# Part A: Deep Convolutional Neural Network:

## Introduction:

Convolutional Neural Networks are deep, feed forward artificial neural networks used for analyzing visual imagery. The aim of this part of the assignment is to design a convolutional neural network to recognize the handwritten digits in the MNIST database. We have used python along with the theano library for this project. The MNIST database contains handwritten digit 28\*28 pixel images for this assignment, we have used the first 12000 records for training and the first 2000 records for testing. We use different algorithms to train the neural networks and evaluate their performance.

## Task 1: Design a two-layer convolutional neural network

Specifications:

• An Input layer of 28x28 dimensions

• A convolution layer 𝐶1 of 15 feature maps and filters of window size 9x9. A max pooling layer 𝑆1with a pooling window of size 2x2.

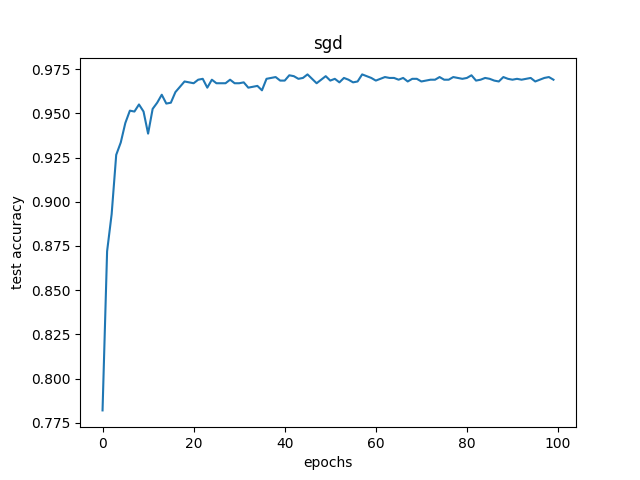
• A convolution layer 𝐶2 of 20 feature maps and filters of window size 5x5. A max pooling layer 𝑆2with a pooling window of size 2x2.

• A fully connected layer 𝐹3 of size 100.

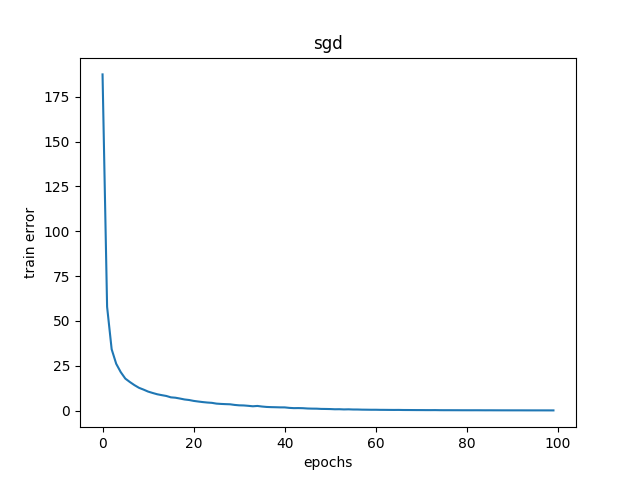
• A softmax layer 𝐹4 of size 10.

We trained the network using ReLu activation functions for neurons and mini batch gradient descent learning. We used the following parameters-batch size 128, learning rate 𝛼 = 0.05 and decay parameter 𝛽 = 10^−4.

### Test Accuracy against the number of epochs:



### Training error against the number of epochs:



As we can see from the graphs, as the convolutional neural network was trained, the training cost decreased while the test accuracy increased against the number of iterations.

### Feature Maps at the convolution and pooling layer:

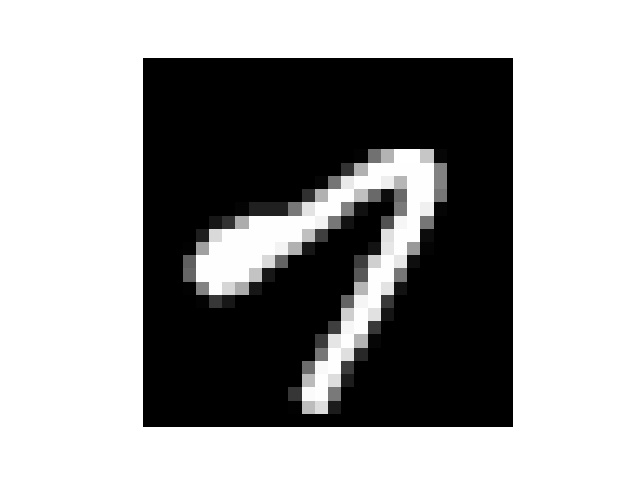
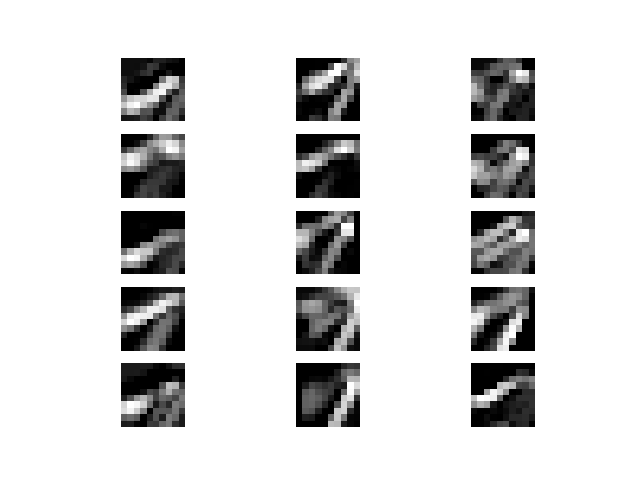
At the convolutional layer, each convolutional neuron processes data for its receptive fields. At the convolution layer feature maps are applied to the images and feature maps are produced. The feature map is the output of one filter applied to the previous layer. Each position results in an activation of a neuron and output is collected in the feature map.

The pooling layer helps to reduce the dimensions of the feature maps.

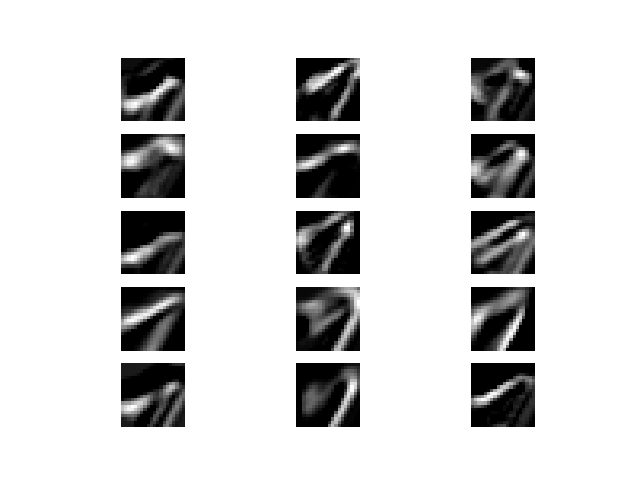
The images given below show the 15 and 20 feature maps for the convolution C1 and convolution C2. It also shows the feature maps at Pooling layer P1 and pooling layer P2.

## Feature Maps- Sample 1:

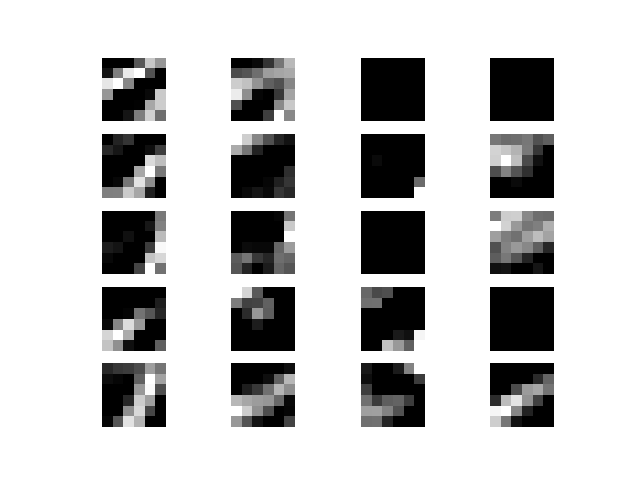
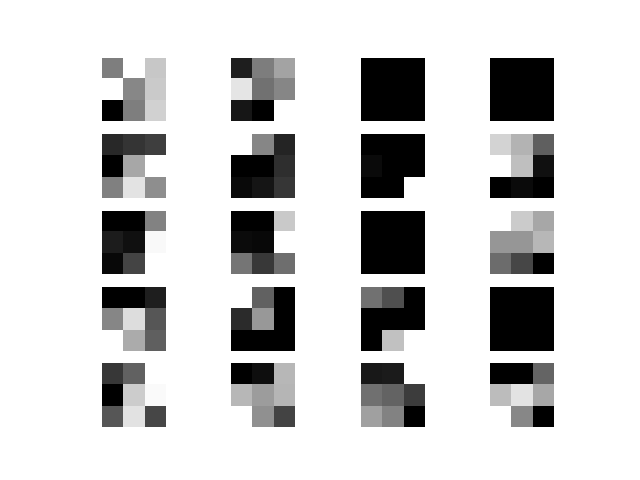
### Input Image:



Pooling P1



Convolution C1



Pooling P2

Convolution C2

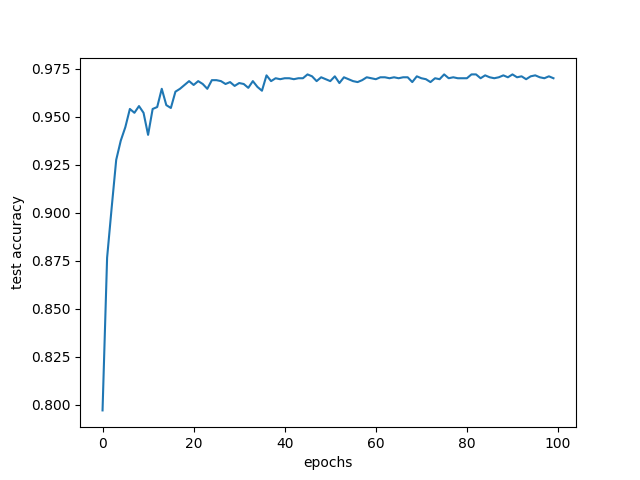
## Task 2: Repeat task 1 with momentum term:

Repeat part 1 by adding the momentum term to mini batch gradient descent learning with momentum parameter 𝛾 = 0.1

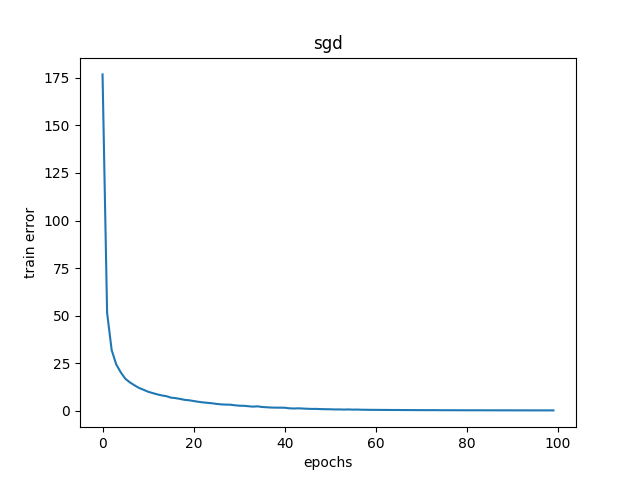
### Stochastic Gradient with Momentum:

The method of momentum is designed to accelerate learning, especially in the face of high curvature, small but consistent gradient or noisy gradient. The momentum algorithm accumulates an exponentially decaying moving average of past gradient and continues to move in their direction.

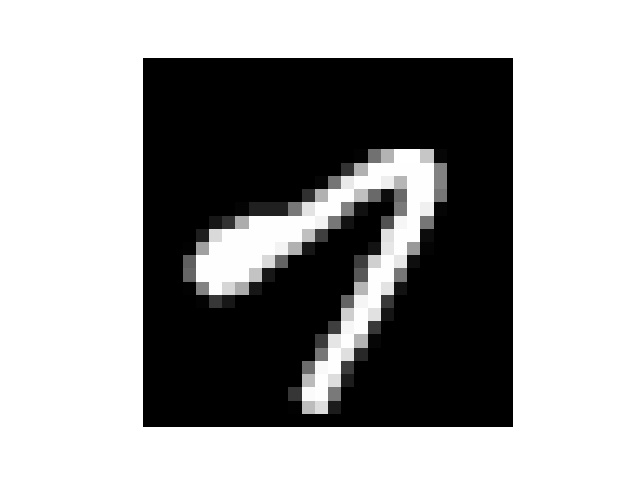
### Test Accuracy against the number of epochs:

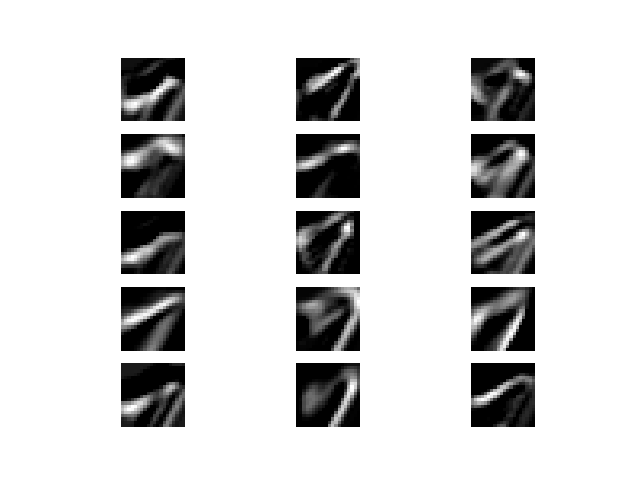
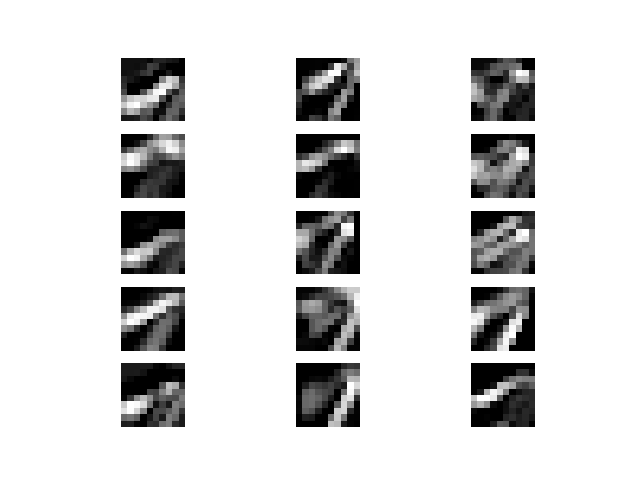
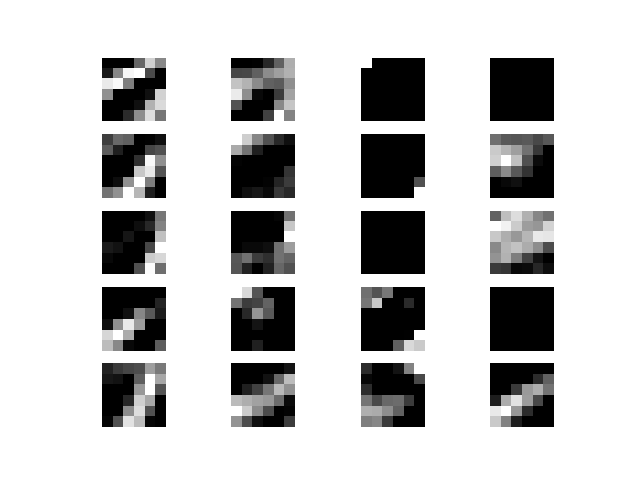
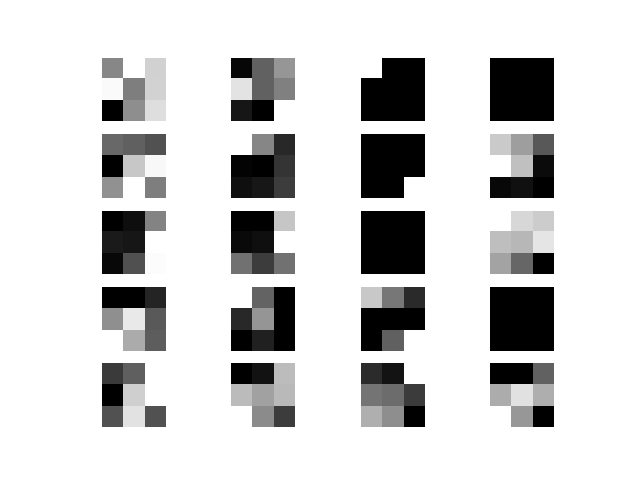


### Training error against the number of epochs:



## Sample 1:





Pooling P1

Convolution C1

Pooling P2

Convolution C2

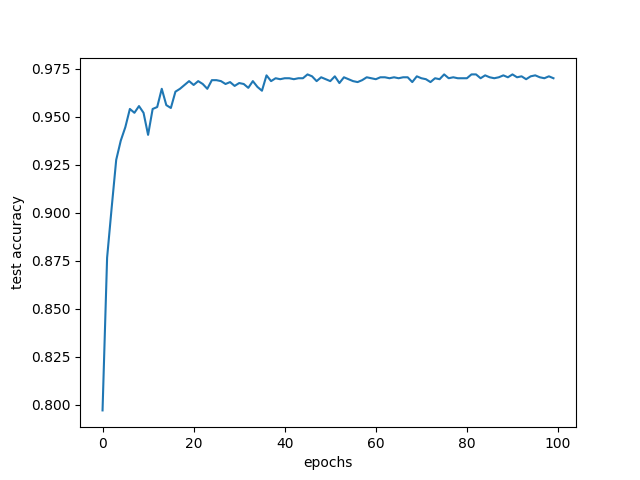
### Task 3: Repeat task 1 with RMSProp Algorithm:

Repeat part 1 by using RMSProp algorithm for learning. Use 𝛼 = 0.001, 𝛽 = 1𝑒 −4, 𝜌 = 0.9, and 𝜖 = 10−6 for RMSProp.

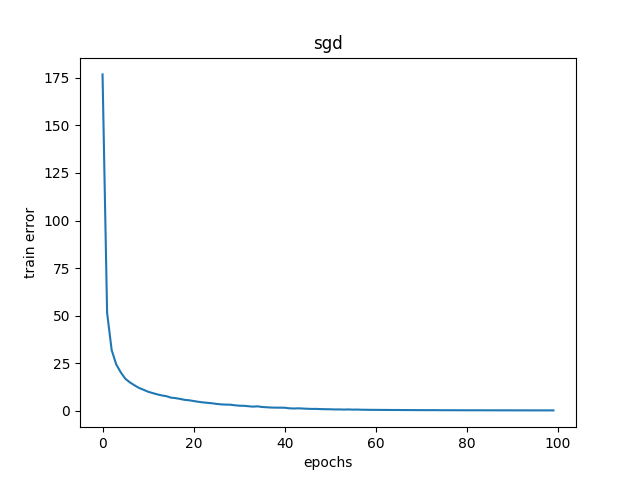
### RMSProp Algorithm:

Adaptive learning rates with annealing usually works with convex cost functions. RMSProp uses an exponentially decaying average to discard the history from extreme past so that it can converge rapidly after finding a convex region

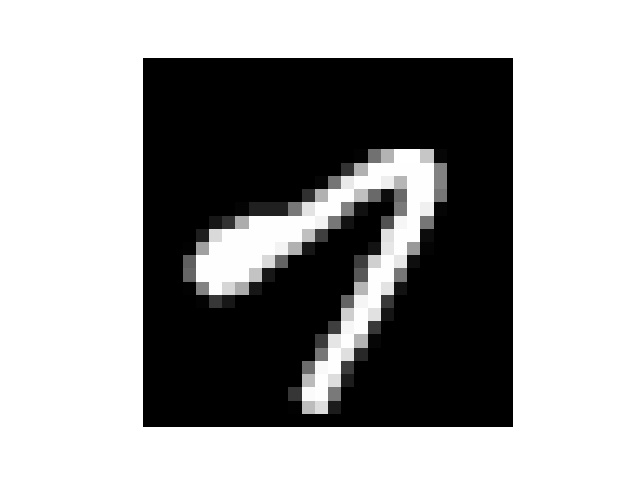
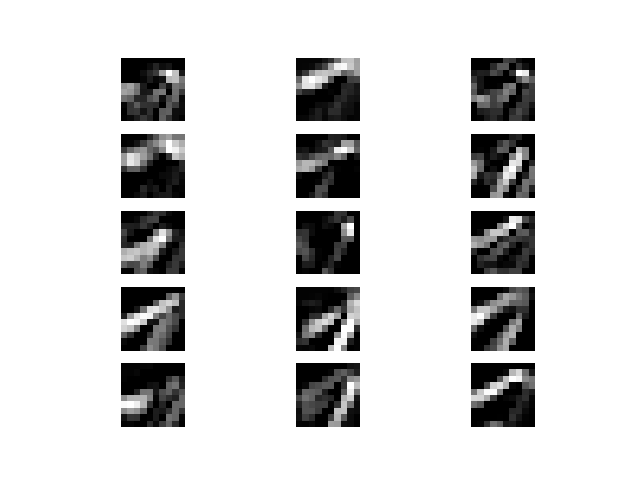
### Test Accuracy against the number of epochs:



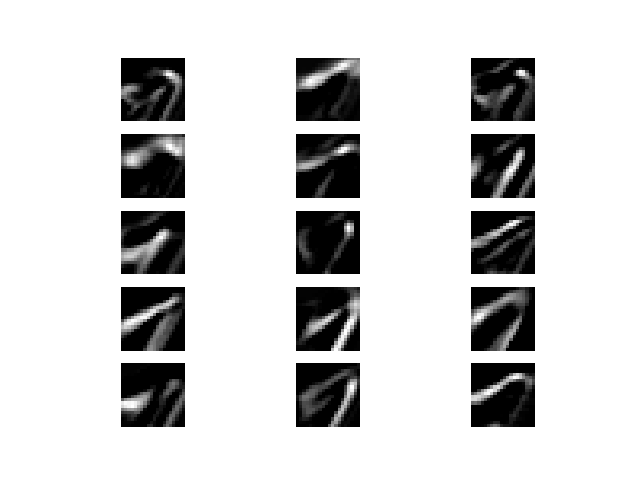
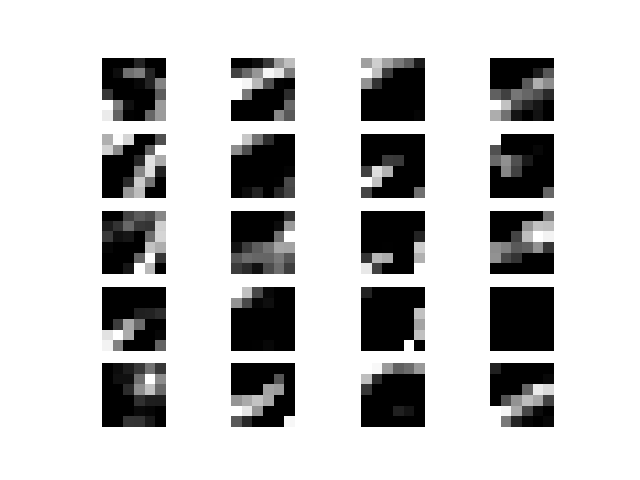
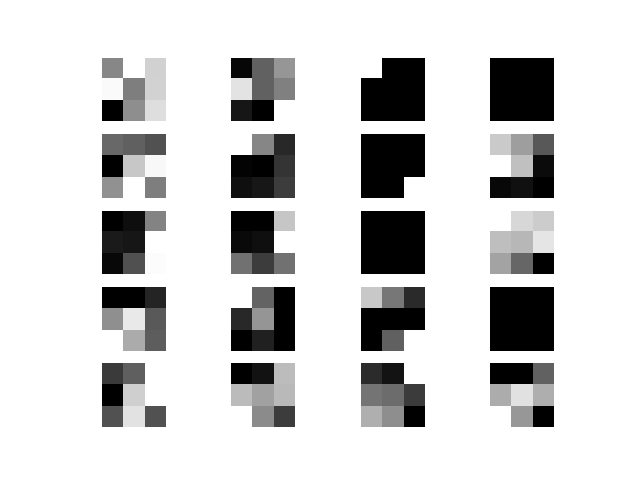
### Training error against the number of epochs:



## Sample 1:



Pooling P1



Pooling P2

Convolution C2

Convolution C1

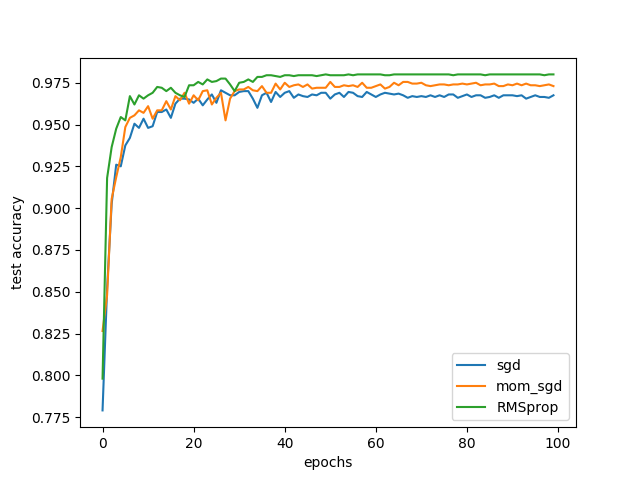
## Comparison of Algorithms:

As we can see in the diagrams below, that CNN with the RMSProp algorithm converges faster than the normal stochastic gradient algorithm and stochastic gradient with momentum.

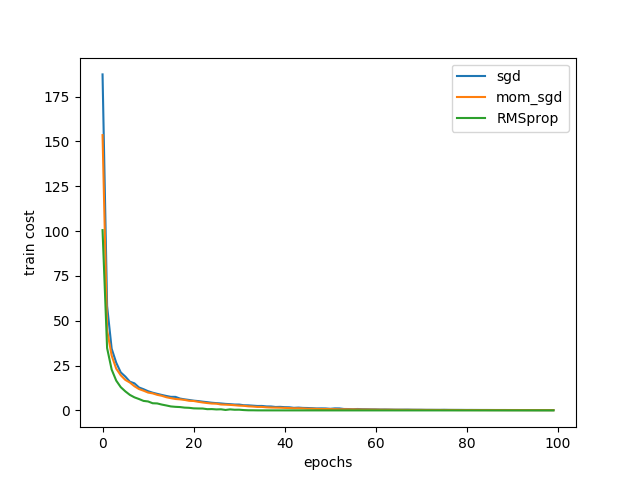
The training errors of RMSProp also seem to decrease at a higher rate than the other two.

I think that RMSProp gives the best performance.

### Test Accuracy against epochs:



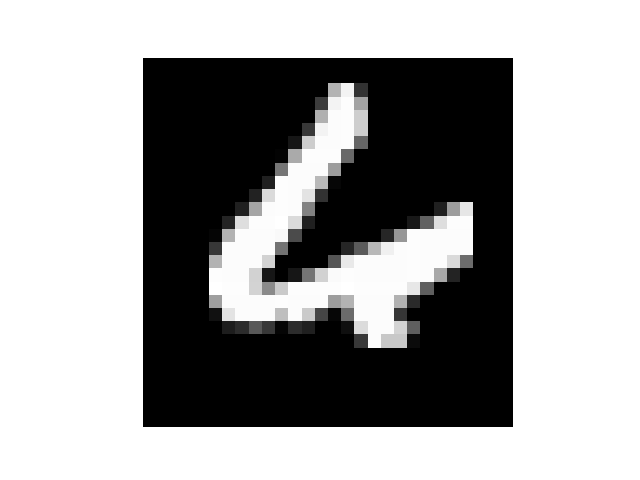
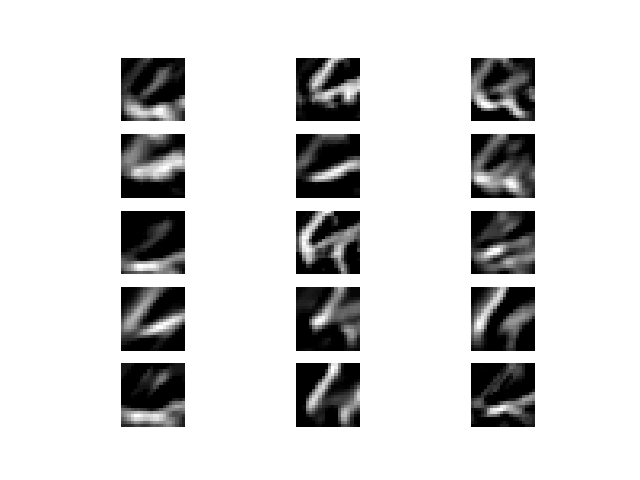
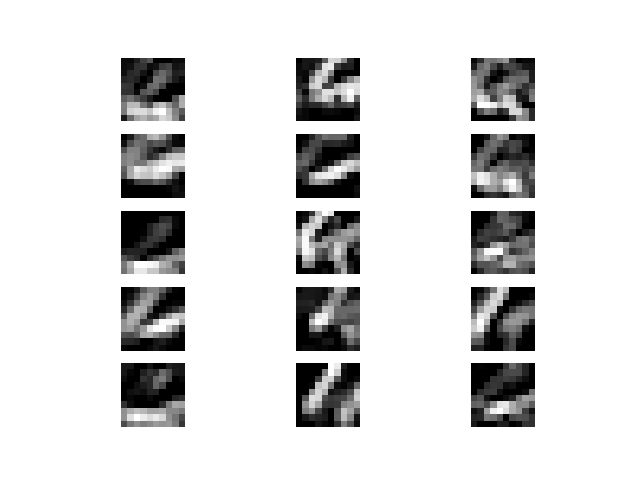
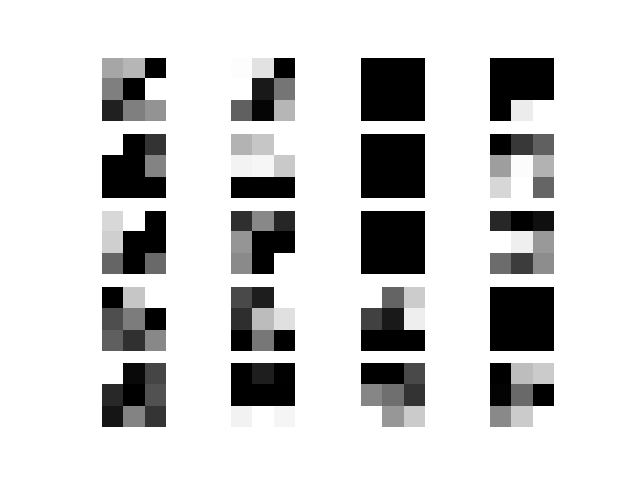
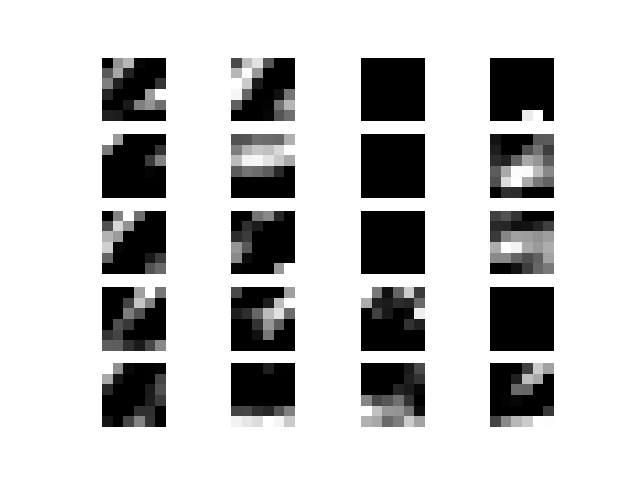
### Training Error against epochs:



Appendix A:

## Task 1: SGD

### Sample 2:



Convolution C2

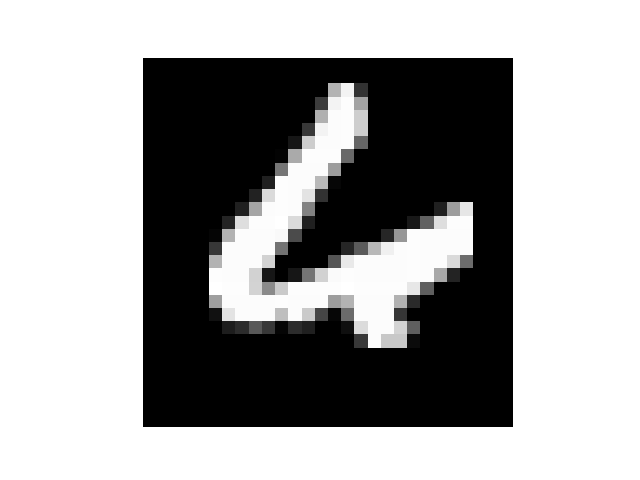
Pooling P2

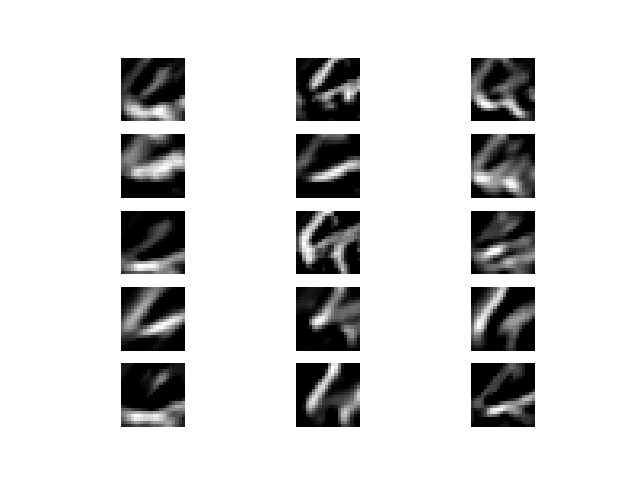
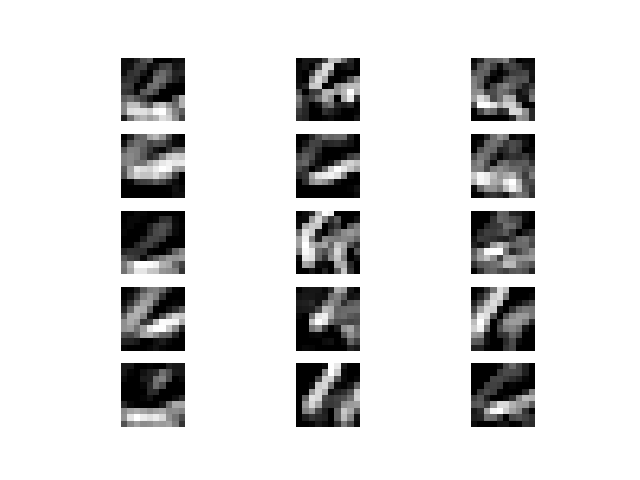
Pooling P1

Convolution C1

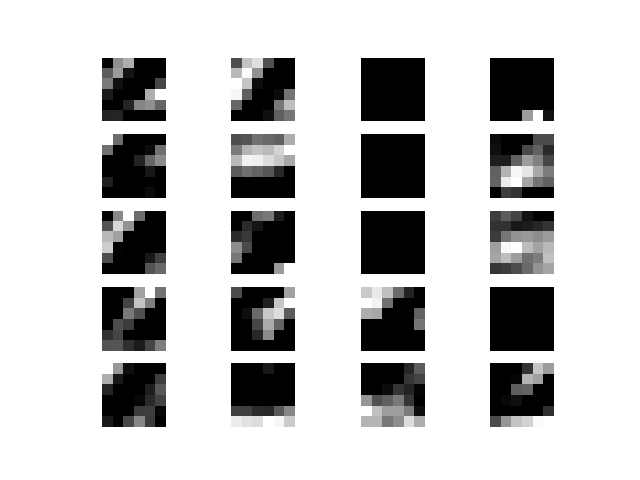
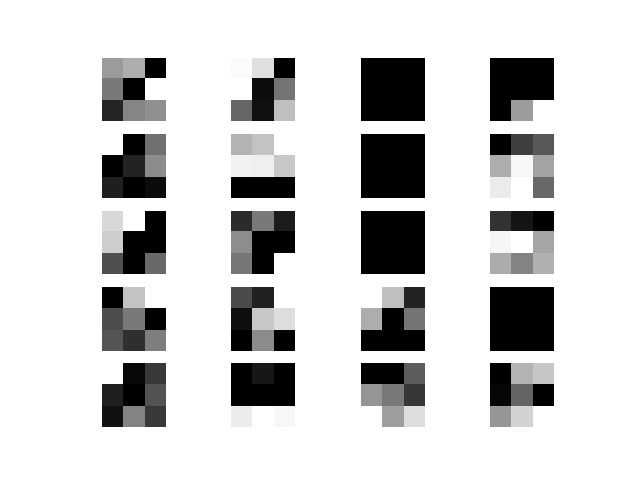
## Task 2: SGD with momentum

## Sample 2:





Convolution C1



Convolution C2

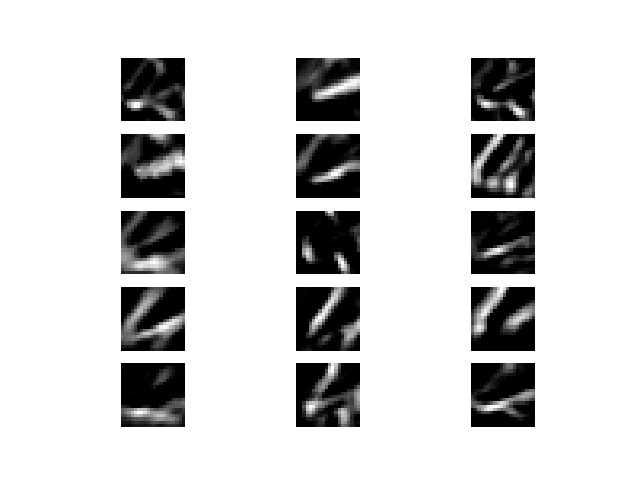
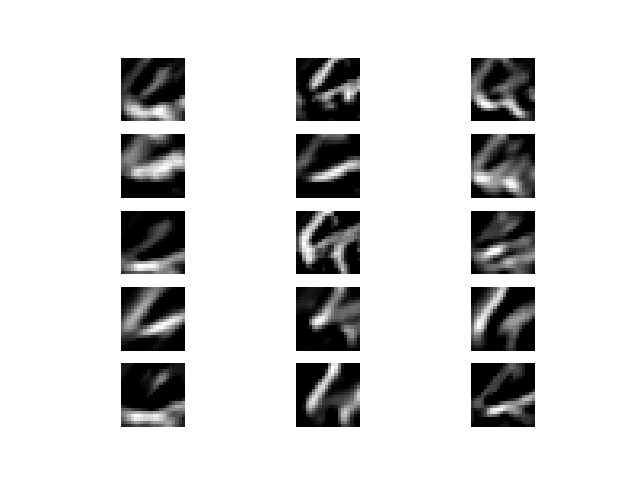
Pooling P2

Pooling P1

## Task 3: RMSProp

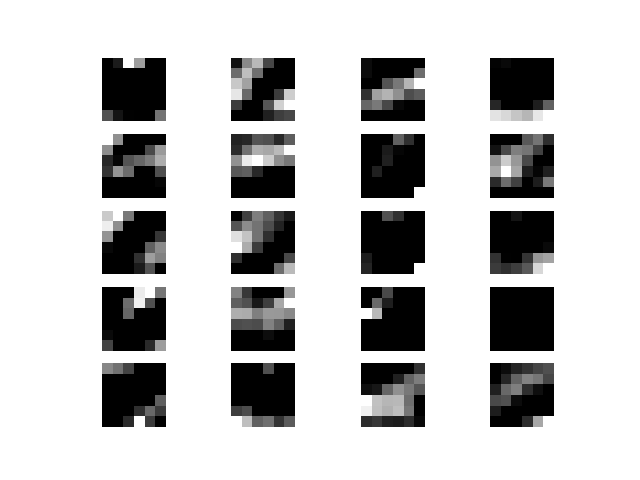
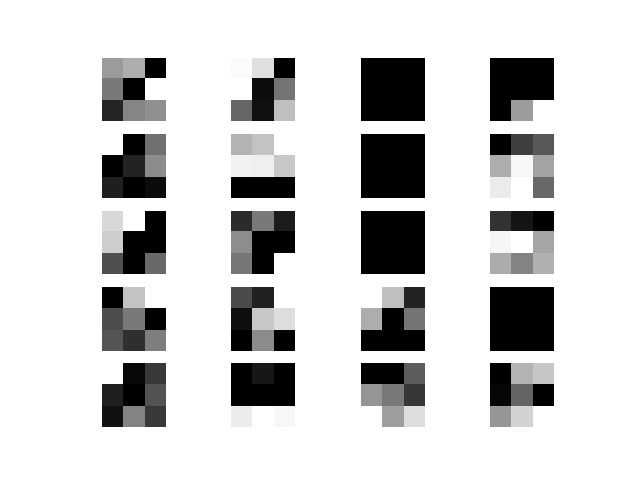
## Sample 2:

### 



Pooling P1

Convolution C1



Pooling P2

Convolution C2