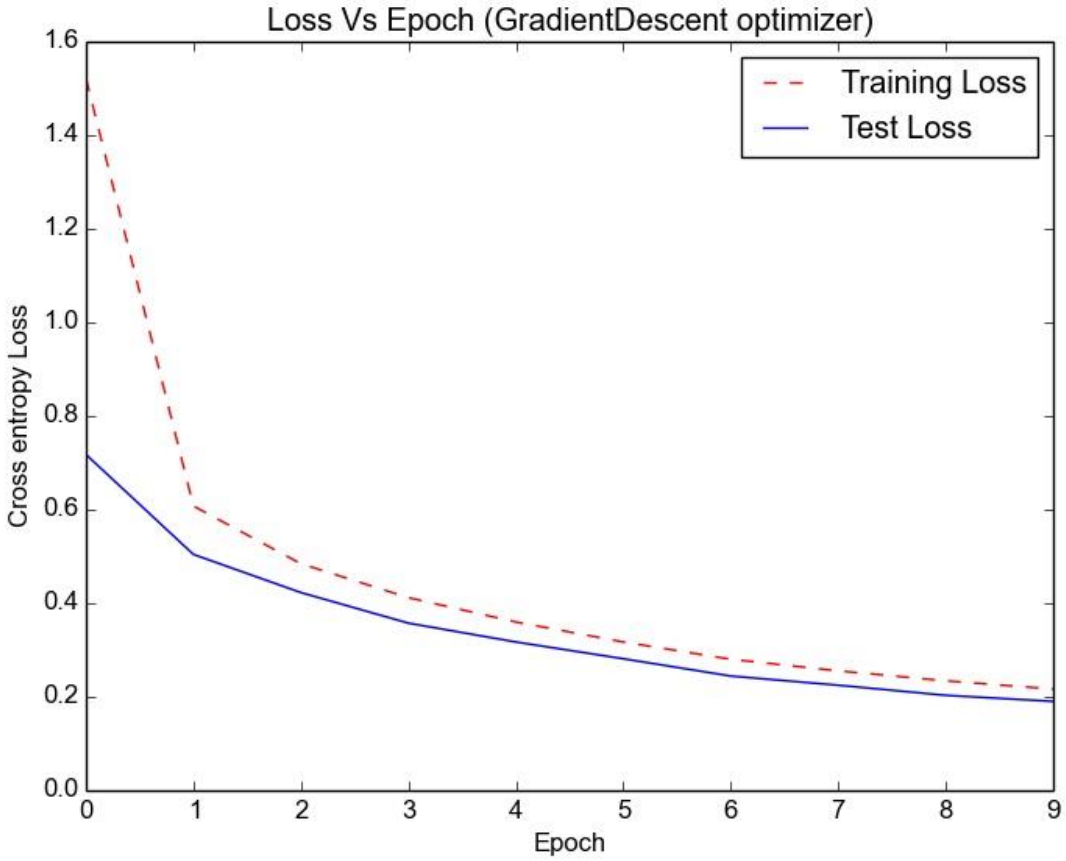

Loss Vs Epoch (Adadelta optimizer)
Training Loss
Test Loss
Cross entropy Loss
Epoch
2.4
2.2
2.0
1.8
1.6
1.4
1.2
0
1
2
3
4
5
6
7
8
9

| Epoch | Train Acc | Test Acc |
|---|---|---|
| **0** | 0.13336 | 0.16273 |
| **1** | 0.18152 | 0.20939 |
| **2** | 0.22958 | 0.25830 |
| **3** | 0.27759 | 0.30400 |
| **4** | 0.32261 | 0.34753 |
| **5** | 0.36421 | 0.38632 |
| **6** | 0.40072 | 0.41974 |
| **7** | 0.43241 | 0.44881 |
| **8** | 0.45963 | 0.47387 |
| **9** | 0.48313 | 0.49538 |

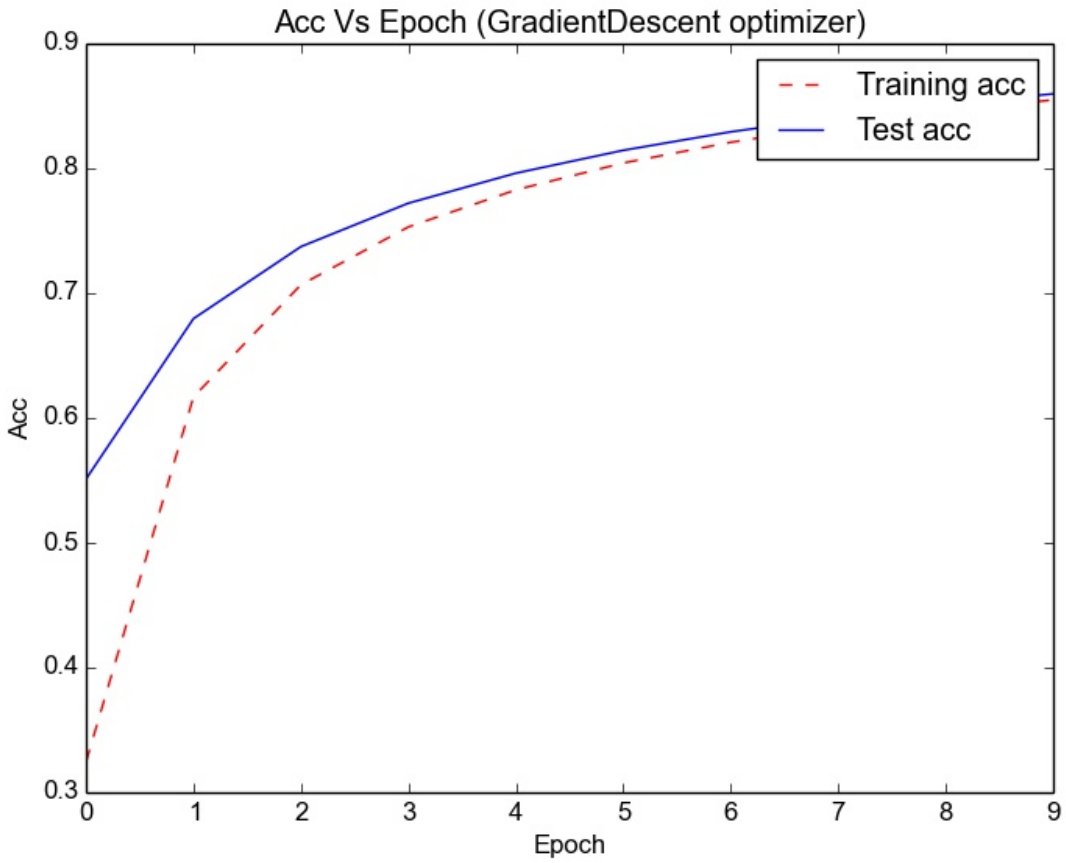


### 3. SGD Optimizer:

| Epoch | Train Loss | Test Loss |
|---|---|---|
| 0 | 1.52085 | 0.71766 |
| 1 | 0.60756 | 0.50417 |
| 2 | 0.48438 | 0.42269 |
| 3 | 0.41184 | 0.35742 |
| 4 | 0.35983 | 0.31713 |
| 5 | 0.31709 | 0.28130 |
| 6 | 0.28018 | 0.24435 |
| 7 | 0.25538 | 0.22472 |
| 8 | 0.23412 | 0.20315 |
| 9 | 0.21650 | 0.19021 |


Loss Vs Epoch (GradientDescent optimizer)
Training Loss
Test Loss
Cross entropy Loss
Epoch
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
7
8
9

| Epoch | Train Acc | Test Acc |
|---|---|---|
| 0 | 0.32436 | 0.55107 |
| 1 | 0.61687 | 0.67960 |
| 2 | 0.70673 | 0.73729 |
| 3 | 0.75308 | 0.77211 |
| 4 | 0.78279 | 0.79617 |
| 5 | 0.80434 | 0.81450 |
| 6 | 0.82090 | 0.82931 |
| 7 | 0.83451 | 0.84122 |
| 8 | 0.84564 | 0.85126 |
| 9 | 0.85492 | 0.85975 |


Acc Vs Epoch (GradientDescent optimizer)
Training acc
Test acc
Acc
Epoch
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
1
2
3
4
5
6
7
8
9

# CONCLUSION:

In this paper, I have presented a deep learning approach
for automatic detection of handwritten numbers images data sets . Due to the huge amount of image data, I have chosen a CNN to extract and recognize image features, and to automatically detect hand-written images. I compared various optimizers performance and evaluated that Adam Optimizer has more accuracy when compared with other optimizers.

# REFERENCES:


- https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners
- https://www.tensorflow.org/install/
- https://www.tensorflow.org/api_guides/python/train
- http://yann.lecun.com/exdb/mnist/
- https://dzone.com/articles/8-best-deep-learning-frameworks
- https://www.researchgate.net/publication/309152468_Deep_Convolutional_Neural_Networks_for_Detection_of_Rail_Surface_Defects
- https://data-flair.training/blogs/deep-learning-vs-machine-learning/