PA4_EE21S063

October 30, 2022

0.1 MANSI KAKKAR (EE21S063)

1 PROGRAMMING ASSIGNMENT 4 OF DEEP LEARNING FOR IMAGING (EE5179)

---

2 Autoencoders

2.1 Libraries Used

[64]:
```
import sys
import numpy as np
import os
import matplotlib.pyplot as plt
import torch
import random
from torchvision import datasets
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader, random_split
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision.utils import make_grid
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error as mse
from PIL import Image

torch.manual_seed(2111)

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 10.0) # set default size of plots
```

2.2 Setting up device

[2]:
```
#Setting the device to GPU
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cpu
```

2.3 Initialisation

[3]:
```
batch_size = 500
data_ind  = [1000, 6, 375, 789, 150, 771, 4150, 6115, 300, 8141]
epochs = 10
```

2.4 Loading the data

[4]:
```
transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.
 ↪1307,), (0.3081,)),])
dataset = datasets.MNIST(root = "data/", train = True, transform = transform,␣
 ↪download = True)
test_dataset = datasets.MNIST(root = "data/", train = False, transform =␣
 ↪transform, download = True)

#Splitting the Training dataset
train_dataset, validation_dataset = train_test_split(dataset, test_size=10000,␣
 ↪train_size = 50000, random_state = None, shuffle = True)

print(f"number of train samples: {len(train_dataset)}")
print(f"number of validation samples: {len(validation_dataset)}")
print(f"number of test samples: {len(test_dataset)}")

train_loader = DataLoader(train_dataset, batch_size = batch_size)
validation_loader = DataLoader(validation_dataset, batch_size = batch_size)
test_loader = DataLoader(test_dataset, batch_size = batch_size)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz to
data/MNIST/raw/train-images-idx3-ubyte.gz

  0%|          | 0/9912422 [00:00<?, ?it/s]


Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz to
data/MNIST/raw/train-labels-idx1-ubyte.gz

  0%|          | 0/28881 [00:00<?, ?it/s]


Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz to
data/MNIST/raw/t10k-images-idx3-ubyte.gz

  0%|          | 0/1648877 [00:00<?, ?it/s]


Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz to
data/MNIST/raw/t10k-labels-idx1-ubyte.gz

  0%|          | 0/4542 [00:00<?, ?it/s]


Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw

number of train samples: 50000
number of validation samples: 10000
number of test samples: 10000
```

[5]:

```python
print(f"size of train dataloader is :{len(train_loader)}")
print(f"size of validation dataloader is :{len(validation_loader)}")
print(f"size of test dataloader is :{len(test_loader)}")
data = next(iter(train_loader))
train_img, train_target = data
print(f"Train image shape:{train_img.shape}")
print(f"Train Targets shape:{train_target.shape}")

data = next(iter(validation_loader))
val_img, val_target = data
print(f"Validation image shape:{val_img.shape}")
print(f"Validation Targets shape:{val_target.shape}")

data = next(iter(test_loader))
test_img, test_target = data
print(f"Test image shape:{test_img.shape}")
print(f"Test Targets shape:{test_target.shape}")
```

```
size of train dataloader is :100
size of validation dataloader is :20
size of test dataloader is :20
```

```
Train image shape:torch.Size([500, 1, 28, 28])
Train Targets shape:torch.Size([500])
Validation image shape:torch.Size([500, 1, 28, 28])
Validation Targets shape:torch.Size([500])
Test image shape:torch.Size([500, 1, 28, 28])
Test Targets shape:torch.Size([500])
```

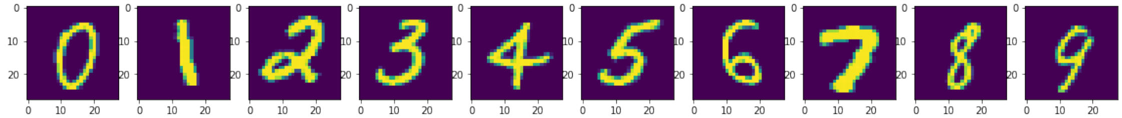
3 1. Comparing PCA and Autoencoders

3.1 PCA

[6]:
```python
def visualise_data(data, data_ind = data_ind):
  fig, ax = plt.subplots(1, 10, figsize = (20,20))
  for i,ind in enumerate(data_ind):
    image_sample = np.asarray(data[ind],dtype = np.uint8).reshape(28,28)
    image_obj = Image.fromarray(image_sample)
    print('for digit '+str(i))
    ax[i].imshow(image_obj)
  plt.show()
```

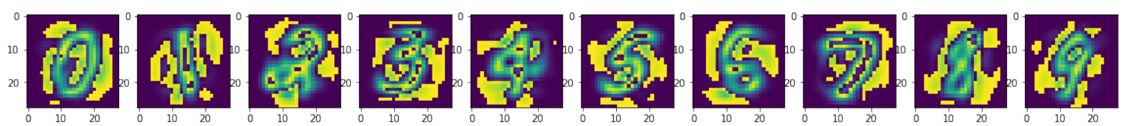
[7]:
```python
pca_train = (dataset.data).reshape(60000,784)
pca_test = (test_dataset.data).reshape(10000,784)
data = torch.cat((pca_train, pca_test))
print(data.shape)
visualise_data(data)
```

```
torch.Size([70000, 784])
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



[8]:
```
pca_data = data.reshape(70000, 784)
pca_data = np.asarray(pca_data)/255
pca_30 = PCA(n_components = 30)
pca_30.fit(pca_data)
train_pca = pca_30.transform(pca_data)
recover_data = pca_30.inverse_transform(train_pca)
pca_error = mse(pca_data, recover_data)
print(f'reconstruction error is : {pca_error}')

visualise_data(recover_data*255)
```

```
reconstruction error is : 0.018056392603431226
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



[9]:
```
# def PCA_implementation(data, num_of_values = 30):
#   mean =  torch.mean(data.float(),0)
#   data = data - mean.expand_as(data)
#   U,S,V = torch.svd(torch.mm(data,torch.t(data)))
#   return torch.mm(data,U[:,:num_of_values]),U[:,:num_of_values]
```

[10]:
```
# recover_data = PCA_implementation(data, 30)
#Visualisation
# for i,ind in enumerate(data_ind):
#   image_sample = np.asarray(255*recover_data[ind],dtype = np.uint8).
↪reshape(28,28)
#   #creating image object
#   image_obj = Image.fromarray(image_sample)
#   print('Digit '+str(i))
#   plt.imshow(image_obj)
#   plt.show()
```

3.2 Training Function

[47]:
```
def Train(model, optimizer, criterion, epochs = 10, sparse = False, l1_reg = 0.
 ↪001, denoise = False, noise_val = 0.1, model_flag = 0):
  model.train()
  training_loss = []
  validation_loss = []
  # validation_acc = []
  for epoch in range(epochs):
    # val_correct = 0
    for i,(images, labels) in enumerate(train_loader):
      images = images.to(device)
      labels = labels.to(device)
      if(model_flag == 0):
        images = images.reshape(batch_size, 784)

      if(denoise == True):
        noisy = Add_Noise(images, noise_val)
        output, encoded = model(noisy)
      else:
        output, encoded = model(images)

      t_loss = criterion(output, images)

      if(sparse == True):
        t_loss += l1_reg*torch.linalg.norm(encoded,1)

      optimizer.zero_grad()
      t_loss.backward()
      optimizer.step()

    training_loss.append(t_loss)

    for i,(images, labels) in enumerate(validation_loader):
      images = images.to(device)
      labels = labels.to(device)

      if(model_flag==0):
        images = images.reshape(batch_size, 784)

      output, encoded = model(images)
      v_loss = criterion(output, images)
      if(sparse == True):
        v_loss += l1_reg*torch.linalg.norm(encoded,1)
    validation_loss.append(v_loss)
    # validation_acc.append(val_correct/500)
```

```
    print('Epochs: {}/{} ||| with Training Loss = {} ||| Validation Loss = {}'.
↪format(epoch+1, epochs, t_loss, v_loss))
  return training_loss, validation_loss
```

3.3 Functions to Visualise Loss and Accuracy

[13]:
```
def visualise_loss(trainingloss, valloss):
  with torch.no_grad():
    plt.figure()
    xloss=np.arange(len(trainingloss))
    plt.plot(xloss, trainingloss, color='Blue',label='Training Loss')
    plt.plot(xloss, valloss, color='Orange',label='Validation Loss')
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
    plt.legend()
    plt.xlabel('Iterations')
    plt.ylabel(' Loss')
    plt.title(' Loss vs epochs')

def visualise_accuracy(loss_type, loss_list, accuracy):
  with torch.no_grad():
    plt.figure()
    xloss=np.arange(len(loss_list))
    plt.plot(xloss, accuracy)
    plt.grid(b=True, which='major', color='#666666', linestyle='-')
    plt.minorticks_on()
    plt.grid(b=True, which='minor', color='#999999', linestyle='-', alpha=0.2)
    plt.xlabel('Epochs')
    plt.ylabel(loss_type + 'Accuracy')
    plt.title('accuracy vs epochs')
```

3.4 Function to Visualise model

[50]:
```
def visualise_model(model, data_ind, data, model_flag = 0):
  with torch.no_grad():
    fig, ax = plt.subplots(1, 10, figsize = (20,20))
    for i,ind in enumerate(data_ind):
      if(model_flag==0):
        output, encoded = model(data[ind].reshape(1,784))#.cuda())
      else:
        output, encoded = model(data[ind].reshape(1, 28, 28))
      image_sample = np.asarray(output,dtype = np.uint8).reshape(28,28)    #.
↪detach().cpu()
```

```
        image_obj = Image.fromarray(image_sample)
        print('for digit '+str(i))
        ax[i].imshow(image_obj)
    plt.show()
```

3.5 Autoencoder

[15]:
```
class autoencoder(nn.Module):
  def __init__(self):
    super(autoencoder, 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):
    encoded = self.encoder(x.float())
    out = self.decoder(encoded)
    return out, encoded
```

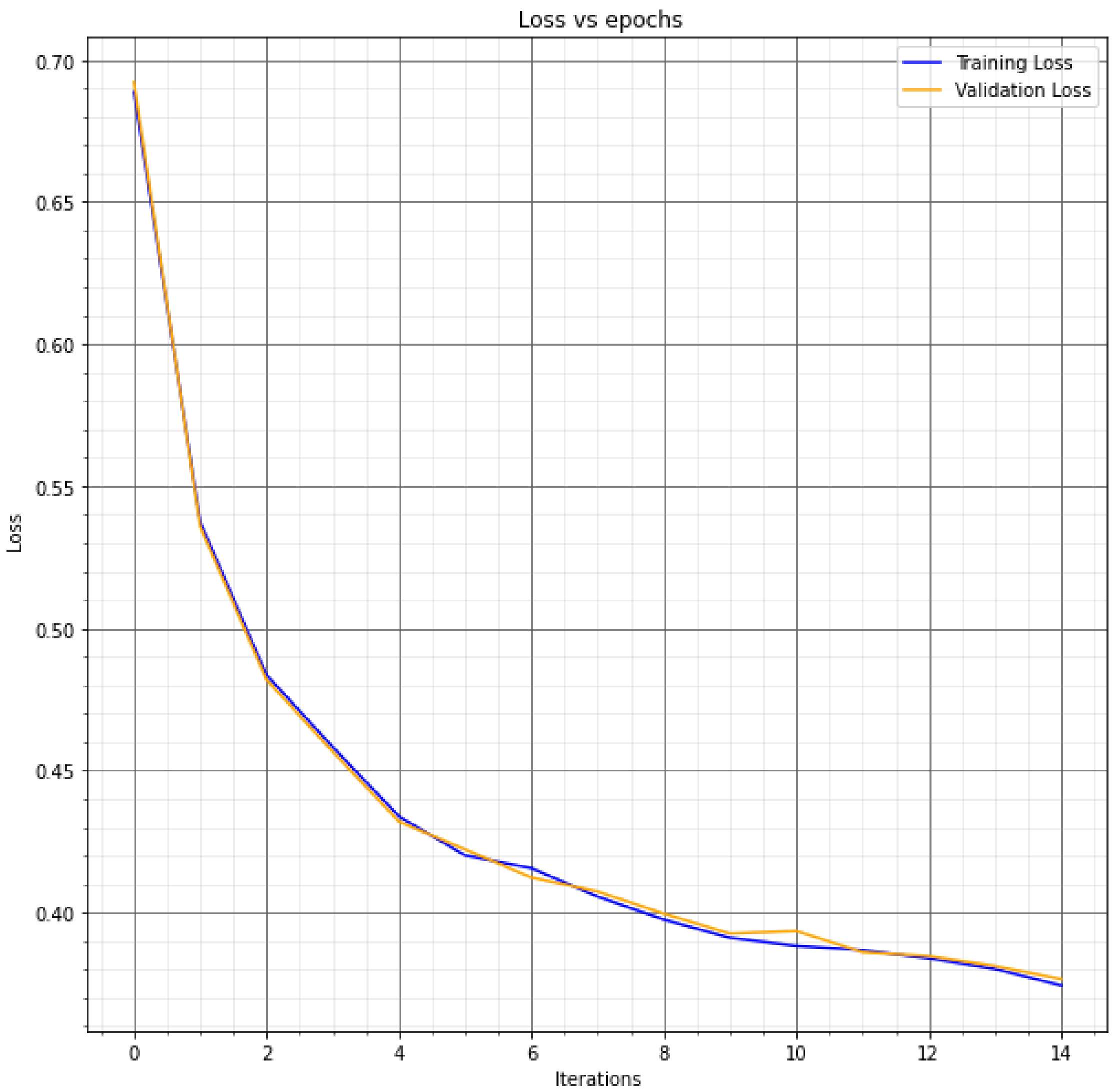
Calling Autoencoder function

[16]:
```
epochs = 15
model1 = autoencoder().to(device)
optimizer1 = optim.Adam(model1.parameters(), lr = 0.003)
criterion = nn.MSELoss()
training_loss, val_loss = Train(model1, optimizer1, criterion, epochs)
```

```
Epochs: 1/15 ||| with Training Loss = 0.6887065768241882 ||| Validation Loss =
0.6920874714851379
Epochs: 2/15 ||| with Training Loss = 0.5373498201370239 ||| Validation Loss =
0.5356813073158264
Epochs: 3/15 ||| with Training Loss = 0.483600914478302 ||| Validation Loss =
0.4819205701351166
Epochs: 4/15 ||| with Training Loss = 0.4582567811012268 ||| Validation Loss =
0.45659059286117554
Epochs: 5/15 ||| with Training Loss = 0.4338372051715851 ||| Validation Loss =
0.43202462792396545
Epochs: 6/15 ||| with Training Loss = 0.4201061725616455 ||| Validation Loss =
0.42222604155540466
Epochs: 7/15 ||| with Training Loss = 0.41564664244651794 ||| Validation Loss =
0.4123659133911133
Epochs: 8/15 ||| with Training Loss = 0.405624121427536 ||| Validation Loss =
0.4074176549911499
Epochs: 9/15 ||| with Training Loss = 0.39755842089653015 ||| Validation Loss =
0.39963024854660034
```

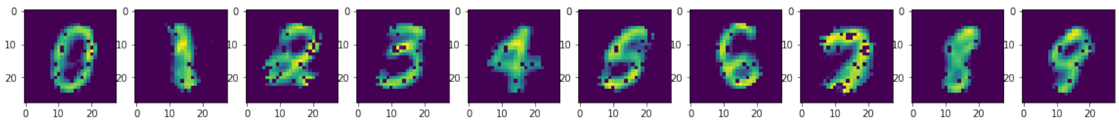
```
Epochs: 10/15 ||| with Training Loss = 0.3911236822605133 ||| Validation Loss =
0.3926449716091156
Epochs: 11/15 ||| with Training Loss = 0.38827431201934814 ||| Validation Loss =
0.3935399055480957
Epochs: 12/15 ||| with Training Loss = 0.38678452372550964 ||| Validation Loss =
0.3860878050327301
Epochs: 13/15 ||| with Training Loss = 0.38392937183380127 ||| Validation Loss =
0.3847467005252838
Epochs: 14/15 ||| with Training Loss = 0.38020917773246765 ||| Validation Loss =
0.3812178373336792
Epochs: 15/15 ||| with Training Loss = 0.3743497431278229 ||| Validation Loss =
0.3766360878944397
```

[17]:
```
visualise_loss(training_loss, val_loss)
```



[51]:
```
visualise_model(model1, data_ind, data)
```

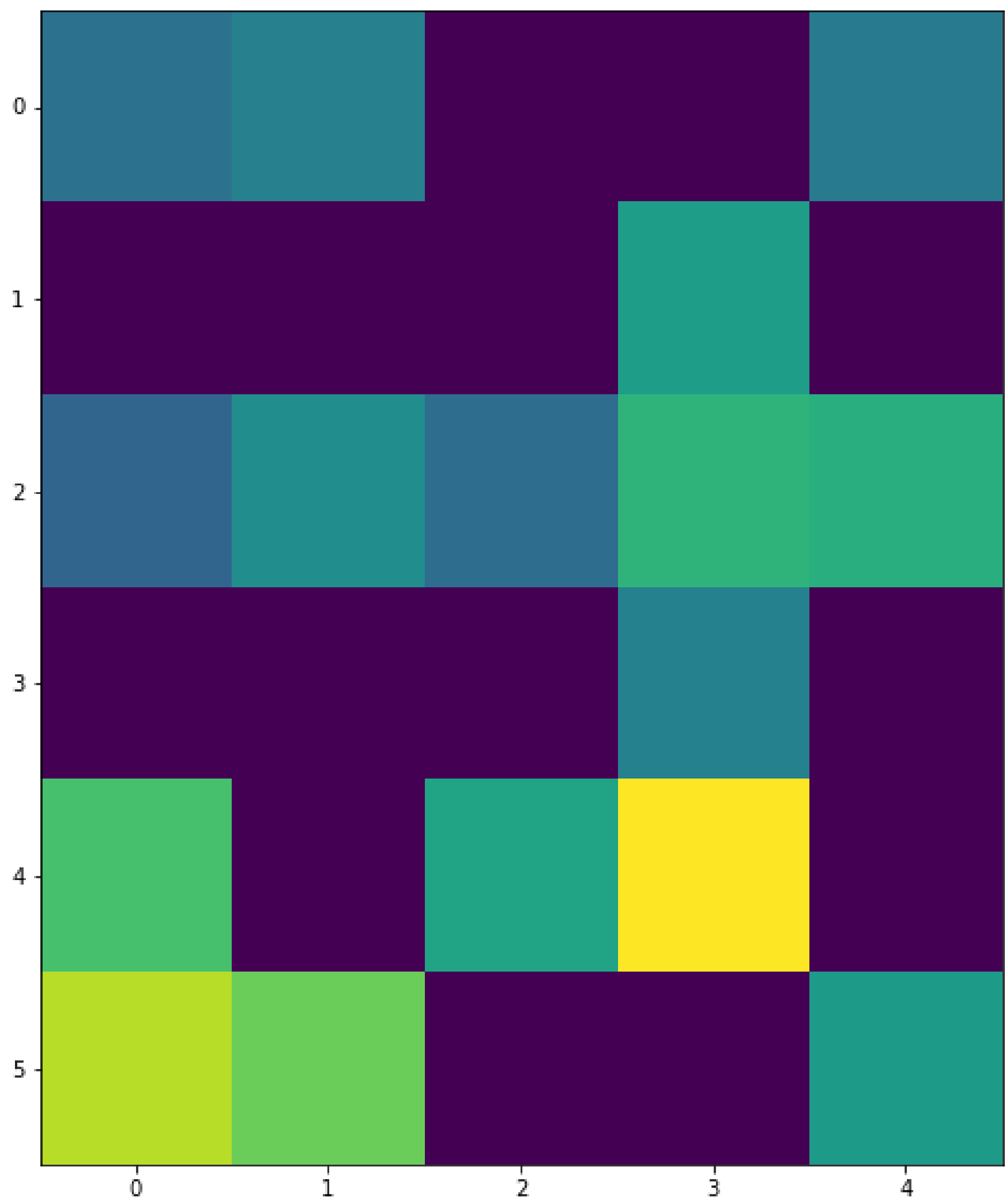
```
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



Visualise the encoded output

[29]:
```
with torch.no_grad():
  output1, encoded1=model1.forward(data[data_ind[3]].reshape(1,784).float())
  print(encoded1.shape)
  plt.imshow(encoded1.reshape(6, 5))
```

```
torch.Size([1, 30])
```



3.6 Observations:

- From the PCA output we can still interpret the digits but its not neatly displayed(includes a lot of noise in the signal)
- Whereas in the standard Autoencoder output, digits are more clearly visible and interpretable.

- Loss decreases to considerable amount for both training as well as validation

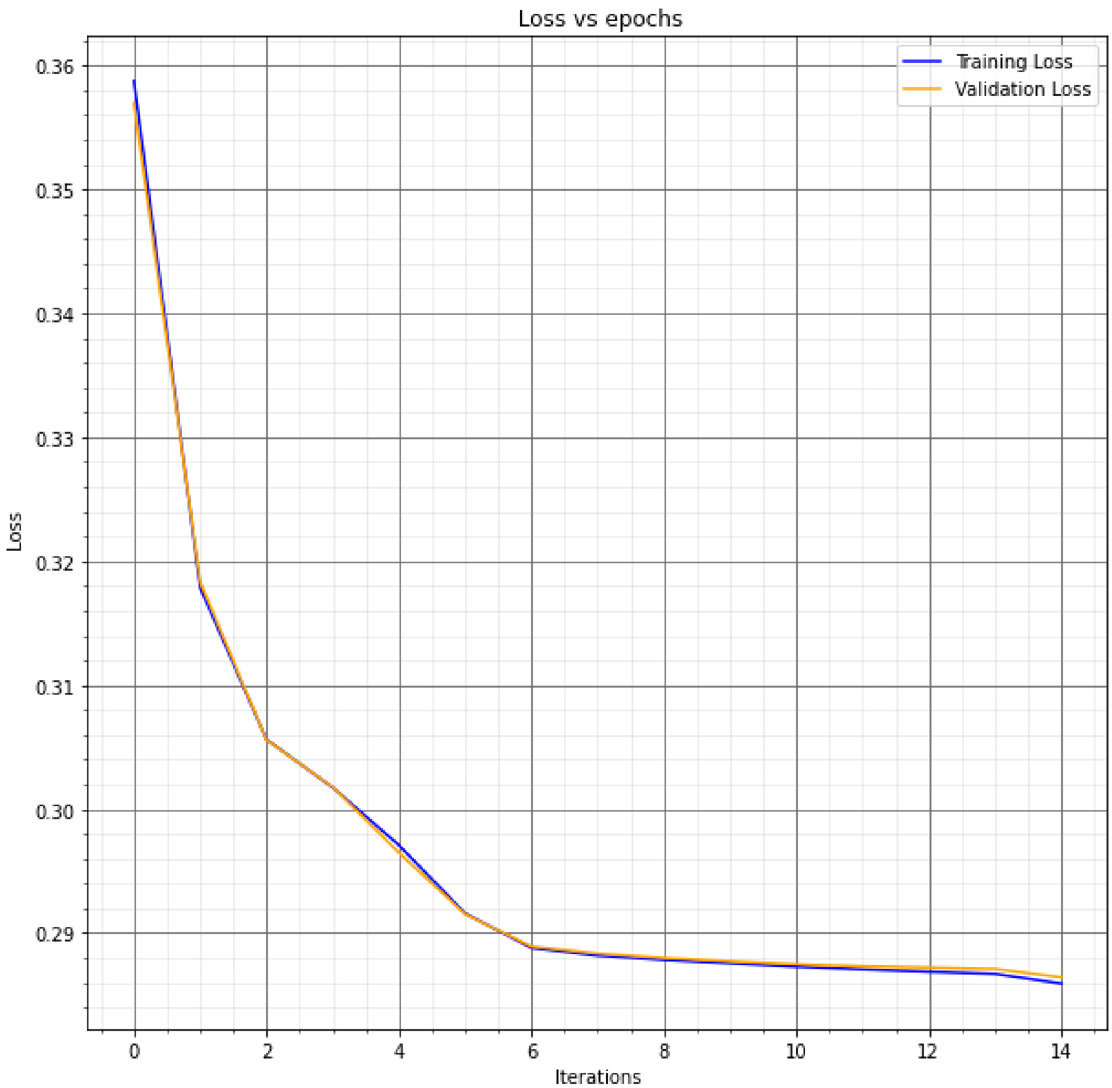
4 2. Experimenting with hidden units of varying sizes

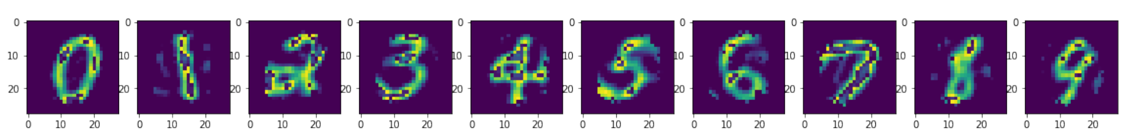
[30]:
```python
class autoencoder_hidden(nn.Module):
  def __init__(self, hidden_size):
    super(autoencoder_hidden, self).__init__()
    self.encoder = nn.Sequential(nn.Linear(784, hidden_size), nn.ReLU())
    self.decoder = nn.Sequential(nn.Linear(hidden_size, 784), nn.ReLU())

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

[63]:
```python
epochs = 15
Xnp=np.random.normal(0.5,0.05,(1,28,28))
X=torch.from_numpy(Xnp)
hidden_size = [64, 128, 256]
for i in range(3):
  print("for Hidden Size = {}".format(hidden_size[i]))
  model2 = autoencoder_hidden(hidden_size[i]).to(device)
  optimizer2 = optim.Adam(model2.parameters(), lr = 0.003)
  criterion = nn.MSELoss()
  training_loss, val_loss = Train(model2, optimizer2, criterion, epochs)
  visualise_loss(training_loss, val_loss)
  visualise_model(model2, data_ind, data)
  with torch.no_grad():
    output1=model2(X.reshape(1,784).float())
    plt.figure(1)
    fig,ax = plt.subplots()
    ax=plt.subplot(2,1,1)
    ax.set_xticks([])
    ax.set_yticks([])
    im=ax.imshow(X.detach().numpy()[0],cmap='gray')
    fig,ax = plt.subplots()
    ax=plt.subplot(2,1,2)
    ax.set_xticks([])
    ax.set_yticks([])
    im=ax.imshow(output1[0].reshape(28,28),cmap='gray')
    plt.show()
```

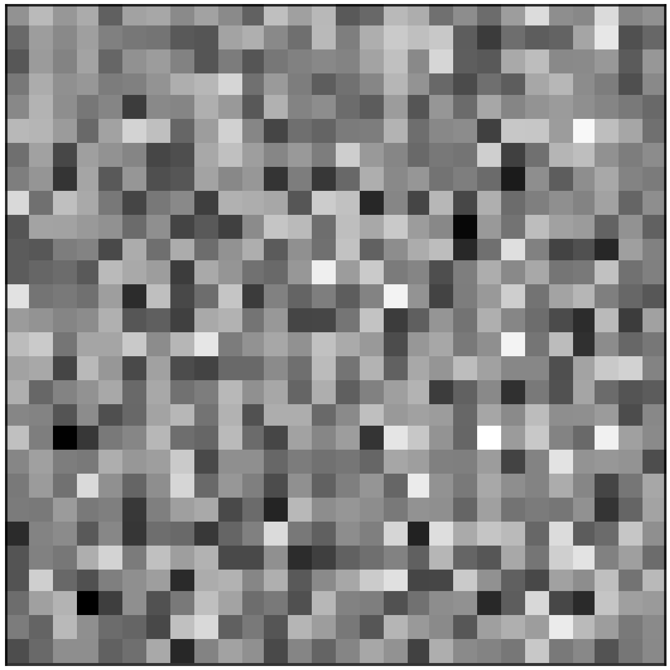
```
for Hidden Size = 64
Epochs: 1/15 ||| with Training Loss = 0.35872408747673035 ||| Validation Loss =
0.3569408357143402
Epochs: 2/15 ||| with Training Loss = 0.3178459703922272 ||| Validation Loss =
```

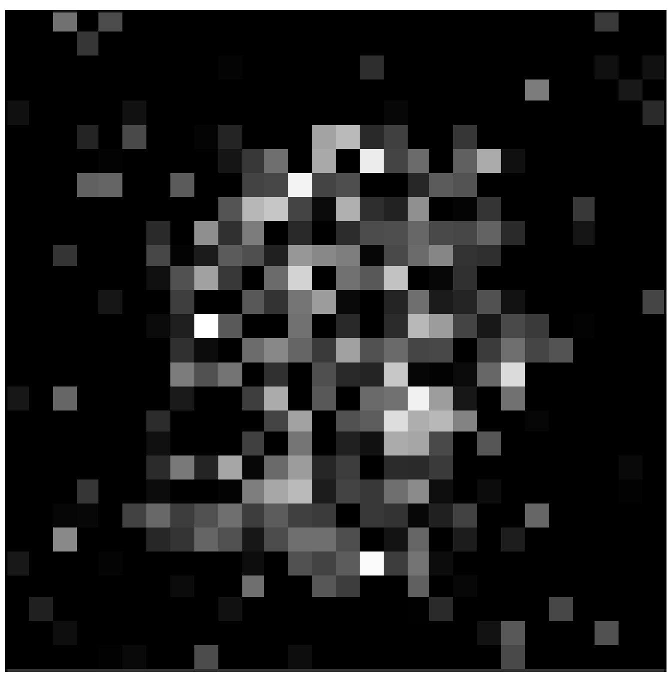
```
0.31832829117774963
Epochs: 3/15 ||| with Training Loss = 0.30561742186546326 ||| Validation Loss =
0.3055766224861145
Epochs: 4/15 ||| with Training Loss = 0.30175521969795227 ||| Validation Loss =
0.30178868770599365
Epochs: 5/15 ||| with Training Loss = 0.29709362983703613 ||| Validation Loss =
0.2964833974838257
Epochs: 6/15 ||| with Training Loss = 0.2916001081466675 ||| Validation Loss =
0.29153499007225037
Epochs: 7/15 ||| with Training Loss = 0.2888093888759613 ||| Validation Loss =
0.2889271080493927
Epochs: 8/15 ||| with Training Loss = 0.2881960868835449 ||| Validation Loss =
0.2883595824241638
Epochs: 9/15 ||| with Training Loss = 0.28784826397895813 ||| Validation Loss =
0.2880071997642517
Epochs: 10/15 ||| with Training Loss = 0.28757244348526 ||| Validation Loss =
0.2877322733402252
Epochs: 11/15 ||| with Training Loss = 0.2873050570487976 ||| Validation Loss =
0.2875053882598877
Epochs: 12/15 ||| with Training Loss = 0.28708416223526 ||| Validation Loss =
0.28734394907951355
Epochs: 13/15 ||| with Training Loss = 0.2868974804878235 ||| Validation Loss =
0.2872314751148224
Epochs: 14/15 ||| with Training Loss = 0.28672245144844055 ||| Validation Loss =
0.2871205806732178
Epochs: 15/15 ||| with Training Loss = 0.2859320044517517 ||| Validation Loss =
0.28645116090774536
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```





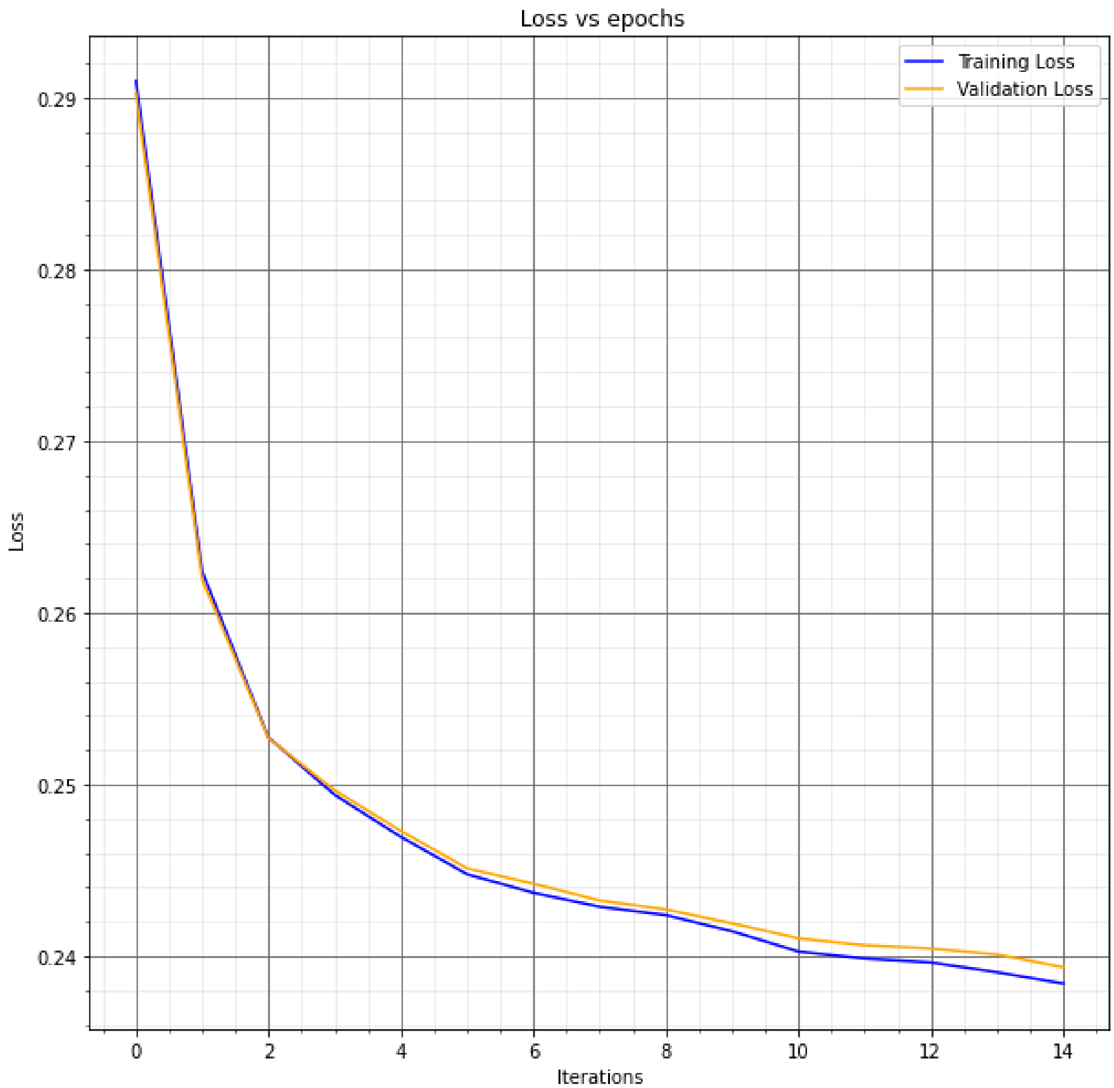
```
<Figure size 720x720 with 0 Axes>
```

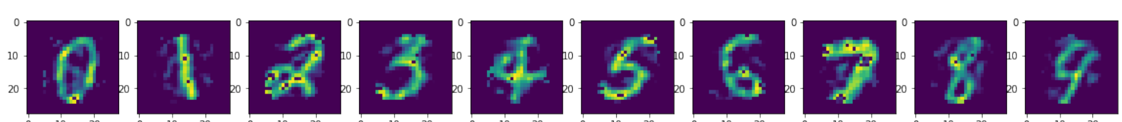




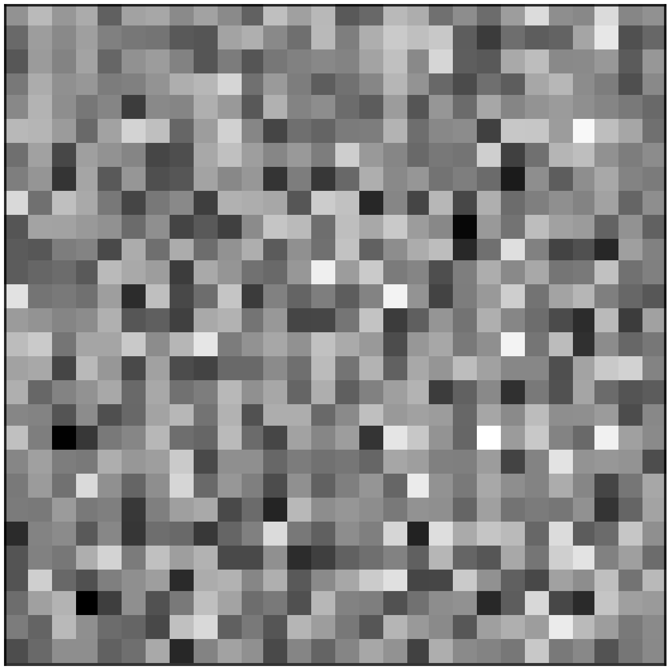
```
for Hidden Size = 128
Epochs: 1/15 ||| with Training Loss = 0.29094651341438293 ||| Validation Loss =
```

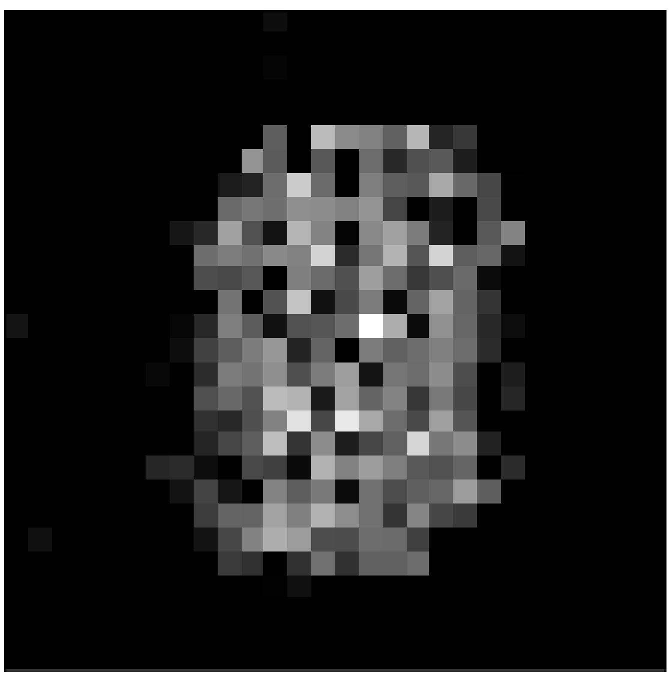
```
0.2902321219444275
Epochs: 2/15 ||| with Training Loss = 0.2623888850212097 ||| Validation Loss =
0.26192641258239746
Epochs: 3/15 ||| with Training Loss = 0.25273483991622925 ||| Validation Loss =
0.25269418954849243
Epochs: 4/15 ||| with Training Loss = 0.2493942528963089 ||| Validation Loss =
0.24966394901275635
Epochs: 5/15 ||| with Training Loss = 0.24694737792015076 ||| Validation Loss =
0.24729479849338531
Epochs: 6/15 ||| with Training Loss = 0.2447780966758728 ||| Validation Loss =
0.2451203614473343
Epochs: 7/15 ||| with Training Loss = 0.2436942160129547 ||| Validation Loss =
0.2442280799150467
Epochs: 8/15 ||| with Training Loss = 0.24288320541381836 ||| Validation Loss =
0.24324625730514526
Epochs: 9/15 ||| with Training Loss = 0.24240043759346008 ||| Validation Loss =
0.242728590965271
Epochs: 10/15 ||| with Training Loss = 0.24145545065402985 ||| Validation Loss =
0.24192163348197937
Epochs: 11/15 ||| with Training Loss = 0.24027711153030396 ||| Validation Loss =
0.2410450428724289
Epochs: 12/15 ||| with Training Loss = 0.23986749351024628 ||| Validation Loss =
0.24063895642757416
Epochs: 13/15 ||| with Training Loss = 0.23964428901672363 ||| Validation Loss =
0.24045178294181824
Epochs: 14/15 ||| with Training Loss = 0.2390761375427246 ||| Validation Loss =
0.2401057481765747
Epochs: 15/15 ||| with Training Loss = 0.23840881884098053 ||| Validation Loss =
0.23936542868614197
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```





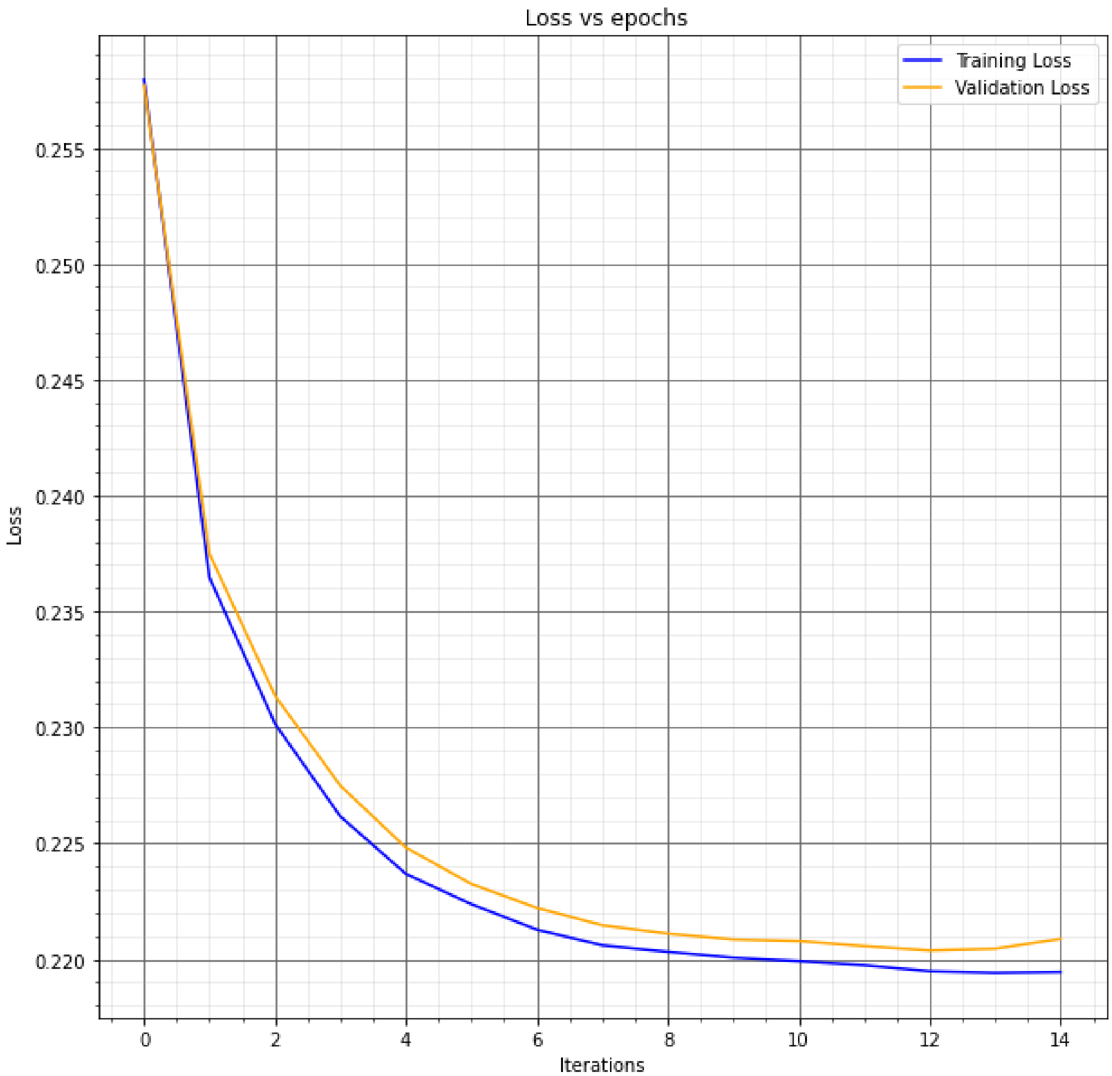
<Figure size 720x720 with 0 Axes>

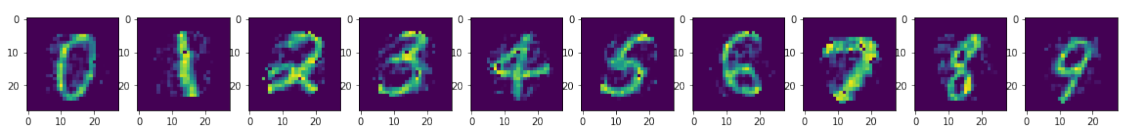




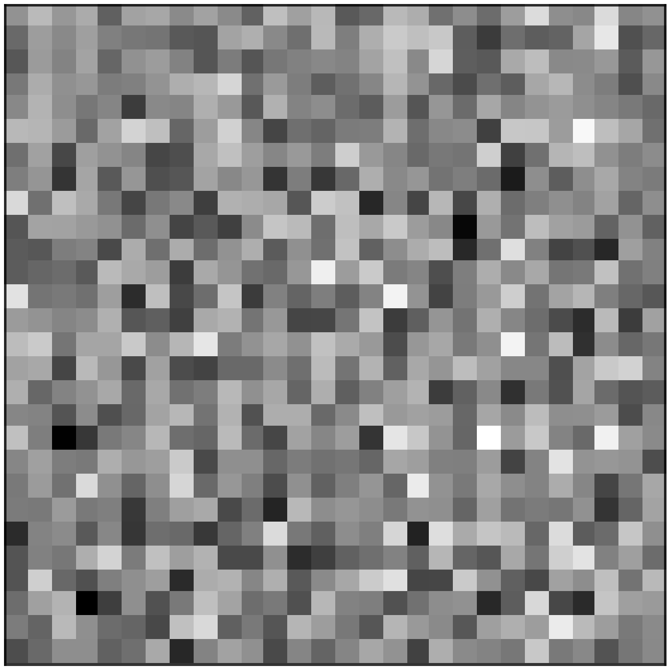
```
for Hidden Size = 256
Epochs: 1/15 ||| with Training Loss = 0.2579302191734314 ||| Validation Loss =
```

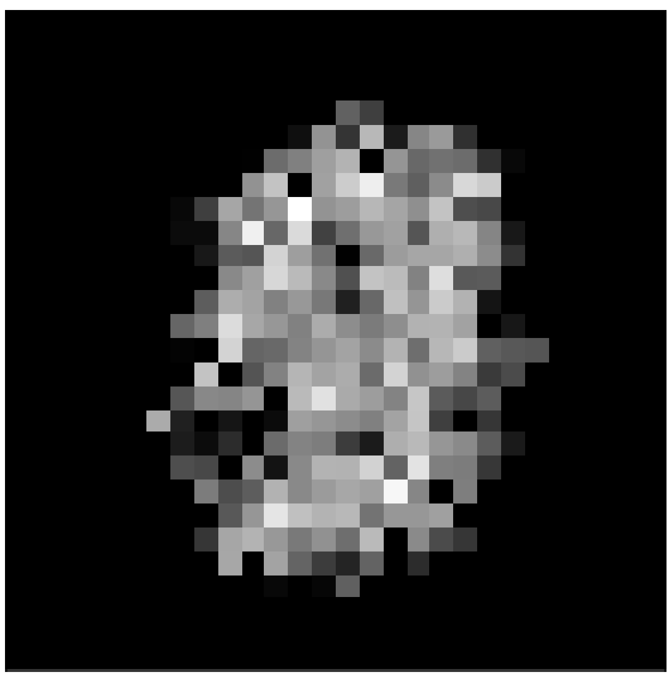
```
0.25769492983818054
Epochs: 2/15 ||| with Training Loss = 0.23645934462547302 ||| Validation Loss =
0.23748131096363068
Epochs: 3/15 ||| with Training Loss = 0.2301545888185501 ||| Validation Loss =
0.23136889934539795
Epochs: 4/15 ||| with Training Loss = 0.22615091502666473 ||| Validation Loss =
0.22747677564620972
Epochs: 5/15 ||| with Training Loss = 0.22367747128009796 ||| Validation Loss =
0.2248115837574005
Epochs: 6/15 ||| with Training Loss = 0.22237077355384827 ||| Validation Loss =
0.22324234247207642
Epochs: 7/15 ||| with Training Loss = 0.22127237915992737 ||| Validation Loss =
0.22221161425113678
Epochs: 8/15 ||| with Training Loss = 0.2205965220928192 ||| Validation Loss =
0.2214614748954773
Epochs: 9/15 ||| with Training Loss = 0.22031590342521667 ||| Validation Loss =
0.22110092639923096
Epochs: 10/15 ||| with Training Loss = 0.22006572782993317 ||| Validation Loss =
0.22084935009479523
Epochs: 11/15 ||| with Training Loss = 0.2199128270149231 ||| Validation Loss =
0.2207784801721573
Epochs: 12/15 ||| with Training Loss = 0.21973708271980286 ||| Validation Loss =
0.2205630987882614
Epochs: 13/15 ||| with Training Loss = 0.2194749265909195 ||| Validation Loss =
0.22037877142429352
Epochs: 14/15 ||| with Training Loss = 0.21941041946411133 ||| Validation Loss =
0.22045297920703888
Epochs: 15/15 ||| with Training Loss = 0.21944372355937958 ||| Validation Loss =
0.22087204456329346
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```





```
<Figure size 720x720 with 0 Axes>
```

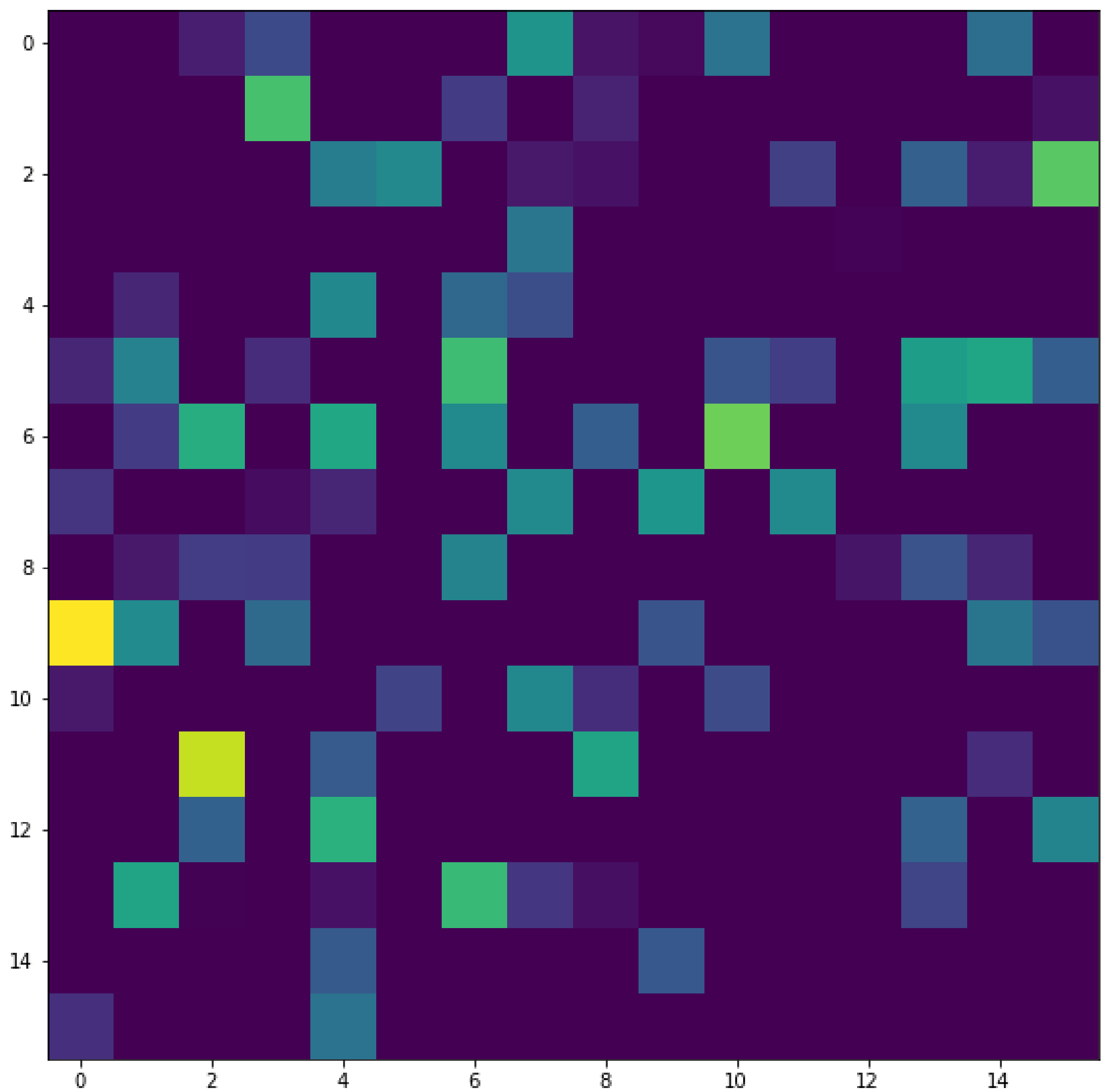




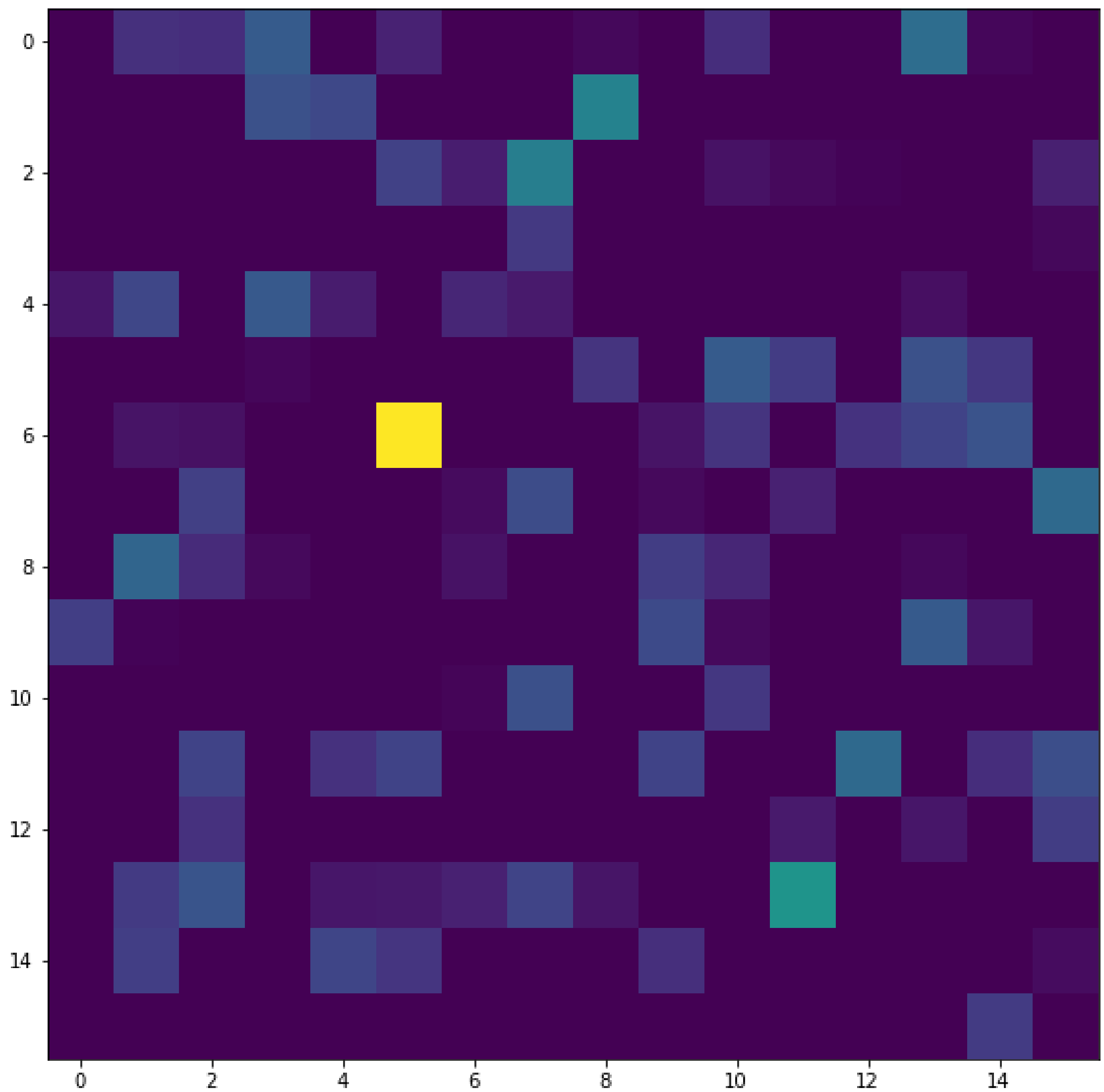
[43]:
```
print("Average Hidden layer activations for 10 images for the hidden layer =␣
↪256")
```

```
with torch.no_grad():
  sum = 0
  for i in range(10):

    output2, encoded2=model2.forward(data[data_ind[i]].reshape(1,784).float())
    avg = torch.norm(encoded2, p=1)/256.0
    print(f'for the digit = {i} average value is = {avg.detach().numpy()}')
    sum+=avg.detach().numpy()
    plt.imshow(encoded2.reshape(16, 16))
    plt.show()
  print("Average of these values",sum/10.0)
```

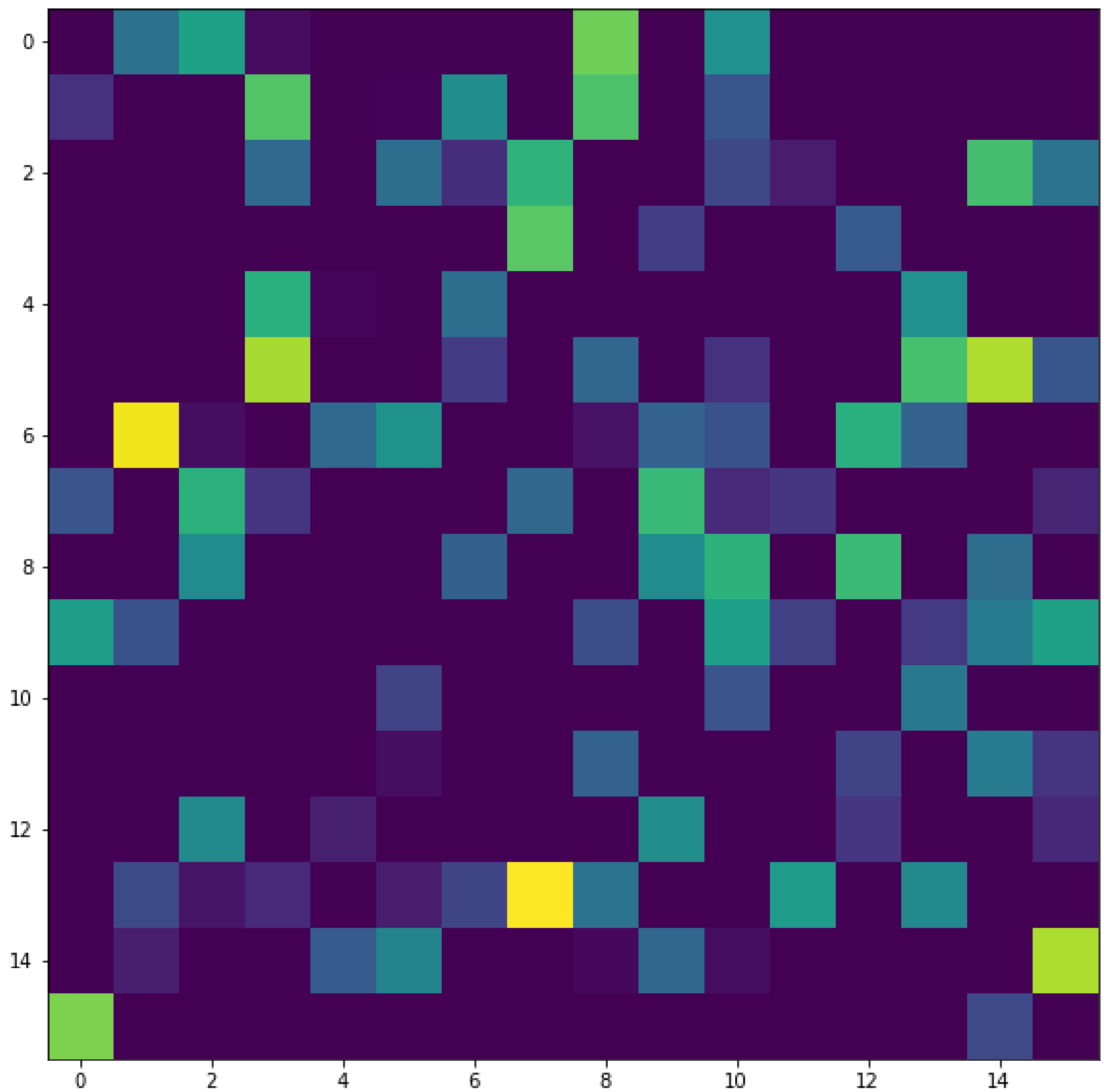
```
Average Hidden layer activations for 10 images for the hidden layer = 256
for the digit = 0 average value is = 44.89842224121094
```



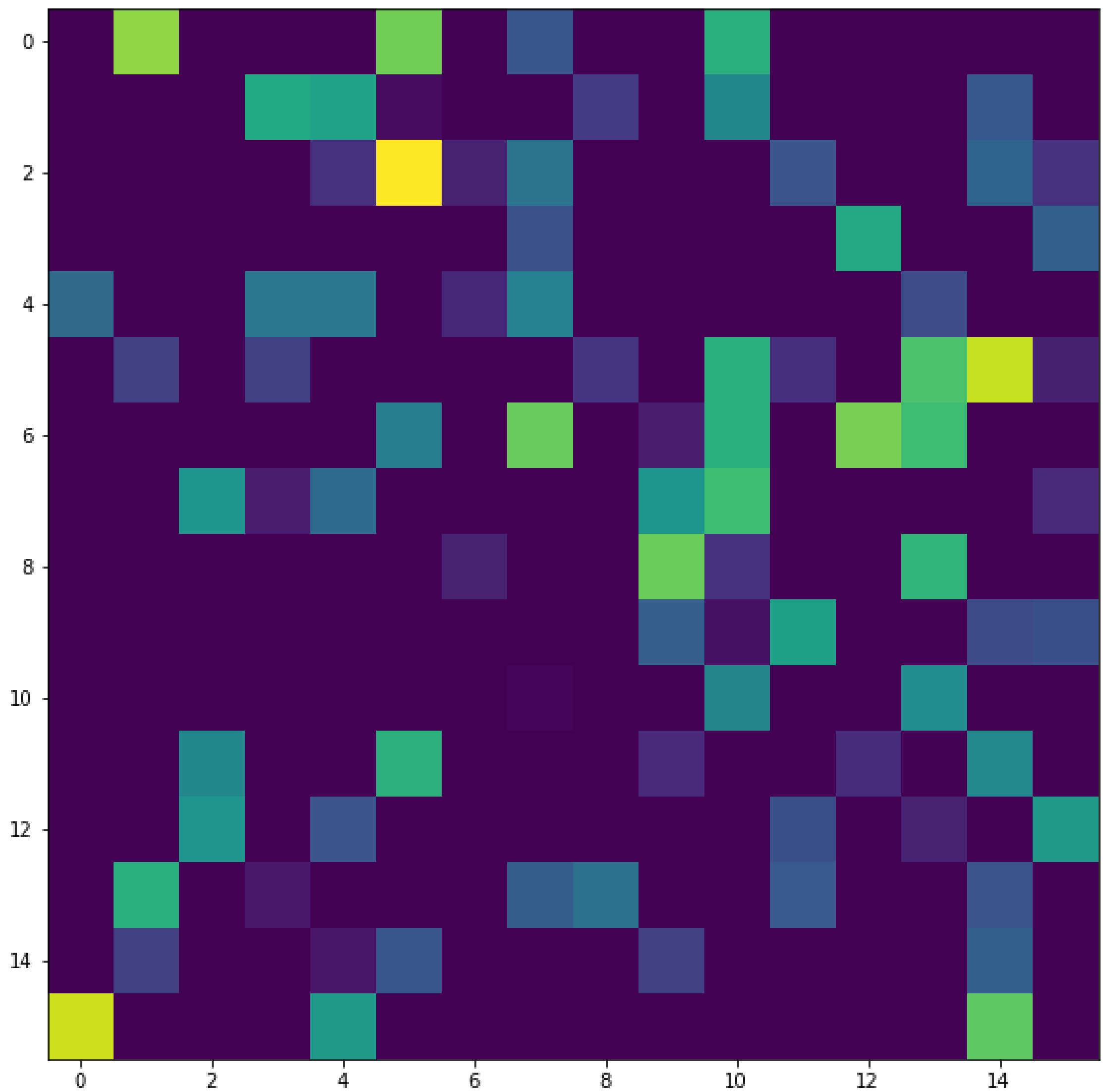
```
for the digit = 1 average value is = 41.207969665527344
```



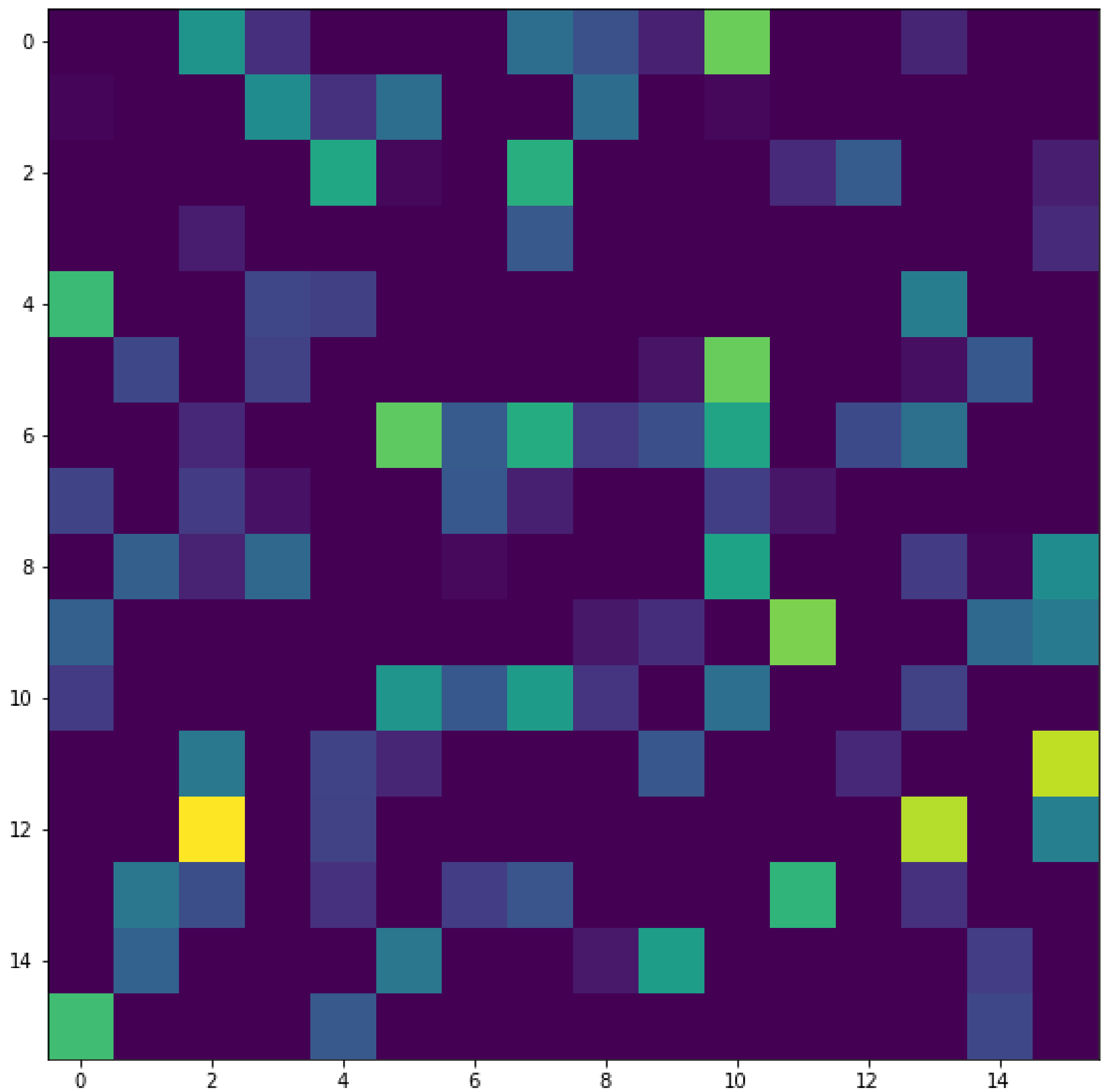

```
for the digit = 2 average value is = 60.484375
```



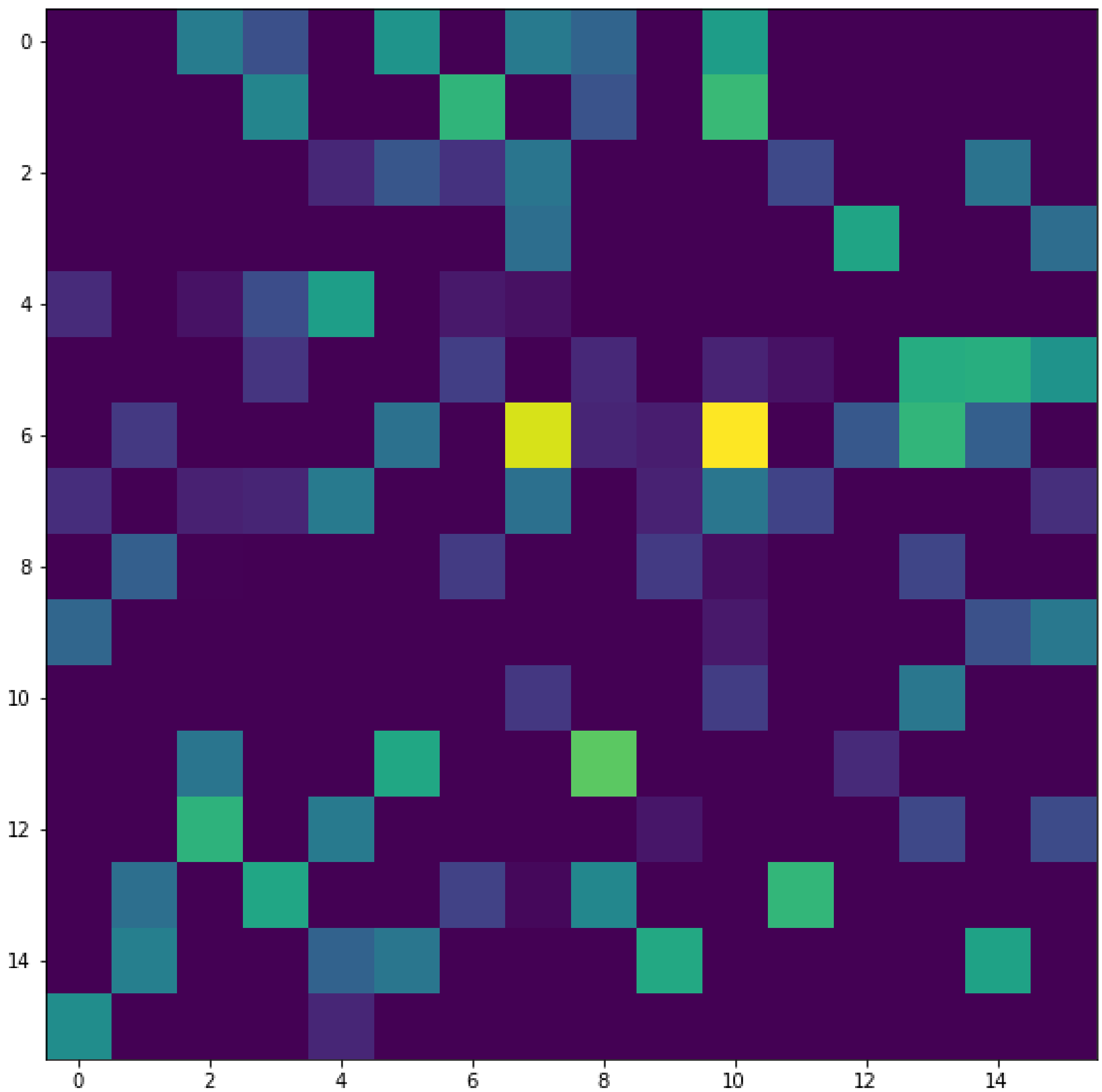

```
for the digit = 3 average value is = 46.20608139038086
```



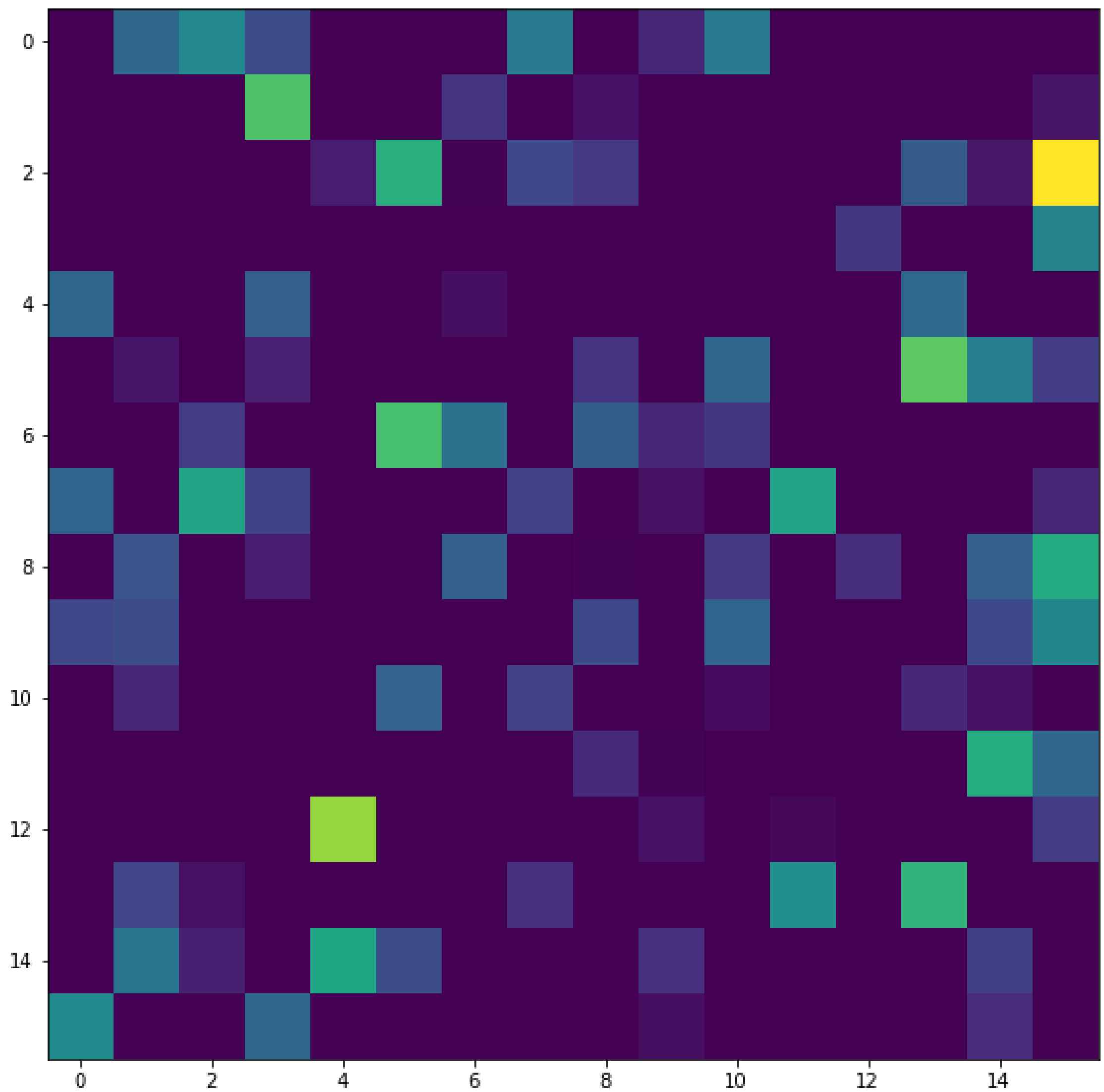

```
for the digit = 4 average value is = 48.670780181884766
```



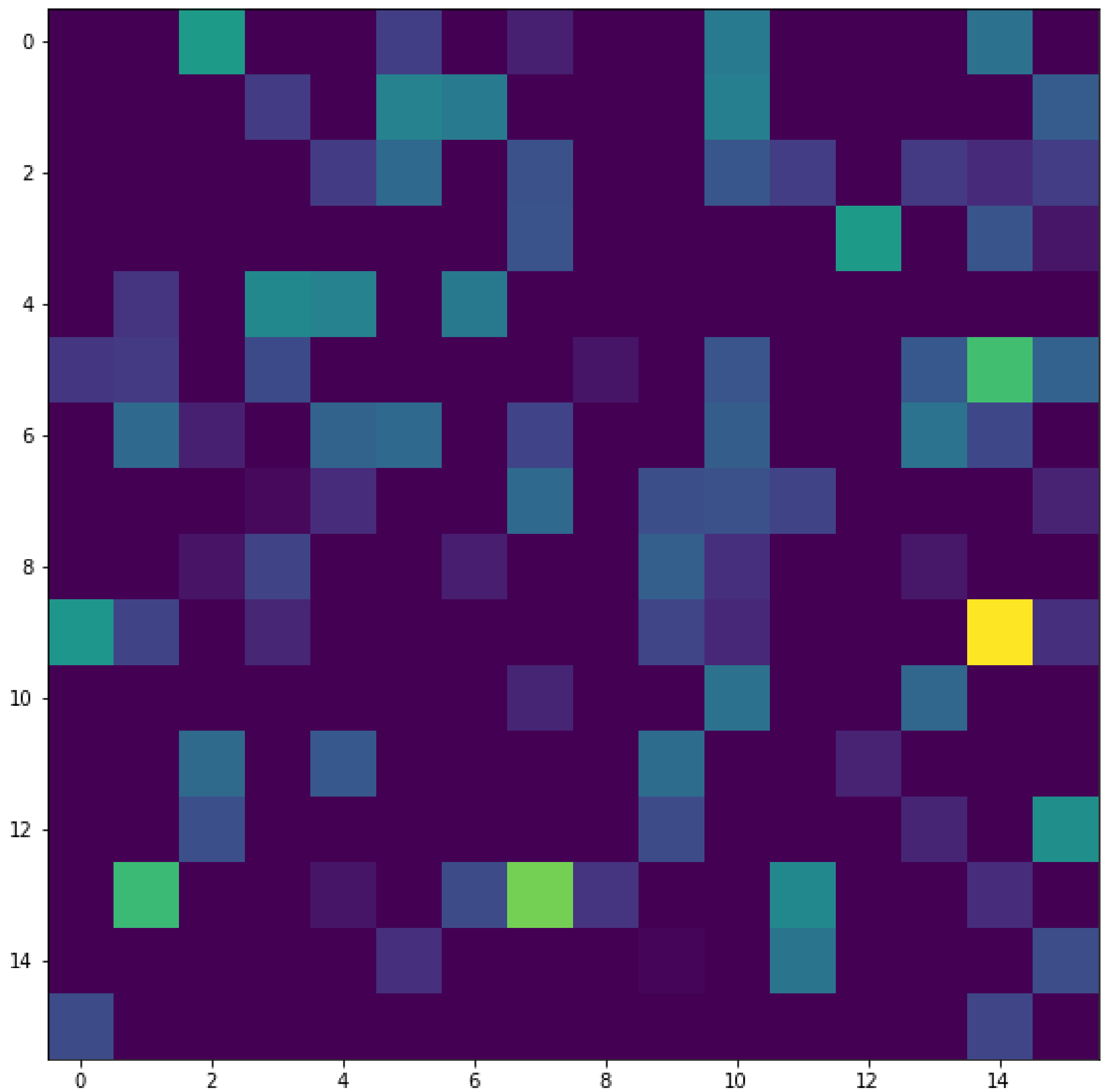
for the digit = 5 average value is = 45.45244216918945



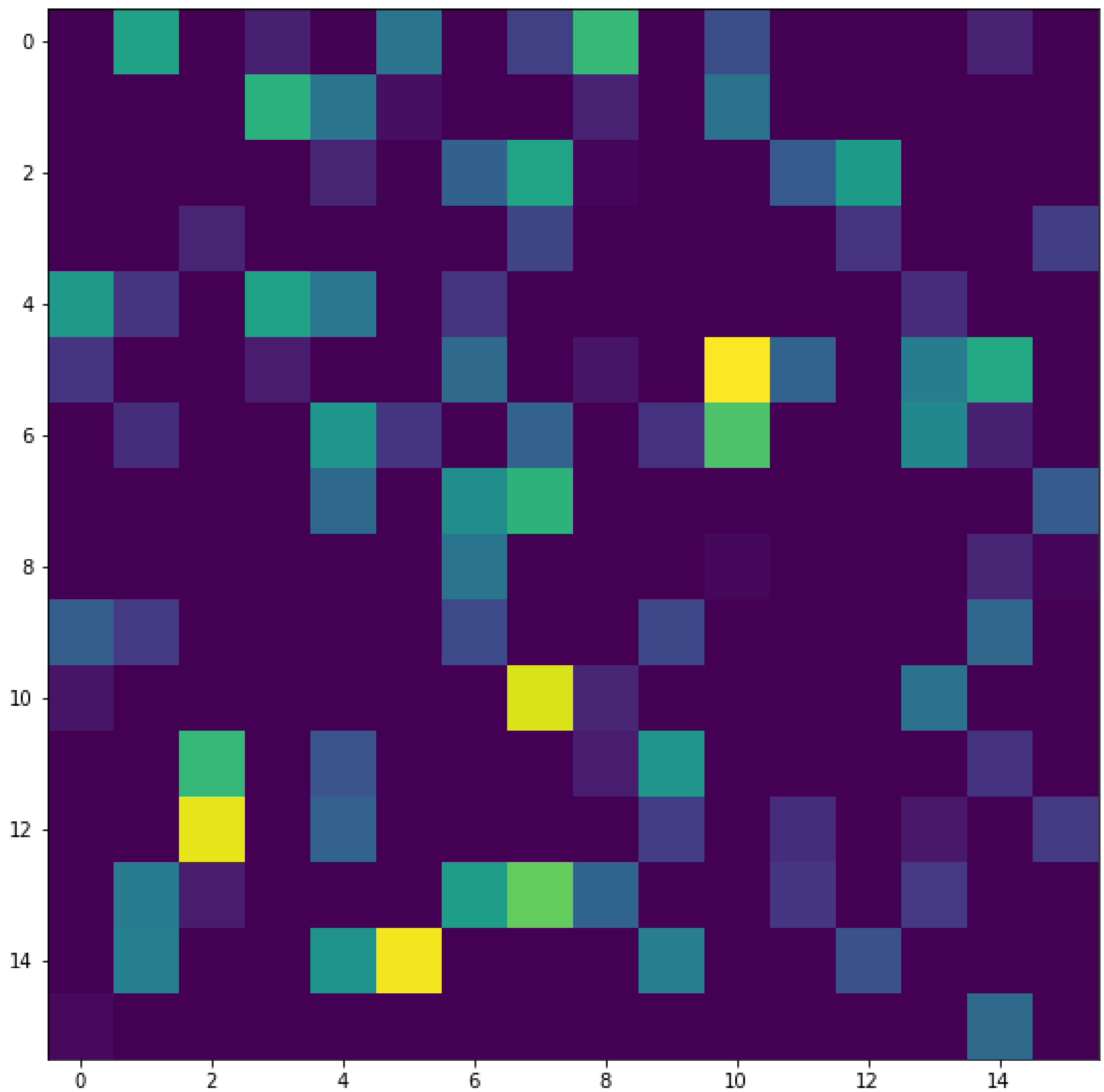
for the digit = 6 average value is = 45.25163650512695



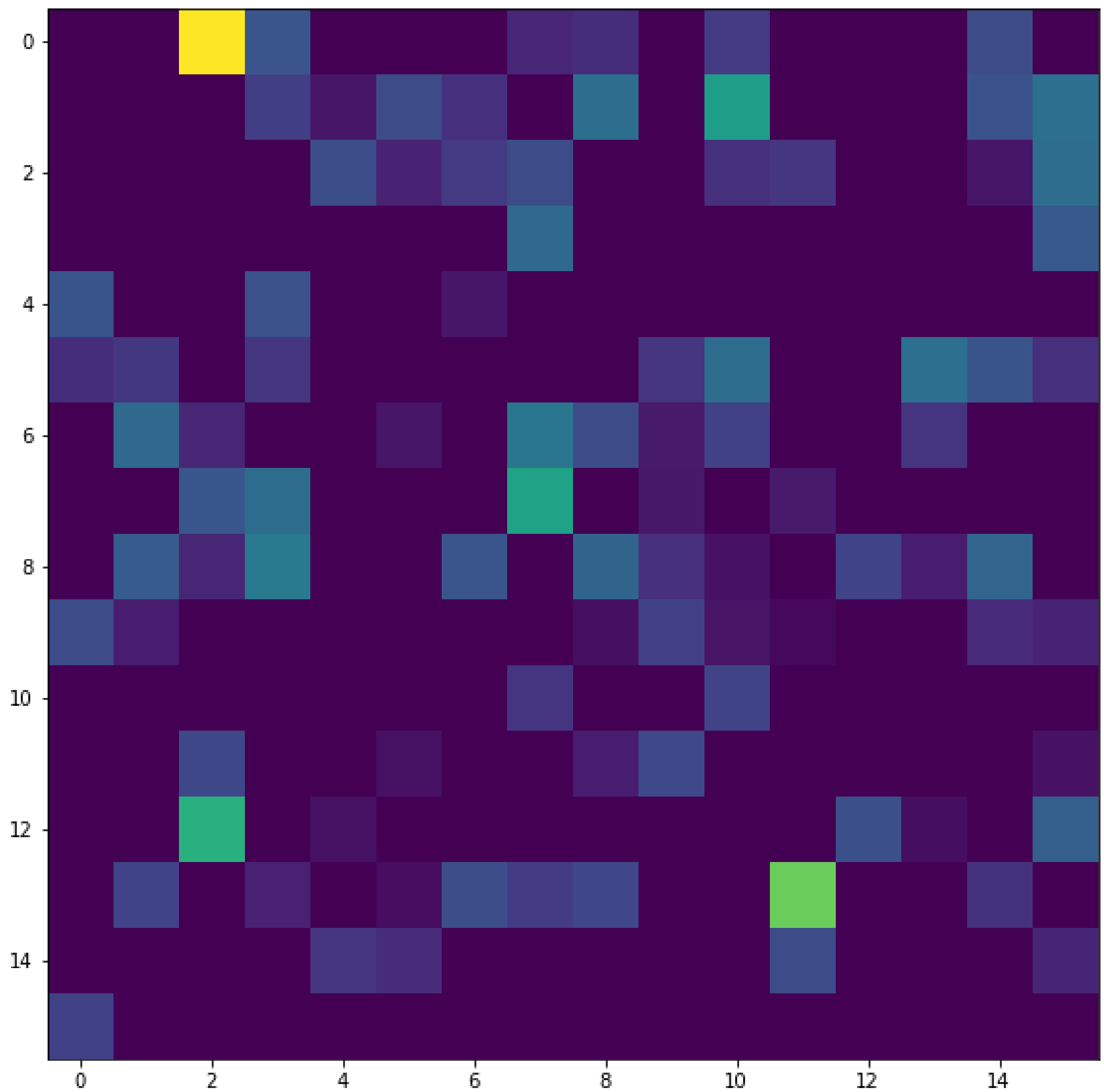
for the digit = 7 average value is = 54.53059768676758



for the digit = 8 average value is = 41.96870803833008



```
for the digit = 9 average value is = 36.63874816894531
```




```
Average of these values 46.530976104736325
```

4.1 Observations:

- We can observe that for hidden layer size = 256 we get the best results as compared to the previous ones.
- We observe the best results for size = 256, by comparing both visually and by the training and validation loss obtained

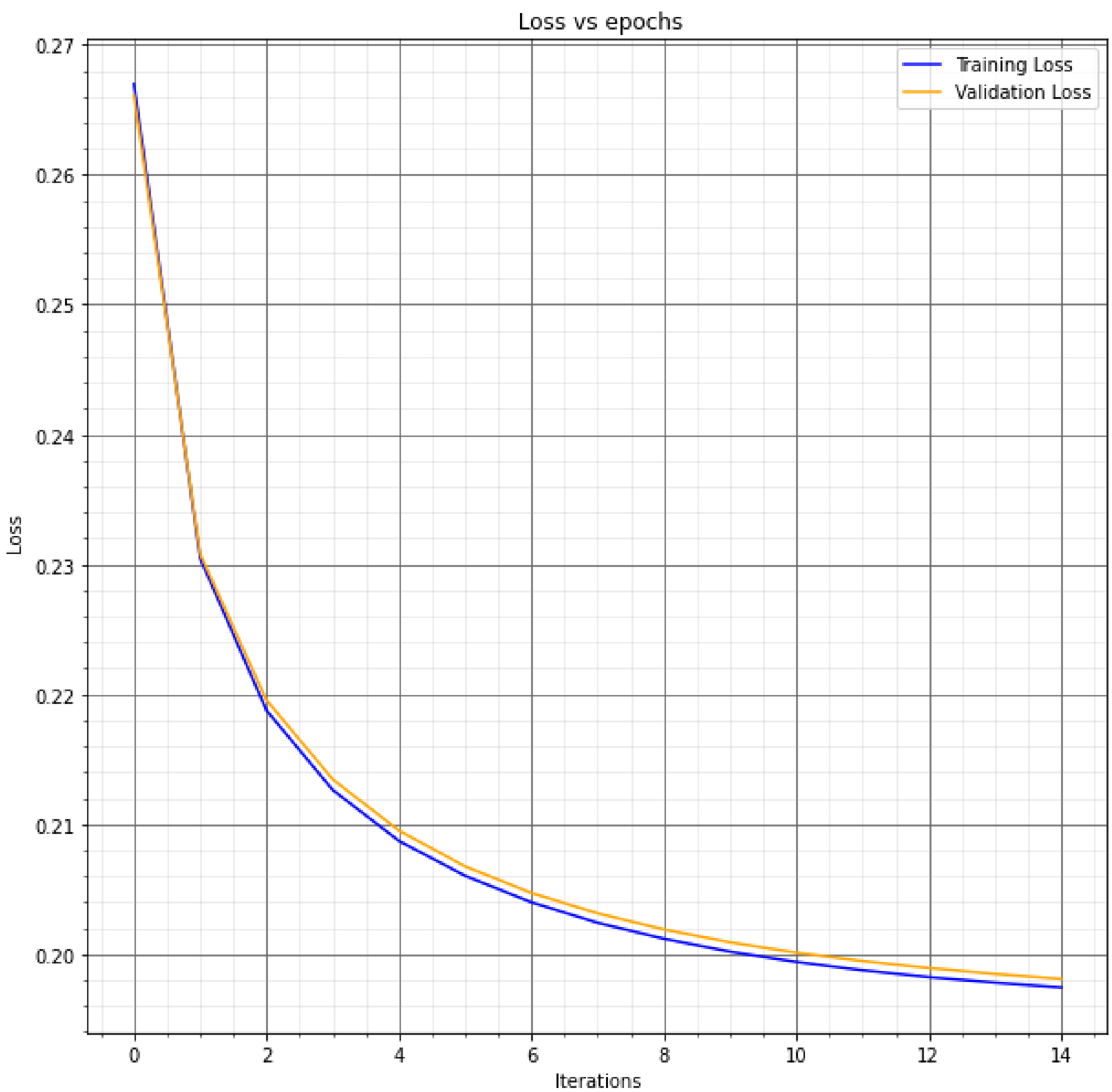
5 3. Sparse Autoencoders

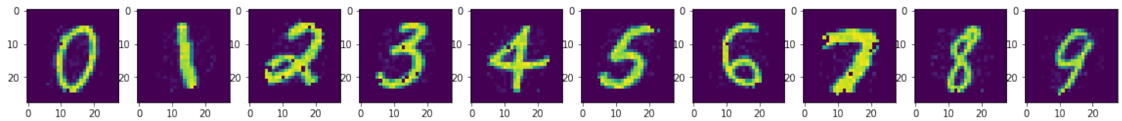
[48]:
```
class autoencoder_sparse(nn.Module):
  def __init__(self):
    super(autoencoder_sparse, self).__init__()
    self.encoder = nn.Sequential(nn.Linear(784, 1024), nn.ReLU())
    self.decoder = nn.Sequential(nn.Linear(1024, 784), nn.ReLU())

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

[52]:
```
model3 = autoencoder_sparse().to(device)
optimizer3 = optim.Adam(model3.parameters(), lr = 0.0003)
criterion = nn.MSELoss()
training_loss, val_loss = Train(model3, optimizer3, criterion, epochs, sparse =␣
↪True, l1_reg = 1e-7)
visualise_loss(training_loss, val_loss)
visualise_model(model3, data_ind, data)
```

```
Epochs: 1/15 ||| with Training Loss = 0.2669614553451538 ||| Validation Loss =
0.2661038041114807
Epochs: 2/15 ||| with Training Loss = 0.2304380089044571 ||| Validation Loss =
0.23078307509422302
Epochs: 3/15 ||| with Training Loss = 0.2187575101852417 ||| Validation Loss =
0.21954764425754547
Epochs: 4/15 ||| with Training Loss = 0.21263164281845093 ||| Validation Loss =
0.21345174312591553
Epochs: 5/15 ||| with Training Loss = 0.20871004462242126 ||| Validation Loss =
0.2095082700252533
Epochs: 6/15 ||| with Training Loss = 0.20602142810821533 ||| Validation Loss =
0.20674504339694977
Epochs: 7/15 ||| with Training Loss = 0.20398883521556854 ||| Validation Loss =
0.20471002161502838
Epochs: 8/15 ||| with Training Loss = 0.20240412652492523 ||| Validation Loss =
0.20313666760921478
Epochs: 9/15 ||| with Training Loss = 0.20117852091789246 ||| Validation Loss =
0.20189400017261505
Epochs: 10/15 ||| with Training Loss = 0.20018678903579712 ||| Validation Loss =
0.2009064108133316
Epochs: 11/15 ||| with Training Loss = 0.19939512014389038 ||| Validation Loss =
0.20010945200920105
Epochs: 12/15 ||| with Training Loss = 0.1987469643354416 ||| Validation Loss =
0.19946467876434326
Epochs: 13/15 ||| with Training Loss = 0.19822199642658234 ||| Validation Loss =
0.19892361760139465
```

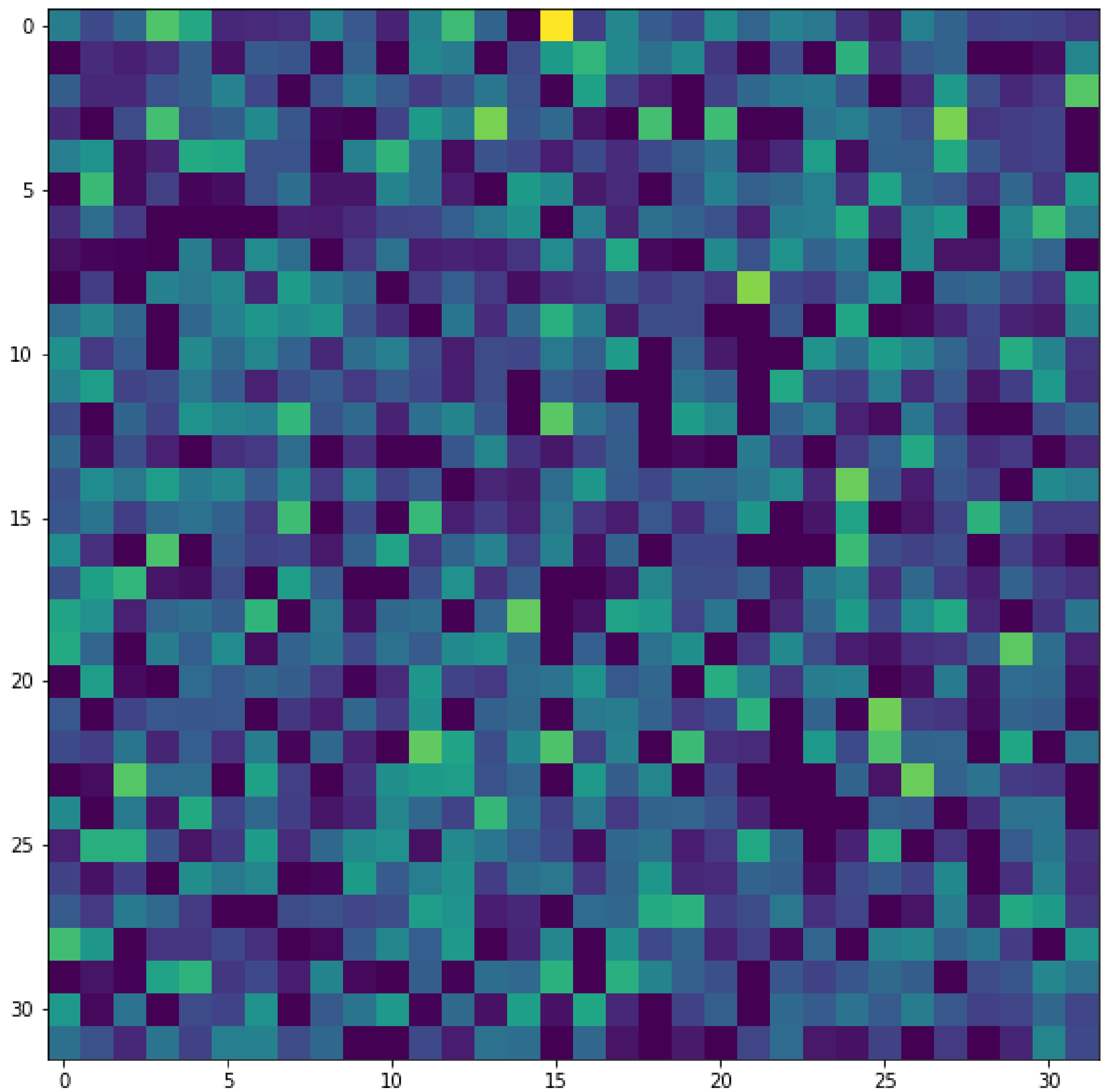
```
Epochs: 14/15 ||| with Training Loss = 0.19779588282108307 ||| Validation Loss =
0.19846349954605103
Epochs: 15/15 ||| with Training Loss = 0.19742344319820404 ||| Validation Loss =
0.19808447360992432
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



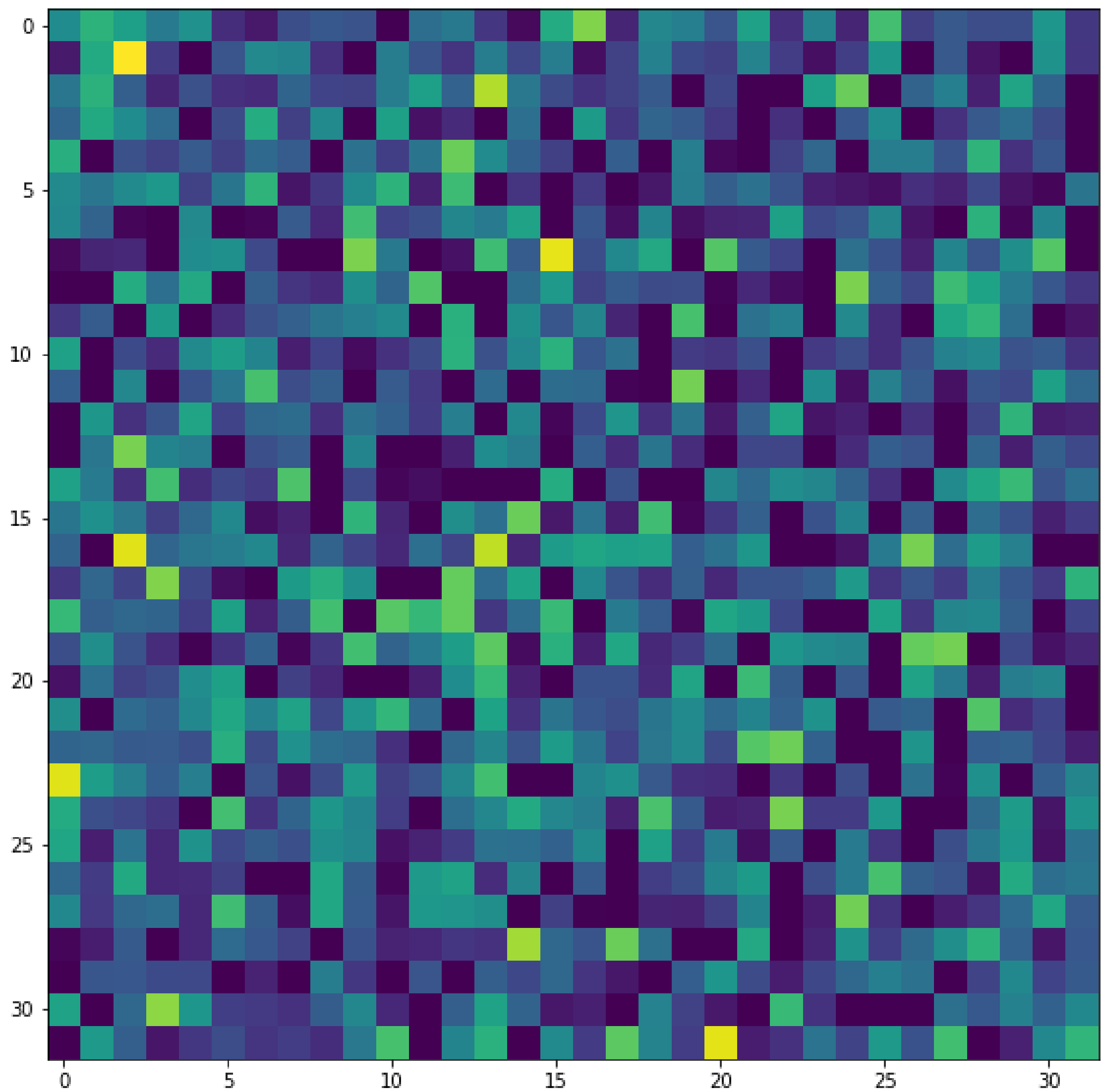


[54]:
```
print("Average Hidden layer activations for 10 images for the hidden layer =␣
↪256")
with torch.no_grad():
  sum = 0
  for i in range(10):

    output3, encoded3=model3.forward(data[data_ind[i]].reshape(1,784).float())
    avg = torch.norm(encoded3, p=1)/1024.0
    print(f'for the digit = {i} average value is = {avg.detach().numpy()}')
    sum+=avg.detach().numpy()
    plt.imshow(encoded3.reshape(32, 32))
    plt.show()
  print("Average of these values",sum/10.0)
```

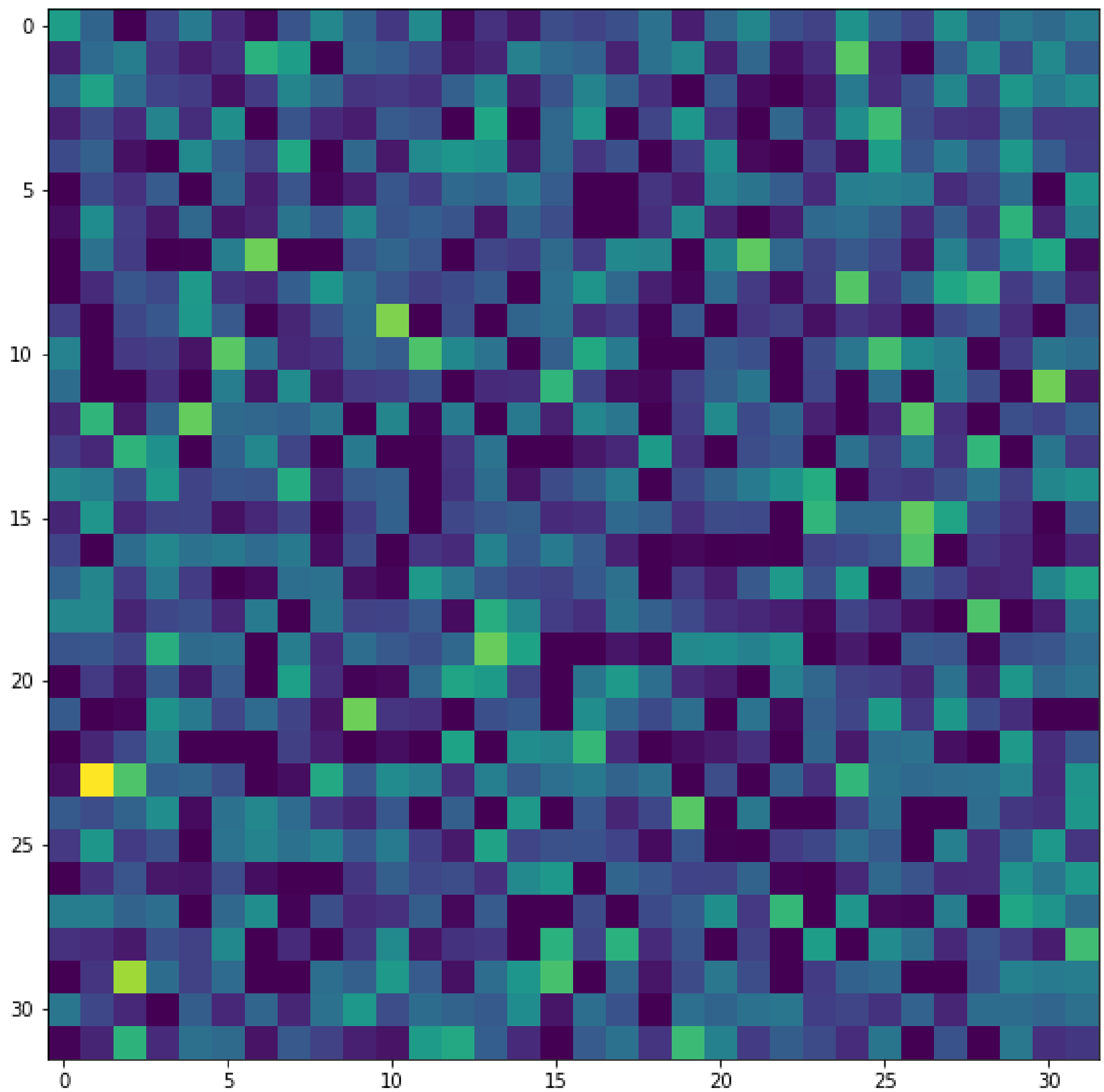
```
Average Hidden layer activations for 10 images for the hidden layer = 256
for the digit = 0 average value is = 76.5195083618164
```



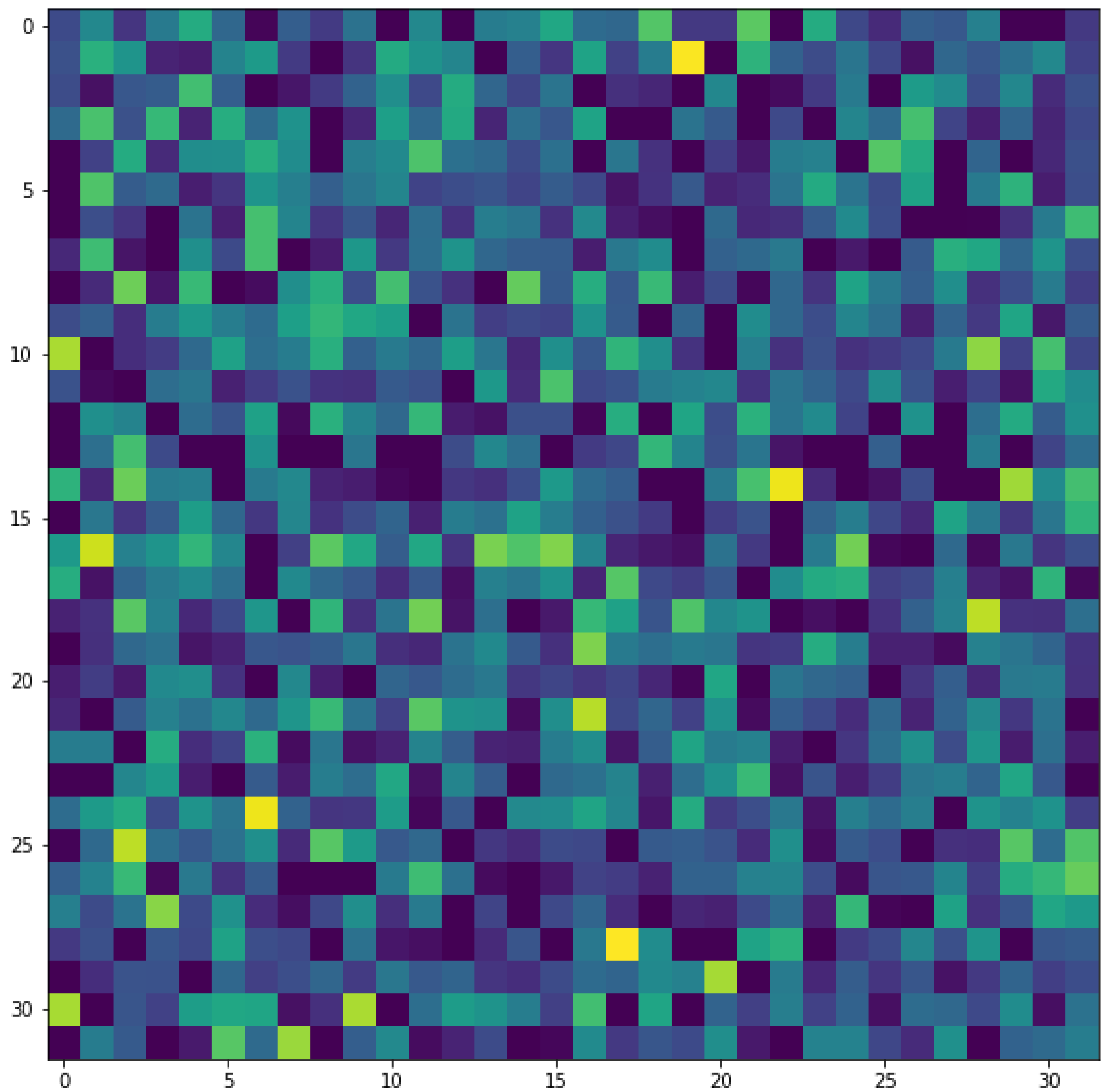

```
for the digit = 1 average value is = 65.42584991455078
```



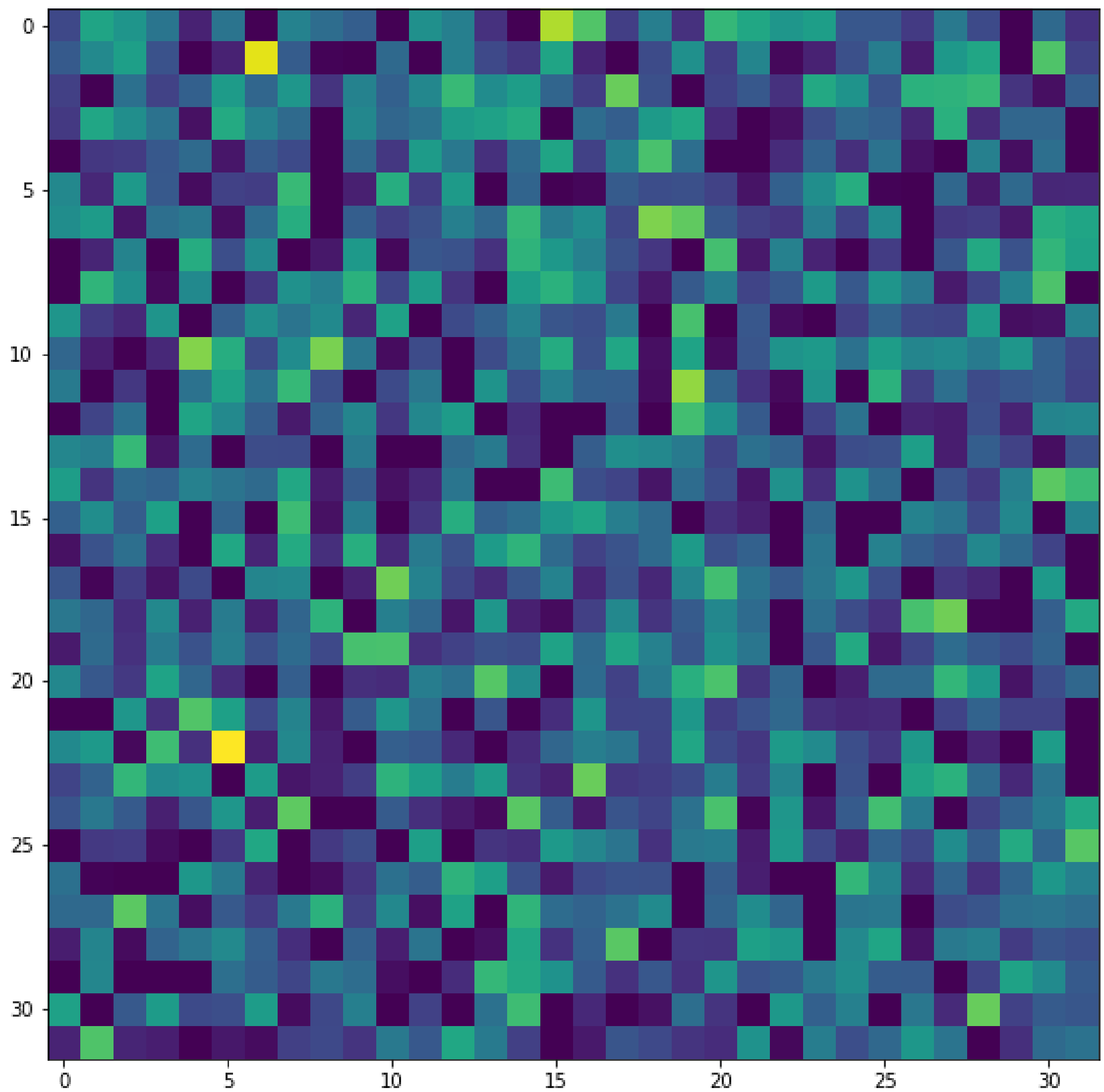
for the digit = 2 average value is = 93.7727279663086



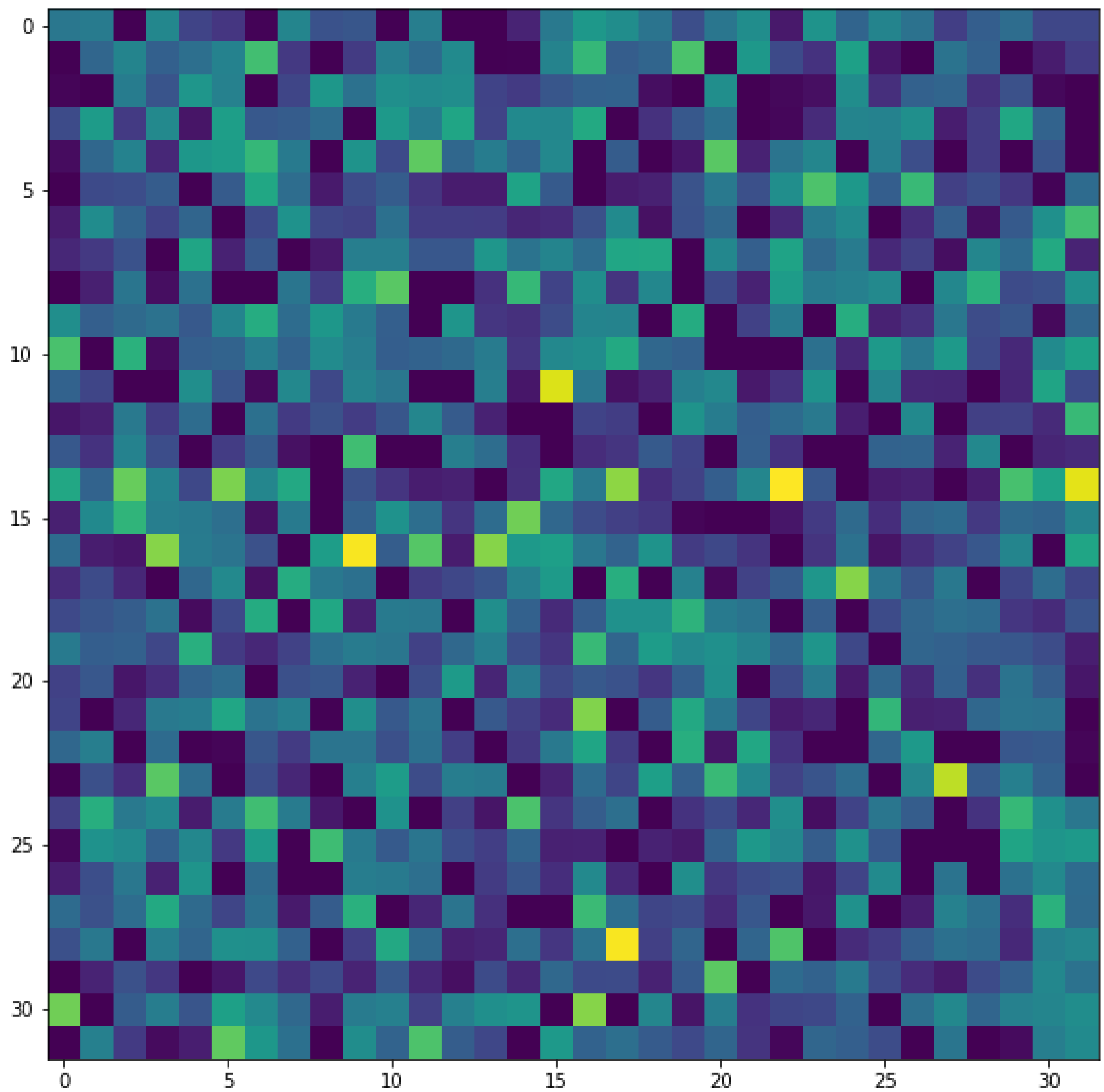
for the digit = 3 average value is = 85.11482238769531



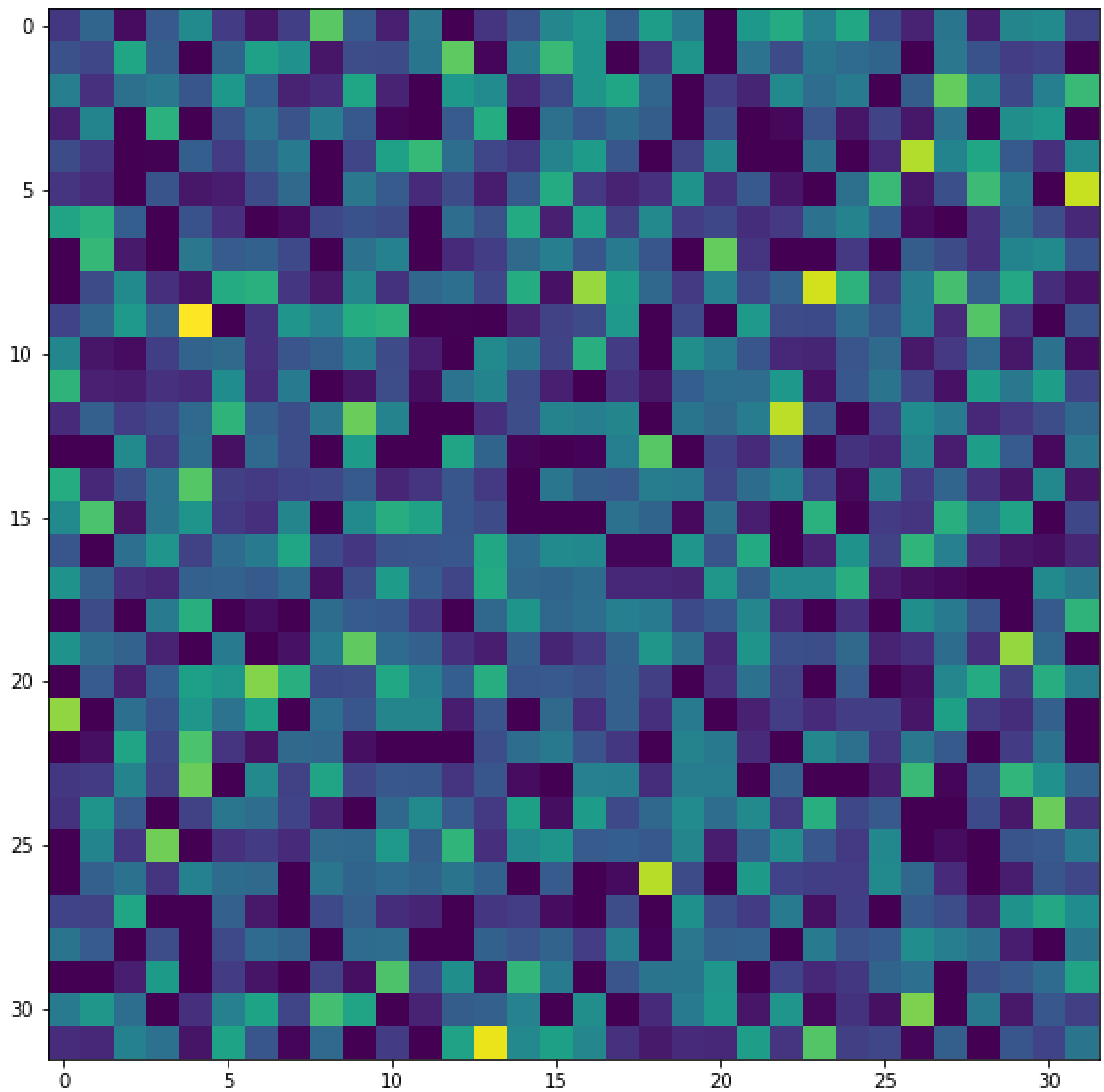
for the digit = 4 average value is = 86.35318756103516



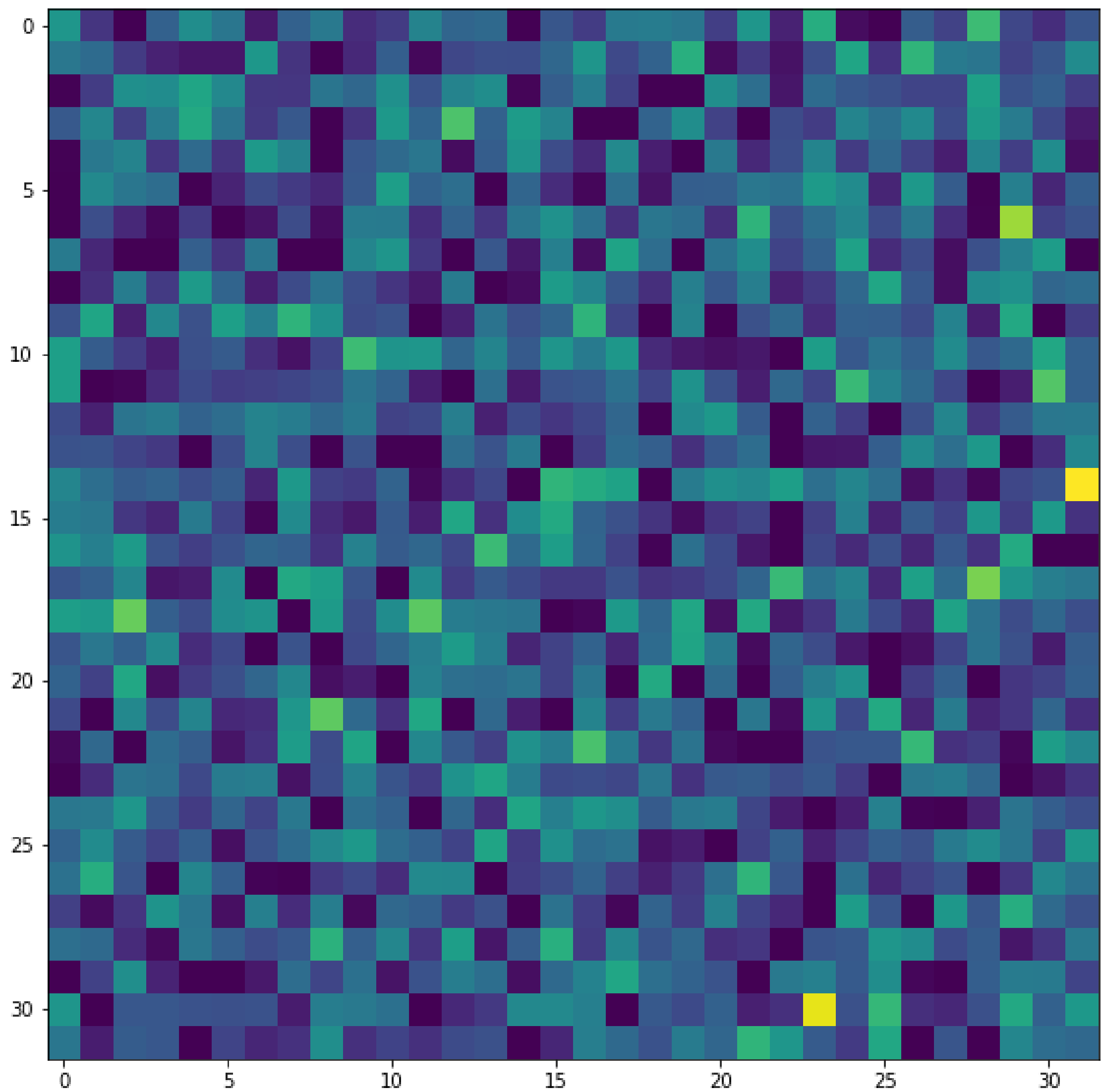

```
for the digit = 5 average value is = 81.92125701904297
```



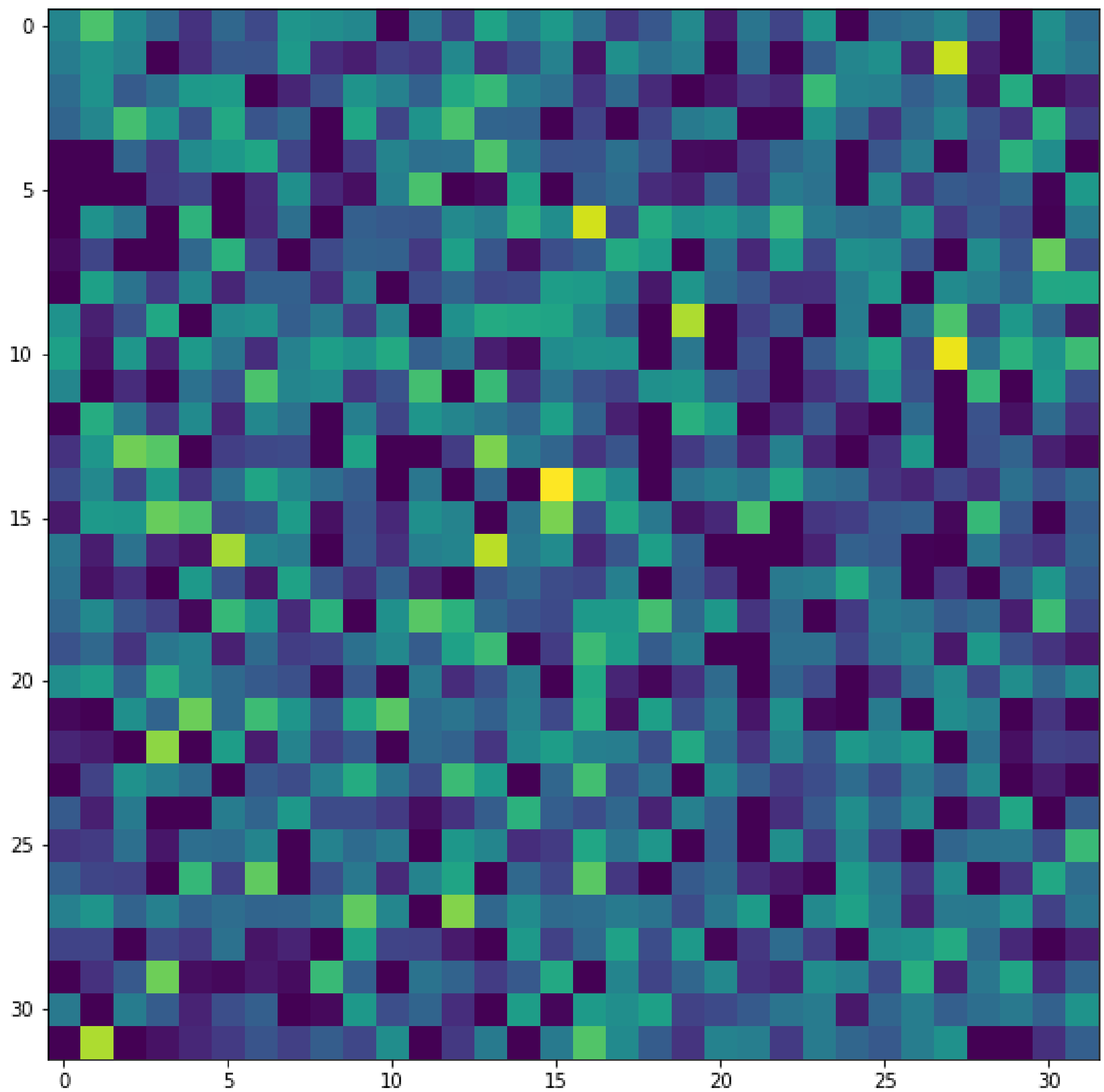
```
for the digit = 6 average value is = 76.35543823242188
```



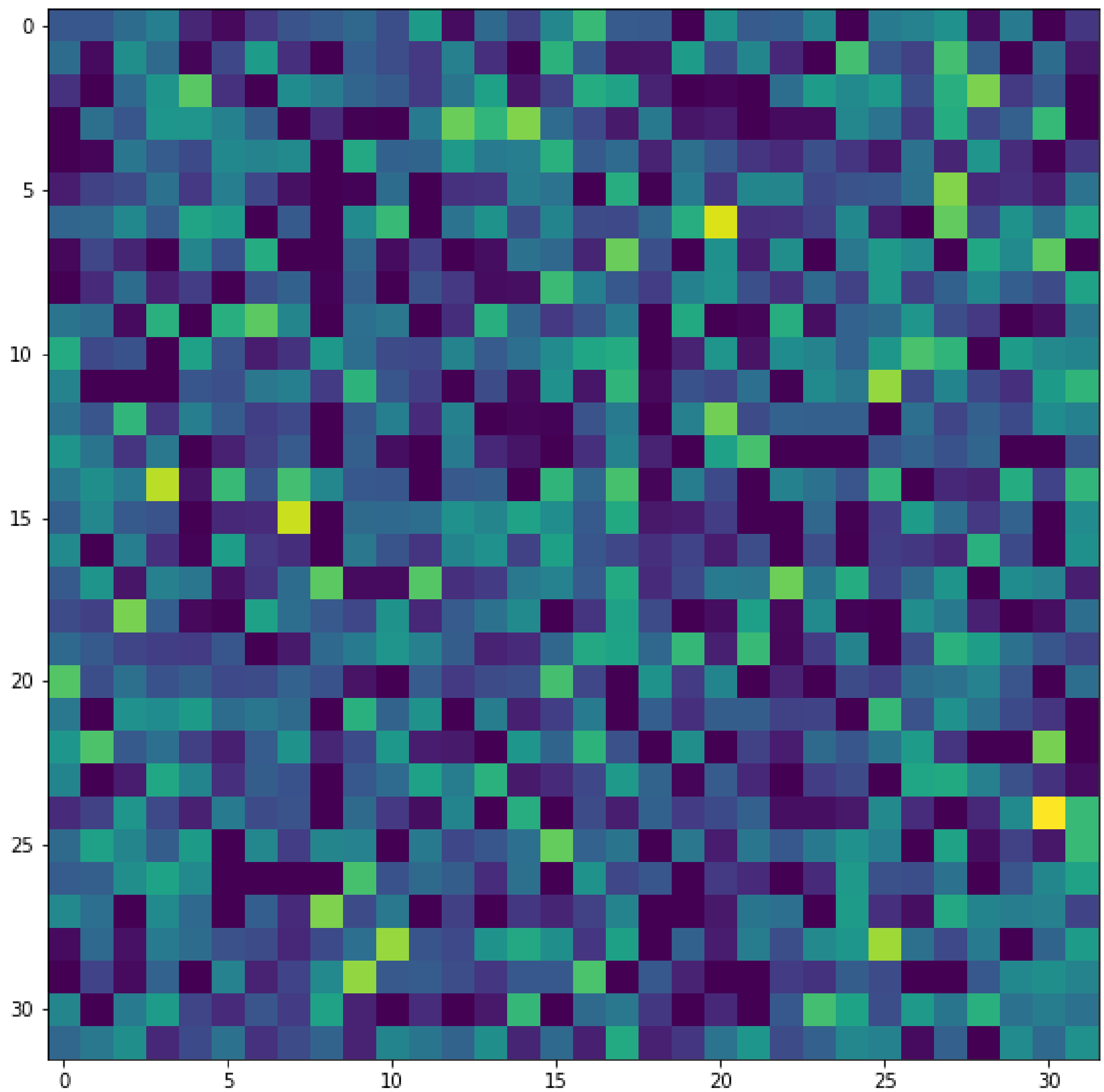

```
for the digit = 7 average value is = 111.32823181152344
```




```
for the digit = 8 average value is = 78.93204498291016
```



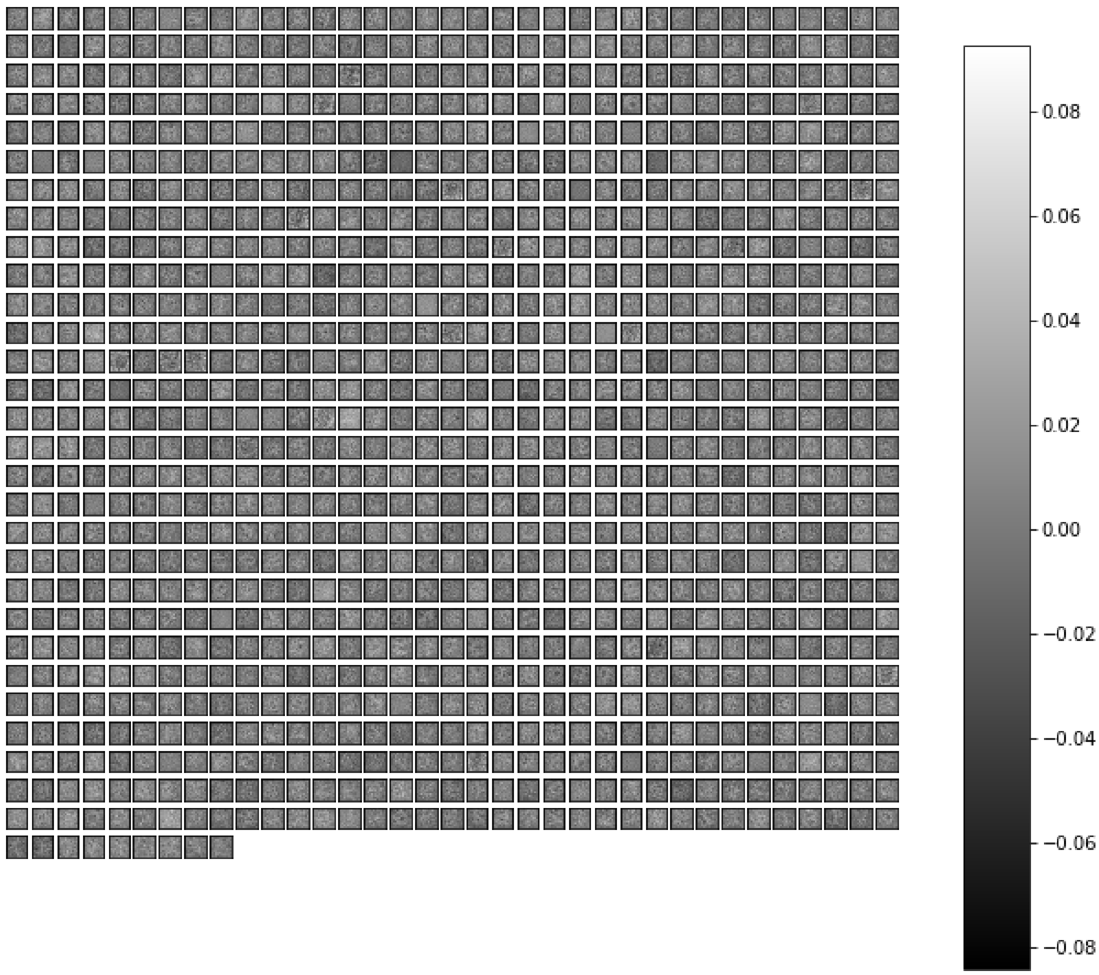
for the digit = 9 average value is = 65.29202270507812




```
Average of these values 82.10150909423828
```

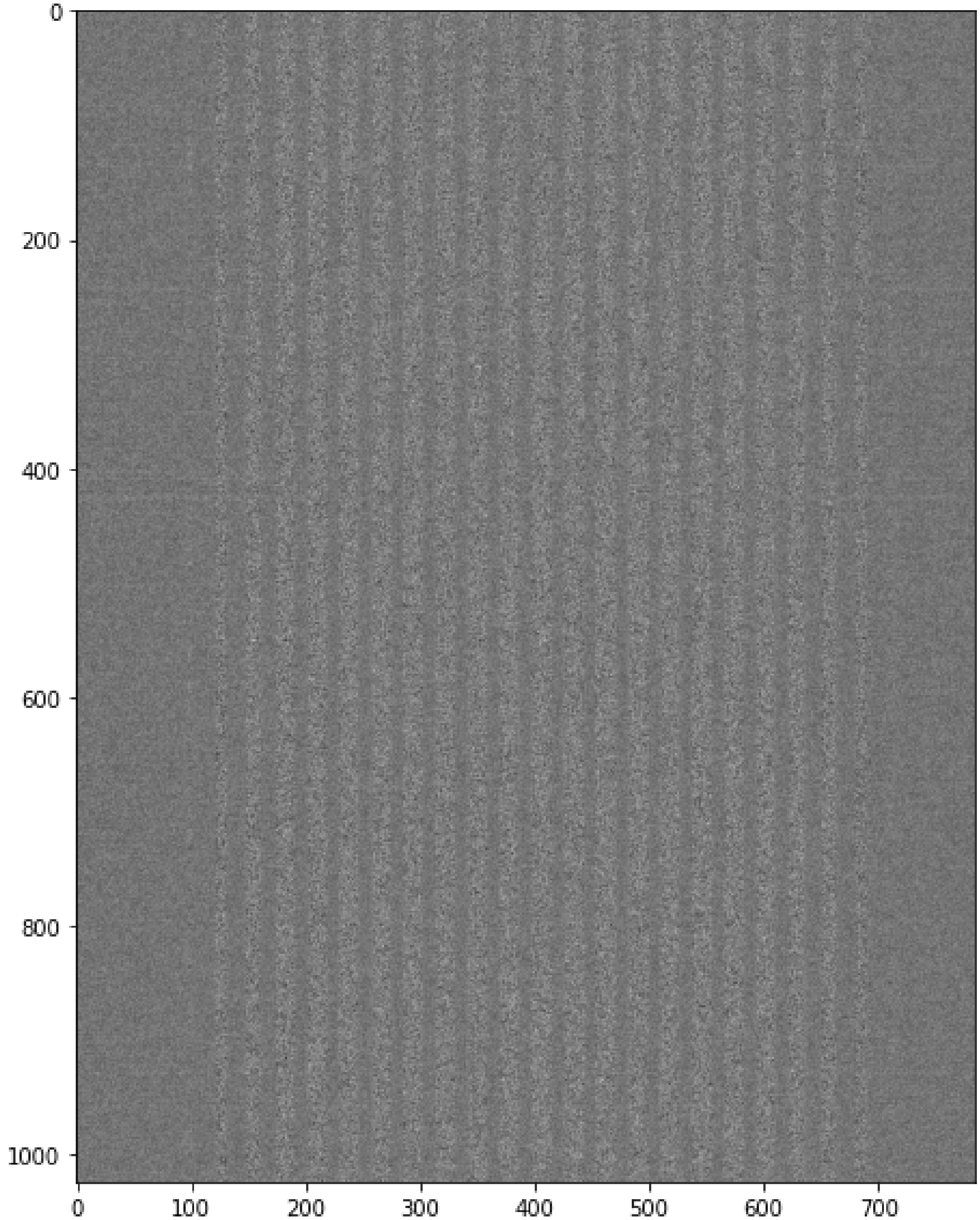
[55]:
```
ix=1
fig,ax=plt.subplots()
for i in range(len(model3.state_dict()['encoder.0.weight'])):
  ax=plt.subplot(35,35,ix)
  ax.set_xticks([])
  ax.set_yticks([])
  im=ax.imshow(model3.state_dict()['encoder.0.weight'][i].
 ↪reshape(28,28),cmap='gray')
  ix+=1

fig.subplots_adjust(right=0.8)
```

```
cbar_ax = fig.add_axes([0.85, 0.15, 0.05, 0.7])
fig.colorbar(im, cax=cbar_ax)
plt.show()
```



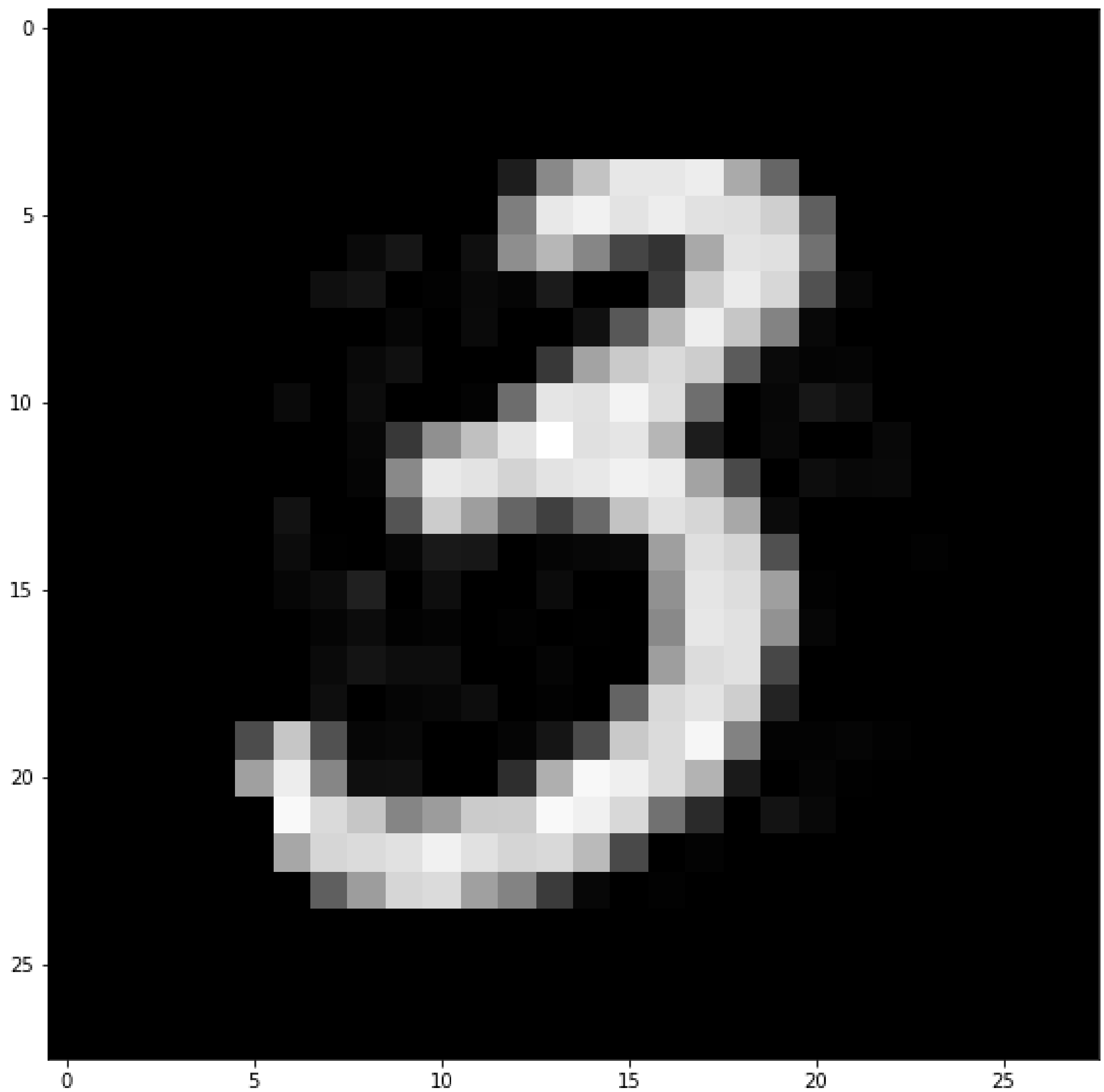
[57]:
```
plt.imshow(model3.state_dict()['encoder.0.weight'],cmap='gray')
```

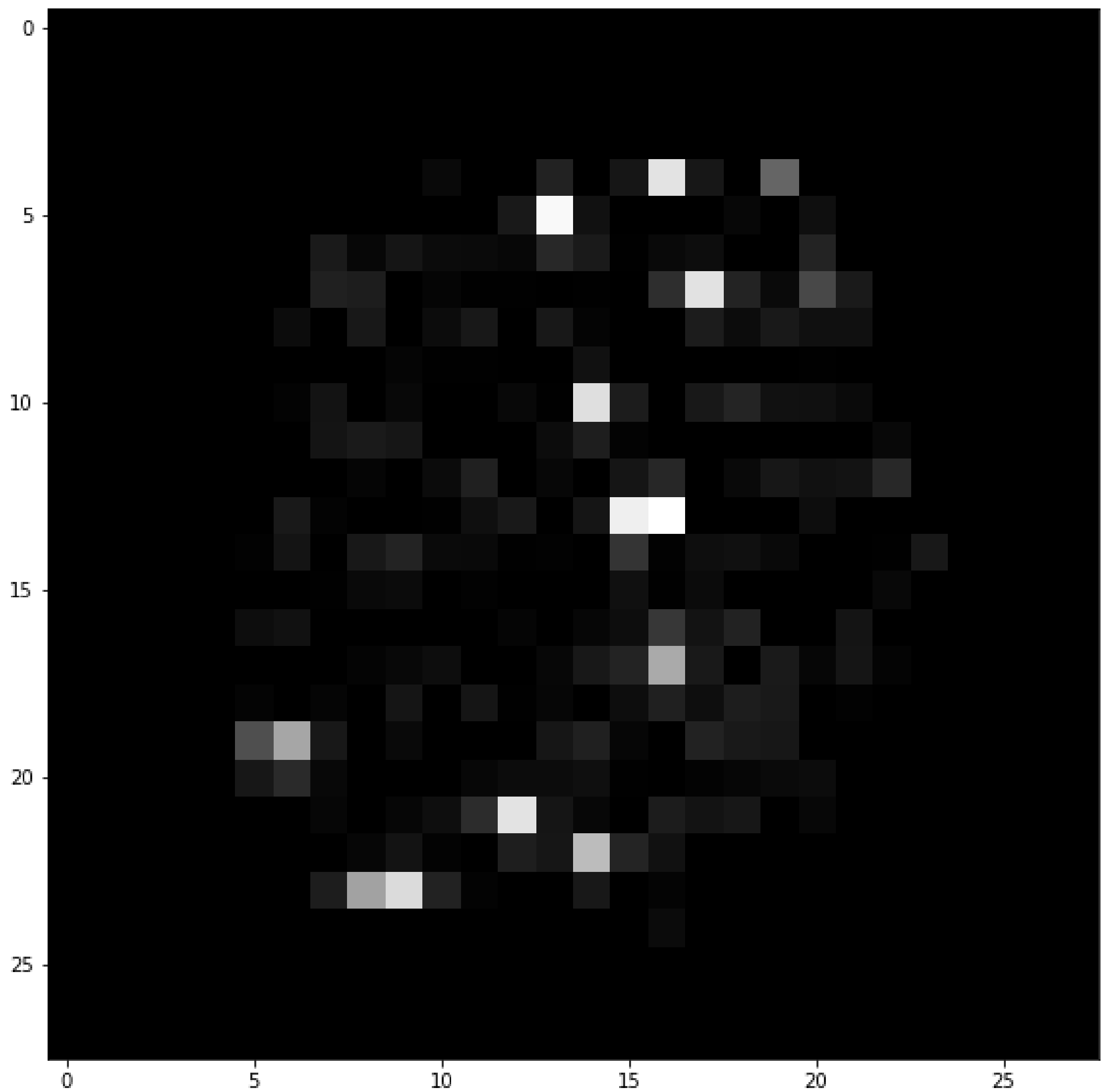
[57]: <matplotlib.image.AxesImage at 0x7fef0ac53d90>



[71]:
```
with torch.no_grad():
  a=random.sample(range(0,784),int(0.9*784))
  X=data[data_ind[3]].clone()
  X=X.reshape(1,784)
  X[0][a]=0
```

```
output1=model3(data[data_ind[3]].reshape(1,784))
plt.imshow(output1[0].reshape(28,28),cmap='gray')
plt.show()

output2=model3(X.reshape(1,784))
plt.imshow(output2[0].reshape(28,28),cmap='gray')
plt.show()
```





6 4. Denoising Encoder

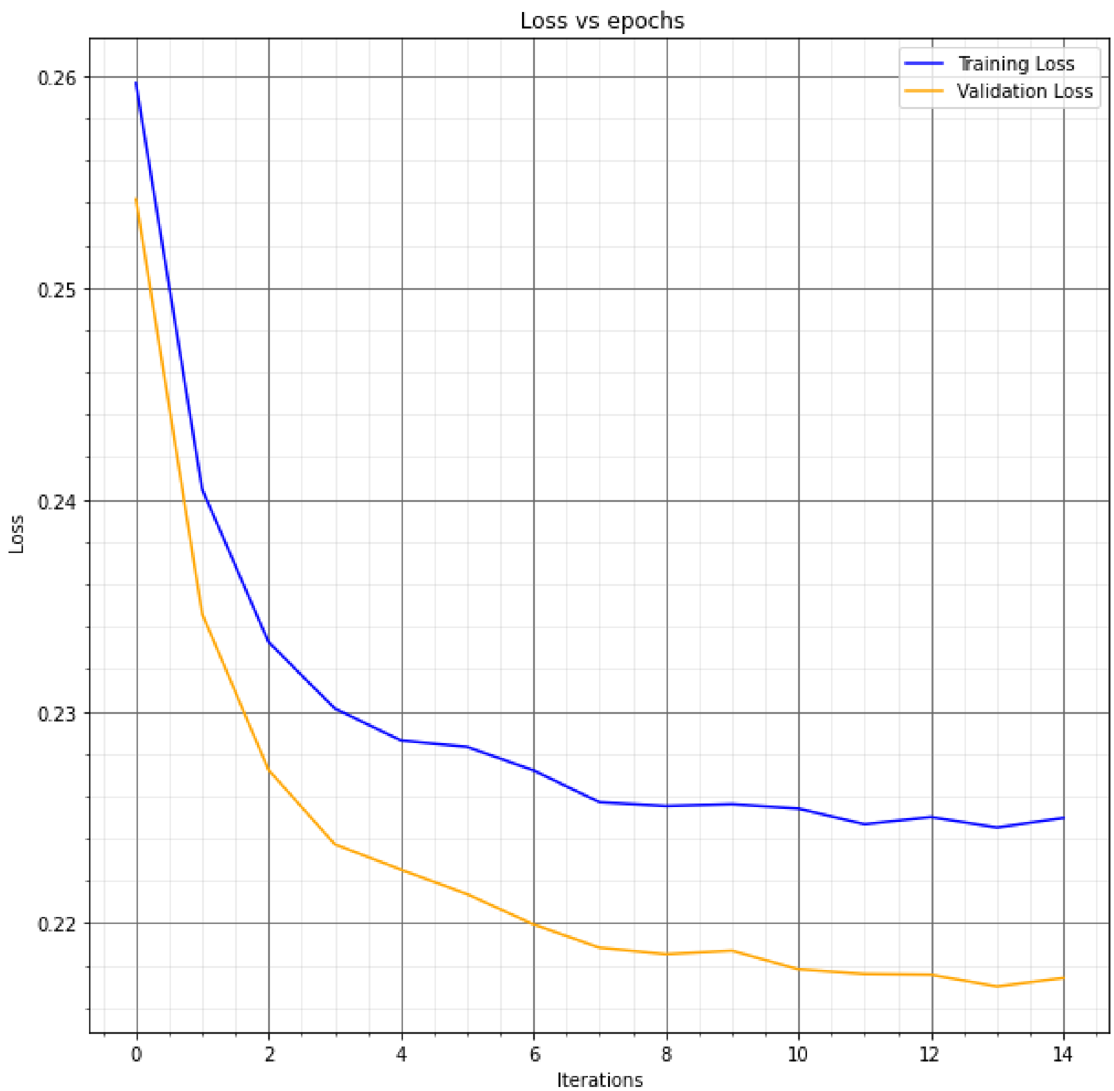
[72]:
```
def Add_Noise(image, noise_val = 0.3):
  noise = torch.randn(image.size())*noise_val
  noisy_image = image + noise
  return noisy_image
```

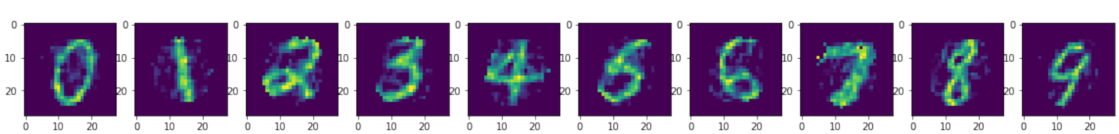
6.1 First using autoencoder in question 2

[73]:
```
noise_val = [0.3, 0.5, 0.8, 0.9]
for i in range(4):
  print(f'\n for noise value = {noise_val[i]}\n')
  model4 = autoencoder_hidden(256).to(device)
  optimizer4 = optim.Adam(model4.parameters(), lr = 0.003)
  criterion = nn.MSELoss()
  training_loss, val_loss = Train(model4, optimizer4, criterion, epochs,␣
↪denoise = True, noise_val = noise_val[i])

  visualise_loss(training_loss, val_loss)
  visualise_model(model4, data_ind, data)
```

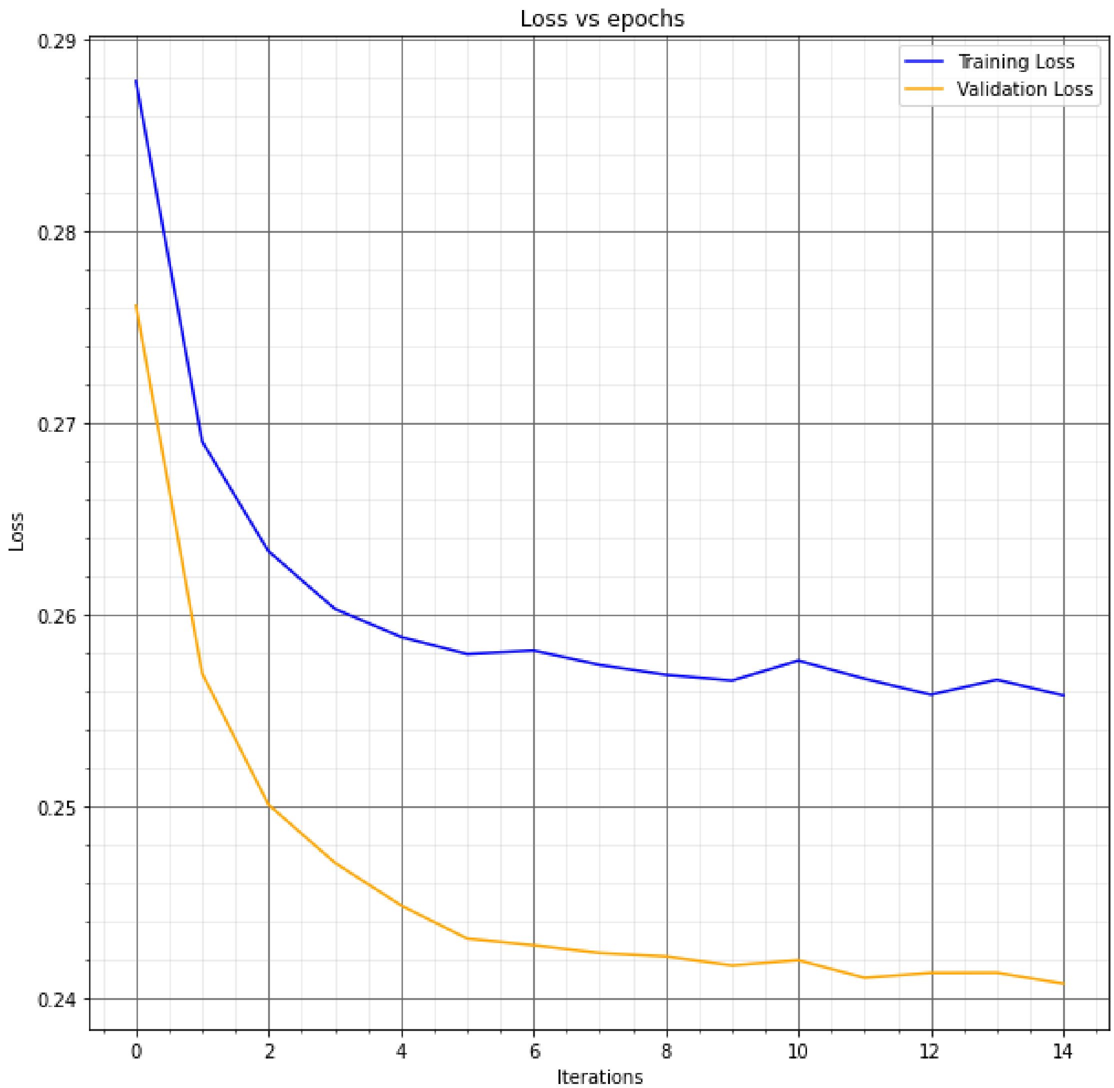
```
 for noise value = 0.3

Epochs: 1/15 ||| with Training Loss = 0.25964343547821045 ||| Validation Loss =
0.2541666626930237
Epochs: 2/15 ||| with Training Loss = 0.2404690831899643 ||| Validation Loss =
0.23458996415138245
Epochs: 3/15 ||| with Training Loss = 0.2332722246646881 ||| Validation Loss =
0.2272350937128067
Epochs: 4/15 ||| with Training Loss = 0.2301328331232071 ||| Validation Loss =
0.22372418642044067
Epochs: 5/15 ||| with Training Loss = 0.2286226749420166 ||| Validation Loss =
0.22252203524112701
Epochs: 6/15 ||| with Training Loss = 0.22831477224826813 ||| Validation Loss =
0.22136977314949036
Epochs: 7/15 ||| with Training Loss = 0.22721241414546967 ||| Validation Loss =
0.2199418842792511
Epochs: 8/15 ||| with Training Loss = 0.2257051020860672 ||| Validation Loss =
0.2188408076763153
Epochs: 9/15 ||| with Training Loss = 0.22552256286144257 ||| Validation Loss =
0.2185373455286026
Epochs: 10/15 ||| with Training Loss = 0.22561019659042358 ||| Validation Loss =
0.21870316565036774
Epochs: 11/15 ||| with Training Loss = 0.22540491819381714 ||| Validation Loss =
0.21781963109970093
Epochs: 12/15 ||| with Training Loss = 0.22466717660427094 ||| Validation Loss =
0.21759797632694244
Epochs: 13/15 ||| with Training Loss = 0.2250085324048996 ||| Validation Loss =
0.21756429970264435
Epochs: 14/15 ||| with Training Loss = 0.22451135516166687 ||| Validation Loss =
0.21700796484947205
Epochs: 15/15 ||| with Training Loss = 0.22496536374092102 ||| Validation Loss =
0.21740809082984924
```

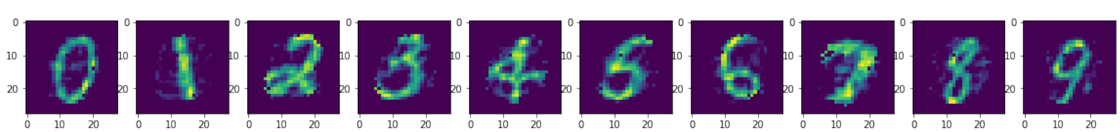
```
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```





```
 for noise value = 0.5

Epochs: 1/15 ||| with Training Loss = 0.2878321409225464 ||| Validation Loss =
0.27613434195518494
Epochs: 2/15 ||| with Training Loss = 0.26901760697364807 ||| Validation Loss =
0.25691479444503784
Epochs: 3/15 ||| with Training Loss = 0.26329925656318665 ||| Validation Loss =
0.25007352232933044
Epochs: 4/15 ||| with Training Loss = 0.2602956295013428 ||| Validation Loss =
0.24704982340335846
Epochs: 5/15 ||| with Training Loss = 0.25883233547210693 ||| Validation Loss =
0.244818314909935
Epochs: 6/15 ||| with Training Loss = 0.25793299078941345 ||| Validation Loss =
0.24308356642723083
Epochs: 7/15 ||| with Training Loss = 0.25811874866485596 ||| Validation Loss =
0.24273598194122314
Epochs: 8/15 ||| with Training Loss = 0.25736838579177856 ||| Validation Loss =
0.24233466386795044
Epochs: 9/15 ||| with Training Loss = 0.2568536698818207 ||| Validation Loss =
0.24215535819530487
Epochs: 10/15 ||| with Training Loss = 0.2565525472164154 ||| Validation Loss =
0.24168632924556732
Epochs: 11/15 ||| with Training Loss = 0.25758954882621765 ||| Validation Loss =
0.24195854365825653
Epochs: 12/15 ||| with Training Loss = 0.25664934515953064 ||| Validation Loss =
0.24104103446006775
Epochs: 13/15 ||| with Training Loss = 0.2558158338069916 ||| Validation Loss =
0.2412956953048706
Epochs: 14/15 ||| with Training Loss = 0.2565968334674835 ||| Validation Loss =
0.241303950548172
Epochs: 15/15 ||| with Training Loss = 0.25577935576438904 ||| Validation Loss =
0.24073509871959686
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```

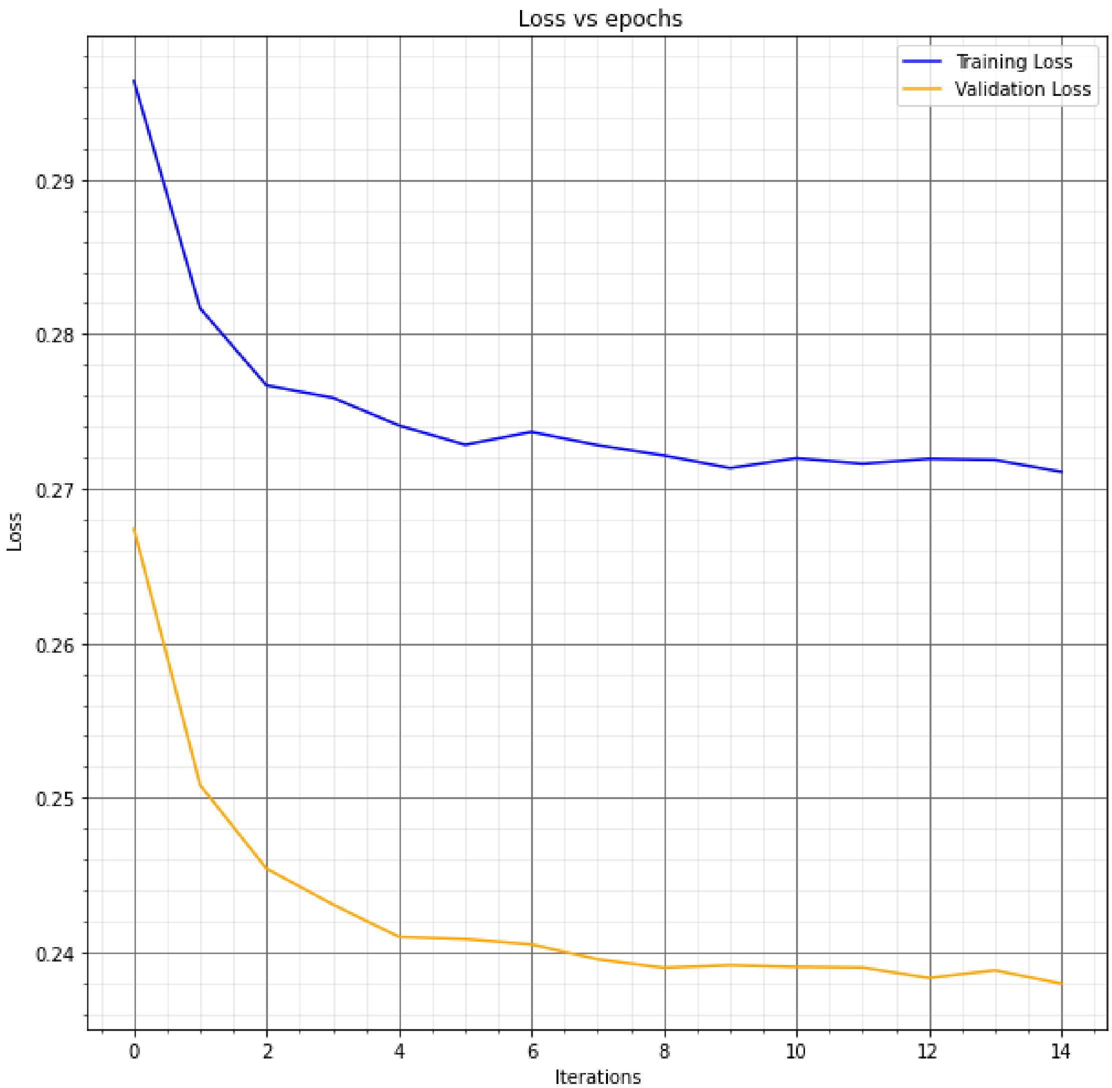


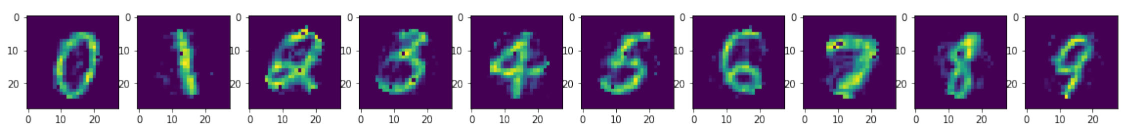


for noise value = 0.8

Epochs: 1/15 ||| with Training Loss = 0.29636359214782715 ||| Validation Loss = 0.26738864183425903
Epochs: 2/15 ||| with Training Loss = 0.2816535234451294 ||| Validation Loss = 0.2507992386817932

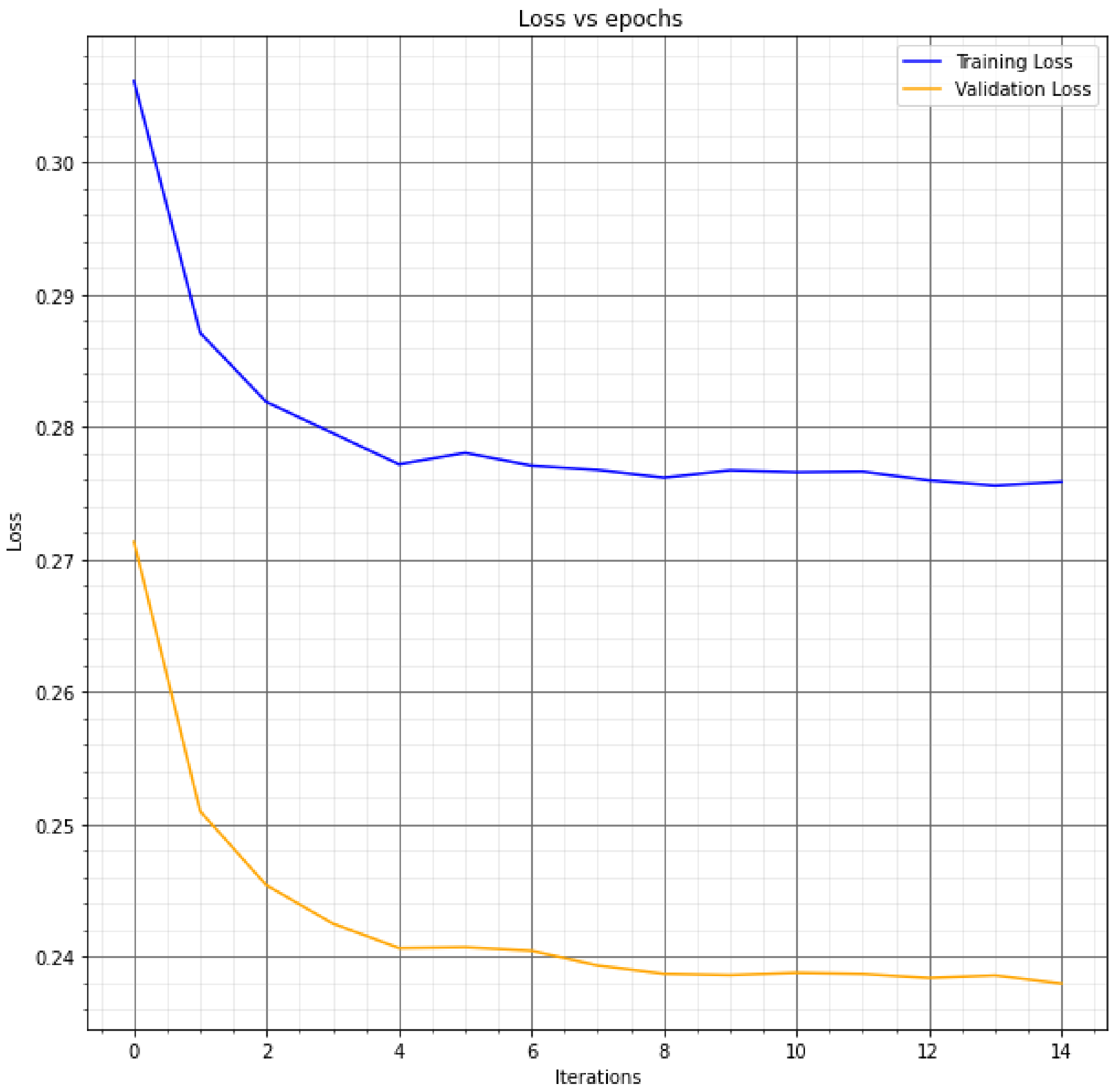
```
Epochs: 3/15 ||| with Training Loss = 0.27668309211730957 ||| Validation Loss =
0.24541570246219635
Epochs: 4/15 ||| with Training Loss = 0.275882750749588 ||| Validation Loss =
0.24309852719306946
Epochs: 5/15 ||| with Training Loss = 0.27408668398857117 ||| Validation Loss =
0.24099692702293396
Epochs: 6/15 ||| with Training Loss = 0.2728397846221924 ||| Validation Loss =
0.2408573031425476
Epochs: 7/15 ||| with Training Loss = 0.2736726701259613 ||| Validation Loss =
0.24050545692443848
Epochs: 8/15 ||| with Training Loss = 0.2728123962879181 ||| Validation Loss =
0.23954950273036957
Epochs: 9/15 ||| with Training Loss = 0.2721427381038666 ||| Validation Loss =
0.23900291323661804
Epochs: 10/15 ||| with Training Loss = 0.27132874727249146 ||| Validation Loss =
0.23917683959007263
Epochs: 11/15 ||| with Training Loss = 0.2719590961933136 ||| Validation Loss =
0.2390621155500412
Epochs: 12/15 ||| with Training Loss = 0.2716068625450134 ||| Validation Loss =
0.23901471495628357
Epochs: 13/15 ||| with Training Loss = 0.2719190716743469 ||| Validation Loss =
0.23834188282489777
Epochs: 14/15 ||| with Training Loss = 0.27185115218162537 ||| Validation Loss =
0.2388356328010559
Epochs: 15/15 ||| with Training Loss = 0.2710857689380646 ||| Validation Loss =
0.2379731684923172
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```

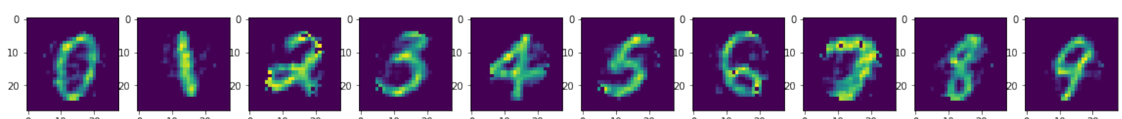




```
for noise value = 0.9

Epochs: 1/15 ||| with Training Loss = 0.3061346113681793 ||| Validation Loss =
0.2713543474674225
Epochs: 2/15 ||| with Training Loss = 0.28712302446365356 ||| Validation Loss =
0.2509884238243103
```

```
Epochs: 3/15 ||| with Training Loss = 0.2818964719772339 ||| Validation Loss =
0.24540047347545624
Epochs: 4/15 ||| with Training Loss = 0.2795545160770416 ||| Validation Loss =
0.24252009391784668
Epochs: 5/15 ||| with Training Loss = 0.2771940231323242 ||| Validation Loss =
0.24065808951854706
Epochs: 6/15 ||| with Training Loss = 0.27806445956230164 ||| Validation Loss =
0.24072566628456116
Epochs: 7/15 ||| with Training Loss = 0.2770940065383911 ||| Validation Loss =
0.24046345055103302
Epochs: 8/15 ||| with Training Loss = 0.27676868438720703 ||| Validation Loss =
0.2393503040075302
Epochs: 9/15 ||| with Training Loss = 0.2761901319026947 ||| Validation Loss =
0.23872050642967224
Epochs: 10/15 ||| with Training Loss = 0.27673855423927307 ||| Validation Loss =
0.23862308263778687
Epochs: 11/15 ||| with Training Loss = 0.2765922546386719 ||| Validation Loss =
0.23879817128181458
Epochs: 12/15 ||| with Training Loss = 0.2766468822956085 ||| Validation Loss =
0.2387128472328186
Epochs: 13/15 ||| with Training Loss = 0.2759872376918793 ||| Validation Loss =
0.2384033203125
Epochs: 14/15 ||| with Training Loss = 0.2755905091762543 ||| Validation Loss =
0.2386041283607483
Epochs: 15/15 ||| with Training Loss = 0.27587398886680603 ||| Validation Loss =
0.2379777580499649
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```





6.2 Observations:

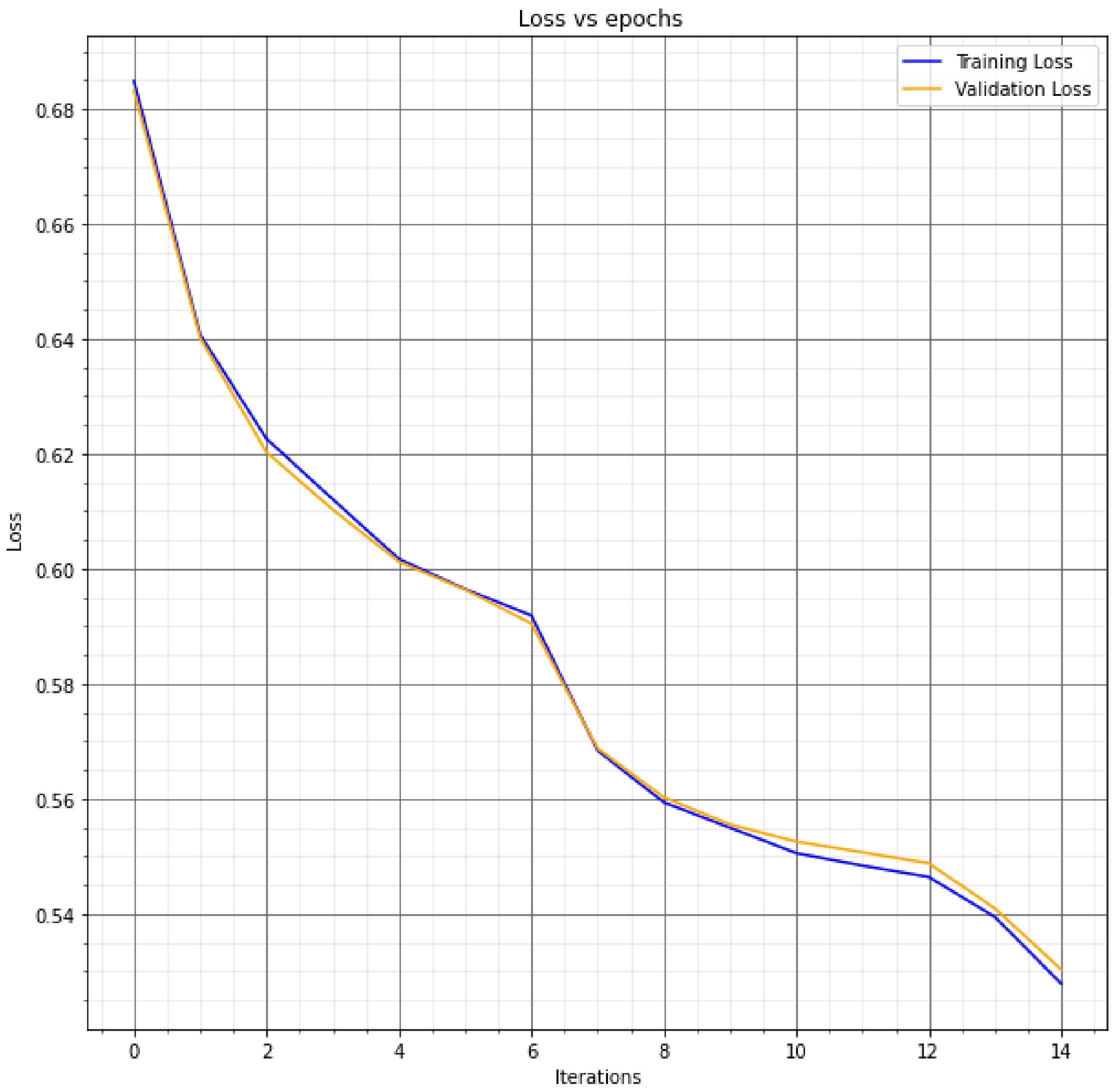
- we can observe that the training loss is minimum for the least noise value for the standard autoencoder in question 2

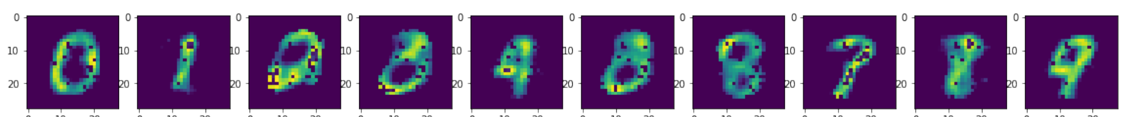
[74]:
```python
class autoencoder_denoising(nn.Module):
  def __init__(self):
    super(autoencoder_denoising, self).__init__()
    self.encoder = nn.Sequential(nn.Linear(784,64), nn.ReLU(), nn.Linear(64,8),␣
↪nn.ReLU())
    self.decoder =nn.Sequential(nn.Linear(8,64), nn.ReLU(), nn.Linear(64,784),␣
↪nn.ReLU())

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

[75]:
```python
noise_val = [0.3, 0.5, 0.8, 0.9]
for i in range(4):
  print(f'\n for noise value = {noise_val[i]}\n')
  model5 = autoencoder_denoising().to(device)
  optimizer5 = optim.Adam(model5.parameters(), lr = 0.003)
  criterion = nn.MSELoss()
  training_loss, val_loss = Train(model5, optimizer5, criterion, epochs,␣
↪denoise = True, noise_val = noise_val[i])

  visualise_loss(training_loss, val_loss)
  visualise_model(model5, data_ind, data)
```

```
 for noise value = 0.3

Epochs: 1/15 ||| with Training Loss = 0.6848006248474121 ||| Validation Loss =
0.6831685304641724
Epochs: 2/15 ||| with Training Loss = 0.6406914591789246 ||| Validation Loss =
0.640123724937439
Epochs: 3/15 ||| with Training Loss = 0.6226150989532471 ||| Validation Loss =
0.620271623134613
Epochs: 4/15 ||| with Training Loss = 0.6120909452438354 ||| Validation Loss =
0.6102980375289917
Epochs: 5/15 ||| with Training Loss = 0.6017044186592102 ||| Validation Loss =
0.601205587387085
Epochs: 6/15 ||| with Training Loss = 0.596468985080719 ||| Validation Loss =
0.5963802337646484
Epochs: 7/15 ||| with Training Loss = 0.5918829441070557 ||| Validation Loss =
0.5904800891876221
Epochs: 8/15 ||| with Training Loss = 0.5683847069740295 ||| Validation Loss =
0.5687422156333923
Epochs: 9/15 ||| with Training Loss = 0.5593767166137695 ||| Validation Loss =
0.5602560639381409
Epochs: 10/15 ||| with Training Loss = 0.5549748539924622 ||| Validation Loss =
```

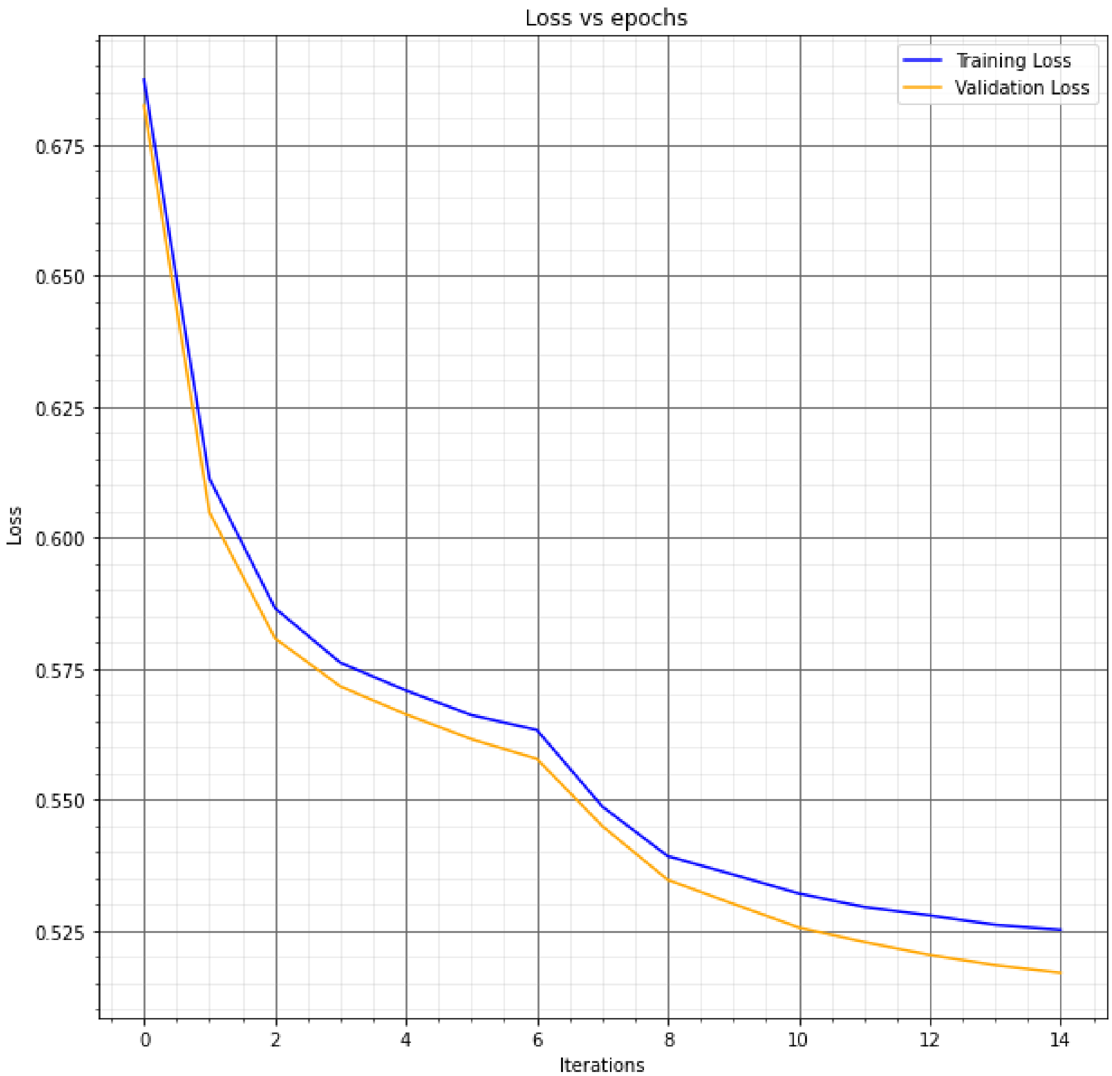
```
0.5555382370948792
Epochs: 11/15 ||| with Training Loss = 0.5505425930023193 ||| Validation Loss =
0.5525535941123962
Epochs: 12/15 ||| with Training Loss = 0.5483794212341309 ||| Validation Loss =
0.550687313079834
Epochs: 13/15 ||| with Training Loss = 0.5464069247245789 ||| Validation Loss =
0.5487973093986511
Epochs: 14/15 ||| with Training Loss = 0.5394445657730103 ||| Validation Loss =
0.5409221053123474
Epochs: 15/15 ||| with Training Loss = 0.5278555750846863 ||| Validation Loss =
0.5303293466567993
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```

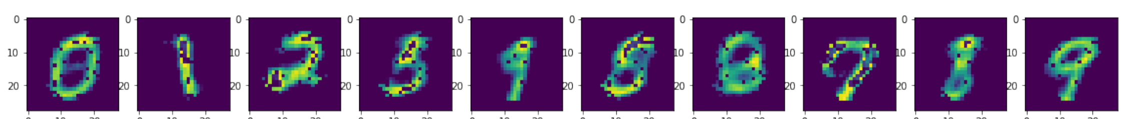




```
for noise value = 0.5

Epochs: 1/15 ||| with Training Loss = 0.6872881054878235 ||| Validation Loss =
0.6823856234550476
Epochs: 2/15 ||| with Training Loss = 0.6112332344055176 ||| Validation Loss =
0.6048036217689514
```

```
Epochs: 3/15 ||| with Training Loss = 0.5865513682365417 ||| Validation Loss =
0.5807957053184509
Epochs: 4/15 ||| with Training Loss = 0.5761266350746155 ||| Validation Loss =
0.5716156959533691
Epochs: 5/15 ||| with Training Loss = 0.5708513855934143 ||| Validation Loss =
0.5662994384765625
Epochs: 6/15 ||| with Training Loss = 0.5661503672599792 ||| Validation Loss =
0.5615786910057068
Epochs: 7/15 ||| with Training Loss = 0.5633235573768616 ||| Validation Loss =
0.5578161478042603
Epochs: 8/15 ||| with Training Loss = 0.5487060546875 ||| Validation Loss =
0.5449833273887634
Epochs: 9/15 ||| with Training Loss = 0.5392489433288574 ||| Validation Loss =
0.5347309708595276
Epochs: 10/15 ||| with Training Loss = 0.5357452034950256 ||| Validation Loss =
0.5301650166511536
Epochs: 11/15 ||| with Training Loss = 0.5321444272994995 ||| Validation Loss =
0.5256530046463013
Epochs: 12/15 ||| with Training Loss = 0.5295639634132385 ||| Validation Loss =
0.522879958152771
Epochs: 13/15 ||| with Training Loss = 0.5279529690742493 ||| Validation Loss =
0.5204227566719055
Epochs: 14/15 ||| with Training Loss = 0.5261601805686951 ||| Validation Loss =
0.5184678435325623
Epochs: 15/15 ||| with Training Loss = 0.5252169370651245 ||| Validation Loss =
0.5170124769210815
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```

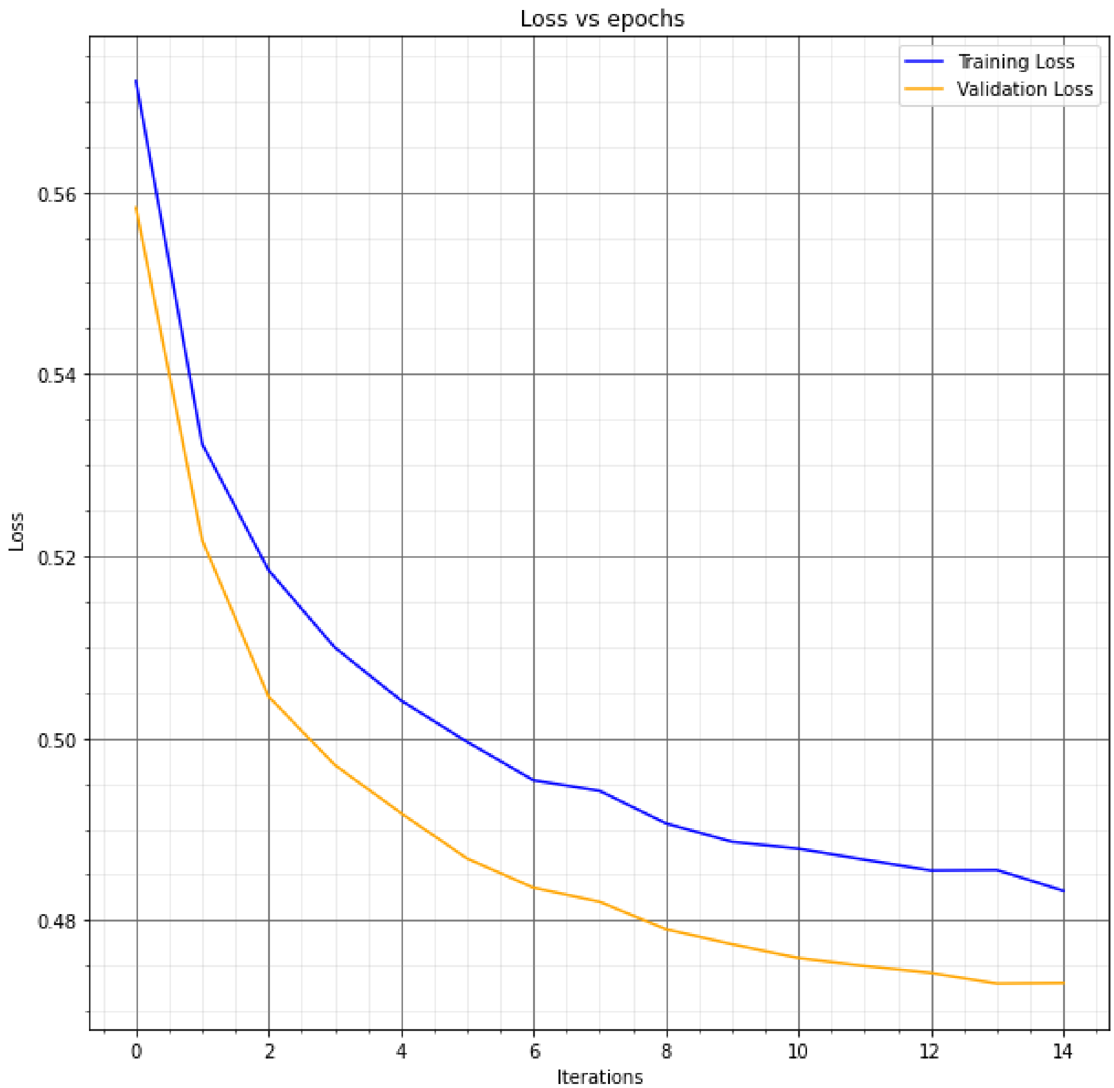


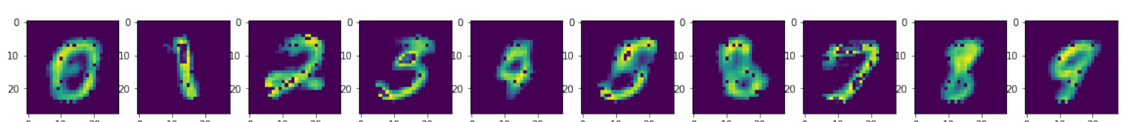


for noise value = 0.8

```
Epochs: 1/15 ||| with Training Loss = 0.5721724033355713 ||| Validation Loss =
0.5582663416862488
Epochs: 2/15 ||| with Training Loss = 0.5323060154914856 ||| Validation Loss =
0.5217342972755432
Epochs: 3/15 ||| with Training Loss = 0.5184629559516907 ||| Validation Loss =
```

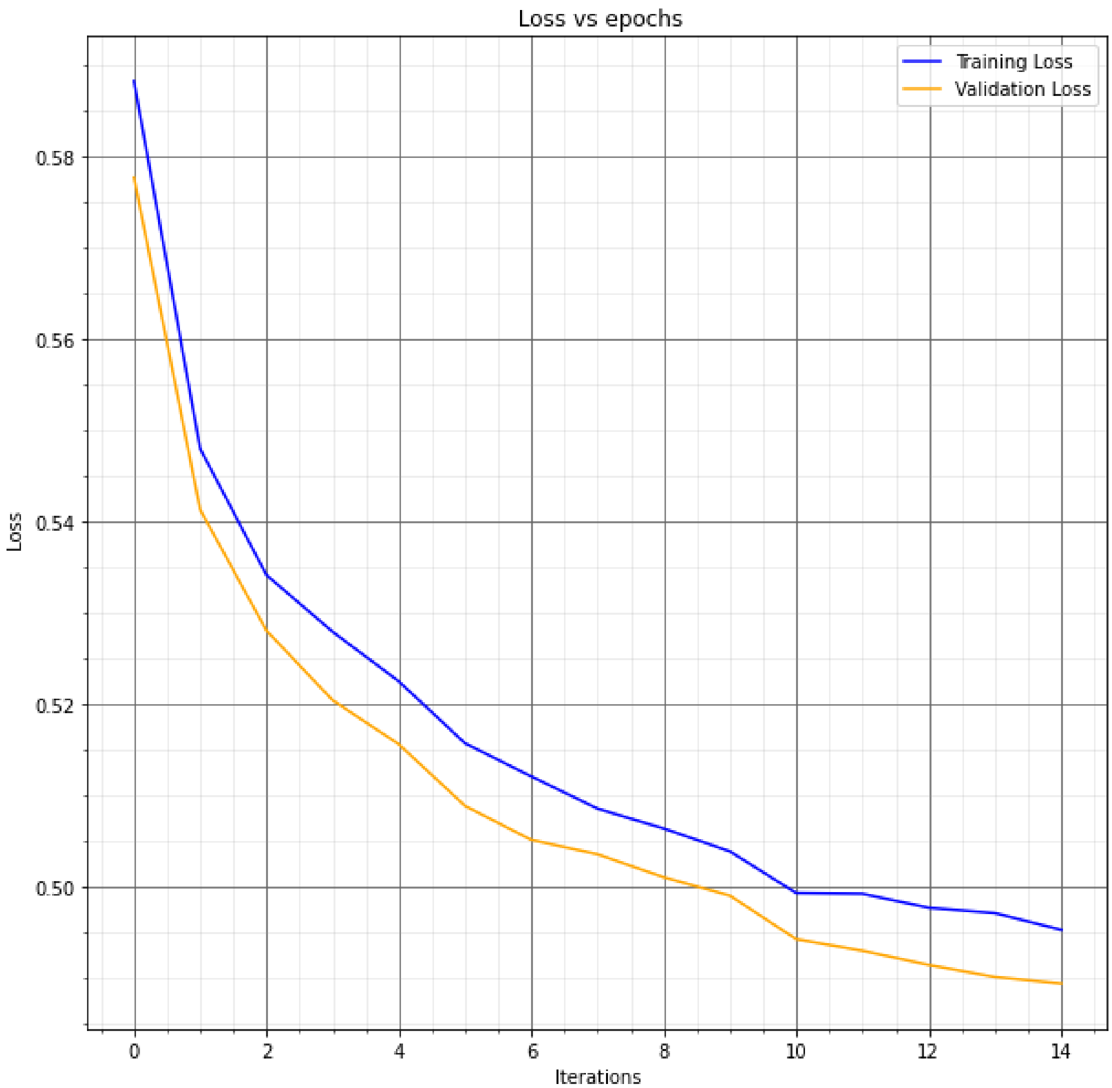
```
0.5045705437660217
Epochs: 4/15 ||| with Training Loss = 0.5099622011184692 ||| Validation Loss =
0.4970572292804718
Epochs: 5/15 ||| with Training Loss = 0.5041430592536926 ||| Validation Loss =
0.49176591634750366
Epochs: 6/15 ||| with Training Loss = 0.4995822310447693 ||| Validation Loss =
0.4867702126502991
Epochs: 7/15 ||| with Training Loss = 0.49537867307662964 ||| Validation Loss =
0.4835708737373352
Epochs: 8/15 ||| with Training Loss = 0.49420687556266785 ||| Validation Loss =
0.4820214807987213
Epochs: 9/15 ||| with Training Loss = 0.4906397759914398 ||| Validation Loss =
0.47901052236557007
Epochs: 10/15 ||| with Training Loss = 0.4886122941970825 ||| Validation Loss =
0.47735312581062317
Epochs: 11/15 ||| with Training Loss = 0.48786231875419617 ||| Validation Loss =
0.475845068693161
Epochs: 12/15 ||| with Training Loss = 0.4866437315940857 ||| Validation Loss =
0.4749729633331299
Epochs: 13/15 ||| with Training Loss = 0.48546794056892395 ||| Validation Loss =
0.47420862317085266
Epochs: 14/15 ||| with Training Loss = 0.4855162501335144 ||| Validation Loss =
0.4730307459831238
Epochs: 15/15 ||| with Training Loss = 0.4832351803779602 ||| Validation Loss =
0.4730837941169739
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```

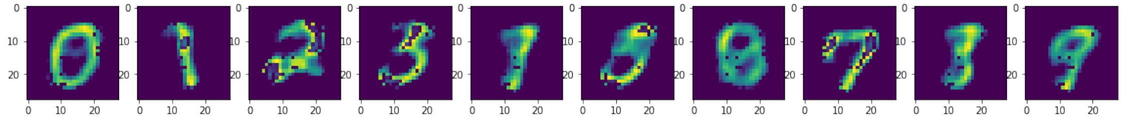




```
for noise value = 0.9

Epochs: 1/15 ||| with Training Loss = 0.5881263017654419 ||| Validation Loss =
0.5775903463363647
Epochs: 2/15 ||| with Training Loss = 0.5478460192680359 ||| Validation Loss =
0.5411717295646667
```

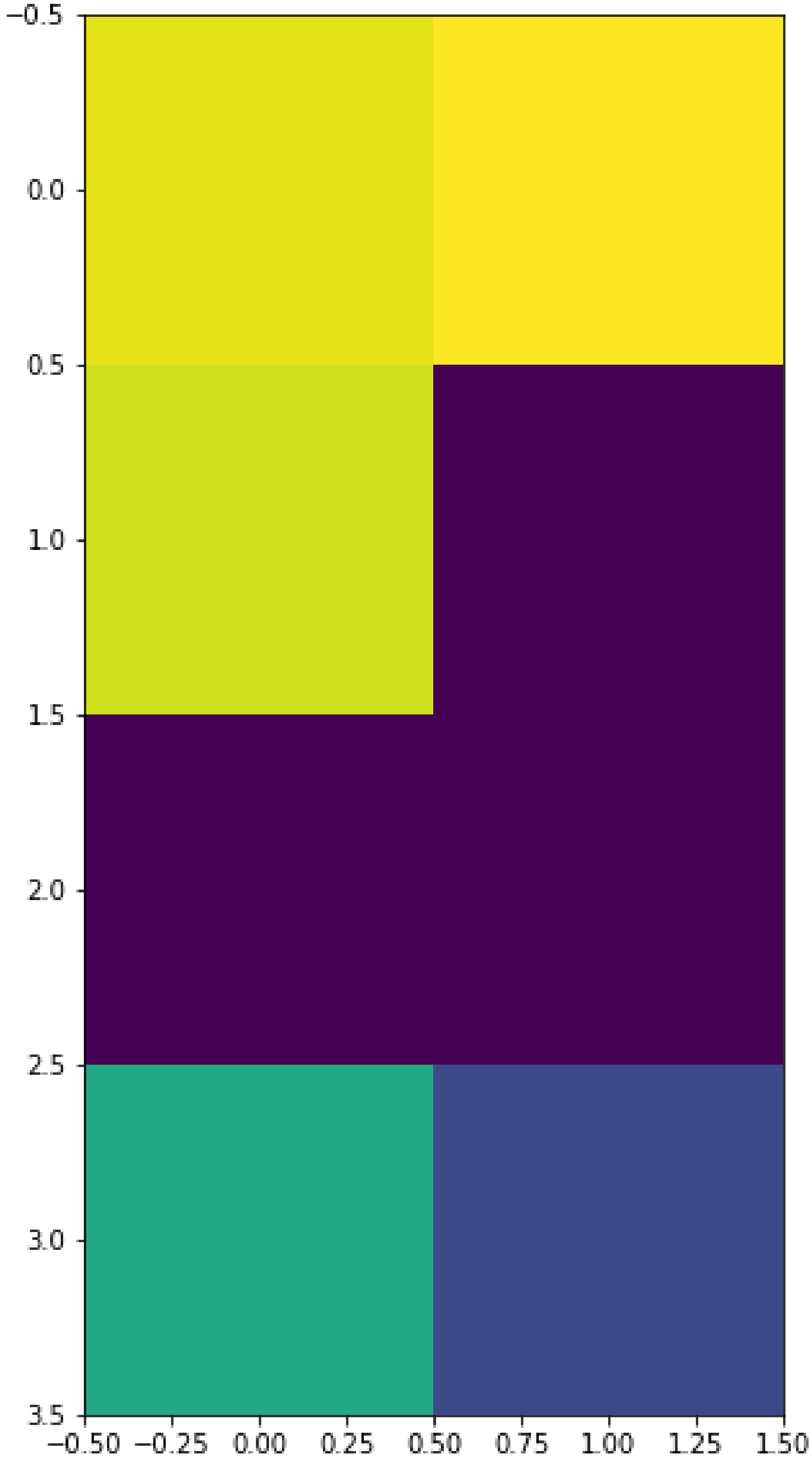
```
Epochs: 3/15 ||| with Training Loss = 0.5340295433998108 ||| Validation Loss =
0.5279478430747986
Epochs: 4/15 ||| with Training Loss = 0.5278148651123047 ||| Validation Loss =
0.5203167200088501
Epochs: 5/15 ||| with Training Loss = 0.5223813652992249 ||| Validation Loss =
0.5154925584793091
Epochs: 6/15 ||| with Training Loss = 0.515595555305481 ||| Validation Loss =
0.508739173412323
Epochs: 7/15 ||| with Training Loss = 0.5119762420654297 ||| Validation Loss =
0.5050249695777893
Epochs: 8/15 ||| with Training Loss = 0.5084444880485535 ||| Validation Loss =
0.5034490823745728
Epochs: 9/15 ||| with Training Loss = 0.5062755346298218 ||| Validation Loss =
0.500933825969696
Epochs: 10/15 ||| with Training Loss = 0.5037600994110107 ||| Validation Loss =
0.4989137053489685
Epochs: 11/15 ||| with Training Loss = 0.4992004930973053 ||| Validation Loss =
0.4941493272781372
Epochs: 12/15 ||| with Training Loss = 0.49911805987358093 ||| Validation Loss =
0.4928869605064392
Epochs: 13/15 ||| with Training Loss = 0.49759629368782043 ||| Validation Loss =
0.49133554100990295
Epochs: 14/15 ||| with Training Loss = 0.4970111548900604 ||| Validation Loss =
0.49001121520996094
Epochs: 15/15 ||| with Training Loss = 0.4951775372028351 ||| Validation Loss =
0.48928576707839966
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```





```
[86]: with torch.no_grad():
        output5, encoded5 = model5.forward(data[data_ind[3]].reshape(1,784).float())
        print(encoded5.shape)
        plt.imshow(encoded5.reshape(4, 2))
```

```
torch.Size([1, 8])
```


−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
−0.50 −0.25 0.00 0.25 0.50 0.75 1.00 1.25 1.50

7 Convolutional Autoencoders

Unpooling Convolutional Autoencoder

Encoder Module: * **28X 28 X 1** *to* **14 X 14 X 8** * **14 X 14 X 8** *to* **7 X 7 X 16** * **7 X 7 X 16** *to* **3 X 3 X 16**

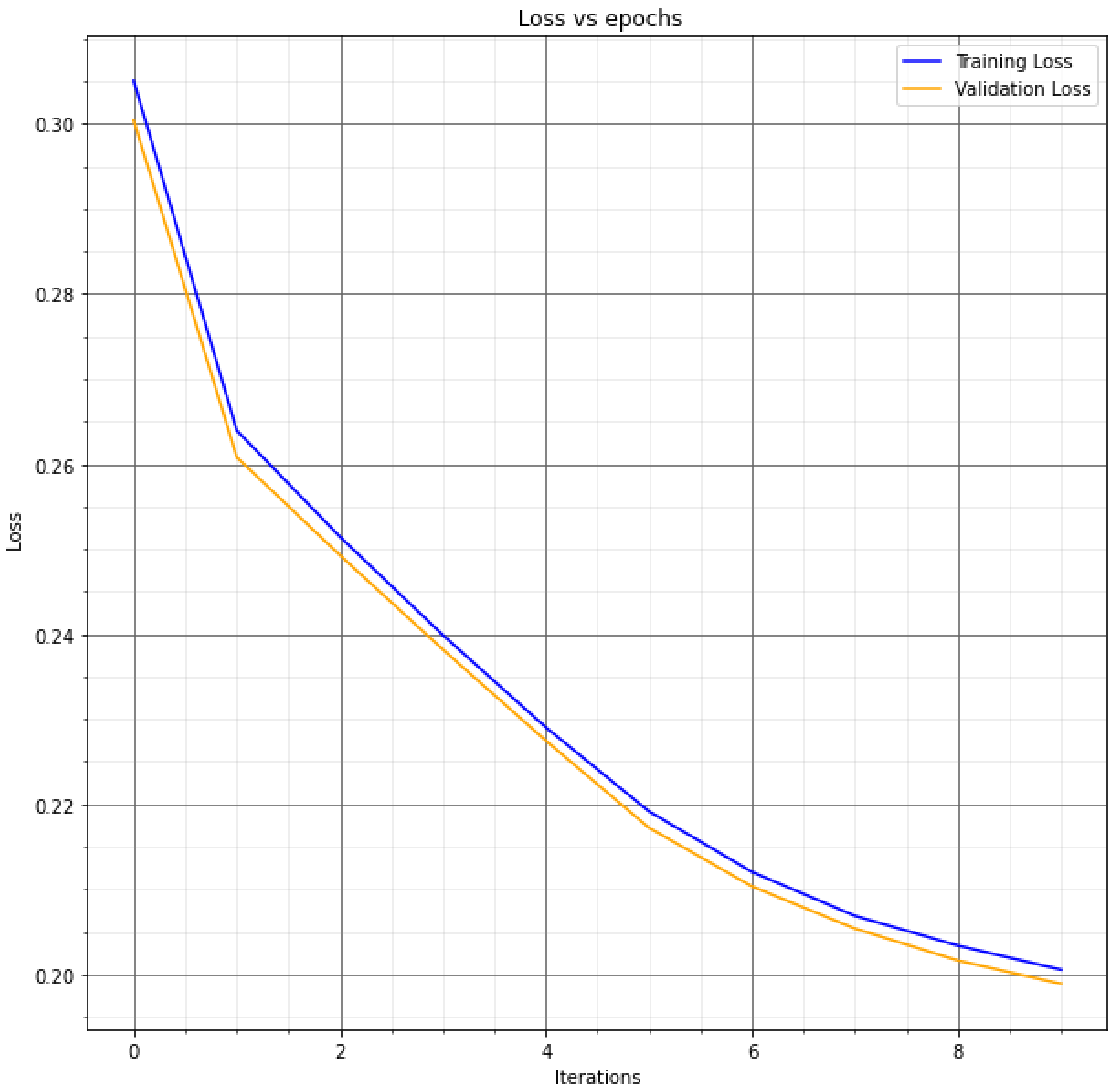
Decoder Module: * **7 X 7 X 16** *to* **7 X 7 X 16** * **7 X 7 X 16** *to* **14 X 14 X 8** * **14 X 14 X 8** *to* **28 X 28 X 1**

[76]:
```
class autoencoder_convolutional_unpool(nn.Module):
  def __init__(self):
    super(autoencoder_convolutional_unpool, self).__init__()
    #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))
    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.Identity())
    self.decoder_conv2 = nn.Sequential(
        nn.Conv2d(16,8, kernel_size = 3, stride = 1,padding= 1),nn.ReLU())
    self.decoder_conv3 = nn.Sequential(
        nn.Conv2d(8,1, kernel_size = 3, stride = 1,padding= 1),nn.ReLU())

    #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())
    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]))
    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
```

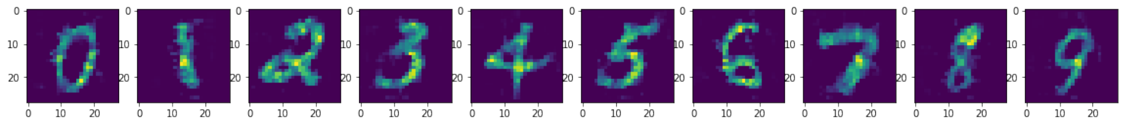
[77]:
```
epochs = 10
model6 = autoencoder_convolutional_unpool().to(device)
optimizer6 = optim.Adam(model6.parameters(), lr = 0.003)
criterion = nn.MSELoss()
training_loss, val_loss = Train(model6, optimizer6, criterion, epochs,␣
 ↪model_flag=1)
visualise_loss(training_loss, val_loss)
```

```
Epochs: 1/10 ||| with Training Loss = 0.30504223704338074 ||| Validation Loss =
0.3003740906715393
Epochs: 2/10 ||| with Training Loss = 0.26394376158714294 ||| Validation Loss =
0.2607918977737427
Epochs: 3/10 ||| with Training Loss = 0.25145184993743896 ||| Validation Loss =
0.24928033351898193
Epochs: 4/10 ||| with Training Loss = 0.23987926542758942 ||| Validation Loss =
0.23818303644657135
Epochs: 5/10 ||| with Training Loss = 0.22902315855026245 ||| Validation Loss =
0.22745725512504578
Epochs: 6/10 ||| with Training Loss = 0.21915584802627563 ||| Validation Loss =
0.21722395718097687
Epochs: 7/10 ||| with Training Loss = 0.21203015744686127 ||| Validation Loss =
0.2103302925825119
Epochs: 8/10 ||| with Training Loss = 0.2068602591753006 ||| Validation Loss =
0.20536252856254578
Epochs: 9/10 ||| with Training Loss = 0.20335638523101807 ||| Validation Loss =
0.2016081064939499
Epochs: 10/10 ||| with Training Loss = 0.2005344033241272 ||| Validation Loss =
0.19886523485183716
```



[78]:
```
visualise_model(model6, data_ind, data, model_flag = 1)
```

```
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



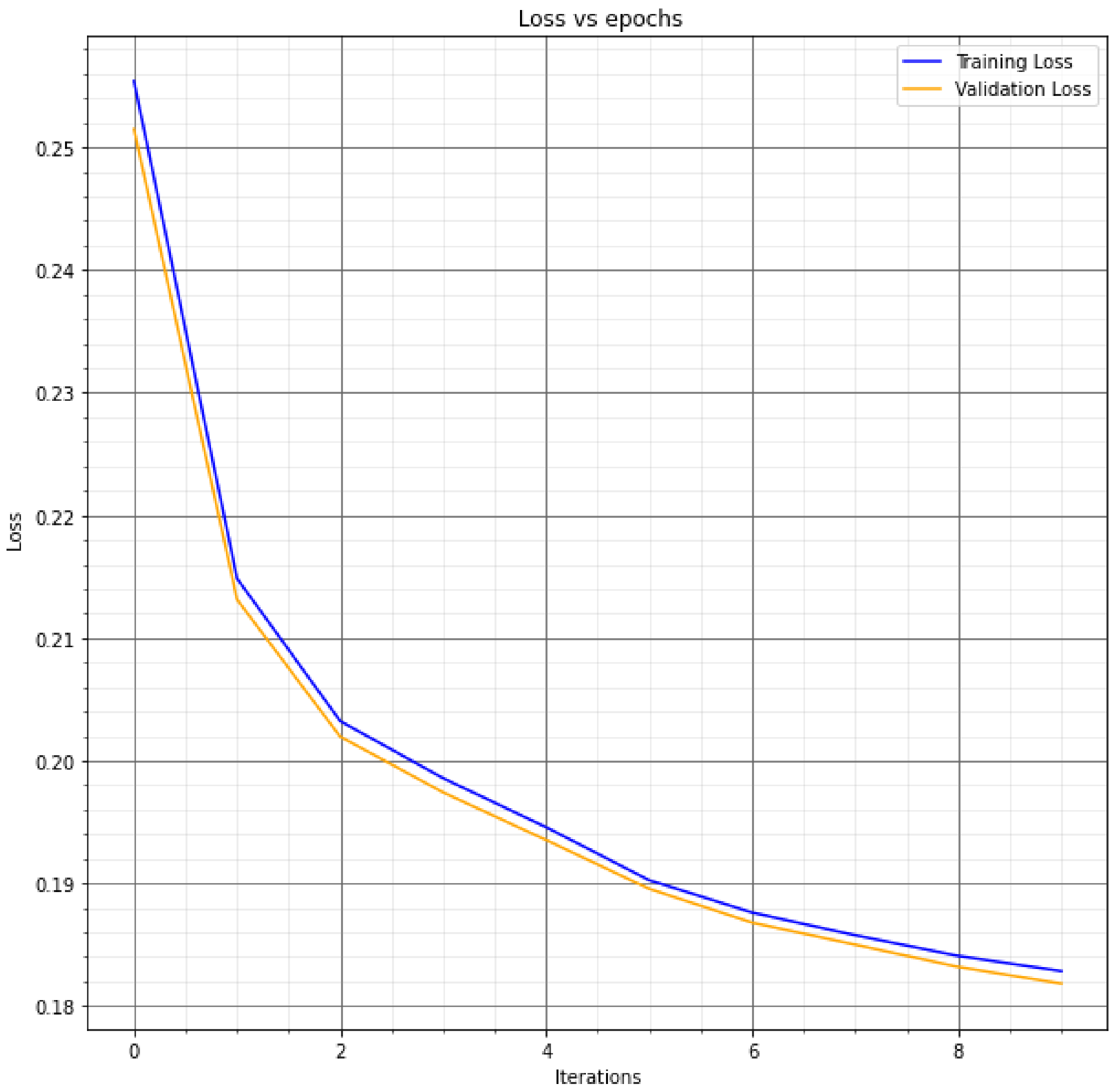
Unpooling + Deconvolution

[79]:
```
class autoencoder_deconv_unpool(nn.Module):
  def __init__(self):
    super(autoencoder_deconv_unpool,self).__init__()

      #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))
    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())

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

  def forward(self,x):

    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]))
```

```
    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
```

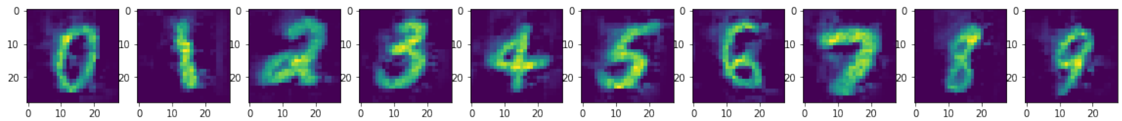
[80]:
```
epochs = 10
model8 = autoencoder_deconv_unpool().to(device)
optimizer8 = optim.Adam(model8.parameters(), lr = 0.003)
criterion = nn.MSELoss()
training_loss, val_loss = Train(model8, optimizer8, criterion, epochs,␣
 ↪model_flag=1)
visualise_loss(training_loss, val_loss)
```

```
Epochs: 1/10 ||| with Training Loss = 0.25539734959602356 ||| Validation Loss =
0.2514868378639221
Epochs: 2/10 ||| with Training Loss = 0.2148585319519043 ||| Validation Loss =
0.2131403684616089
Epochs: 3/10 ||| with Training Loss = 0.2032325565814972 ||| Validation Loss =
0.20196329057216644
Epochs: 4/10 ||| with Training Loss = 0.1985708326101303 ||| Validation Loss =
0.19741162657737732
Epochs: 5/10 ||| with Training Loss = 0.1945735067129135 ||| Validation Loss =
0.19352756440639496
Epochs: 6/10 ||| with Training Loss = 0.1902538686990738 ||| Validation Loss =
0.18954743444919586
Epochs: 7/10 ||| with Training Loss = 0.1876154989004135 ||| Validation Loss =
0.18679484724998474
Epochs: 8/10 ||| with Training Loss = 0.18576161563396454 ||| Validation Loss =
0.18499350547790527
Epochs: 9/10 ||| with Training Loss = 0.18407995998859406 ||| Validation Loss =
0.18318553268909454
Epochs: 10/10 ||| with Training Loss = 0.18284496665000916 ||| Validation Loss =
0.18181955814361572
```



[81]:
```
visualise_model(model8, data_ind, data, model_flag = 1)
```

```
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



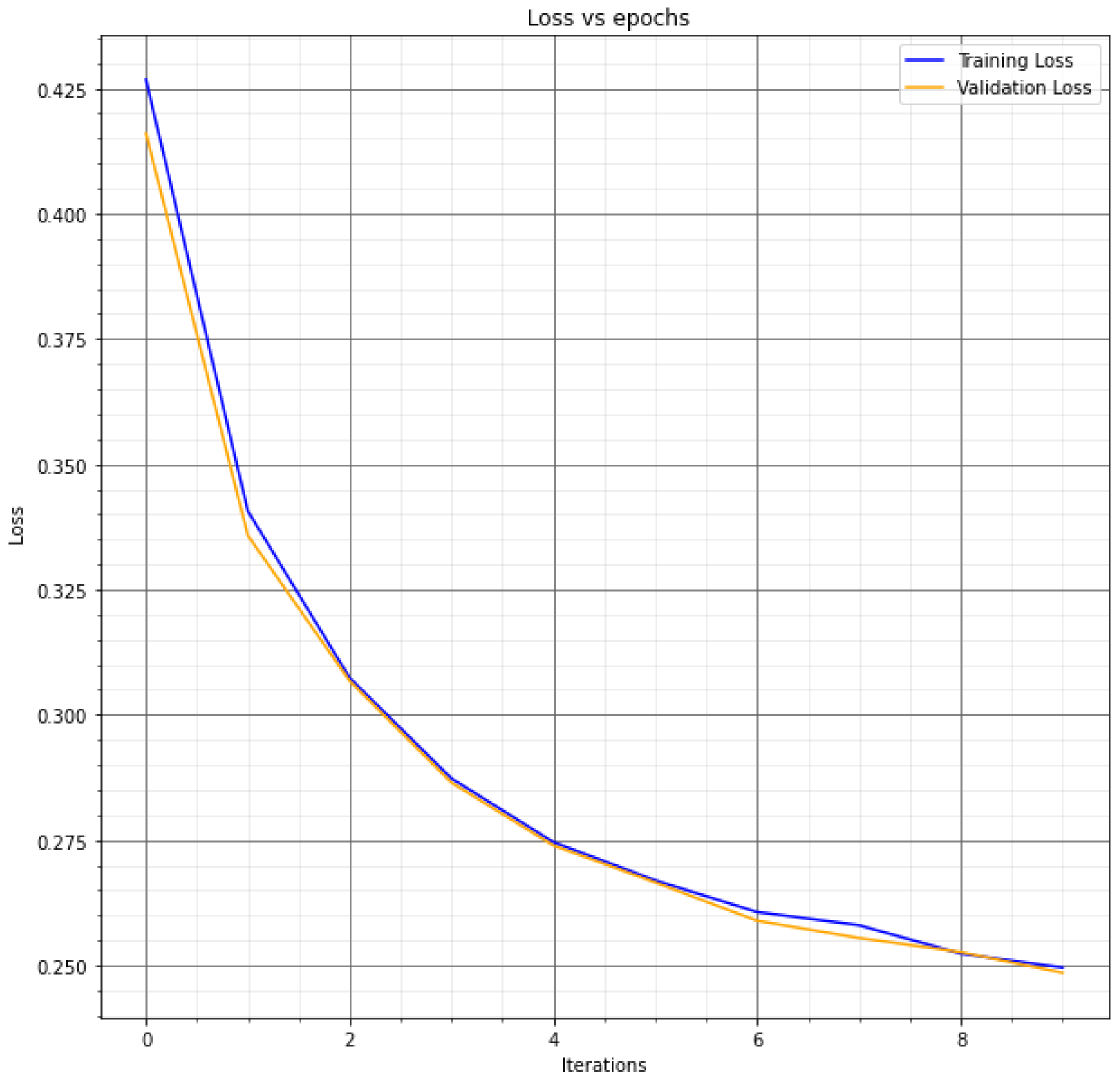
Deconvolution

[82]:
```
class autoencoder_conv_deconv(nn.Module):
  def __init__(self):
    super(autoencoder_conv_deconv,self).__init__()

    #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)))
    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)))

    #initializing the 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
```

[83]:

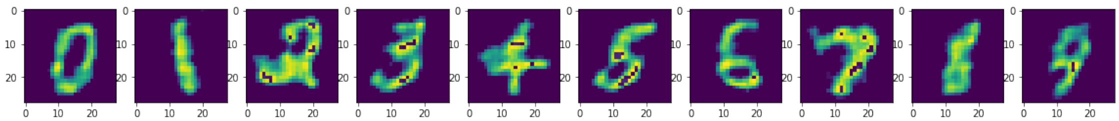
```
epochs = 10
model7 = autoencoder_conv_deconv().to(device)
optimizer7 = optim.Adam(model7.parameters(), lr = 0.003)
criterion = nn.MSELoss()
training_loss, val_loss = Train(model7, optimizer7, criterion, epochs,␣
 ↪model_flag=1)
visualise_loss(training_loss, val_loss)
```

```
Epochs: 1/10 ||| with Training Loss = 0.42669978737831116 ||| Validation Loss =
0.41601505875587463
Epochs: 2/10 ||| with Training Loss = 0.34065744280815125 ||| Validation Loss =
0.3357812762260437
Epochs: 3/10 ||| with Training Loss = 0.30736157298088074 ||| Validation Loss =
0.3067431151866913
Epochs: 4/10 ||| with Training Loss = 0.28730306029319763 ||| Validation Loss =
0.28645655512809753
Epochs: 5/10 ||| with Training Loss = 0.27460142970085144 ||| Validation Loss =
0.2739414572715759
Epochs: 6/10 ||| with Training Loss = 0.2669924795627594 ||| Validation Loss =
0.266570508480072
Epochs: 7/10 ||| with Training Loss = 0.260661780834198 ||| Validation Loss =
0.25891631841659546
Epochs: 8/10 ||| with Training Loss = 0.25803104043006897 ||| Validation Loss =
0.25548702478408813
Epochs: 9/10 ||| with Training Loss = 0.2523624002933502 ||| Validation Loss =
0.25267475843429565
Epochs: 10/10 ||| with Training Loss = 0.24958175420761108 ||| Validation Loss =
0.24853822588920593
```



[84]:
```
visualise_model(model7, data_ind, data, model_flag = 1)
```

```
for digit 0
for digit 1
for digit 2
for digit 3
for digit 4
for digit 5
for digit 6
for digit 7
for digit 8
for digit 9
```



7.1 Observations:

- We observe that for unpooling + deconvolution we get the best output for convolutional autoencoders , suceeded by unpooling and deconvolution.
- Convolutional autoencoders take more time to run though, as compared to the standard autoencoders