```
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=tor
test_data = datasets.MNIST(root='./data',train=False,download=True,transform=tor
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.softmax = torch.nn.Softmax()
        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)
        self.fc9 = torch.nn.Linear(128,10)

    def forward(self,x):
        xe = self.encode(x)
        x1 = self.decode(xe)
        y_pred = self.classify(xe)
        return x1,y_pred
    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
    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
    def classify(self,x):
        x = self.fc9(x)
        x = self.softmax(x)
        return x


model = Autoencoder().to(device)
epochs = 10
optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
criterion = torch.nn.MSELoss()
criterion2 = torch.nn.CrossEntropyLoss()


for epoch in range(epochs):
    correct = 0
    total = 0
    for data in tqdm(train_loader):
        img,label = data
        img = img.to(device)
        label = label.to(device)
        output,y_pred = model(img)
        loss = criterion(output,img) + criterion2(y_pred,label)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        _,predicted = torch.max(y_pred.data,1)
        total += label.size(0)
        correct += (predicted == label).sum().item()
    print(f'epoch: {epoch}, loss: {loss.item()}, Accuracy: {correct*100//total}
```

```
  0%|          | 0/938 [00:00<?, ?it/s]/usr/local/lib/python3.11/dist-packa
  return self._call_impl(*args, **kwargs)
100%|██████████| 938/938 [00:09<00:00, 101.56it/s]
epoch: 0, loss: 1.6411274671554565, Accuracy: 82
100%|██████████| 938/938 [00:08<00:00, 113.73it/s]
epoch: 1, loss: 1.6092846393585205, Accuracy: 93
100%|██████████| 938/938 [00:08<00:00, 105.08it/s]
epoch: 2, loss: 1.554251790046692, Accuracy: 94
100%|██████████| 938/938 [00:08<00:00, 105.19it/s]
epoch: 3, loss: 1.4880242347717285, Accuracy: 95
```

```
100%|██████████| 938/938 [00:08<00:00, 113.33it/s]
epoch: 4, loss: 1.5409038066864014, Accuracy: 95
100%|██████████| 938/938 [00:08<00:00, 105.49it/s]
epoch: 5, loss: 1.6112562417984009, Accuracy: 95
100%|██████████| 938/938 [00:08<00:00, 105.12it/s]
epoch: 6, loss: 1.5471490621566772, Accuracy: 95
100%|██████████| 938/938 [00:08<00:00, 106.52it/s]
epoch: 7, loss: 1.5513253211975098, Accuracy: 95
100%|██████████| 938/938 [00:08<00:00, 112.88it/s]
epoch: 8, loss: 1.5851624011993408, Accuracy: 95
100%|██████████| 938/938 [00:08<00:00, 106.50it/s]epoch: 9, loss: 1.5168906
```

```
correct = 0
total = 0
for i,(input,label) in enumerate(test_loader):
  input = input.to(device)
  label = label.to(device)
  output,y_pred = model(input)
  _,predicted = torch.max(y_pred.data,1)
  total += label.size(0)
  correct += (predicted == label).sum().item()

print(f'Test Accuracy: {(correct/total)*100}%')
```

```
/usr/local/lib/python3.11/dist-packages/torch/nn/modules/module.py:1736: Us
  return self._call_impl(*args, **kwargs)
Test Accuracy: 94.08999999999999%
```