```
from torch.utils.data import DataLoader,Dataset
import torch
import torchvision.transforms as transforms
from torchvision import datasets
import torchvision.models as models
import torchvision
import matplotlib.pyplot as plt
import numpy as np
from tqdm import tqdm


device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

```
train_data = datasets.MNIST(root='./data',train=True,download=True,transform=tc
test_data = datasets.MNIST(root='./data',train=False,download=True,transform=tc
train_loader = DataLoader(train_data,batch_size=64,shuffle=True)
test_loader = DataLoader(test_data,batch_size=64,shuffle=True)


class Autoencoder(torch.nn.Module):
    def __init__(self):
        super(Autoencoder,self).__init__()
        self.relu = torch.nn.ReLU()
        self.sigmoid = torch.nn.Sigmoid()
        self.flatten = torch.nn.Flatten()
        self.unflatten = torch.nn.Unflatten(1,(1,28,28))
        self.fc1 = torch.nn.Linear(784,256)
        self.fc2 = torch.nn.Linear(256,256)
        self.fc3 = torch.nn.Linear(256,256)
        self.fc4 = torch.nn.Linear(256,128)
        self.fc5 = torch.nn.Linear(128,256)
        self.fc6 = torch.nn.Linear(256,256)
        self.fc7 = torch.nn.Linear(256,256)
        self.fc8 = torch.nn.Linear(256,784)

    def forward(self,x):
        x = self.encode(x)
        x = self.decode(x)
        return x
    def encode(self,x):
        x = self.flatten(x)
        x = self.fc1(x)
        x = self.relu(x)
        x = self.fc2(x)
        x = self.relu(x)
        x = self.fc3(x)
        x = self.relu(x)
        x = self.fc4(x)
        x = self.relu(x)
        return x
```

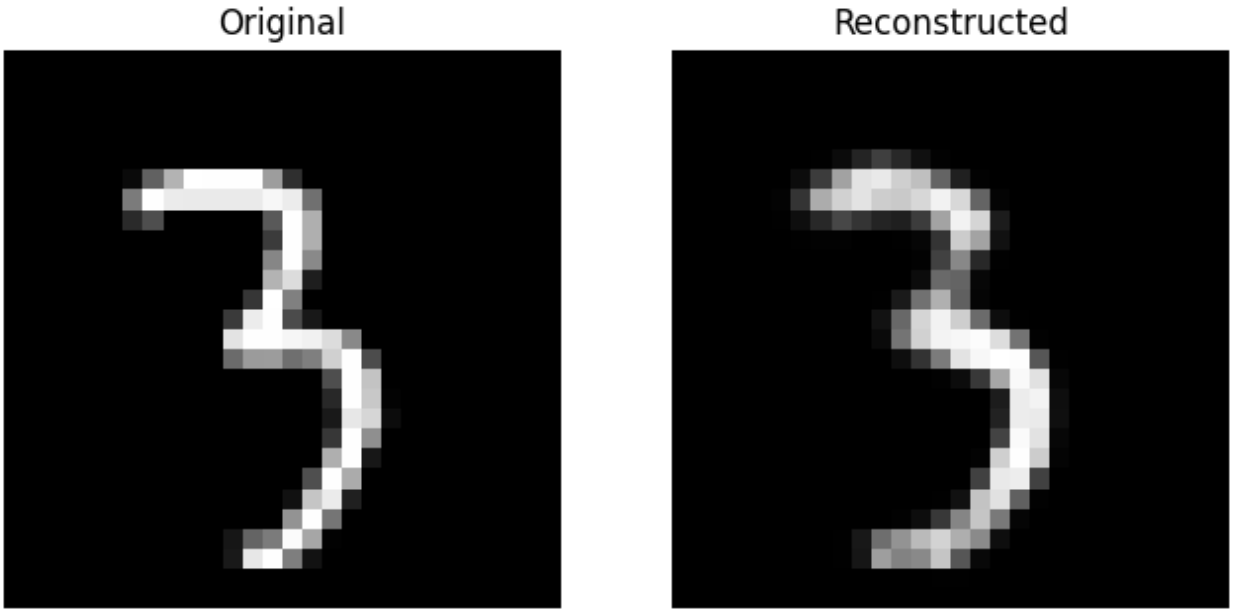
```
        return x
    def decode(self,x):
        x = self.relu(x)
        x = self.fc5(x)
        x = self.relu(x)
        x = self.fc6(x)
        x = self.relu(x)
        x = self.fc7(x)
        x = self.relu(x)
        x = self.fc8(x)
        x = self.sigmoid(x)
        x = self.unflatten(x)
        return x


model = Autoencoder().to(device)
epochs = 10
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
criterion = torch.nn.MSELoss()


for epoch in range(epochs):
    for data in tqdm(train_loader):
        img,label = data
        img = img.to(device)
        output = model(img)
        loss = criterion(output,img)
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    print(f'epoch: {epoch}, loss: {loss.item()}')
```

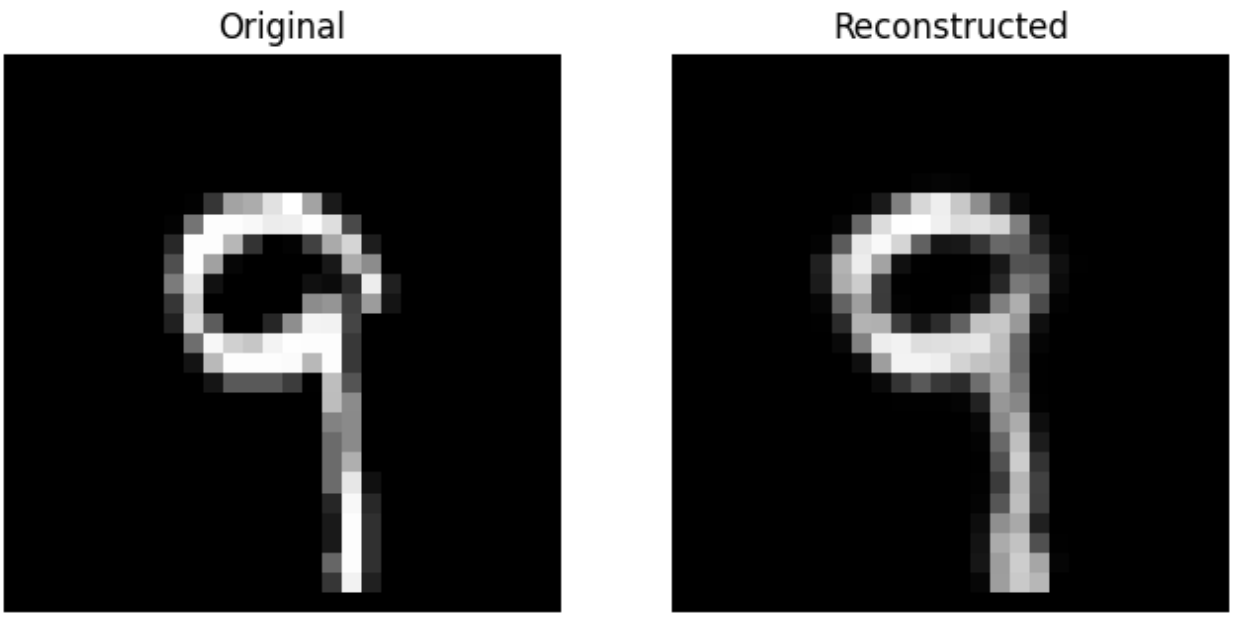
```
100%|██████████| 938/938 [00:09<00:00, 102.24it/s]
epoch: 0, loss: 0.03282281011343002
100%|██████████| 938/938 [00:08<00:00, 108.72it/s]
epoch: 1, loss: 0.0268026664853096
100%|██████████| 938/938 [00:08<00:00, 110.79it/s]
epoch: 2, loss: 0.0216627549380064
100%|██████████| 938/938 [00:07<00:00, 120.39it/s]
epoch: 3, loss: 0.017871670424938202
100%|██████████| 938/938 [00:08<00:00, 114.15it/s]
epoch: 4, loss: 0.014790568500757217
100%|██████████| 938/938 [00:08<00:00, 110.28it/s]
epoch: 5, loss: 0.014715279452502728
100%|██████████| 938/938 [00:07<00:00, 119.05it/s]
epoch: 6, loss: 0.014392596669495106
100%|██████████| 938/938 [00:08<00:00, 110.61it/s]
epoch: 7, loss: 0.010834861546754837
100%|██████████| 938/938 [00:08<00:00, 105.58it/s]
epoch: 8, loss: 0.012006720528006554
100%|██████████| 938/938 [00:09<00:00, 101.45it/s]epoch: 9, loss: 0.0124042
```

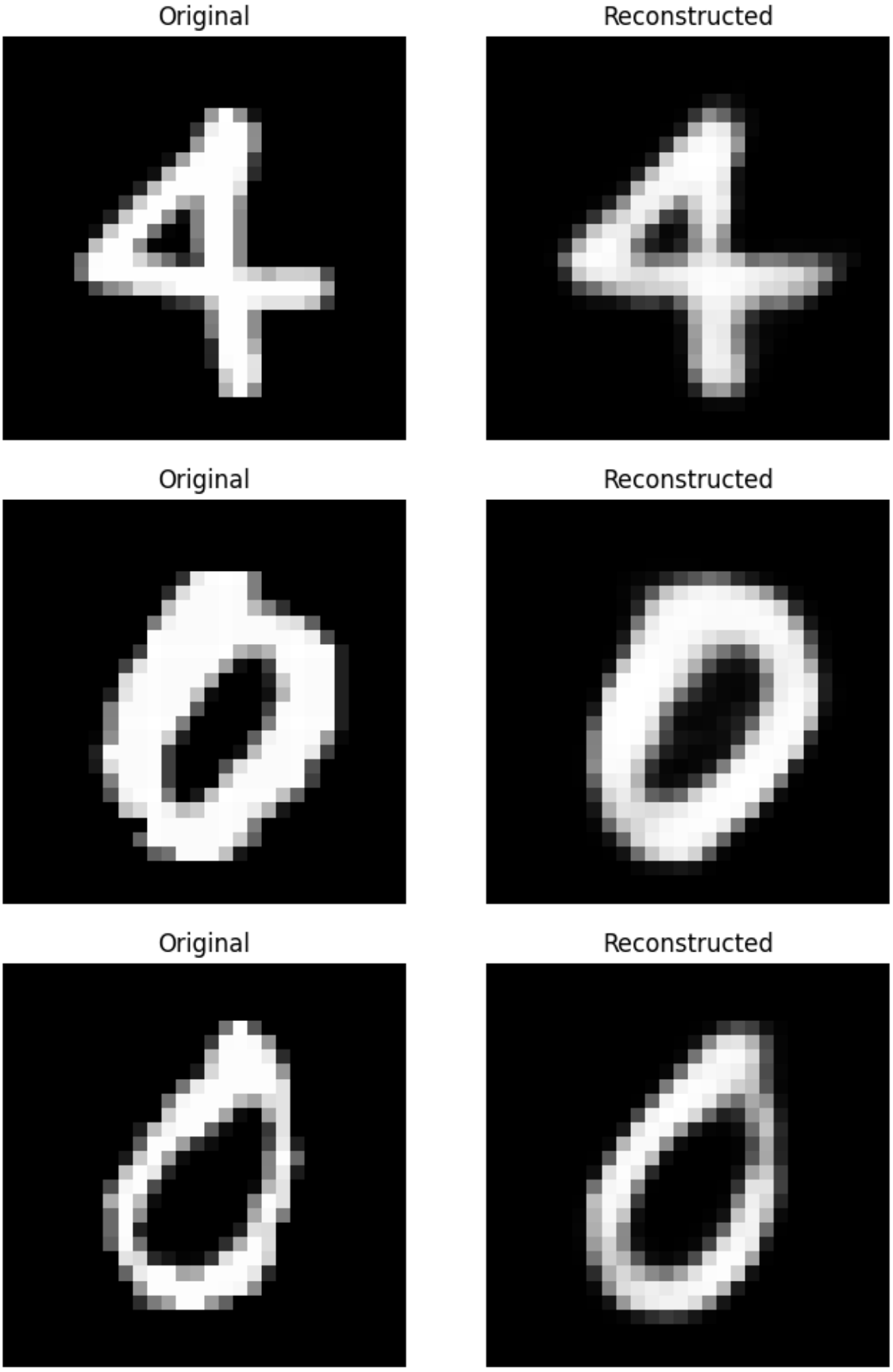
```
data_iter = iter(train_loader)
images,labels = next(data_iter)

with torch.no_grad():
```

```
with torch.no_grad():
    output = model(images.to(device))
    output = output.cpu()
    output = output.numpy()
    print(output.shape)
    output = np.reshape(output,(64,28,28))



for j in range(5):
    image = images[j].numpy()
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))
    axes[0].imshow(image[0], cmap='gray')
    axes[0].set_title('Original')
    axes[0].axis('off')
    axes[1].imshow(output[j], cmap='gray')
    axes[1].set_title('Reconstructed')
    axes[1].axis('off')
    plt.show()
```

```
(64, 1, 28, 28)
```






Original
Reconstructed
Original
Reconstructed
Original
Reconstructed