Exercise Sheet 8 (Autoencoder)

January 19, 2025

Exercises for the course Deep Learning 1 Winter Semester 2022/25

Machine Learning Group Faculty IV – Electrical Engineering and Computer Science Technische Universität Berlin

Exercise Sheet 10 - Autoencoders

In this exercise, we would like to train an auto-encoder on the MNIST dataset.

We seek to optimize the objective:

$$\min_\theta \quad \underbrace{\frac{1}{N}\sum_{i=1}^{N}\|x_i - \hat{x}_i\|^2}_{\text{reconstruction}}$$

The reconstruction term is the standard mean square error between the data points and their reconstructions.

1 Load MNIST dataset

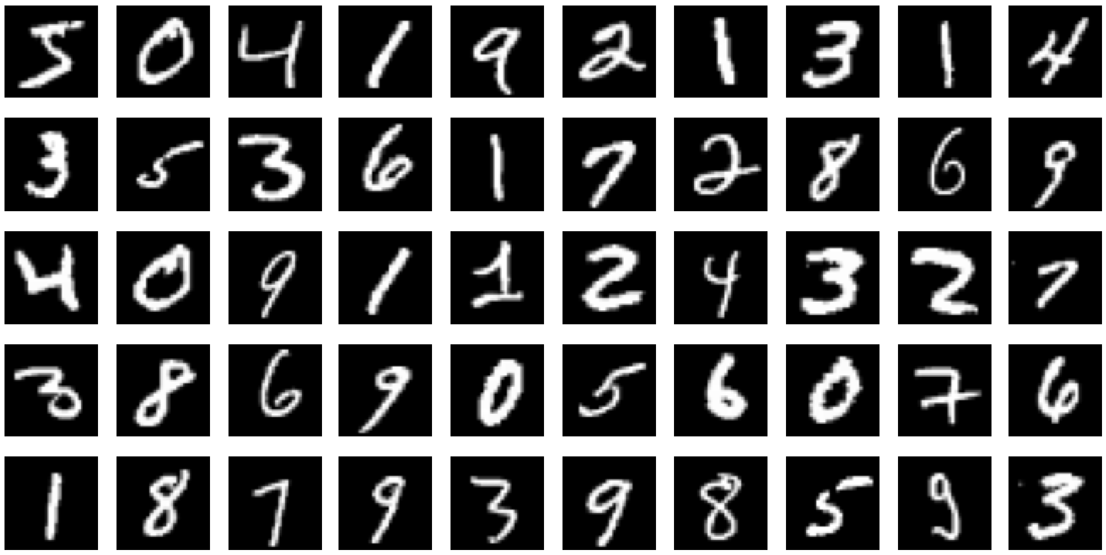
First, we load the MNIST dataset and display some example images which show the general distribution of the data. The dataset is built into torchvision which automatically downloads the data into the `data` directory.

[2]:
```
from torchvision.datasets import MNIST
from torchvision import transforms as T
import matplotlib.pyplot as plt
import torch

data_root = './data'
train_dataset = MNIST(data_root, train=True, download=True, transform=T.
  ↪ToTensor())
test_dataset = MNIST(data_root, train=False, download=True, transform=T.
  ↪ToTensor())

def show_samples(dataset):
    h, w = 5, 10
    fig, ax = plt.subplots(h, w)
```

```python
    fig.set_size_inches((w, h))
    ax = ax.ravel()
    for i in range(h * w):
        img, _ = dataset[i]
        ax[i].imshow(torch.permute(img, (1, 2, 0)), cmap='gray')
        ax[i].axis('off')
    plt.show()

show_samples(train_dataset)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
<urlopen error [Errno 110] Connection timed out>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-
idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-
idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyte.gz

100%|     | 9.91M/9.91M [00:00<00:00, 17.8MB/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
Failed to download (trying next):
<urlopen error [Errno 110] Connection timed out>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-
idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-
idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyte.gz

100%|     | 28.9k/28.9k [00:00<00:00, 476kB/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
Failed to download (trying next):
<urlopen error [Errno 110] Connection timed out>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.gz

100%|     | 1.65M/1.65M [00:00<00:00, 4.44MB/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
```

```
Failed to download (trying next):
<urlopen error [Errno 110] Connection timed out>

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|      | 4.54k/4.54k [00:00<00:00, 8.56MB/s]

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



We can use a DataLoader together with the torchvision dataset which batches the data and can be used as an iterator.

[47]:
```python
from torch.utils.data import DataLoader
batch_size = 128

train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
```

2 1) Forward Pass (20P)

Please implement the forward pass of the network which receives a batch of image vectors $x \in \mathbb{R}^{N \times 1 \times 28 \times 28}$ and returns reconstructed images.

[49]:
```python
from torch import nn
#import solution
```

```
class Autoencoder(nn.Module):

    def __init__(self, h: int = 32):
        super().__init__()

        self.encoder = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 12000),
            nn.Dropout(0.2),
            nn.LeakyReLU(),
            nn.Linear(12000, h),
            nn.LeakyReLU(),
        )

        self.decoder = nn.Sequential(
            nn.Linear(h, 12000),
            nn.Dropout(0.2),
            nn.LeakyReLU(),
            nn.Linear(12000, 784),
            nn.Sigmoid(),
        )

    def forward(self, x: torch.Tensor):
        # --------------------------------------------------
        # TODO: Replace by your code
        # --------------------------------------------------
        #return solution.solution_1_forward(self, x)
        y = x.reshape(x.shape[0],-1)
        enc_outputs = self.encoder(y)
        dec_state = self.decoder(enc_outputs)
        return dec_state.reshape(x.shape)

        # --------------------------------------------------
```

3 2) Training Loop (20P)

The following method implements the training loop of the Autoencoder. Fill in the code for training a single iteration. The loss function to use should be clear from the lecture.

[50]:
```
import torch
from torch import nn, optim
from tqdm.notebook import tqdm
```

```python
def train(loader, h=32, epochs=5):
    model = Autoencoder(h)
    optimizer = optim.Adam(model.parameters(), lr=0.001)

    #  we use GPU
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model.to(device)

    for epoch in range(epochs):
        print(f"Epoch {epoch+1:>2}")
        pbar = tqdm(loader)
        for batch, _ in pbar:
            # --------------------------------------------------
            # TODO: Replace by your code
            # --------------------------------------------------
            #loss = solution.solution_2_train_one_step(model, batch, optimizer)
            batch = batch.to(device)

            loss = torch.sum((batch - model(batch)) ** 2) / batch.shape[0]

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            # --------------------------------------------------
            pbar.set_description(f"Loss: {loss.item():.4f}")

    return model
```

[51]:
```python
# Train the Autoencoder and receive respective parameters with hidden dimension␣
↪32
ae = train(train_loader, h=32, epochs=5)
```

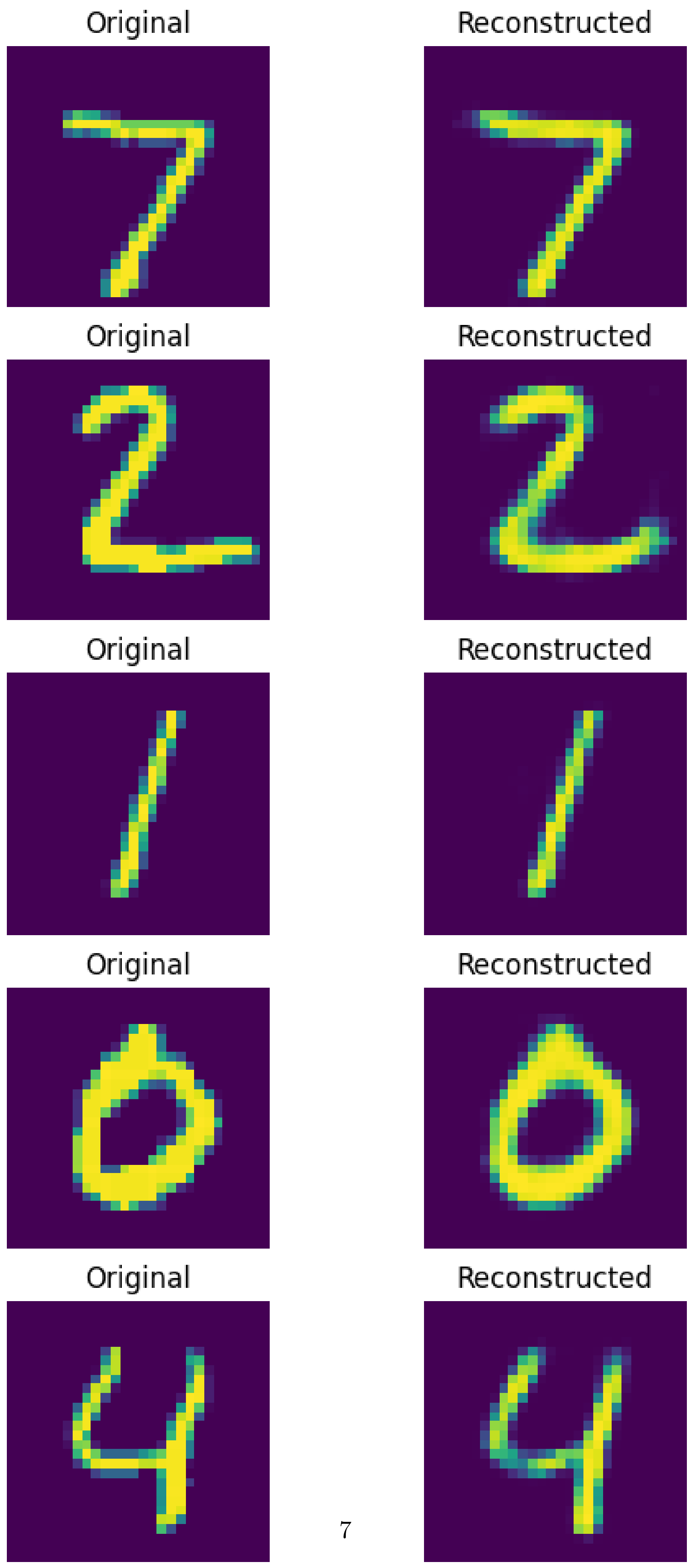
```
Epoch  1

  0%|          | 0/469 [00:00<?, ?it/s]

Epoch  2

  0%|          | 0/469 [00:00<?, ?it/s]

Epoch  3

  0%|          | 0/469 [00:00<?, ?it/s]

Epoch  4

  0%|          | 0/469 [00:00<?, ?it/s]

Epoch  5

  0%|          | 0/469 [00:00<?, ?it/s]
```

4 3) Reconstruction (20P)

After the training, we want to see how good the model reconstructs the encoded example with the decoder.

Plot a few samples of the test set and next to them the reconstruction of the previously trained autoencoder

[52]:
```
# --------------------------------------------------
# TODO: Replace by your code
# --------------------------------------------------
#solution.solution_3_reconstruct_images(ae, test_loader)

ae.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
ae.to(device)

num_samples = 5

test_iter = iter(test_loader)
images, labels = next(test_iter)

fig, ax = plt.subplots(num_samples,2)
fig.set_size_inches(6, 12)

for i in range(num_samples):

    img = images[i].unsqueeze(0).to(device)

    with torch.no_grad():
        reconstructed = ae(img)

    ax[i, 0].imshow(images[i][0])
    ax[i, 0].axis('off')
    ax[i, 0].set_title("Original")

    ax[i, 1].imshow(reconstructed.squeeze(0).cpu().numpy()[0])
    ax[i, 1].axis('off')
    ax[i, 1].set_title("Reconstructed")

plt.show()
# --------------------------------------------------
```

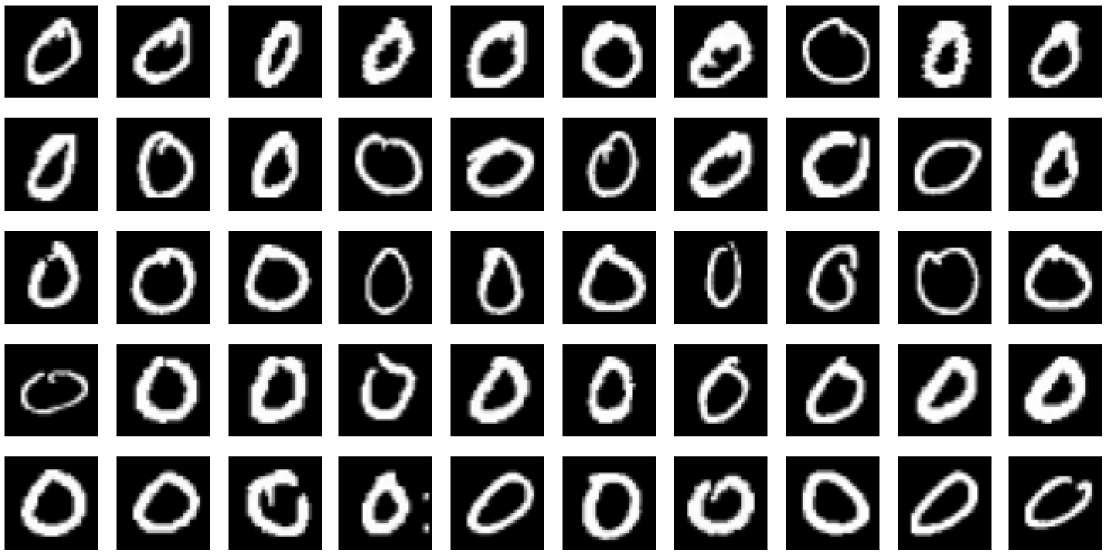

Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed

We can see that we can still recover most of the digit details from the hidden representation.

5 Anomaly Detection

Now we want to use an Autoencoder for anomaly detection. Given a set of normal data points, anomaly detection wants to assign a high anomaly score for samples that are in some way abnormal from the set of normal samples. In our example the normal samples will be all digits up until `threshold_class` (exclusive). All other samples are considered anomalies. You may play around with this parameter, for now we will fix it to 1, meaning we will only consider 0's.

```
[53]: import copy
      from torch.utils.data import Subset
      import numpy as np

      threshold_class = 1
      threshold_class_train = Subset(dataset=copy.deepcopy(train_dataset), indices=np.
       ↪where(train_dataset.targets < threshold_class)[0])
      train_threshold_class_loader = DataLoader(threshold_class_train, batch_size=64,␣
       ↪shuffle=True)
      show_samples(threshold_class_train)
```



```
[54]: # We increase number of epochs because the dataset is now smaller.
      anomaly_ae = train(train_threshold_class_loader, h=32, epochs=10)
```

```
Epoch  1
```

```
  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  2

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  3

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  4

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  5

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  6

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  7

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  8

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch  9

  0%|          | 0/93 [00:00<?, ?it/s]

Epoch 10

  0%|          | 0/93 [00:00<?, ?it/s]
```

6 4) Anomaly Score (20P)

Given now the trained autoencoder, we want to assign each sample an anomaly score. By using your knowledge from the lecture, implement the anomaly score for a batch of samples x, given its reconstruction from the autoencoder. The function should return an array that holds an anomaly score for each sample of the batch.

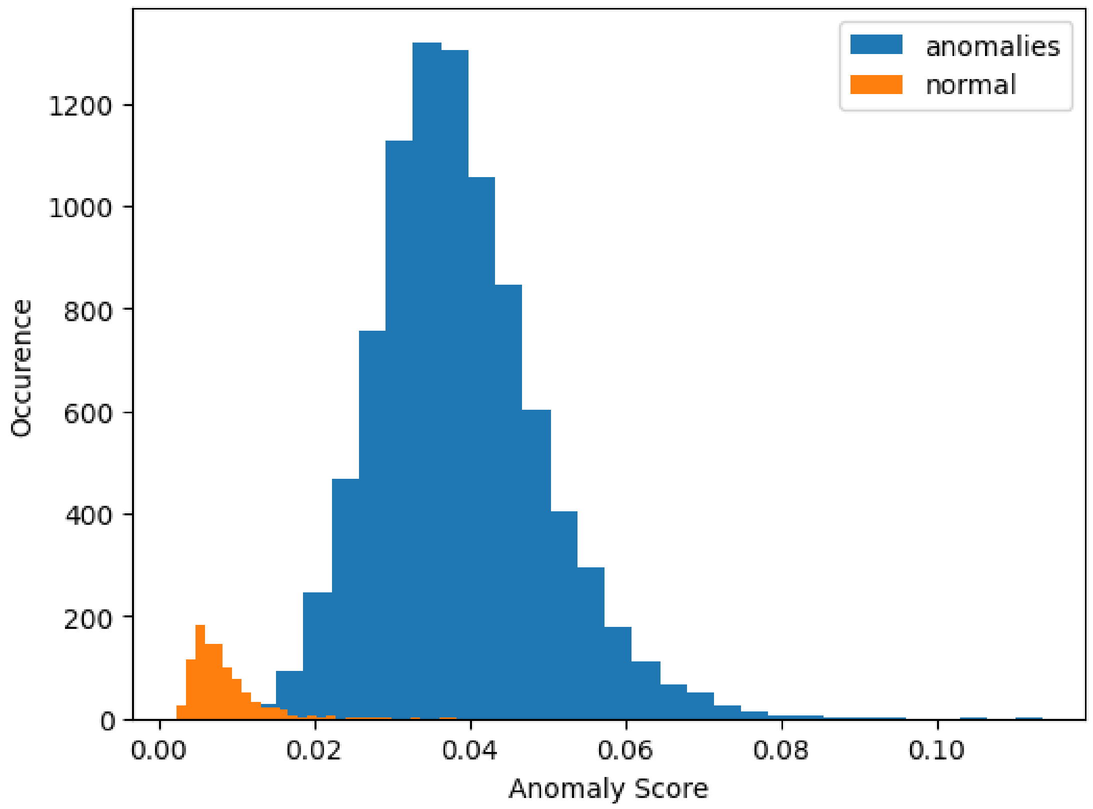
[56]:
```python
#import solution

def anomaly_score(x, reconstruction):
    # --------------------------------------------------
    # TODO: Replace by your code
    # --------------------------------------------------
    #return solution.solution_4_anomaly_score(x, reconstruction)

    #return ((x - reconstruction) ** 2).sum(axis=-1).sum(axis=-1)

    return torch.mean((x - reconstruction) ** 2, dim=(2, 3))
    # --------------------------------------------------
```

6.0.1 Evaluating

For all the samples from the test set, we compute the anomaly score. Then, we want to check how good the anomaly score seperates the normal samples from the anomalies. For that purpose, the `roc_auc_score` can be used. A larger score indicates that the normal samples can be better seperated from the anomalies. We also plot a histogram with the anomaly scores to visualize the separation.

[57]:
```
from sklearn.metrics import roc_auc_score


def compute_anomaly_scores(test_loader, ae):
    anomaly_scores = []
    labels = []
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    ae.to(device)

    for x, y in test_loader:
        x = x.to(device)
        reconstruction = ae(x).detach()

        batch_anomaly_scores = anomaly_score(x, reconstruction)
        # label 0 if sample is normal, otherwise 1
        batch_labels = (y >= threshold_class).cpu().numpy()
        anomaly_scores.append(batch_anomaly_scores.cpu().numpy())
        labels.append(batch_labels)

    anomaly_scores = np.concatenate(anomaly_scores)
    labels = np.concatenate(labels)
    return anomaly_scores, labels


test_loader = DataLoader(test_dataset, batch_size=64)
anomaly_scores, labels = compute_anomaly_scores(test_loader, anomaly_ae)
score = roc_auc_score(labels, anomaly_scores)

print('AUC:', score)
for name, label in [('anomalies', 1), ('normal', 0)]:
    plt.hist(anomaly_scores[labels == label], bins=30, label=name)
plt.ylabel('Occurence')
plt.xlabel('Anomaly Score')
plt.legend()
plt.show()
```

```
AUC: 0.9963164622833612
```

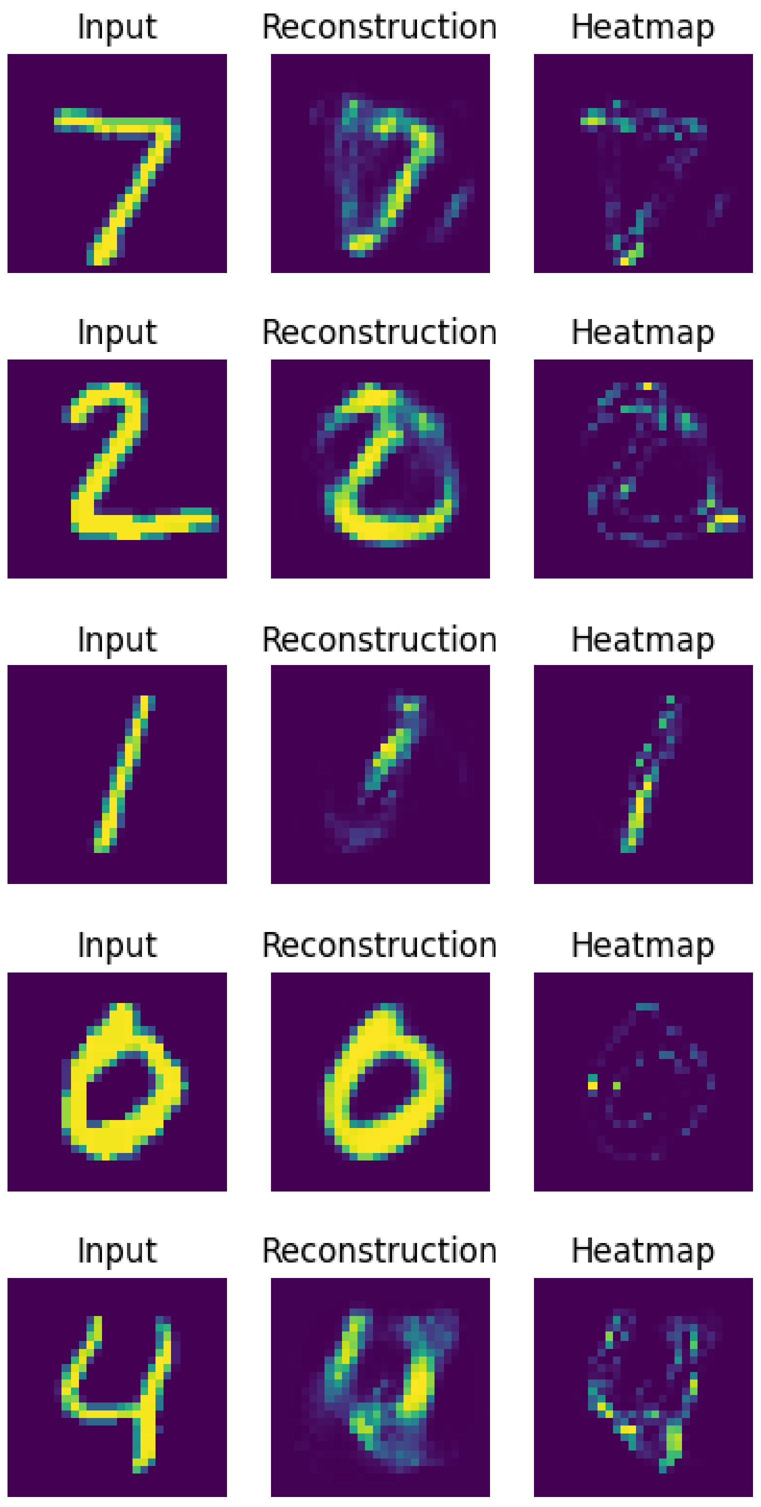


We can see that the zero digit (normal data) and all other digits are well separated with the anomaly score.

6.1 Anomaly Heatmaps

One advantage of the autoencoder anomaly detection method is that we can use the reconstructions as an anomaly heatmap. These show which parts could not be reconstructed well from the model. The code below plots for selected samples the reconstruction and the anomaly heatmap.

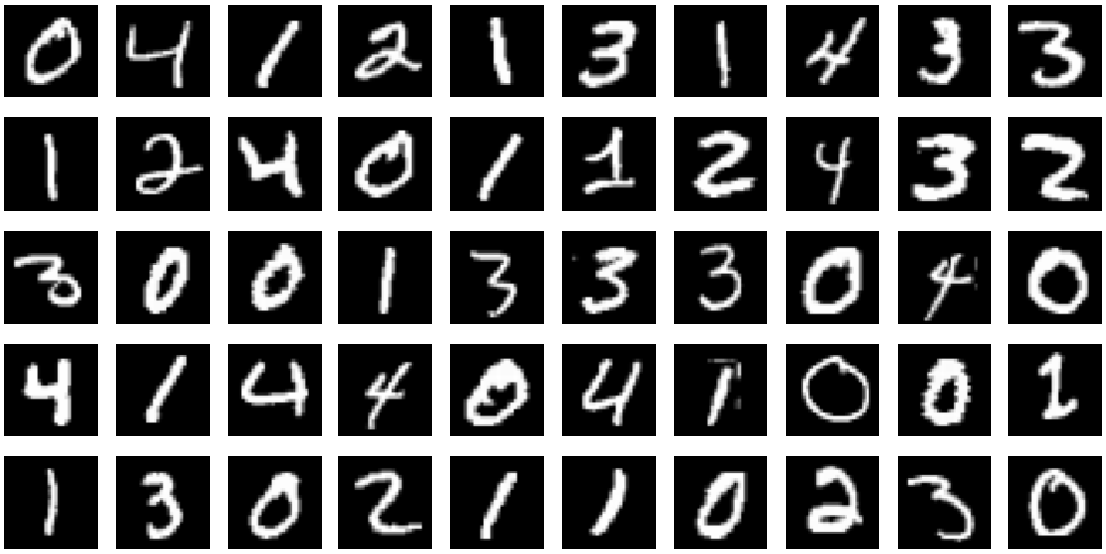
[58]:
```
num_images = 5
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
test_batch = next(iter(test_loader))[0][:num_images].to(device)
reconstructions = anomaly_ae(test_batch).squeeze().detach().cpu()  #     CPU

fig, ax = plt.subplots(num_images, 3)
fig.set_size_inches(5, 10)
for i in range(num_images):
    ax[i][0].imshow(test_batch[i].cpu().numpy()[0])
    ax[i][0].set_title('Input')
    ax[i][1].imshow(reconstructions[i].numpy())
    ax[i][1].set_title('Reconstruction')
    ax[i][2].imshow((reconstructions[i].numpy() - test_batch[i].cpu().
↪numpy()[0])**2)
    ax[i][2].set_title('Heatmap')
    for a in ax[i]:
        a.axis('off')
```


Input
Reconstruction
Heatmap
Input
Reconstruction
Heatmap
Input
Reconstruction
Heatmap
Input
Reconstruction
Heatmap
Input
Reconstruction
Heatmap

7 Latent Space Visualization

Another interesting thing is the latent space of the autoencoder. Here we will visualize it by training a new autoencoder with a hidden dimension of 2, so that we can plot the latent representations of images using matplotlib.

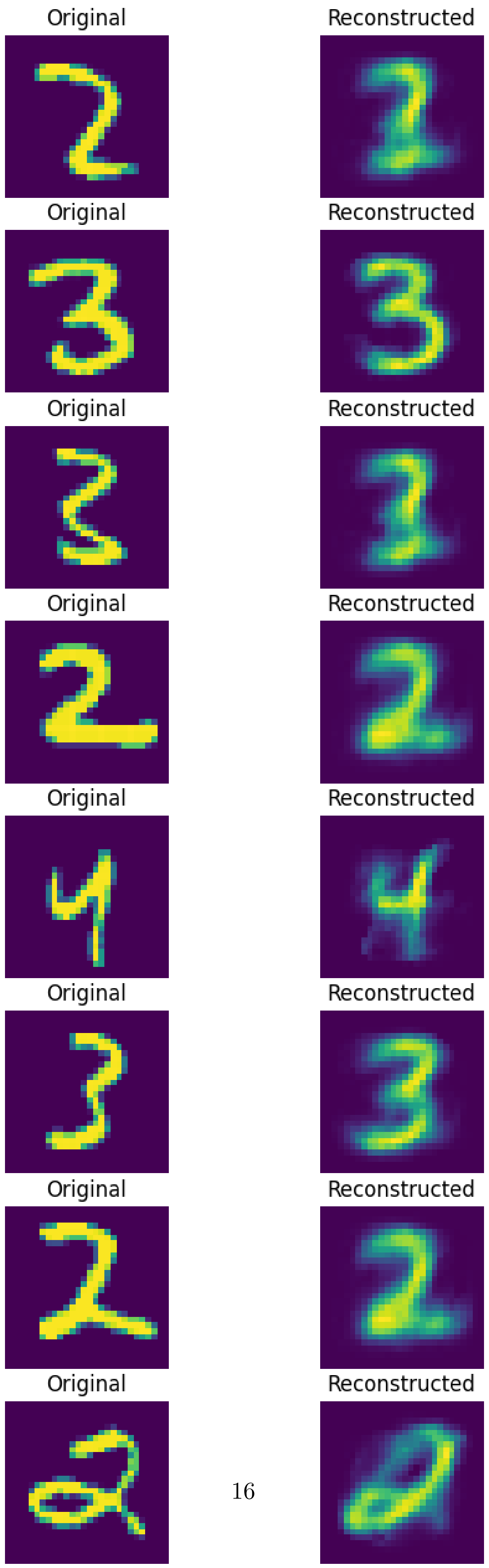
```
[67]: threshold_class = 5
      threshold_class_train = Subset(dataset=copy.deepcopy(train_dataset), indices=np.
        ↪where(train_dataset.targets < threshold_class)[0])
      train_threshold_class_loader = DataLoader(threshold_class_train, batch_size=64,␣
        ↪shuffle=True)
      show_samples(threshold_class_train)

      small_ae = train(train_threshold_class_loader, h=2, epochs=5)
```



```
Epoch   1

  0%|          | 0/479 [00:00<?, ?it/s]

Epoch   2

  0%|          | 0/479 [00:00<?, ?it/s]

Epoch   3

  0%|          | 0/479 [00:00<?, ?it/s]

Epoch   4

  0%|          | 0/479 [00:00<?, ?it/s]

Epoch   5
```

```
0%|          | 0/479 [00:00<?, ?it/s]
```

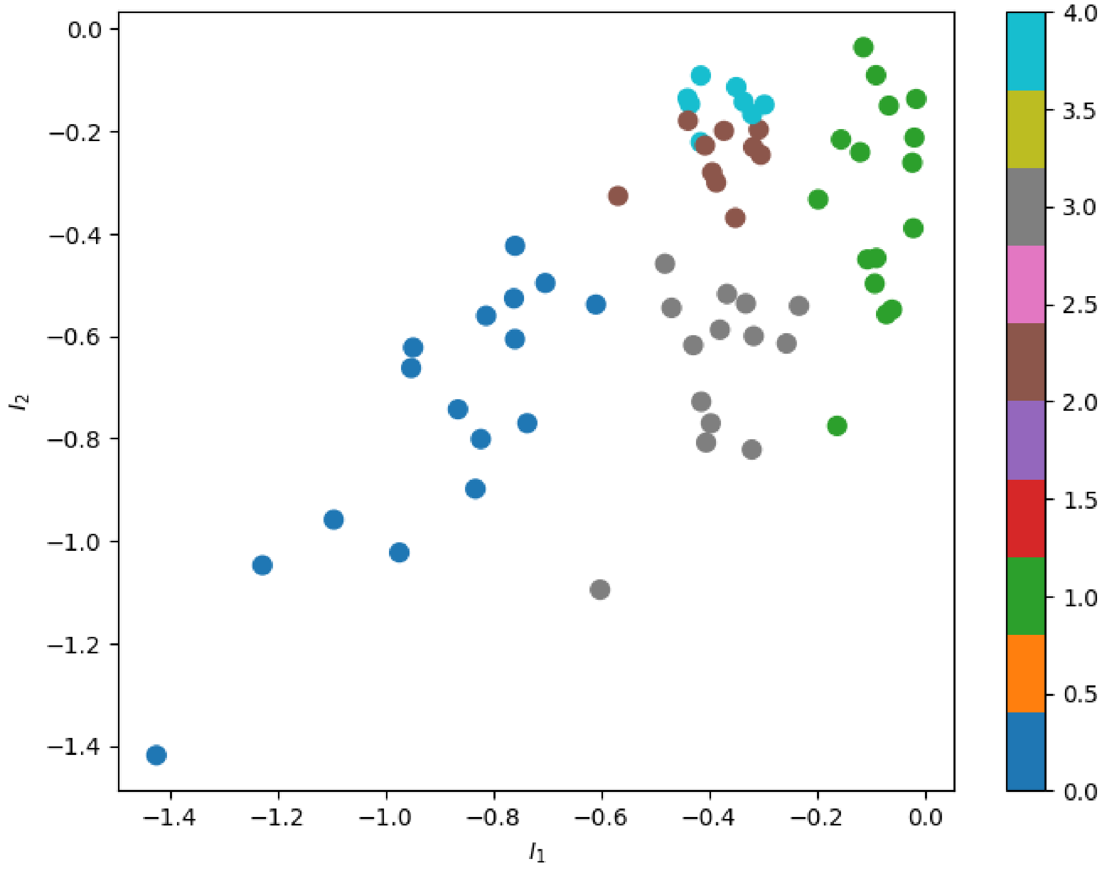
[68]:
```
#solution.solution_3_reconstruct_images(small_ae, train_threshold_class_loader,␣
↪num_images=10)
small_ae.eval()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
small_ae.to(device)

num_samples = 8

test_iter = iter(train_threshold_class_loader)
images, labels = next(test_iter)

fig, ax = plt.subplots(num_samples,2)
fig.set_size_inches(6, 16)

for i in range(num_samples):

    img = images[i].unsqueeze(0).to(device)

    with torch.no_grad():
        reconstructed = small_ae(img)

    ax[i, 0].imshow(images[i][0])
    ax[i, 0].axis('off')
    ax[i, 0].set_title("Original")

    ax[i, 1].imshow(reconstructed.squeeze(0).cpu().numpy()[0])
    ax[i, 1].axis('off')
    ax[i, 1].set_title("Reconstructed")

plt.show()
```


Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed
Original
Reconstructed

8 5) Latent scatter plot (20P)

In this task you should plot the latent representations. You can use the labels as color information using `plt.scatter(c=labels)`.

[71]:
```
#import solution
# ----------------------------------------------------
# TODO: Replace by your code
# ----------------------------------------------------
#solution.solution_5_latent_scatter_plot(small_ae, train_threshold_class_loader)
import matplotlib.pyplot as plt
import numpy as np


small_ae.eval()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
small_ae.to(device)

test_iter = iter(train_threshold_class_loader)
images, labels = next(test_iter)

images = images.to(device)
with torch.no_grad():
    latent_representations = small_ae.encoder(images.view(images.size(0), -1))

latent_representations = latent_representations.cpu().numpy()
labels = labels.numpy()



plt.figure(figsize=(8, 6))
plt.scatter(latent_representations[:, 0], latent_representations[:, 1],␣
 ↪c=labels, cmap='tab10',s=60)
plt.colorbar()
plt.xlabel("$I_1$")
plt.ylabel("$I_2$")
plt.show()

# ----------------------------------------------------
```



l_1
l_2