```
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 VarAutoencoder(torch.nn.Module):
    def __init__(self):
        super(VarAutoencoder,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.mu = torch.nn.Linear(128,128)
        self.log_var = torch.nn.Linear(128,128)
        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,mu,log_var = self.encode(x)
        x = self.reparameterize(mu,log_var)
        x = self.decode(x)
        return x,mu,log_var
    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)
        mu = self.mu(x)
        log_var = self.log_var(x)
        return x,mu,log_var
    def decode(self,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
    def reparameterize(self,mu,logvar):
        std = torch.exp(0.5*logvar)
        eps = torch.randn_like(std)
        return mu + eps*std


model = VarAutoencoder().to(device)
epochs = 10
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
criterion = torch.nn.BCELoss(reduction='sum')


for epoch in range(epochs):
    train_loss = 0.0
    for data in tqdm(train_loader):
        img,label = data
        img = img.to(device)
        output,mu,log_var = model(img)
        bce_loss = criterion(output,img)
        kld_loss = -0.5 * torch.sum(1 + log_var - mu.pow(2) - log_var.exp())
        loss = bce_loss + kld_loss
        train_loss += loss.item()
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
    print(f'epoch: {epoch}, loss: {train_loss}')
```

```
100%|██████████| 938/938 [00:11<00:00, 82.52it/s]
epoch: 0, loss: 11479841.763183594
100%|██████████| 938/938 [00:10<00:00, 93.69it/s]
epoch: 1, loss: 9250264.611328125
100%|██████████| 938/938 [00:09<00:00, 96.62it/s]
epoch: 2, loss: 8386616.513183594
100%|██████████| 938/938 [00:09<00:00, 96.70it/s]
epoch: 3, loss: 8065842.874511719
100%|██████████| 938/938 [00:09<00:00, 100.67it/s]
epoch: 4, loss: 7903603.431640625
100%|██████████| 938/938 [00:09<00:00, 94.56it/s]
epoch: 5, loss: 7799225.8408203125
100%|██████████| 938/938 [00:09<00:00, 97.52it/s]
```

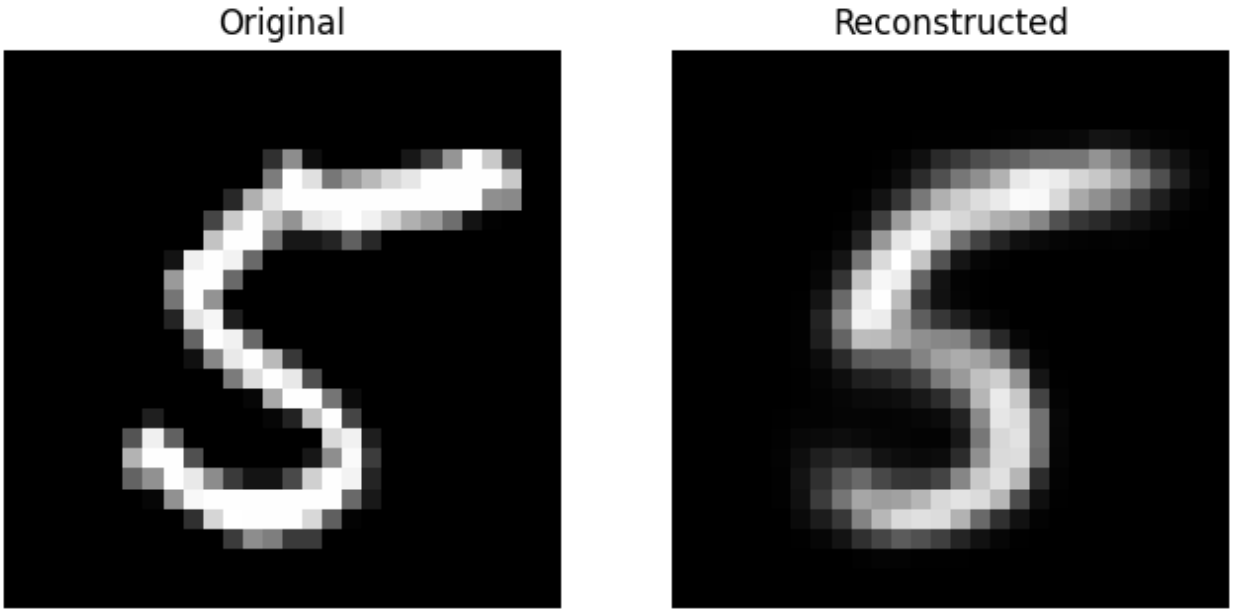
```
100%|██████████| 938/938 [00:09<00:00, 97.52it/s]
epoch: 6, loss: 7708976.699951172
100%|██████████| 938/938 [00:09<00:00, 97.91it/s]
epoch: 7, loss: 7640695.8779296875
100%|██████████| 938/938 [00:10<00:00, 90.15it/s]
epoch: 8, loss: 7557496.510253906
100%|██████████| 938/938 [00:09<00:00, 98.63it/s] epoch: 9, loss: 7457637.0
```

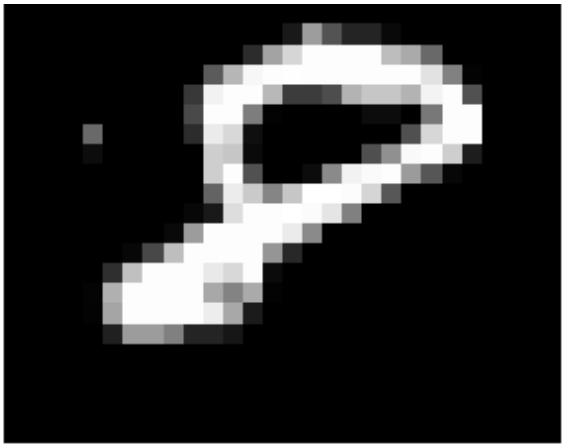
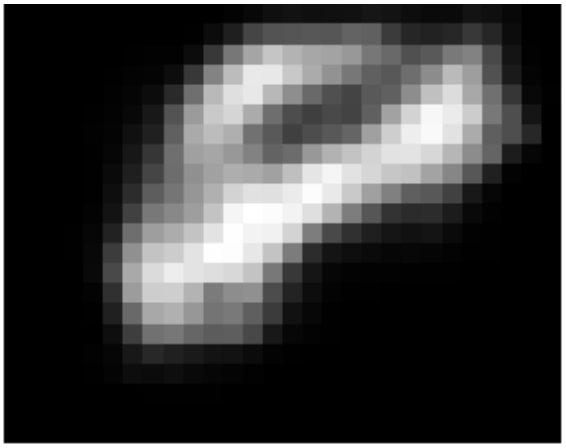
```
data_iter = iter(train_loader)
images,labels = next(data_iter)

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

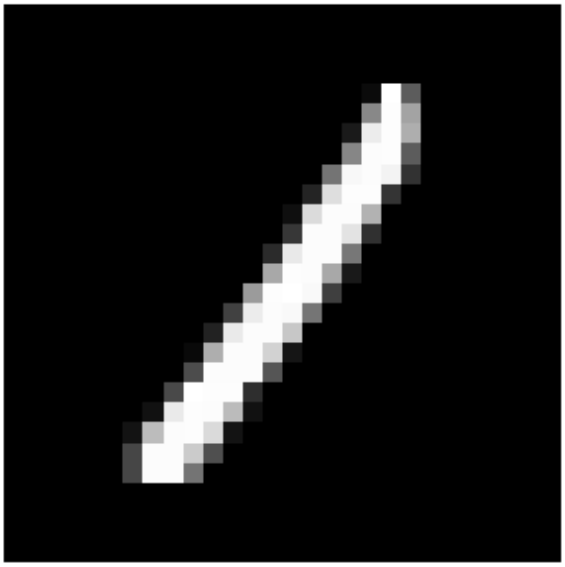
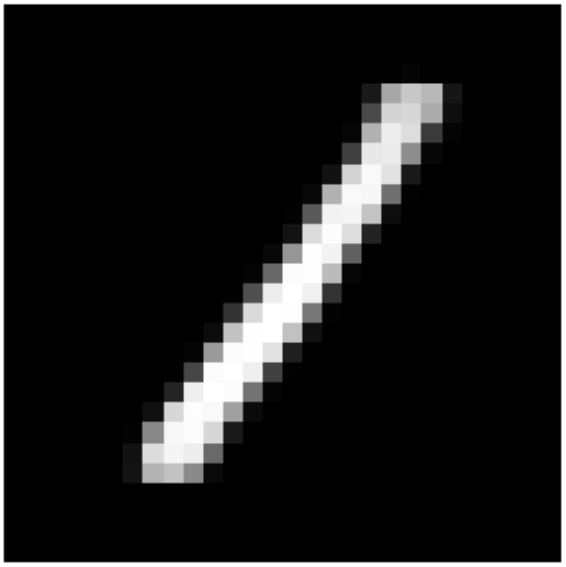
```
for j in range(5):
    image = images[j].numpy()
    fig, axes = plt.subplots(1, 2, figsize=(8, 4))
    axes[0].imshow(image[0], cmap='gray')
    axes[0].set_title('Original')
    axes[0].axis('off')
    axes[1].imshow(output[j], cmap='gray')
    axes[1].set_title('Reconstructed')
    axes[1].axis('off')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

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

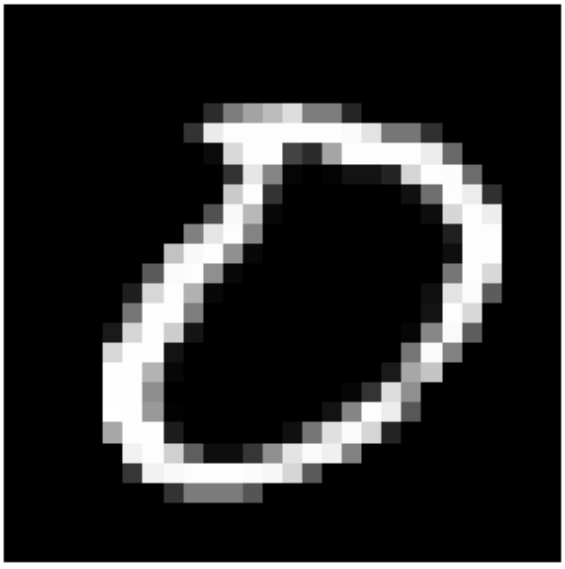
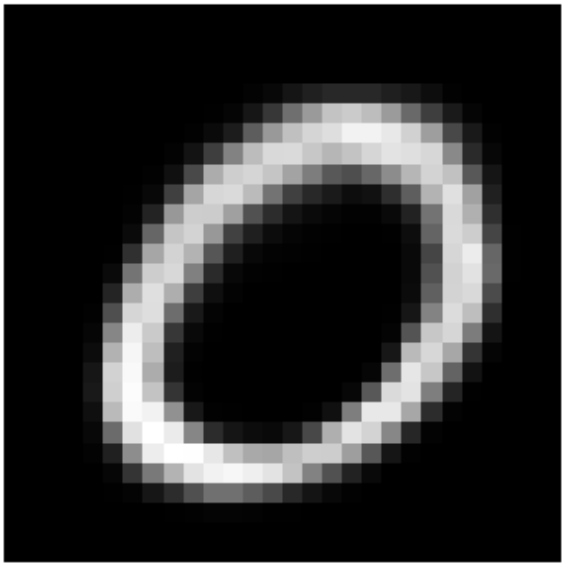


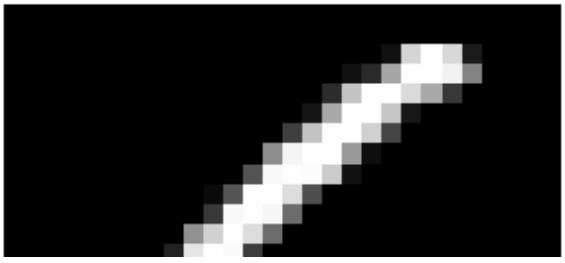
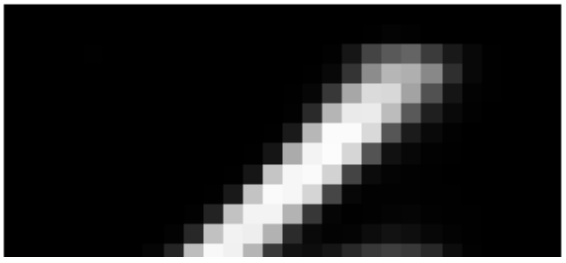
Original Reconstructed

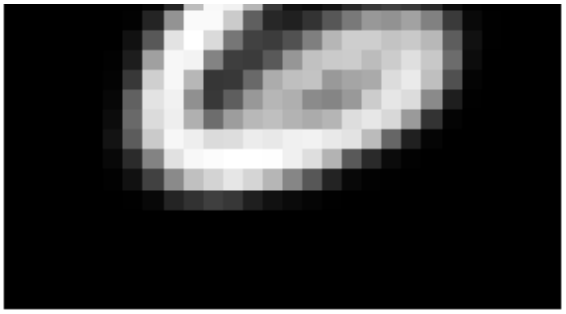



Original Reconstructed

Original Reconstructed



```
with torch.no_grad():
    z = torch.randn(16, 128).to(device)
    samples = model.decode(z).cpu().view(-1, 28, 28)

fig, axes = plt.subplots(4, 4, figsize=(5, 5))
for i, ax in enumerate(axes.flatten()):
    ax.imshow(samples[i], cmap="gray")
    ax.axis("off")
plt.show()
```