```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torchvision import datasets
import torch.optim as optim
from torchvision.transforms import transforms
from torchvision.utils import save_image

import numpy as np
import matplotlib.pyplot as plt

lr = 0.001
batch_size = 100
epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

'''
Step 1:
'''

# MNIST dataset
dataset = datasets.MNIST(root='./mnist_data/',
                         train=True,
                         transform=transforms.ToTensor(),
                         download=True)

train_dataset, validation_dataset = torch.utils.data.random_split(dataset, [50000, 10000])

test_dataset = datasets.MNIST(root='./mnist_data/',
                        train=False,
                        transform=transforms.ToTensor())

# KMNIST dataset, only need test dataset
anomaly_dataset = datasets.KMNIST(root='./kmnist_data/',
                        train=False,
                        transform=transforms.ToTensor(),
                        download=True)

# print(len(train_dataset))  # 50000
# print(len(validation_dataset))  # 10000
# print(len(test_dataset))  # 10000
# print(len(anomaly_dataset))  # 10000

'''
Step 2: AutoEncoder
'''
# Define Encoder
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(784, 256)
        self.fc2 = nn.Linear(256, 128)
```

```
        self.fc3 = nn.Linear(128, 32)
    def forward(self, x):
        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        z = F.relu(self.fc3(x))
        return z

# Define Decoder
class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(32, 128)
        self.fc2 = nn.Linear(128, 256)
        self.fc3 = nn.Linear(256, 784)
    def forward(self, z):
        z = F.relu(self.fc1(z))
        z = F.relu(self.fc2(z))
        x = F.sigmoid(self.fc3(z))  # to make output's pixels are 0~1
        x = x.view(x.size(0), 1, 28, 28)
        return x

'''
Step 3: Instantiate model & define loss and optimizer
'''
enc = Encoder().to(device)
dec = Decoder().to(device)
loss_function = nn.MSELoss()
optimizer = optim.Adam(list(enc.parameters()) + list(dec.parameters()), lr=lr)


'''
Step 4: Training
'''
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True)


train_loss_list = []

import time
start = time.time()
for epoch in range(epochs) :
    print("{}th epoch starting.".format(epoch))
    enc.train()
    dec.train()
    for batch, (images, _) in enumerate(train_loader) :
        images = images.to(device)
        z = enc(images)
        reconstructed_images = dec(z)

        optimizer.zero_grad()
        train_loss = loss_function(images, reconstructed_images)
        train_loss.backward()
```

```
        train_loss_list.append(train_loss.item())

        optimizer.step()

          print(f"[Epoch {epoch:3d}] Processing batch #{batch:3d} reconstruction loss: {train_loss.item():.6f}",
end='\r')
end = time.time()
print("Time ellapsed in training is: {}".format(end - start))

# plotting train loss
plt.plot(range(1,len(train_loss_list)+1), train_loss_list, 'r', label='Training loss')
plt.title('Training loss')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.legend()
plt.savefig('loss.png')

enc.eval()
dec.eval()

'''
Step 5: Calculate standard deviation by using validation set
'''
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_size=batch_size)

val_score = []

for images, _ in validation_loader:
    images = images.to(device)
    z = enc(images)
    reconstructed_images = dec(z)
    diff = (images - reconstructed_images) ** 2
    diff = diff.reshape(batch_size, -1)
    val_score += torch.sum(diff, axis=1)


val_score_np = []

for i in val_score:
    val_score_np.append(i.to('cpu').detach().numpy())

mean = np.mean(val_score_np)
std = np.std(val_score_np)
threshold = mean + 3 * std
print("threshold: ", threshold)


'''
Step 6: Anomaly detection (mnist)
'''
test_loader = torch.utils.data.DataLoader(dataset=test_dataset, batch_size=batch_size)

mnist_anomalies = 0
```

```
mnist_val_score = []

for images, _ in test_loader:
    images = images.to(device)
    z = enc(images)
    reconstructed_images = dec(z)
    diff = (images - reconstructed_images) ** 2
    diff = diff.reshape(batch_size, -1)
    mnist_anomalies+=sum((diff.sum(dim=1)>threshold).int())


'''
Step 7: Anomaly detection (kmnist)
'''
anomaly_loader = torch.utils.data.DataLoader(dataset=anomaly_dataset, batch_size=batch_size)

kmnist_anomalies = 0

for images, _ in anomaly_loader:
    images = images.to(device)
    z = enc(images)
    reconstructed_images = dec(z)
    diff = (images - reconstructed_images) ** 2
    diff = diff.reshape(batch_size, -1)
    kmnist_anomalies+=sum((diff.sum(dim=1)>threshold).int())

print('type I error rate:',int(mnist_anomalies)/len(test_dataset)*100)
print('type II error rate:',int(kmnist_anomalies)/len(anomaly_dataset) * 100)
```

Result

0th epoch starting.

1th epoch starting.

2th epoch starting.

3th epoch starting.

4th epoch starting.

5th epoch starting.

6th epoch starting.

7th epoch starting.

8th epoch starting.

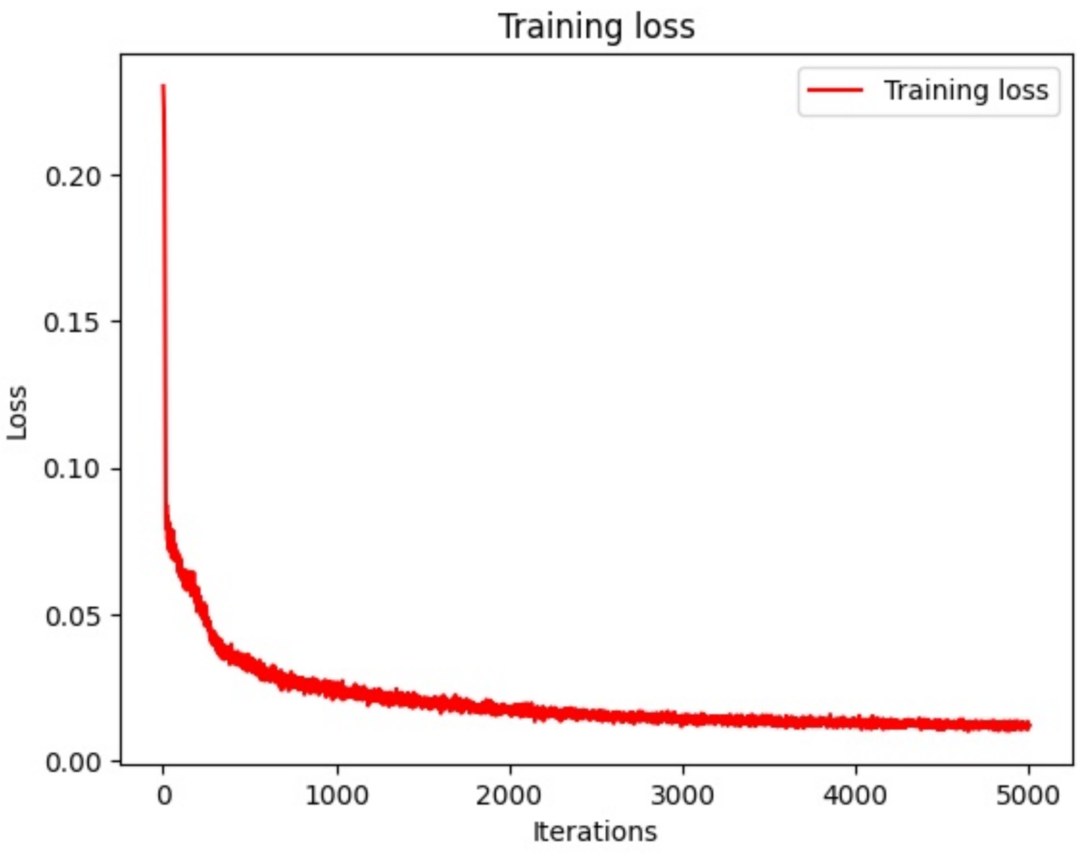
9th epoch starting.

Time ellapsed in training is: 73.14015173912048

threshold: 26.504478454589844

type I error rate: 1.03

type II error rate: 96.54


Training loss
Training loss
0.20
0.15
0.10
0.05
0.00
Loss
0
1000
2000
3000
4000
5000
Iterations