EE5179: DEEP LEARNING IN IMAGING

ASSIGNMENT 4: AUTOENCODERS

REPORT

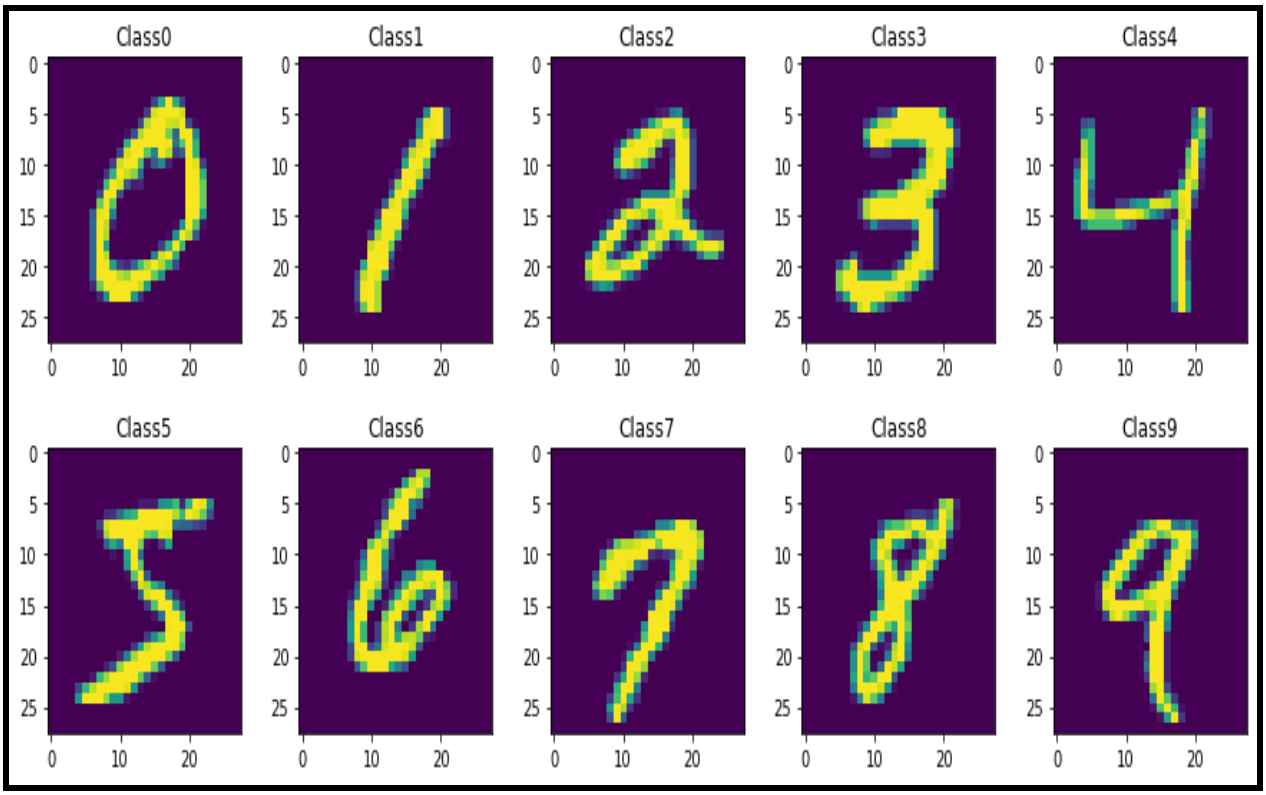
RAVEENA RAI (ED21S006)

Resources used:

1. Google Colab
2. MNIST Dataset of handwritten digits
3. Tutorials provided in the class of the course ‘Deep Learning for Imaging’
4. Online references.

About the dataset:

1. The dataset used in the algorithm is the MNIST Digits Recognition dataset.
2. It comprises handwritten digits, pre-processed to ensure that the digits are centered and size normalized.
3. The train set consists of 60,000 images, and the test set consists of 10,000 images.
4. The training dataset was divided into two sets: training (50000 images) and validation(10000 images).
5. The train data has been normalised before using it in the assignment.
6. The images have been flatten (28*28=784).


Class0
Class1
Class2
Class3
Class4
Class5
Class6
Class7
Class8
Class9

Q-1: COMPARING PCA AND AUTOENCODER:

A) PCA:

1. PCA was performed on the MNIST dataset and first 30 eigenvalues with their corresponding eigenvectors were used.
2. To observe the performance of the method, reconstruction was performed.

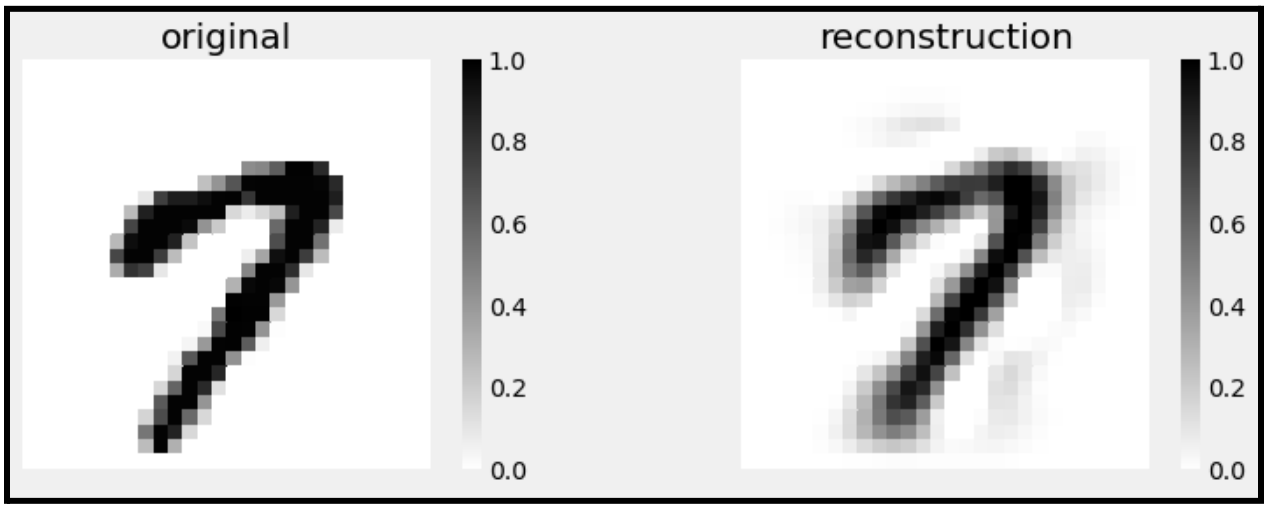


Fig: reconstruction


Original Image
30-Vector Reconstruction
Original-Reconstruction Difference

1. The figure shows the difference as obtained, between the original image and reconstructed image.
2. It can be seen that there is a lot of difference observed in the case of digits which have holes like 6, 0, 9, etc.

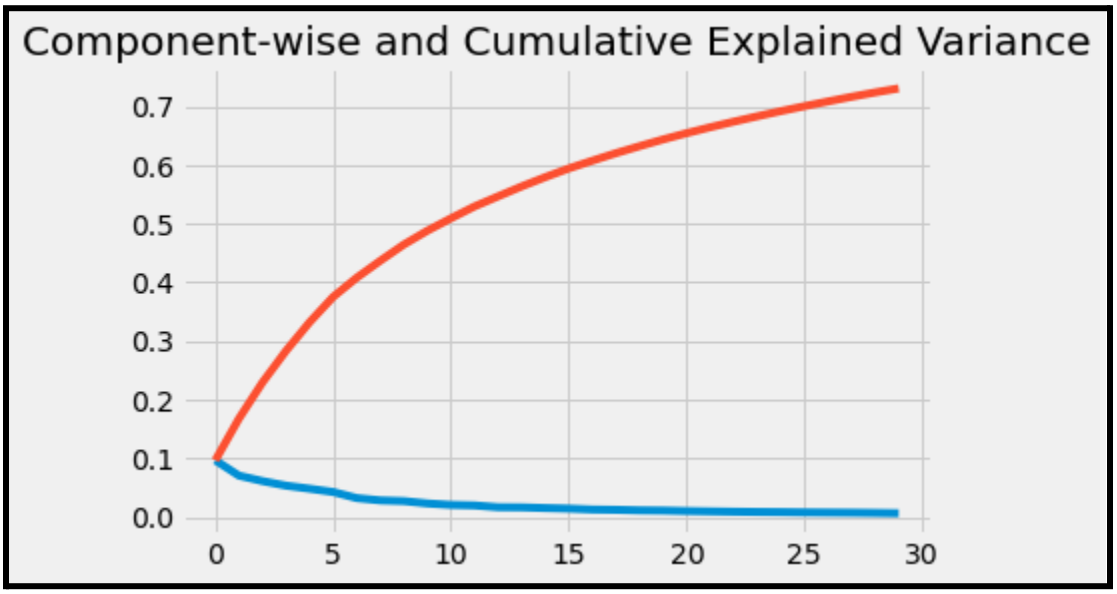


Fig: component- wise and cumulative explained variance

3. Looking at this plot, it seems like the first and second components add a lot of information about the variables to the model, but every component after that is not particularly worthwhile.
4. If we were using PCA as a processing step ourselves, we might want to cut off at n=2.

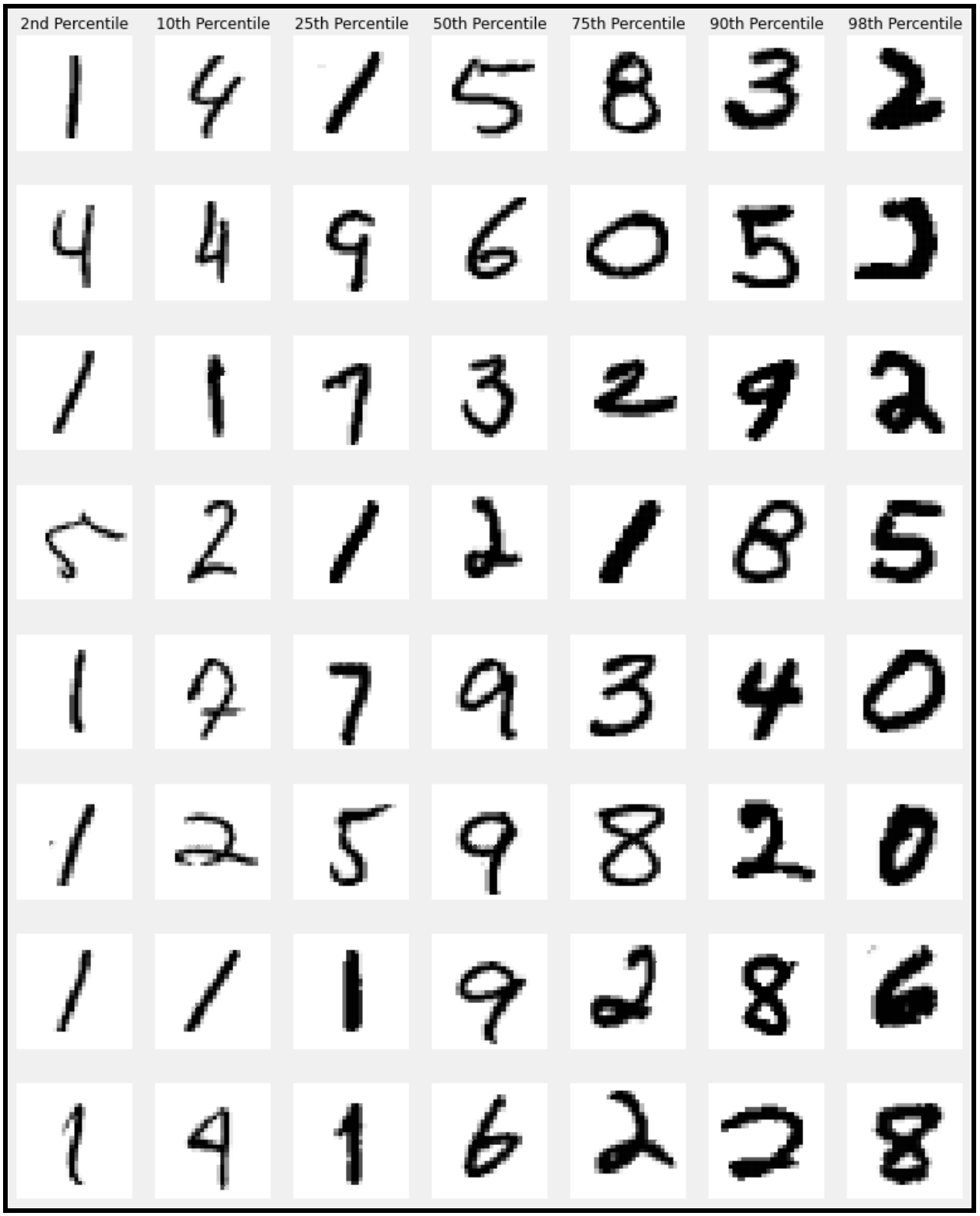


fig: Returns the data, which is the q-quartile fit for the given vector.

5. The 0th, 25th, 50th, 75th, and 100th percentile of the dataset. Records (quantiles) very close to the edges of the usefulness of a variable are often very informative as to what that variable "does".
6. This is true of ordinary variables, and all the more true of our computed ones, which we need all the help we can get interpreting.

Autoencoders:

1. A simple autoencoder was constructed as follows:

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_1 (Flatten)         (None, 784)               0

 dense_7 (Dense)             (None, 512)               401920

 dense_8 (Dense)             (None, 256)               131328

 dense_9 (Dense)             (None, 128)               32896

 dense_10 (Dense)            (None, 30)                3870

 dense_11 (Dense)            (None, 128)               3968

 dense_12 (Dense)            (None, 256)               33024

 dense_13 (Dense)            (None, 784)               201488

 reshape_1 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 808,494
Trainable params: 808,494
Non-trainable params: 0
_________________________________________________________________
```

2. reLu was used as an activation function.
3. The train and validation plots were obtained.
4. Batch size was selected arbitrarily as 32, and the model was trained for 10 epochs with mse loss and learning rate as 0.001 (arbitrary).
5. validation accuracy obtained was 81.43 %.

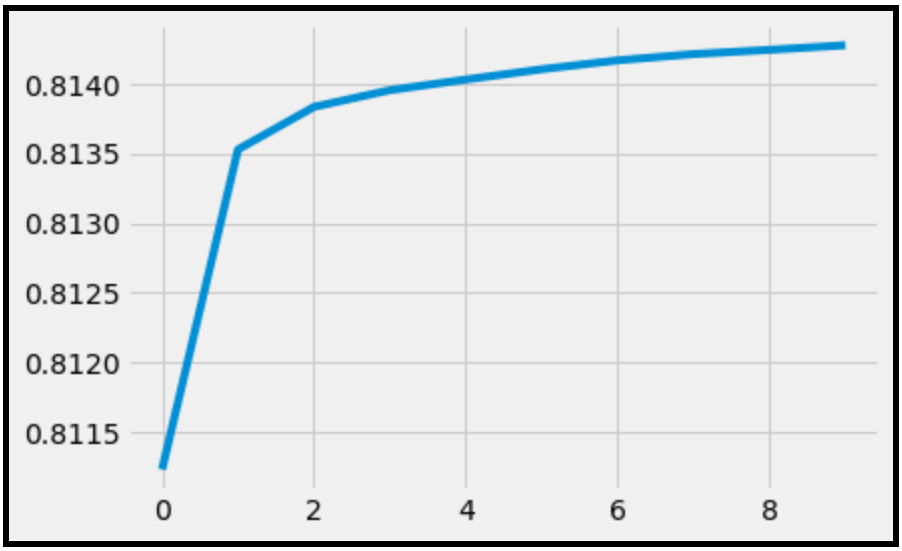


Fig: training accuracy

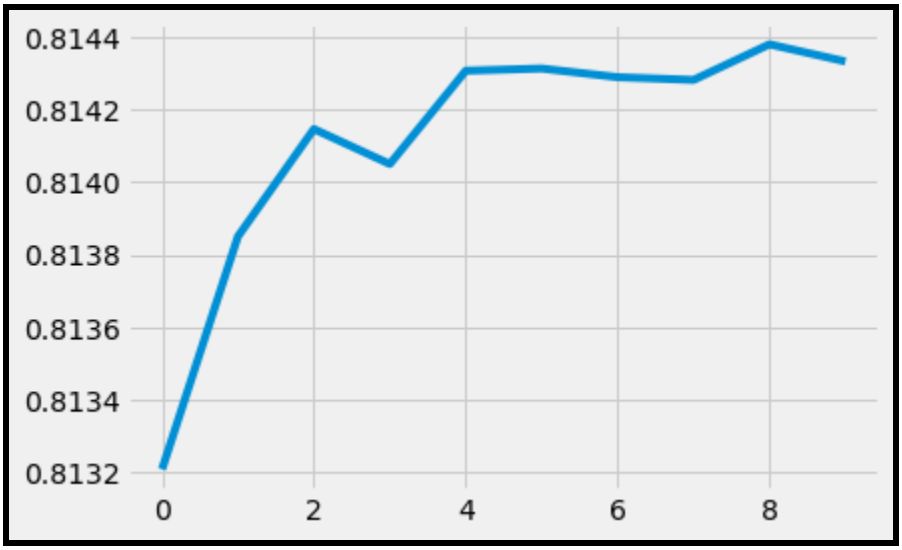


Fig: validation accuracy

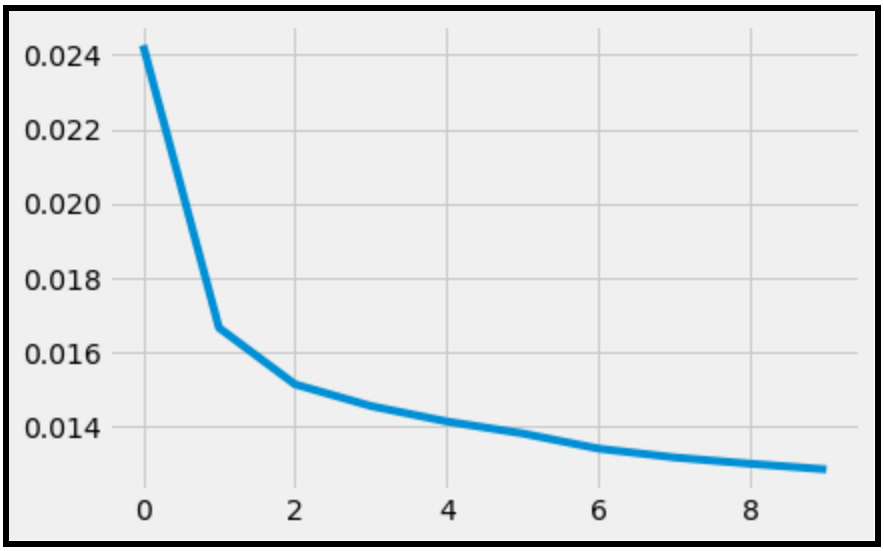


Fig: training loss

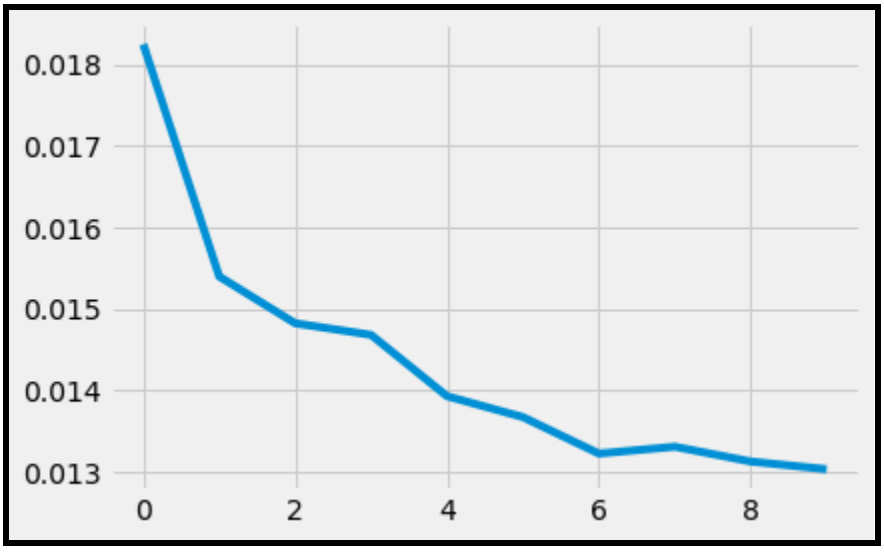


Fig: validation loss

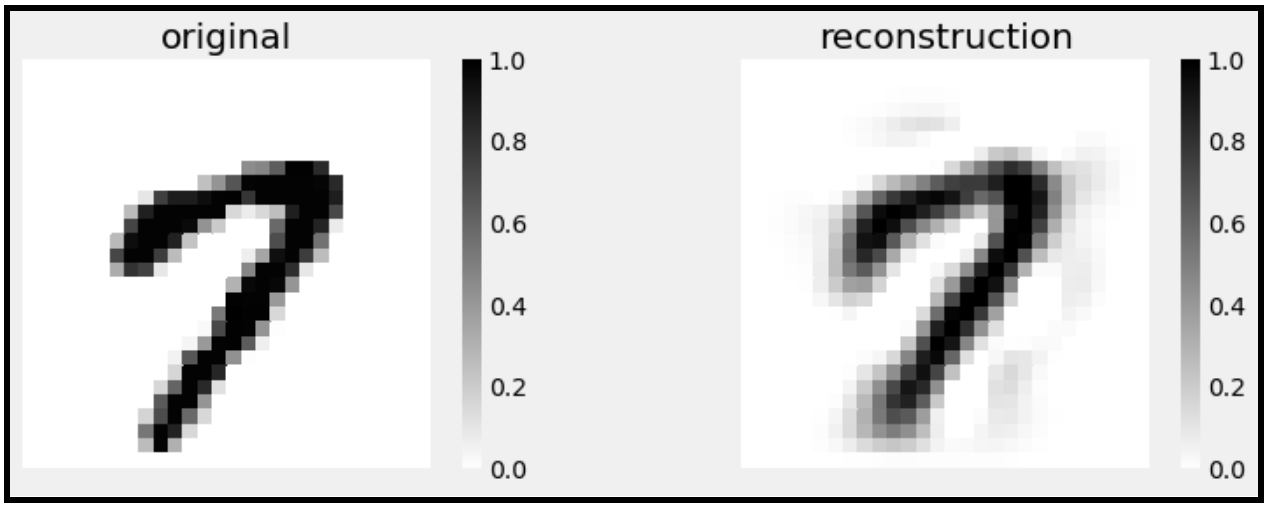


Fig: reconstruction in PCA

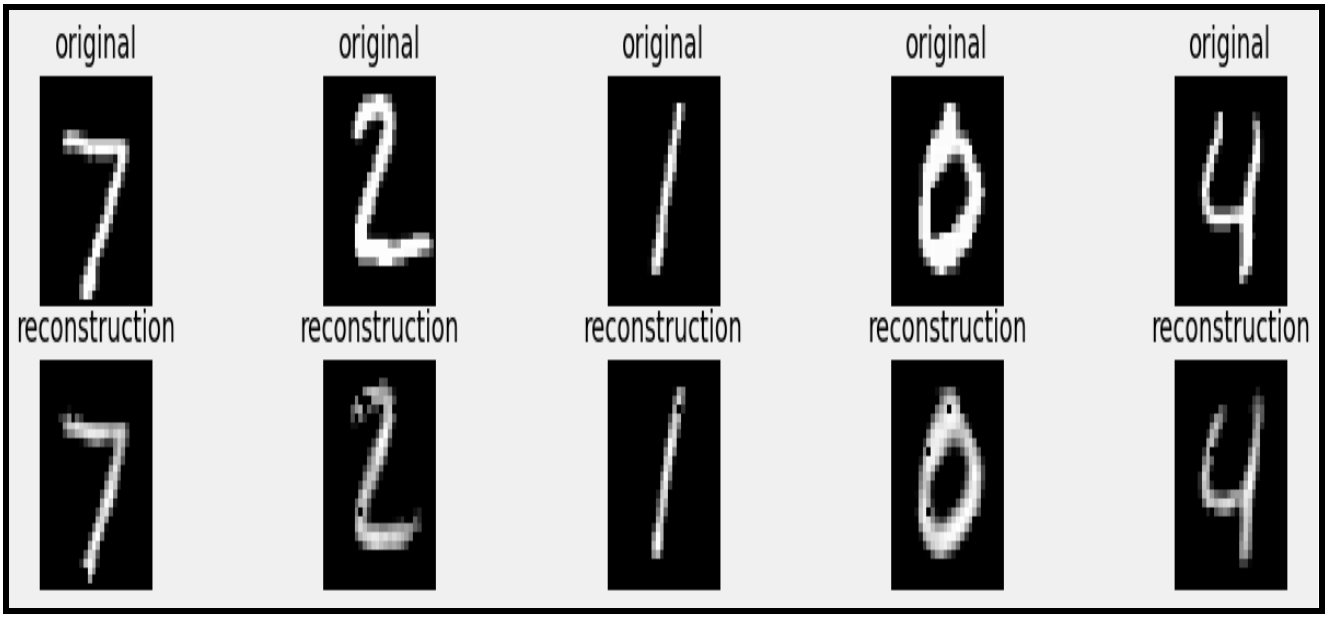


Fig: reconstruction in autoencoder

Q-2: Experimenting with hidden units of varying sizes

1. Training a standard auto-encoder with the following architecture:

   • fc(x)-fc(784)

Here, x is the size of the hidden unit. The architecture consists of only a hidden layer and the output layer. Using x = [64, 128, 256]

1. fc(64)-fc(784)

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_4 (Flatten)         (None, 784)               0

 dense_20 (Dense)            (None, 64)                50240

 dense_21 (Dense)            (None, 64)                4160

 dense_22 (Dense)            (None, 784)               50960

 reshape_4 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 105,360
Trainable params: 105,360
Non-trainable params: 0
_________________________________________________________________
```

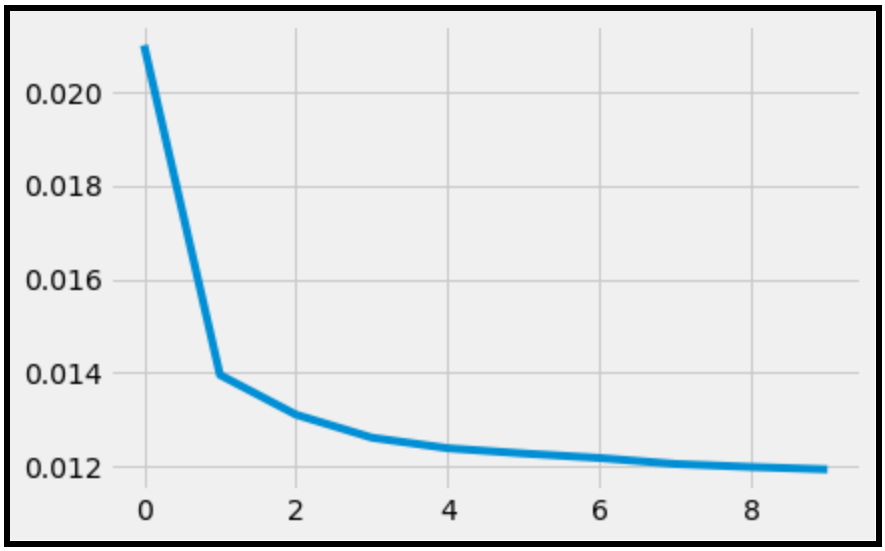


Fig: training loss

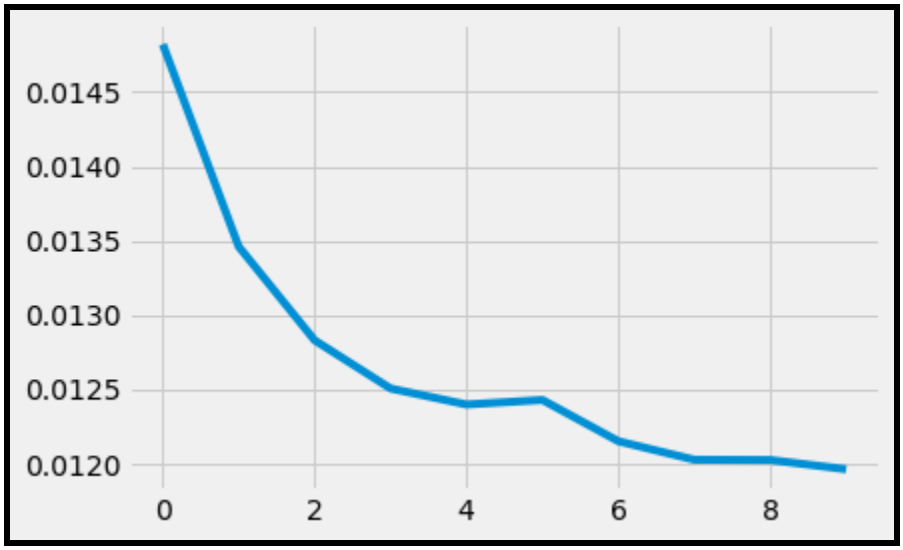


Fig:validation loss

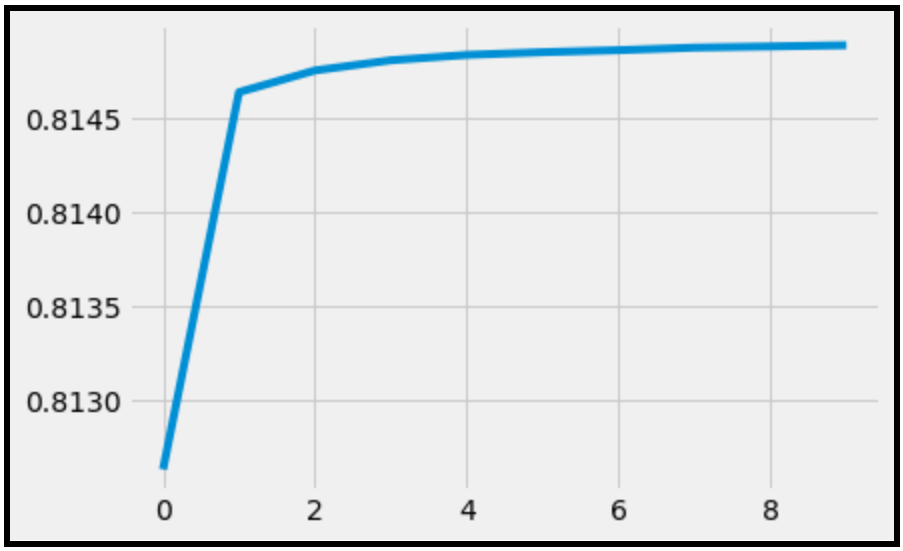


Fig: training accuracy

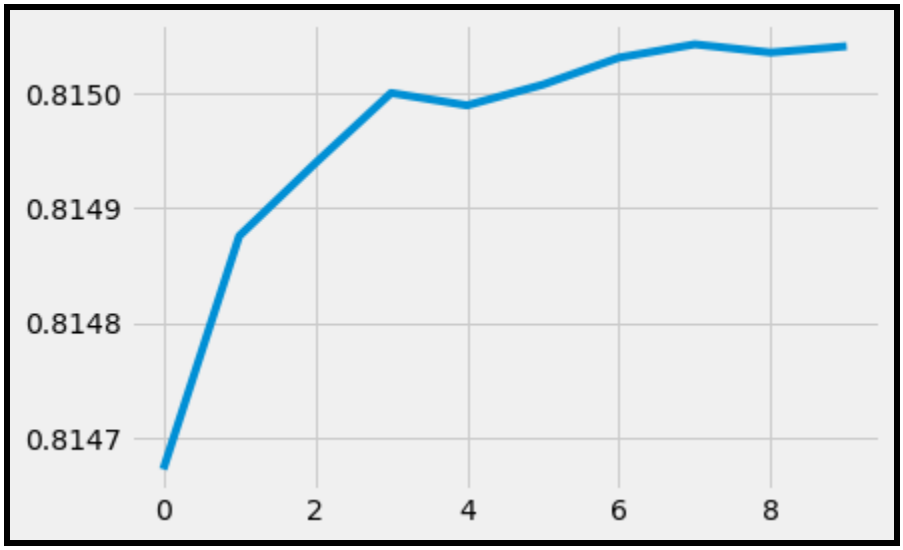


Fig: validation accuracy

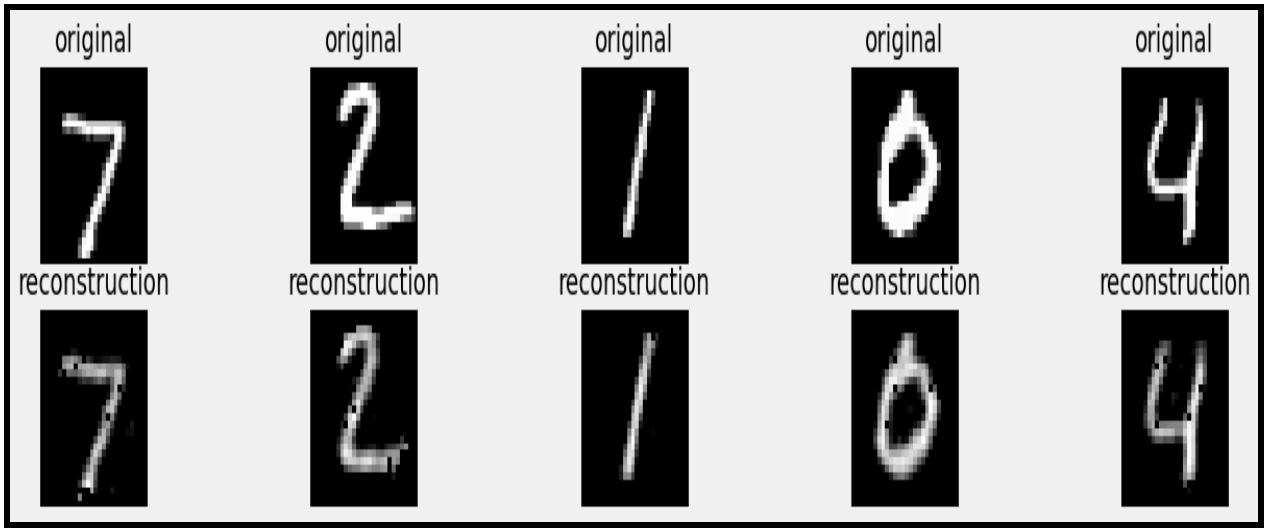


Fig: reconstruction

fc(128)-fc(784)

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_5 (Flatten)         (None, 784)               0

 dense_23 (Dense)            (None, 128)               100480

 dense_24 (Dense)            (None, 128)               16512

 dense_25 (Dense)            (None, 784)               101136

 reshape_5 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 218,128
Trainable params: 218,128
Non-trainable params: 0
_________________________________________________________________
```

Validation accuracy =81.53%

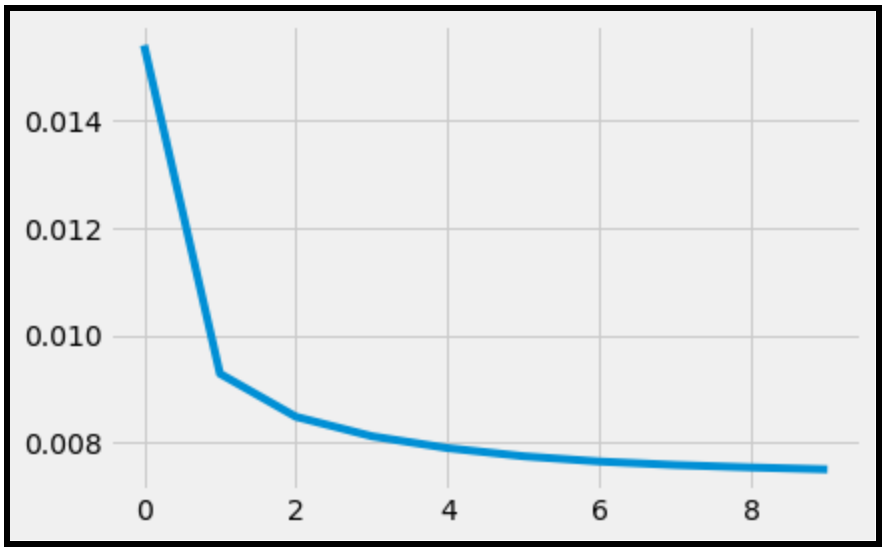
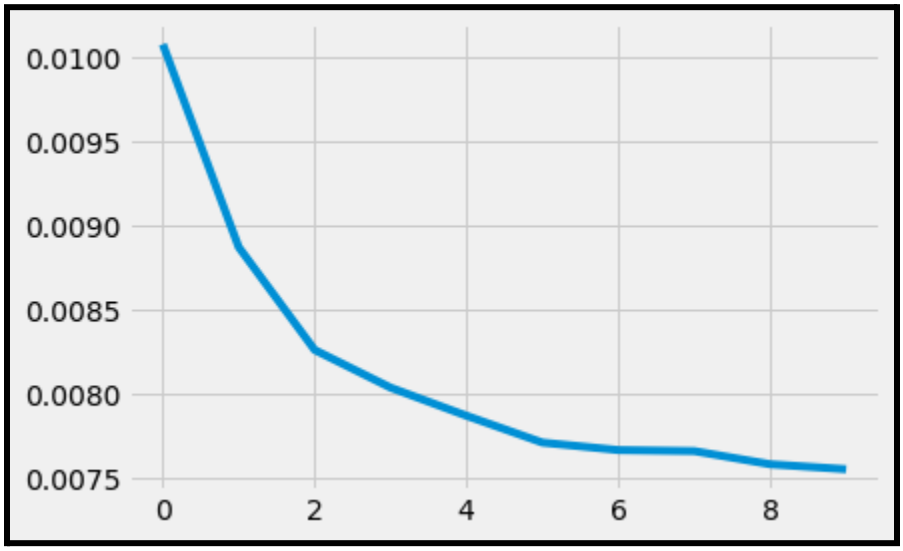


Fig:training loss



Validation loss

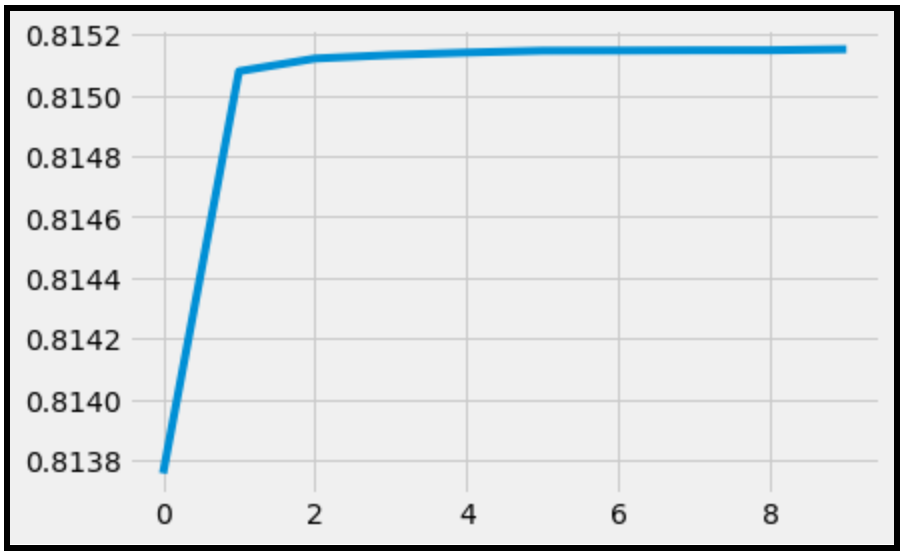


Fig: training accuracy

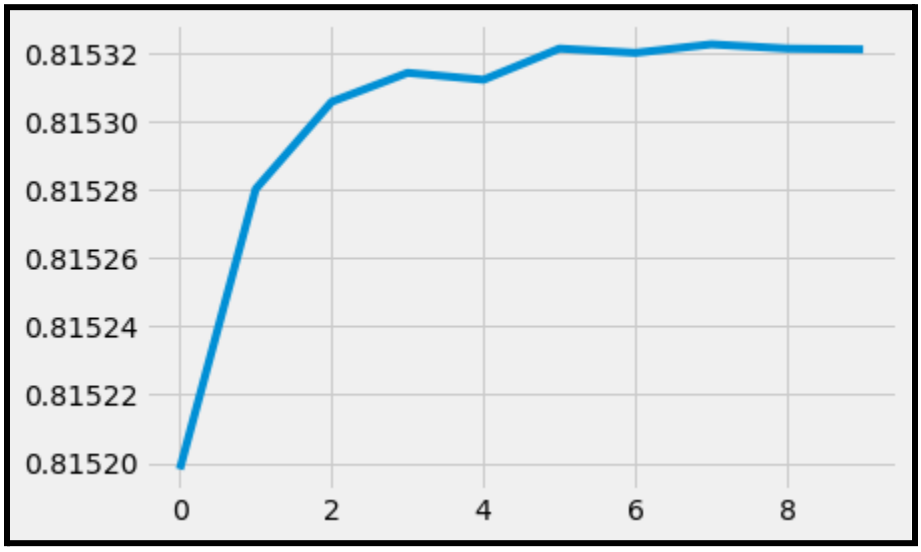
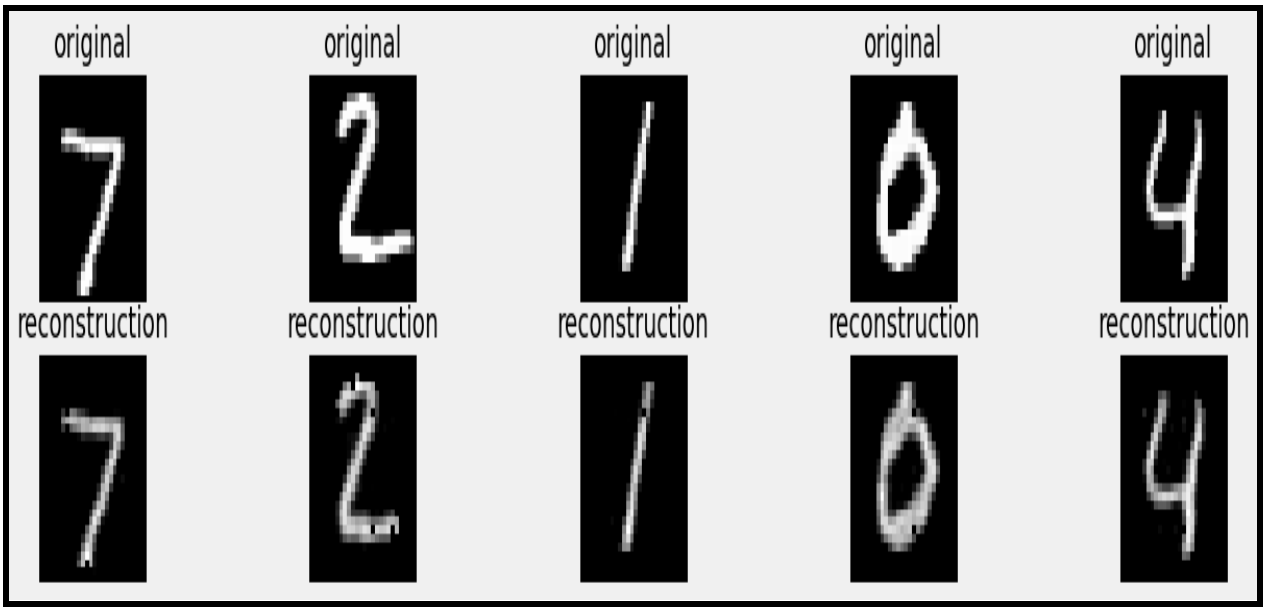
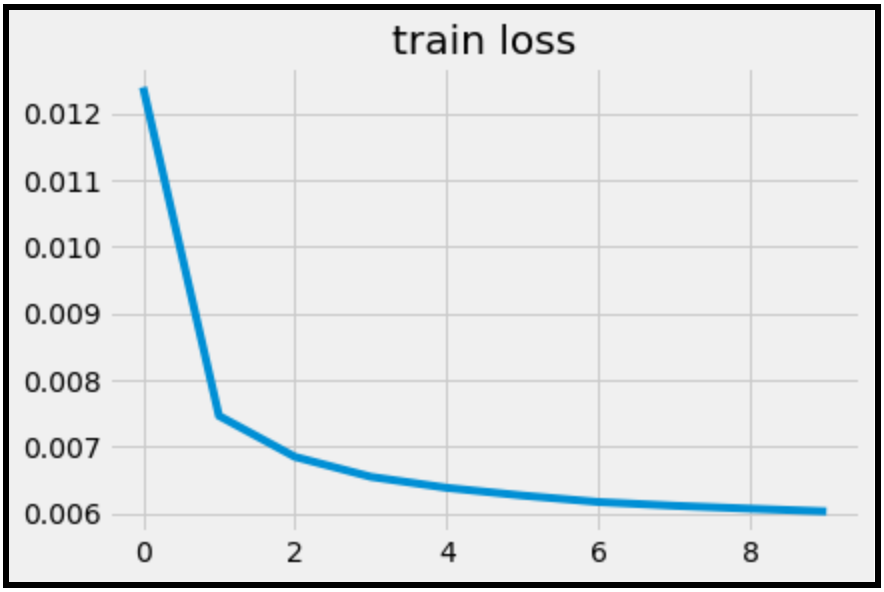


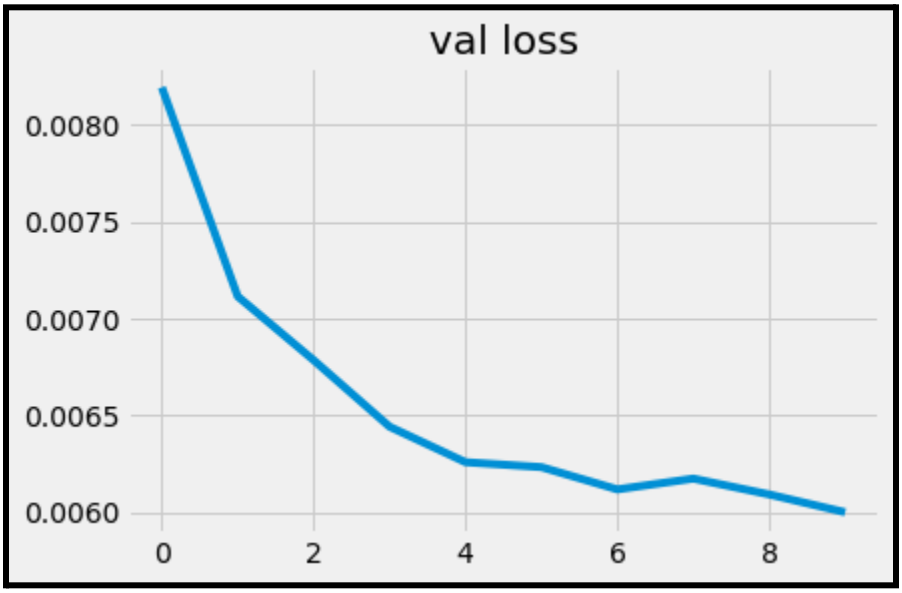
Fig: validation accuracy

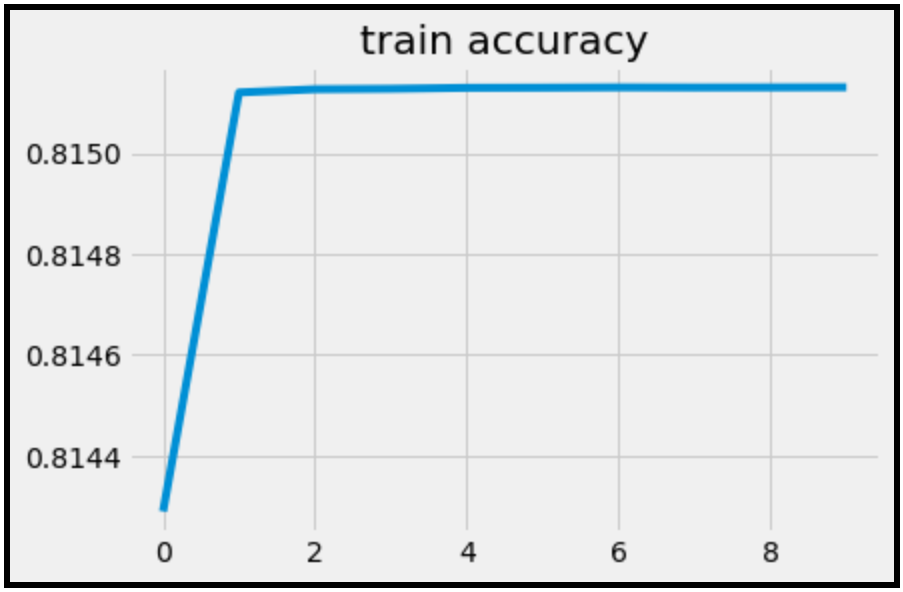


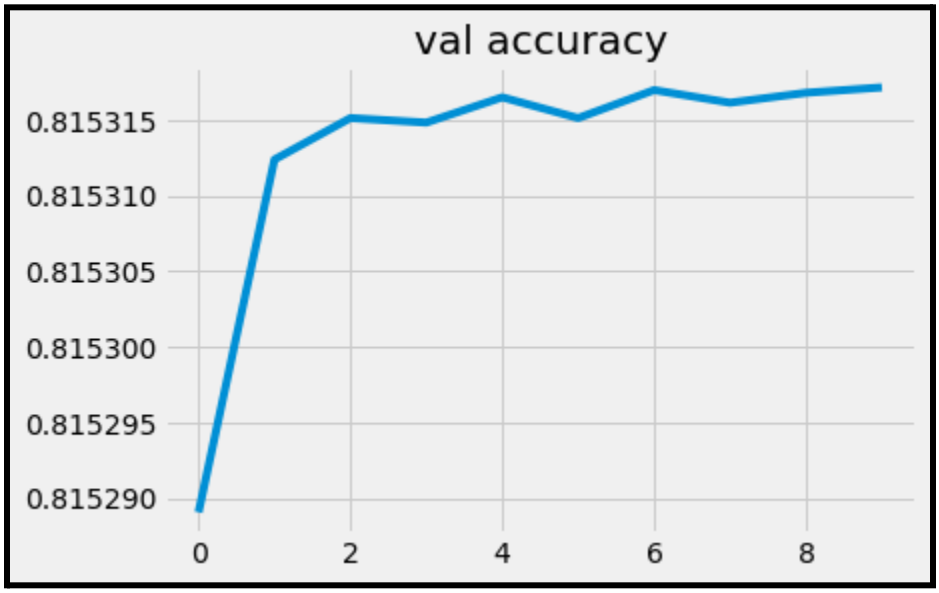
fc(256)-fc(784)

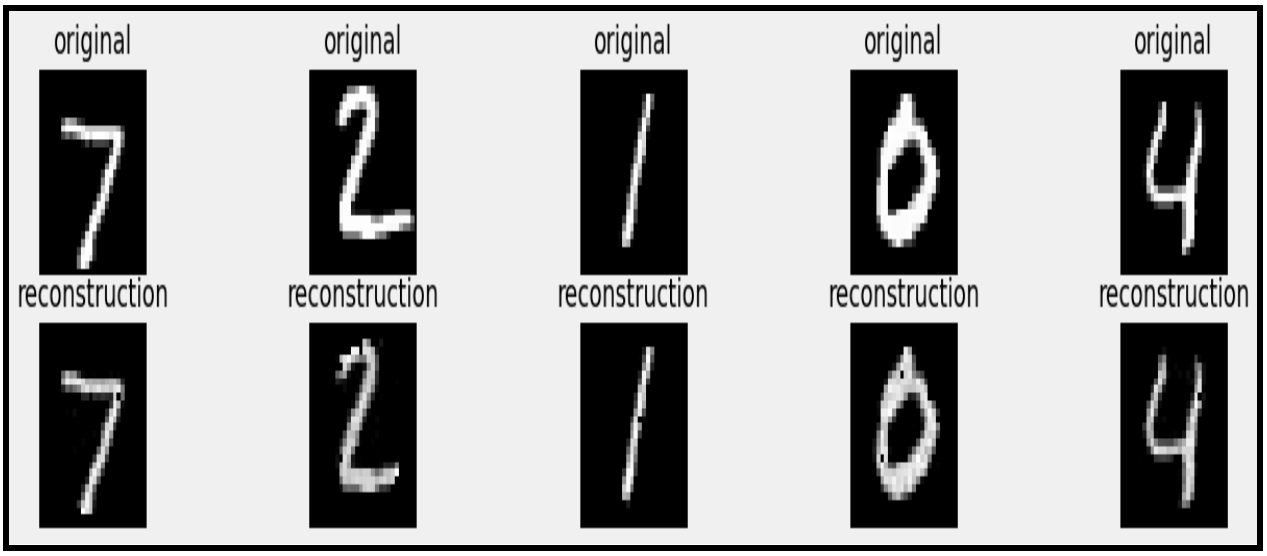
```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_6 (Flatten)         (None, 784)               0

 dense_26 (Dense)            (None, 256)               200960

 dense_27 (Dense)            (None, 256)               65792

 dense_28 (Dense)            (None, 784)               201488

 reshape_6 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 468,240
Trainable params: 468,240
Non-trainable params: 0
```


train loss
0.012
0.011
0.010
0.009
0.008
0.007
0.006
0
2
4
6
8


val loss
0.0080
0.0075
0.0070
0.0065
0.0060
0
2
4
6
8


train accuracy
0.8150
0.8148
0.8146
0.8144
0
2
4
6
8


val accuracy
0.815315
0.815310
0.815305
0.815300
0.815295
0.815290
0
2
4
6
8



Comparison between the model with varying hidden sizes :

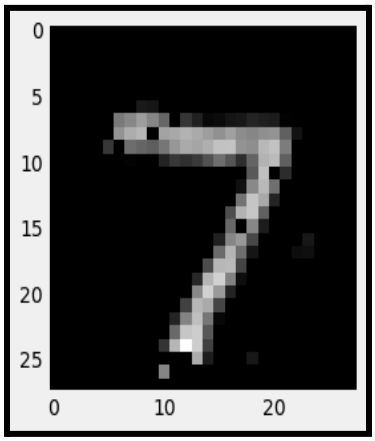


Fig: fc(64)

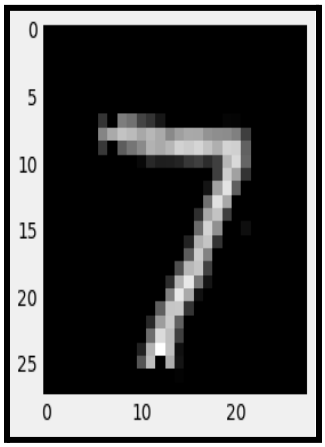


Fig: fc(128)

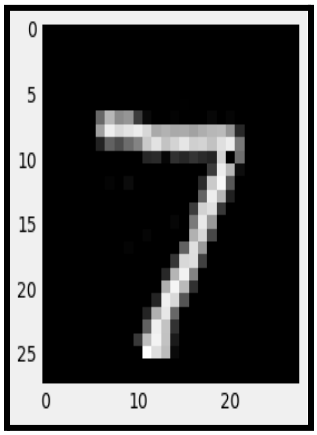


Fig: fc(256)

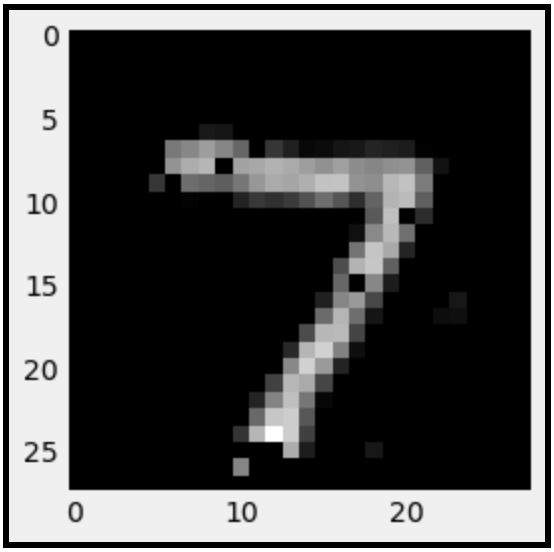


Fig: fc(64)

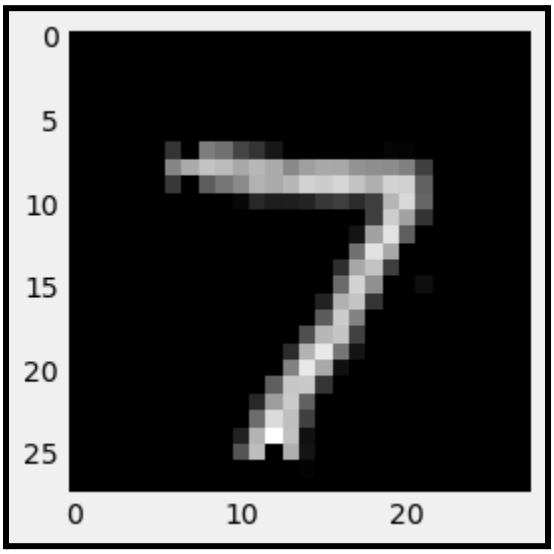


Fig: fc(128)

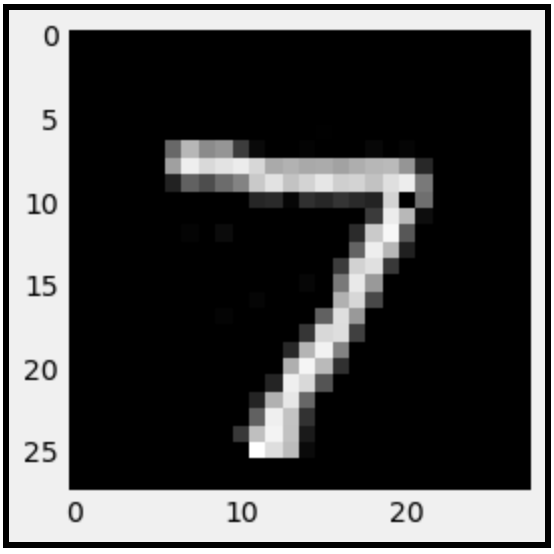


Fig: fc(256)

MNIST Fashion dataset:

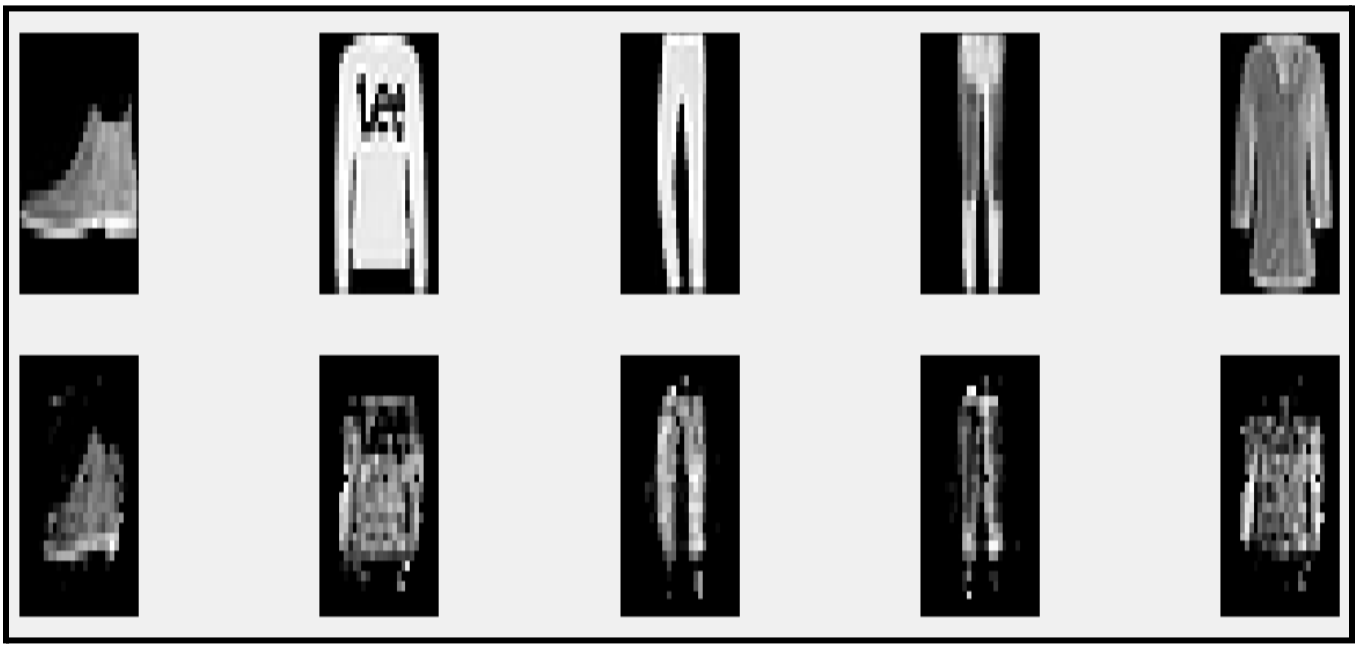


fig : reconstruction of mnist fashion dataset images with trained autoencoder

Random noise images through trained autoencoder:

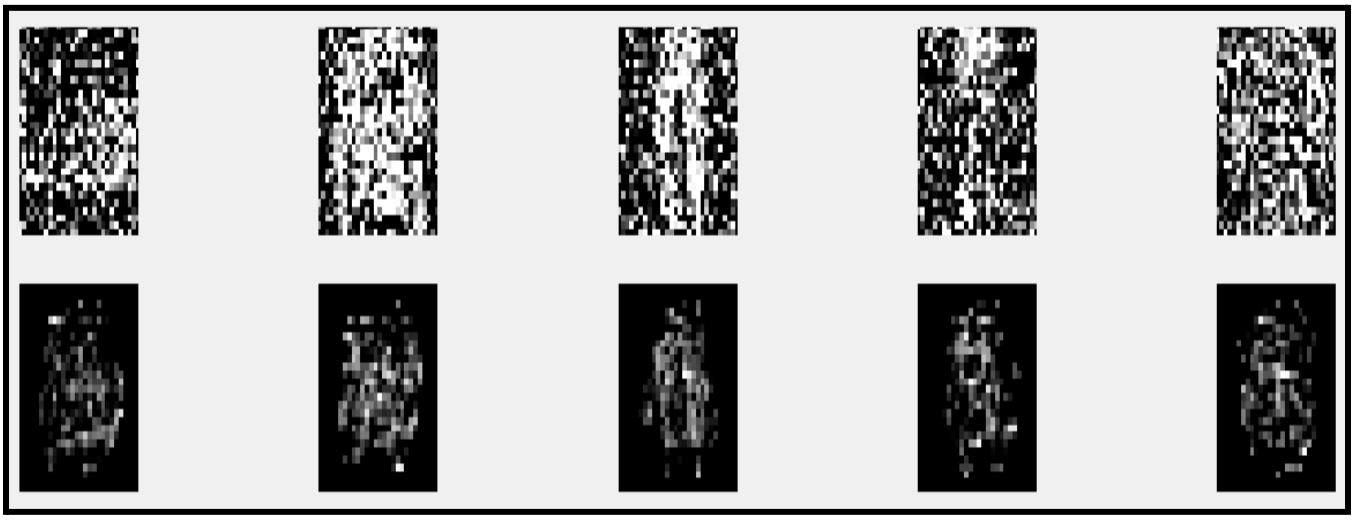
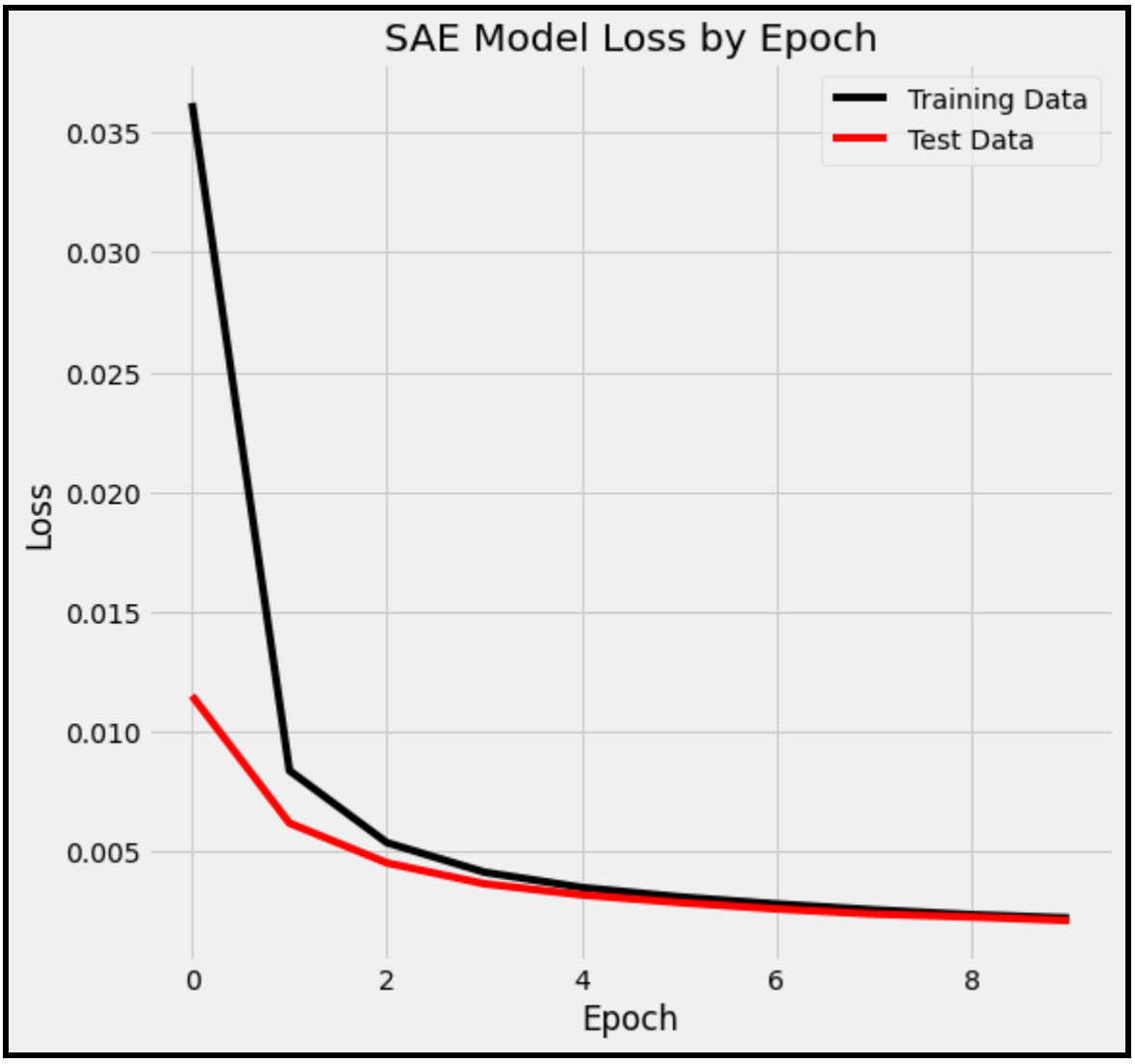


Fig:random noise image reconstruction using trained encoder.

Q-3: Sparse autoencoders:

Trained an over-complete autoencoder with L1 sparsity regurgitation.

1. Sparsity value: 0.0001



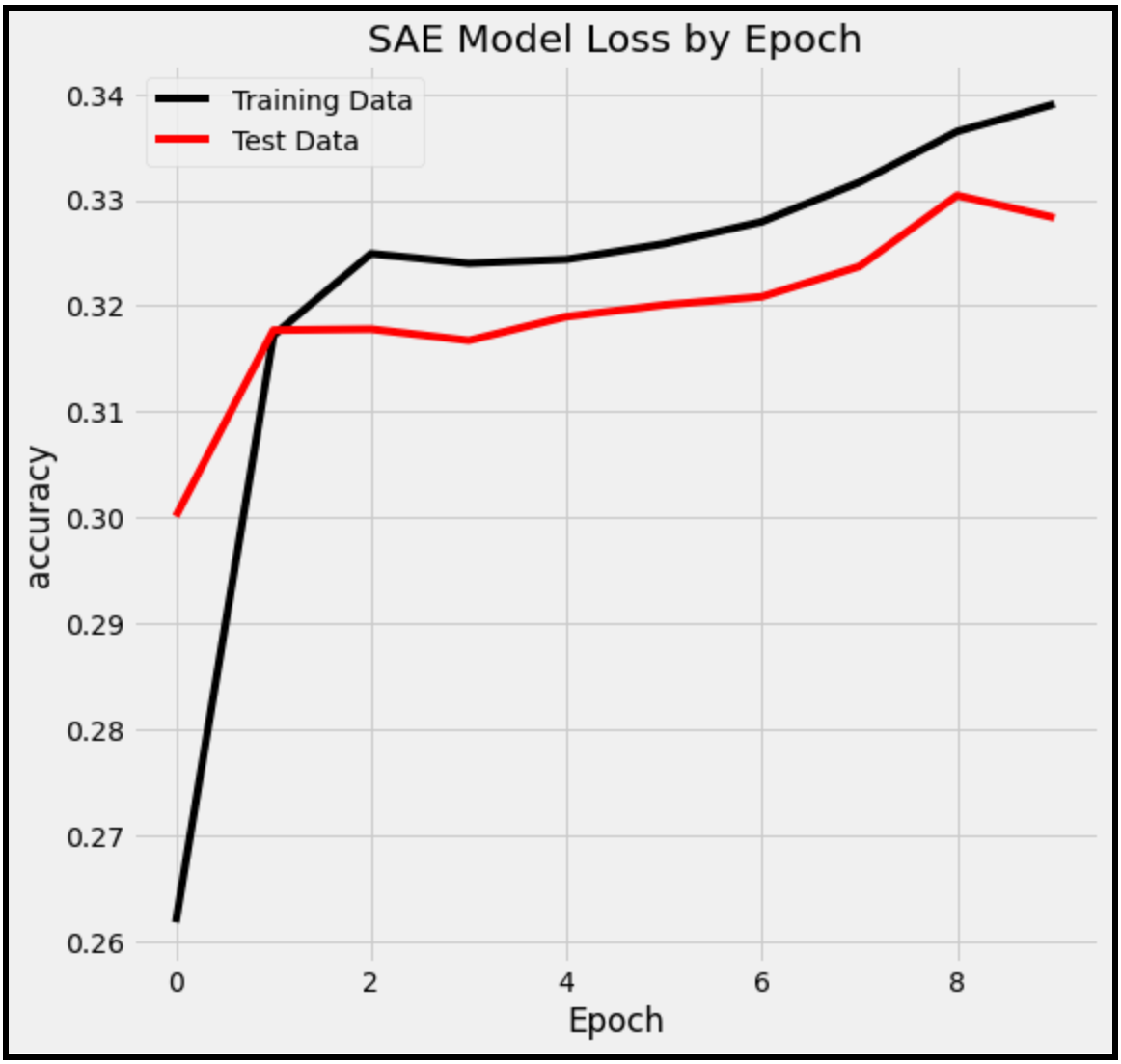


Fig: accuracy plot

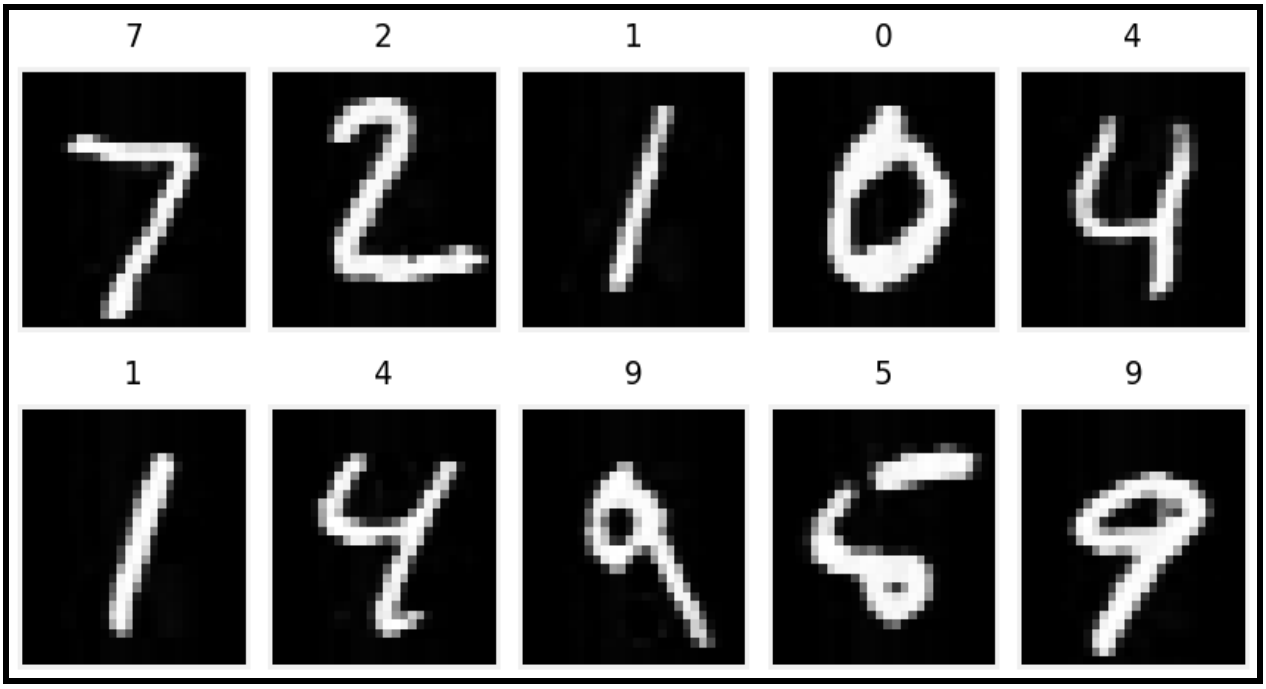
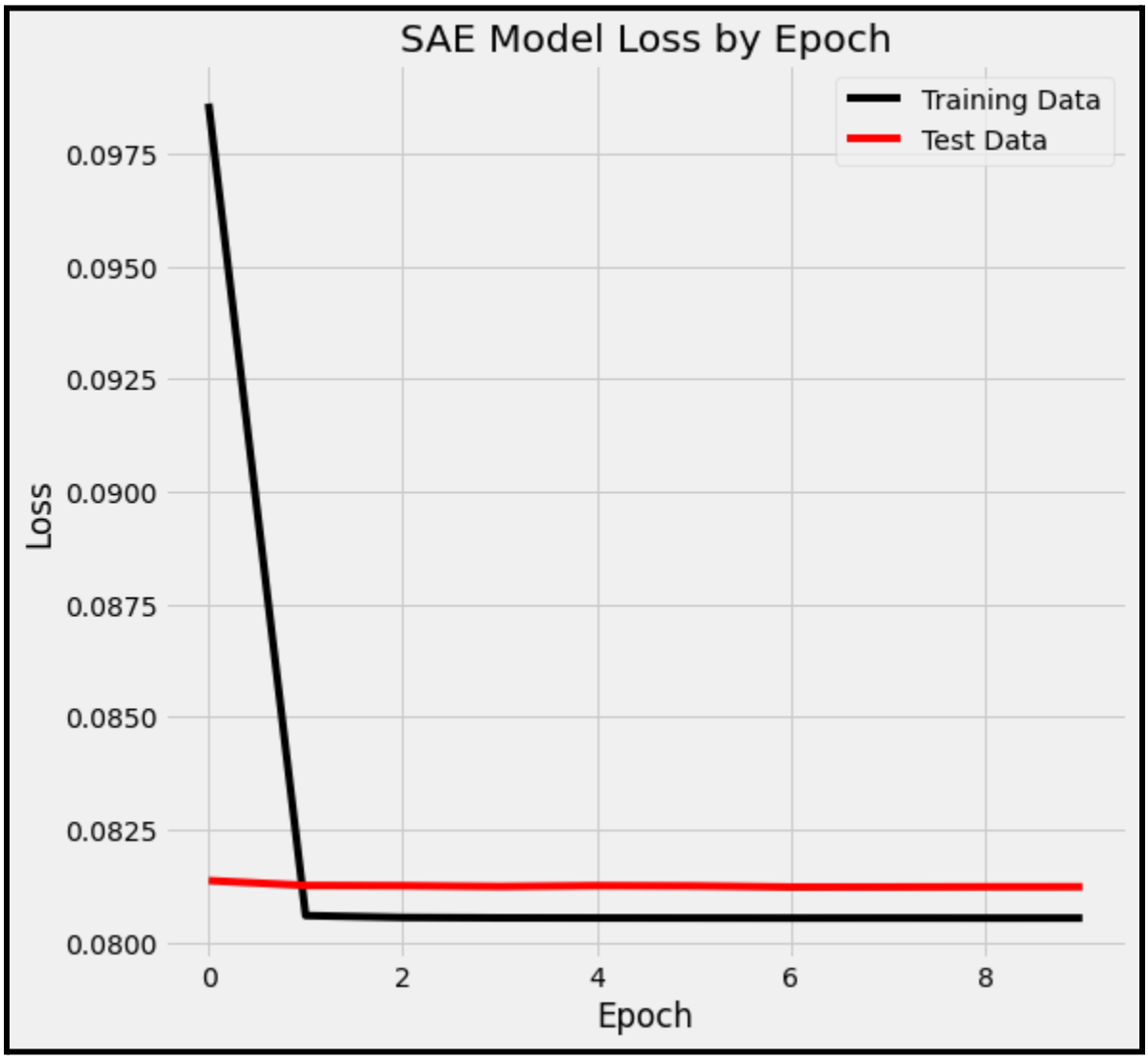


Fig: reconstruction

2. Sparsity value: 0.001



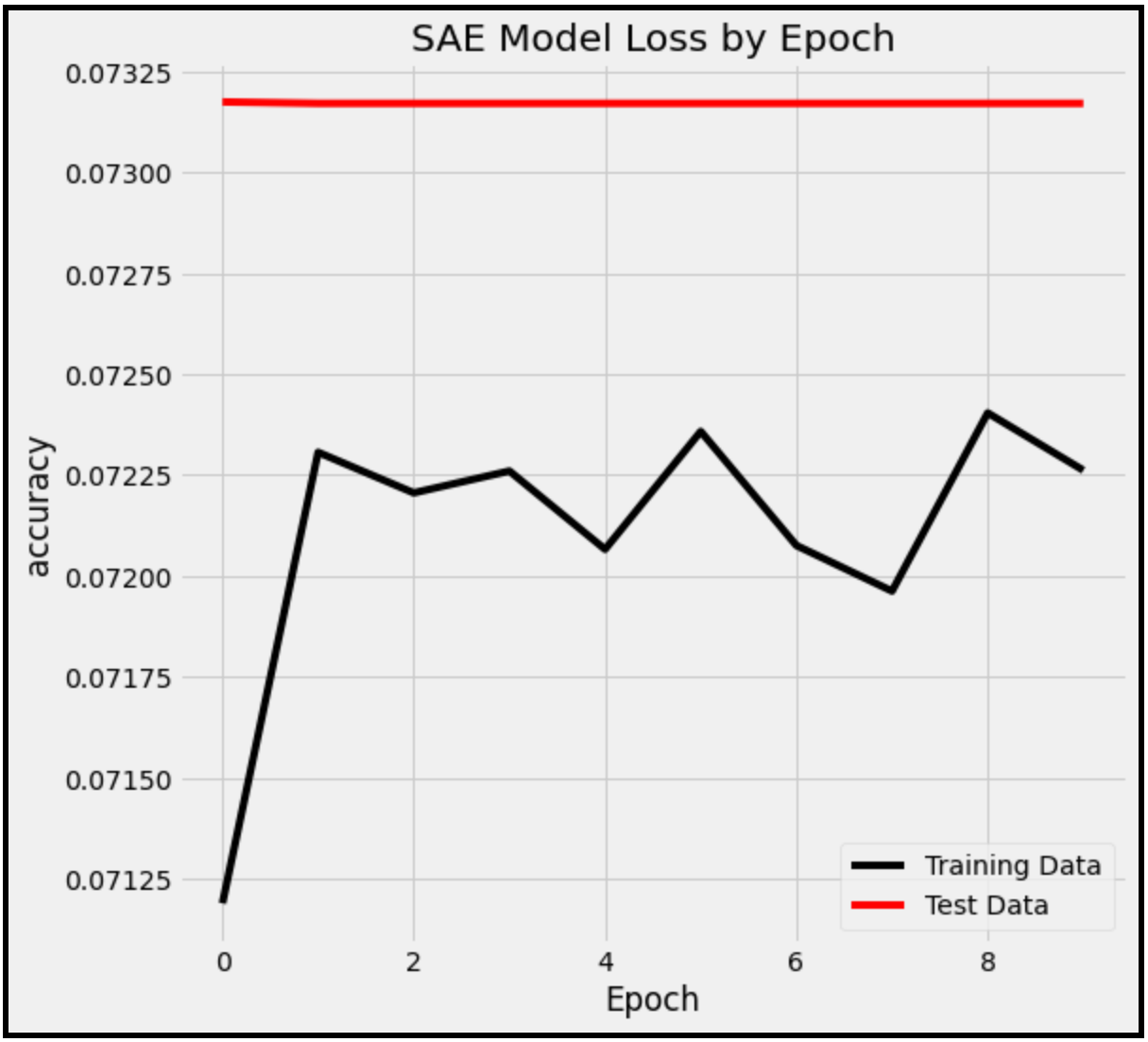


Fig: model accuracy

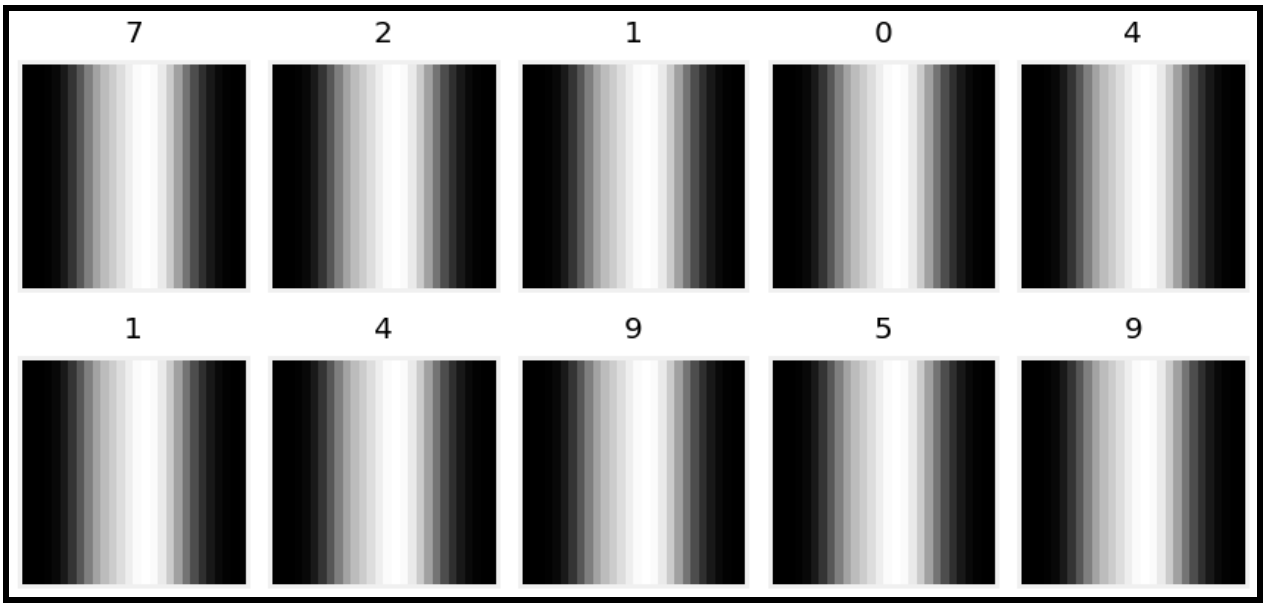
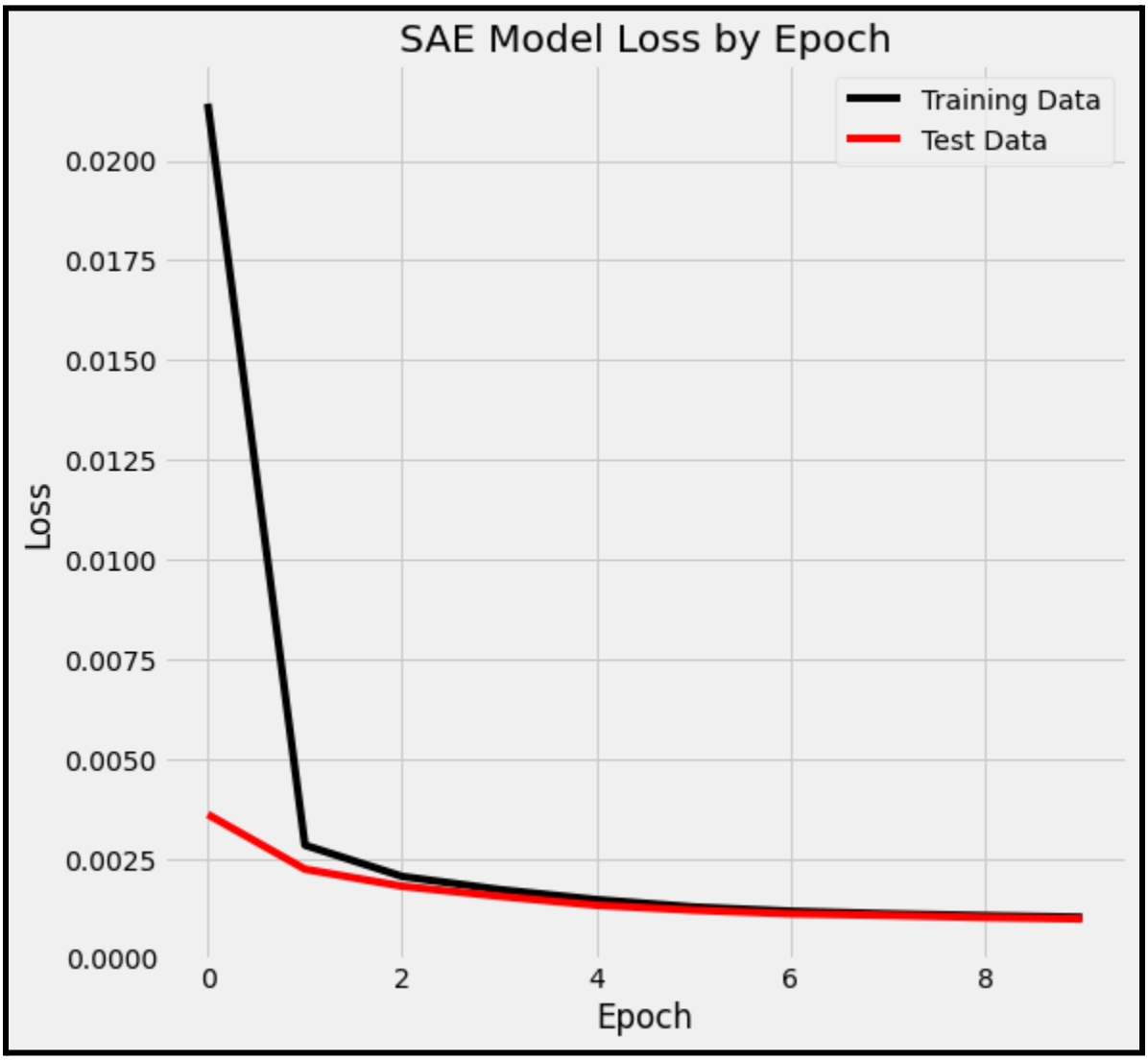
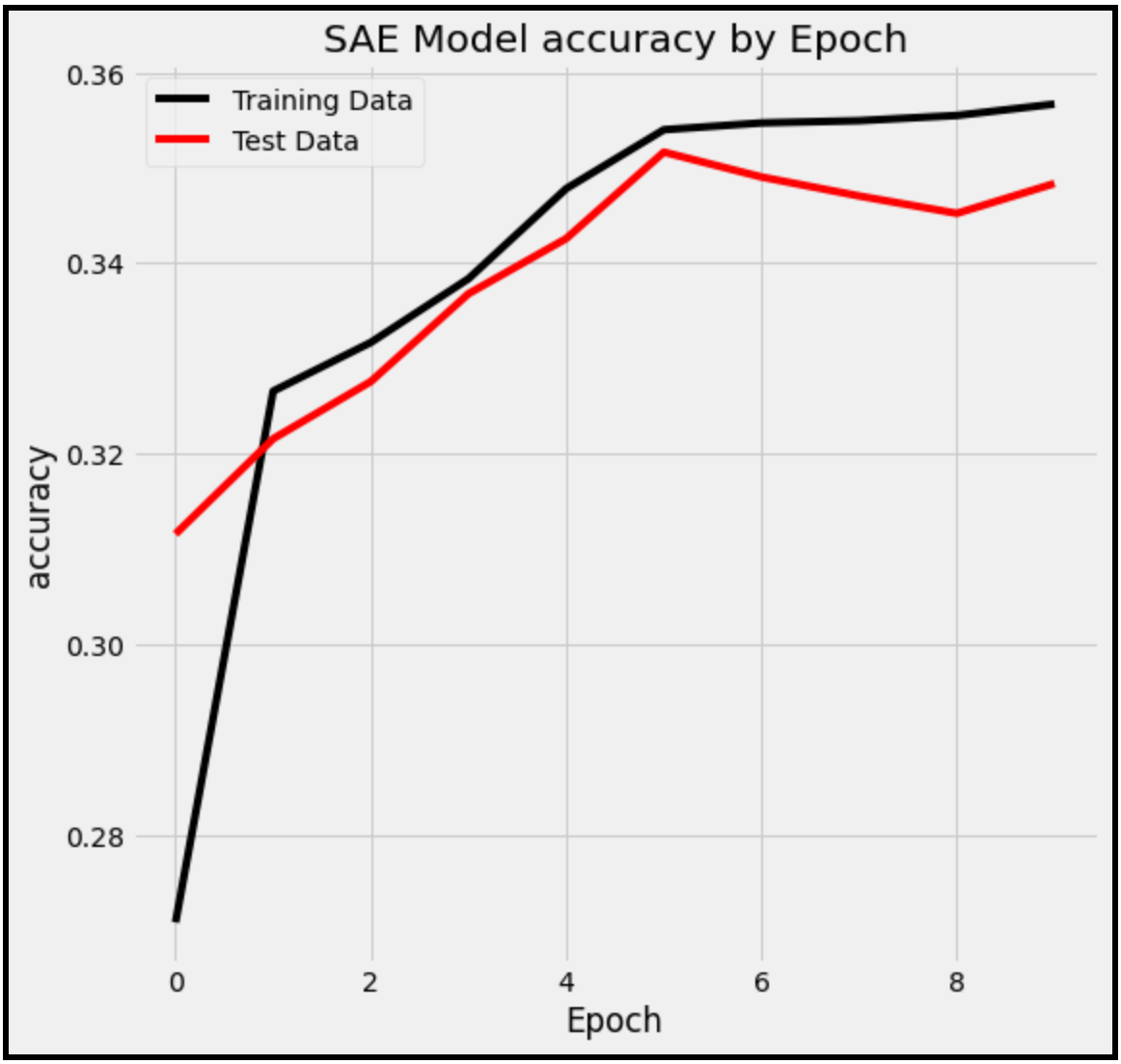


Fig: reconstruction

3. Sparsity value: 0.000001




SAE Model accuracy by Epoch
Training Data
Test Data
0.36
0.34
0.32
0.30
0.28
accuracy
0
2
4
6
8
Epoch

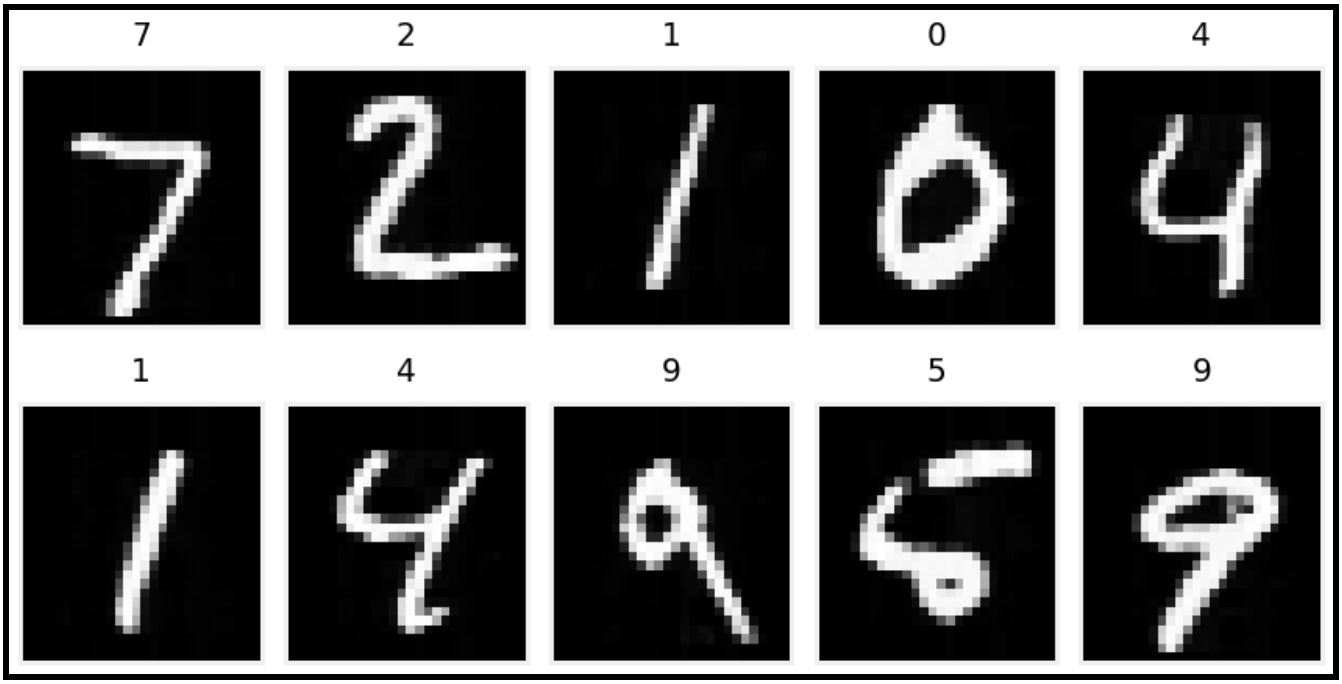


Fig: reconstruction

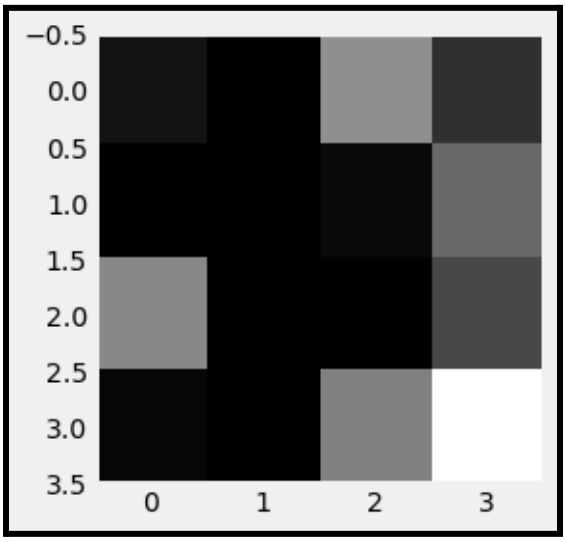


Fig: filter for sparsity

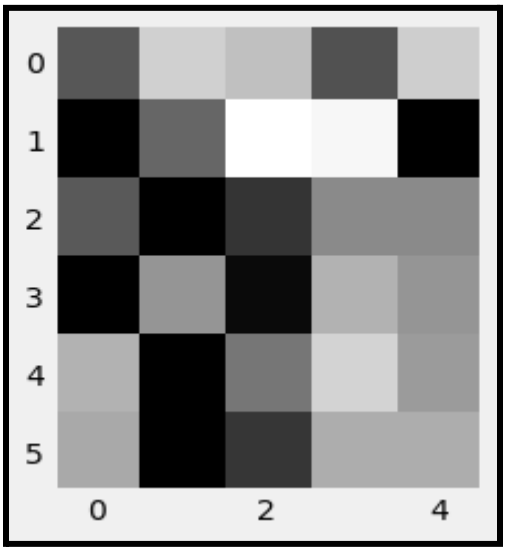


Fig: filter for standard encoder

Observation:

1. In sparsity regulated autoencoder, many values are hidden compared to a standard autoencoder.

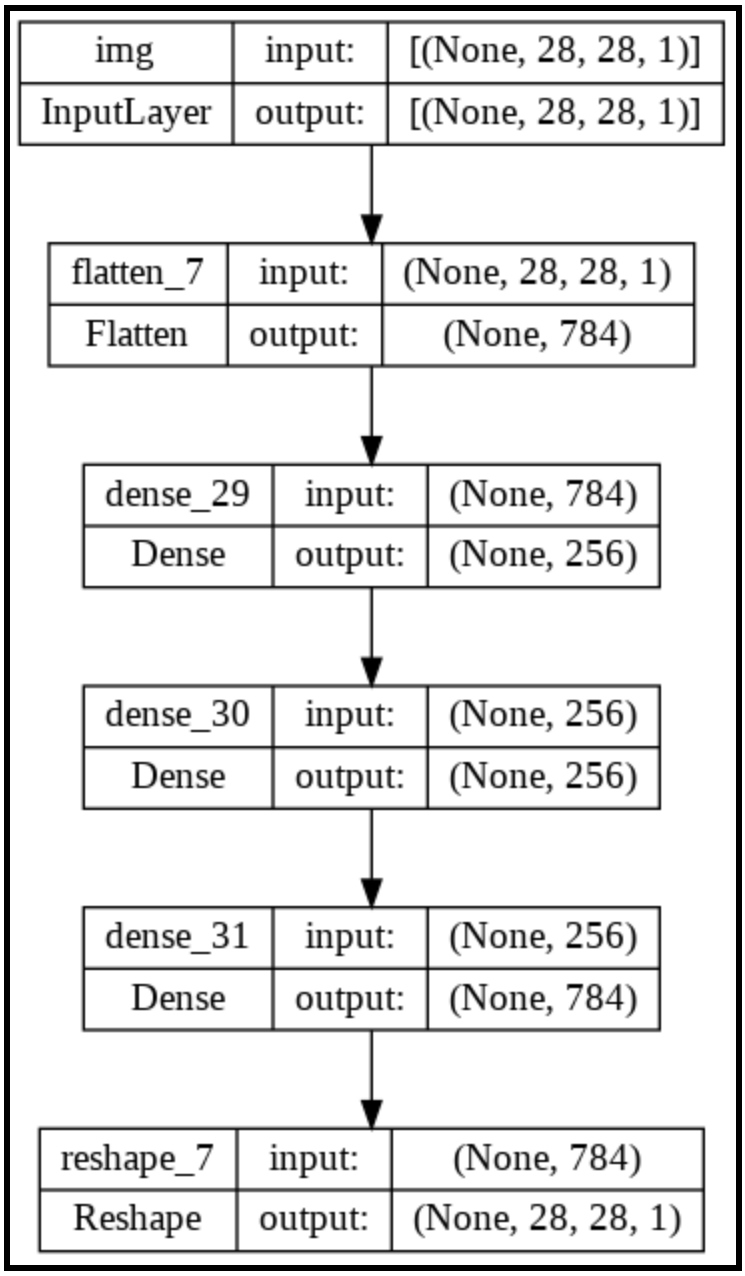
Q-4: Denoising Autoencoder:

Designing denoising autoencoder with one hidden unit(256).
All models trained with batch size= 32 and epoch=10

Noise level=0.7

Fig: example images as generated in noisy data.

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 img (InputLayer)            [(None, 28, 28, 1)]       0

 flatten_7 (Flatten)         (None, 784)               0

 dense_29 (Dense)            (None, 256)               200960

 dense_30 (Dense)            (None, 256)               65792

 dense_31 (Dense)            (None, 784)               201488

 reshape_7 (Reshape)         (None, 28, 28, 1)         0

=================================================================
Total params: 468,240
Trainable params: 468,240
Non-trainable params: 0
```



validation accuracy= 81.03%

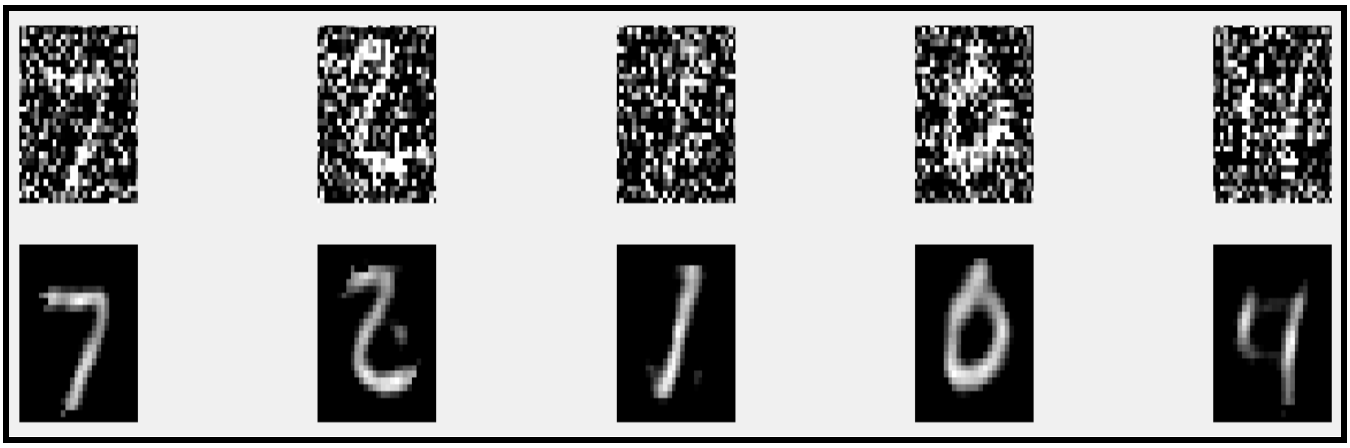


Fig: original and reconstructed

Noise level: 0.3

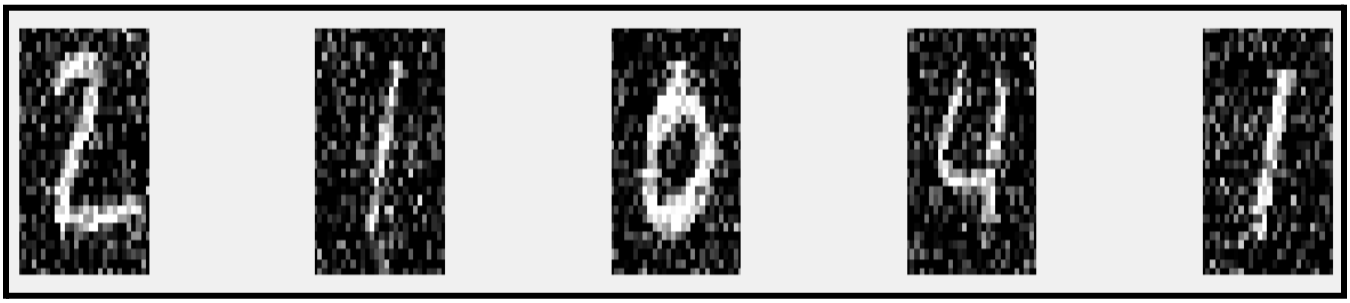


Fig: noisy images

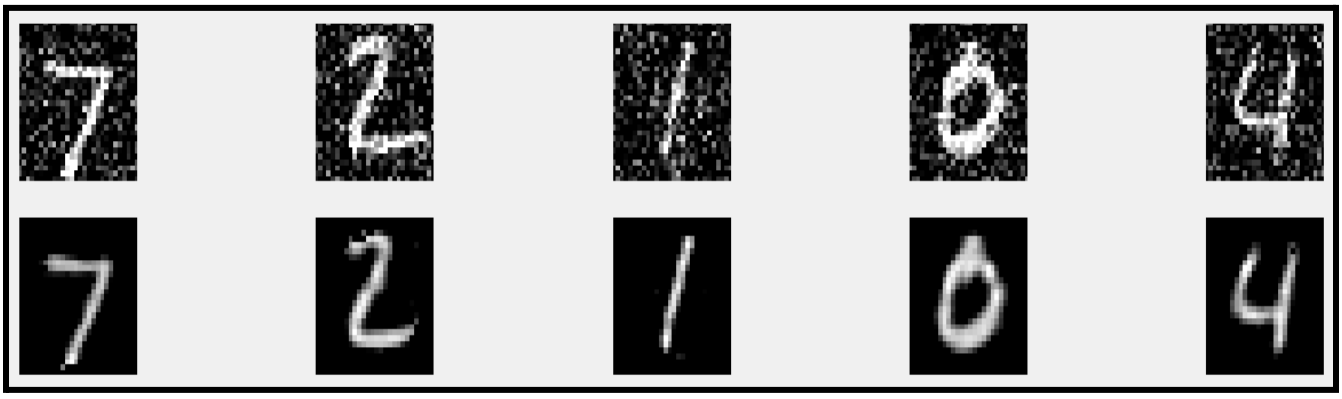


Fig: original and reconstructed

Noise level = 0.5

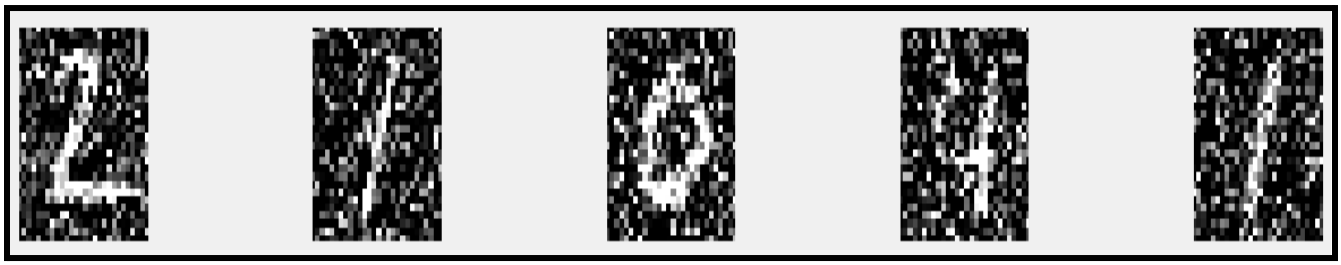


Fig: noisy images

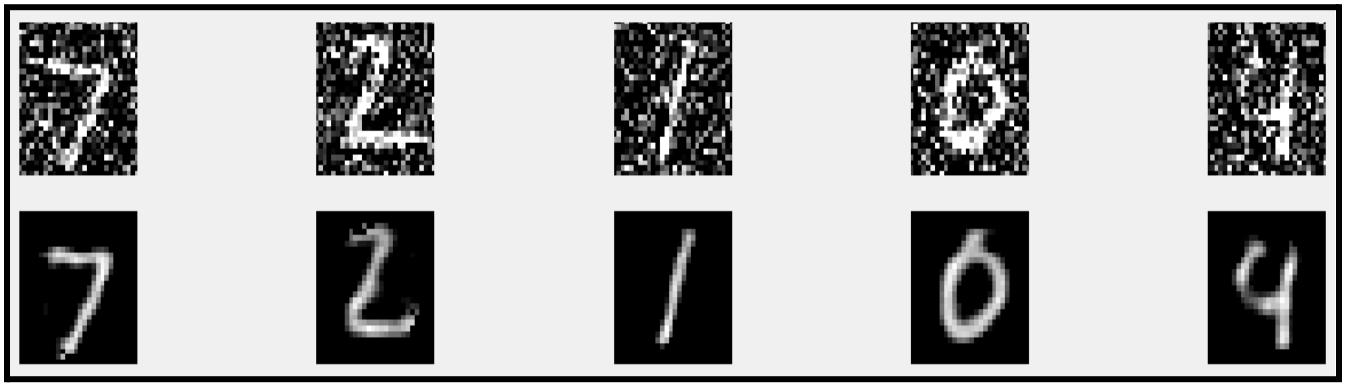


Fig: original and reconstructed

Noise level = 0.8

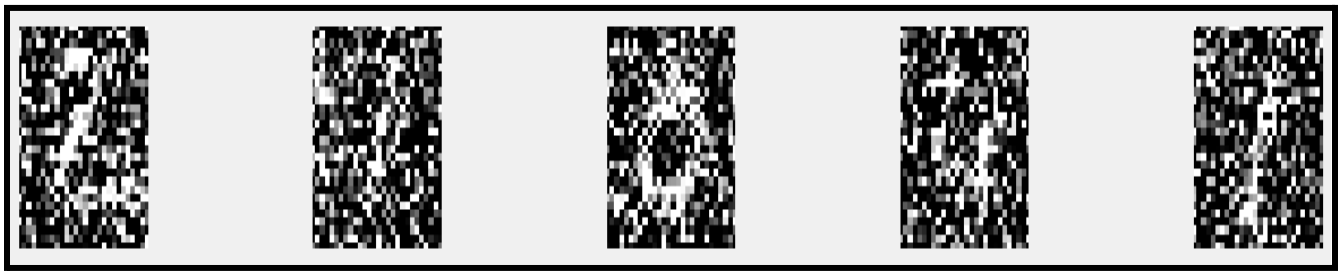


Fig: noisy images

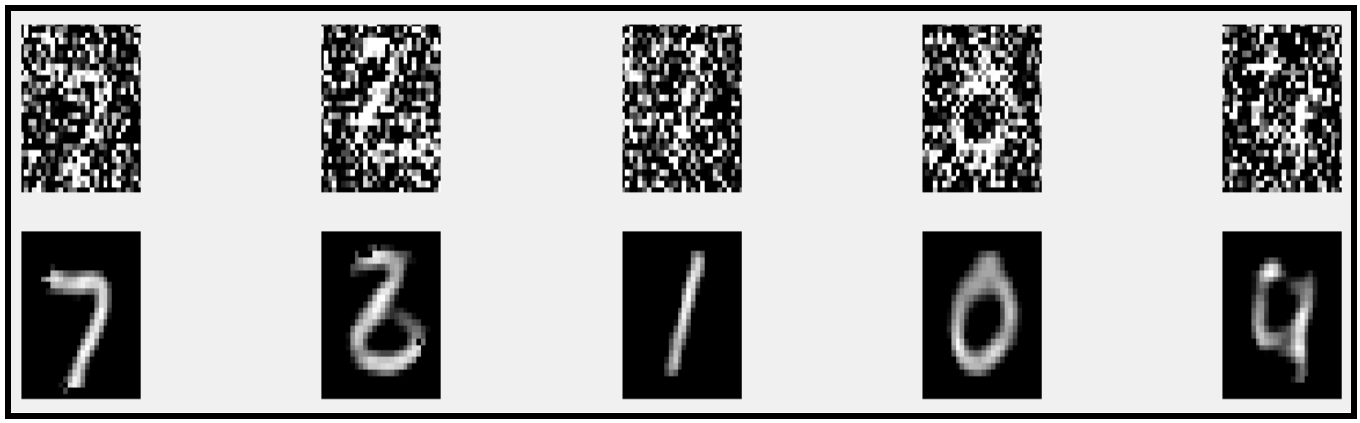


Fig: original and reconstructed

Noise level= 0.9

validation accuracy= 80.7%

Fig: noisy images

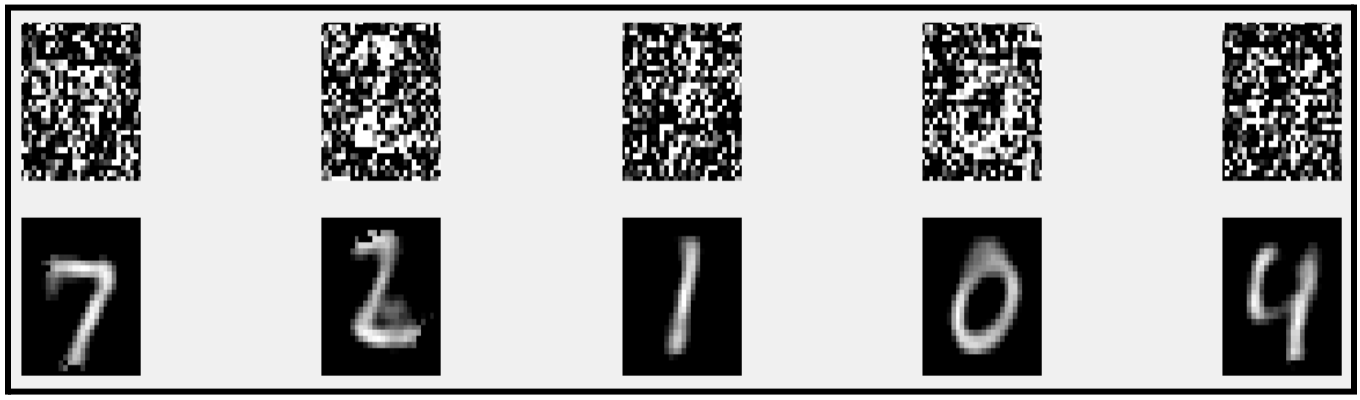
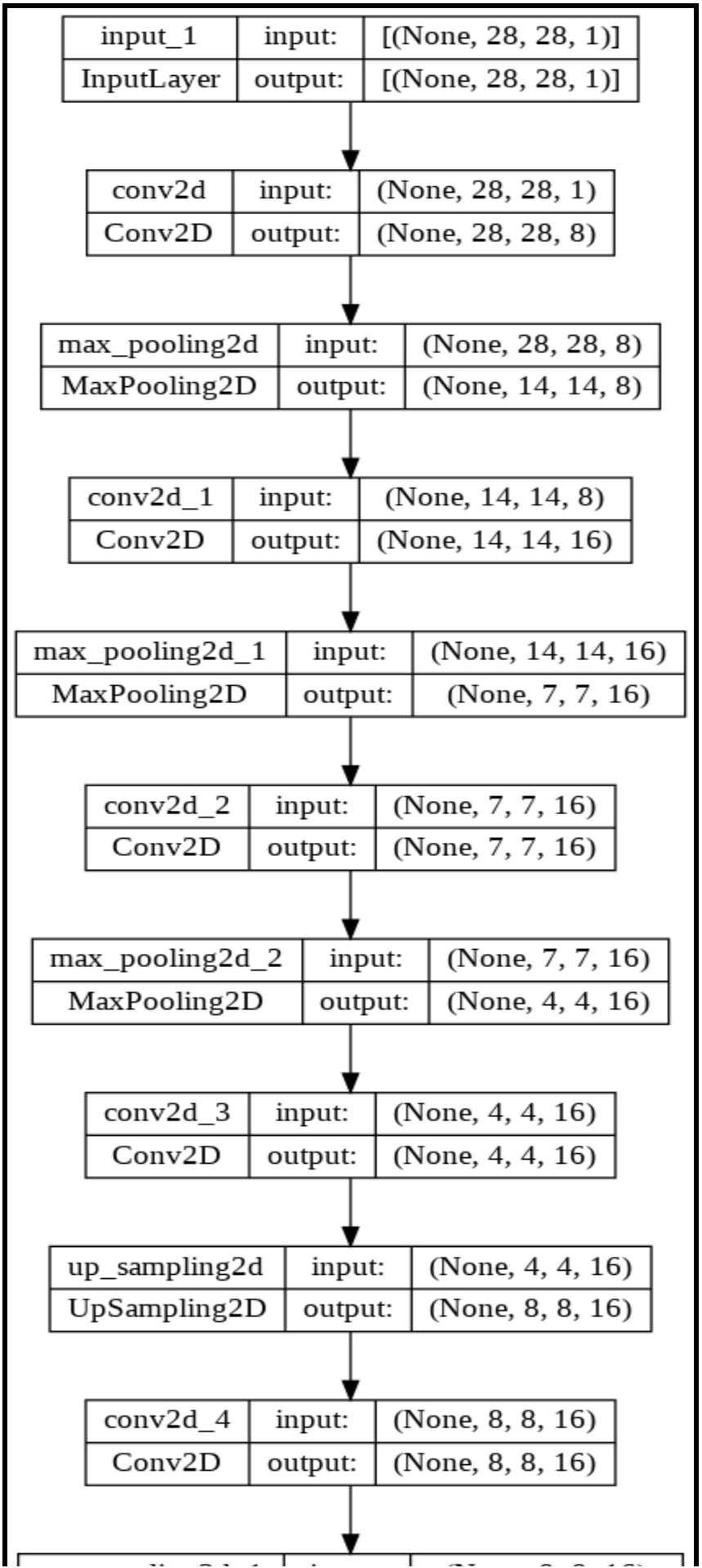


Fig: original and reconstructed

Q-5: convolutional autoencoder:


input_1 | input: | [(None, 28, 28, 1)]
InputLayer | output: | [(None, 28, 28, 1)]
conv2d | input: | (None, 28, 28, 1)
Conv2D | output: | (None, 28, 28, 8)
max_pooling2d | input: | (None, 28, 28, 8)
MaxPooling2D | output: | (None, 14, 14, 8)
conv2d_1 | input: | (None, 14, 14, 8)
Conv2D | output: | (None, 14, 14, 16)
max_pooling2d_1 | input: | (None, 14, 14, 16)
MaxPooling2D | output: | (None, 7, 7, 16)
conv2d_2 | input: | (None, 7, 7, 16)
Conv2D | output: | (None, 7, 7, 16)
max_pooling2d_2 | input: | (None, 7, 7, 16)
MaxPooling2D | output: | (None, 4, 4, 16)
conv2d_3 | input: | (None, 4, 4, 16)
Conv2D | output: | (None, 4, 4, 16)
up_sampling2d | input: | (None, 4, 4, 16)
UpSampling2D | output: | (None, 8, 8, 16)
conv2d_4 | input: | (None, 8, 8, 16)
Conv2D | output: | (None, 8, 8, 16)

Trained with epoch=10 and batch size = 32:

1. unpooling:
   Validation accuracy: 81.4%
   Error: 0.086

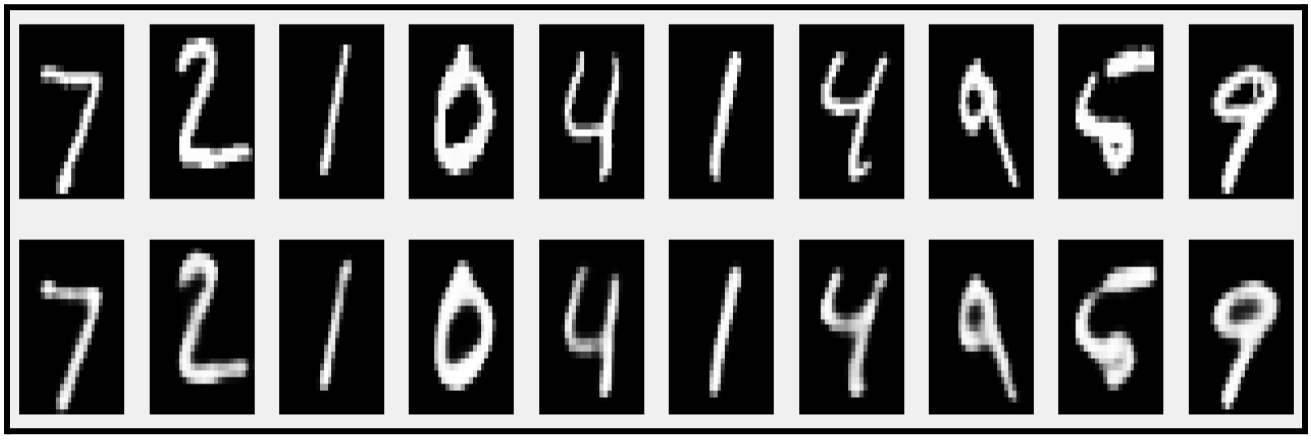


Fig: reconstructed images from test dataset

(Please note: the remaining part of the assignment will be submitted as a different report and code due to time shortage.)