PA4

October 29, 2022

[67]:
```
## import libraries
import numpy as np
import matplotlib.pyplot as plt
import skimage

#torch
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torch.nn.functional as F
from torchvision.datasets import MNIST

#import pca
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error as mse
```

[2]:
```
batch_size = 256
epochs =10
learning_rate  = 0.001
```

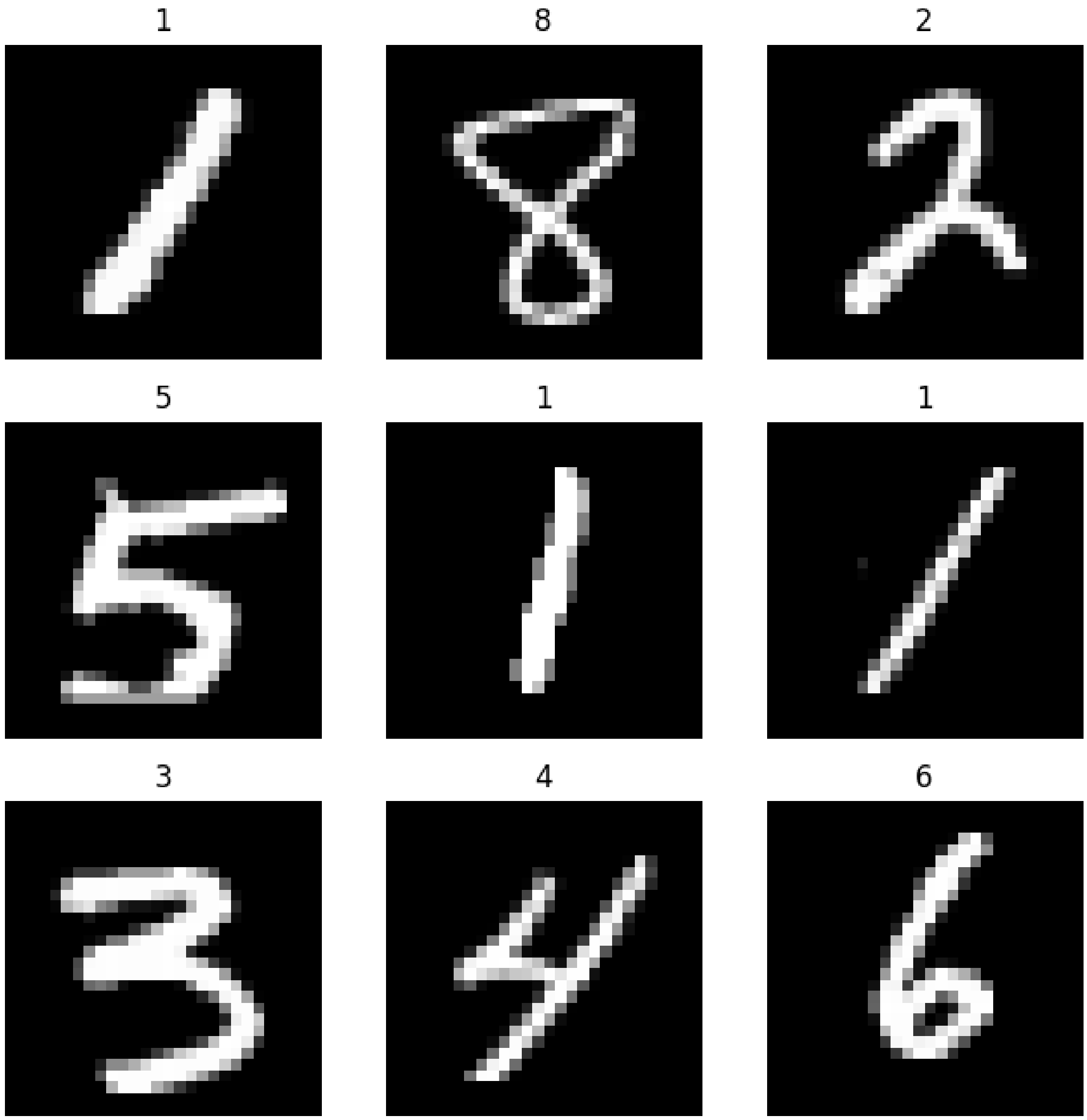
[3]:
```
#Download the dataset
train_dataset = MNIST(root='./data',
                    train=True,
                    transform=transforms.ToTensor(),
                    download=True)

test_dataset = MNIST(root='./data',
                    train=False,
                    transform=transforms.ToTensor())
```

[4]:
```
# Data loader
train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)


test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
```

```
                                            batch_size=batch_size,
                                            shuffle=False)
```

[5]:

```
#code taken from the official documentation
labels_map = {
    0: "0",
    1: "1",
    2: "2",
    3: "3",
    4: "4",
    5: "5",
    6: "6",
    7: "7",
    8: "8",
    9: "9",
}
figure = plt.figure(figsize=(8, 8))
cols, rows = 3, 3
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(train_dataset), size=(1,)).item()
    img, label = train_dataset[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```



[6]:
```
train_images = train_dataset.data
train_labels = train_dataset.targets
print(f"Training image shape:{train_images.shape}")
print(f"Training Targets shape:{train_labels.shape}")


val_images = test_dataset.data
val_labels = test_dataset.targets
print(f"validation image shape:{val_images.shape}")
print(f"validation Targets shape:{val_labels.shape}")
```

```
Training image shape:torch.Size([60000, 28, 28])
```

```
Training Targets shape:torch.Size([60000])
validation image shape:torch.Size([10000, 28, 28])
validation Targets shape:torch.Size([10000])
```

0.1 Q1

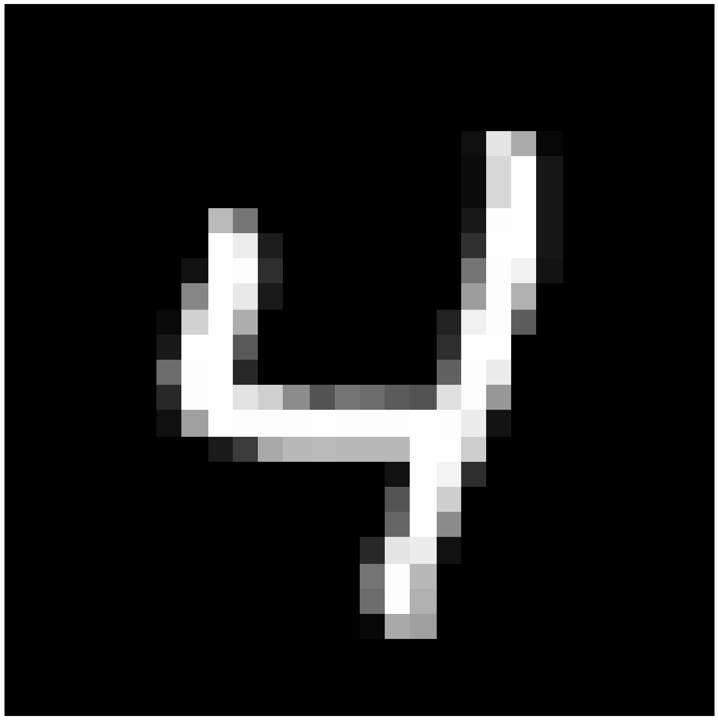
[7]:
```
##PCA and Autoencoder
```

[8]:
```
#first 30 eigenvectors corresponding to top 30 eigenvalues
pca_comps_n =30
print('PCA with', pca_comps_n ,'principal components')
train_data = np.asarray(train_images)/255
test_data = np.asarray(val_images)/255
total_data =  np.concatenate((train_data,test_data))
PCA_data = total_data.reshape(-1,28*28)
# PCA_data = train_data.reshape(-1,28*28)

pca1 = PCA(n_components = pca_comps_n)
pca1.fit(PCA_data)
train_pca = pca1.transform(PCA_data)
reconstructed_data = pca1.inverse_transform(train_pca)
PCA_error = mse(PCA_data,reconstructed_data)
print('Reconstruction error made by PCA: ',PCA_error)
```

```
PCA with 30 principal components
Reconstruction error made by PCA:  0.018056401844447451
```

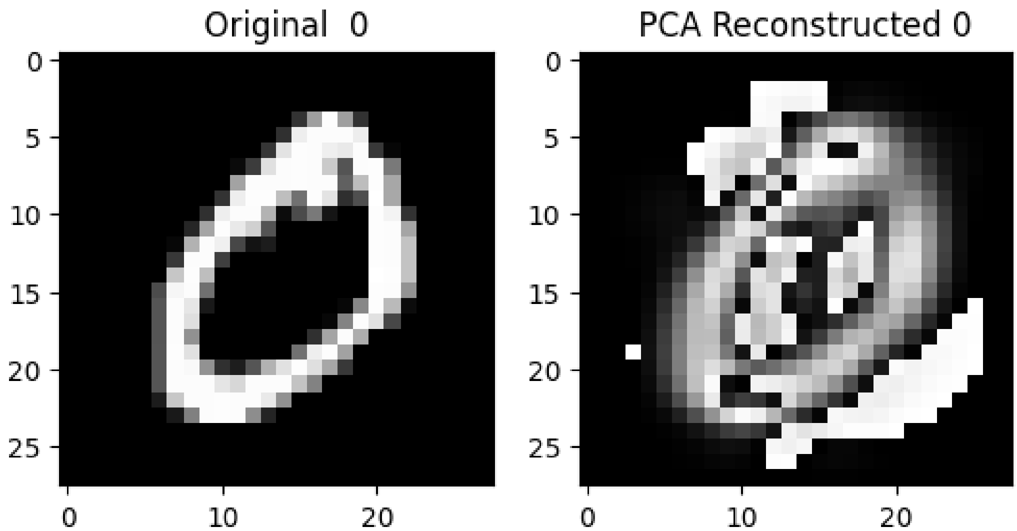
[9]:
```
#0-5,1-0,2-4,3-1,4-9,5-2,7-3,13-6,15-7,17-8
keys = [1,3,5,7,2,0,13,15,17,4]
```

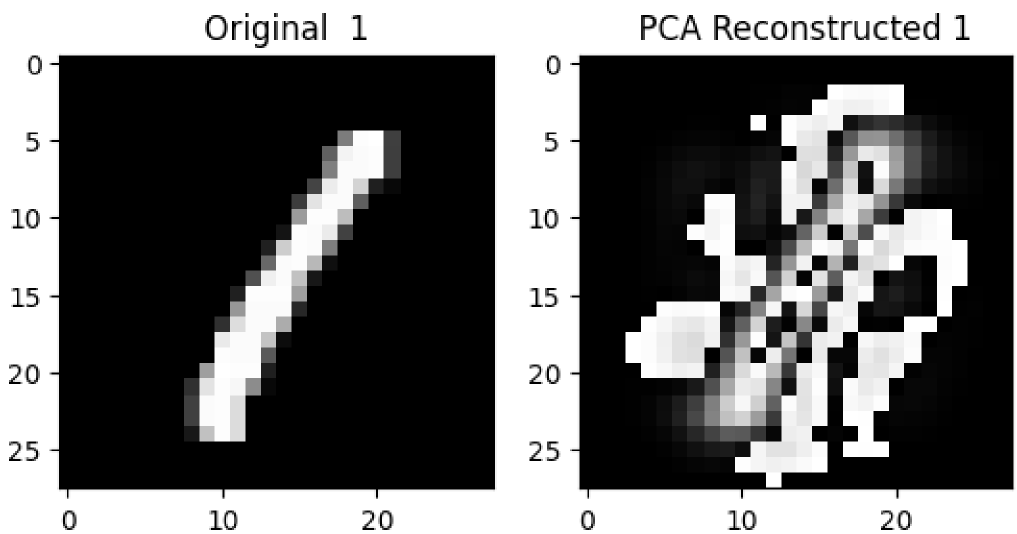
[10]:
```
for i in range(1, cols * rows + 1):
    sample_idx = torch.randint(len(train_dataset), size=(1,)).item()
    img, label = train_dataset[sample_idx]
    figure.add_subplot(rows, cols, i)
    plt.title(labels_map[label])
    plt.axis("off")
    plt.imshow(img.squeeze(), cmap="gray")
plt.show()
```

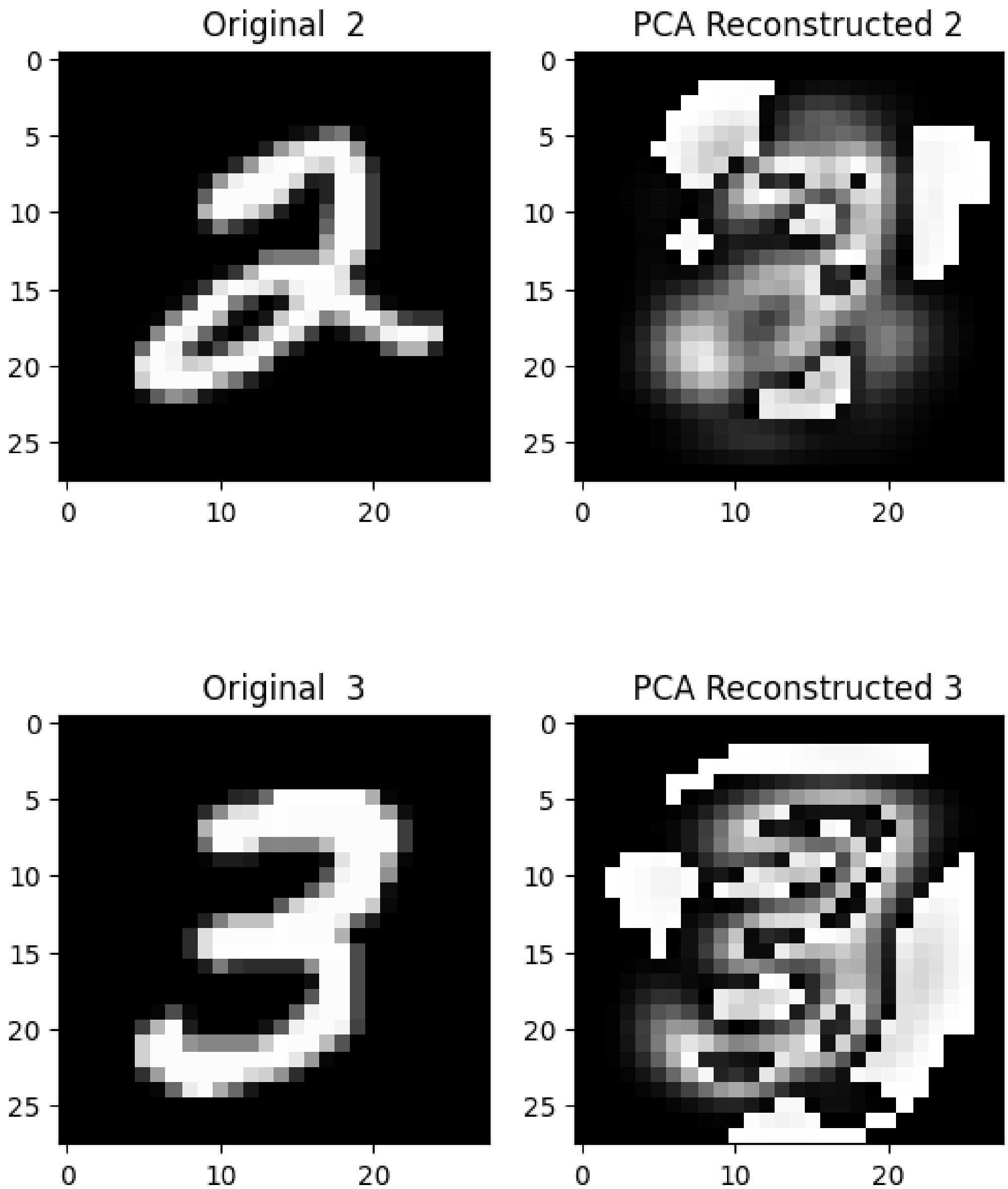


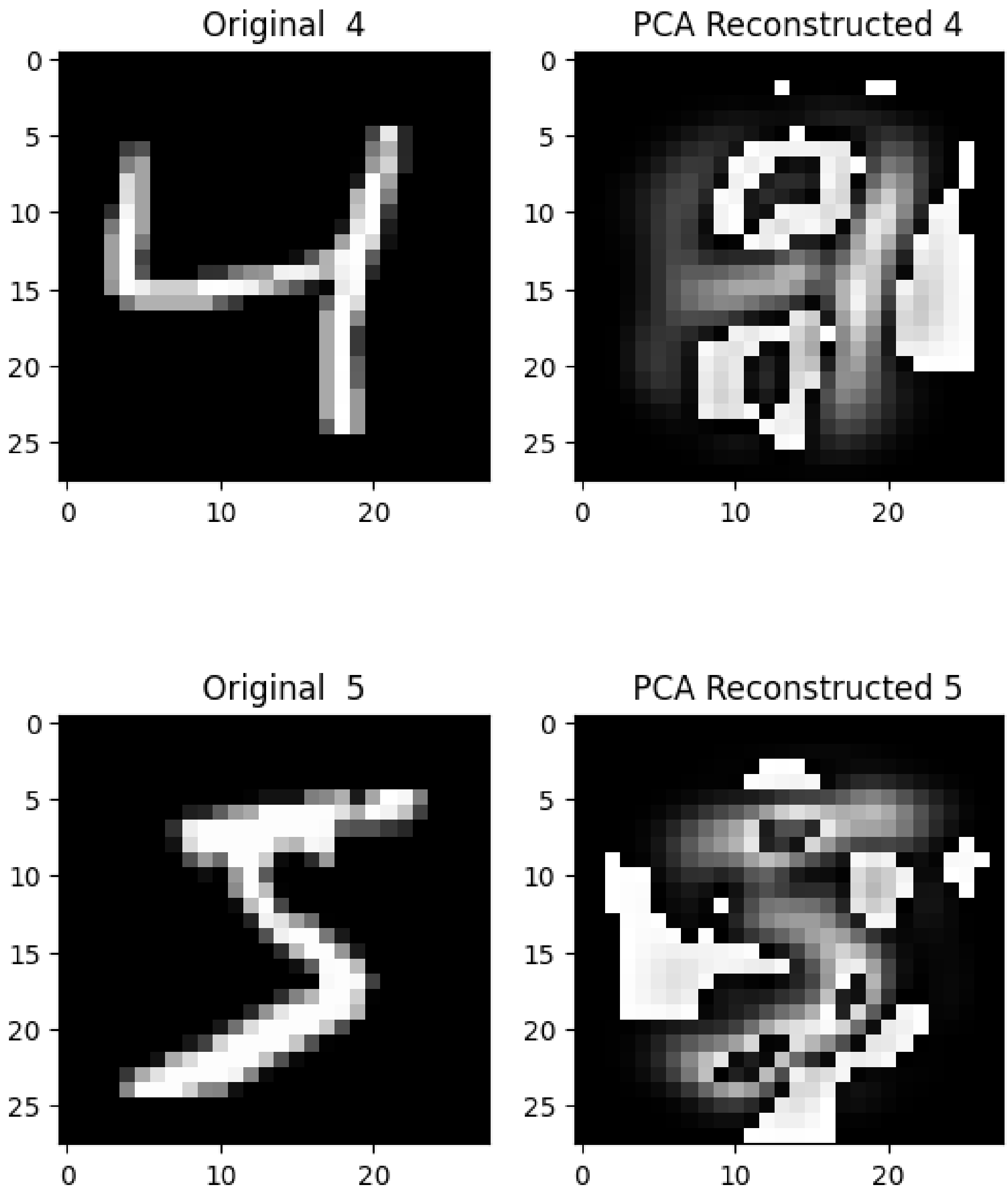
[11]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

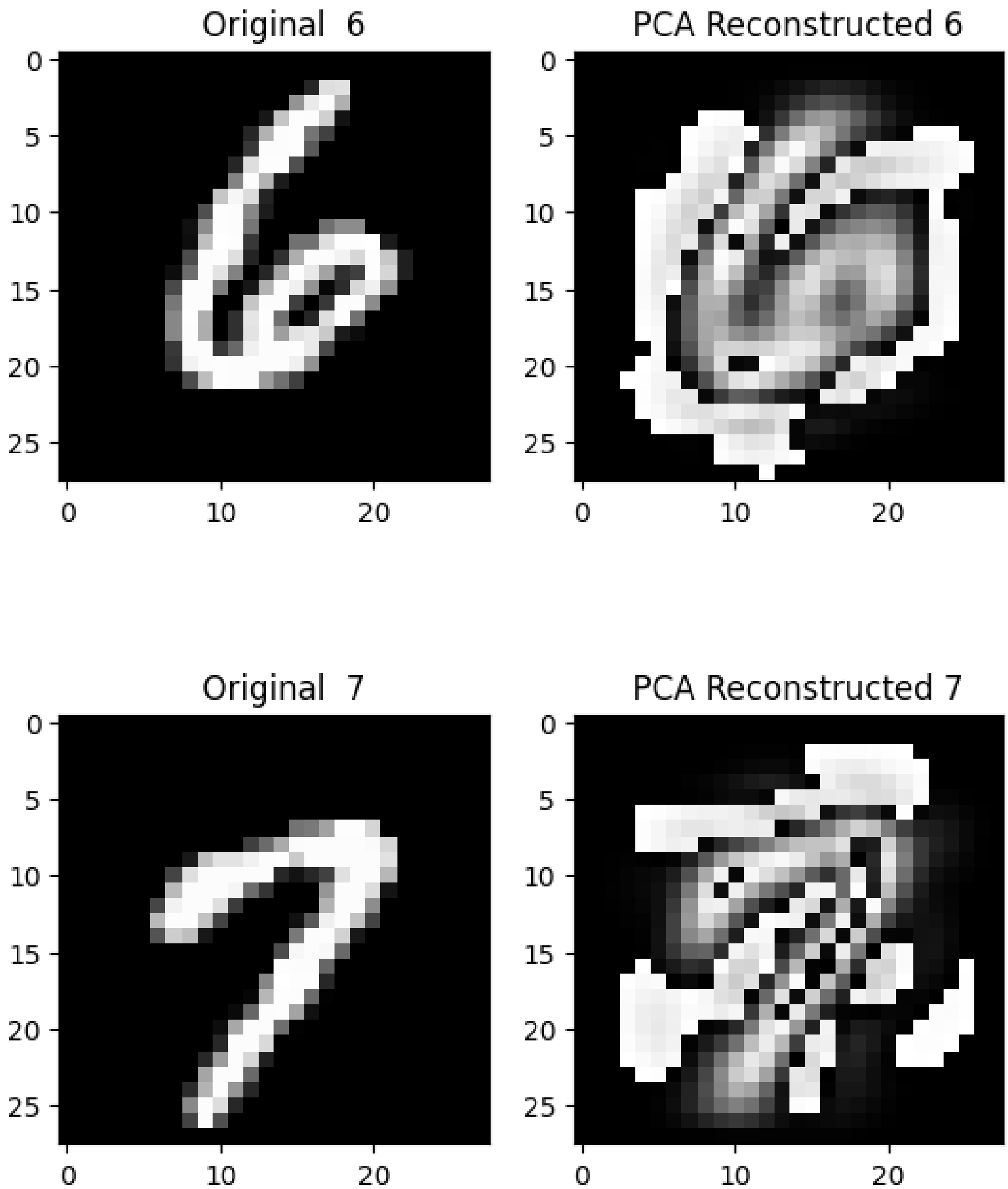
[12]:
```
from numpy import uint8

for t,j in zip(range(10),keys):
    input_image =  np.asarray(255*PCA_data[j],dtype=uint8).reshape(28,28)
    reconstruct_image = np.asarray(255*reconstructed_data[j],dtype=uint8).
 ↪reshape(28,28)
    plt.subplot(1,2,1)

    plt.imshow(input_image,'gray')
    plt.title('  Original  '+str(t))
    plt.subplot(1,2,2)

    plt.imshow(reconstruct_image,'gray')
    plt.title('  PCA Reconstructed '+str(t))
    plt.show()
```

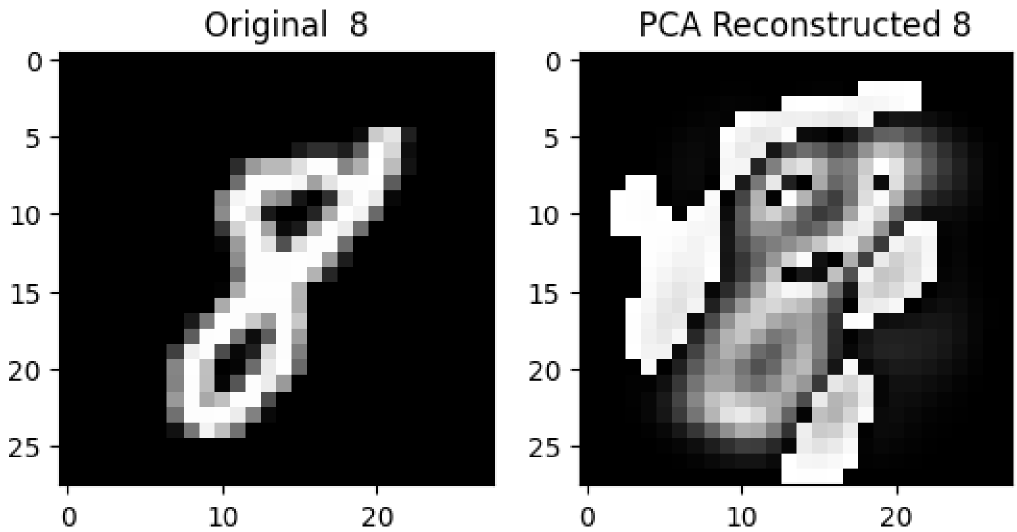

Original 0
PCA Reconstructed 0
0
5
10
15
20
25
0
10
20

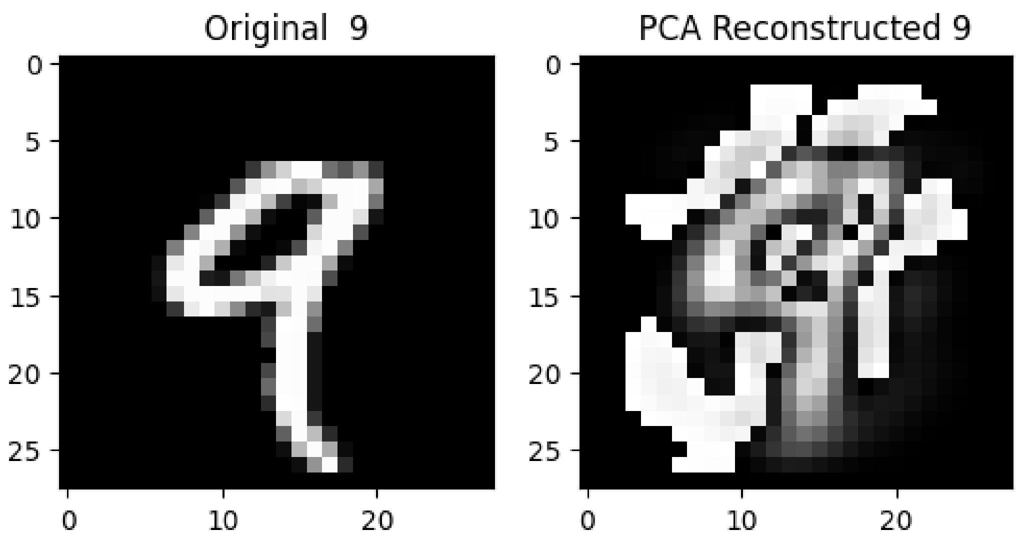

Original 1
PCA Reconstructed 1
0
5
10
15
20
25
0
10
20


Original 2
PCA Reconstructed 2
Original 3
PCA Reconstructed 3
0
5
10
15
20
25
0
10
20


Original 4
PCA Reconstructed 4
Original 5
PCA Reconstructed 5


Original 6
PCA Reconstructed 6
Original 7
PCA Reconstructed 7





[13]:
```python
from torch import flatten
```

[14]:
```python
class AE1(nn.Module):
  def __init__(self):
    super(AE1, self).__init__()
    self.encoder = nn.Sequential(
        nn.Linear(784,512),
        nn.ReLU(),
        nn.Linear(512,256),
        nn.ReLU(),
```

```
        nn.Linear(256,128),
        nn.ReLU(),
        nn.Linear(128,30),
        nn.ReLU())
    self.decoder =nn.Sequential(
        nn.Linear(30,128),
        nn.ReLU(),
        nn.Linear(128,256),
        nn.ReLU(),
        nn.Linear(256,784),
        nn.ReLU())

  def forward(self,x):
    x = flatten(x,1)
    encoded=self.encoder(x.float())
    reconstructed =self.decoder(encoded)
    return reconstructed    , encoded
```

[15]:
```
# Loss function
lossfn = nn.MSELoss()
```

[16]:
```
from torch import flatten
def␣
↪train(model,device,train_dataloader,optimizer,lossfn,lambda_reg=0,sparse=False,denoise=Fals
↪3, q5_flag = False):
    model.train() #setting the model in training mode
    #initializing the total training loss to 0
    train_loss    = 0
    #loop over the training set
    for batch_idx, (data,label) in enumerate(train_dataloader):  # (data,label):
↪ Training data for that batch
        if denoise==True:
            img = data.clone()
            data = add_noise(img,noise_val)
            data = data.to(device)
        else:
            (data,label) = (data.to(device),label.to(device))  #sending the␣
↪data to the device we've chosen

        reconstruction,encoded = model(data) #our reconstruction
        if q5_flag==True:
            loss = lossfn(reconstruction,data) #loss
        else:
            loss = lossfn(reconstruction,flatten(data,1)) #loss
        if sparse==True:
            loss += lambda_reg*torch.linalg.norm(encoded,1)
        optimizer.zero_grad() #zeroing out the gradients before backprop
```

```
        loss.backward()       #backprop from the loss
        optimizer.step()      #updating the weights
        train_loss    += loss/len(train_dataloader)
    return train_loss #returning loss
```

[17]:
```
def␣
↪test(model,device,test_dataloader,lossfn,lambda_reg=0,sparse=False,denoise=False,noise_val=
↪3, q5_flag = False):
    model.eval()
    test_loss    = 0
    with torch.no_grad():
        for (data,label) in test_dataloader: # (data,label): Test data for that␣
↪batch
            if denoise==True:
                img = data.clone()
                data = add_noise(img,noise_val)
                data = data.to(device)
            else:
                (data,label) = (data.to(device),label.to(device))  #sending the␣
↪data to the device we've chosen
            #perform forward pass and compute the loss
            reconstruction,encoded = model(data) #our prediction
            if q5_flag==True:
                loss = lossfn(reconstruction,data) #loss
            else:
                 loss = lossfn(reconstruction,flatten(data,1)) #loss
            if sparse==True:
                loss += lambda_reg*torch.linalg.norm(encoded,1)
            test_loss    += loss/len(test_dataloader)
    return test_loss #returning loss
```

[18]:
```
def␣
↪train_test(model,device,train_loader,test_loader,optimizer,lossfn,lambda_reg=0,sparse=False
↪3,q5_flag=False):
    train_losses   = []
    test_losses    = []
    for epoch in range(epochs+1):
        #train the model
        train_loss =␣
↪train(model,device,train_loader,optimizer,lossfn,lambda_reg,sparse,denoise,noise_val,q5_fla
        train_losses.append(train_loss.item())
        #test the model
        test_loss =␣
↪test(model,device,test_loader,lossfn,lambda_reg,sparse,denoise,noise_val,q5_flag)
        test_losses.append(test_loss.item())
    return train_losses, test_losses
```

[19]:
```
def plot_reconstructed_image(model,device,img,model_name,q5_flag=False):
#     img = torch.from_numpy(img)
    with torch.no_grad():
        if q5_flag == False:
            if (device==torch.device("cuda")):
                img = img.view(-1,28,28).cuda().float()
            else:
                img = img.view(-1,28,28).float()
        else:
            if (device==torch.device("cuda")):
                img = img.reshape(1,1,28,28).cuda().float()
            else:
                img = img.reshape(1,1,28,28).float()
        reconstructed_image,encoded = model.forward(img) #as it is a single␣
 ↪image we directly run the forward pass
        reconstructed_image=reconstructed_image.detach().cpu().numpy()
        img = img.reshape(28,28).detach().cpu().numpy()
        plt.subplot(1,2,1)
        plt.imshow(img , cmap='gray')
        plt.title('input image')
        plt.subplots_adjust(right=1.5)
        plt.subplot(1,2,2)
        plt.imshow(reconstructed_image.reshape(28,28),cmap ='gray') #our␣
 ↪reconstructed image
        plt.title("Reconstructed image using "+ str(model_name))
        plt.show()
```

[20]:
```
def plot_losses(train_losses, test_losses, model_name):
    train_interval = int(len(train_losses)/epochs)
    plt.plot(np.asarray(train_losses)[::train_interval])
    plt.title("MSE train loss using "+ str(model_name))
    plt.xlabel("Epoch")
    plt.ylabel("Loss")
    plt.show()

    test_interval = int(len(test_losses)/epochs)
    plt.plot(np.asarray(test_losses)[::test_interval])
    plt.title("MSE test loss using "+ str(model_name))
    plt.xlabel("Epoch")
    plt.ylabel("loss")
    plt.show()
```

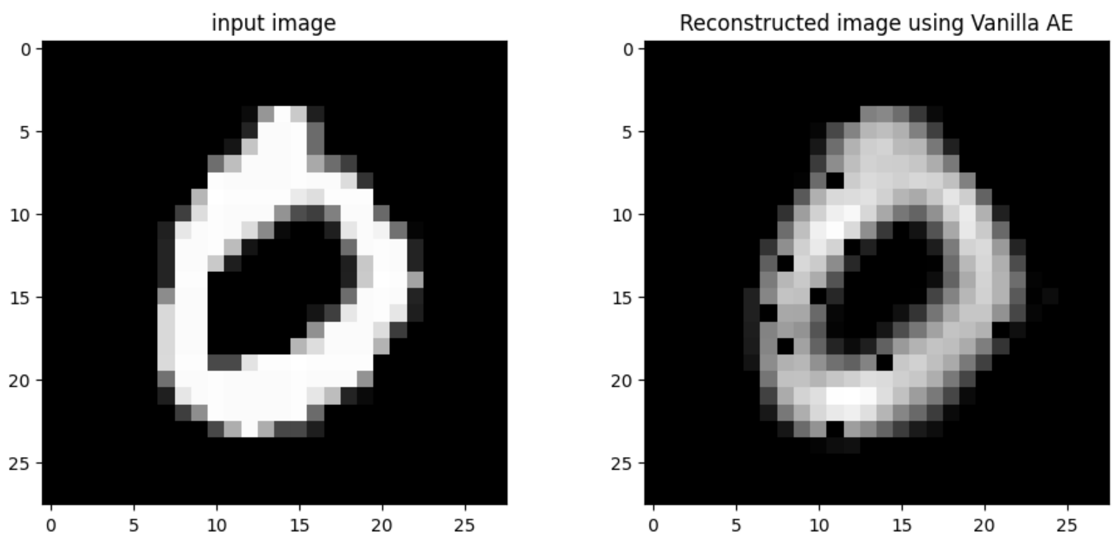
[21]:
```
model_Q1 = AE1().to(device)
optimizer = torch.optim.Adam(model_Q1.parameters(), lr=learning_rate)
train_losses_AE_Q1 , test_losses_AE_Q1 =␣
 ↪train_test(model_Q1,device,train_loader,test_loader,optimizer,lossfn)
```

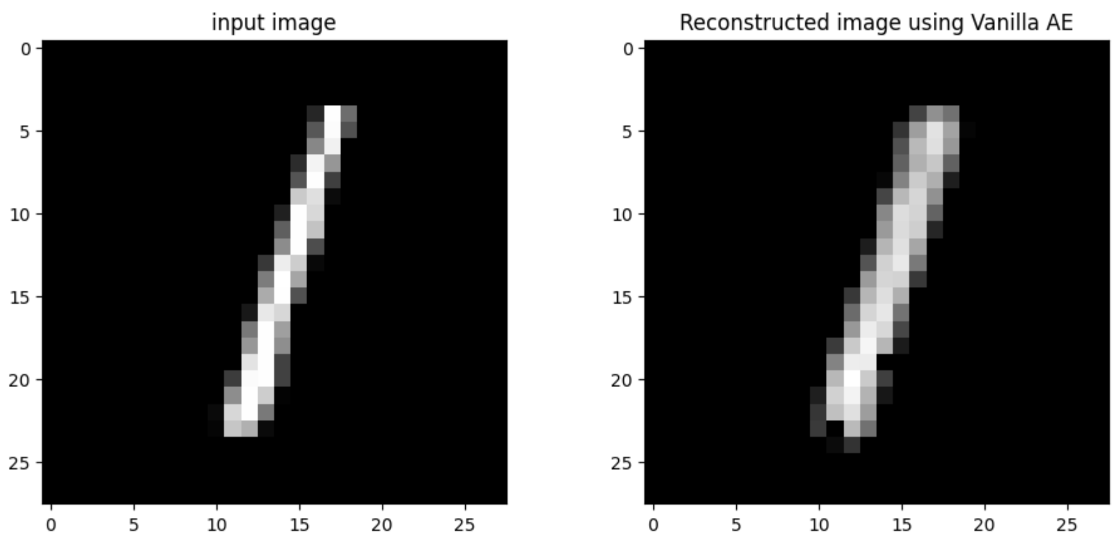
[22]:
```
'''
0-7
1-2
2-1
3-0
4-4
7-9
8-5
11-6
18-3
84-8
'''
```

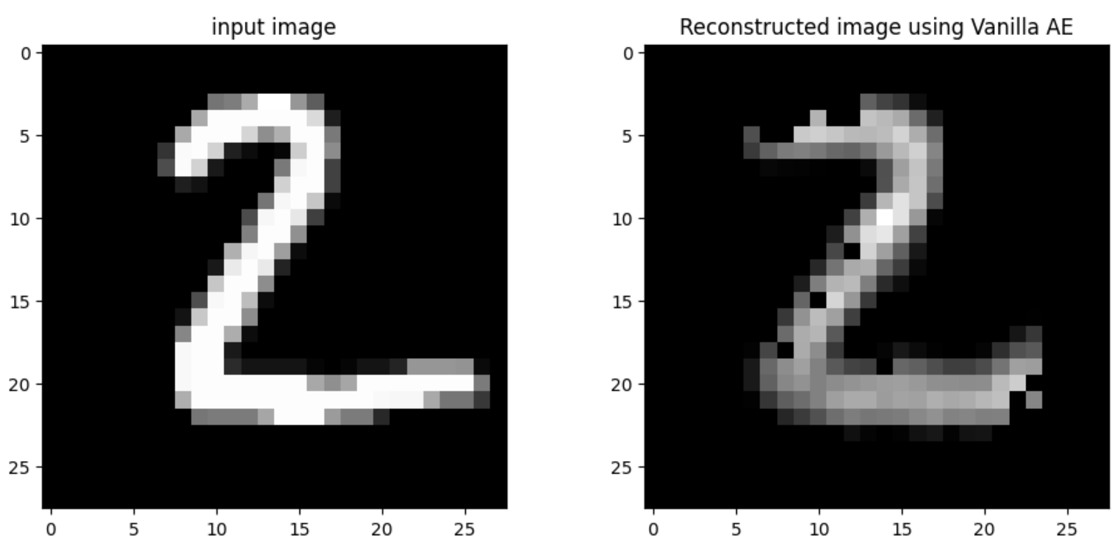
[22]: '\n0-7\n1-2\n2-1\n3-0\n4-4\n7-9\n8-5\n11-6\n18-3\n84-8\n'

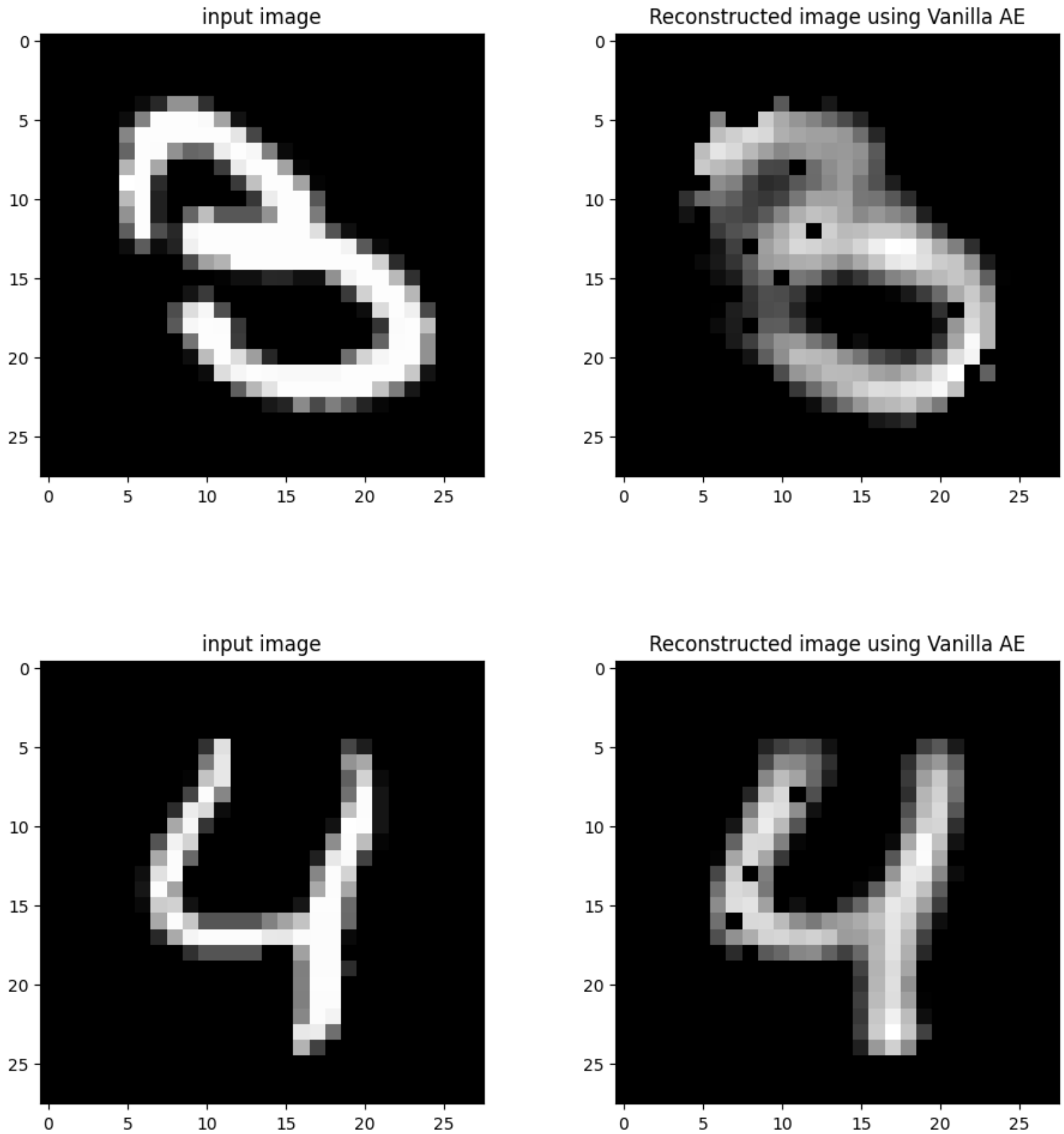
[23]:
```
keys_test=[3,2,1,18,4,8,11,0,84,7]
```

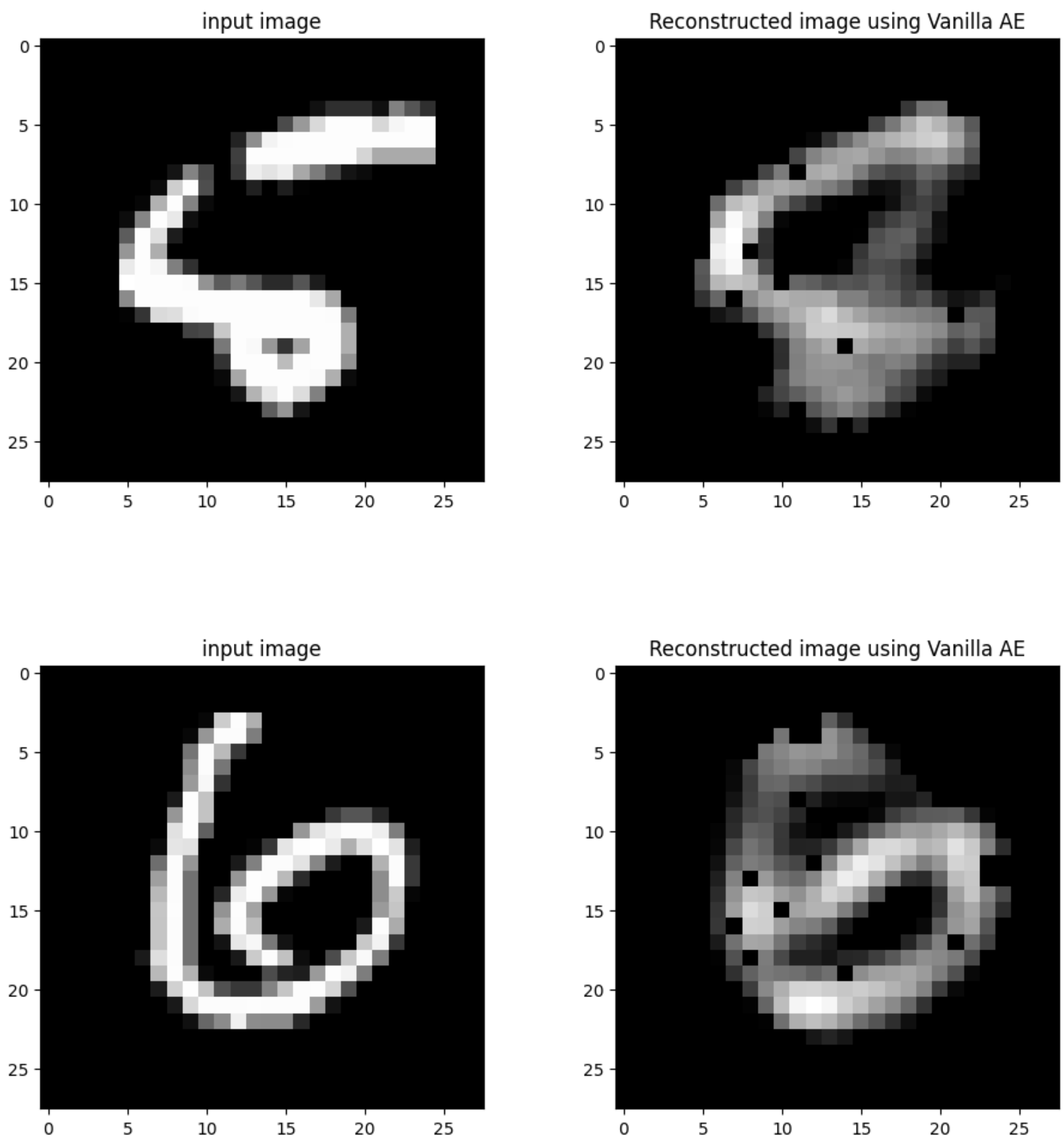
[24]:
```
import random
# index = random.randint(0,9999)
for i in keys_test:
    test_image = test_loader.dataset.data[i, :, :].clone()
    plot_reconstructed_image(model_Q1,device,test_image, model_name="Vanilla␣
 ↪AE")
# print(type(test_image))
```

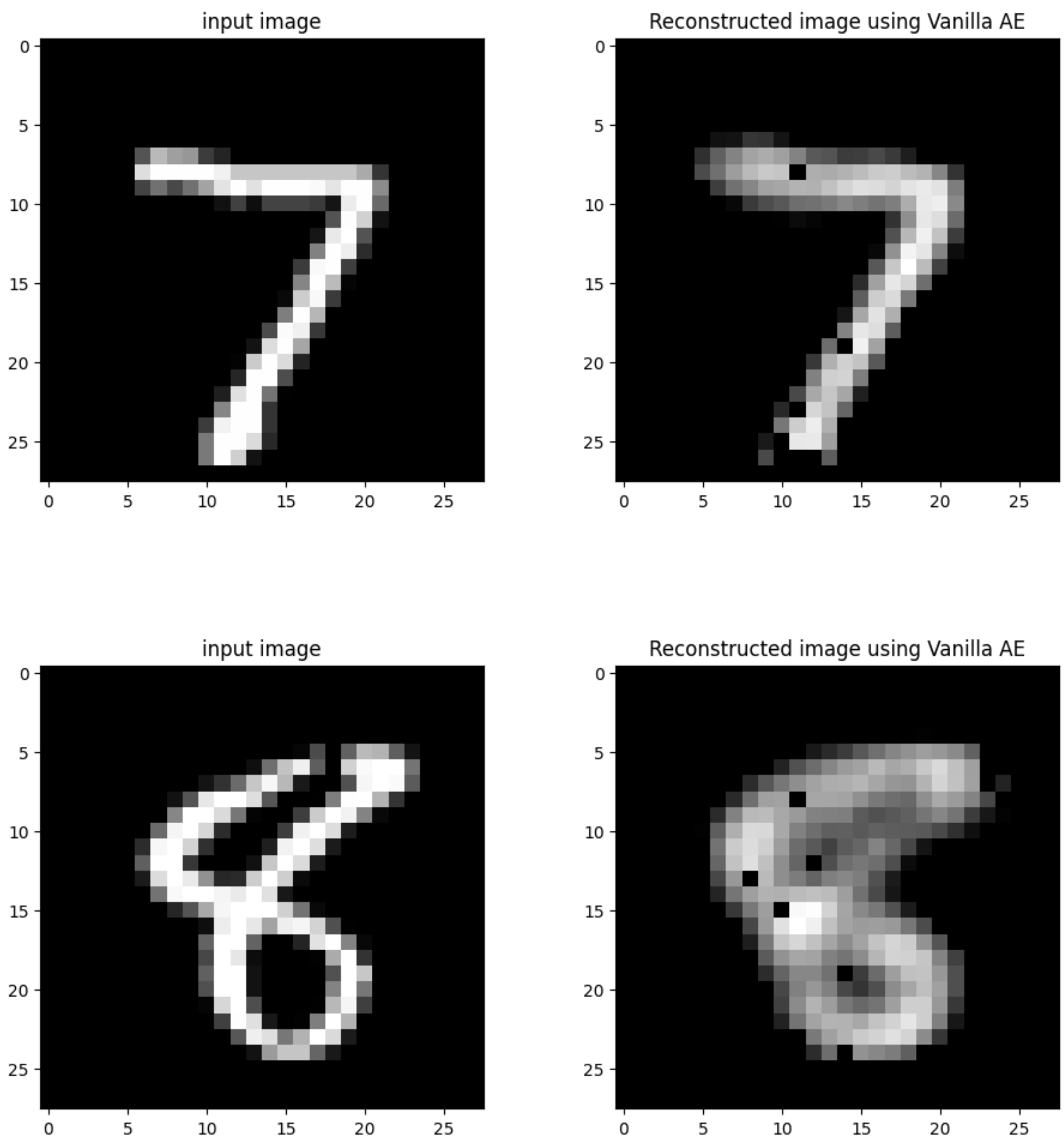


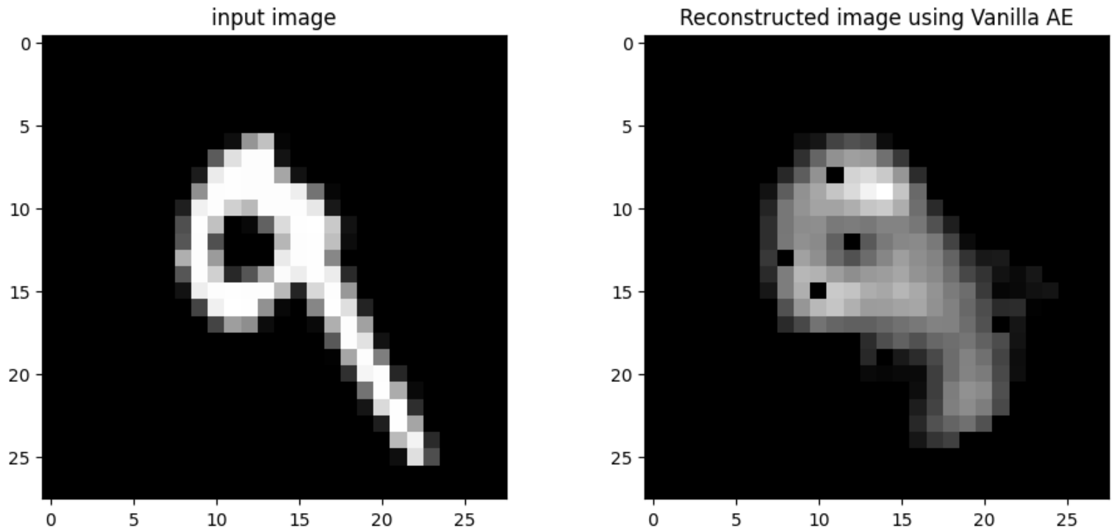

input image
Reconstructed image using Vanilla AE
0
5
10
15
20
25


input image
Reconstructed image using Vanilla AE
0
5
10
15
20
25


input image
Reconstructed image using Vanilla AE
0
5
10
15
20
25
input image
Reconstructed image using Vanilla AE

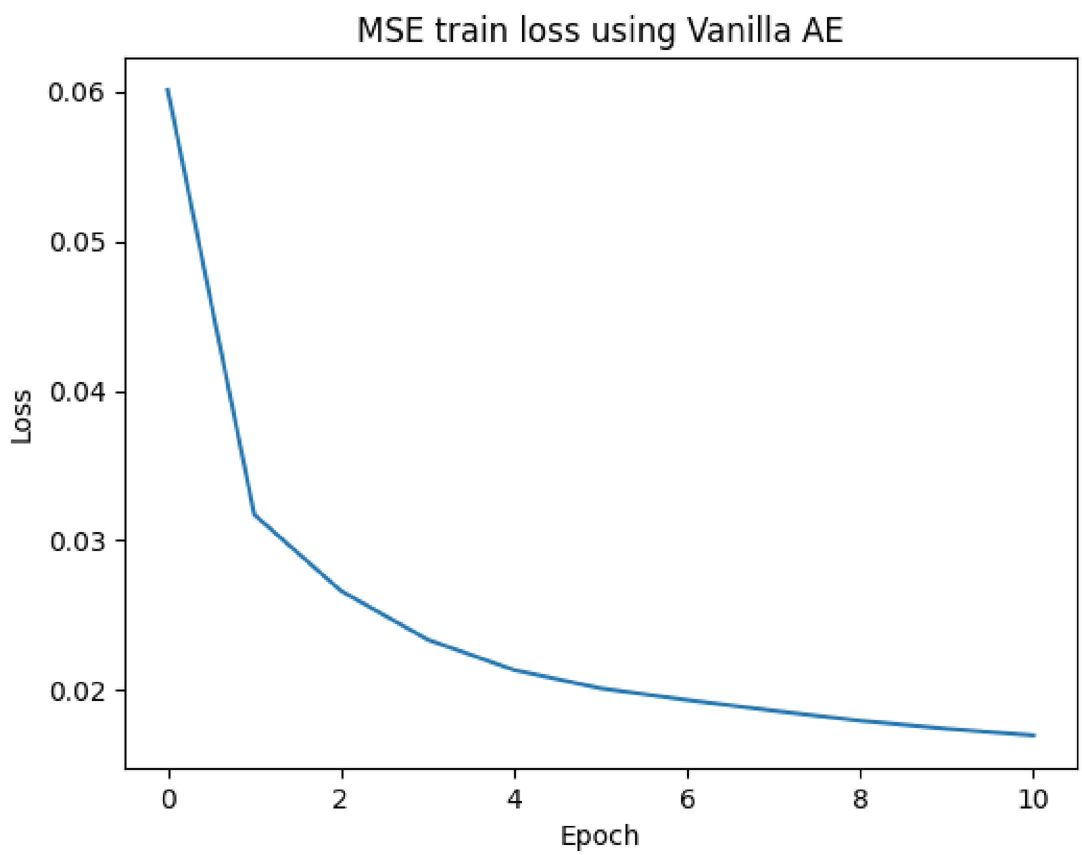

input image
Reconstructed image using Vanilla AE
input image
Reconstructed image using Vanilla AE

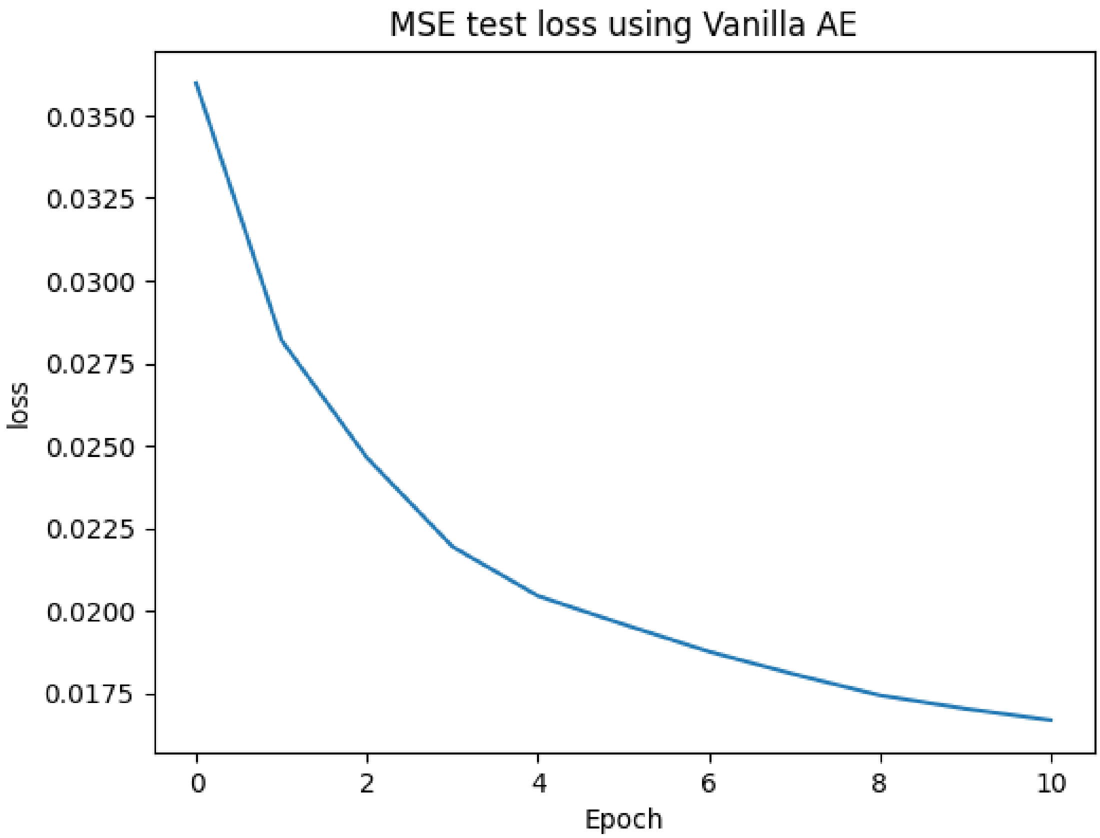

input image
Reconstructed image using Vanilla AE
input image
Reconstructed image using Vanilla AE



[25]:

```
plot_losses(train_losses_AE_Q1, test_losses_AE_Q1, model_name = "Vanilla AE")
```





[26]:
```
#MSE recomnstruction error for vanilla AE
mse_error = test(model_Q1,device,test_loader,lossfn)
print("MSE Reconstruction error for Vanilla AE is ", mse_error.item())
```

```
MSE Reconstruction error for Vanilla AE is  0.016686905175447464
```

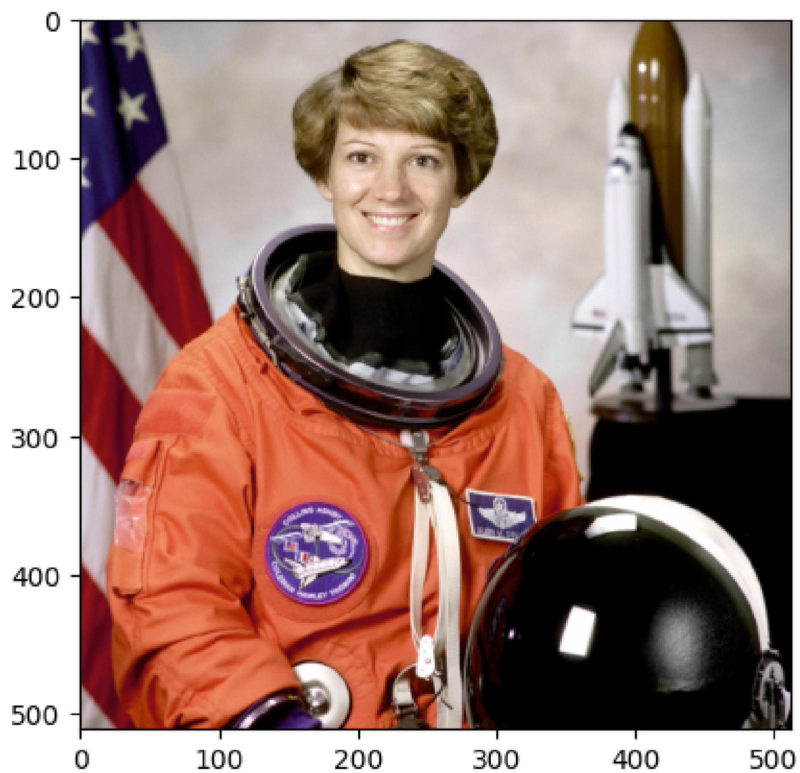
0.2 Q2

[27]:
```
class AE_Q2(nn.Module):
  def __init__(self,hidden_layer):
    super(AE_Q2, self).__init__()

    self.encoder = nn.Sequential(
        nn.Linear(784,hidden_layer),
        nn.ReLU())
    self.decoder =nn.Sequential(
        nn.Linear(hidden_layer,784),
        nn.ReLU())
```

```
  def forward(self,x):
    x = flatten(x,1)
    encoded=self.encoder(x.float())
    reconstructed =self.decoder(encoded)
    return reconstructed    , encoded
```

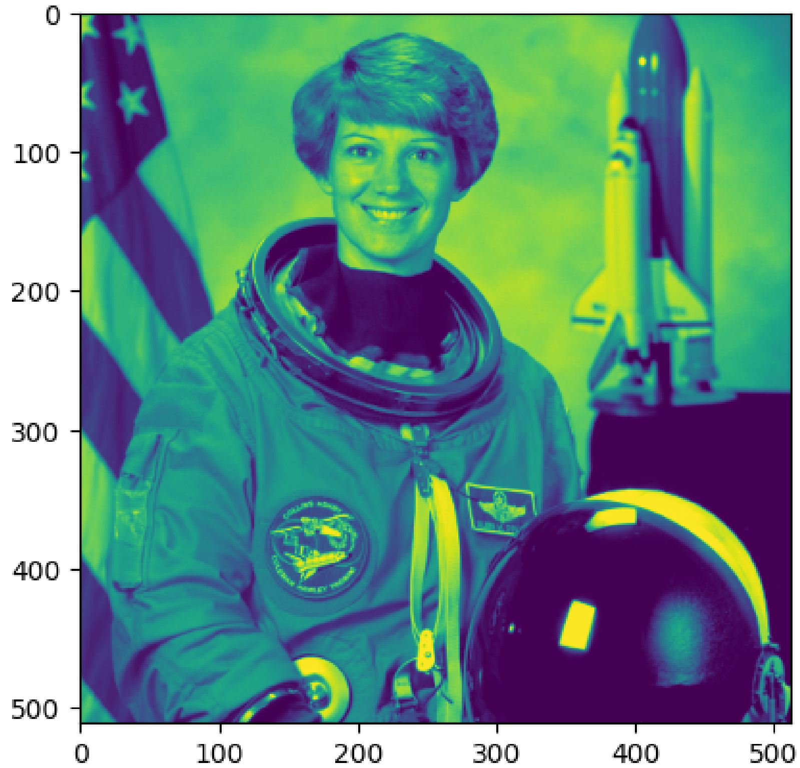
[28]:

```
from skimage import data
astronaut = data.astronaut()
plt.imshow(astronaut)
plt.show()
print(astronaut.shape)
from skimage.color import rgb2gray
grayscale_astro = rgb2gray(astronaut)
plt.imshow(grayscale_astro)
print(grayscale_astro.shape)
plt.show()
# flower = cv2.imread("flower.png")
# flower = cv2.cvtColor(flower, cv2.COLOR_BGR2GRAY)
# plt.imshow(flower)
```




```
(512, 512, 3)
```

(512, 512)

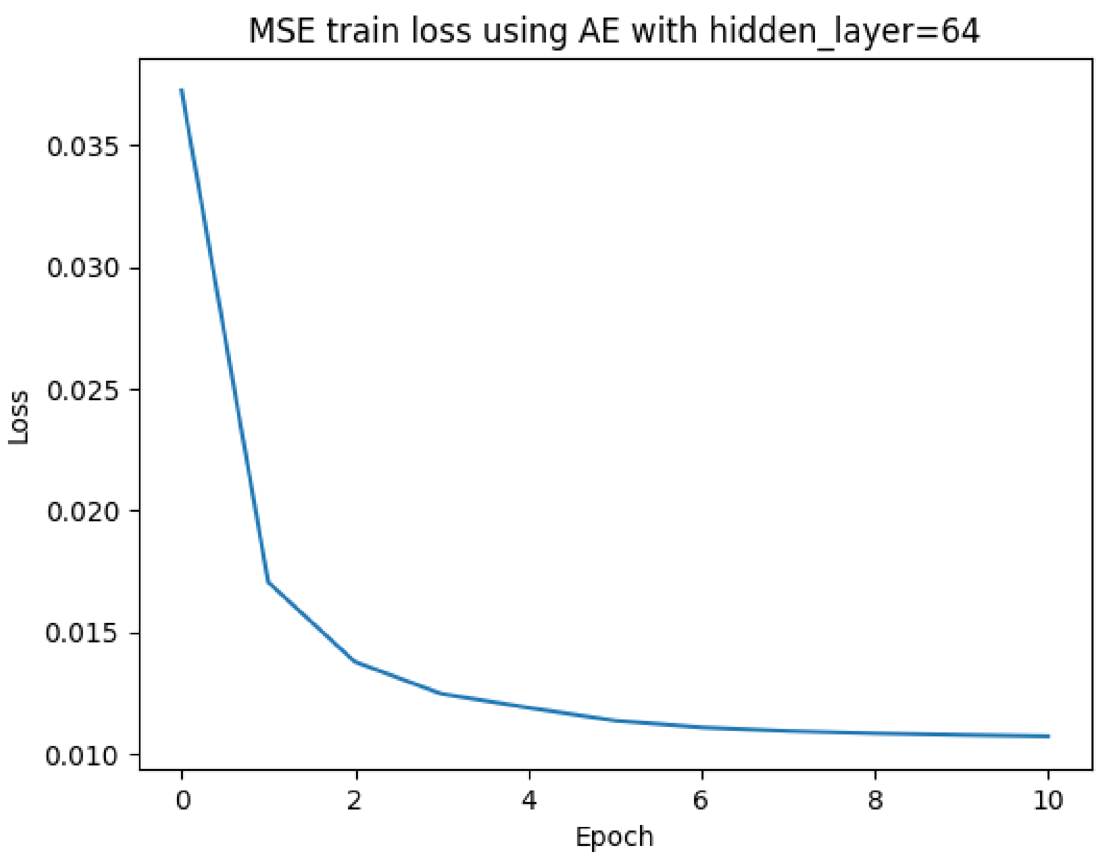


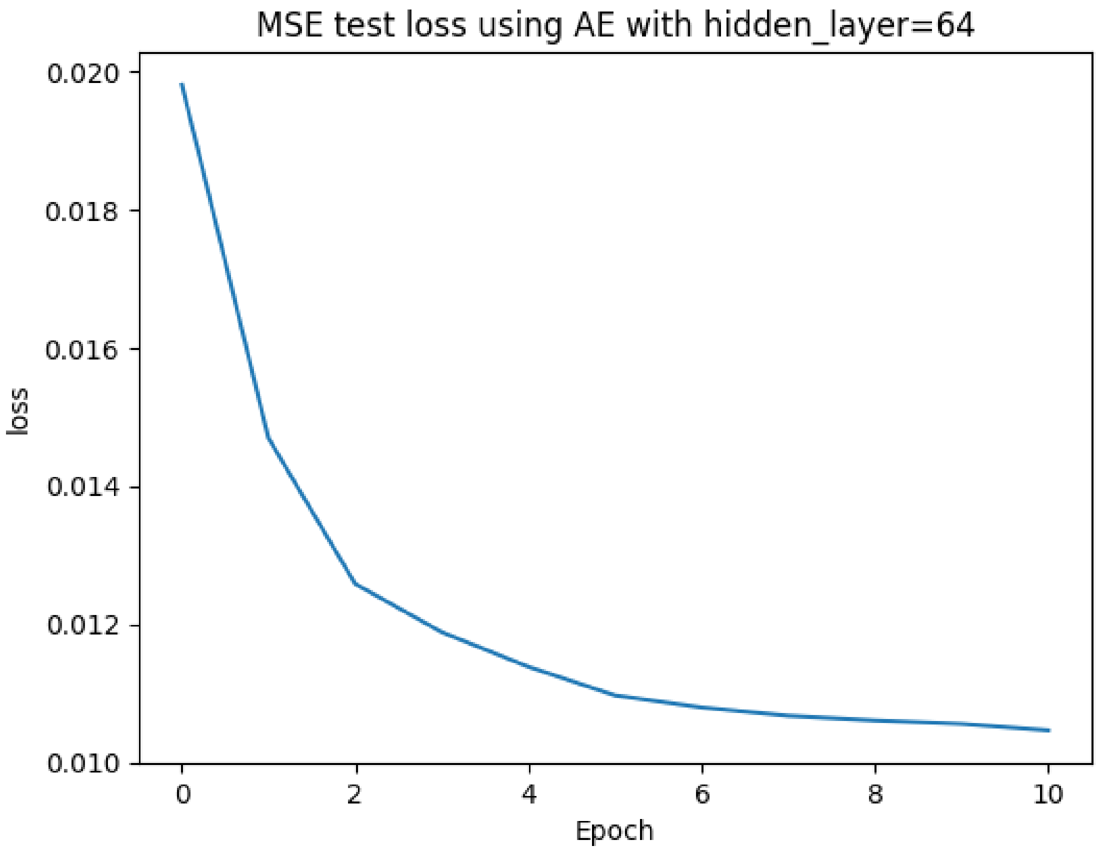
[29]:
```
import skimage.transform
from skimage import img_as_ubyte
grayscale_astro = np.asarray(grayscale_astro)
non_digit_image = torch.from_numpy(img_as_ubyte(skimage.transform.
 ↪resize(grayscale_astro, (28,28))))
```

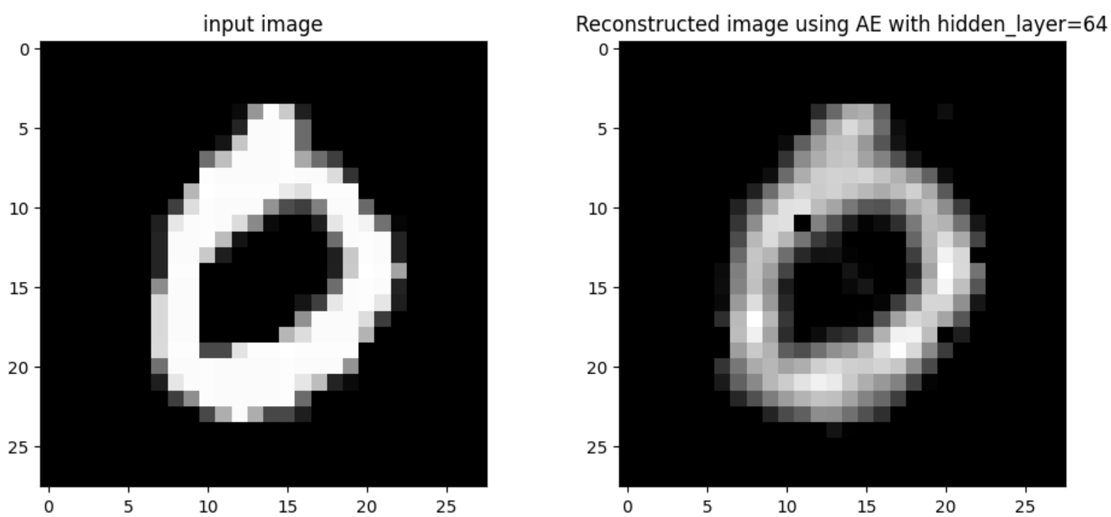
[30]:
```
X = np.random.normal(loc=128,scale=10,size=(28,28)) #Initializing a 28x28␣
 ↪matrix with gaussian noise 1ith mean = 128, std=10
#converting X to Tensor
if(device==torch.device("cuda")):
    noisy_image = torch.from_numpy(X).reshape(1,1,28,28).cuda().float()
else:
    noisy_image = torch.from_numpy(X).reshape(1,1,28,28).float()
```

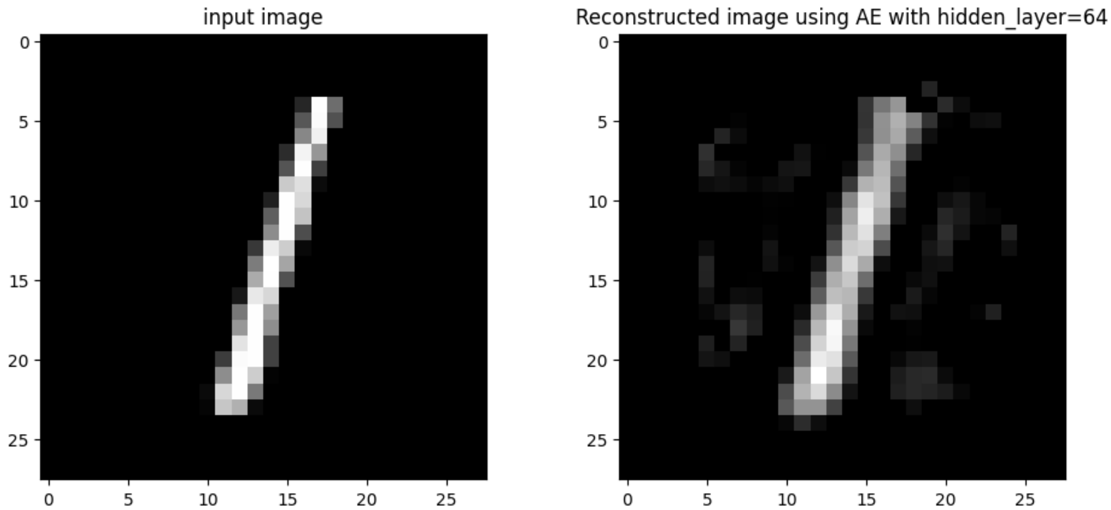
[31]:
```
x = [64,128,256] #size of the hidden layer
for hidden_layer in x:
    model_Q2 = AE_Q2(hidden_layer=hidden_layer).to(device)
    optimizer = torch.optim.Adam(model_Q2.parameters(), lr=learning_rate)
    train_losses_AE_h , test_losses_AE_h =␣
 ↪train_test(model_Q2,device,train_loader,test_loader,optimizer,lossfn)
```

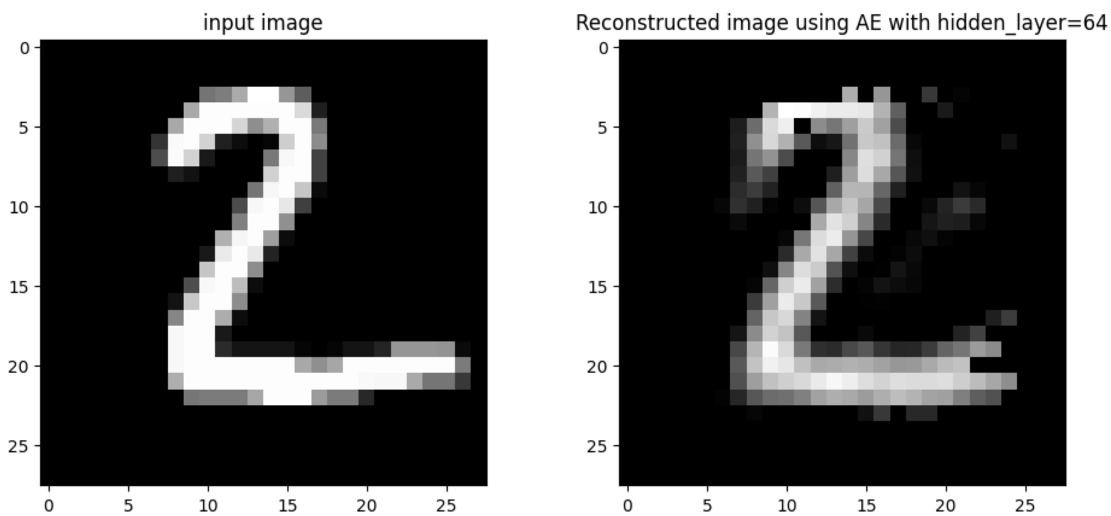
```
    plot_losses(train_losses_AE_h, test_losses_AE_h, model_name = "AE with␣
↪hidden_layer="+str(hidden_layer))
    for i in keys_test:
        test_image = test_loader.dataset.data[i, :, :].clone()
        plot_reconstructed_image(model_Q2,device,test_image, model_name="AE␣
↪with hidden_layer="+str(hidden_layer))
    plot_reconstructed_image(model_Q2,device,non_digit_image, model_name="AE␣
↪with hidden_layer="+str(hidden_layer)+"with non digit image")
    plot_reconstructed_image(model_Q2,device,noisy_image, model_name="AE with␣
↪hidden_layer="+str(hidden_layer)+"with noisy image")
```

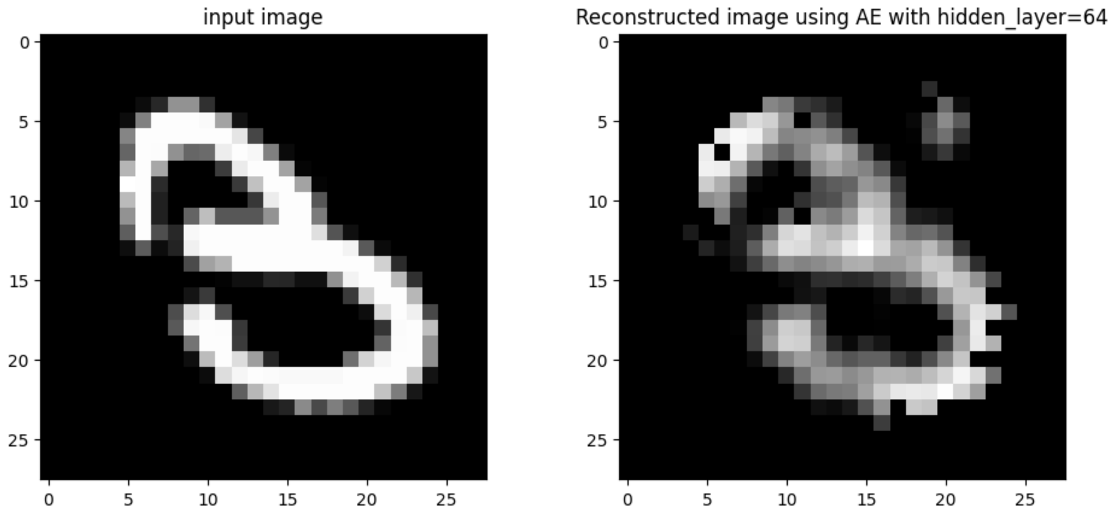


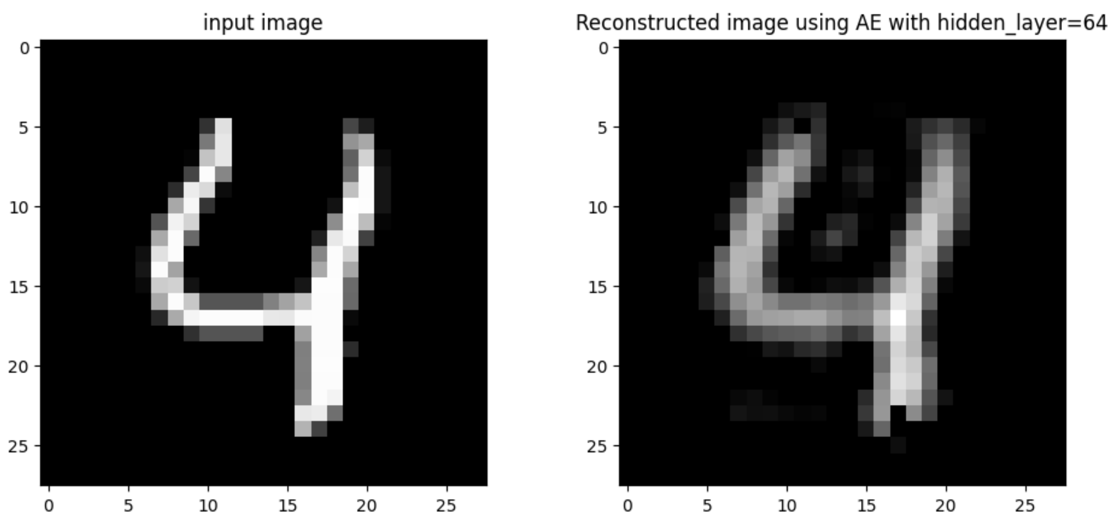

MSE test loss using AE with hidden_layer=64
0.020
0.018
0.016
0.014
0.012
0.010
loss
0
2
4
6
8
10
Epoch

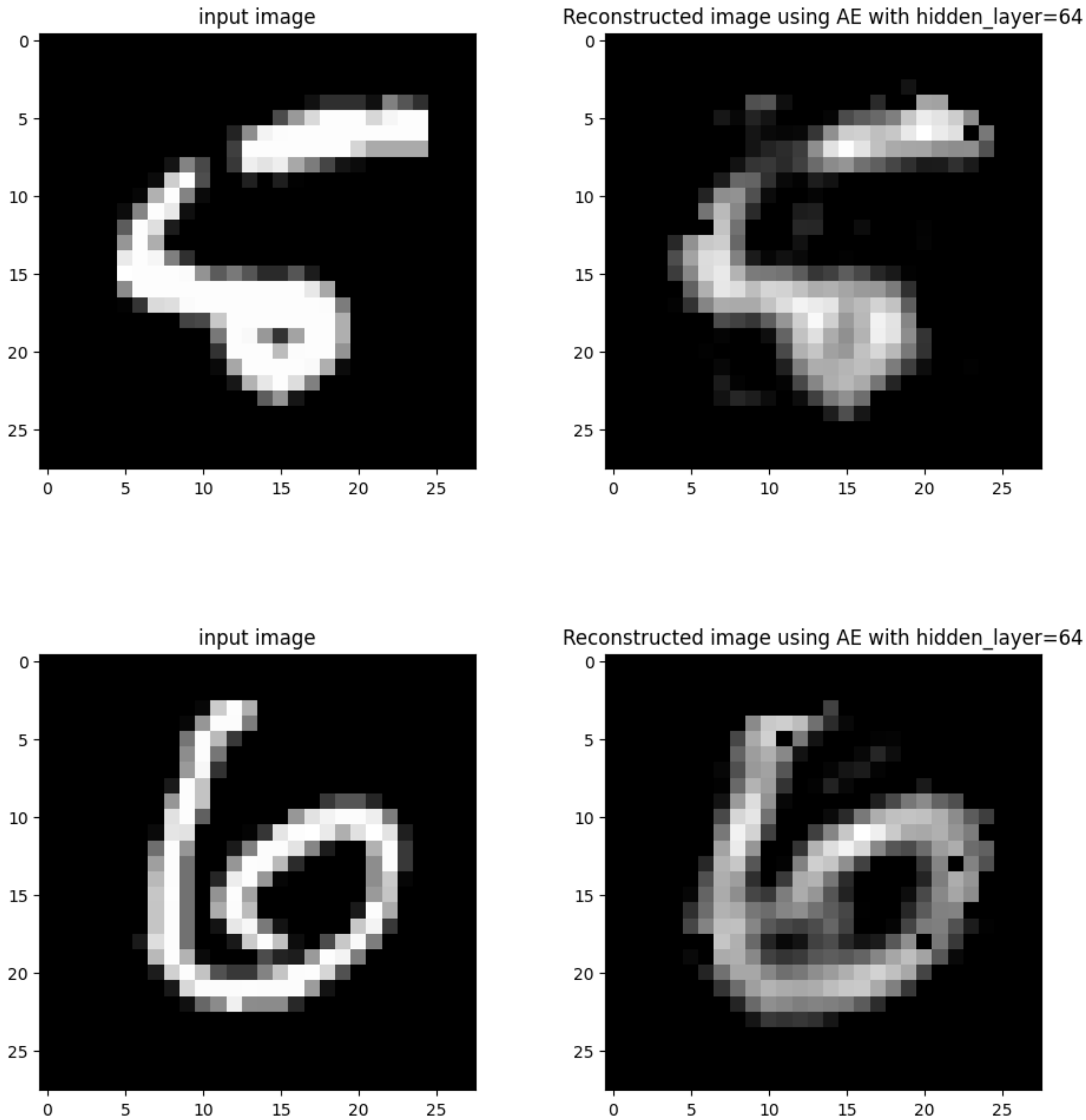

input image
Reconstructed image using AE with hidden_layer=64
0
5
10
15
20
25

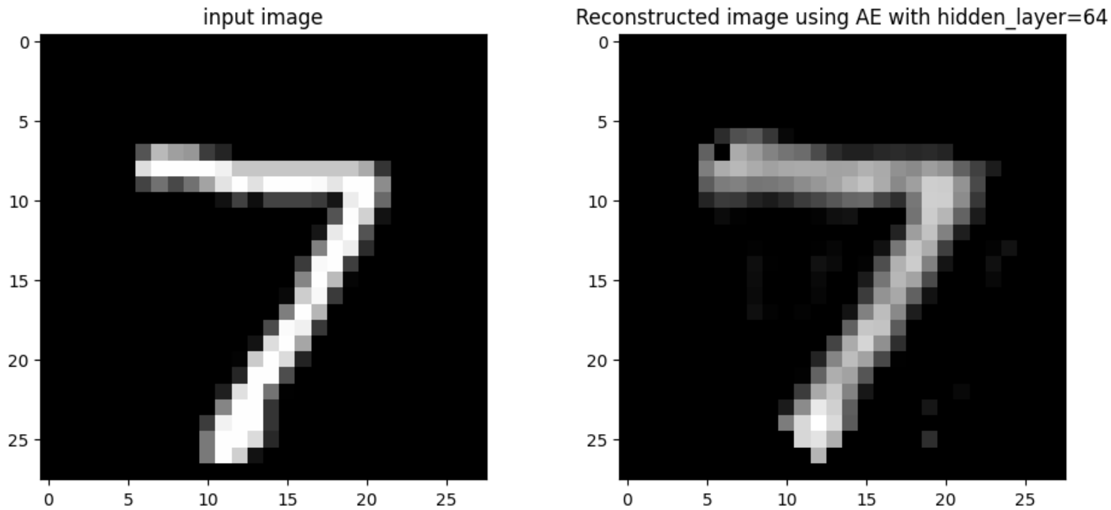

input image
Reconstructed image using AE with hidden_layer=64

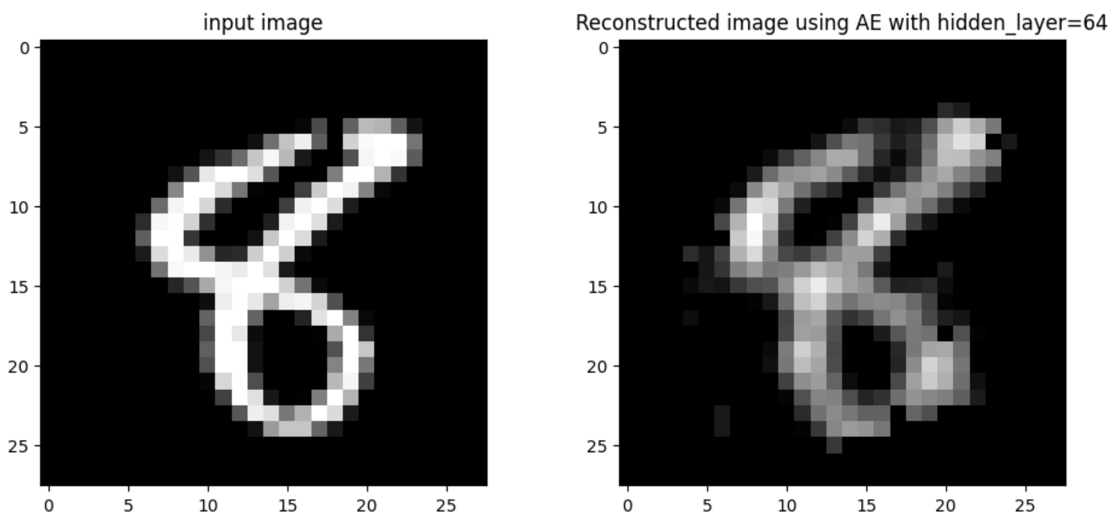

input image
Reconstructed image using AE with hidden_layer=64

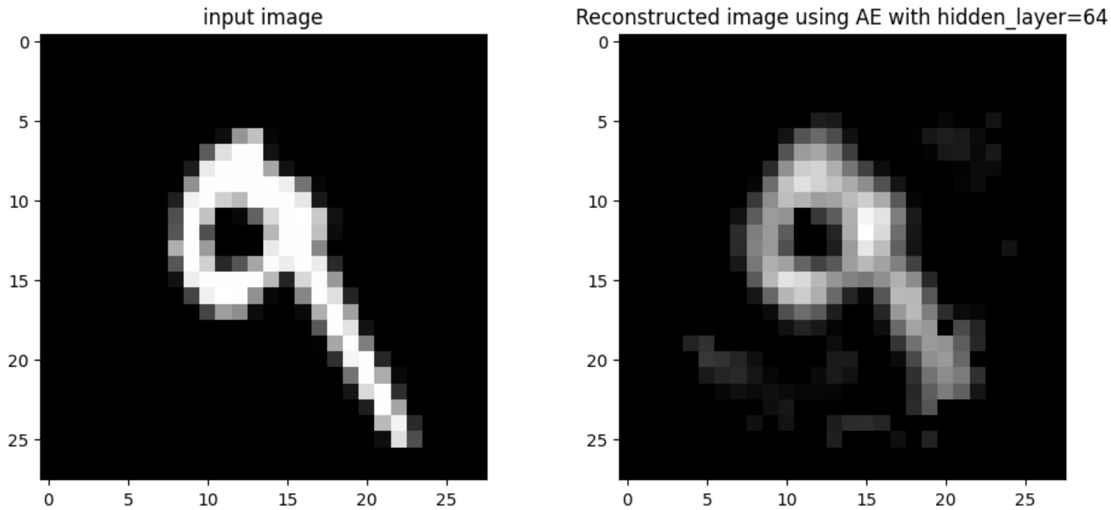

input image
Reconstructed image using AE with hidden_layer=64

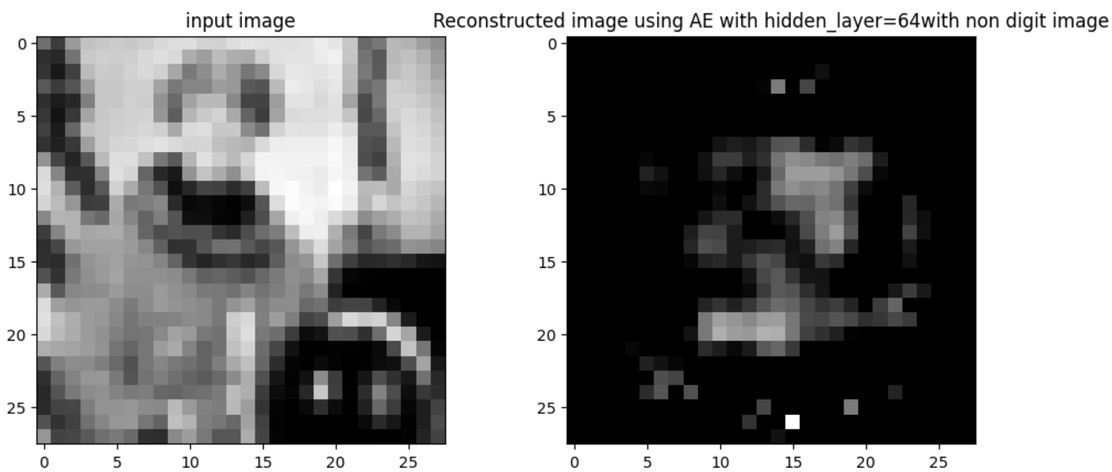

input image
Reconstructed image using AE with hidden_layer=64

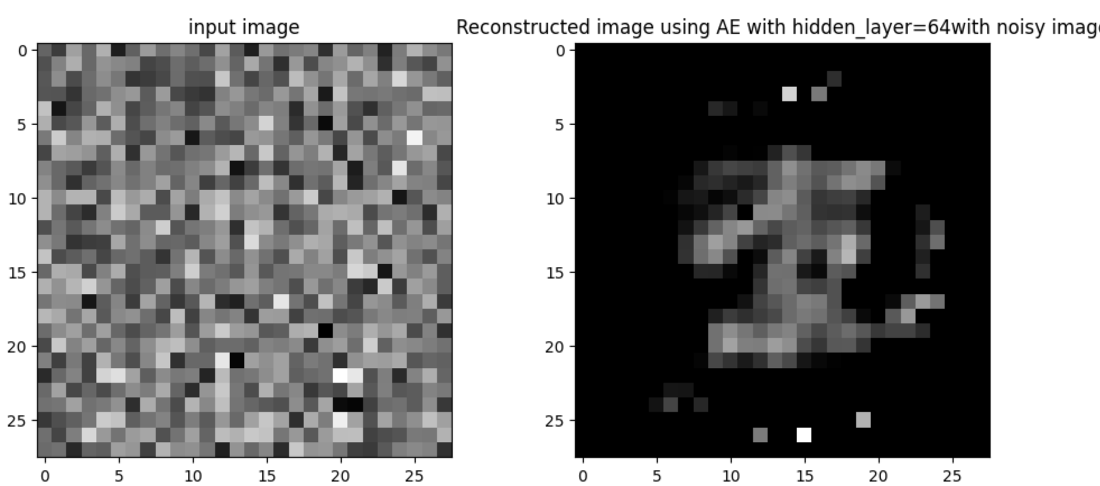

input image
Reconstructed image using AE with hidden_layer=64
input image
Reconstructed image using AE with hidden_layer=64

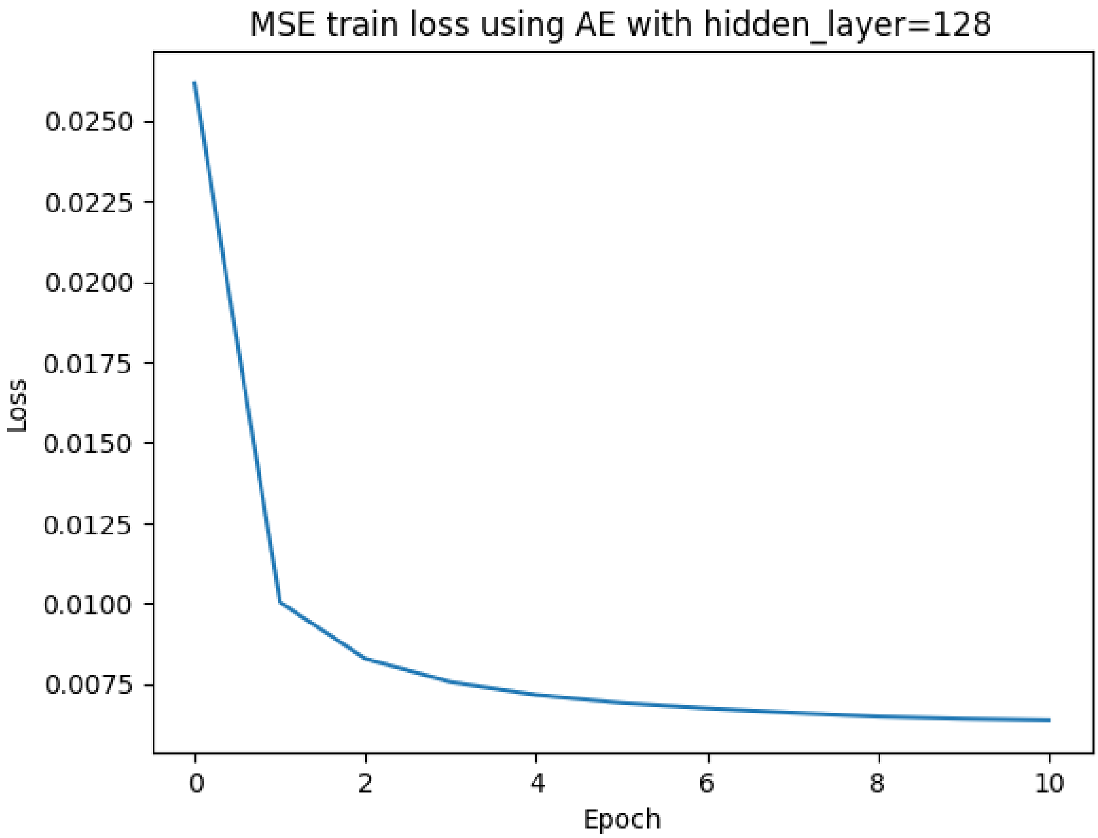

input image
Reconstructed image using AE with hidden_layer=64
0
5
10
15
20
25


input image
Reconstructed image using AE with hidden_layer=64
0
5
10
15
20
25

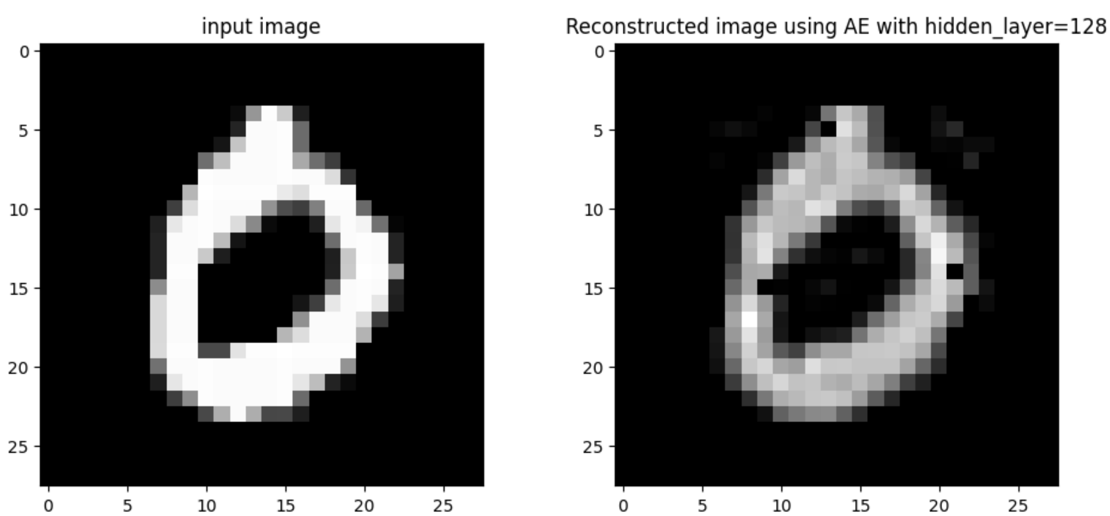

input image
Reconstructed image using AE with hidden_layer=64

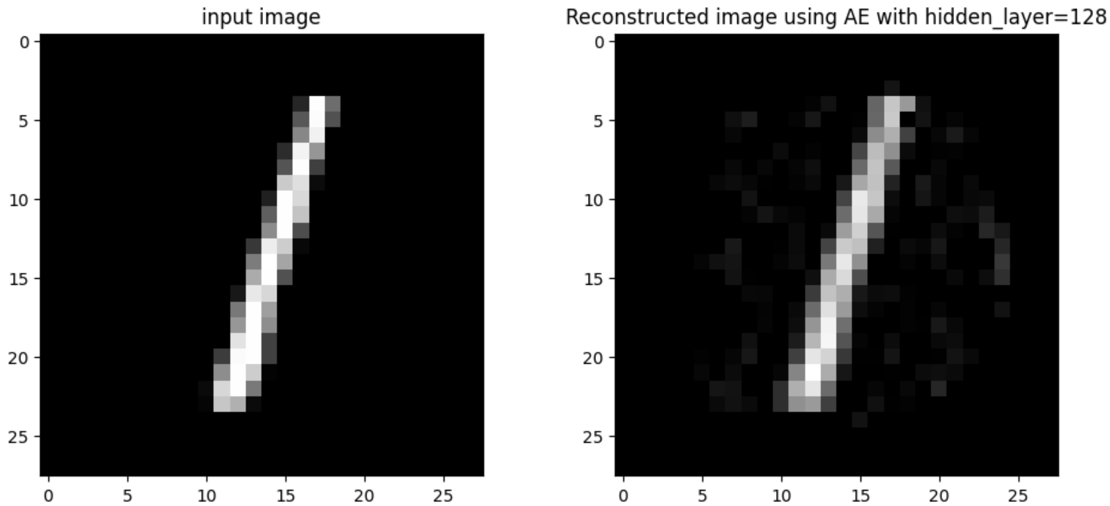

input image
Reconstructed image using AE with hidden_layer=64with non digit image

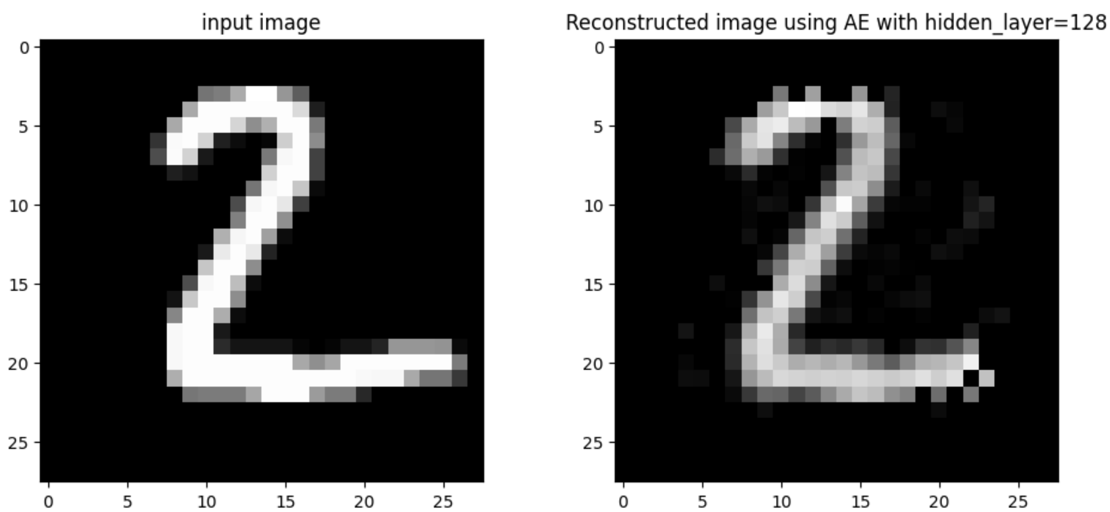

input image
Reconstructed image using AE with hidden_layer=64with noisy image

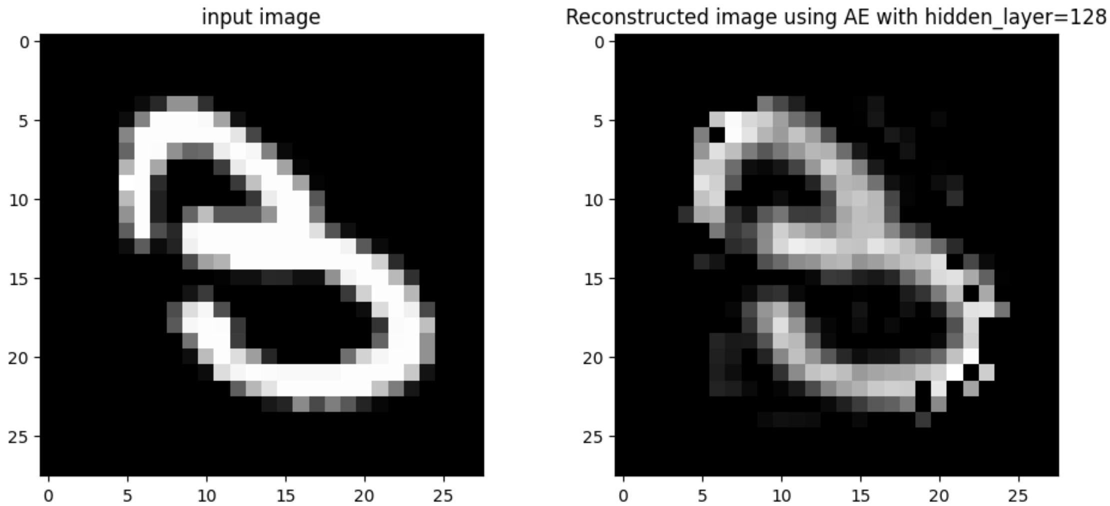

MSE train loss using AE with hidden_layer=128
0.0250
0.0225
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
Loss
0
2
4
6
8
10
Epoch

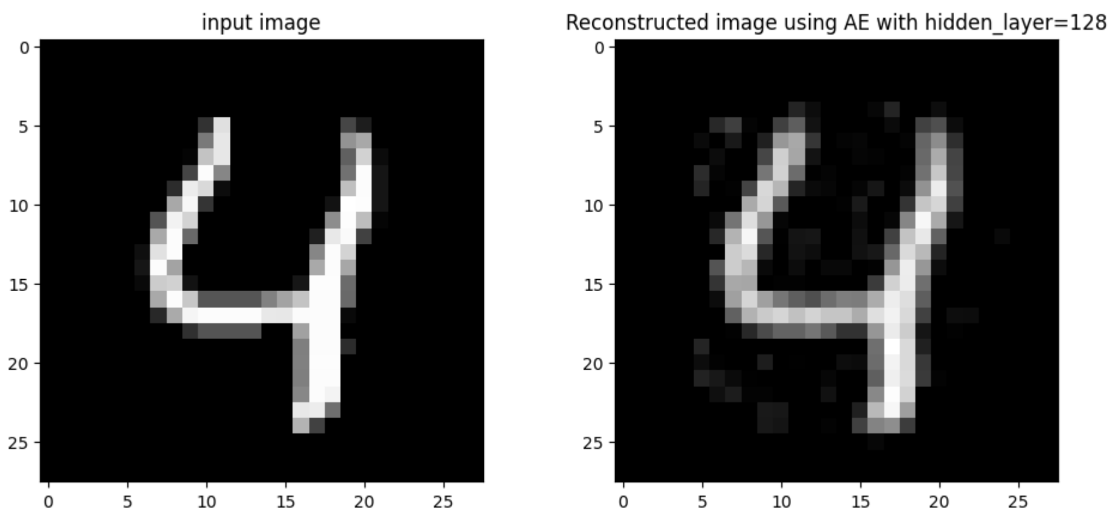

MSE test loss using AE with hidden_layer=128
loss
Epoch
0.011
0.010
0.009
0.008
0.007
0.006
0
2
4
6
8
10

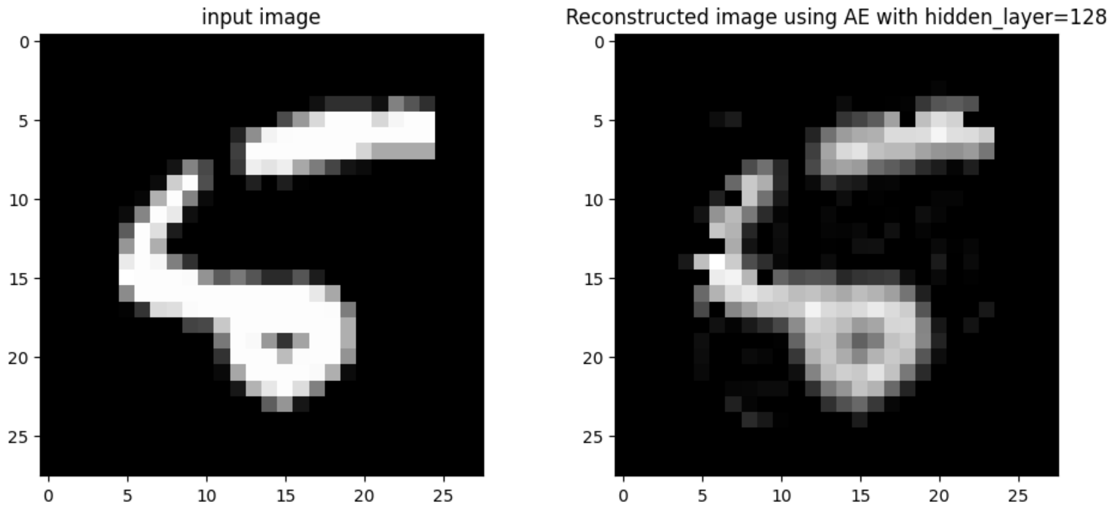

input image
Reconstructed image using AE with hidden_layer=128
0
5
10
15
20
25

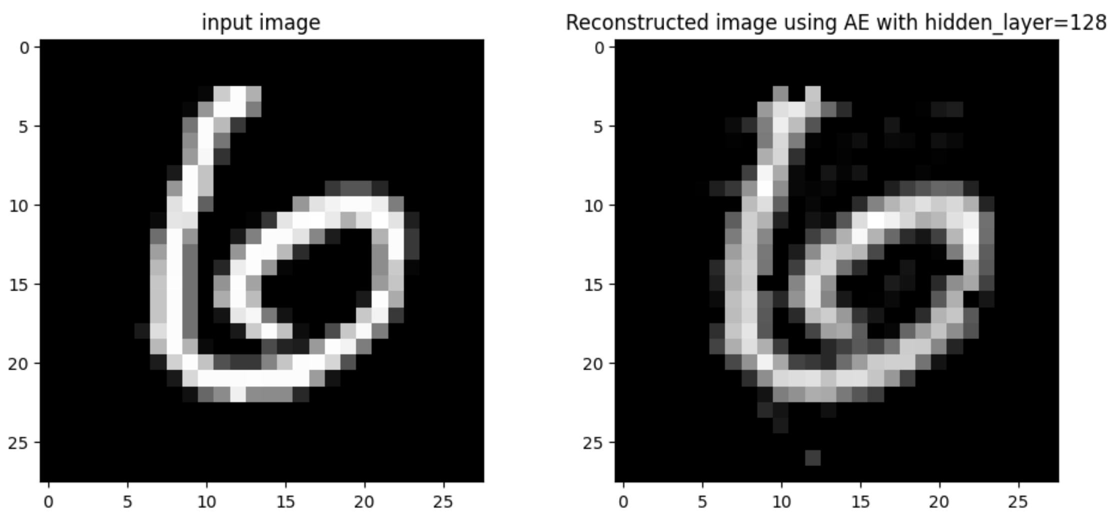

input image
Reconstructed image using AE with hidden_layer=128

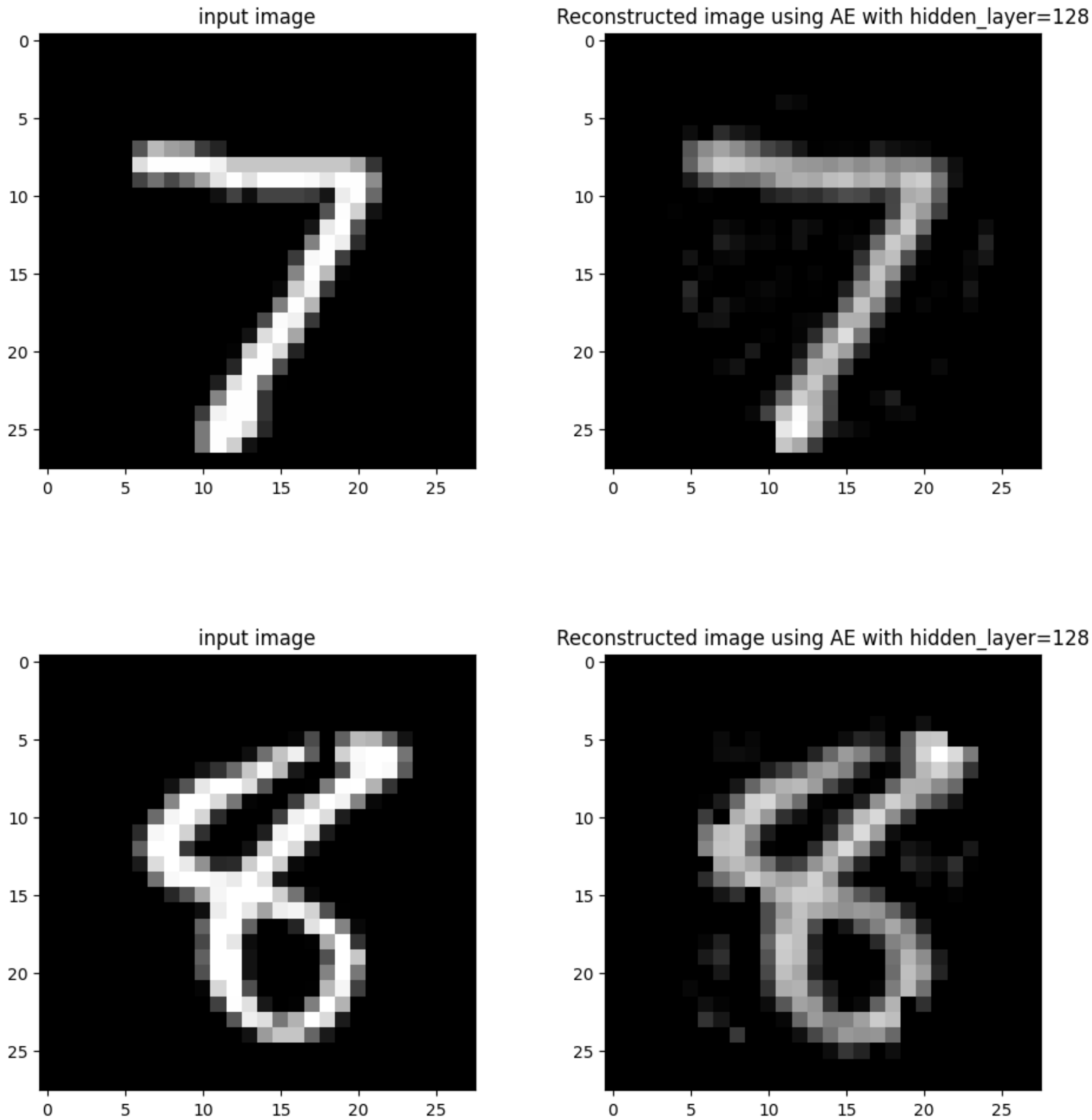

input image
Reconstructed image using AE with hidden_layer=128

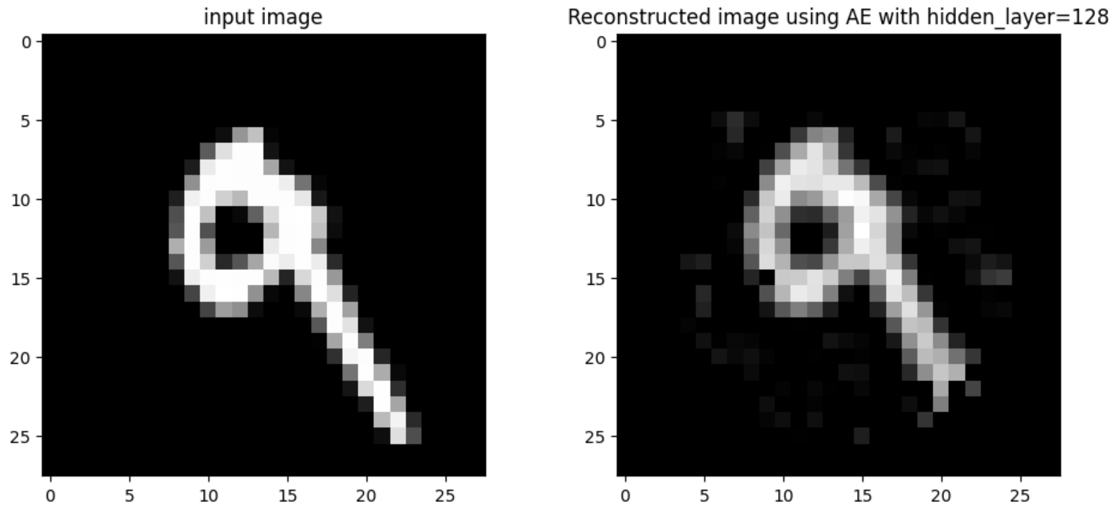

input image
Reconstructed image using AE with hidden_layer=128

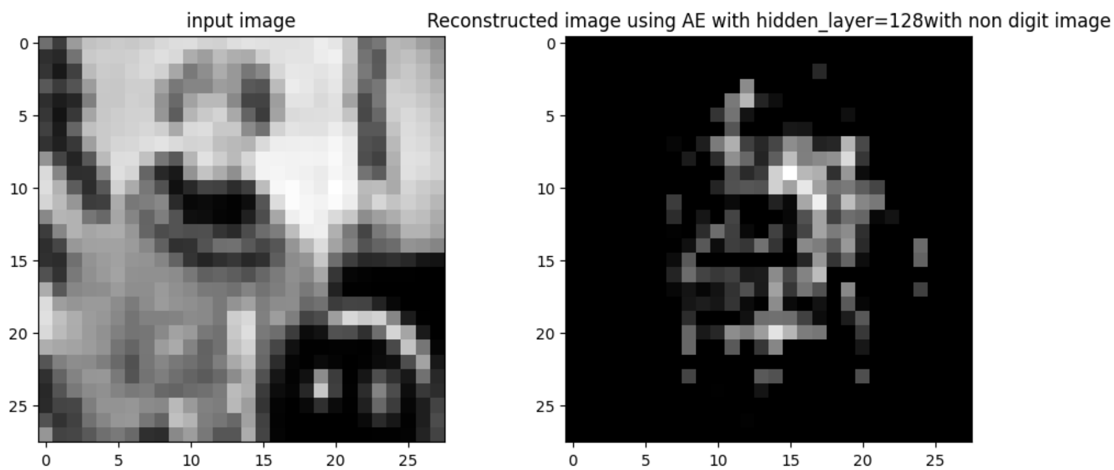

input image
Reconstructed image using AE with hidden_layer=128

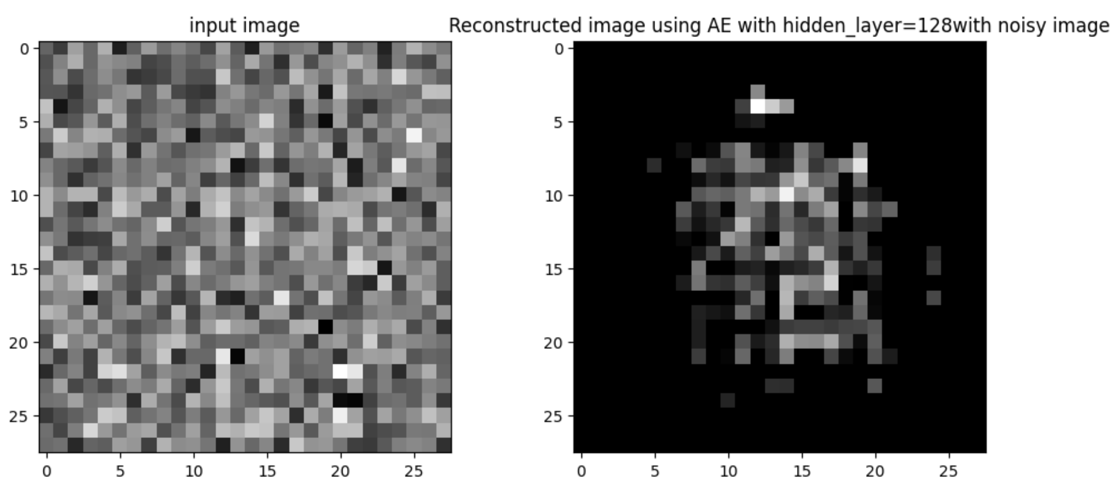

input image
Reconstructed image using AE with hidden_layer=128

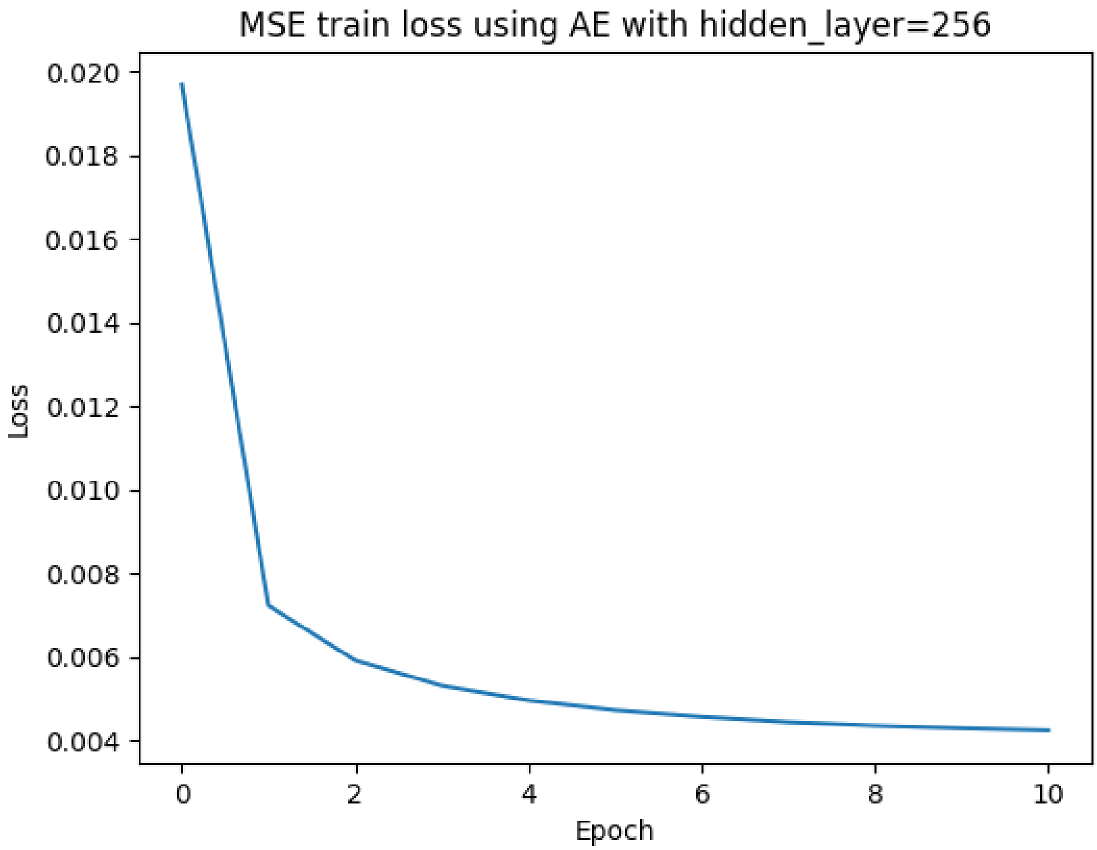

input image
Reconstructed image using AE with hidden_layer=128


input image
Reconstructed image using AE with hidden_layer=128
input image
Reconstructed image using AE with hidden_layer=128

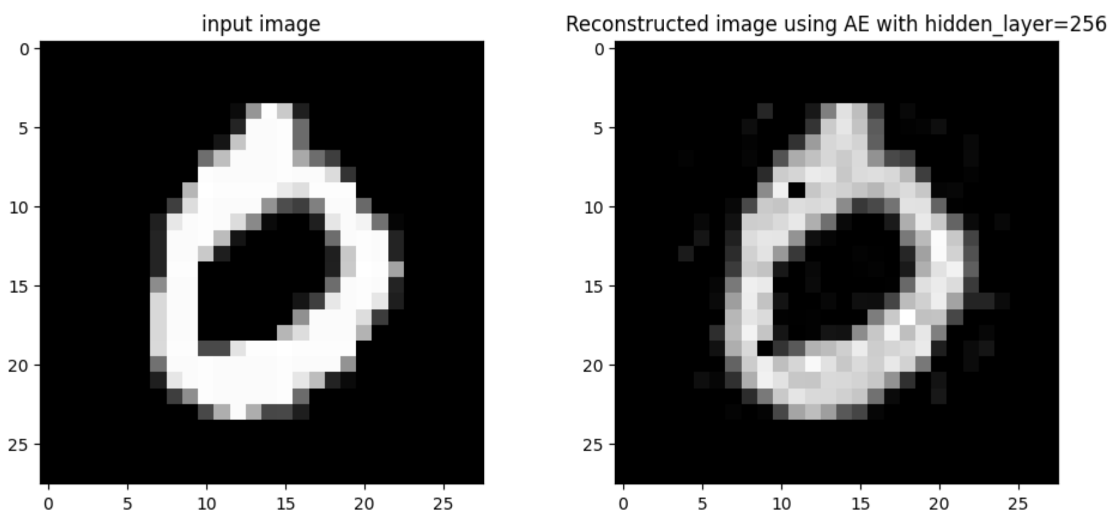

input image
Reconstructed image using AE with hidden_layer=128

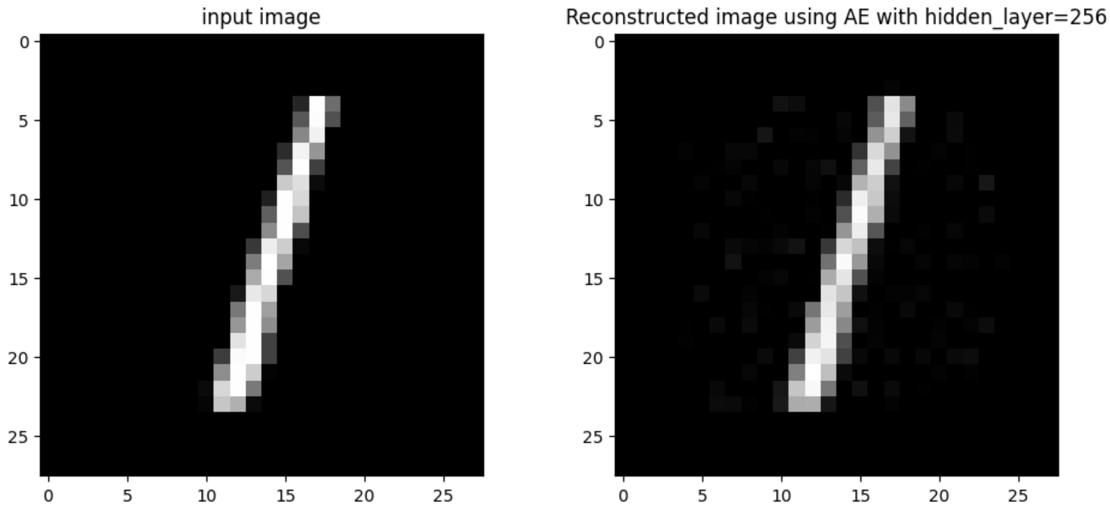

input image
Reconstructed image using AE with hidden_layer=128with non digit image

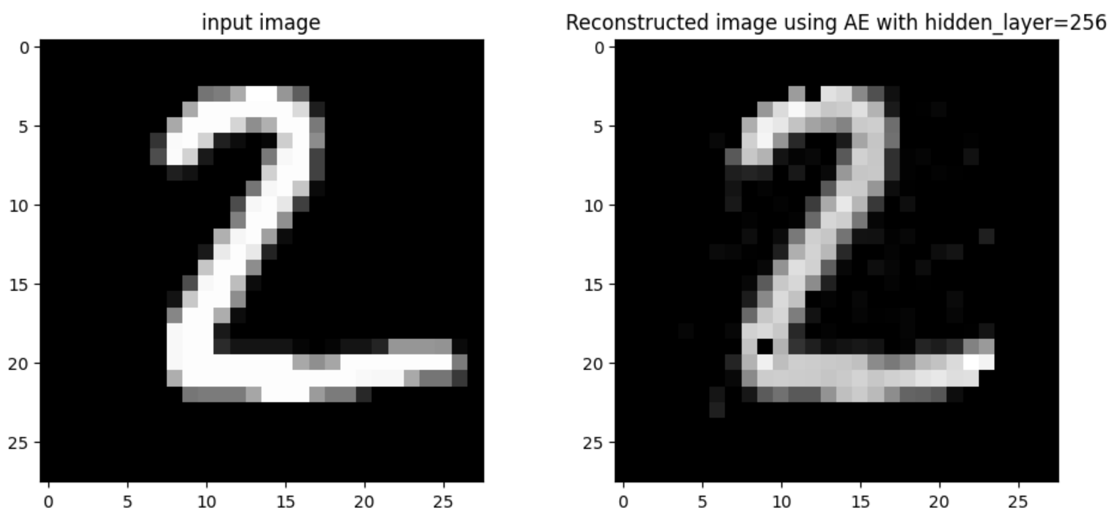

input image
Reconstructed image using AE with hidden_layer=128with noisy image

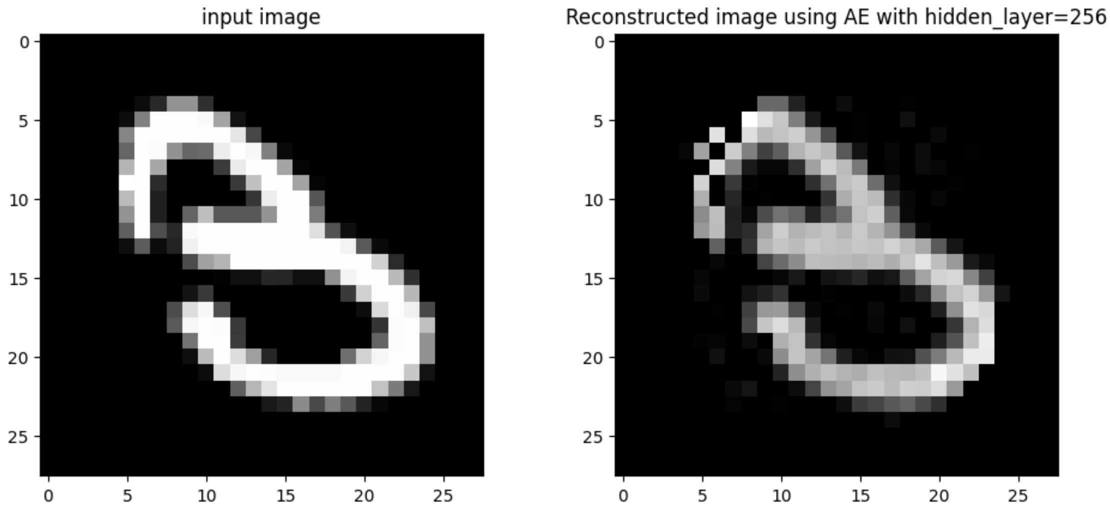

MSE train loss using AE with hidden_layer=256
Loss
Epoch
0.020
0.018
0.016
0.014
0.012
0.010
0.008
0.006
0.004
0
2
4
6
8
10

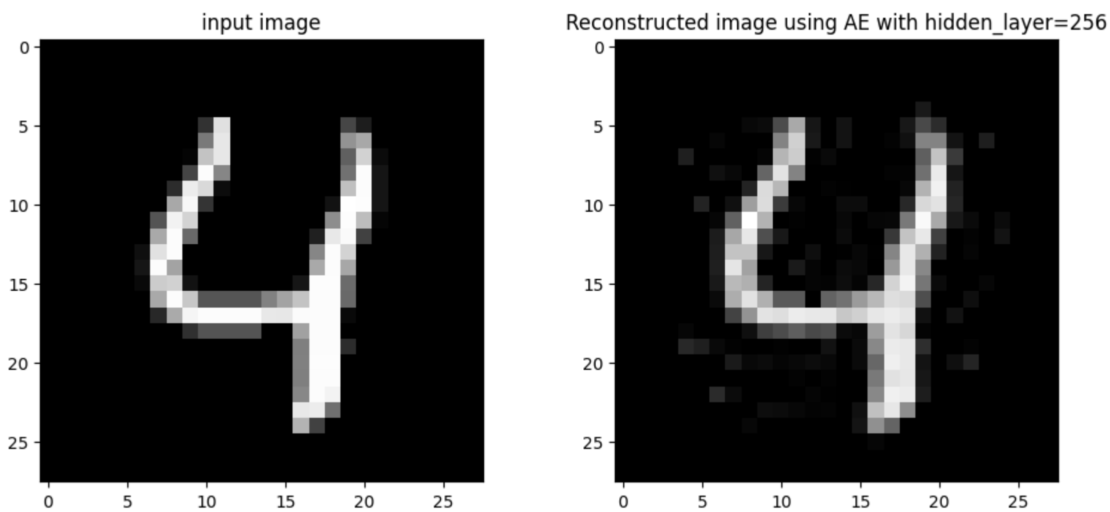

MSE test loss using AE with hidden_layer=256
0.008
0.007
0.006
0.005
0.004
loss
0
2
4
6
8
10
Epoch

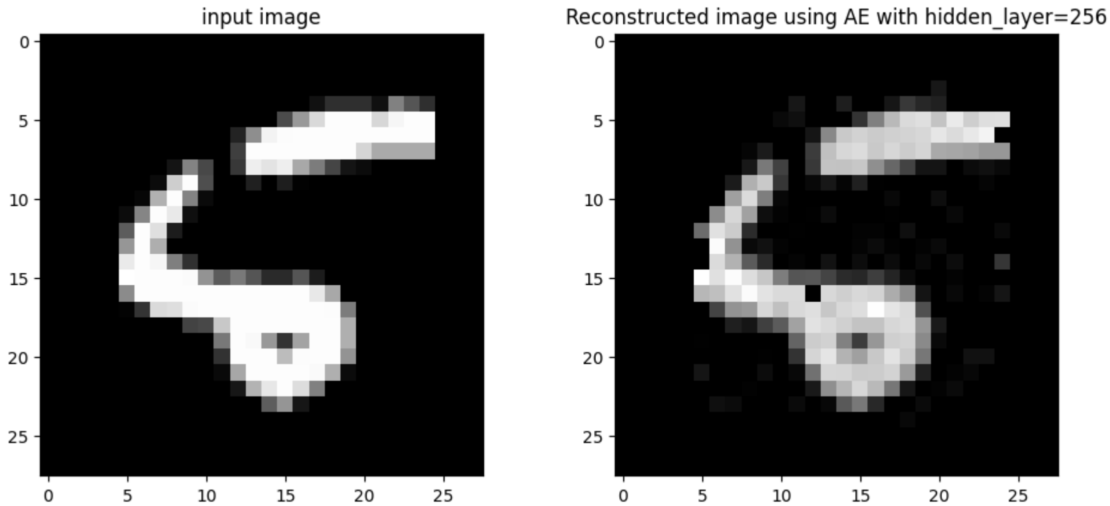

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

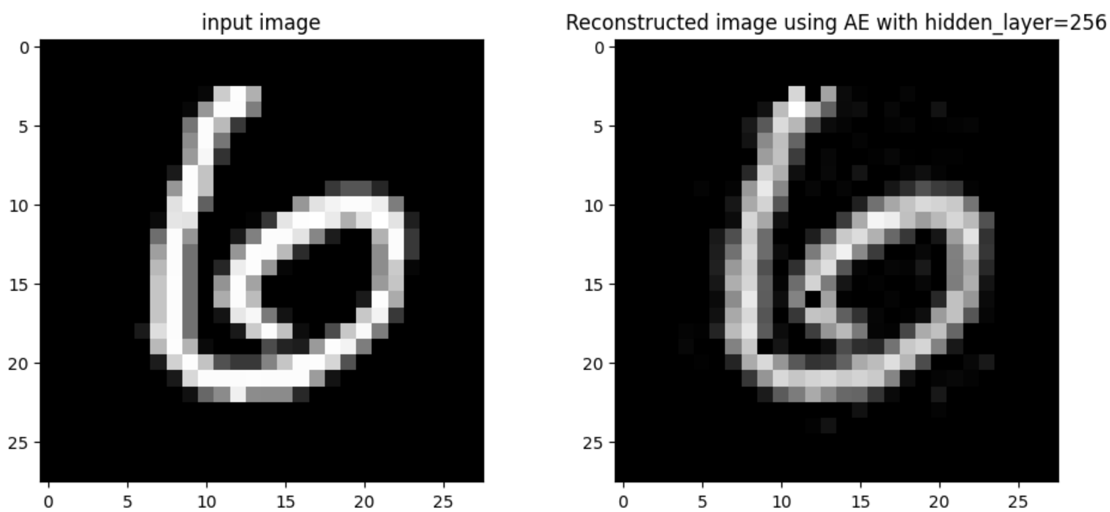

input image
Reconstructed image using AE with hidden_layer=256

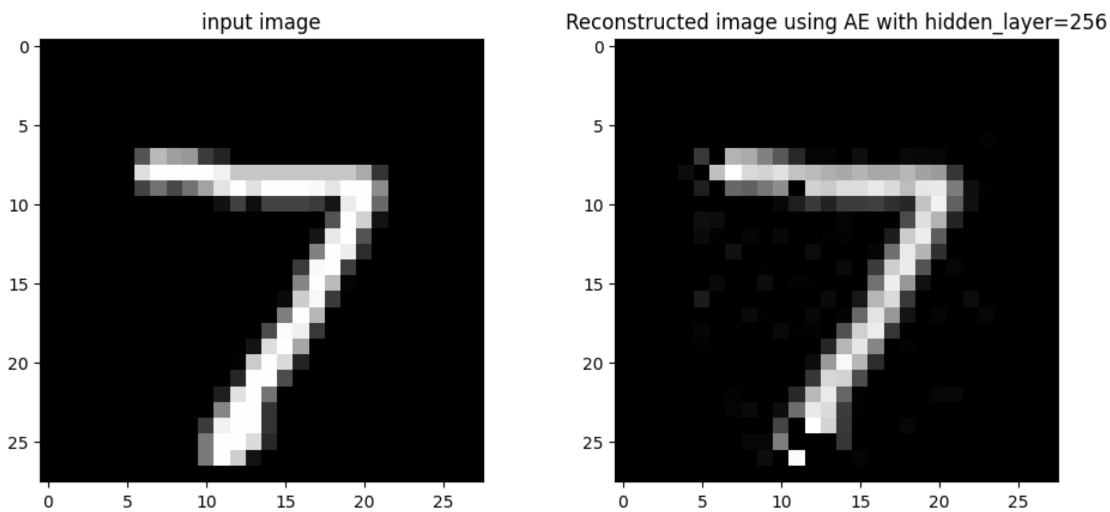

input image
Reconstructed image using AE with hidden_layer=256

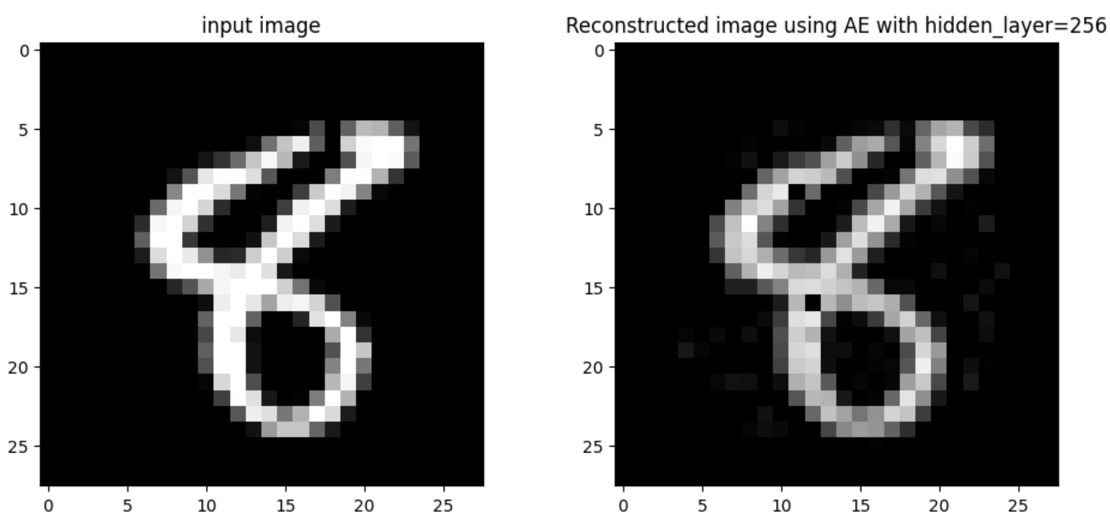

input image
Reconstructed image using AE with hidden_layer=256

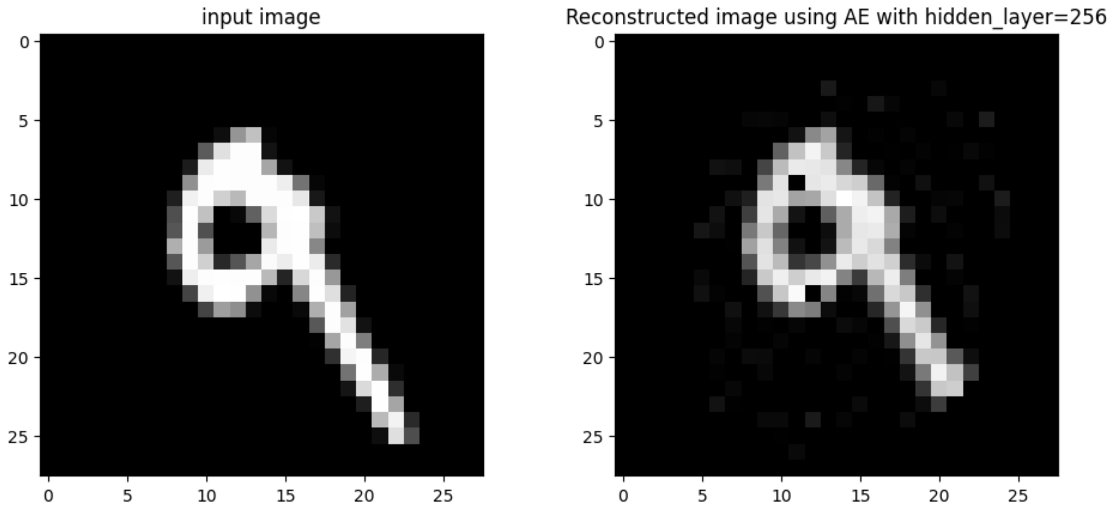

input image
Reconstructed image using AE with hidden_layer=256

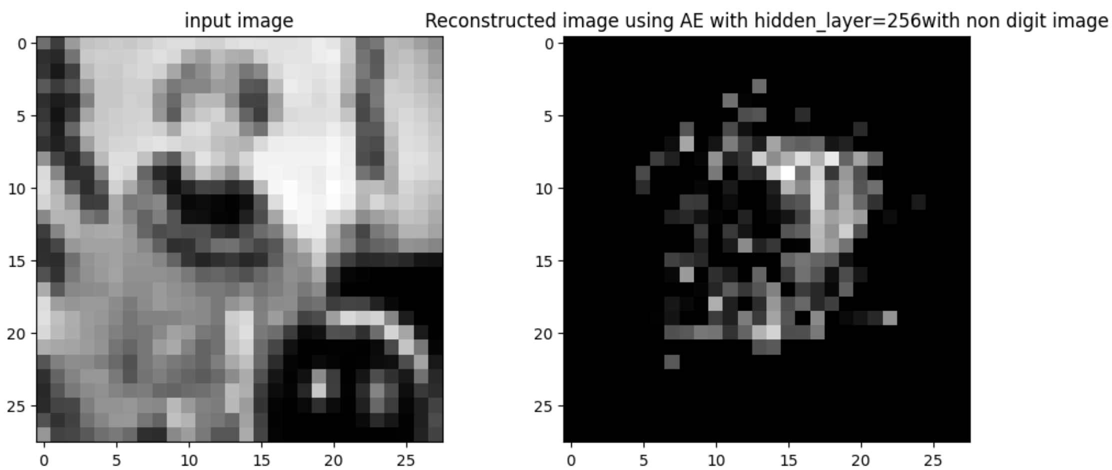

input image
Reconstructed image using AE with hidden_layer=256

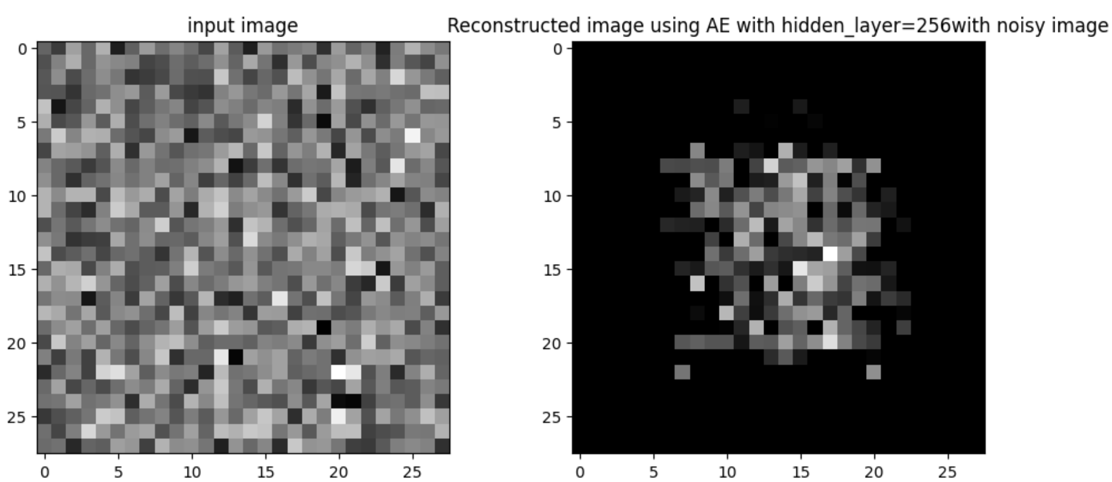

input image
Reconstructed image using AE with hidden_layer=256


input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25


input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25


input image
Reconstructed image using AE with hidden_layer=256


input image
Reconstructed image using AE with hidden_layer=256with non digit image


input image
Reconstructed image using AE with hidden_layer=256with noisy image

[ ]:

```

```

0.3 Q3

[32]:
```
class AE_Q3(nn.Module):
  def __init__(self):
    super(AE_Q3, self).__init__()
    self.encoder = nn.Sequential(
        nn.Linear(784,1225),
        nn.ReLU())
    self.decoder =nn.Sequential(
        nn.Linear(1225,784),
        nn.ReLU())

  def forward(self,x):
    x = flatten(x,1) #flatten the image to a 784x1 vector
    encoded_input = self.encoder(x.float())
    reconstructed_input = self.decoder(encoded_input)
    return reconstructed_input,encoded_input
```

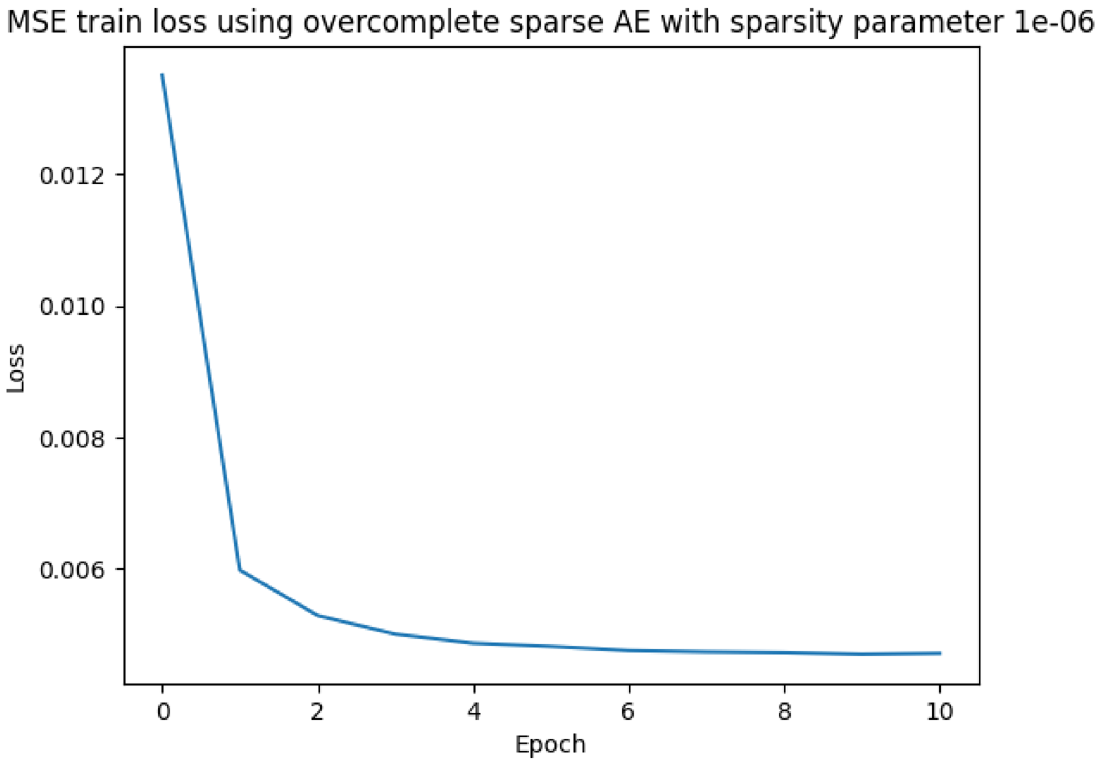
[33]:
```
def avg_hl_activations(model,test_dataloader,model_name):
    model.eval()
    avg_act_val = 0
    with torch.no_grad():
        for (data,label) in test_dataloader:
            (data,label) = (data.to(device),label.to(device))
            reconstruction,encoded = model(data) #our prediction
            avg_act_val += float(torch.mean(encoded))
    avg_act_val /= len(test_dataloader)
    print("The average activation of "+ str(model_name)+" is",avg_act_val)
```

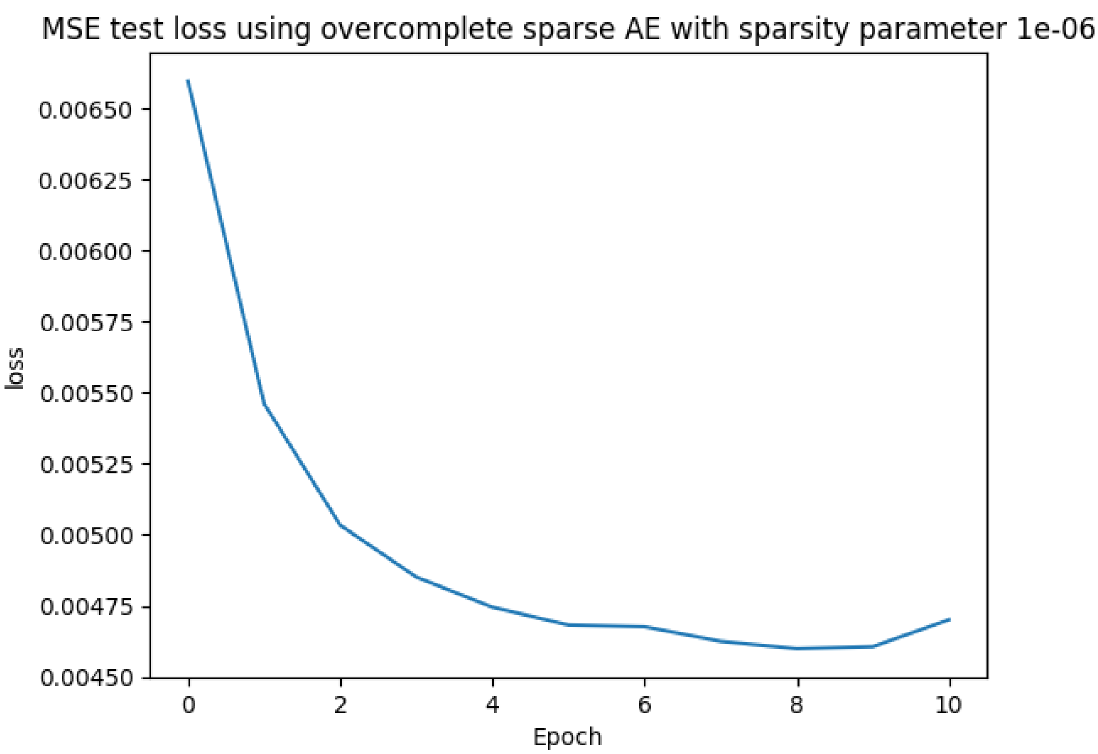
[34]:
```
def encoder_decoder_filters_plots(model,model_name,device):
    with torch.no_grad():
        encoder_filters = model.encoder[0].weight.detach().cpu().numpy()
        decoder_filters = model.decoder[0].weight.detach().cpu().numpy()
        #plot the encoder and decoder weights as an image for 0th neuron
        plt.imshow(encoder_filters[0].reshape(28,28), cmap='gray')
        plt.colorbar()
        plt.title('Encoder Filters for '+str(0)+'th neuron of '+␣
 ↪str(model_name))
        plt.show()

        plt.imshow(decoder_filters[:,0].reshape(28,28), cmap='gray')
        plt.colorbar()
```

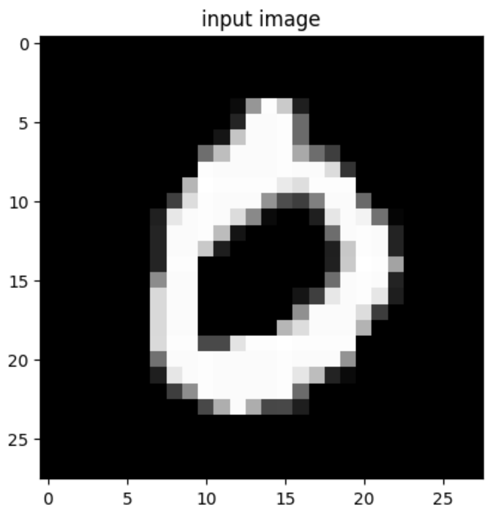
```
        plt.title('Decoder Filters for '+str(0)+'th neuron of '+␣
↪str(model_name))
        plt.show()
```

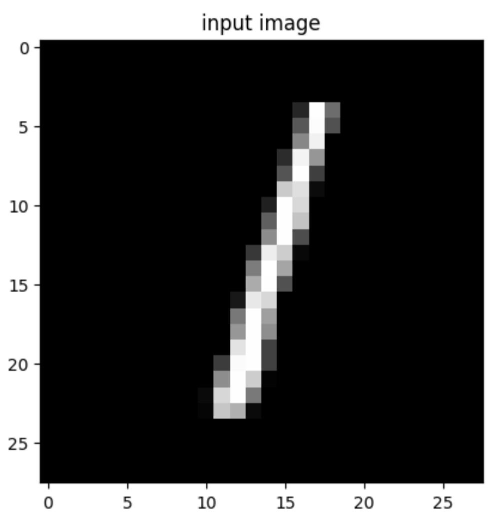
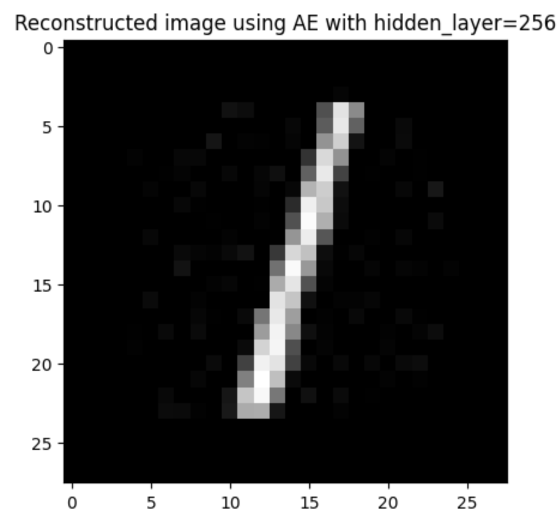
[35]:
```
def visualize_activations(model,test_dataloader,model_name,device,hidden_layer):
↪ #visualize the activations
    # data_ind  = np.random.randint(low=0, high=9999, size=5)
    for i,ind in enumerate(keys_test):
        test_image = test_dataloader.dataset.data[ind].clone()
        test_label = test_dataloader.dataset.targets[ind].clone()
        with torch.no_grad():
            if(device == torch.device("cuda")):
                test_image = test_image.reshape(1,1,28,28).cuda().float()
            else:
                test_image = test_image.reshape(1,1,28,28).float()
            reconstructed_image,encoded = model.forward(test_image)
            encoded = encoded.detach().cpu().numpy()
            plt.imshow(encoded.reshape(int(np.sqrt(hidden_layer)),int(np.
↪sqrt(hidden_layer))), cmap='gray')
            str_title = "Activation for digit "+str(test_label.item())
            plt.title(str_title)
            plt.show()
```

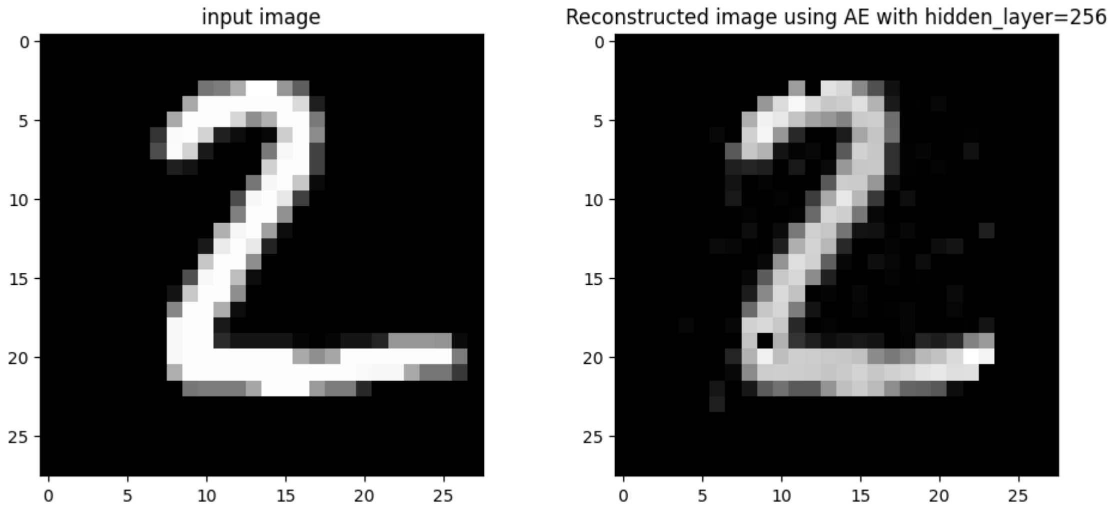
[36]:
```
lambda_reg_vals=[0.000001, 0.001, 0.1]
for lambda_reg in lambda_reg_vals:
    model_Q3 = AE_Q3().to(device)
    optimizer = torch.optim.Adam(model_Q3.parameters(), lr=learning_rate)
    train_losses_AE_Q3 , test_losses_AE_Q3 =␣
↪train_test(model_Q3,device,train_loader,test_loader,optimizer,lossfn,lambda_reg,sparse=True
    plot_losses(train_losses_AE_Q3, test_losses_AE_Q3, model_name =␣
↪"overcomplete sparse AE with sparsity parameter "+str(lambda_reg))
    for i in keys_test:
        test_image = test_loader.dataset.data[i, :, :].clone()
        plot_reconstructed_image(model_Q2,device,test_image, model_name="AE␣
↪with hidden_layer="+str(hidden_layer))
    plot_reconstructed_image(model_Q3,device,test_image,␣
↪model_name="overcomplete sparse AE with sparsity parameter "+str(lambda_reg))
    avg_hl_activations(model_Q3,test_loader,"overcomplete sparse AE with␣
↪sparsity parameter "+str(lambda_reg))
    encoder_decoder_filters_plots(model_Q3,"sparse AE with sparsity␣
↪parameter"+str(lambda_reg),device)
    visualize_activations(model_Q3,test_loader,"sparse AE with sparsity␣
↪parameter"+str(lambda_reg),device,1225)
```

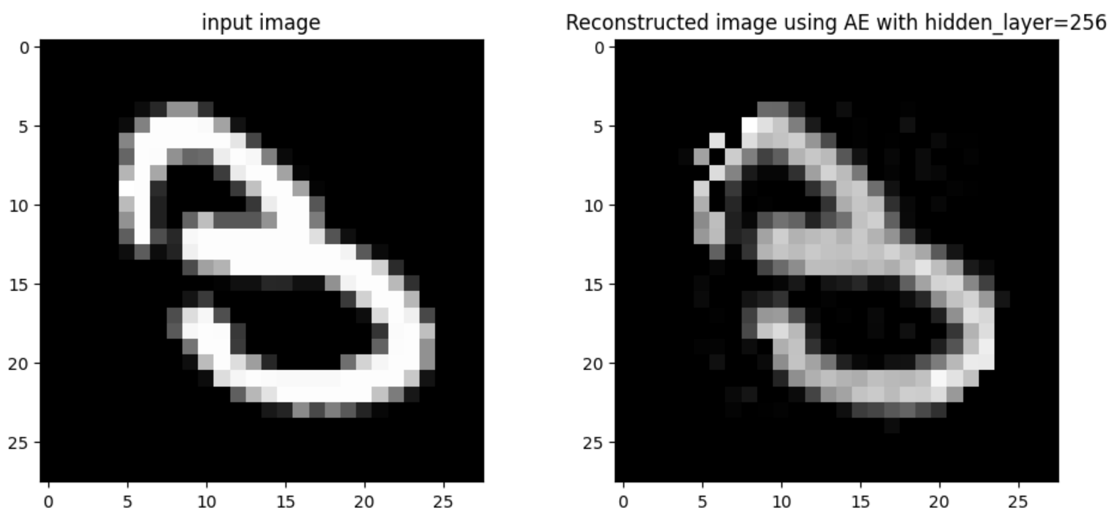

MSE train loss using overcomplete sparse AE with sparsity parameter 1e-06
0.012
0.010
0.008
0.006
Loss
0
2
4
6
8
10
Epoch

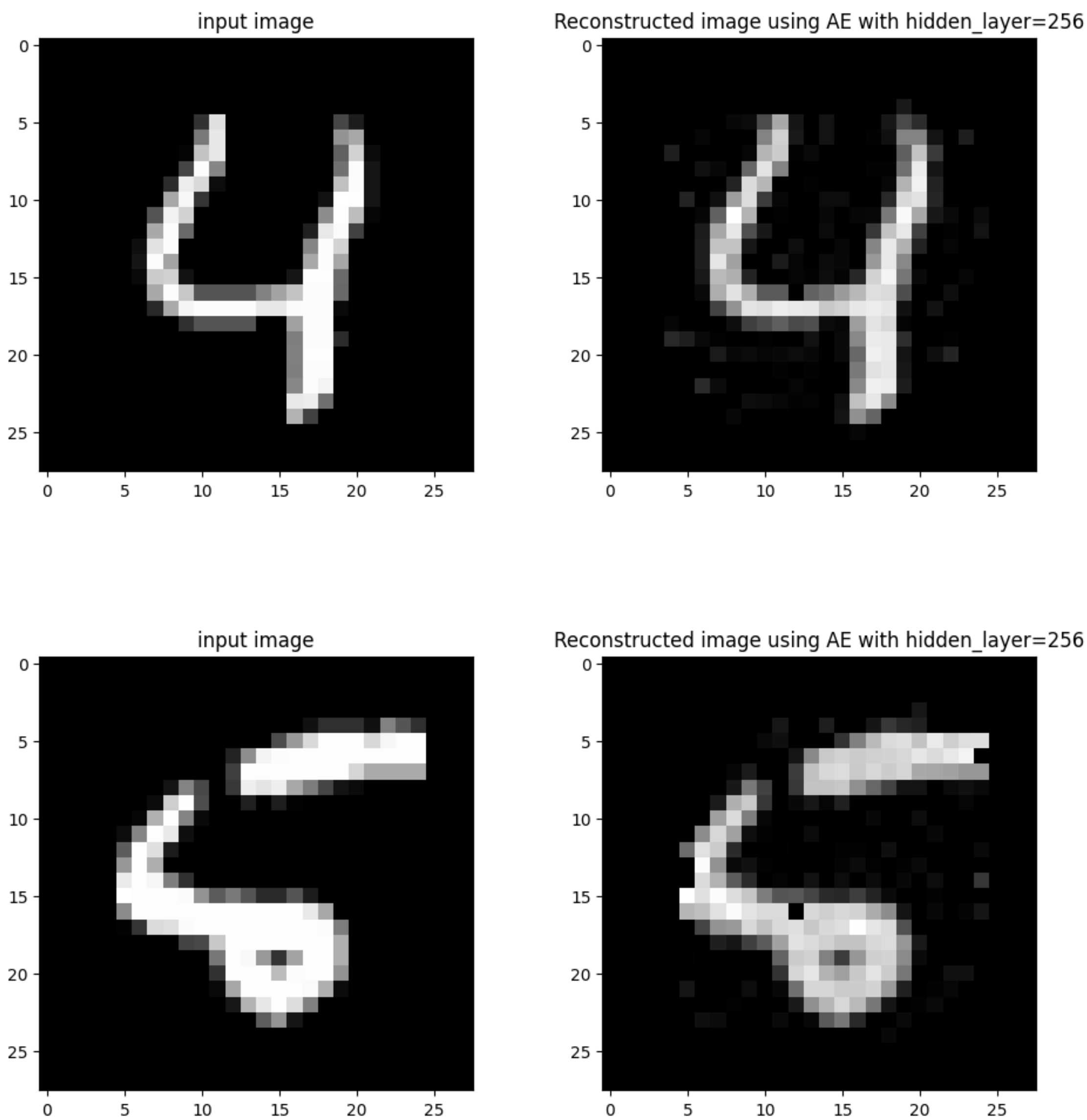

MSE test loss using overcomplete sparse AE with sparsity parameter 1e-06
0.00650
0.00625
0.00600
0.00575
0.00550
0.00525
0.00500
0.00475
0.00450
loss
0
2
4
6
8
10
Epoch

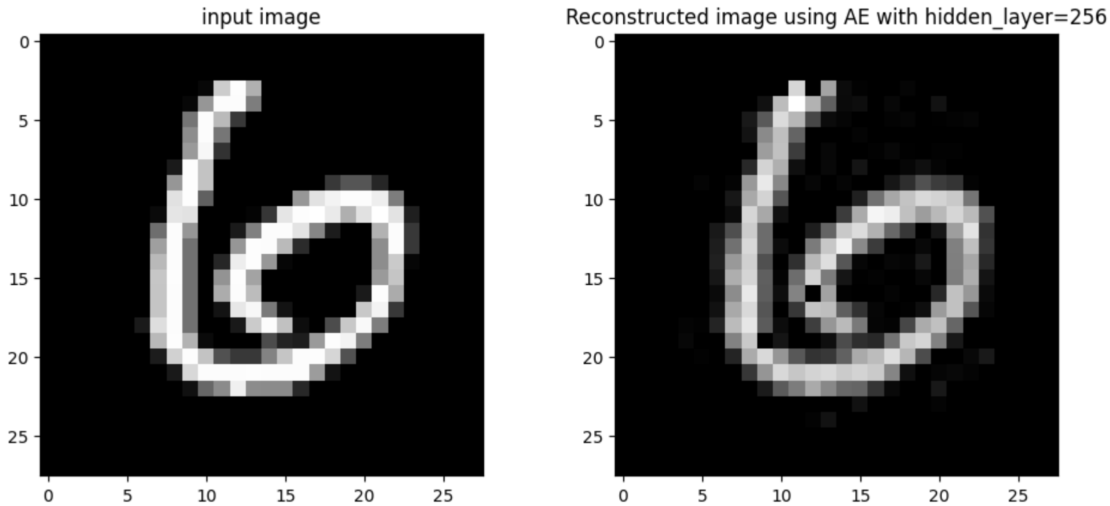

input image

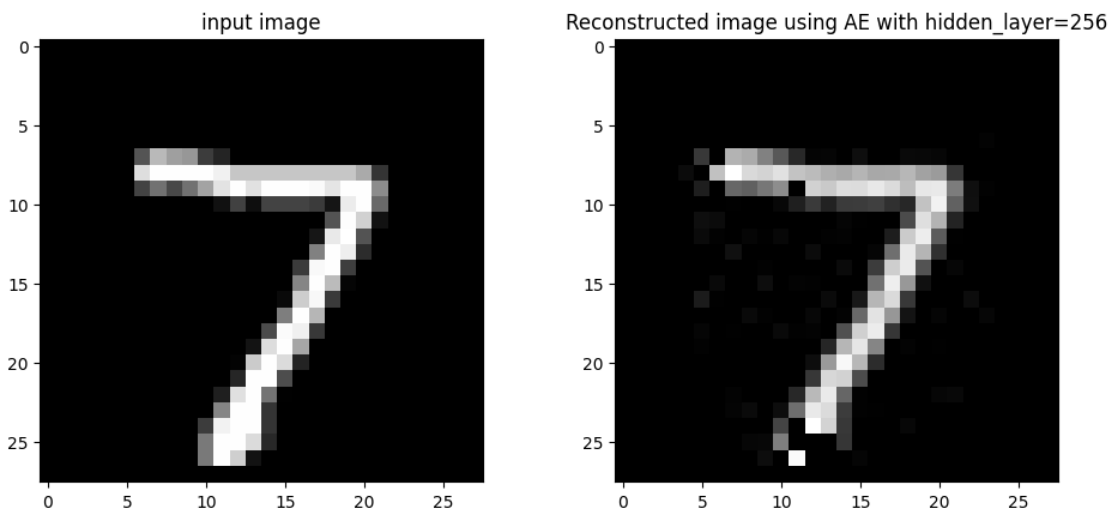
Reconstructed image using AE with hidden_layer=256

input image

Reconstructed image using AE with hidden_layer=256

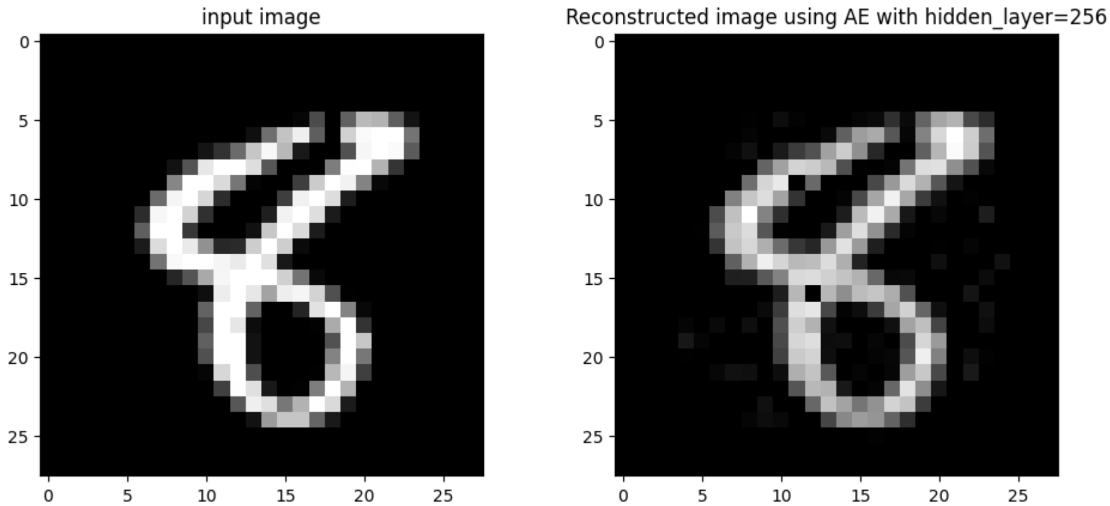

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

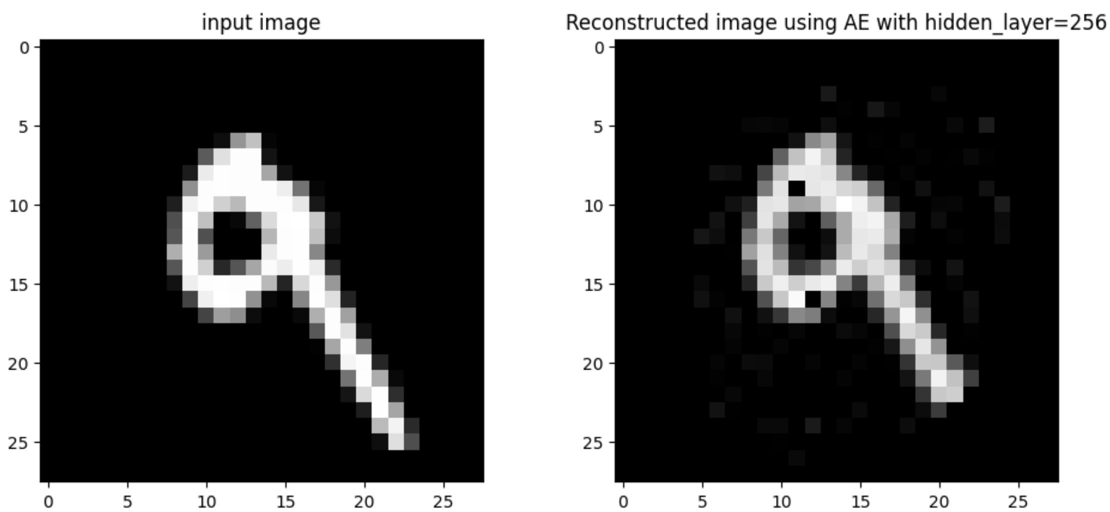

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

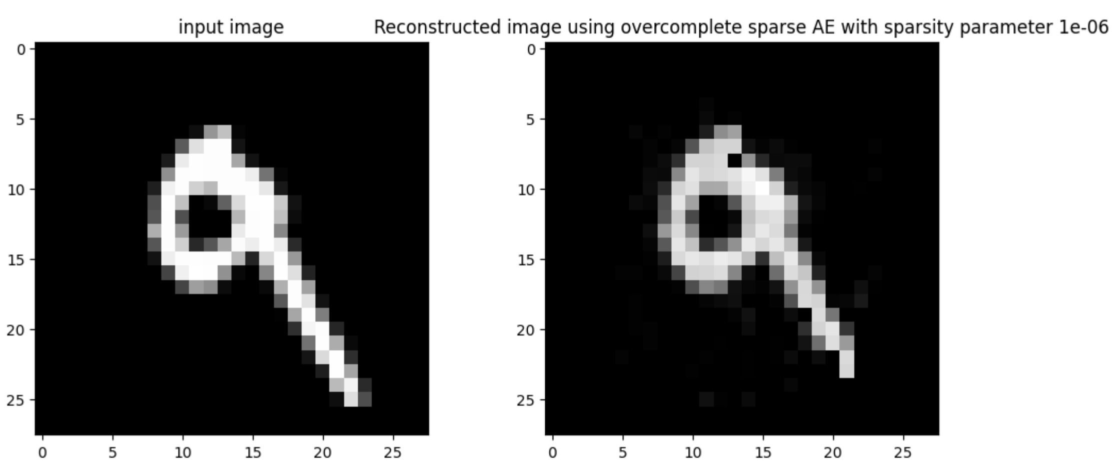

input image
Reconstructed image using AE with hidden_layer=256
input image
Reconstructed image using AE with hidden_layer=256


input image
Reconstructed image using AE with hidden_layer=256

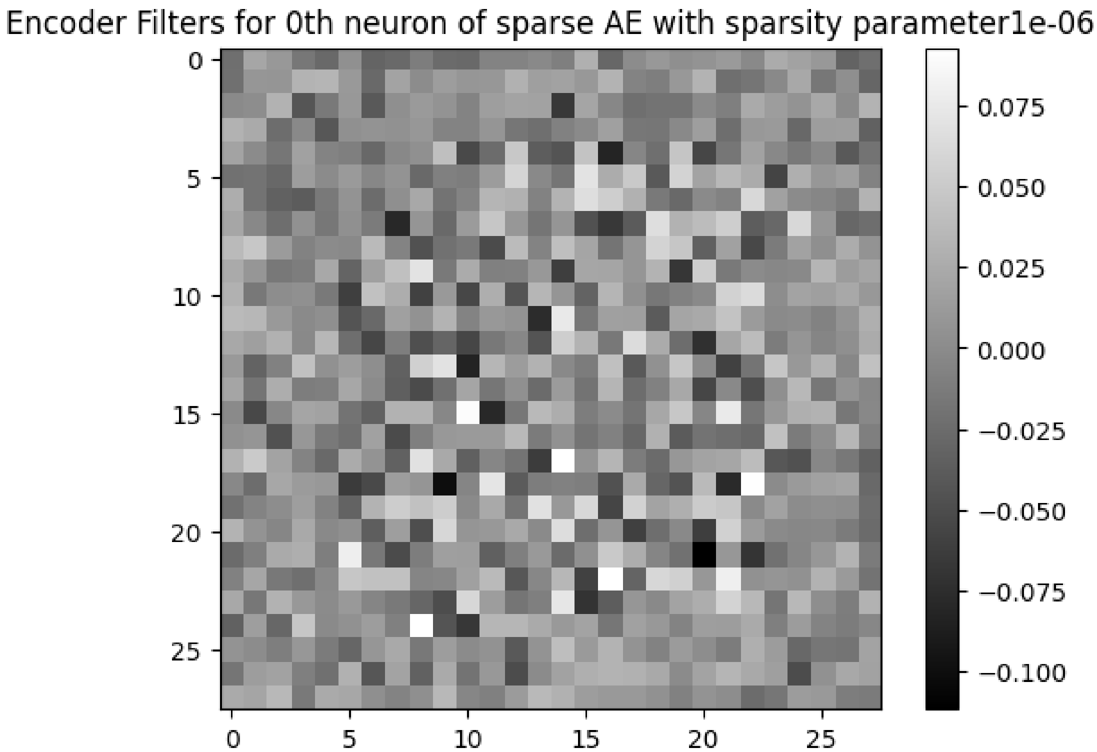

input image
Reconstructed image using AE with hidden_layer=256

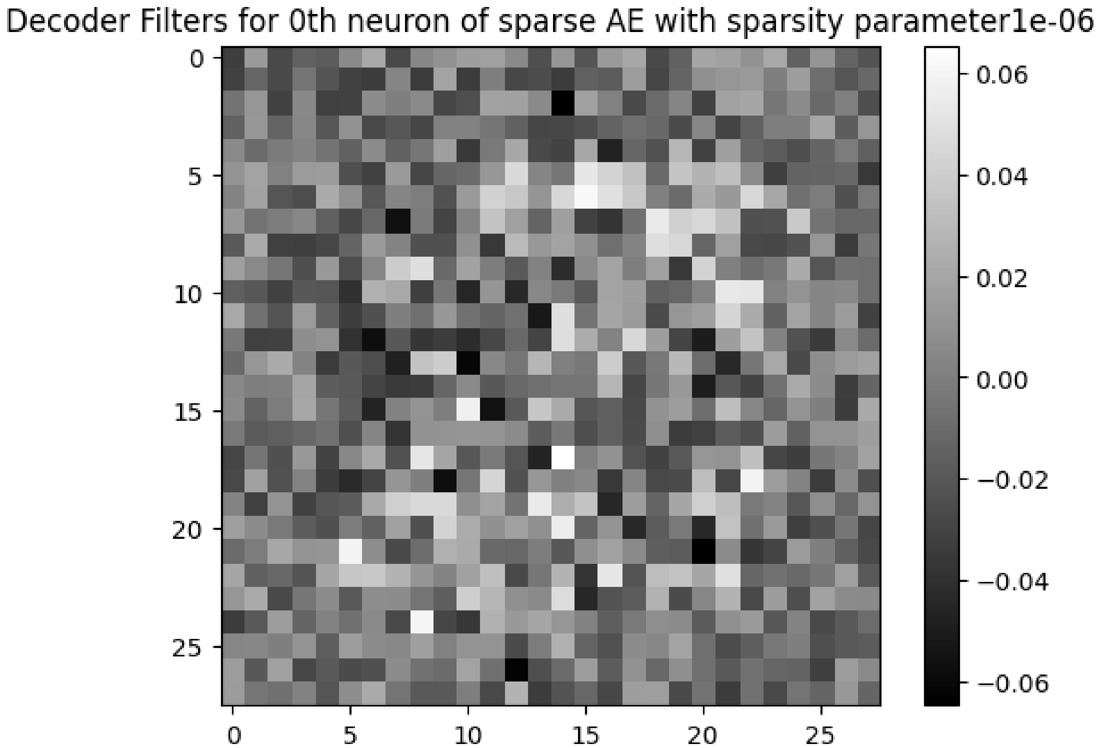

input image
Reconstructed image using AE with hidden_layer=256


input image
Reconstructed image using AE with hidden_layer=256

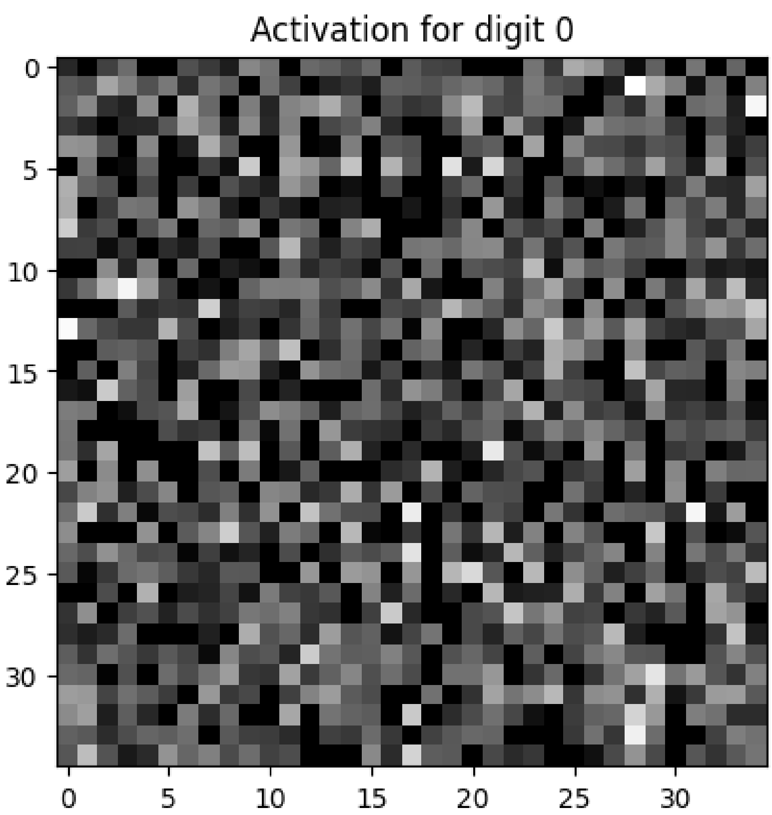

input image
Reconstructed image using overcomplete sparse AE with sparsity parameter 1e-06

```
The average activation of overcomplete sparse AE with sparsity parameter 1e-06
is 0.27716308906674386
```

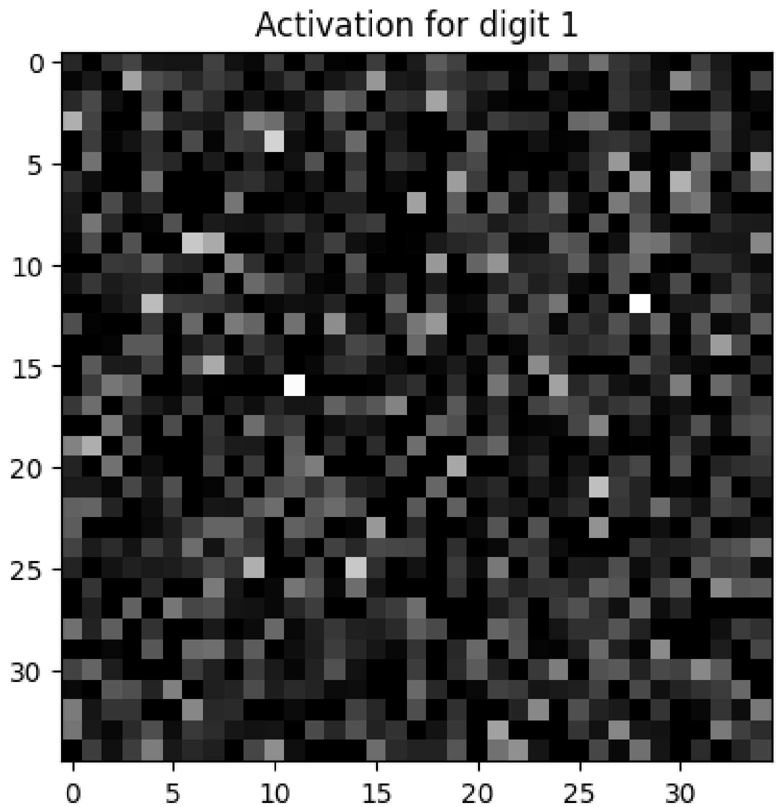




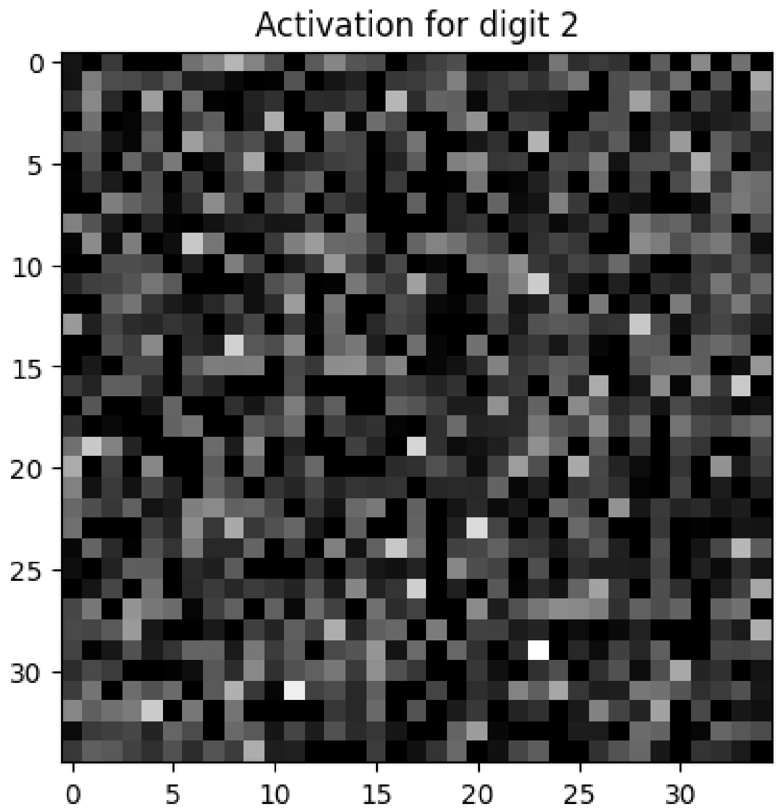
Decoder Filters for 0th neuron of sparse AE with sparsity parameter1e-06
0
5
10
15
20
25
0.06
0.04
0.02
0.00
−0.02
−0.04
−0.06

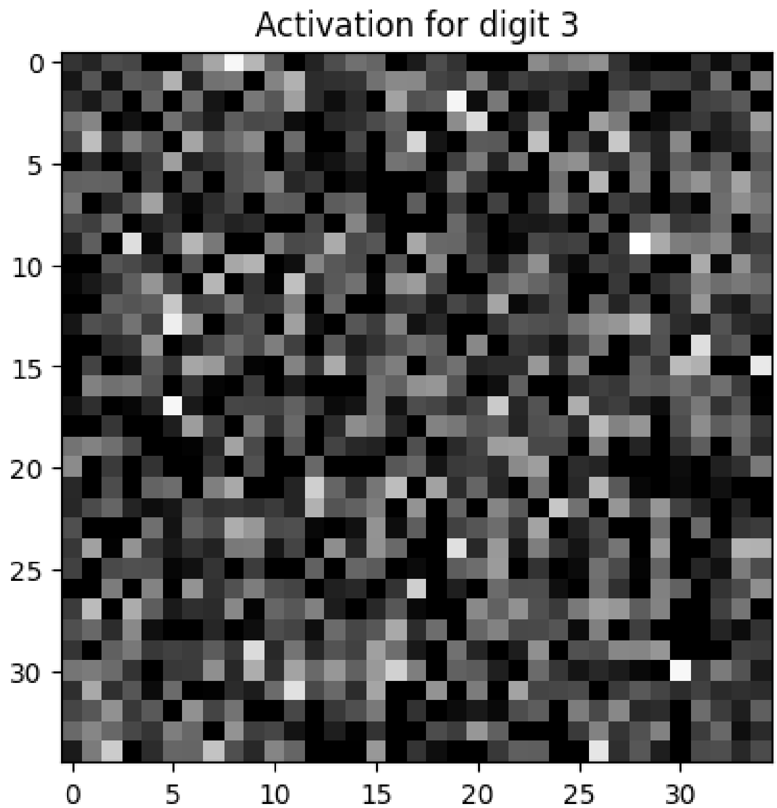


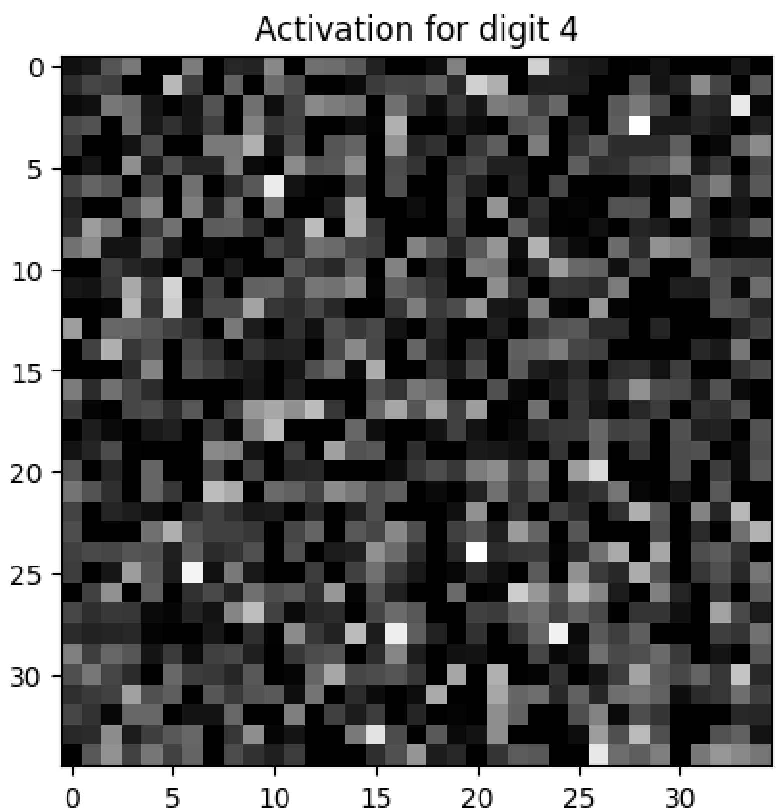
Activation for digit 0
0
5
10
15
20
25
30

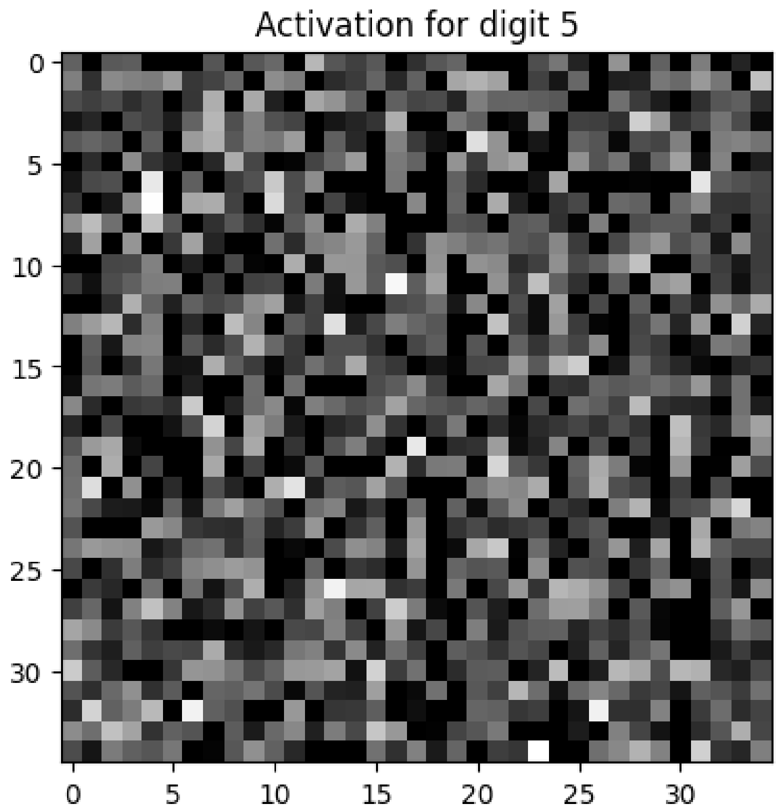


Activation for digit 1
0
5
10
15
20
25
30

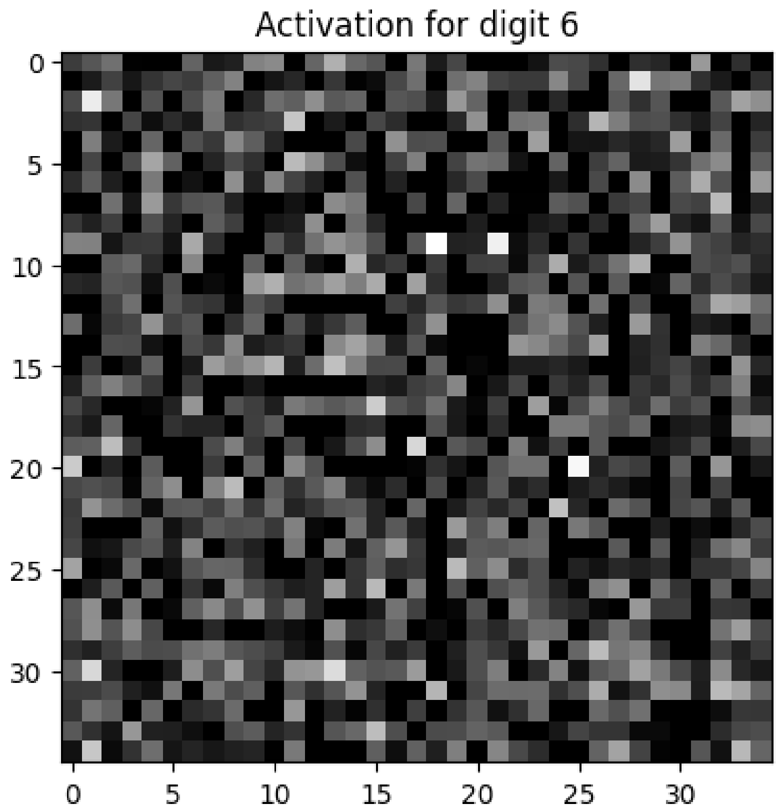

Activation for digit 2
0
5
10
15
20
25
30
0
5
10
15
20
25
30

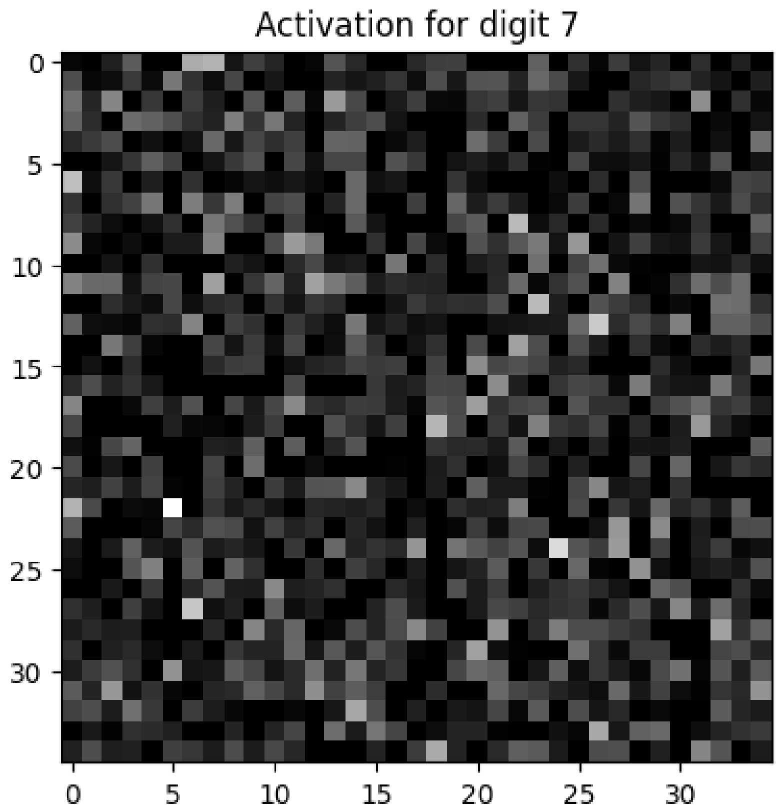

Activation for digit 3
0
5
10
15
20
25
30
0
5
10
15
20
25
30

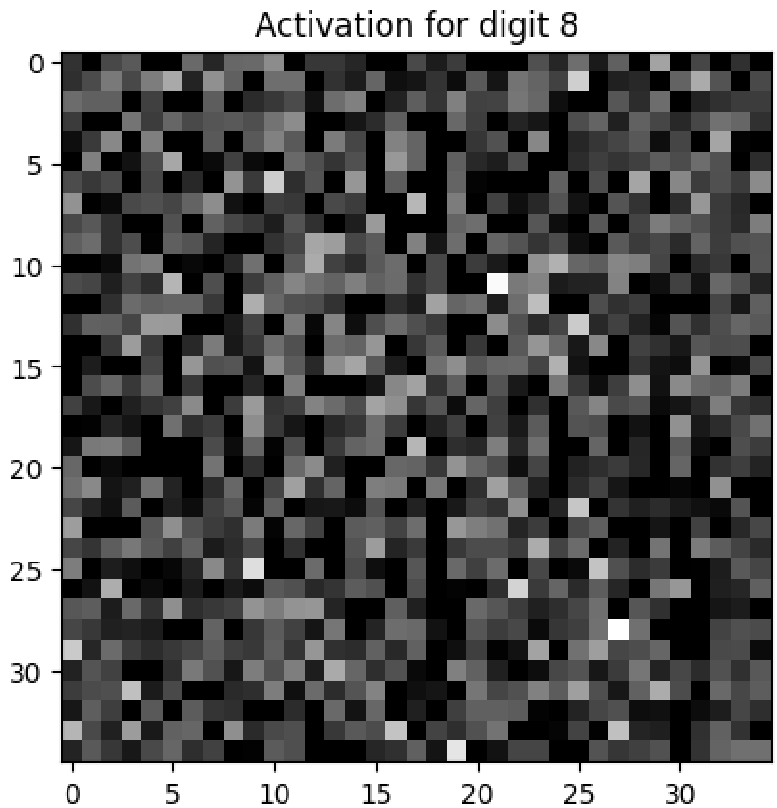

Activation for digit 4
0
5
10
15
20
25
30
0
5
10
15
20
25
30



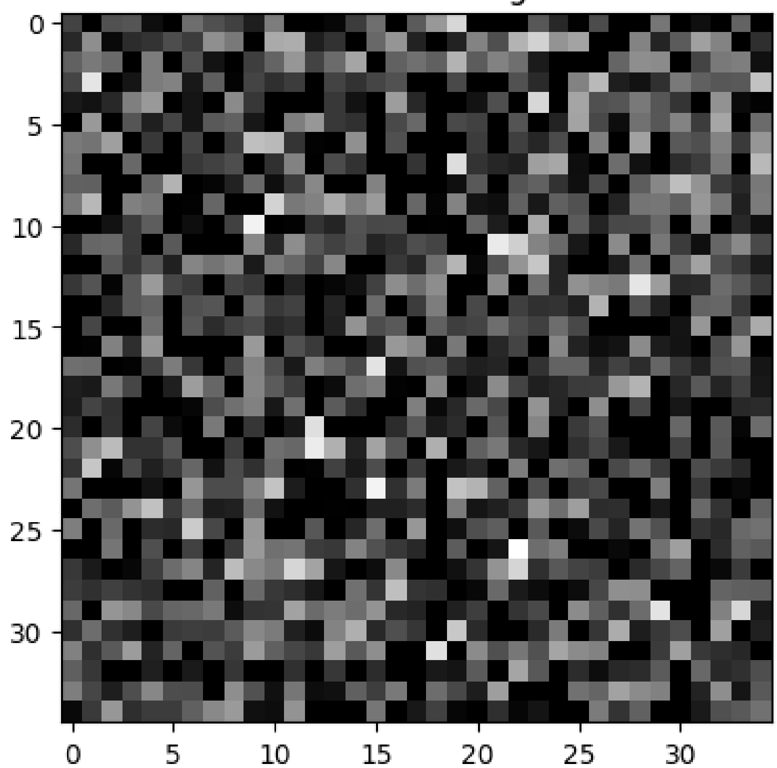
Activation for digit 5
0
5
10
15
20
25
30
0
5
10
15
20
25
30

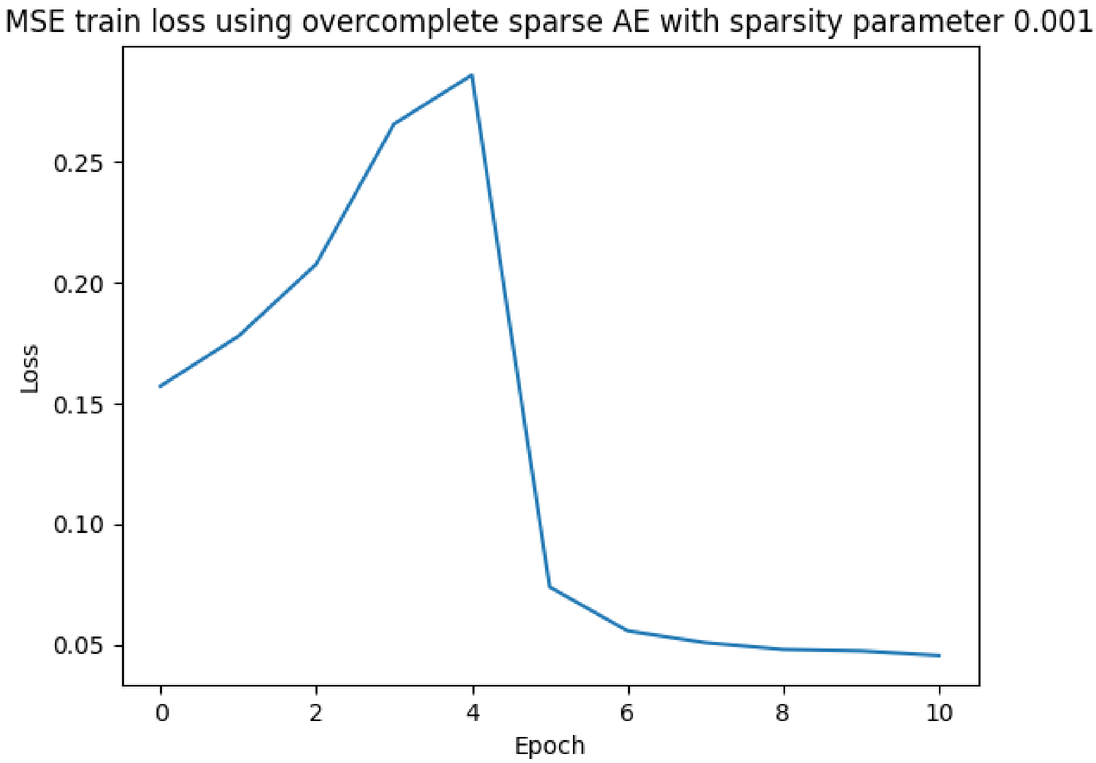

Activation for digit 6
0
5
10
15
20
25
30
0
5
10
15
20
25
30

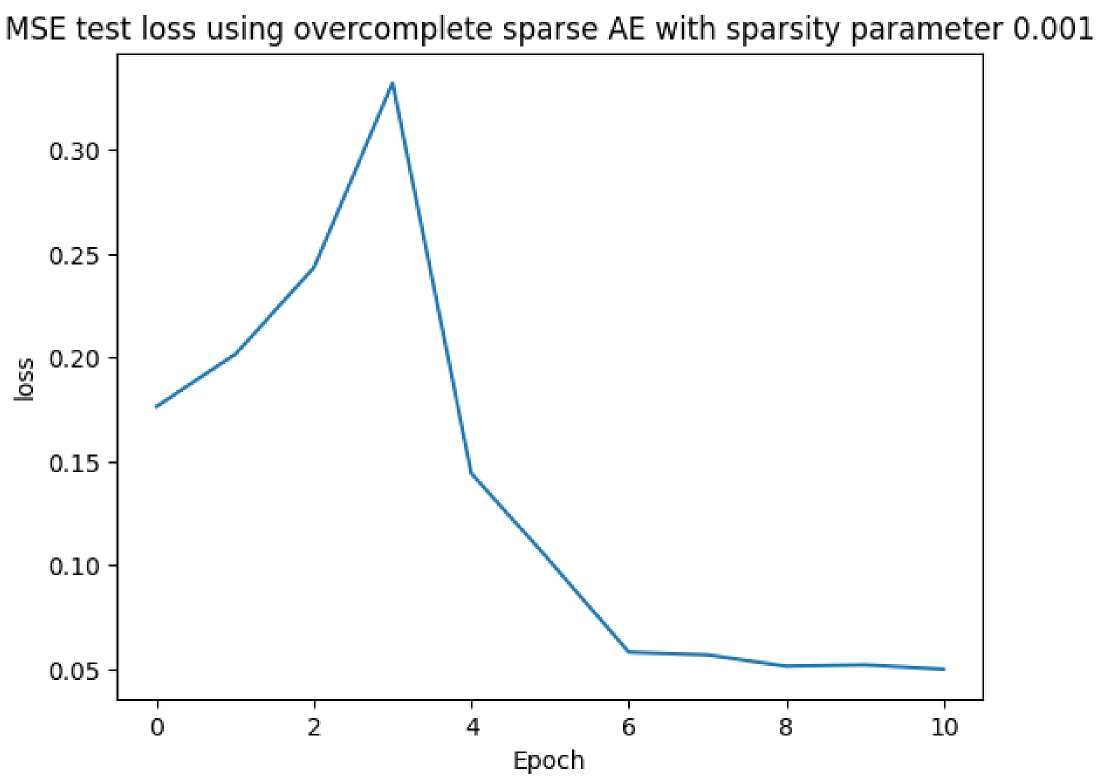

Activation for digit 7
0
5
10
15
20
25
30
0
5
10
15
20
25
30

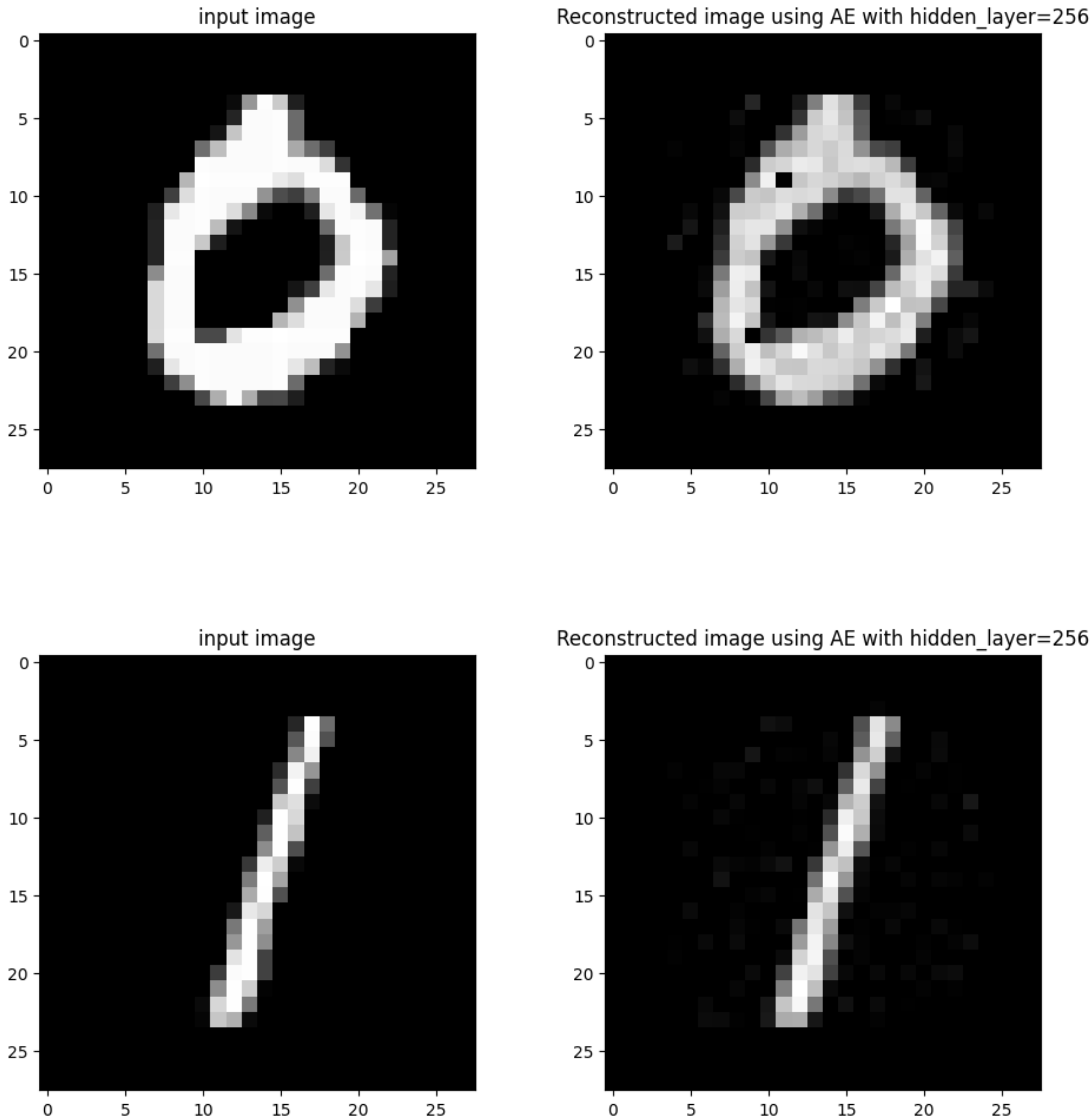

Activation for digit 8
0
5
10
15
20
25
30
0
5
10
15
20
25
30

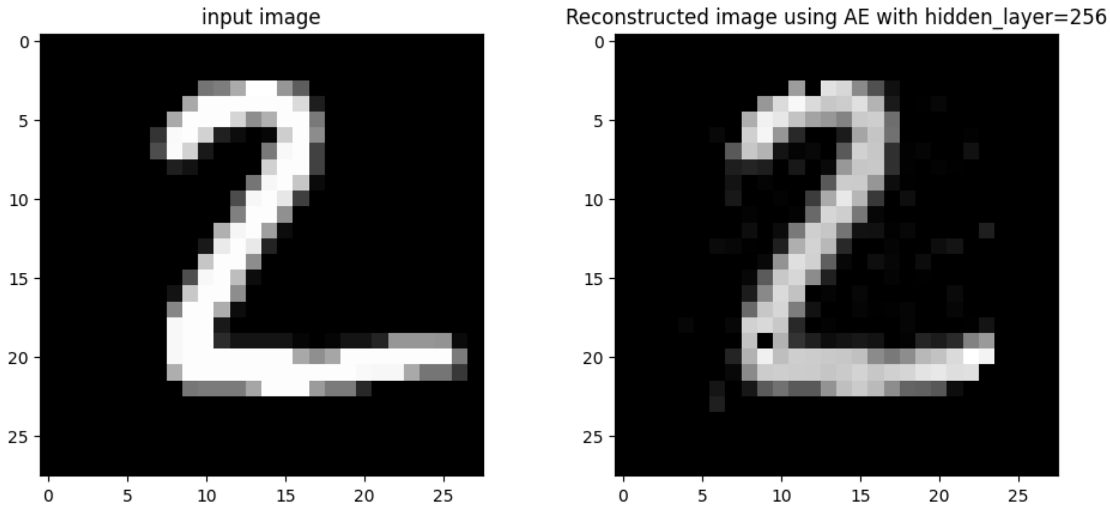
Activation for digit 9

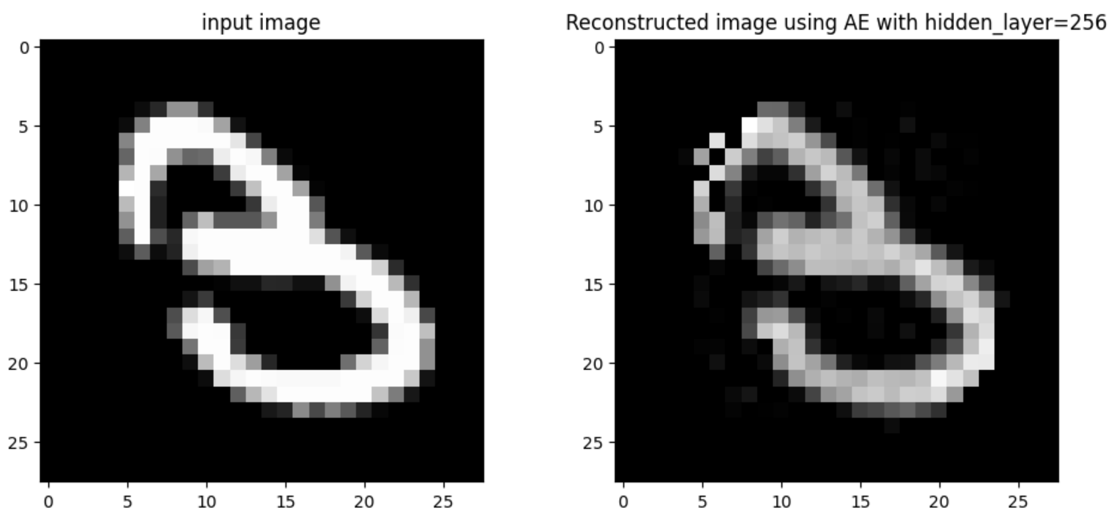


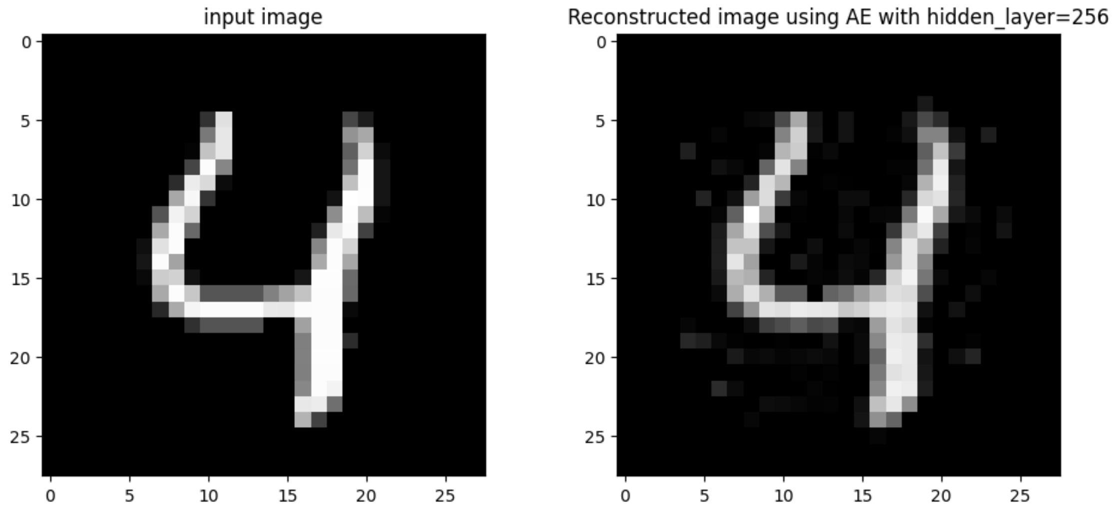

MSE train loss using overcomplete sparse AE with sparsity parameter 0.001
Loss
Epoch
0.25
0.20
0.15
0.10
0.05
0
2
4
6
8
10

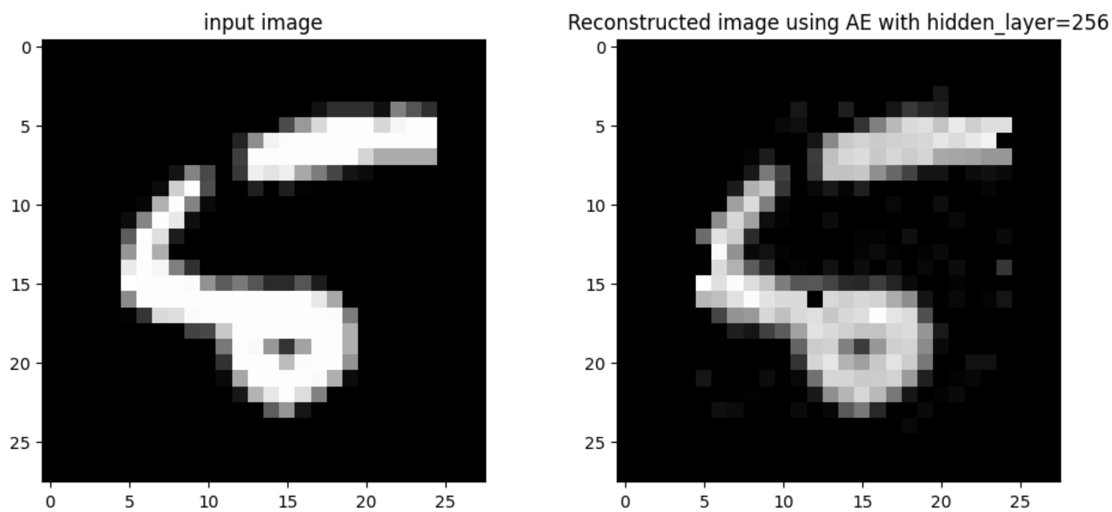

MSE test loss using overcomplete sparse AE with sparsity parameter 0.001
loss
Epoch
0.30
0.25
0.20
0.15
0.10
0.05
0
2
4
6
8
10

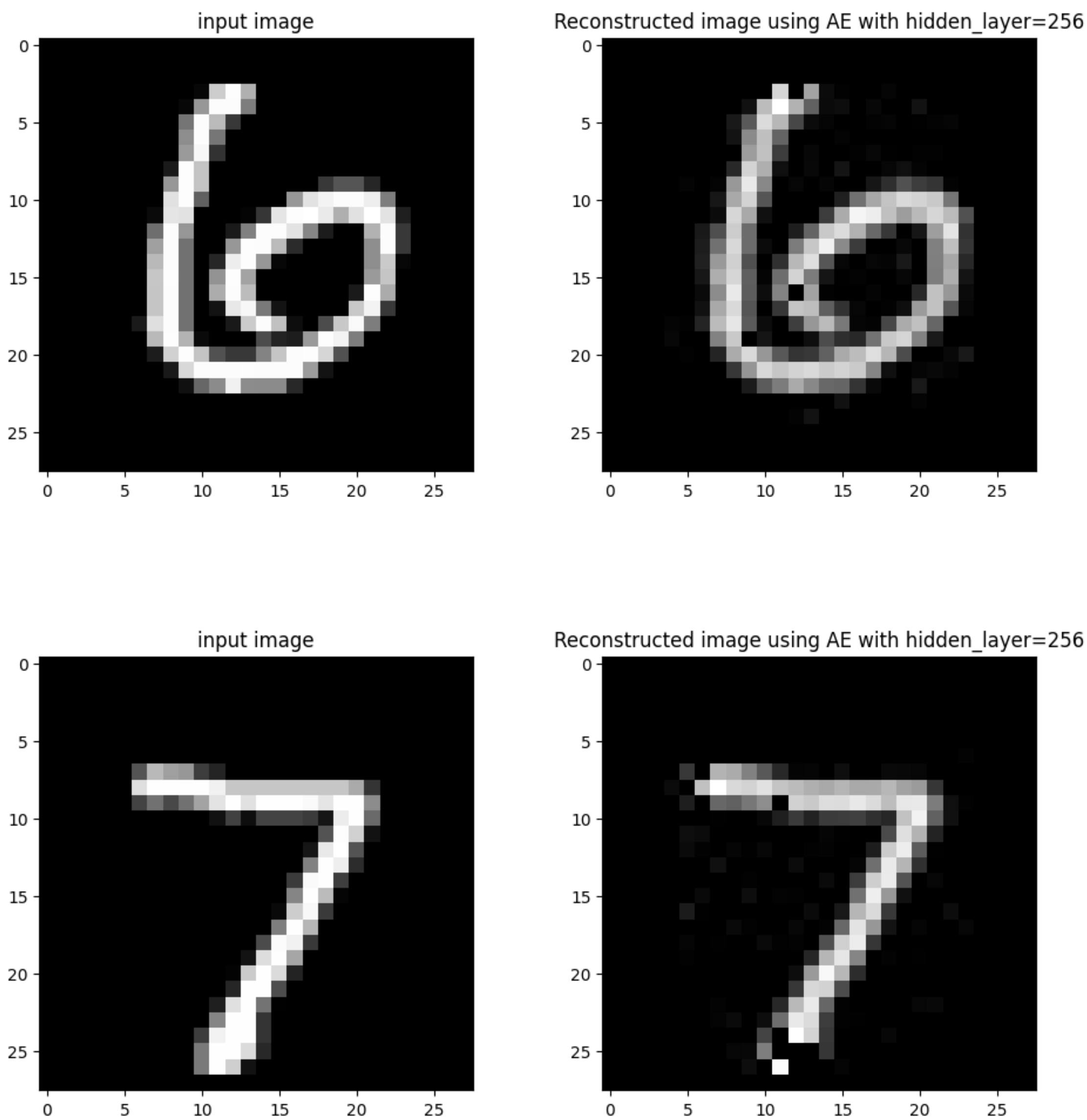

input image
Reconstructed image using AE with hidden_layer=256
input image
Reconstructed image using AE with hidden_layer=256

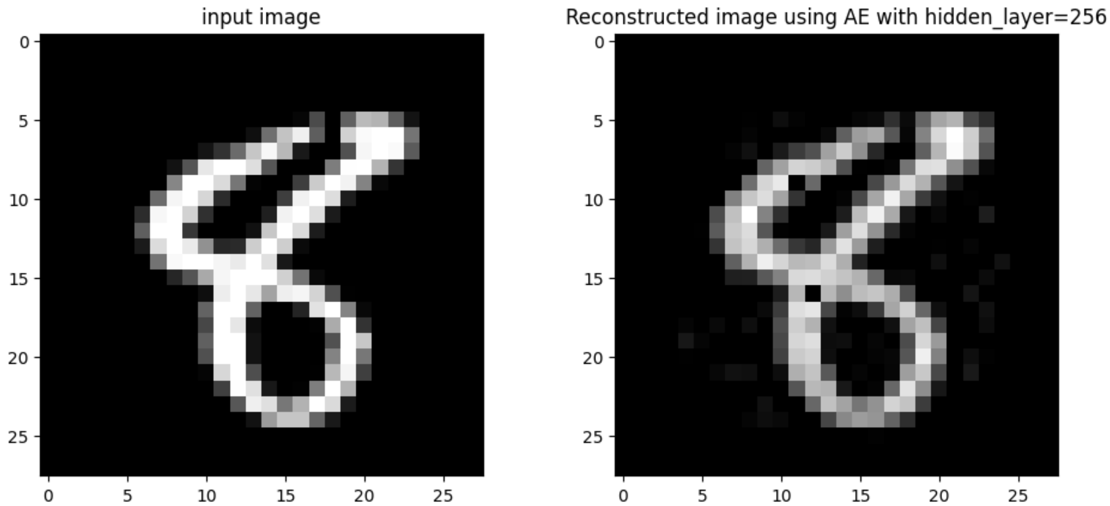

input image
Reconstructed image using AE with hidden_layer=256

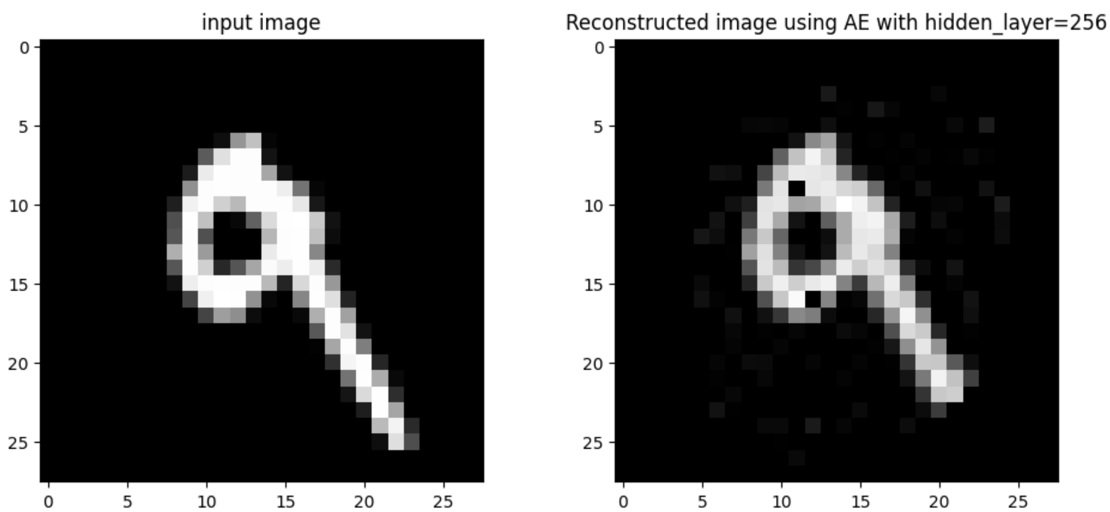

input image
Reconstructed image using AE with hidden_layer=256

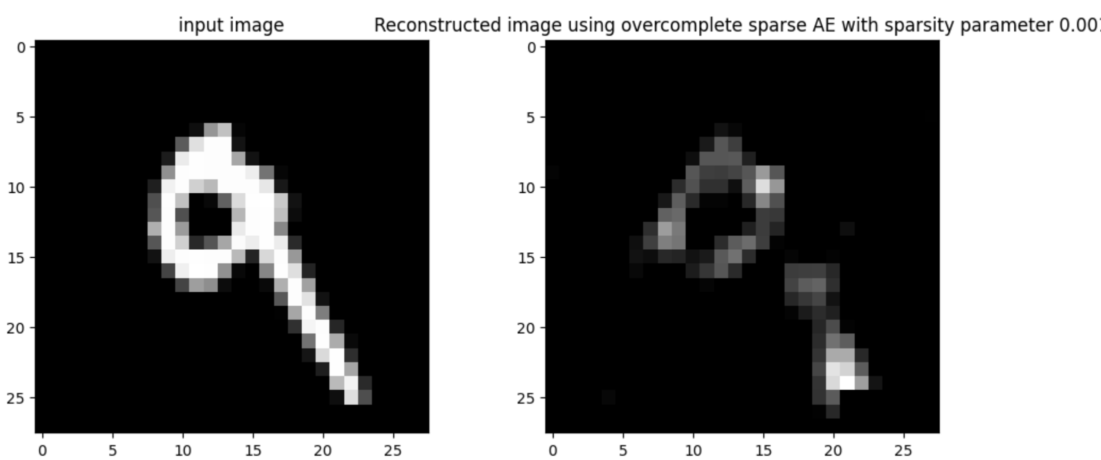

input image
Reconstructed image using AE with hidden_layer=256


input image
Reconstructed image using AE with hidden_layer=256

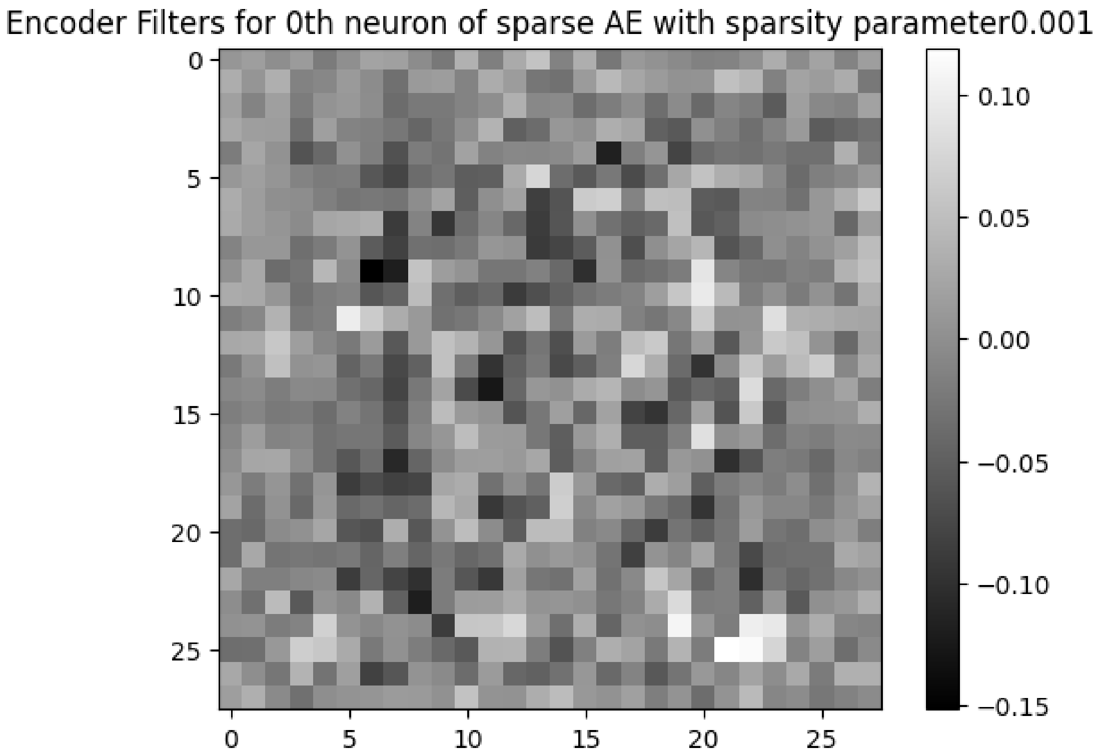

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25
input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

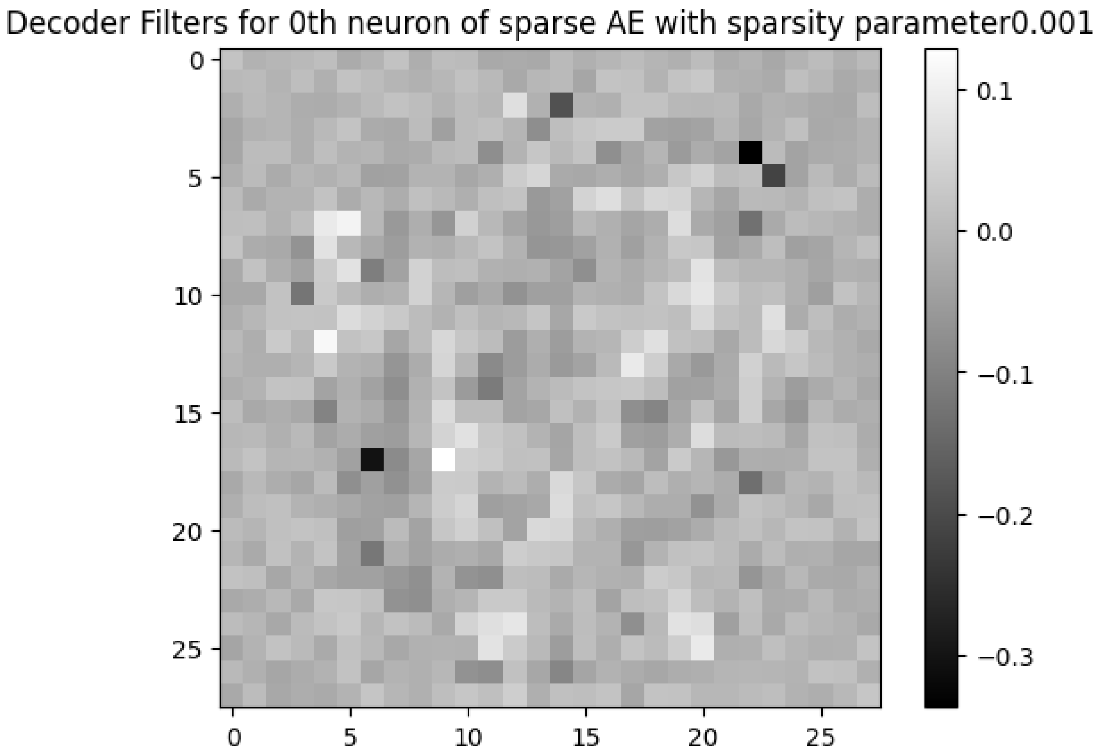

input image
Reconstructed image using AE with hidden_layer=256

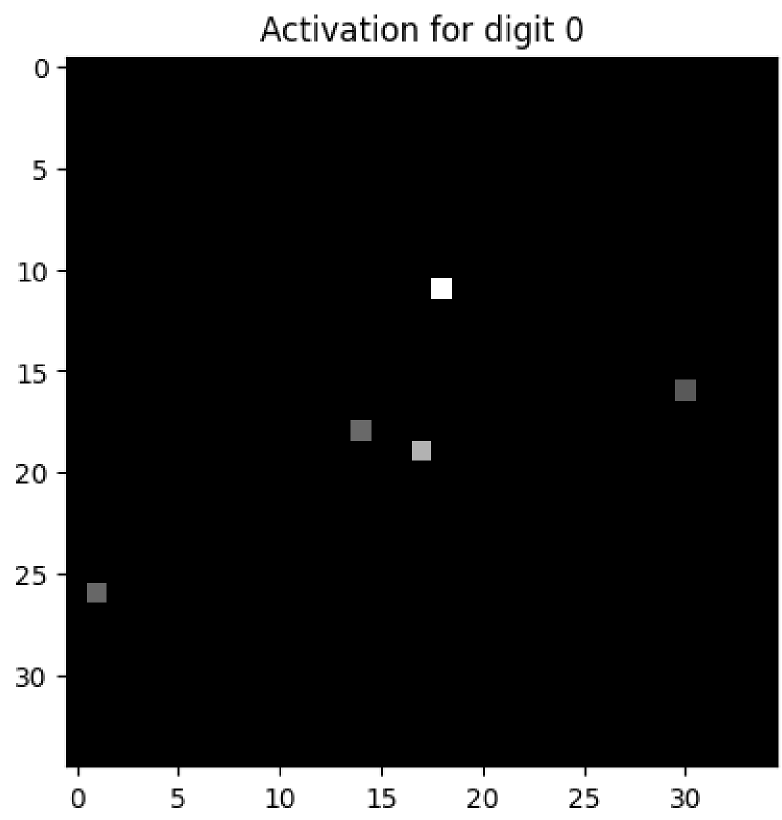

input image
Reconstructed image using AE with hidden_layer=256

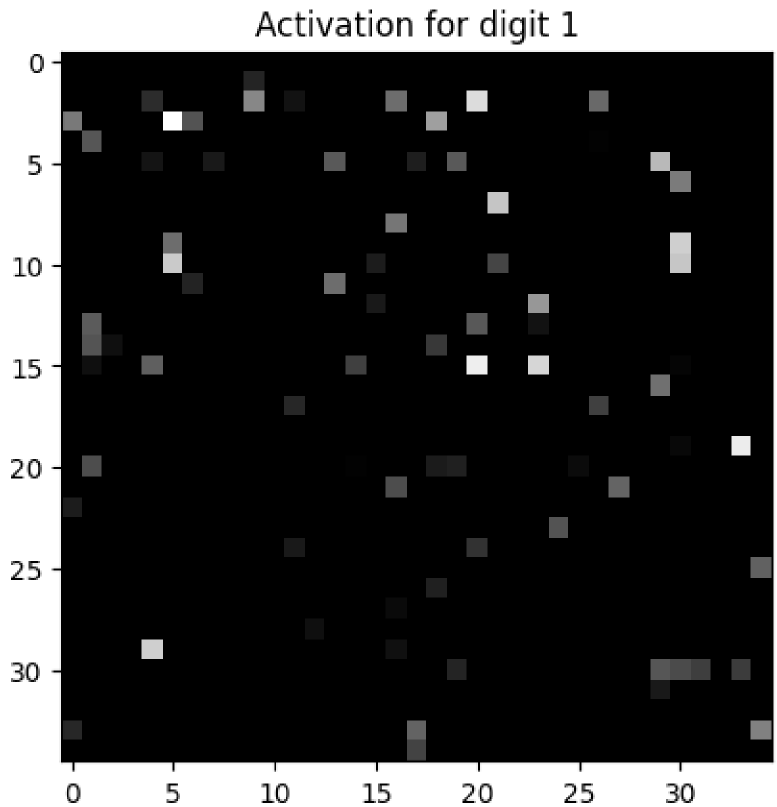

input image
Reconstructed image using overcomplete sparse AE with sparsity parameter 0.001

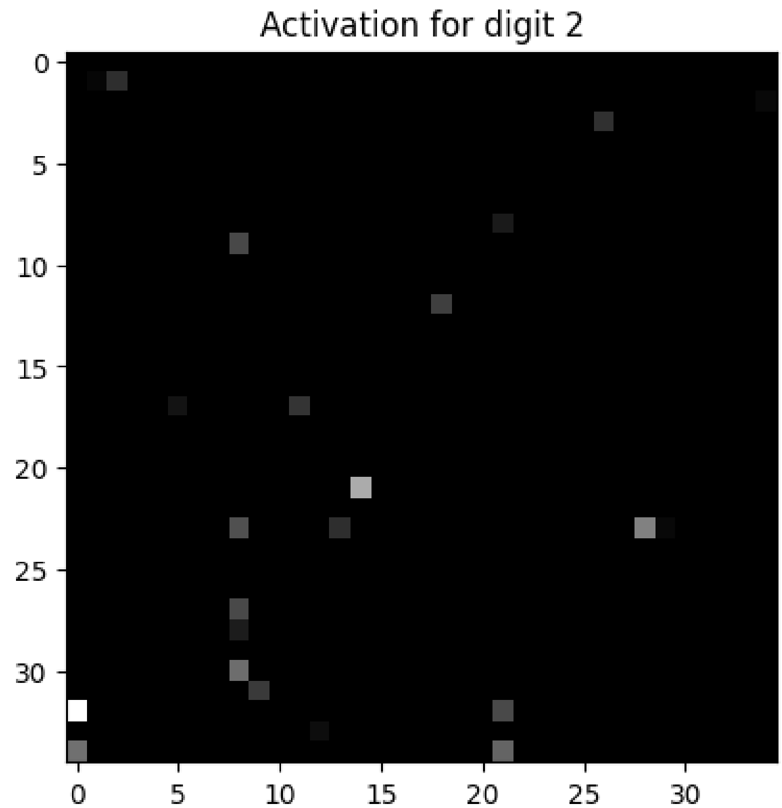
The average activation of overcomplete sparse AE with sparsity parameter 0.001 is 0.004904146178159863



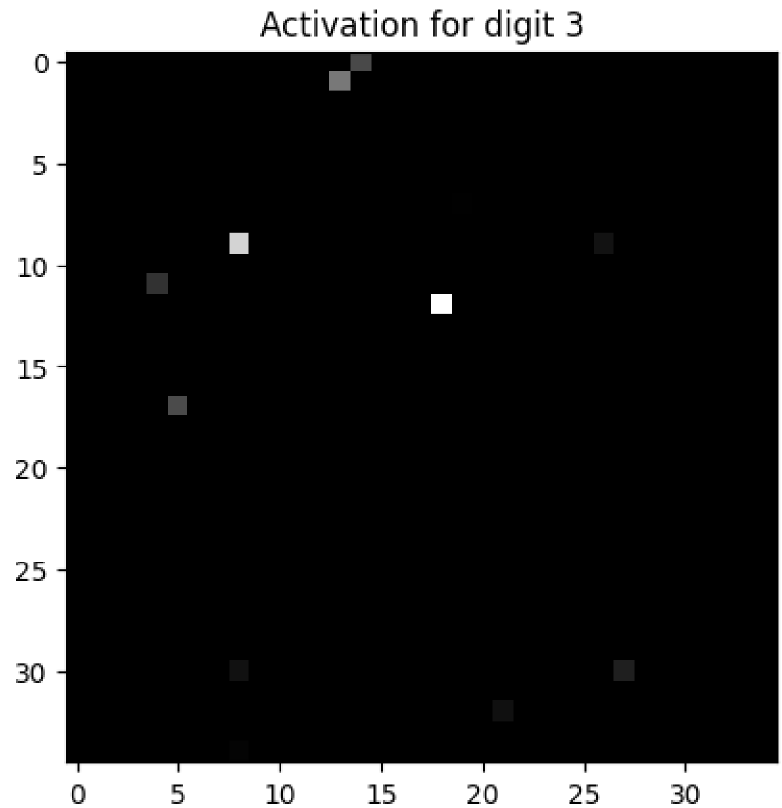

Decoder Filters for 0th neuron of sparse AE with sparsity parameter0.001
0
5
10
15
20
25
0
5
10
15
20
25
0.1
0.0
−0.1
−0.2
−0.3


Activation for digit 0
0
5
10
15
20
25
30
0
5
10
15
20
25
30

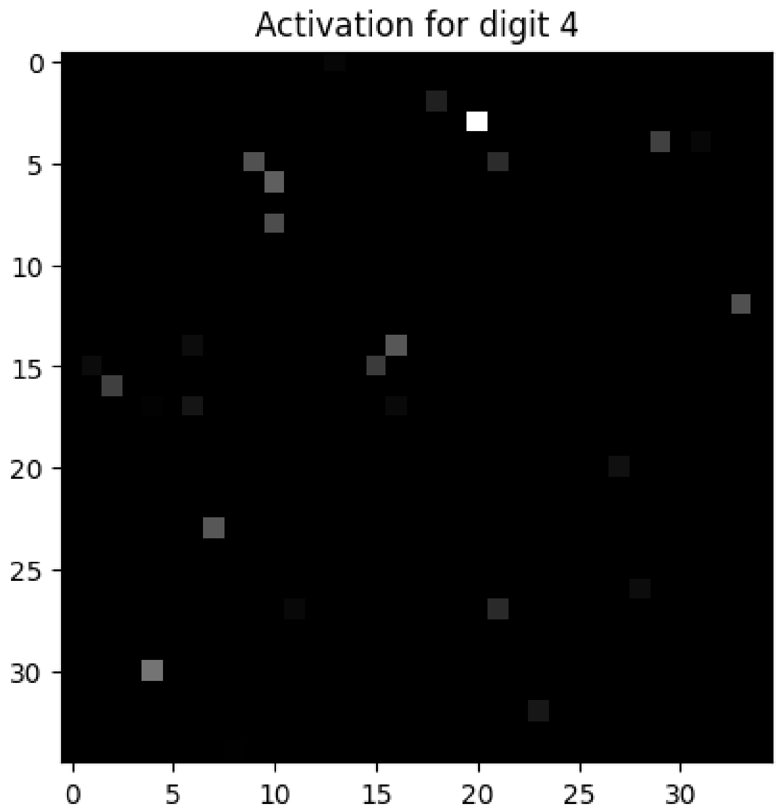

Activation for digit 1
0
5
10
15
20
25
30
0
5
10
15
20
25
30

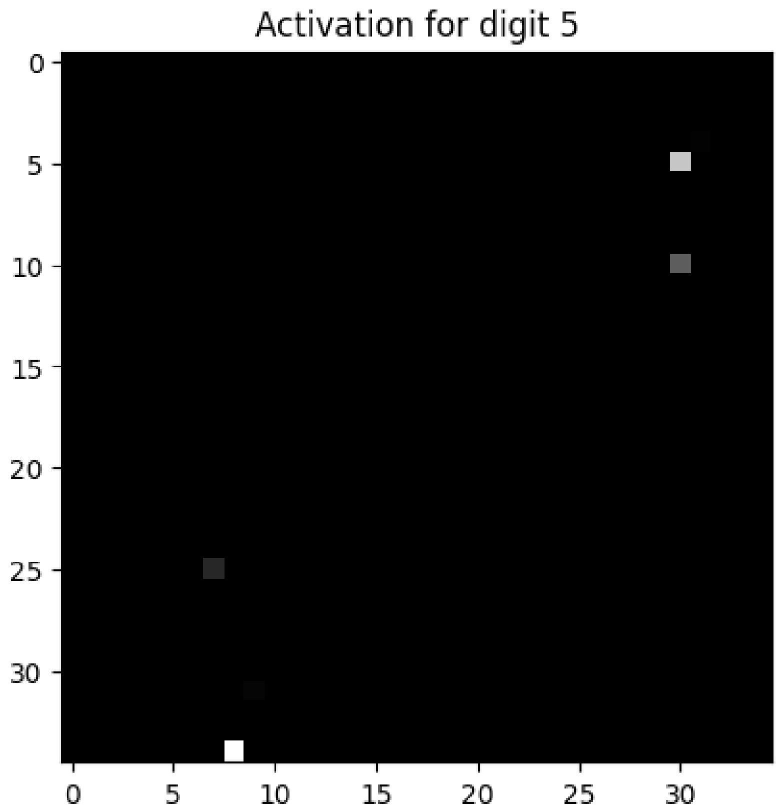


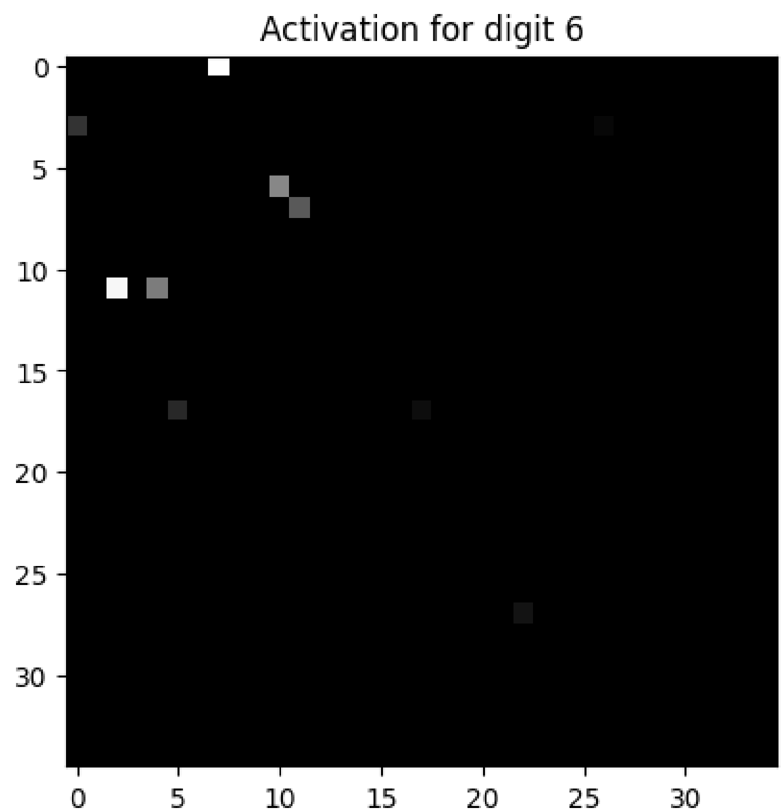
Activation for digit 2
0
5
10
15
20
25
30
0
5
10
15
20
25
30

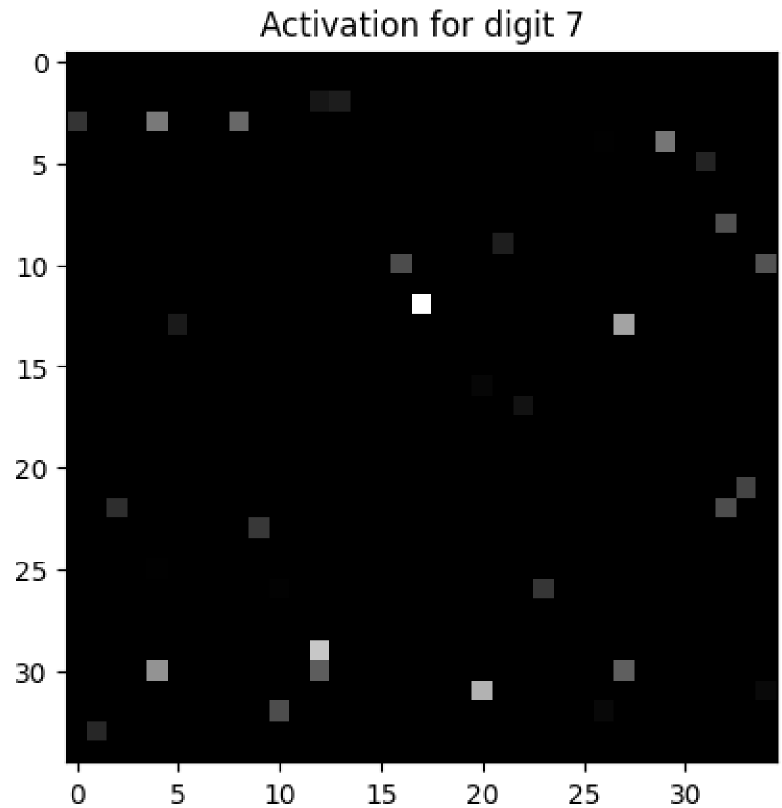


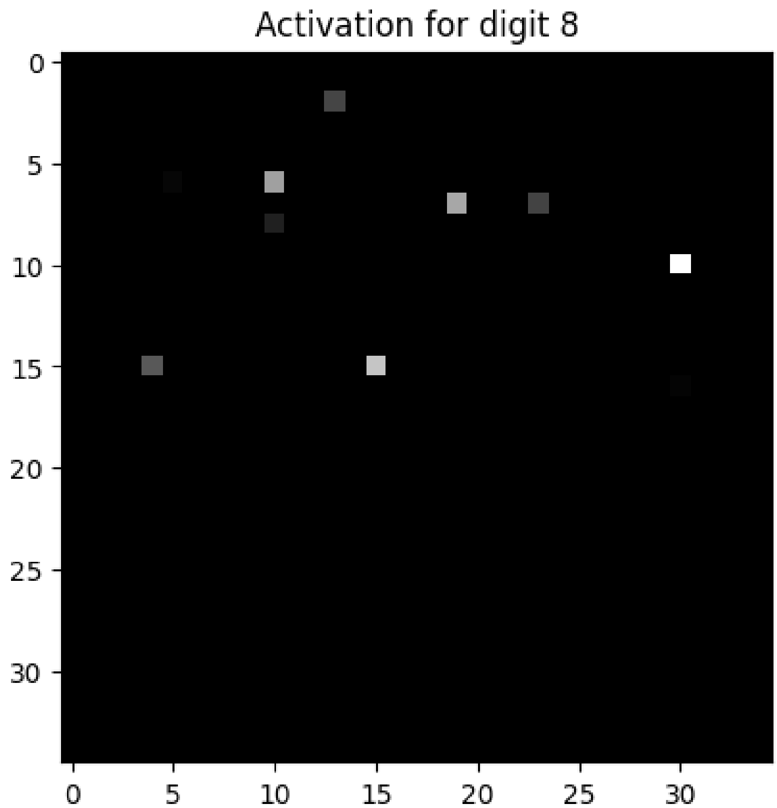
Activation for digit 3
0
5
10
15
20
25
30
0
5
10
15
20
25
30

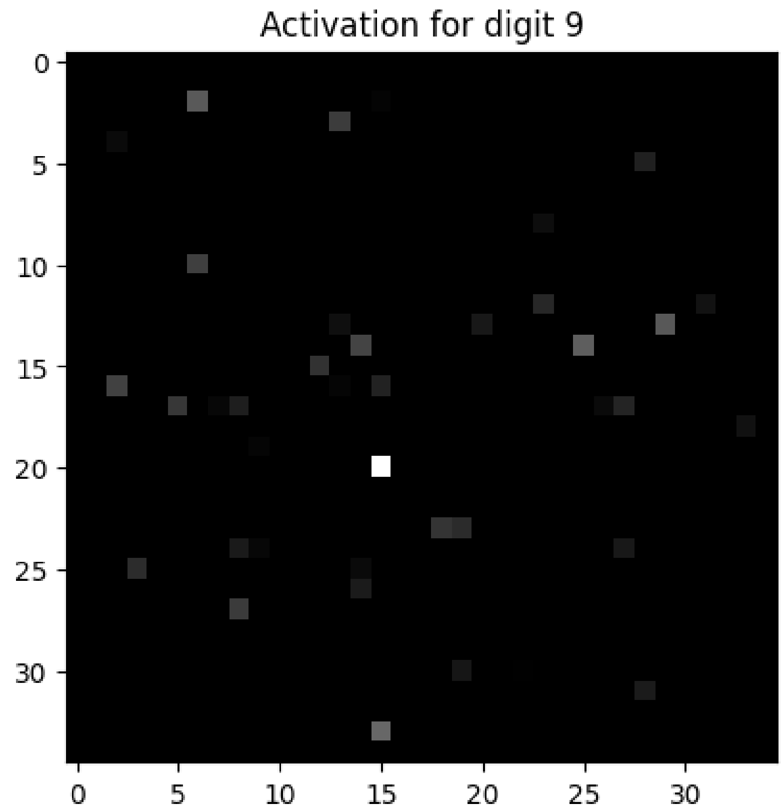

Activation for digit 4
0
5
10
15
20
25
30
0
5
10
15
20
25
30

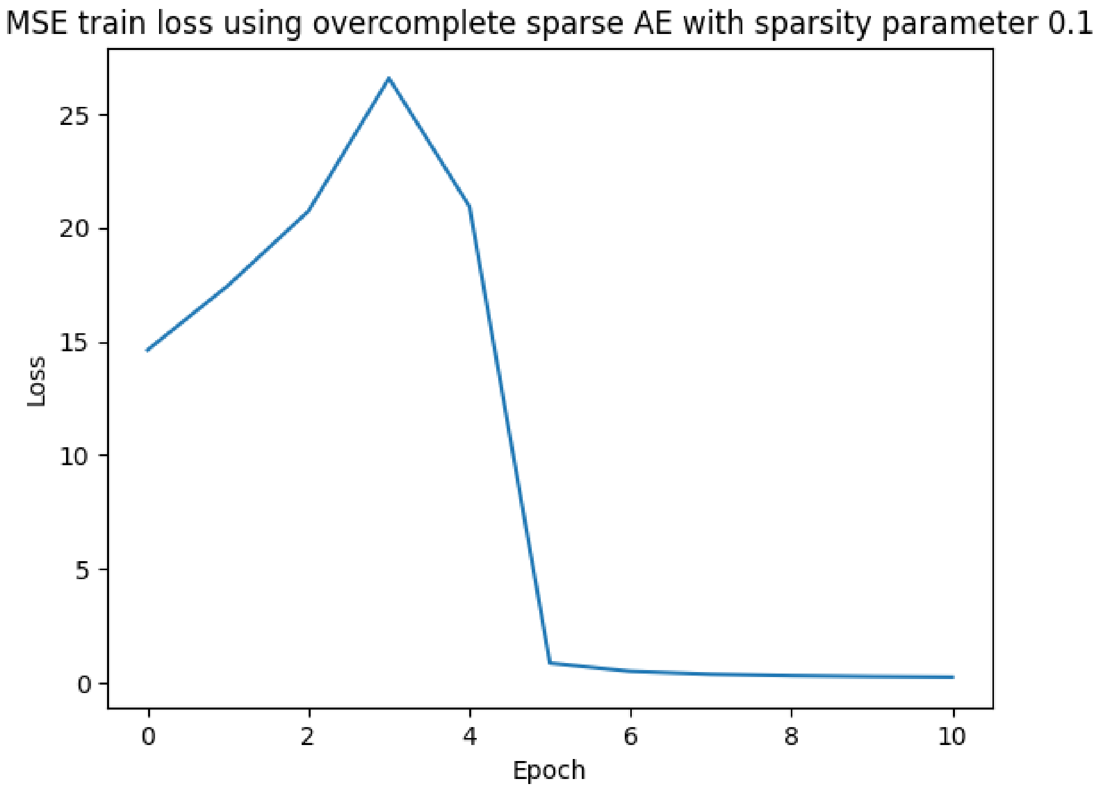

Activation for digit 5
0
5
10
15
20
25
30
0
5
10
15
20
25
30

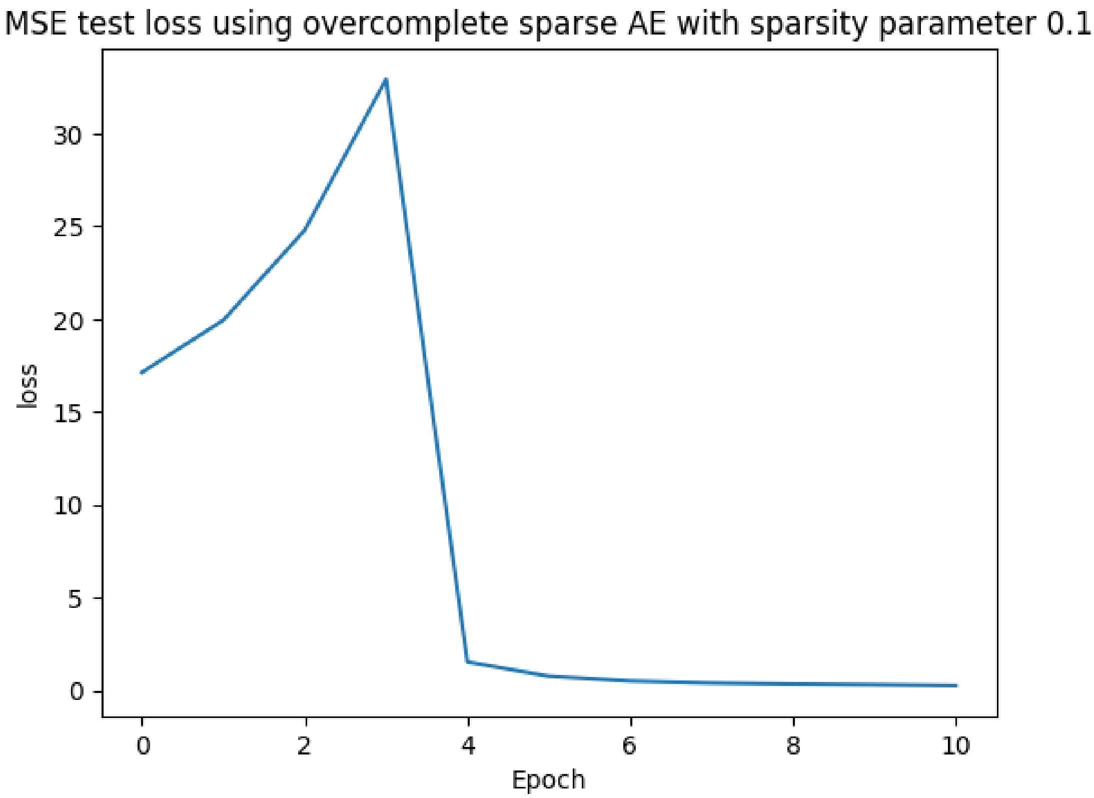

Activation for digit 6
0
5
10
15
20
25
30
0
5
10
15
20
25
30

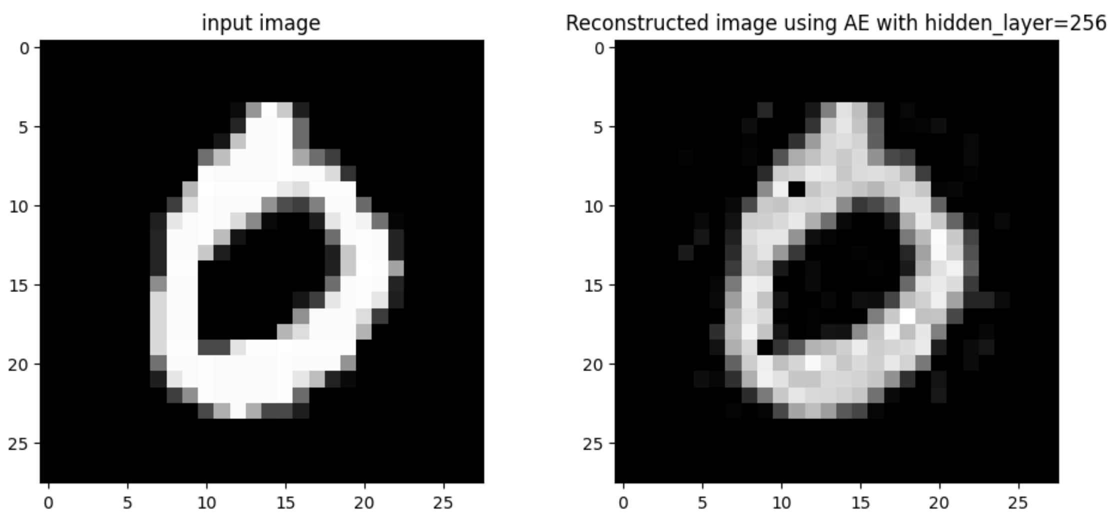

Activation for digit 7
0
5
10
15
20
25
30
0
5
10
15
20
25
30

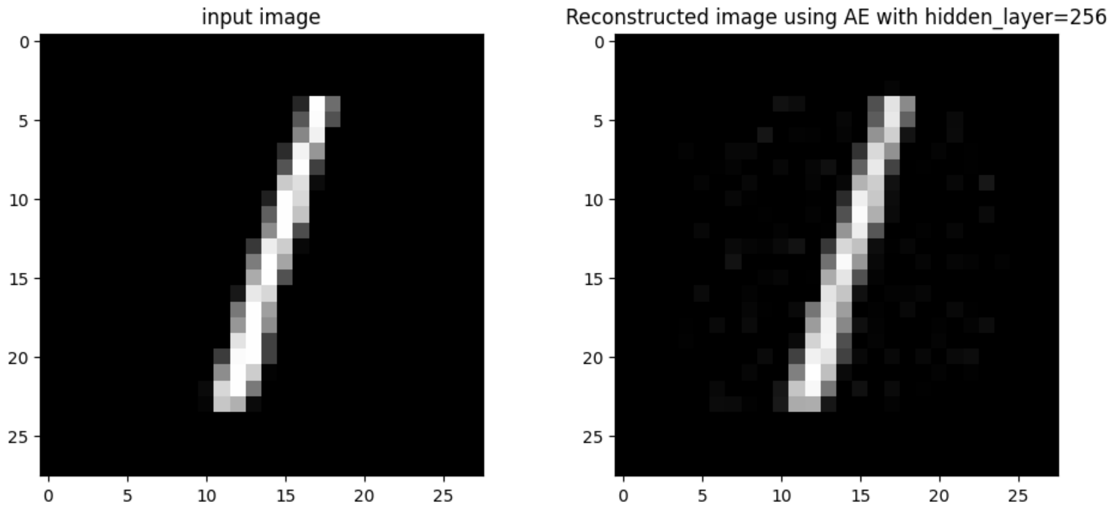

Activation for digit 8
0
5
10
15
20
25
30
0
5
10
15
20
25
30

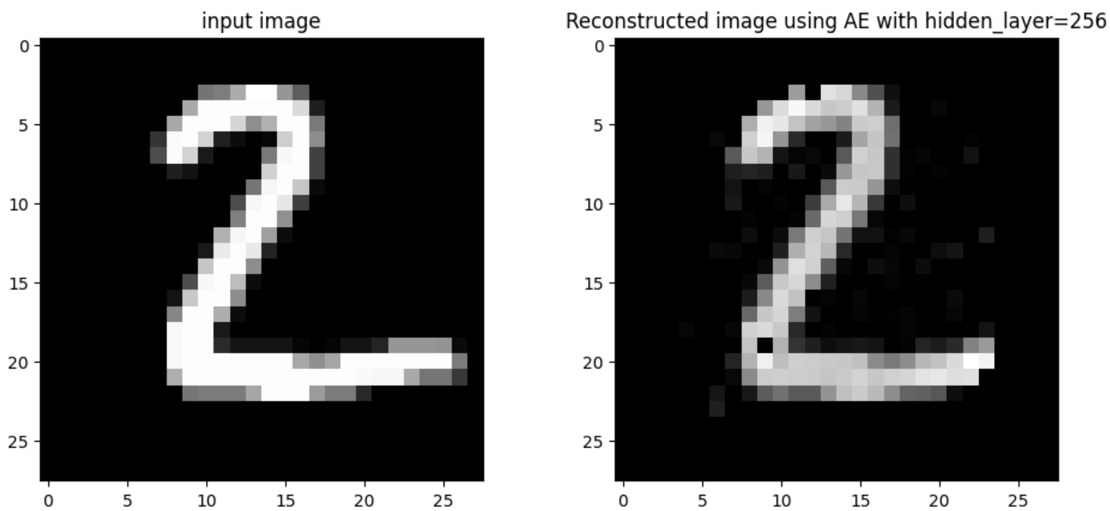

Activation for digit 9
0
5
10
15
20
25
30
0
5
10
15
20
25
30

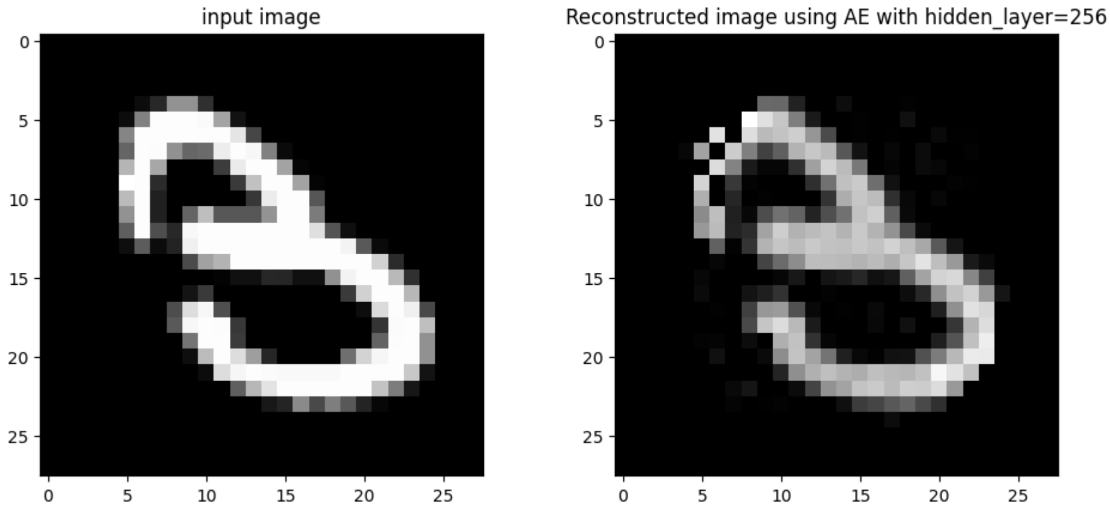

MSE train loss using overcomplete sparse AE with sparsity parameter 0.1
Loss
25
20
15
10
5
0
0
2
4
6
8
10
Epoch

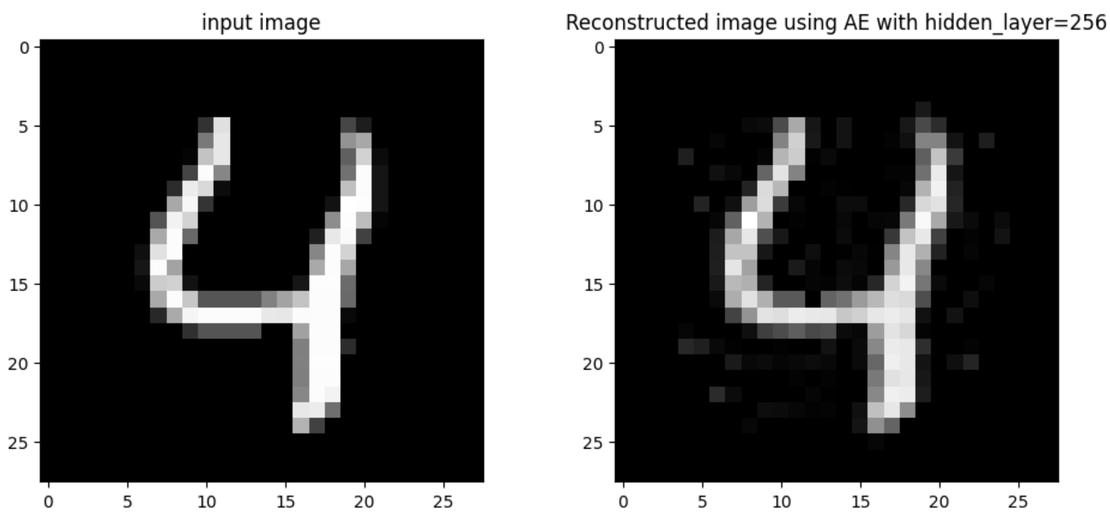

MSE test loss using overcomplete sparse AE with sparsity parameter 0.1
loss
Epoch
0
5
10
15
20
25
30
0
2
4
6
8
10

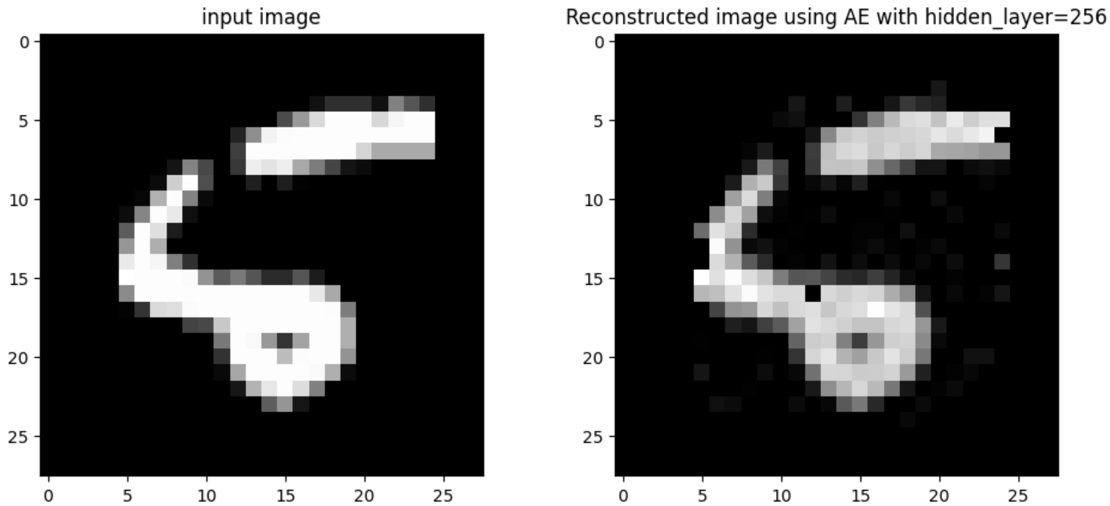

input image
Reconstructed image using AE with hidden_layer=256

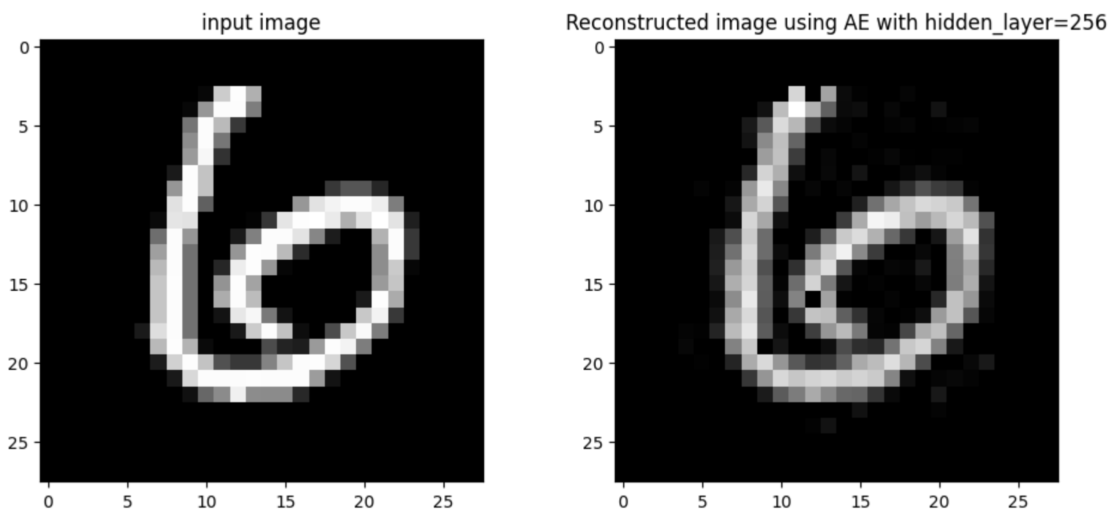

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

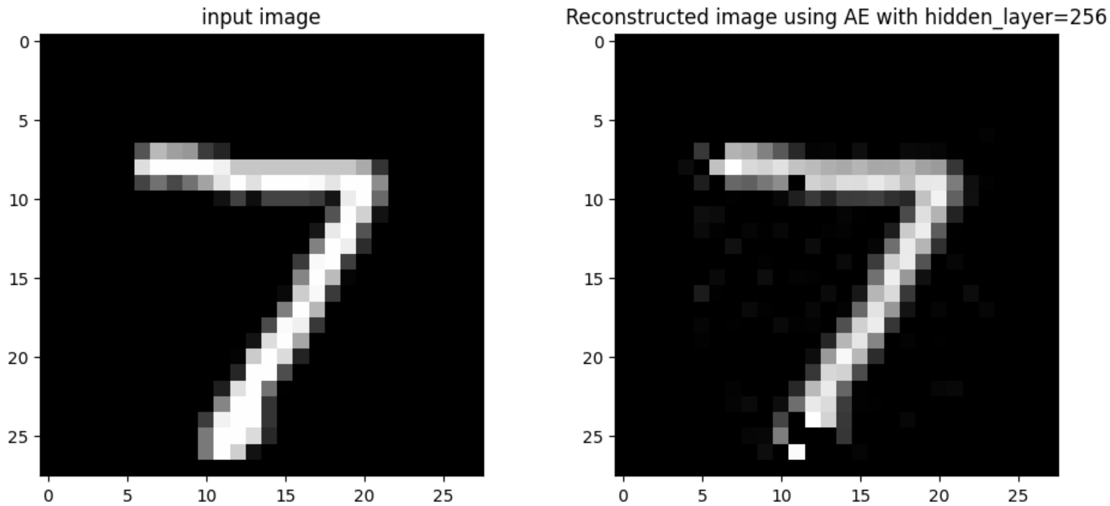

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

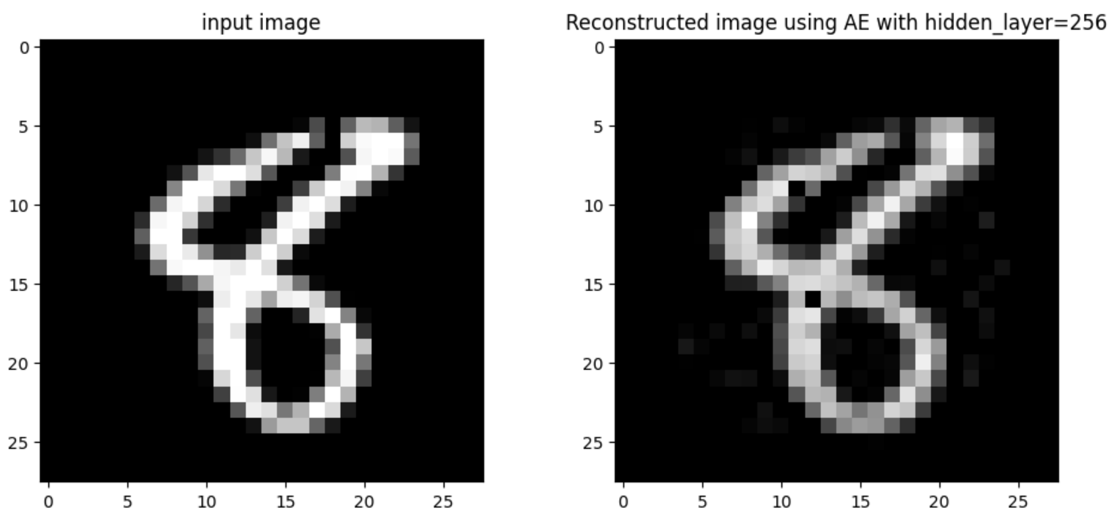

input image
Reconstructed image using AE with hidden_layer=256

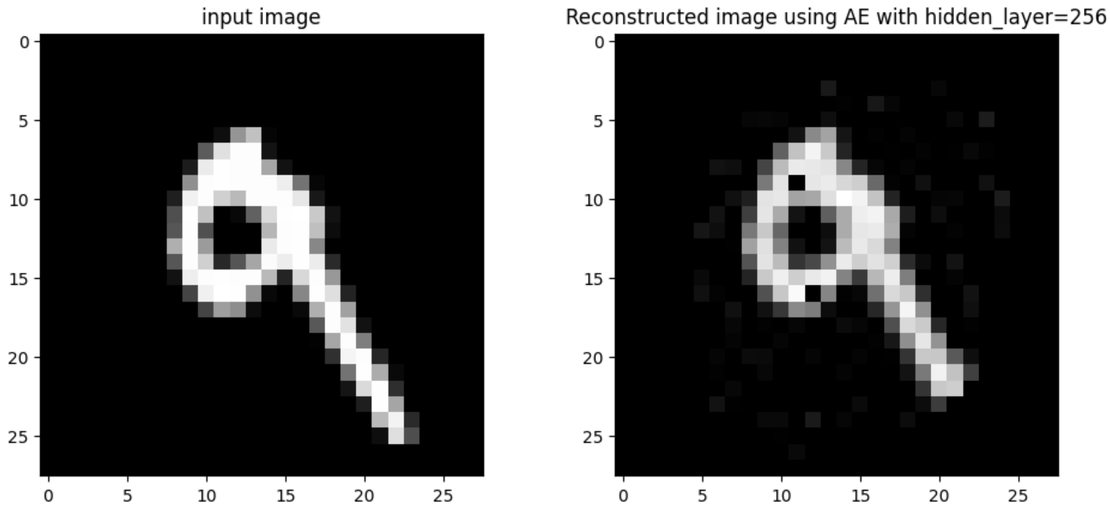

input image
Reconstructed image using AE with hidden_layer=256

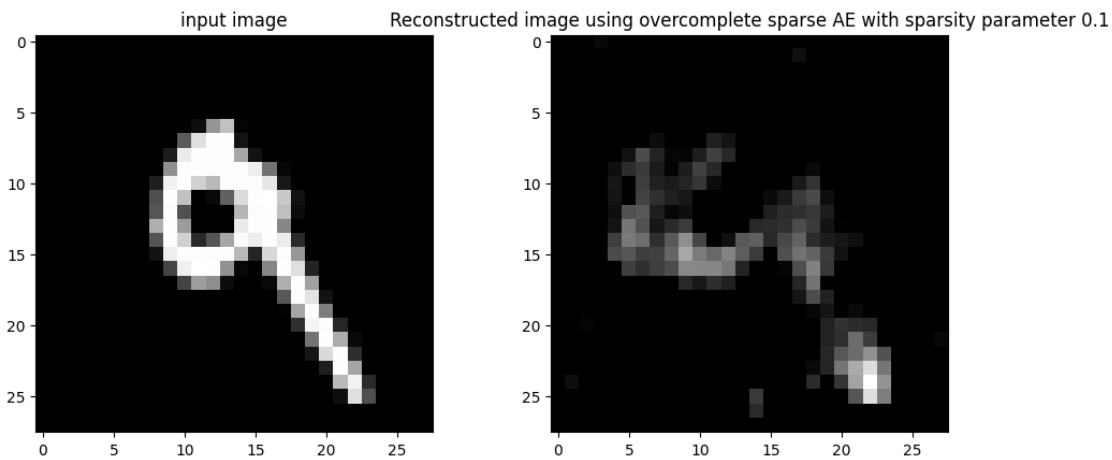

input image
Reconstructed image using AE with hidden_layer=256


input image
Reconstructed image using AE with hidden_layer=256

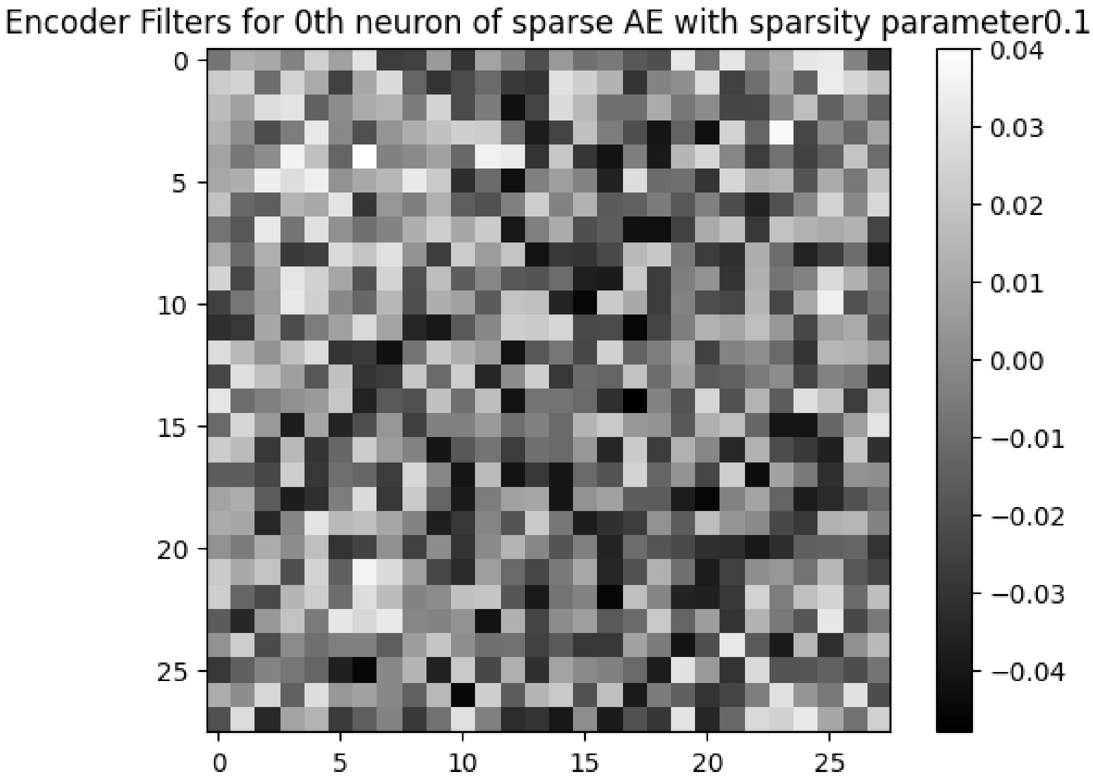

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

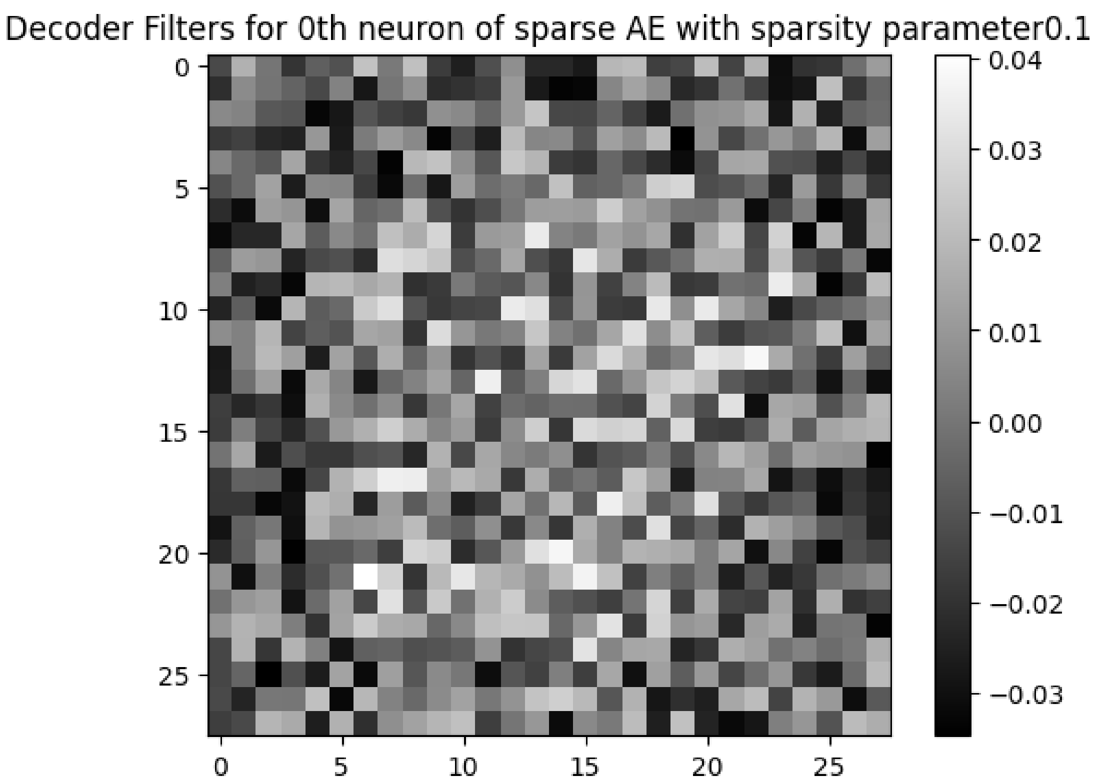

input image
Reconstructed image using AE with hidden_layer=256
0
5
10
15
20
25

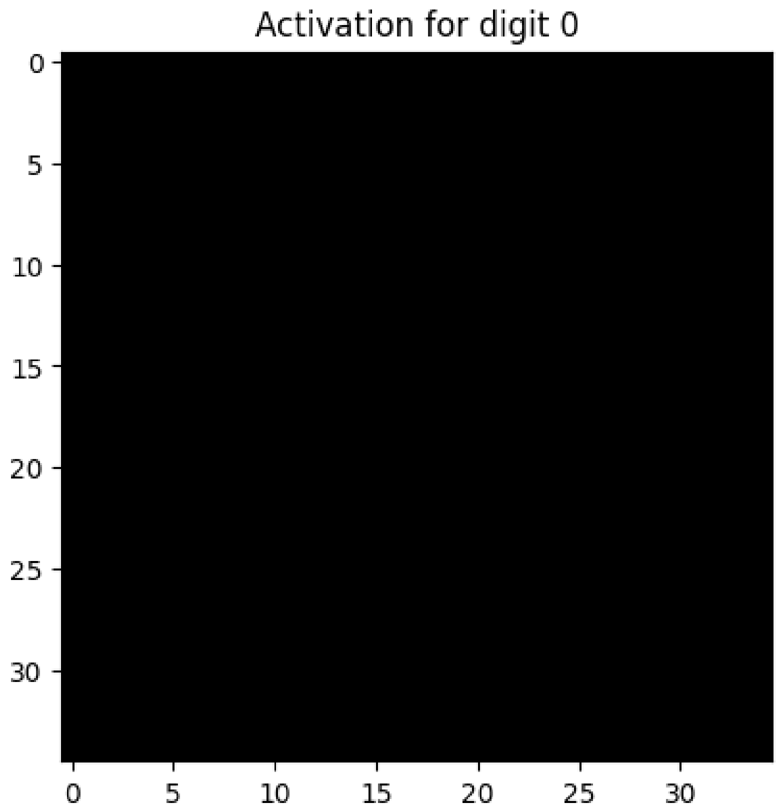


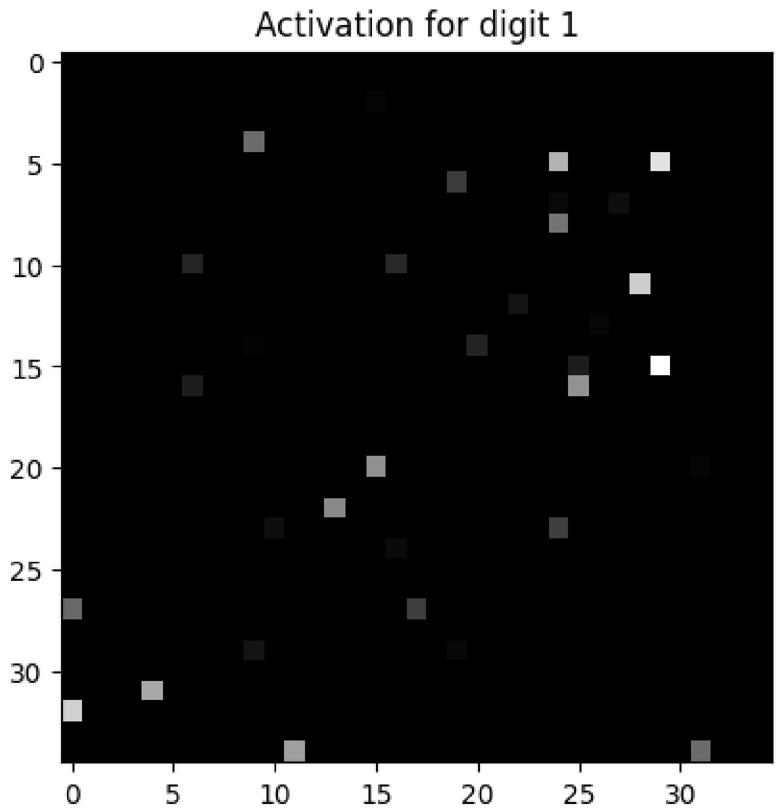


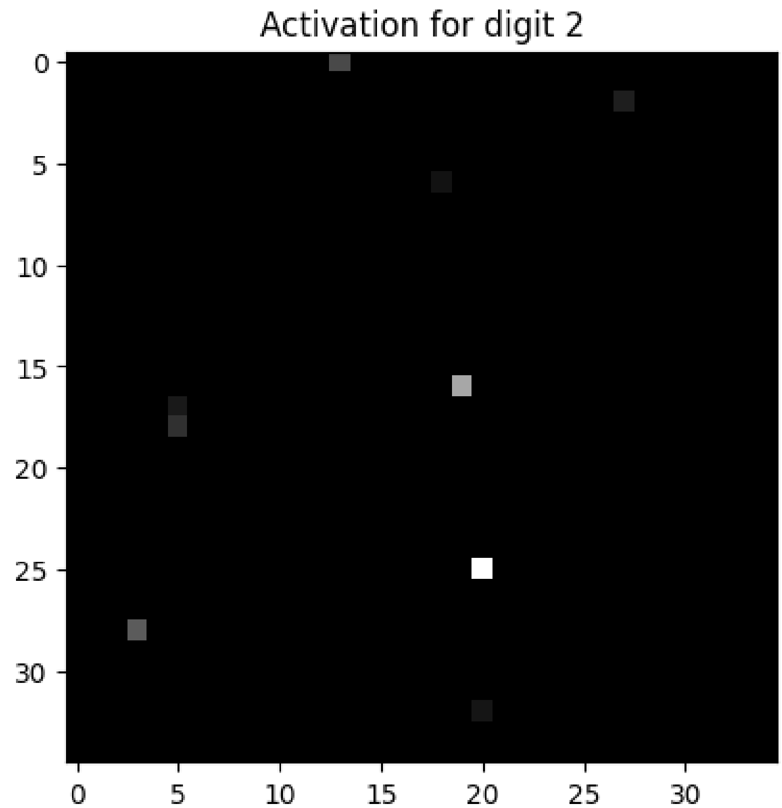

```
The average activation of overcomplete sparse AE with sparsity parameter 0.1 is
0.0004624879096809309
```

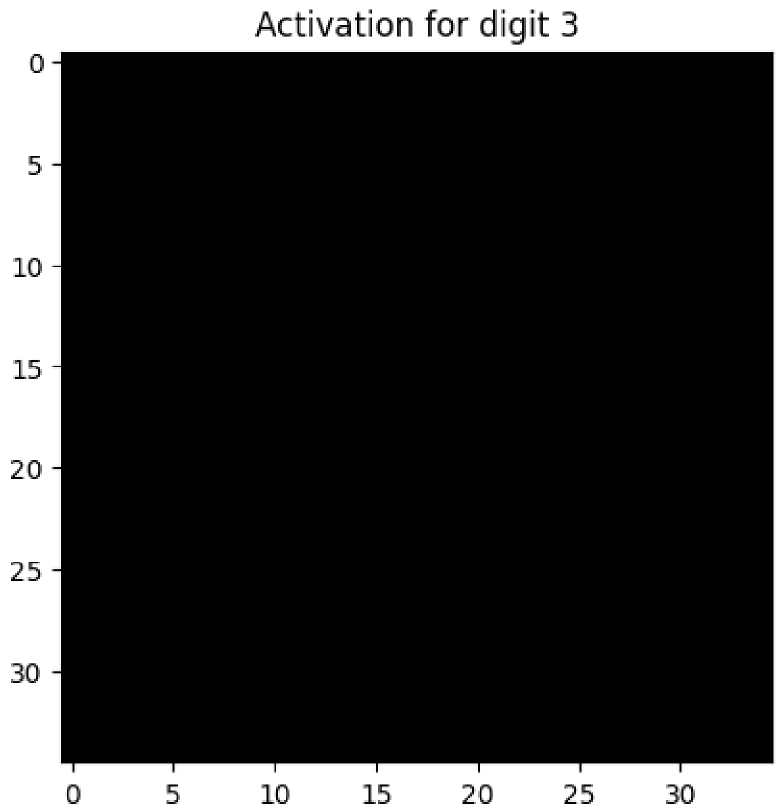

Encoder Filters for 0th neuron of sparse AE with sparsity parameter0.1
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04

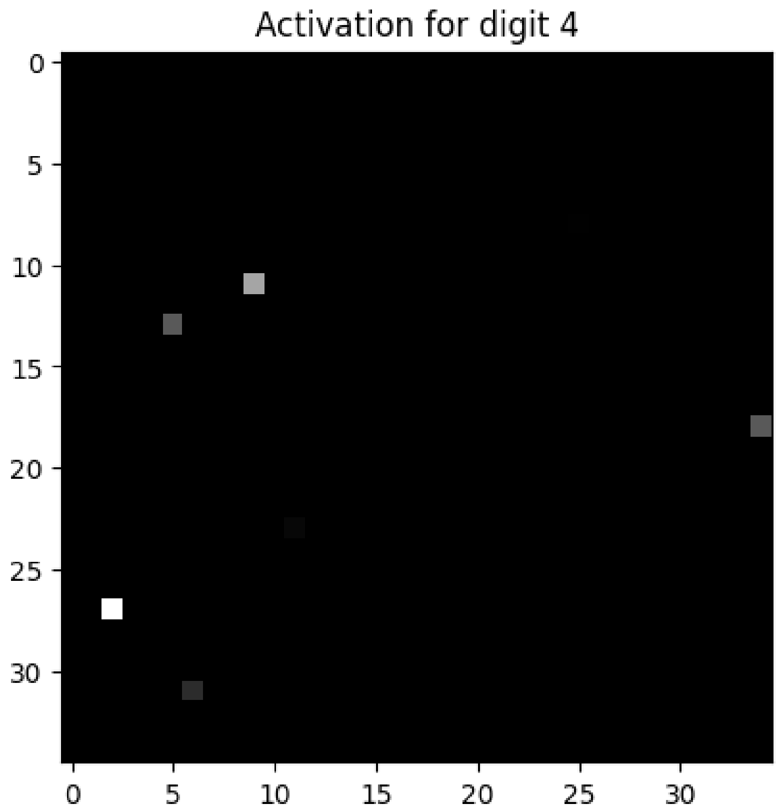

Decoder Filters for 0th neuron of sparse AE with sparsity parameter0.1
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03

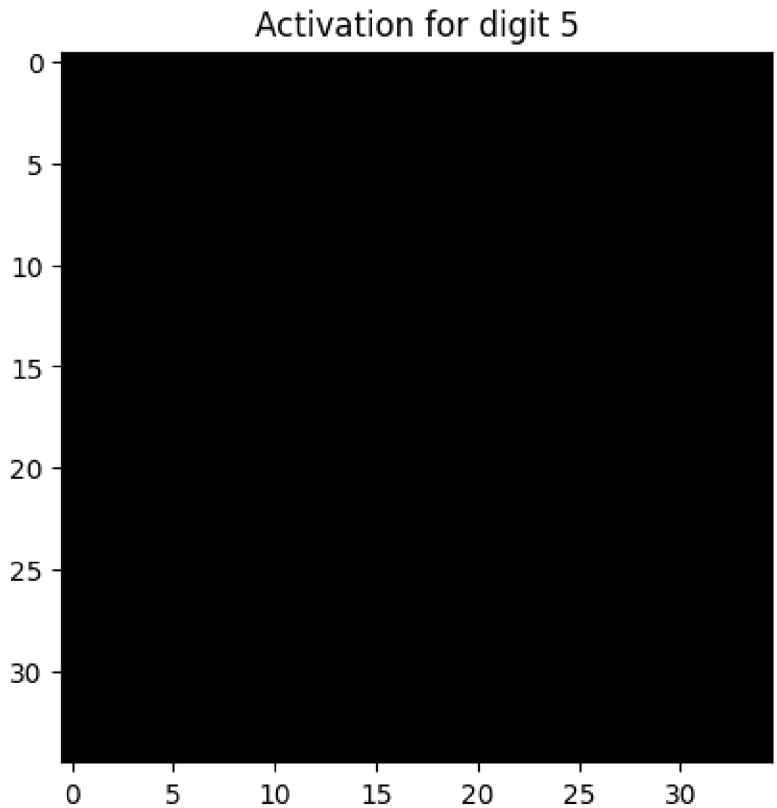

Activation for digit 0
0
5
10
15
20
25
30
0
5
10
15
20
25
30

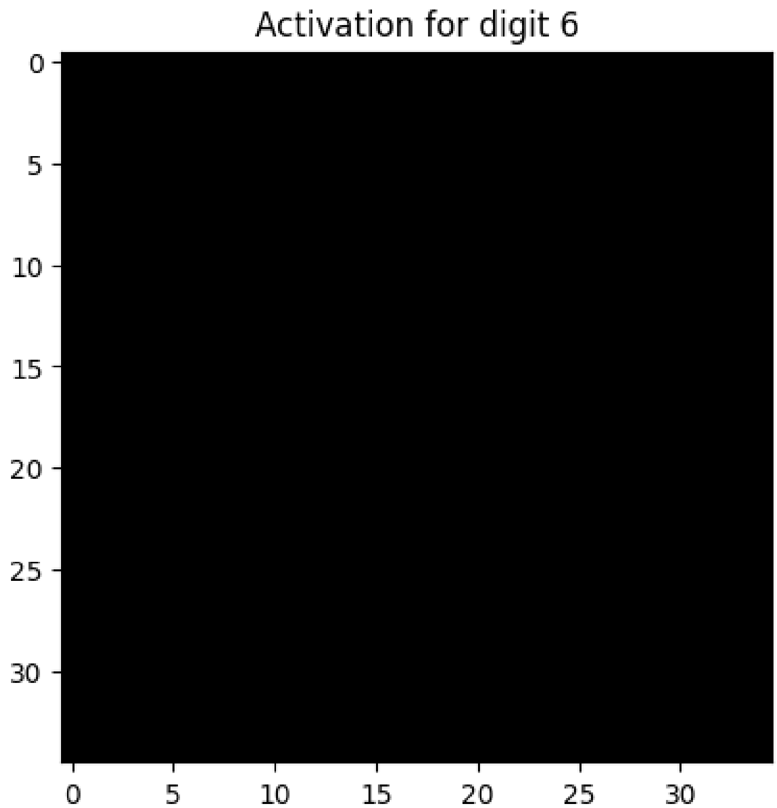

Activation for digit 1
0
5
10
15
20
25
30

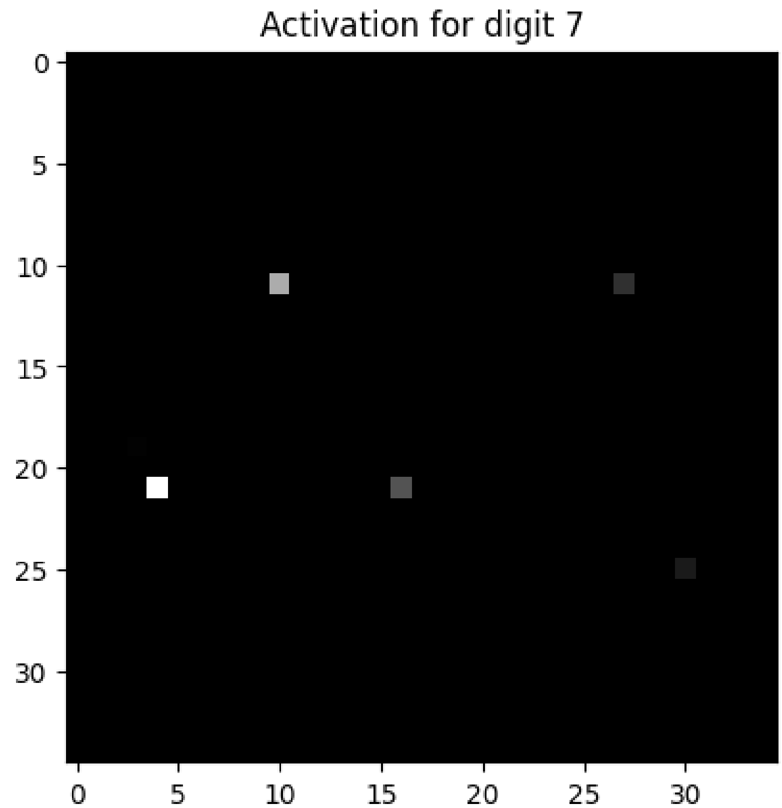

Activation for digit 2
0
5
10
15
20
25
30
0
5
10
15
20
25
30

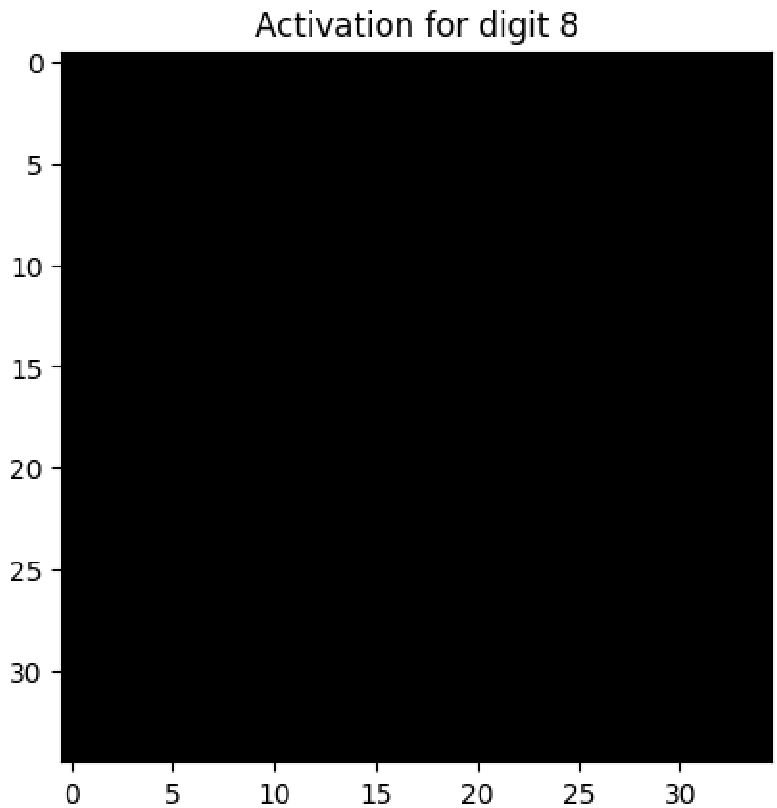

Activation for digit 3
0
5
10
15
20
25
30

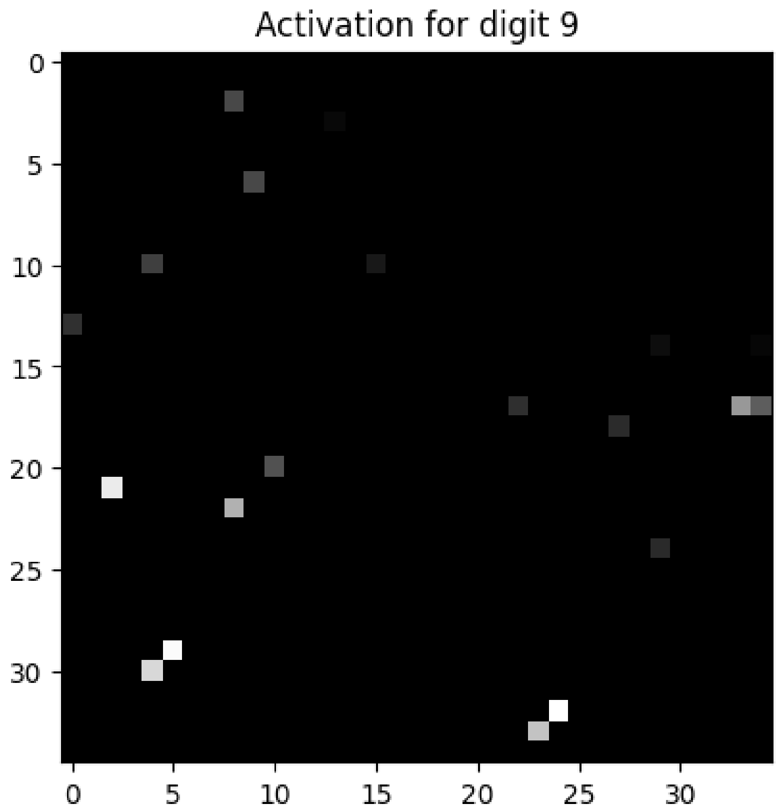

Activation for digit 4
0
5
10
15
20
25
30
0
5
10
15
20
25
30


Activation for digit 5
0
5
10
15
20
25
30


Activation for digit 6
0
5
10
15
20
25
30
0
5
10
15
20
25
30

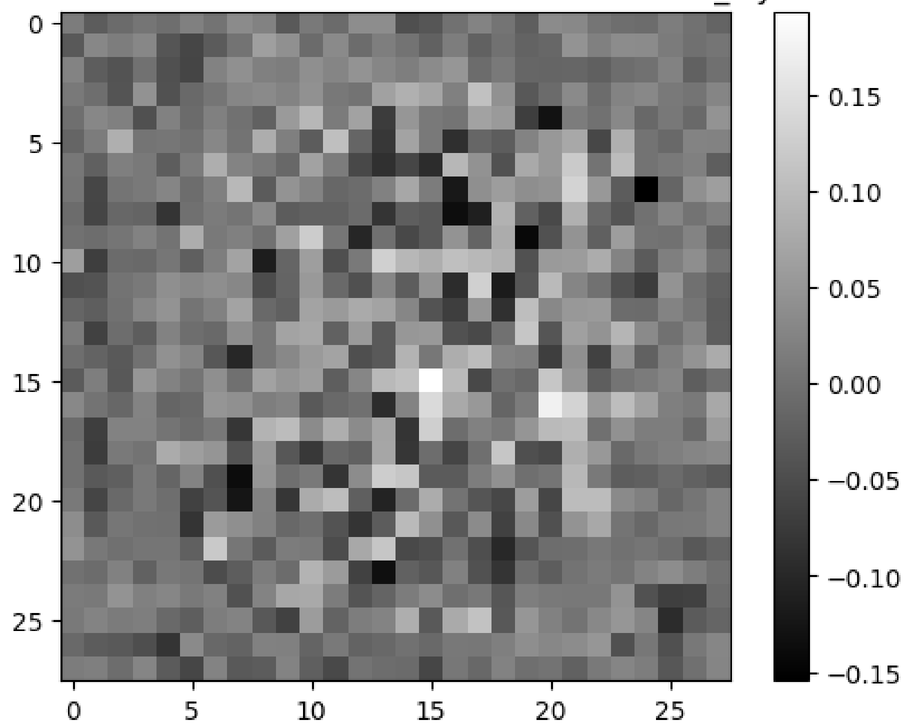

Activation for digit 7


Activation for digit 8
0
5
10
15
20
25
30
0
5
10
15
20
25
30

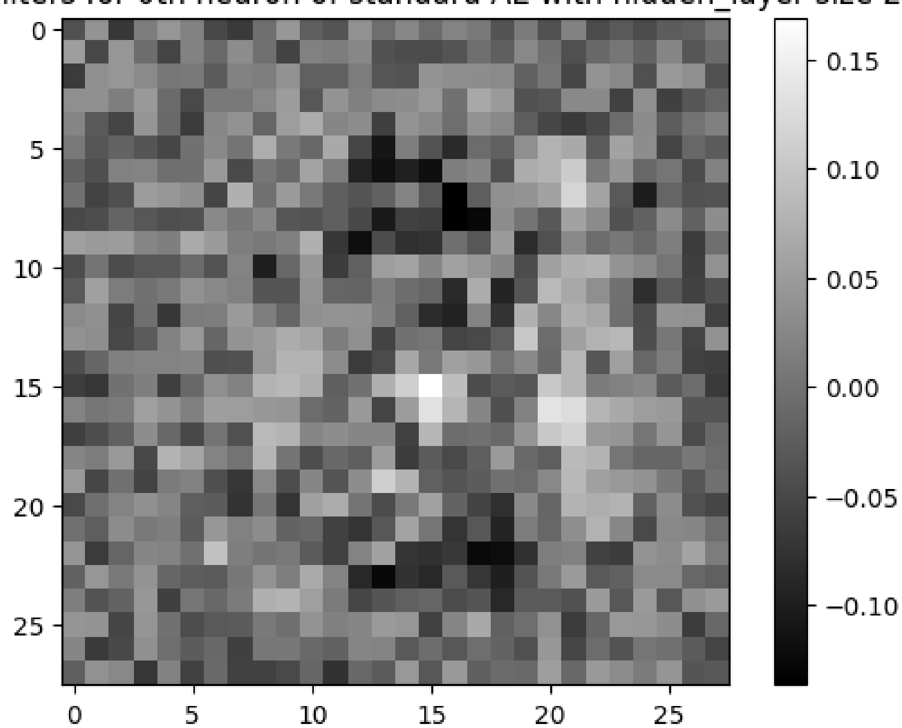


[37]:
```
encoder_decoder_filters_plots(model_Q2,"standard AE with hidden_layer size␣
↪256",device)
```

Encoder Filters for 0th neuron of standard AE with hidden_layer size 256

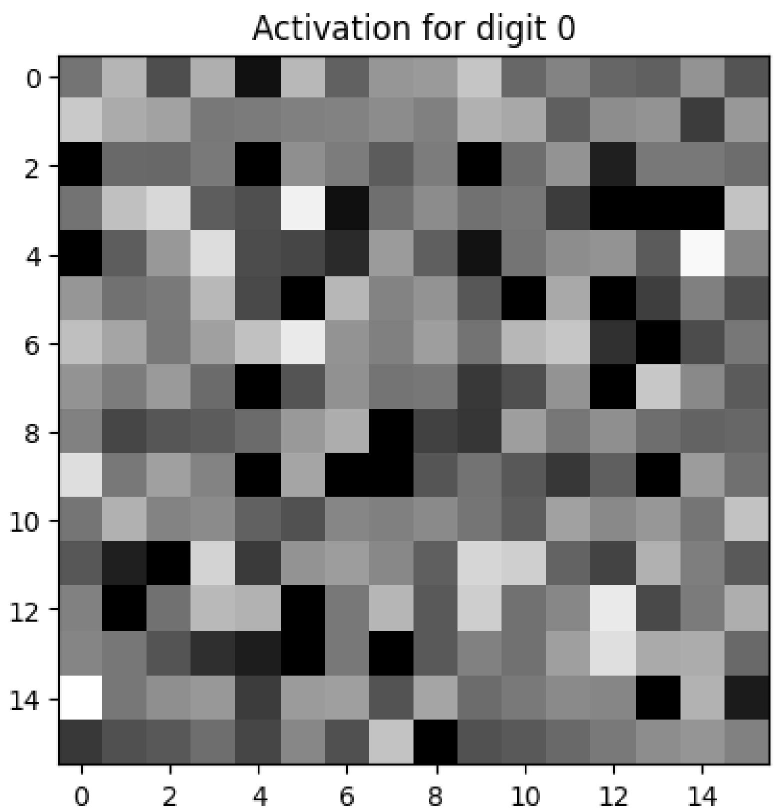


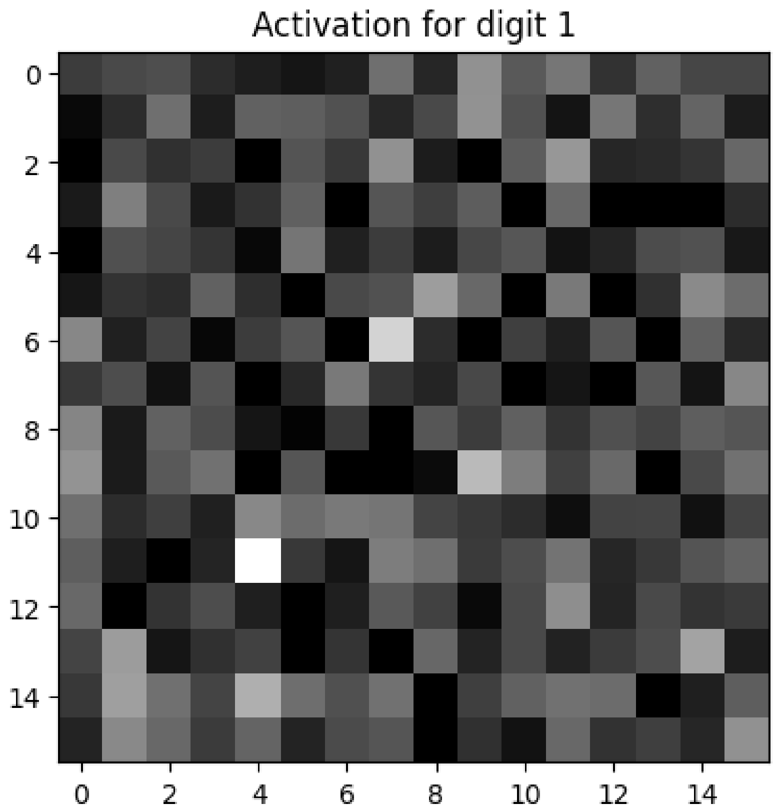
Decoder Filters for 0th neuron of standard AE with hidden_layer size 256

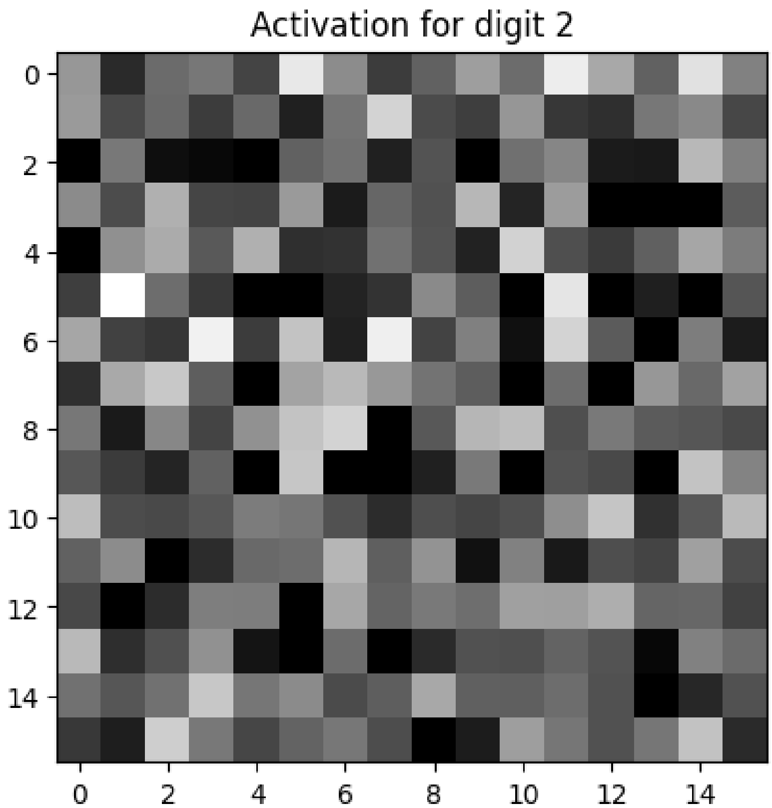


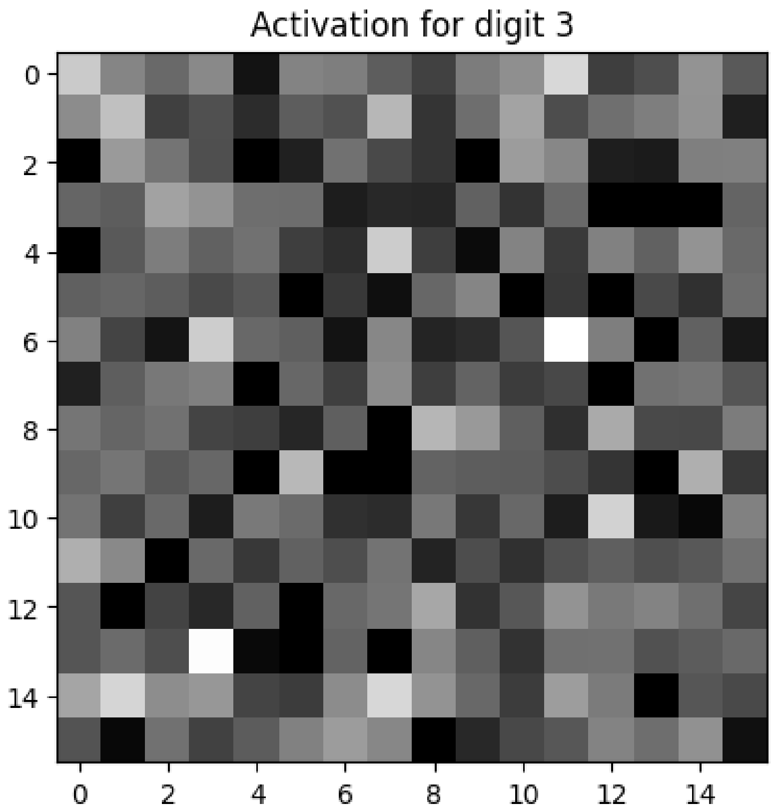
[38]:

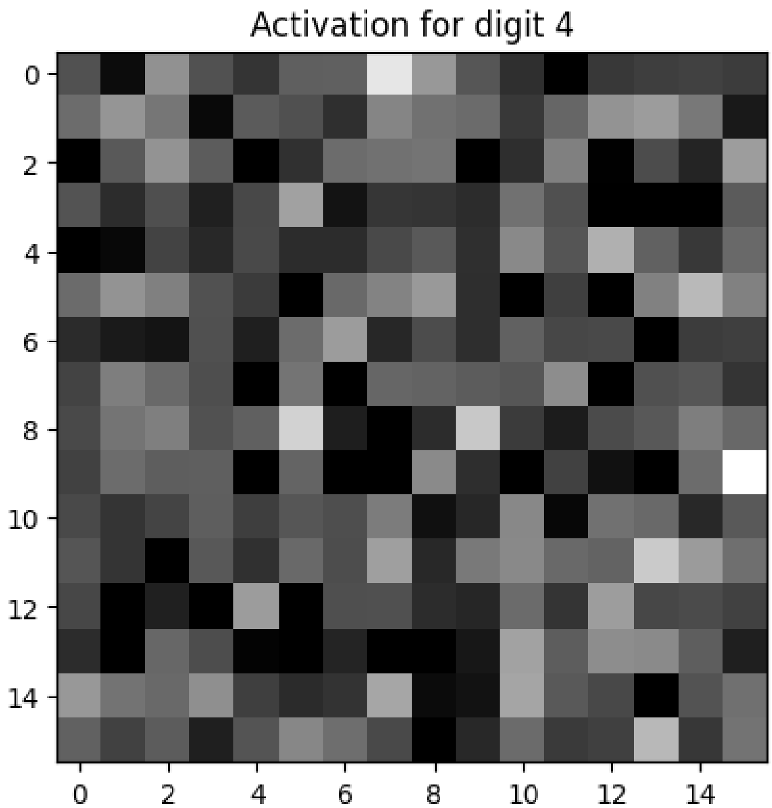
```
visualize_activations(model_Q2,test_loader,"Standard AE with␣
↪hidden_layer=256",device,256)
```

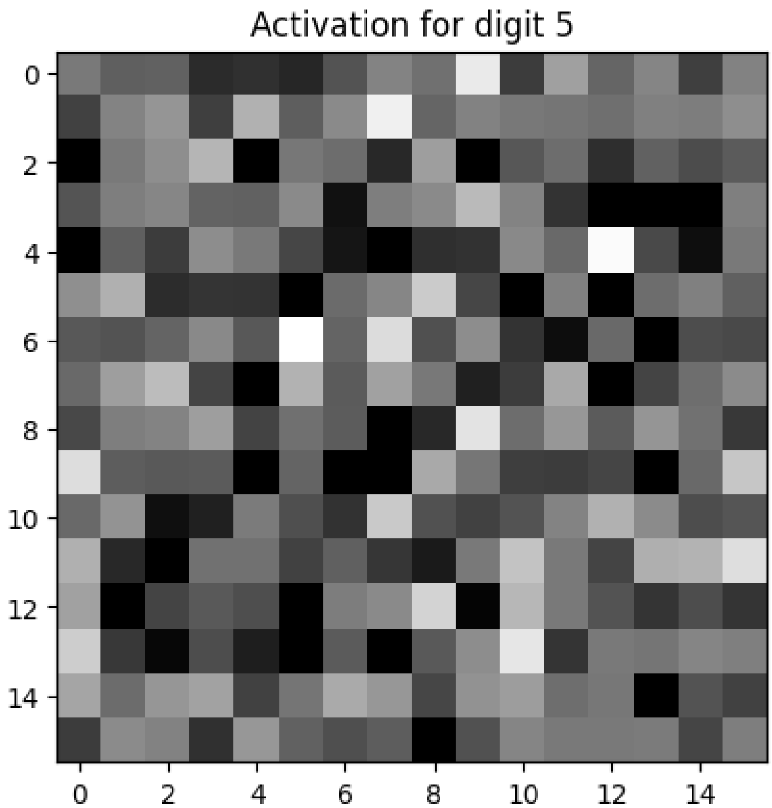


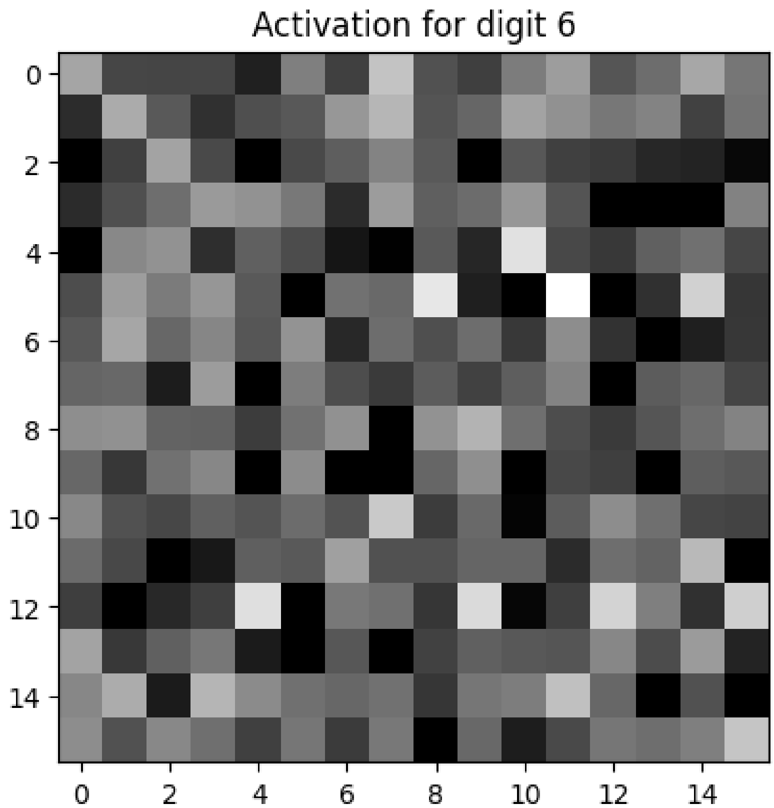

Activation for digit 1
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14

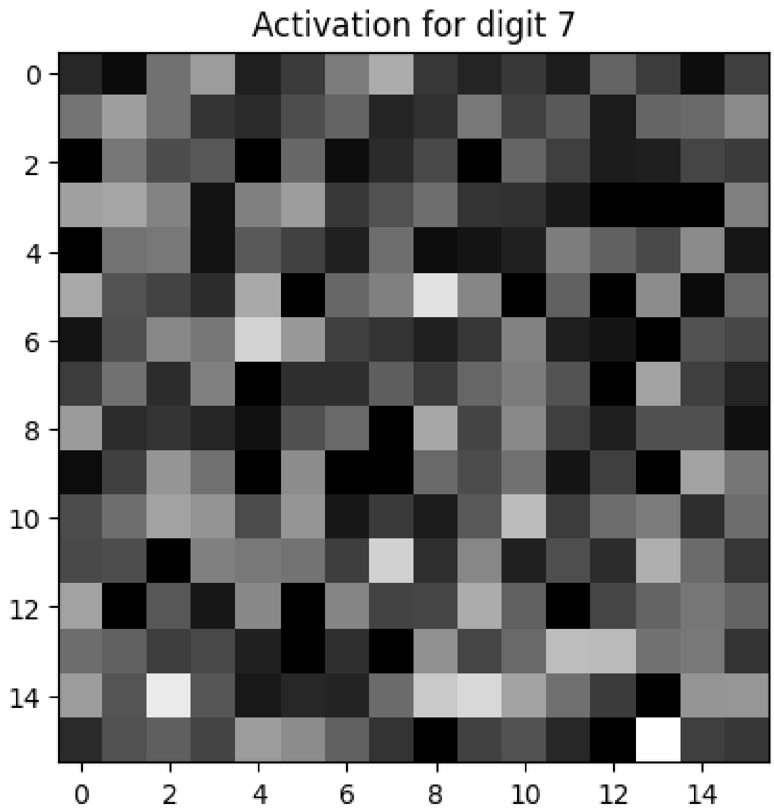

Activation for digit 2
0
2
4
6
8
10
12
14

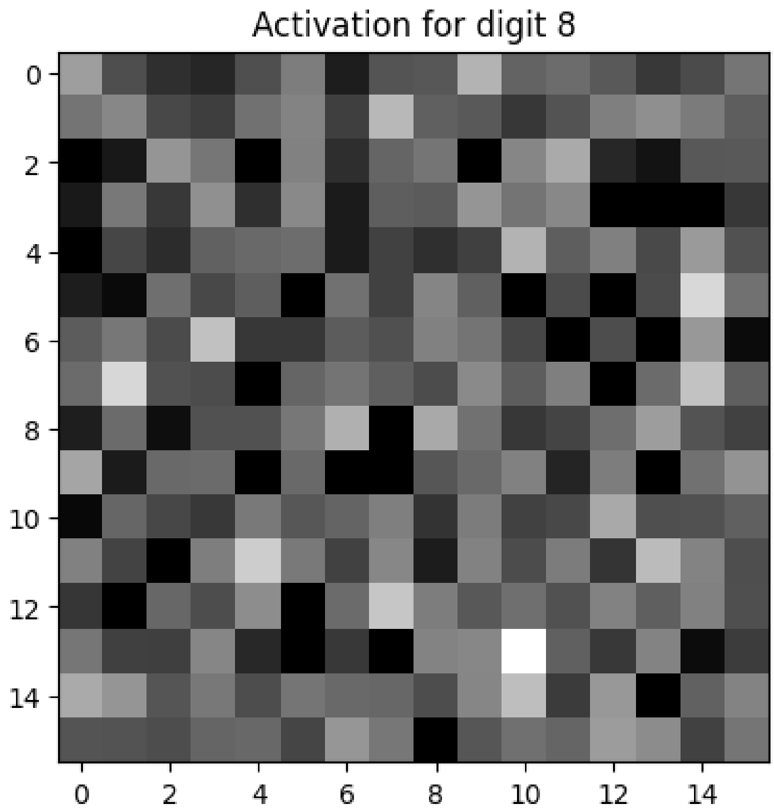

Activation for digit 3
0
2
4
6
8
10
12
14

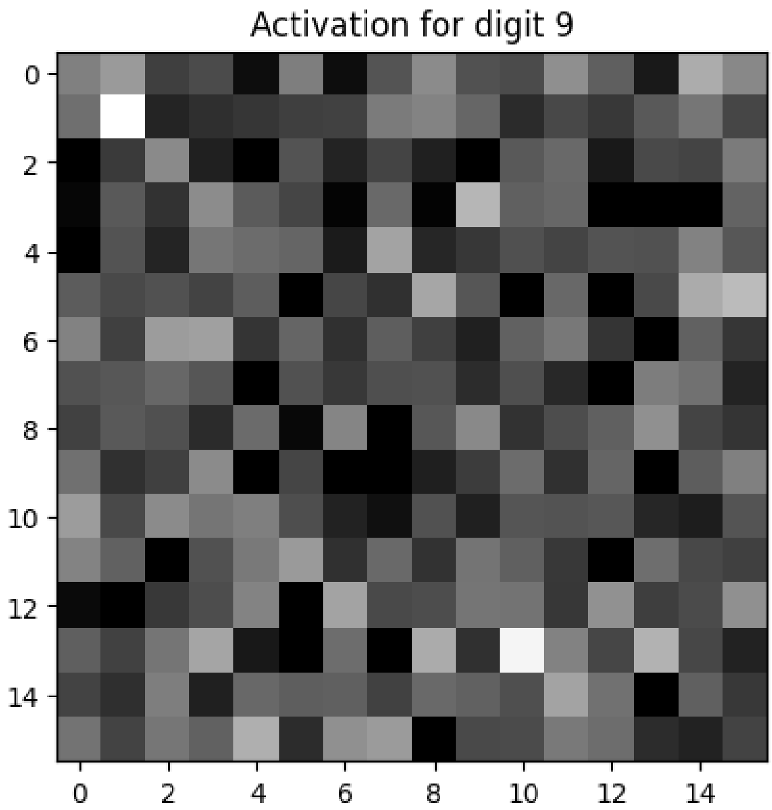

Activation for digit 4
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14


Activation for digit 5
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14


Activation for digit 6
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14


Activation for digit 7
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14


Activation for digit 8
0
2
4
6
8
10
12
14
0
2
4
6
8
10
12
14

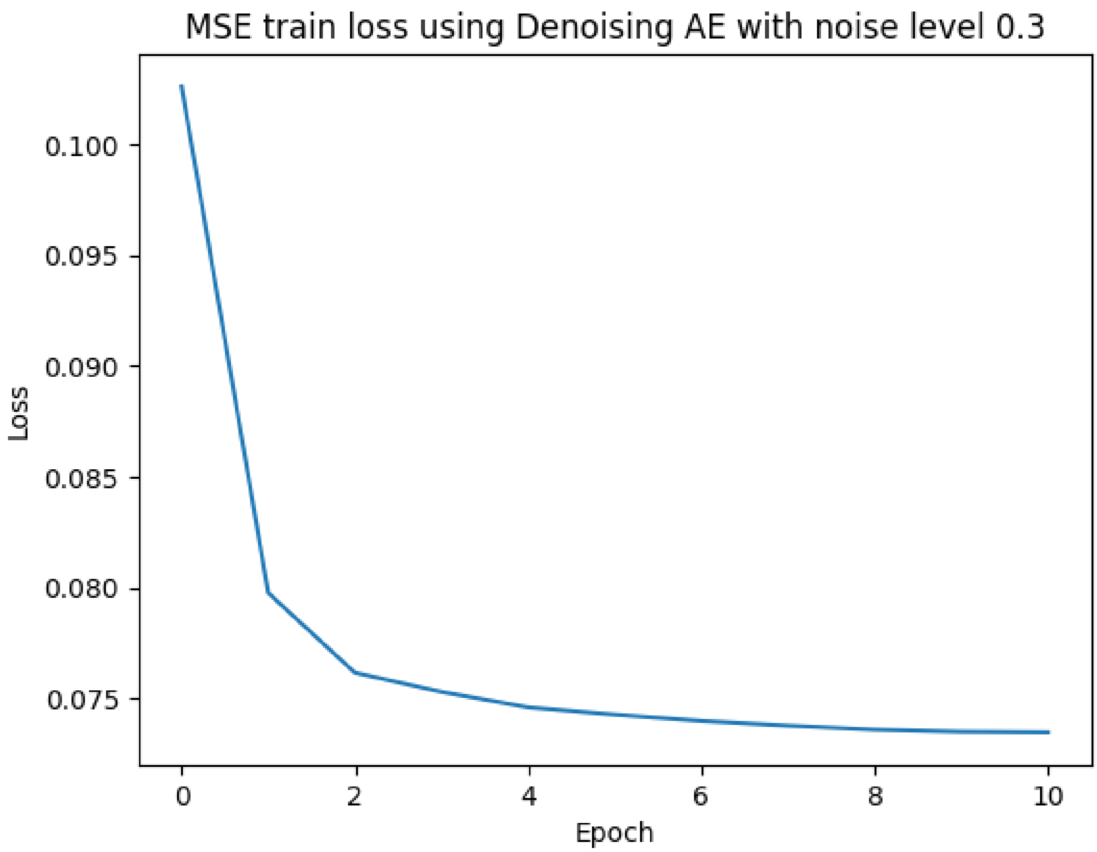


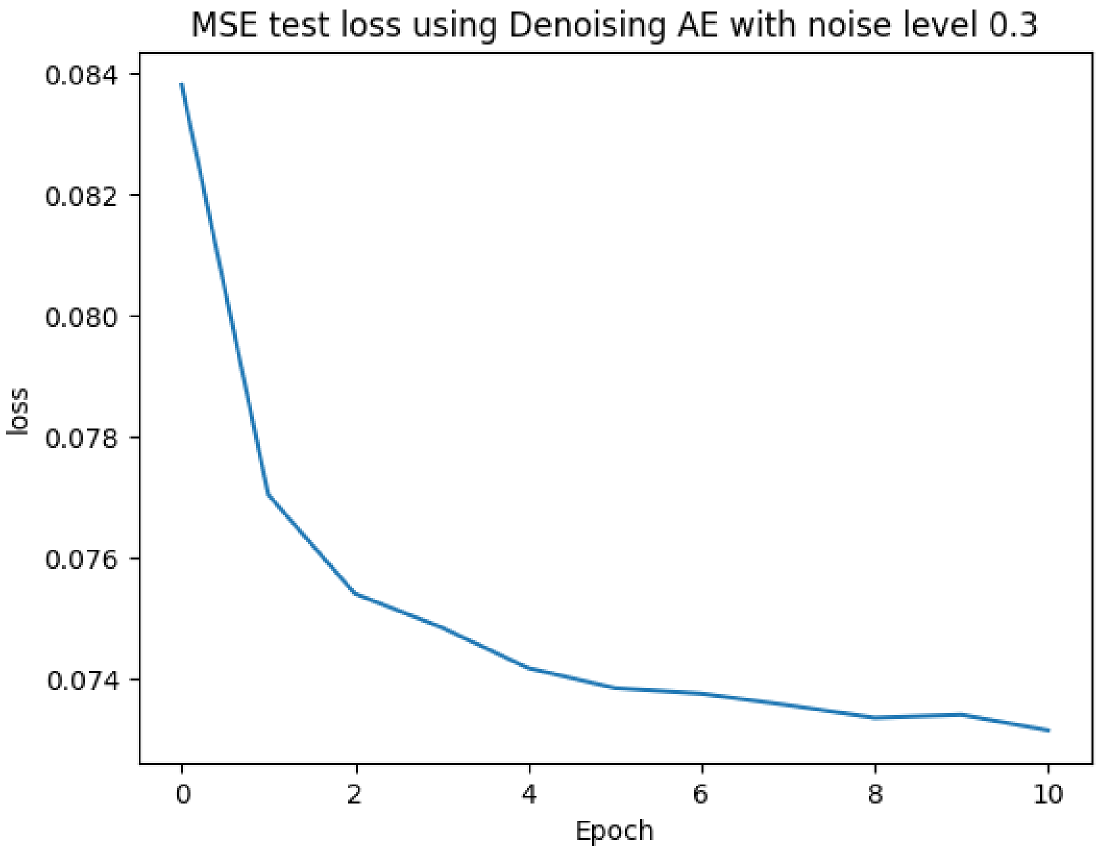
0.4 Q4

[39]:
```
def add_noise(img, noise_val):
    noise = torch.randn(img.size())*noise_val
    noisy_img = img + noise
    return noisy_img
```

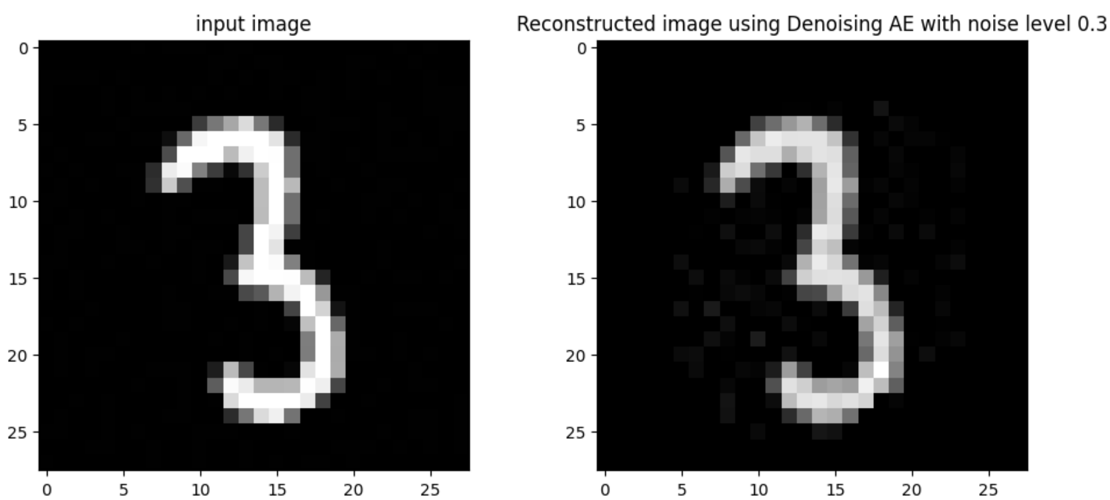
[41]:
```
noise_vals = [0.3,0.5,0.8,0.9]
for noise_val in noise_vals:
    model_Q4 = AE_Q2(256).to(device)
    optimizer = torch.optim.Adam(model_Q4.parameters(), lr=learning_rate)
    train_losses_AE_Q4 , test_losses_AE_Q4 =␣
 ↪train_test(model_Q4,device,train_loader,test_loader,optimizer,lossfn,denoise=True,noise_val
    plot_losses(train_losses_AE_Q4, test_losses_AE_Q4, model_name = "Denoising␣
 ↪AE with noise level "+str(noise_val))
    #MSE recomnstruction error for vanilla AE
    mse_error = test(model_Q4,device,test_loader,lossfn)
    print(" MSE for the ",noise_val,"=", mse_error.item())
    index = random.randint(0,9999)
    test_image = test_loader.dataset.data[index, :, :].clone()
```

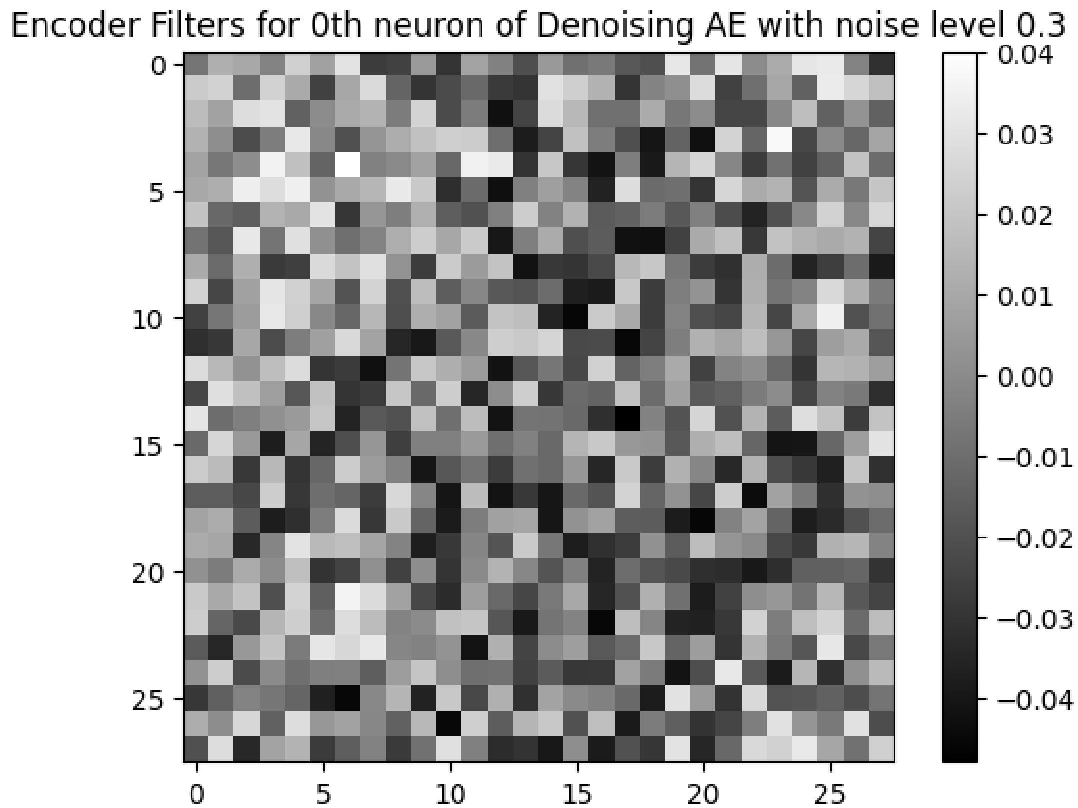
```
    test_image = add_noise(test_image,noise_val)
    plot_reconstructed_image(model_Q4,device,test_image, model_name="Denoising␣
↪AE with noise level "+str(noise_val))
    encoder_decoder_filters_plots(model_Q3,"Denoising AE with noise level␣
↪"+str(noise_val),device)
```

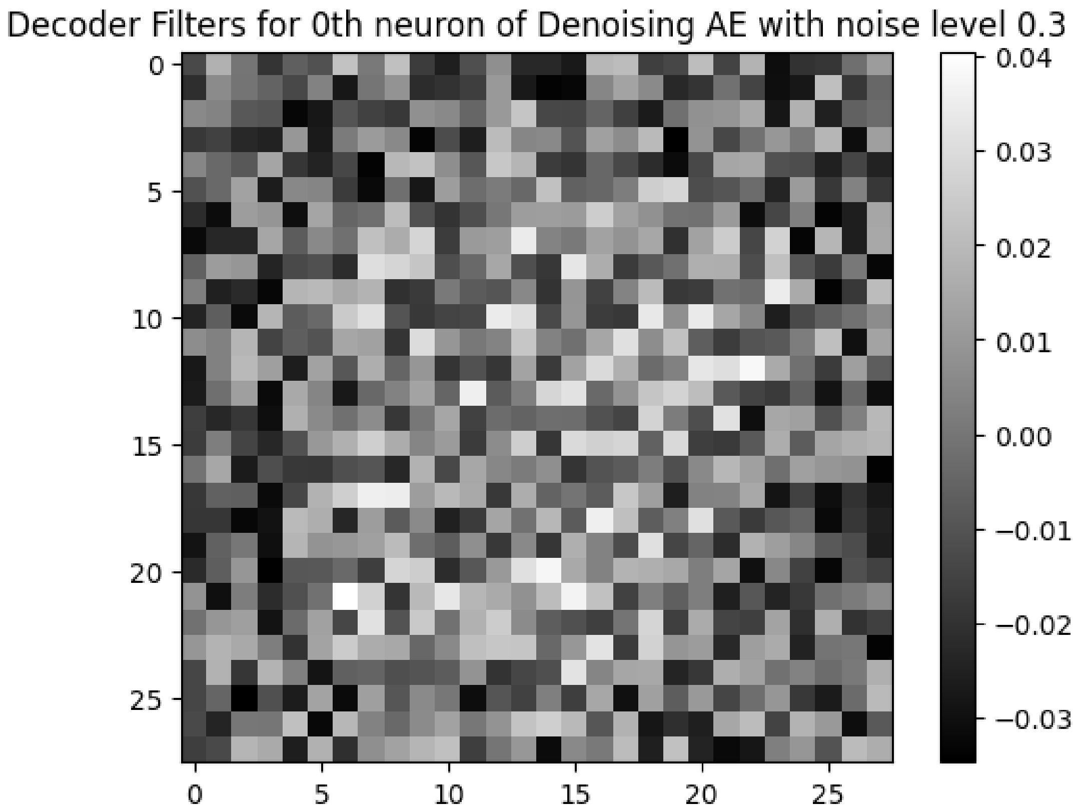


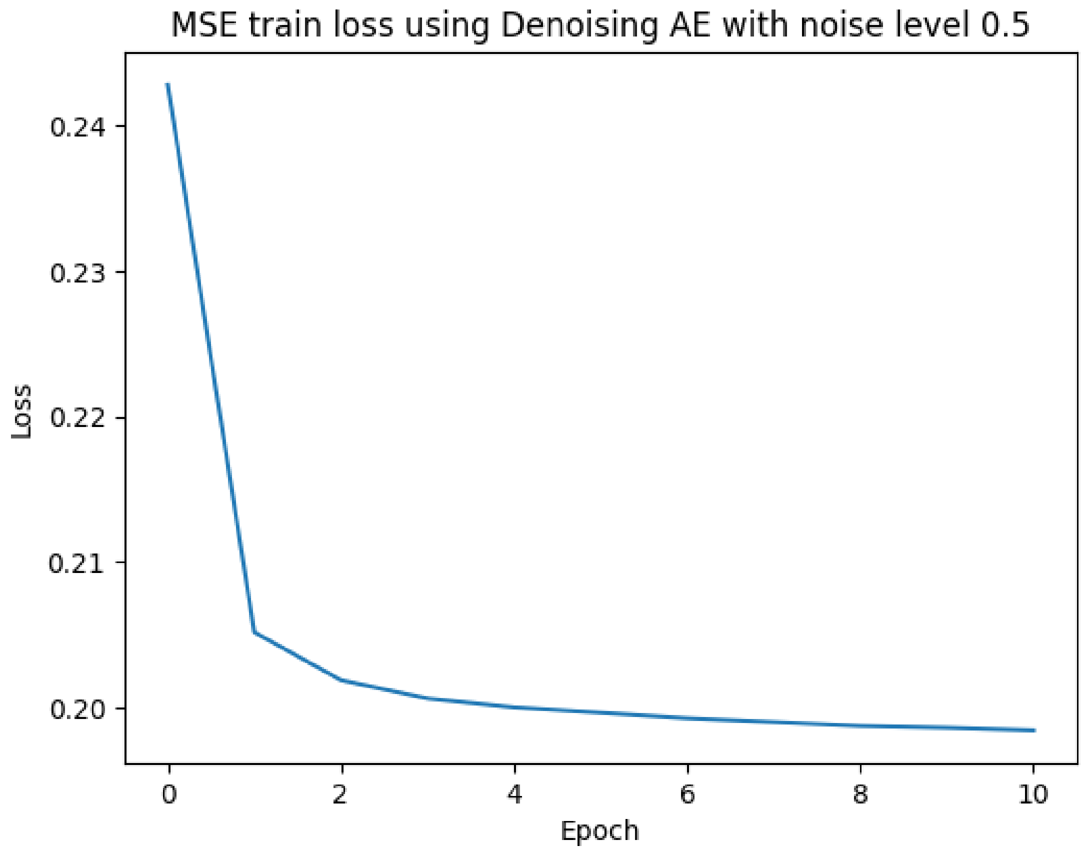


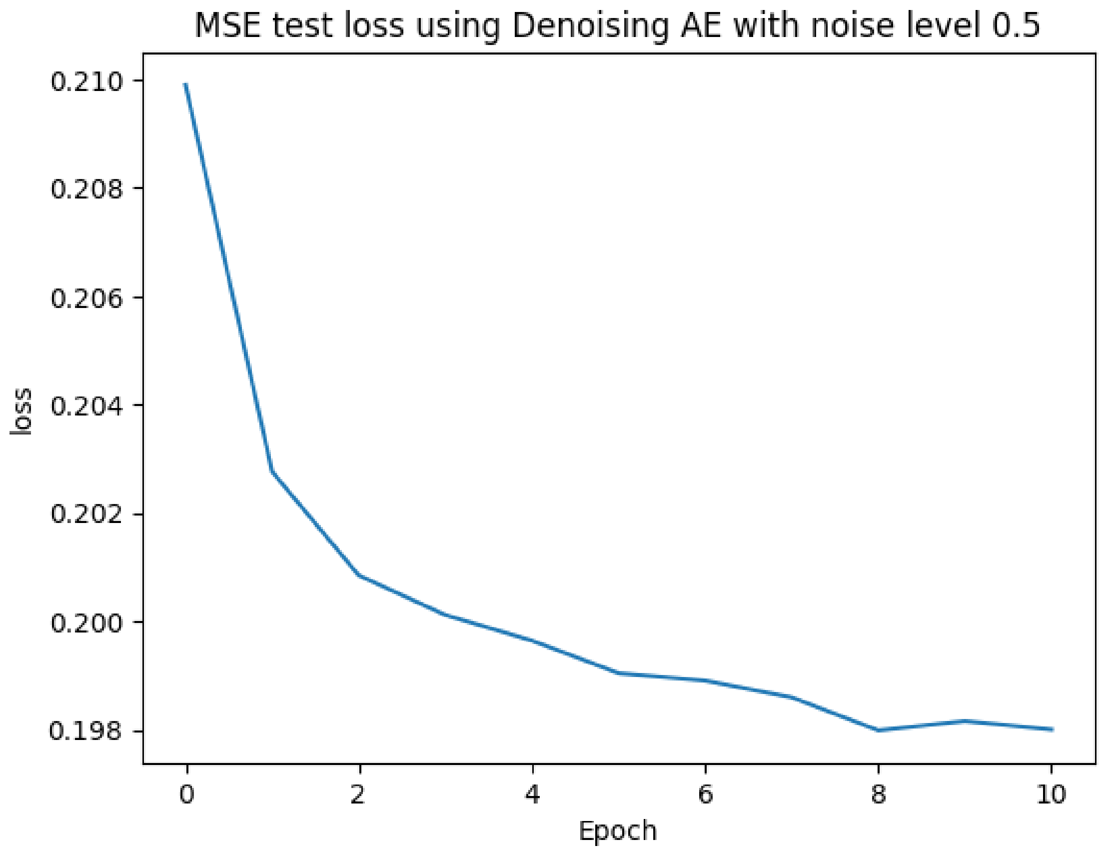

```
MSE for the  0.3 = 0.0033181174658238888
```



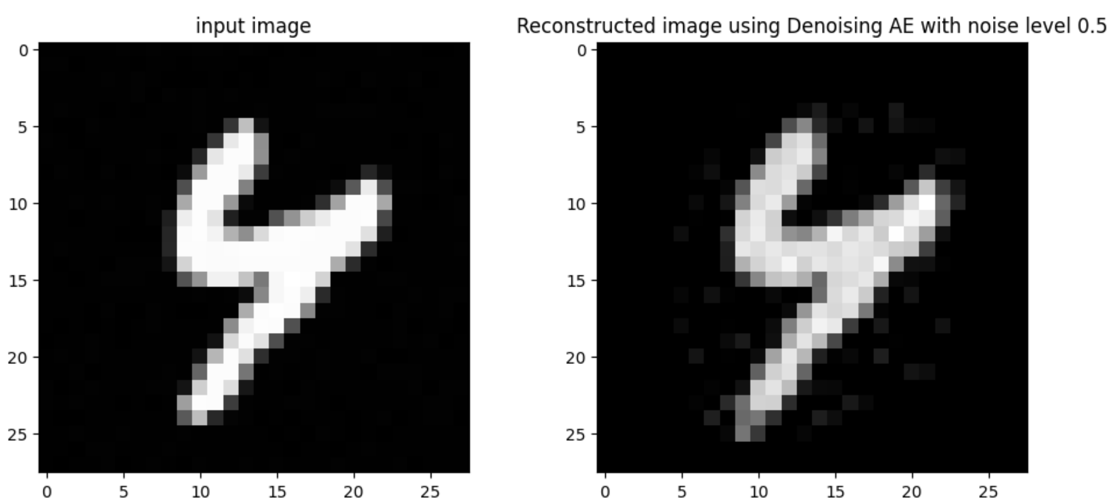

Encoder Filters for 0th neuron of Denoising AE with noise level 0.3
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
0
5
10
15
20
25

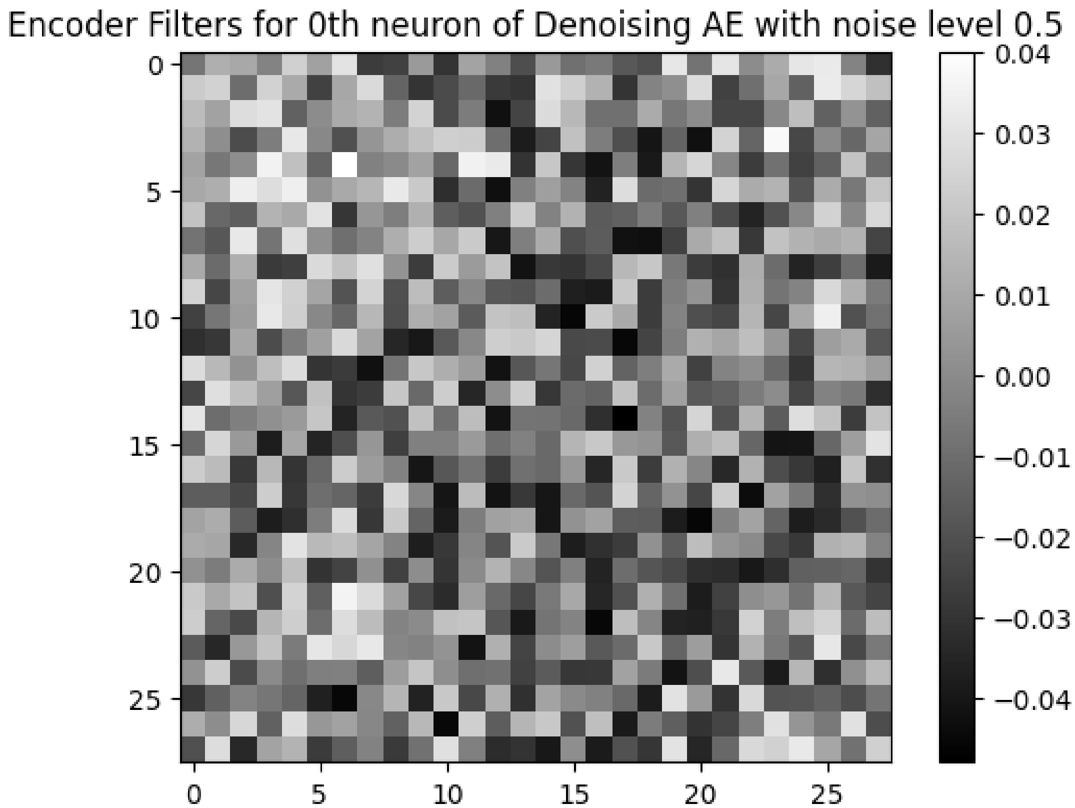

Decoder Filters for 0th neuron of Denoising AE with noise level 0.3
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03

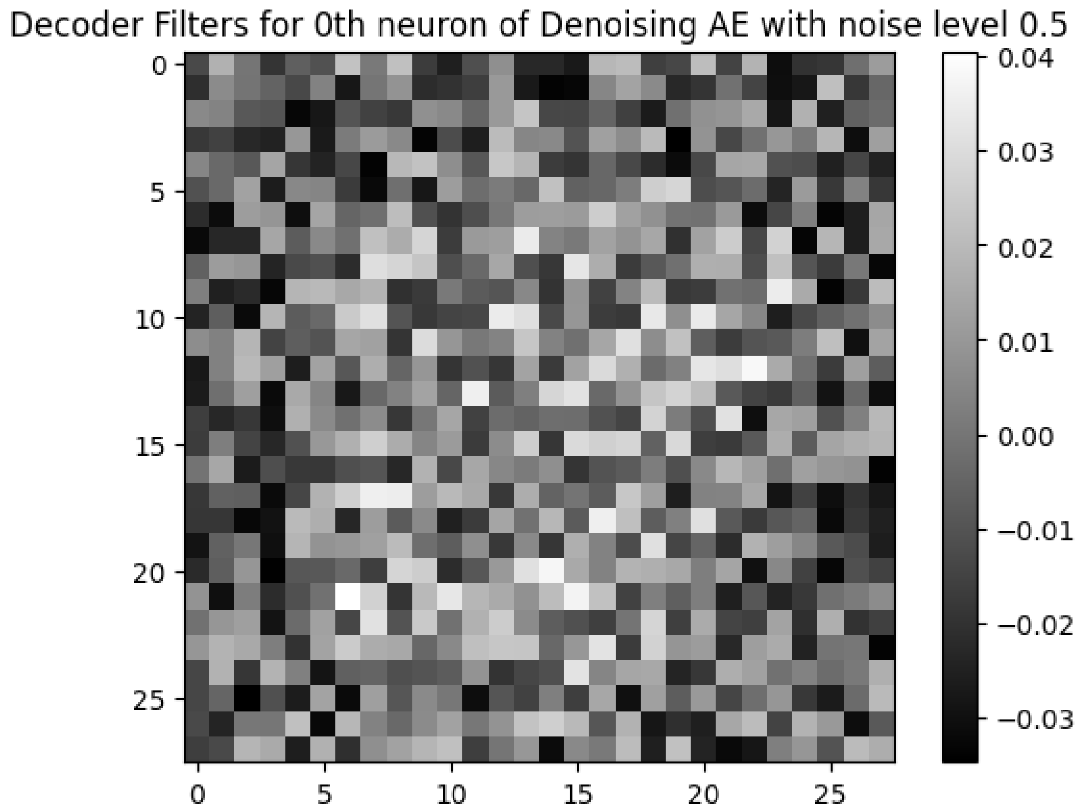

MSE train loss using Denoising AE with noise level 0.5
Loss
0.24
0.23
0.22
0.21
0.20
0
2
4
6
8
10
Epoch

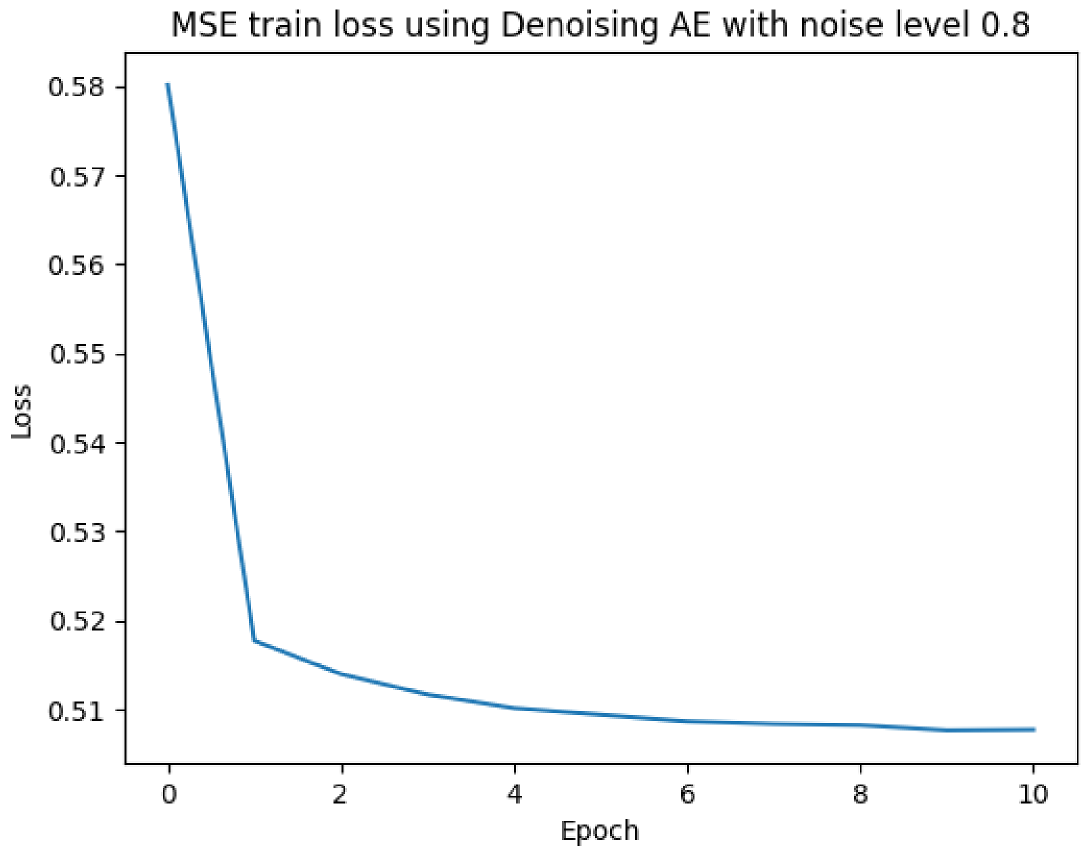


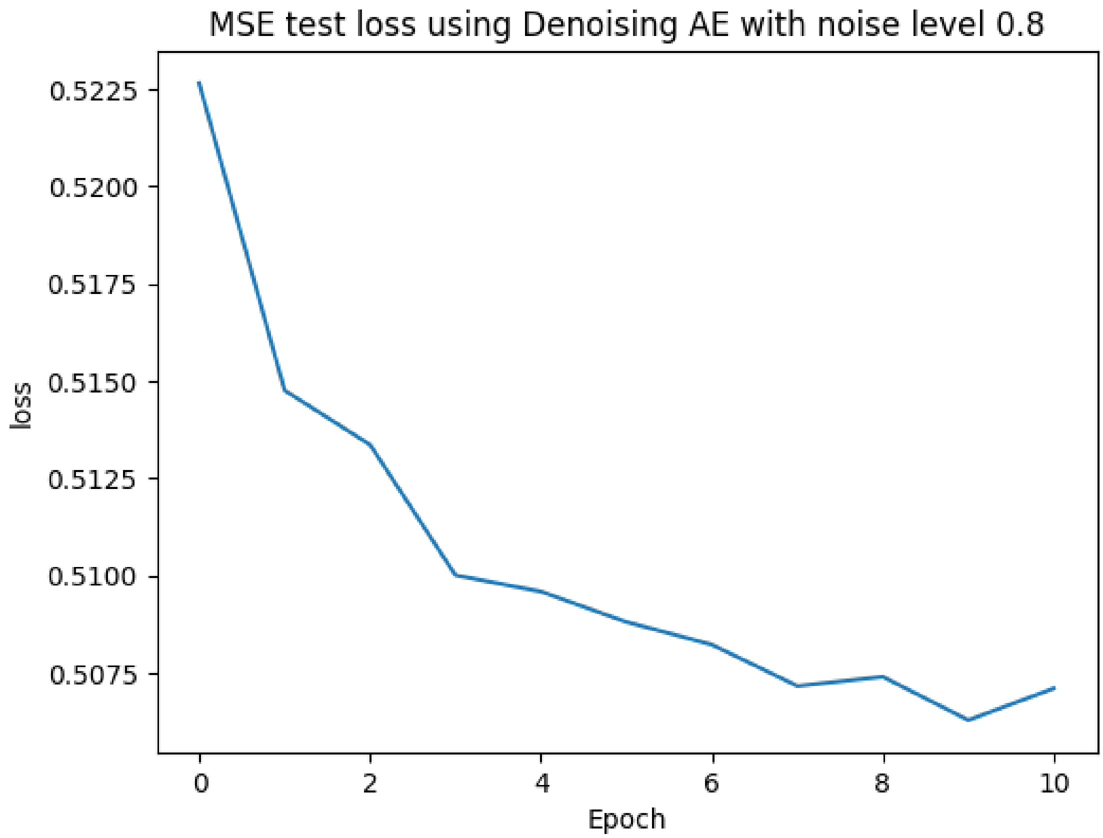

```
MSE for the  0.5 = 0.0038982063997536898
```



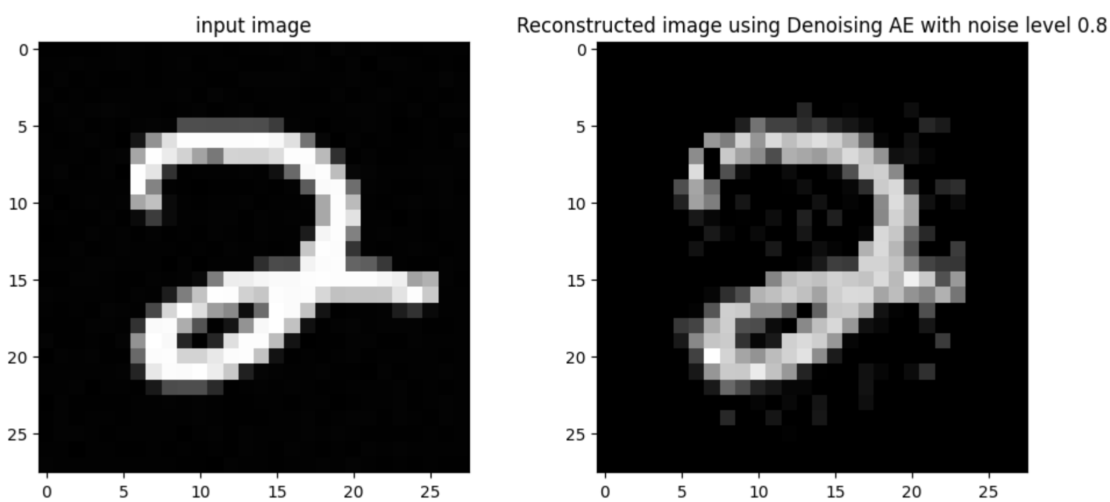

Encoder Filters for 0th neuron of Denoising AE with noise level 0.5
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04

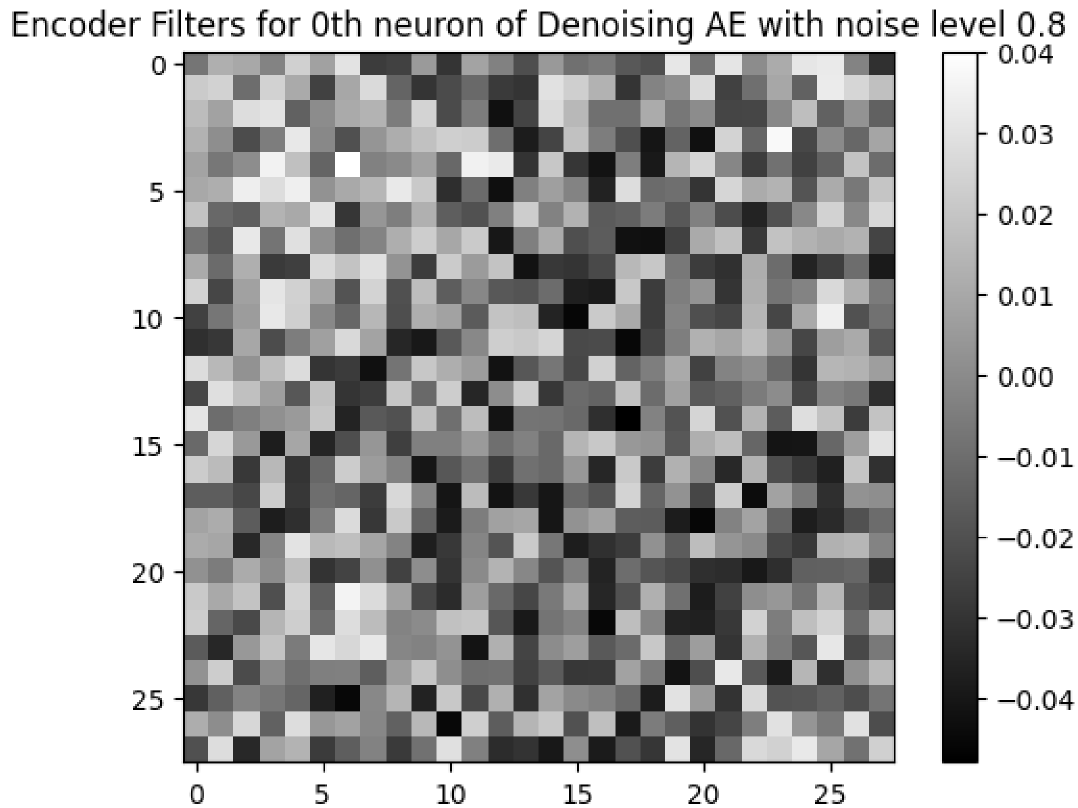

Decoder Filters for 0th neuron of Denoising AE with noise level 0.5
0
5
10
15
20
25
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03

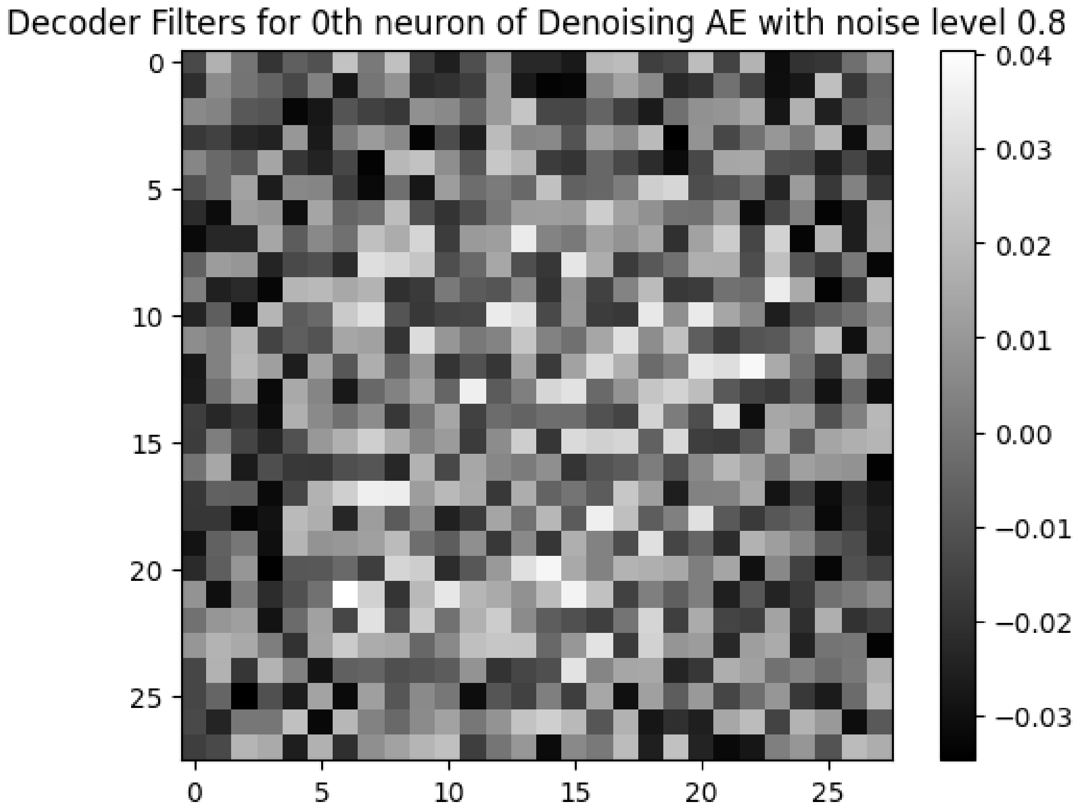

MSE train loss using Denoising AE with noise level 0.8
0.58
0.57
0.56
0.55
0.54
0.53
0.52
0.51
Loss
0
2
4
6
8
10
Epoch

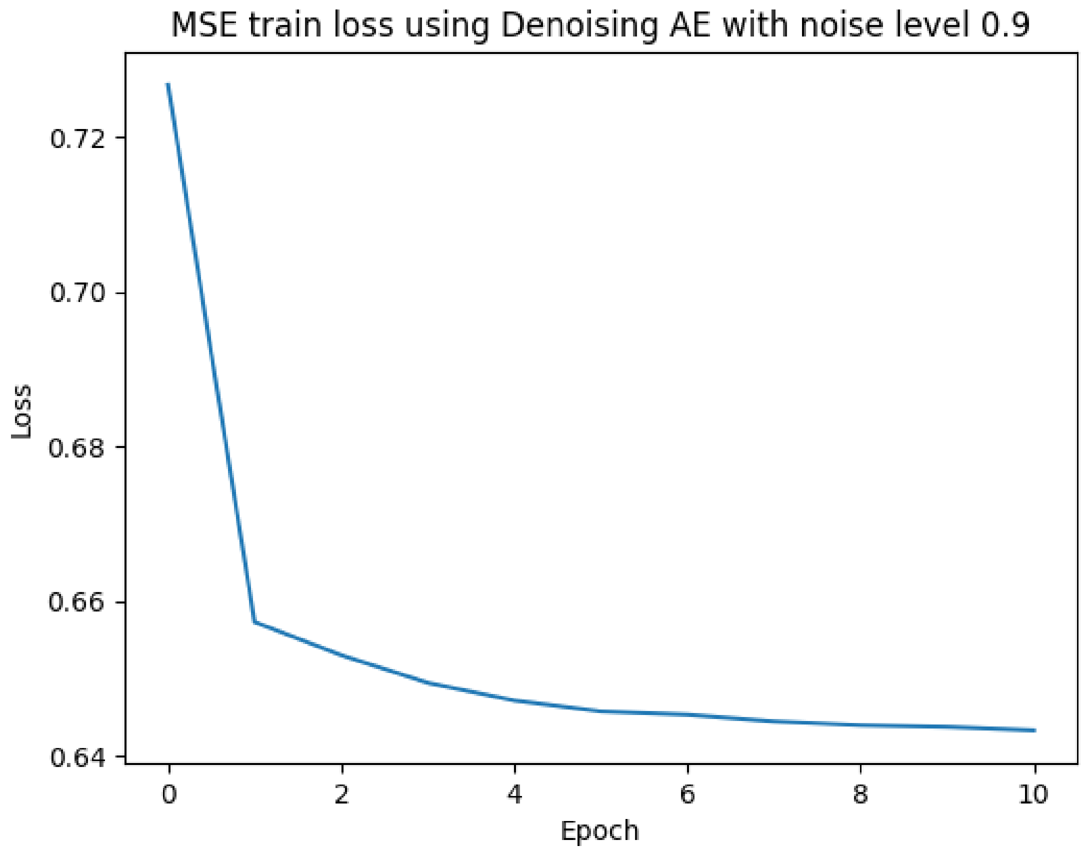


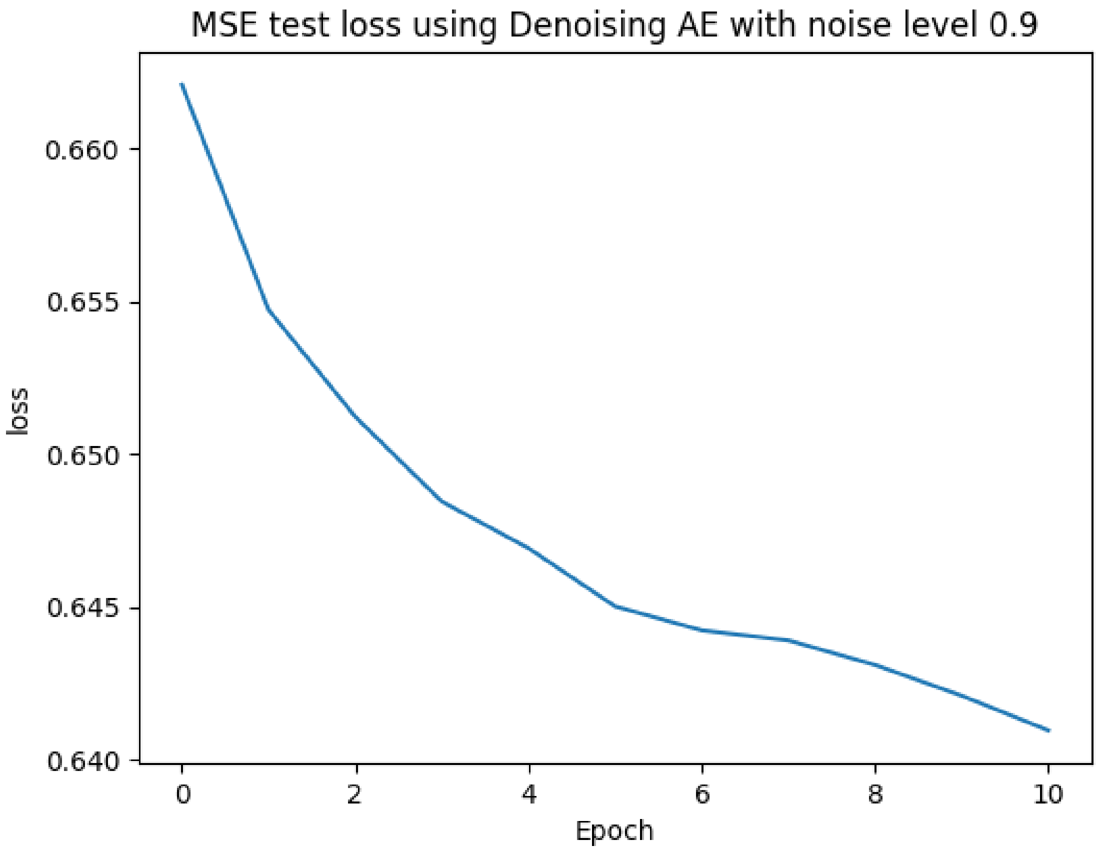

```
MSE for the  0.8 = 0.006269432138651609
```



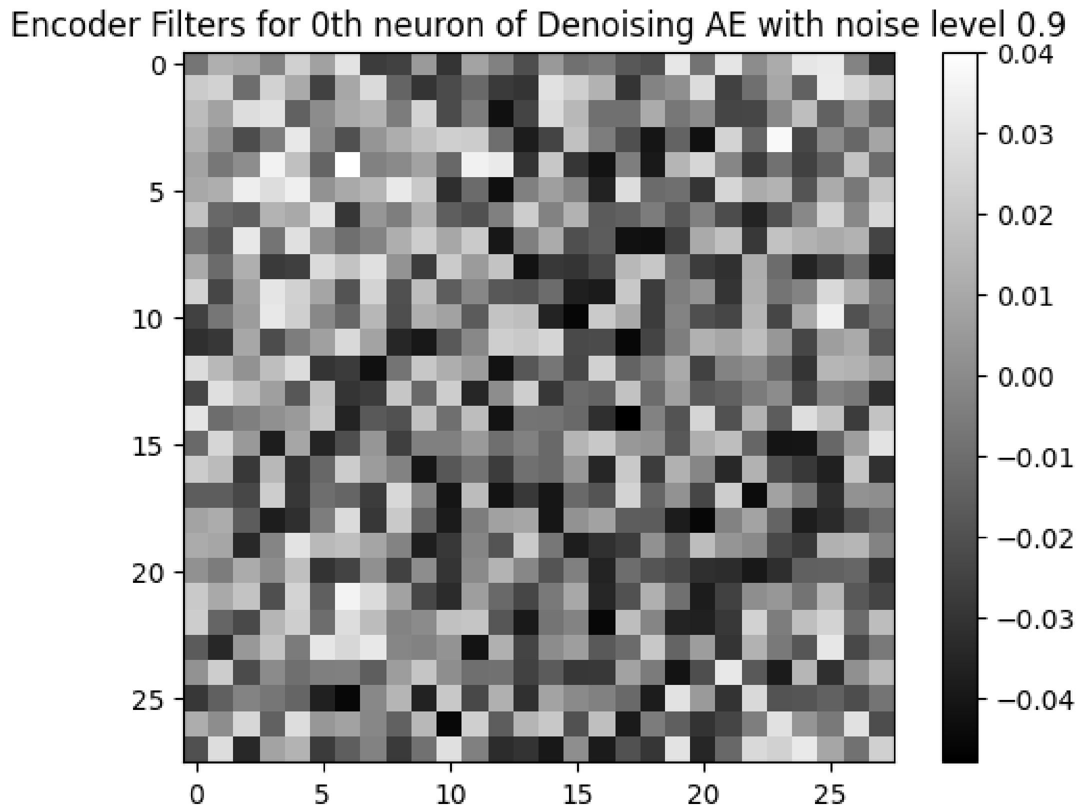

Encoder Filters for 0th neuron of Denoising AE with noise level 0.8
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04

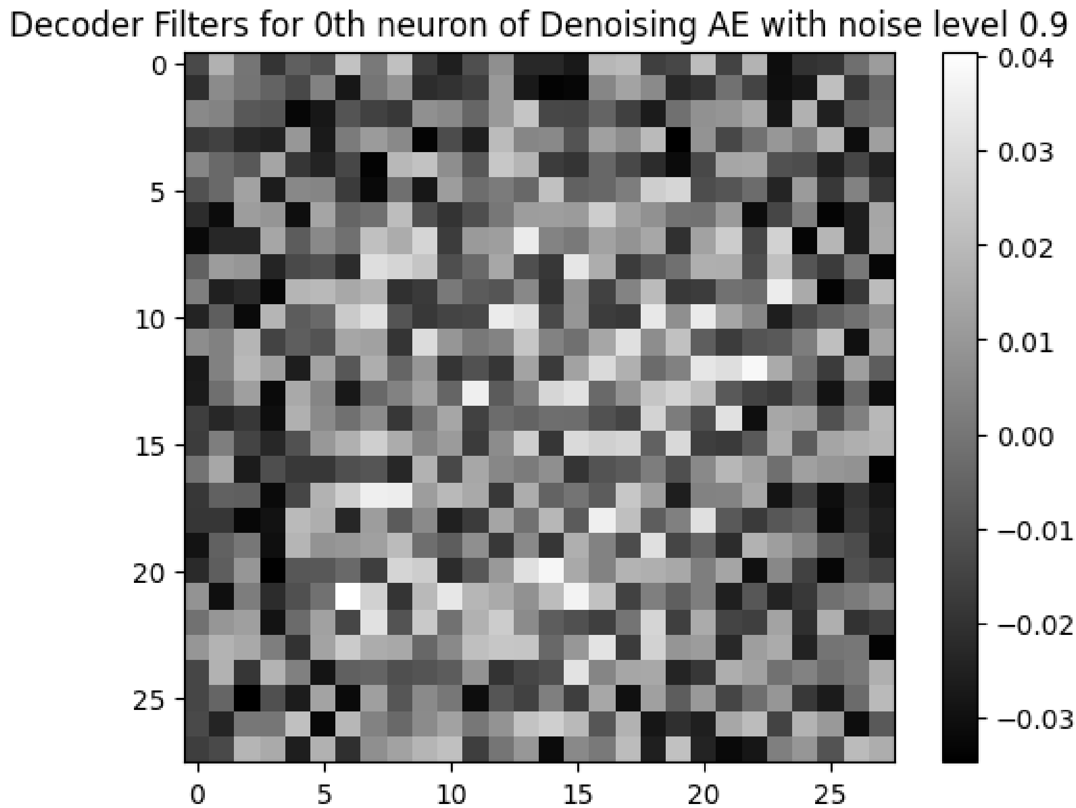

Decoder Filters for 0th neuron of Denoising AE with noise level 0.8
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03


MSE train loss using Denoising AE with noise level 0.9
Loss
0.72
0.70
0.68
0.66
0.64
0
2
4
6
8
10
Epoch




```
MSE for the  0.9 = 0.006520132068544626
```


Encoder Filters for 0th neuron of Denoising AE with noise level 0.9
0
5
10
15
20
25
0
5
10
15
20
25
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04




```
[ ]: # Convolutional Autoencoders
```

0.5 Q5

```
[42]: class conv_AE_with_unpooling(nn.Module): #define unpooling outside the decoder␣
      ↪and separately in forward nn.Sequential just takes one input

          def __init__(self): #class constructor
              super(conv_AE_with_unpooling,self).__init__() #calls the parent␣
      ↪constructor

              #initializing the encoder module
              self.encoder_conv1 = nn.Sequential(
                  nn.Conv2d(1,8, kernel_size = 3, stride = 1,padding= 1),
                  nn.ReLU(),
                  nn.MaxPool2d(kernel_size = (2,2),return_indices = True)
              ) # 28x28x1 to 14x14x8
              self.encoder_conv2 = nn.Sequential(
                  nn.Conv2d(8,16, kernel_size = 3, stride = 1,padding= 1),
                  nn.ReLU(),
```

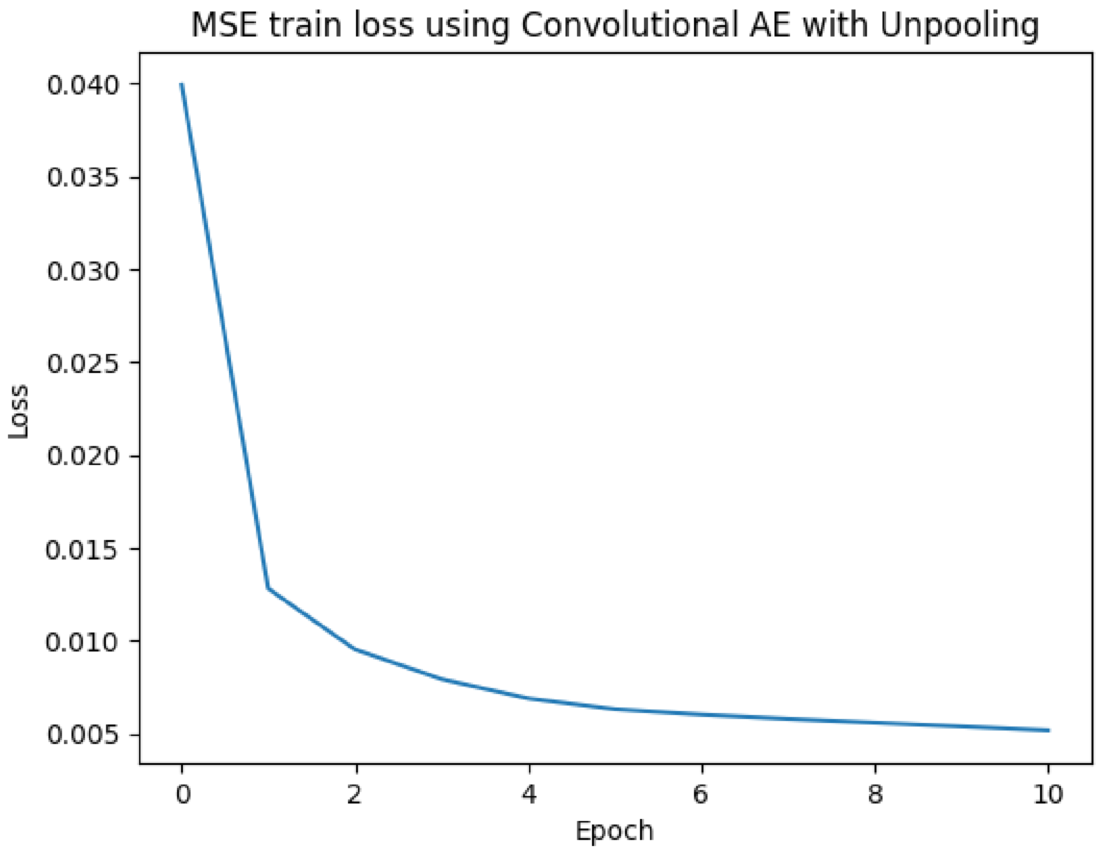
```
            nn.MaxPool2d(kernel_size = (2,2),return_indices = True)
        ) #14x14x8 to 7x7x16
        self.encoder_conv3 = nn.Sequential(
            nn.Conv2d(16,16, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = (2,2),return_indices = True)
        ) #7x7x16 to 3x3x16

        #initializing the decoder module
        self.decoder_conv1 = nn.Sequential(nn.Identity()) #7x7x16 to 7x7x16
        self.decoder_conv2 = nn.Sequential(
            nn.Conv2d(16,8, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU()
        ) #14x14x16 to 14x14x8
        self.decoder_conv3 = nn.Sequential(
            nn.Conv2d(8,1, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU()
        ) #28x28x8 to 28x28x1

        #defining the unpooling operation
        self.unpool = nn.MaxUnpool2d(kernel_size = (2,2))


    def forward(self,x): #defines the forward pass and also the structure of␣
↪the network thus helping backprop

        encoded_input,indices1  = self.encoder_conv1(x.float())  # 28x28x1 to␣
↪14x14x8
        encoded_input,indices2  = self.encoder_conv2(encoded_input) #14x14x8 to␣
↪7x7x16
        encoded_input,indices3  = self.encoder_conv3(encoded_input) #7x7x16 to␣
↪3x3x16

        reconstructed_input     = self.
↪unpool(encoded_input,indices3,output_size=torch.Size([batch_size, 16, 7,␣
↪7])) #3x3x16 to 7x7x16
        reconstructed_input     = self.decoder_conv1(reconstructed_input)␣
↪#7x7x16 to 7x7x16
        reconstructed_input     = self.unpool(reconstructed_input,indices2)␣
↪#7x7x16 to 14x14x16
        reconstructed_input     = self.
↪decoder_conv2(reconstructed_input)#14x14x16 to 14x14x8
        reconstructed_input     = self.
↪unpool(reconstructed_input,indices1)#14x14x8 to 28x28x8
        reconstructed_input     = self.
↪decoder_conv3(reconstructed_input)#28x28x8 to 28x28x1
```

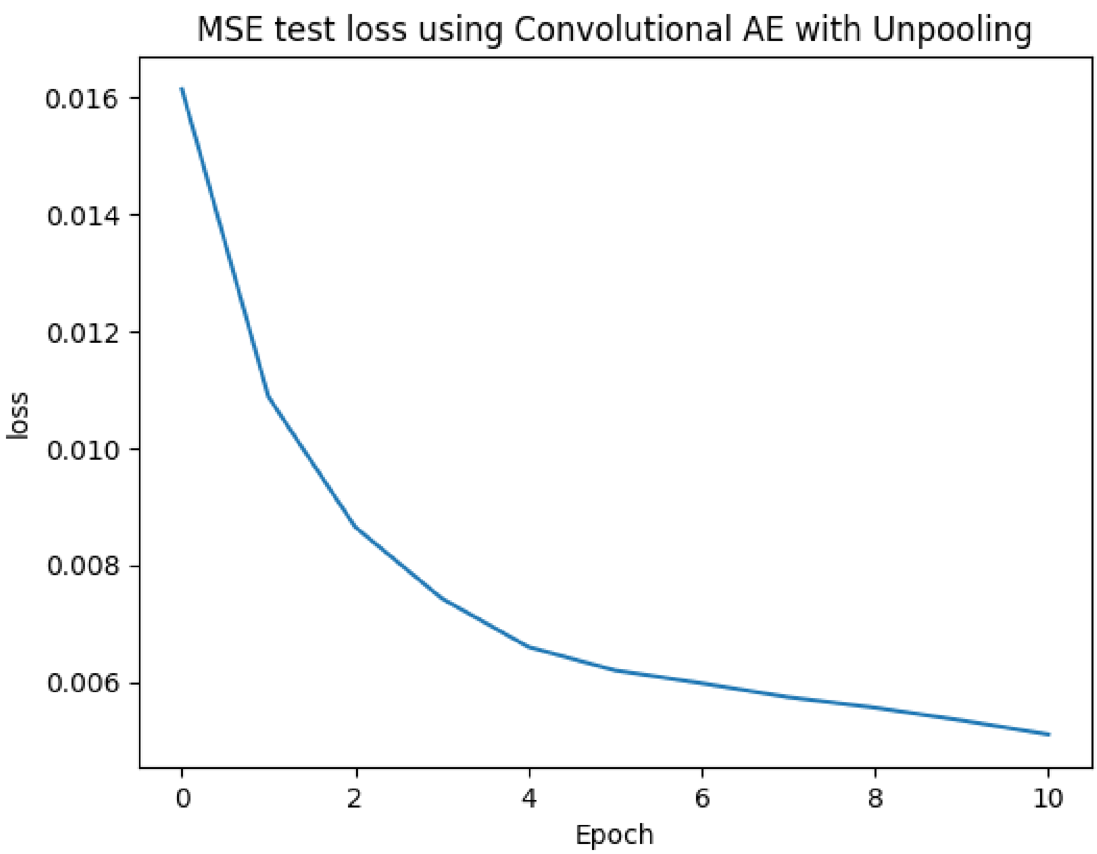
```python

        return reconstructed_input,encoded_input
```

[43]:
```python
from torchsummary import summary
print(conv_AE_with_unpooling().to(device))
```

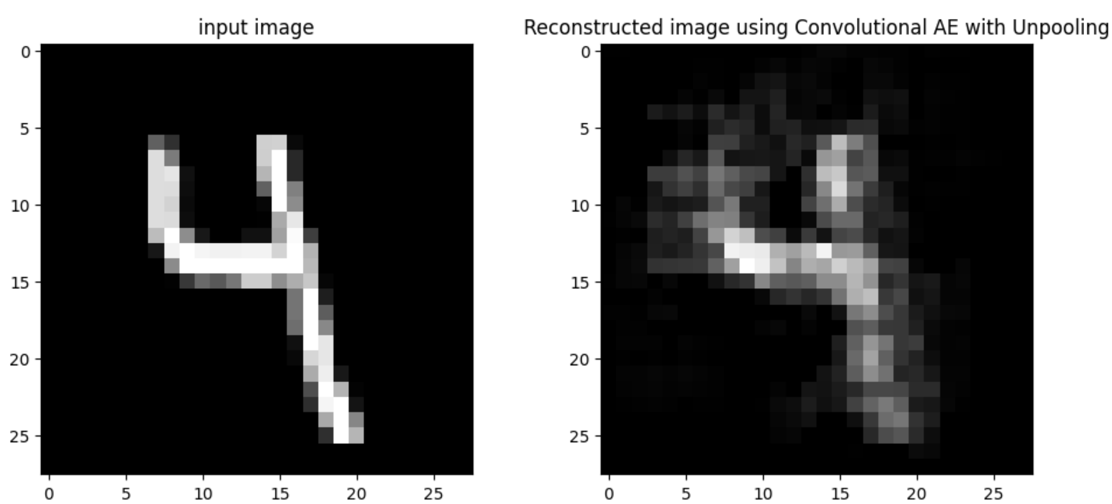
```
conv_AE_with_unpooling(
  (encoder_conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_conv3): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (decoder_conv1): Sequential(
    (0): Identity()
  )
  (decoder_conv2): Sequential(
    (0): Conv2d(16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (decoder_conv3): Sequential(
    (0): Conv2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
  )
  (unpool): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
)
```

[45]:
```python
model_Q5_a = conv_AE_with_unpooling().to(device)
optimizer = torch.optim.Adam(model_Q5_a.parameters(), lr=learning_rate)
train_losses_AE_Q5_a , test_losses_AE_Q5_a =␣
 ↪train_test(model_Q5_a,device,train_loader,test_loader,optimizer,lossfn,q5_flag=True)
plot_losses(train_losses_AE_Q5_a, test_losses_AE_Q5_a, model_name =␣
 ↪"Convolutional AE with Unpooling")
```


MSE train loss using Convolutional AE with Unpooling
Loss
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0
2
4
6
8
10
Epoch



[46]:
```
index = random.randint(0,9999)
test_image = test_loader.dataset.data[index, :, :].clone()
plot_reconstructed_image(model_Q5_a,device,test_image,␣
↪model_name="Convolutional AE with Unpooling",q5_flag=True)
```

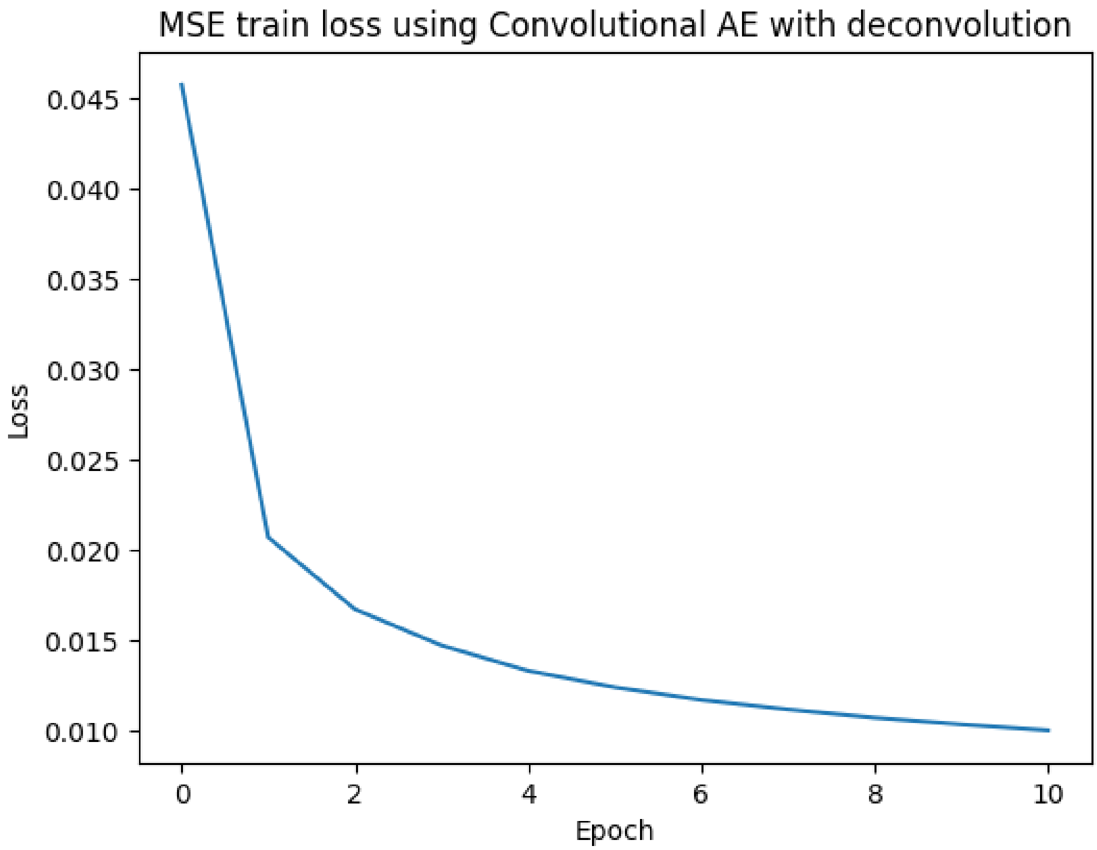


[47]:
```python
class conv_AE_with_deconv(nn.Module):
    def __init__(self):
        super(conv_AE_with_deconv,self).__init__()

        #encoder
        self.encoder_conv1 = nn.Sequential(
            nn.Conv2d(1,8, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = (2,2))
        )
        self.encoder_conv2 = nn.Sequential(
            nn.Conv2d(8,16, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = (2,2))
        )
        self.encoder_conv3 = nn.Sequential(
            nn.Conv2d(16,16, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = (2,2))
        )

        #decoder module
        self.decoder_conv1 = nn.Sequential(
            nn.ConvTranspose2d(16,16, kernel_size = 3, stride = 2),
            nn.ReLU()
        )
        self.decoder_conv2 = nn.Sequential(
            nn.ConvTranspose2d(16,8, kernel_size = 4, stride = 2, padding = 1),
            nn.ReLU()
        )
        self.decoder_conv3 = nn.Sequential(
            nn.ConvTranspose2d(8,1, kernel_size = 4, stride = 2, padding = 1),
            nn.ReLU()
        )

    def forward(self,x):

        encoded_input  = self.encoder_conv1(x.float())
        encoded_input  = self.encoder_conv2(encoded_input)
        encoded_input  = self.encoder_conv3(encoded_input)

        reconstructed_input = self.decoder_conv1(encoded_input)
        reconstructed_input = self.decoder_conv2(reconstructed_input)
        reconstructed_input = self.decoder_conv3(reconstructed_input)

        return reconstructed_input,encoded_input
```

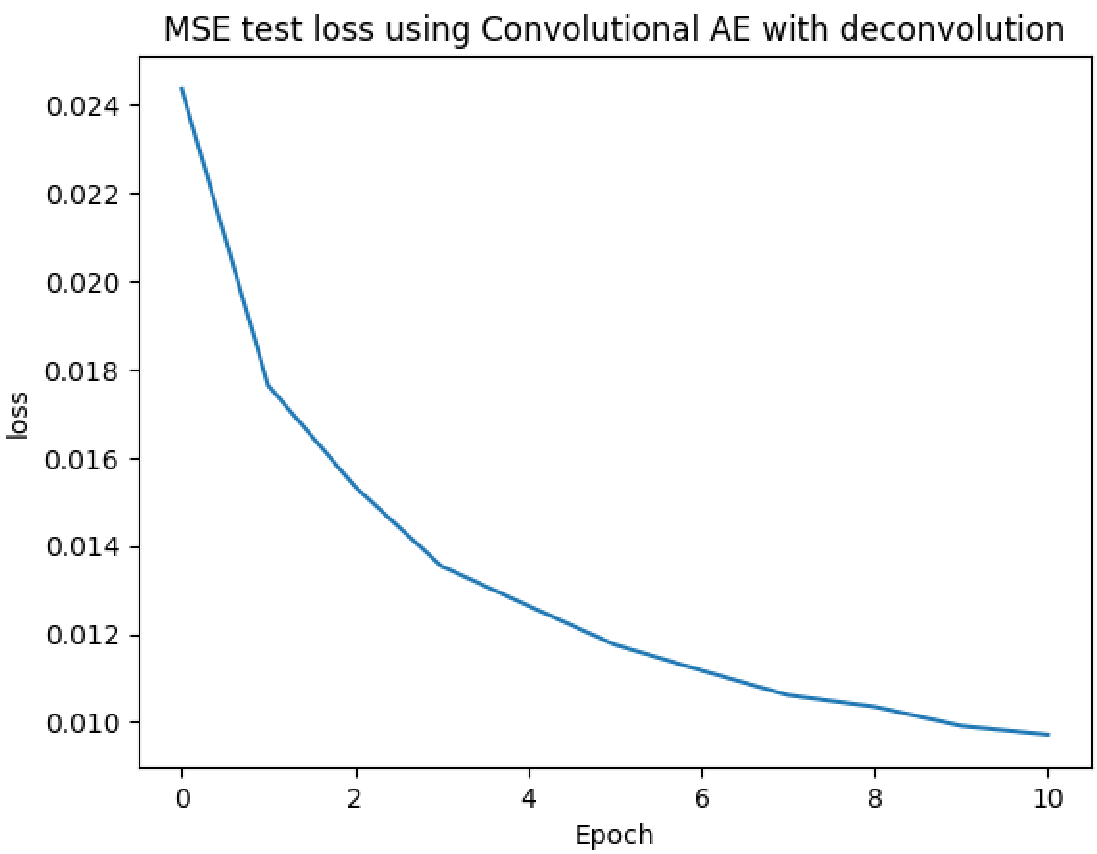
[48]:
```
print(conv_AE_with_deconv().to(device))
```

```
conv_AE_with_deconv(
  (encoder_conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_conv3): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (decoder_conv1): Sequential(
    (0): ConvTranspose2d(16, 16, kernel_size=(3, 3), stride=(2, 2))
    (1): ReLU()
  )
  (decoder_conv2): Sequential(
    (0): ConvTranspose2d(16, 8, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1))
    (1): ReLU()
  )
  (decoder_conv3): Sequential(
    (0): ConvTranspose2d(8, 1, kernel_size=(4, 4), stride=(2, 2), padding=(1,
1))
    (1): ReLU()
  )
)
```

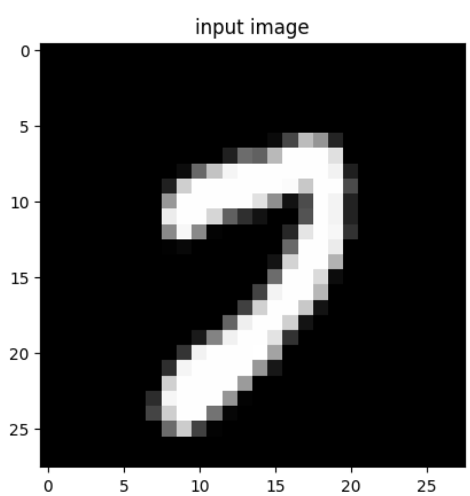
[49]:
```
model_Q5_b = conv_AE_with_deconv().to(device)
optimizer = torch.optim.Adam(model_Q5_b.parameters(), lr=learning_rate)
train_losses_AE_Q5_b , test_losses_AE_Q5_b =␣
  ↪train_test(model_Q5_b,device,train_loader,test_loader,optimizer,lossfn,q5_flag=True)
plot_losses(train_losses_AE_Q5_b, test_losses_AE_Q5_b, model_name =␣
  ↪"Convolutional AE with deconvolution")
```

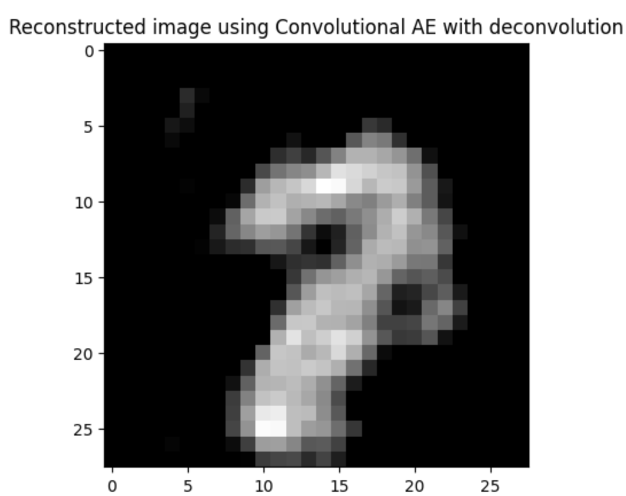


MSE train loss using Convolutional AE with deconvolution
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
Loss
0
2
4
6
8
10
Epoch



[50]:
```
index = random.randint(0,9999)
test_image = test_loader.dataset.data[index, :, :].clone()
plot_reconstructed_image(model_Q5_b,device,test_image,␣
↪model_name="Convolutional AE with deconvolution",q5_flag=True)
```





```
[51]: class conv_AE_with_deconv_unpool(nn.Module):
    def __init__(self):
        super(conv_AE_with_deconv_unpool,self).__init__()

         #encoder
        self.encoder_conv1 = nn.Sequential(
            nn.Conv2d(1,8, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = (2,2),return_indices = True)
        )
        self.encoder_conv2 = nn.Sequential(
            nn.Conv2d(8,16, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = (2,2),return_indices = True)
        )
        self.encoder_conv3 = nn.Sequential(
            nn.Conv2d(16,16, kernel_size = 3, stride = 1,padding= 1),
            nn.ReLU(),nn.MaxPool2d(kernel_size = (2,2),return_indices = True)
        )

        #initializing the decoder module
        self.decoder_conv1 = nn.Sequential(
            nn.ConvTranspose2d(16,16, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU()
        )
        self.decoder_conv2 = nn.Sequential(
            nn.ConvTranspose2d(16,8, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU()
        )
        self.decoder_conv3 = nn.Sequential(
            nn.ConvTranspose2d(8,1, kernel_size = 3, stride = 1, padding = 1),
            nn.ReLU()
        )

        #unpooling
        self.unpool = nn.MaxUnpool2d(kernel_size = (2,2))

    def forward(self,x): #defines the forward pass and also the structure of␣
 ↪the network thus helping backprop

        encoded_input,indices1  = self.encoder_conv1(x.float())
        encoded_input,indices2  = self.encoder_conv2(encoded_input)
        encoded_input,indices3  = self.encoder_conv3(encoded_input)

        reconstructed_input = self.
 ↪unpool(encoded_input,indices3,output_size=torch.Size([batch_size, 16, 7, 7]))
```

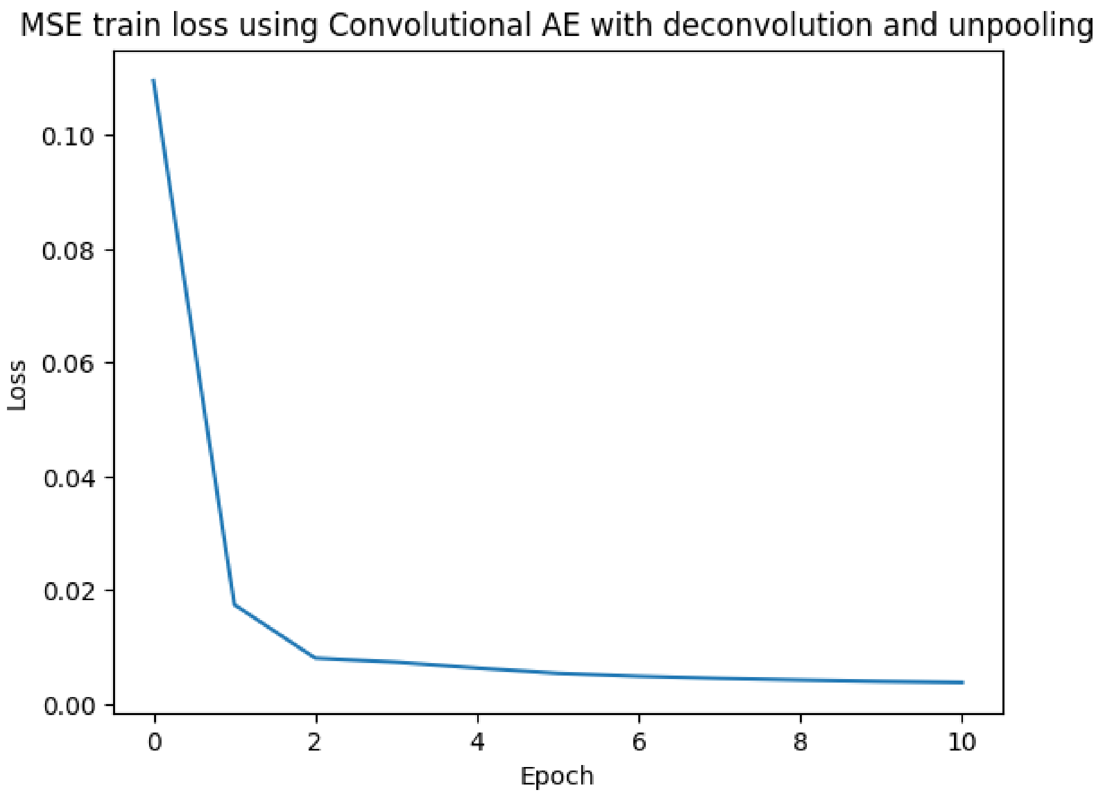
```python
        reconstructed_input = self.decoder_conv1(reconstructed_input)
        reconstructed_input = self.unpool(reconstructed_input,indices2)
        reconstructed_input = self.decoder_conv2(reconstructed_input)
        reconstructed_input = self.unpool(reconstructed_input,indices1)
        reconstructed_input = self.decoder_conv3(reconstructed_input)

        return reconstructed_input,encoded_input
```

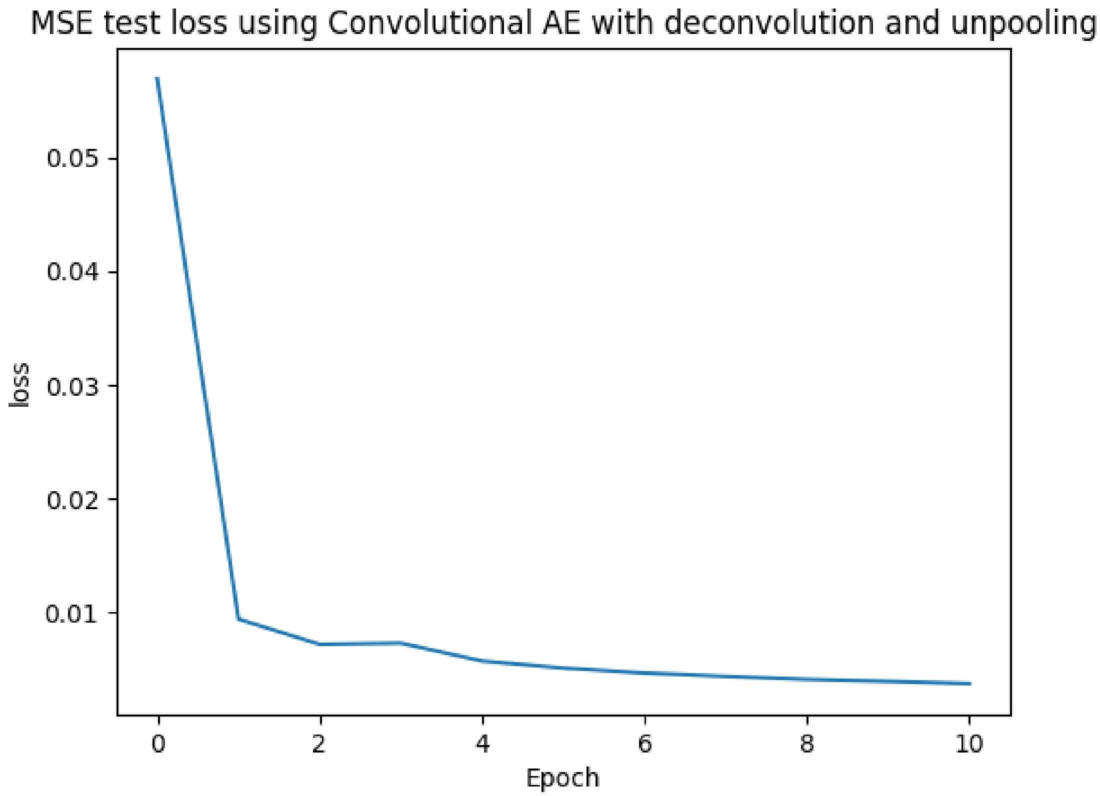
[52]:
```python
from torchsummary import summary
print(conv_AE_with_deconv_unpool().to(device))
```

```
conv_AE_with_deconv_unpool(
  (encoder_conv1): Sequential(
    (0): Conv2d(1, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_conv2): Sequential(
    (0): Conv2d(8, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (encoder_conv3): Sequential(
    (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), padding=0, dilation=1,
ceil_mode=False)
  )
  (decoder_conv1): Sequential(
    (0): ConvTranspose2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU()
  )
  (decoder_conv2): Sequential(
    (0): ConvTranspose2d(16, 8, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU()
  )
  (decoder_conv3): Sequential(
    (0): ConvTranspose2d(8, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (1): ReLU()
  )
  (unpool): MaxUnpool2d(kernel_size=(2, 2), stride=(2, 2), padding=(0, 0))
)
```

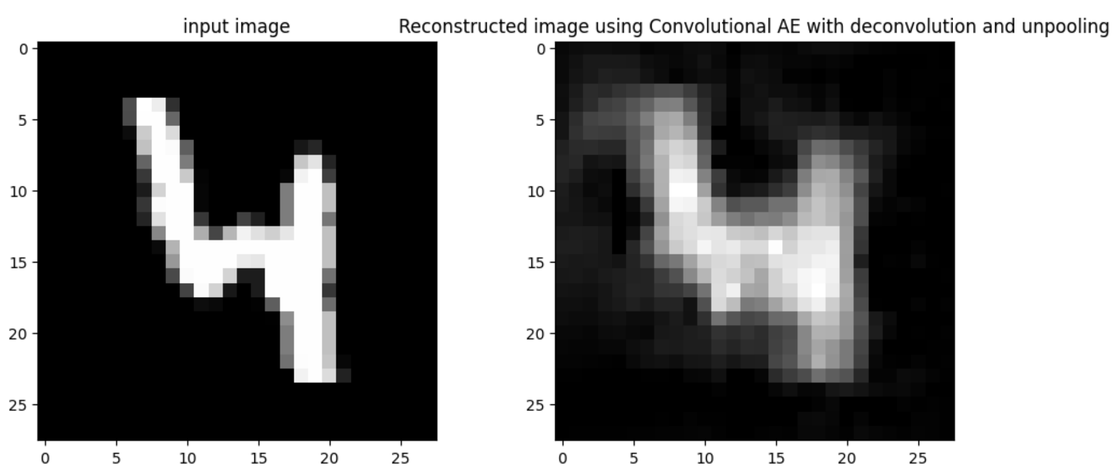
[53]:

```
model_Q5_c = conv_AE_with_deconv_unpool().to(device)
optimizer = torch.optim.Adam(model_Q5_c.parameters(), lr=learning_rate)
train_losses_AE_Q5_c , test_losses_AE_Q5_c =␣
↪train_test(model_Q5_c,device,train_loader,test_loader,optimizer,lossfn,q5_flag=True)
plot_losses(train_losses_AE_Q5_c, test_losses_AE_Q5_c, model_name =␣
↪"Convolutional AE with deconvolution and unpooling")
```





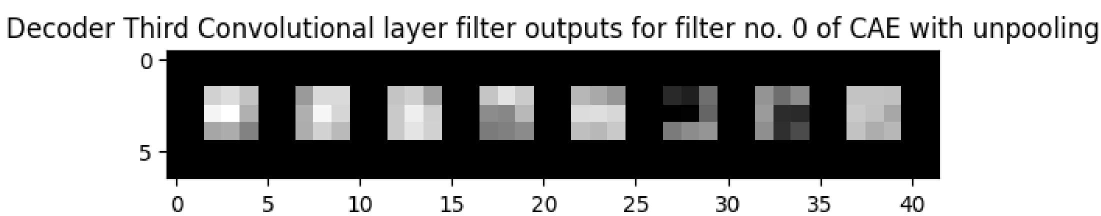
[54]:
```
index = random.randint(0,9999)
test_image = test_loader.dataset.data[index, :, :].clone()
plot_reconstructed_image(model_Q5_c,device,test_image,␣
↪model_name="Convolutional AE with deconvolution and unpooling",q5_flag=True)
```

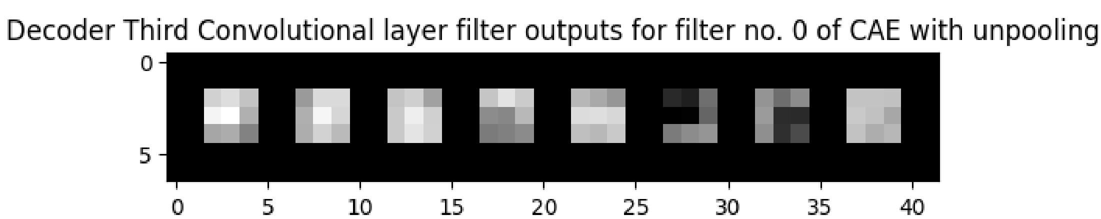


[55]:
```
from torchvision.utils import make_grid
```

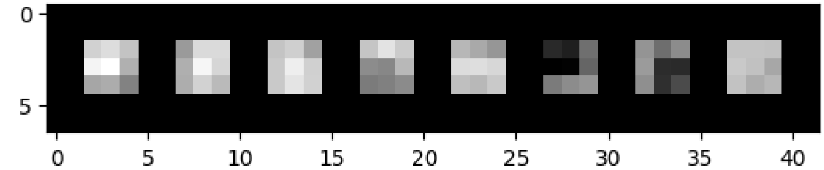
[56]:
```
def␣
↪conv_AE_visualize_decoder_weights(model,model_name,device,decoder_conv_filter,decoder_layer
↪
    if(device == torch.device('cuda')):
        decoder_conv_filter = decoder_conv_filter.cpu()
    #normalize the filter weights
    decoder_conv_filter -= decoder_conv_filter.min()
    decoder_conv_filter /= decoder_conv_filter.max()
    (x,y,z,w) = decoder_conv_filter.size()
    filt_ind = np.random.randint(0 ,decoder_conv_filter.size()[0],3)
    for ind in filt_ind:
        image = make_grid(decoder_conv_filter[ind].reshape(y,1,z,w))
        image = image.permute(1,2,0)
        plt.imshow(image)
        plt.title("Decoder "+str(decoder_layer)+" Convolutional layer filter␣
↪outputs for filter no. "+str(ind)+" of "+str(model_name))
        plt.show()
```

[57]:
```
conv_3_filter = model_Q5_a.decoder_conv3[0].weight.detach().clone()
conv_AE_visualize_decoder_weights(model=model_Q5_a,model_name="CAE with␣
↪unpooling",device=device,decoder_conv_filter=conv_3_filter,decoder_layer="Third")
```



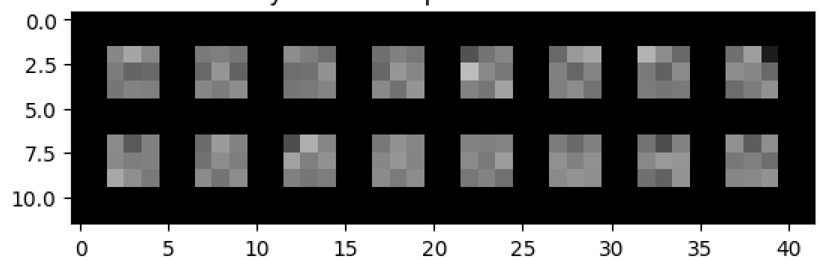


Decoder Third Convolutional layer filter outputs for filter no. 0 of CAE with unpooling

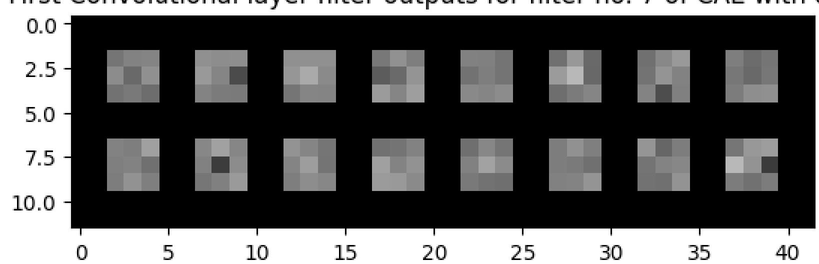


[58]:
```
conv_1_filter = model_Q5_b.decoder_conv1[0].weight.detach().clone()
conv_AE_visualize_decoder_weights(model=model_Q5_b,model_name="CAE with deconvolution",device=device,decoder_conv_filter=conv_1_filter,decoder_layer="First")
```

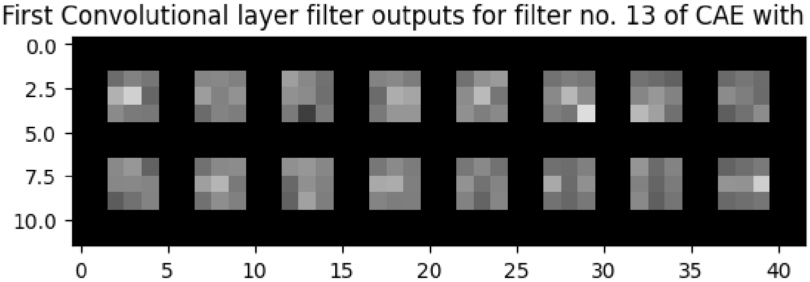
Decoder First Convolutional layer filter outputs for filter no. 1 of CAE with deconvolution



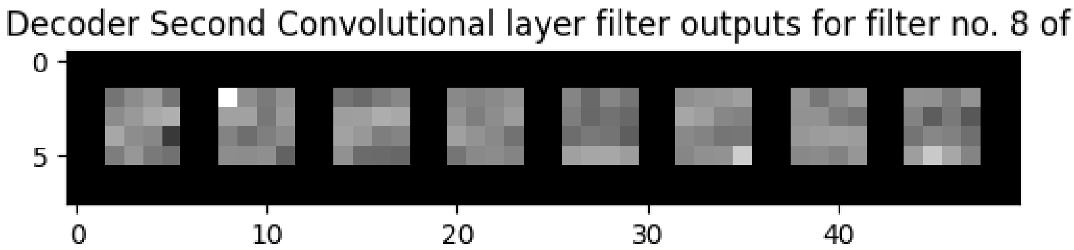
Decoder First Convolutional layer filter outputs for filter no. 7 of CAE with deconvolution

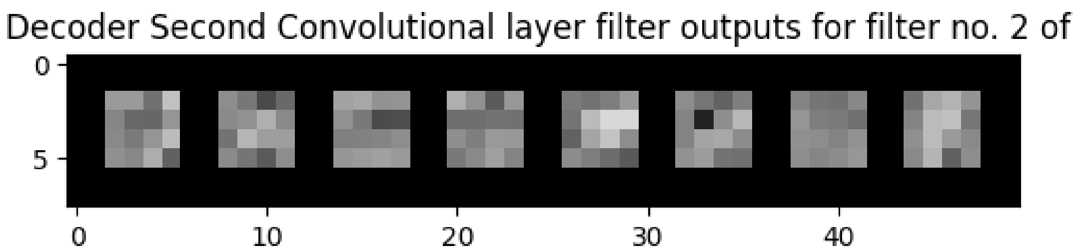


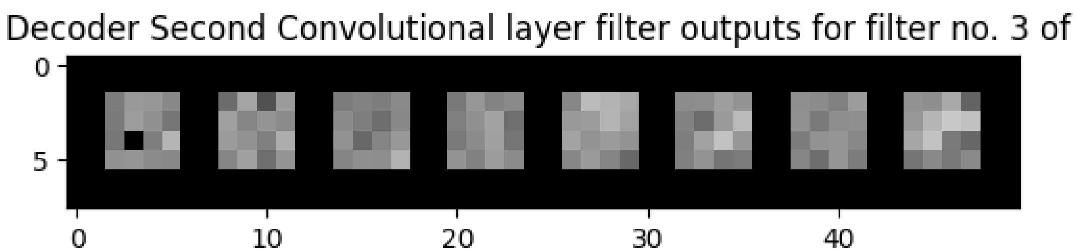
Decoder First Convolutional layer filter outputs for filter no. 13 of CAE with deconvolution



[65]:
```
conv_2_filter = model_Q5_b.decoder_conv2[0].weight.detach().clone()
conv_AE_visualize_decoder_weights(model=model_Q5_b,model_name="",device=device,decoder_conv_fi
```

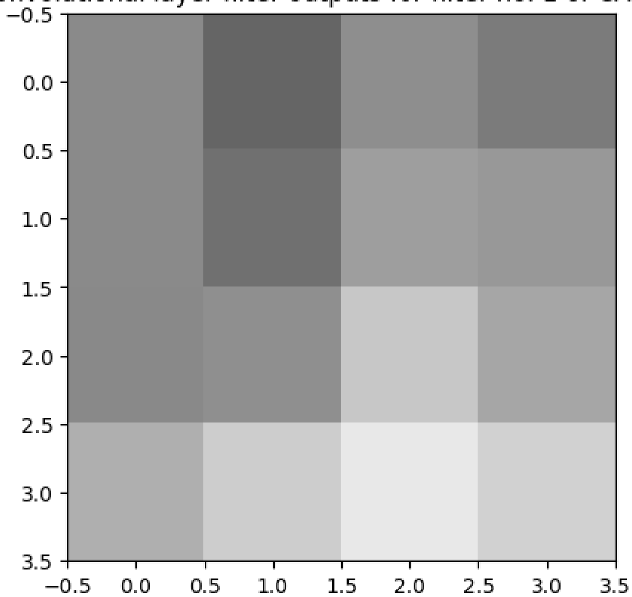




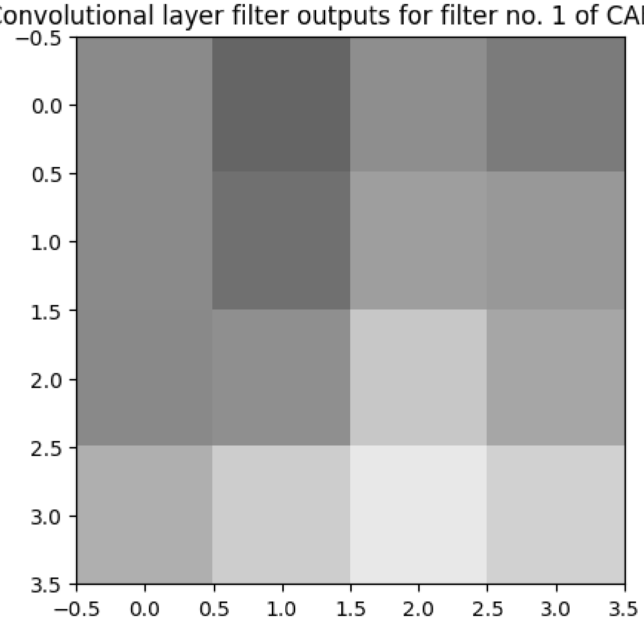


[66]:
```
conv_3_filter = model_Q5_b.decoder_conv3[0].weight.detach().clone()
conv_AE_visualize_decoder_weights(model=model_Q5_b,model_name="CAE with␣
↪deconvolution",device=device,decoder_conv_filter=conv_3_filter,decoder_layer="Third")
```

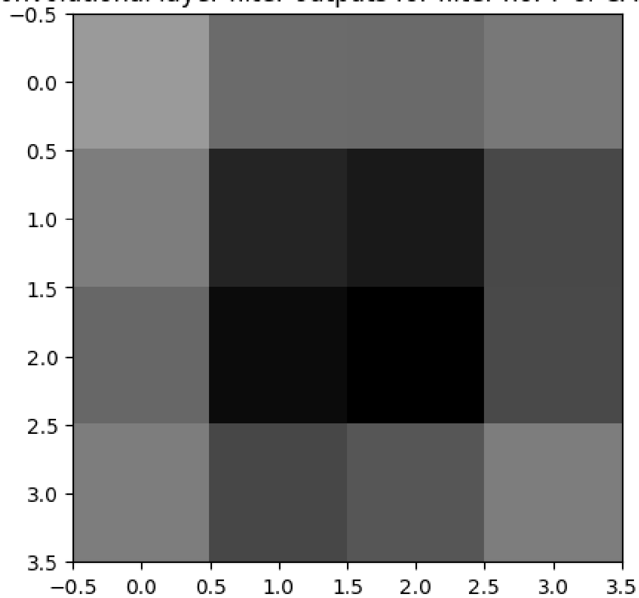
Decoder Third Convolutional layer filter outputs for filter no. 1 of CAE with deconvolution



Decoder Third Convolutional layer filter outputs for filter no. 1 of CAE with deconvolution



Decoder Third Convolutional layer filter outputs for filter no. 7 of CAE with deconvolution



[ ]: