# A7 特征学习

## 一、任务目标

### 1、基本实验要求完成：

a. **任务1. PCA 或 kernel PCA (10 points)**

b. **使用PCA或kernel PCA对手写数字数据集MINST进行降维。观察前两个特征向量所对应的图像，即将数据嵌入到R2空间。绘制降维后的数据，并分析二维特征是否能够足以完成对输入的分类，对结果进行分析和评价。**

c. **任务2. Autoencoder (10 points)**

d. **使用自动编码器学习输入的特征表示。尝试设计一个全链接前馈神经网络或卷积神经网络。尝试使用不同的损失函数和正则化方法。**

### 2、进阶要求：

a. **模型训练中，你可以尝试任何可以提升模型性能的合理的方法。例如其它的网络结构、设计多个隐藏层、引入降噪自动编码器等任何你能想到的方法。计算模型在训练集和测试集上的损失，并对结果进行讨论。**

## 二、实验实现过程及结果：

### 1、实验平台概述：

本次实验采用 Python 完成。

依赖库有：

| 依赖库 | 版本 |
| --- | --- |
| numpy | 1.24.3 |
| torch | 1.13.0+cu116 |
| matplotlib | 3.6.0 |
| sklearn | 1.3.2 |

## 2、任务代码实现及实现效果：

### ① PCA数据降维可视化及二维特征分类尝试

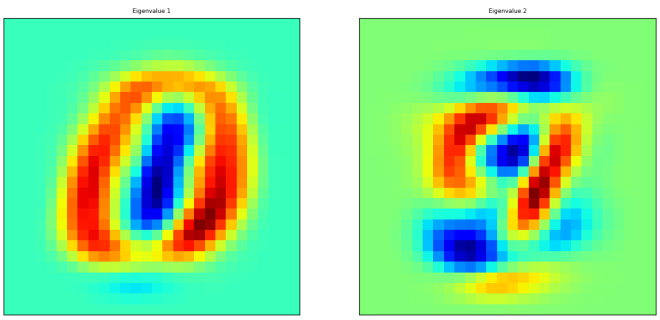
为使用sklearn中PCA方法，先将MINIST数据集中手写数字数据集转为numpy数组类型，并将其一维化得到(784,)大小数据，之后将所有数据进行PCA降维，并截取前2维度特征向量进行可视化。

具体代码如下：

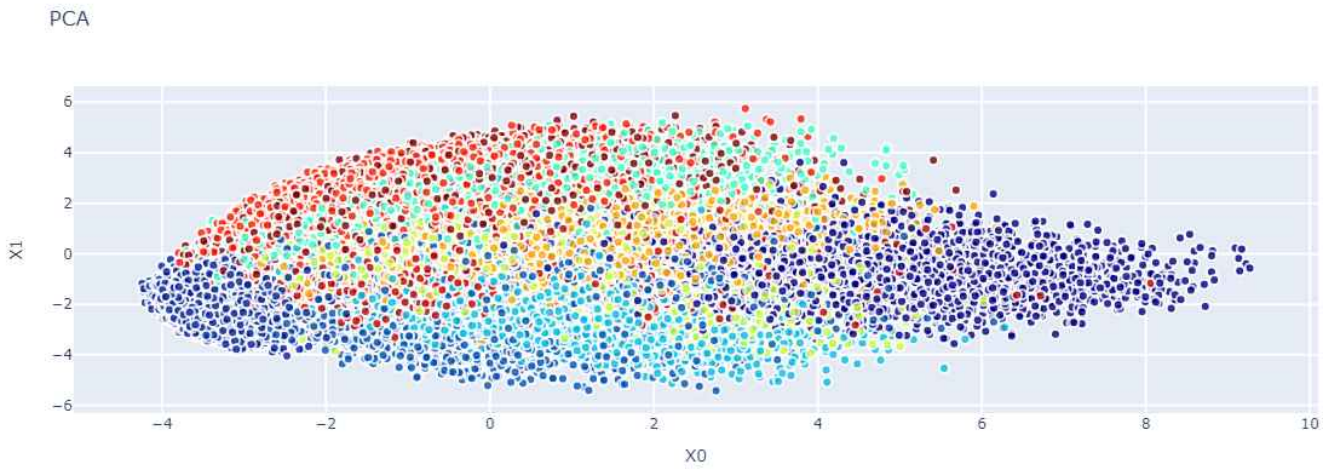
```

train_dataset=torchvision.datasets.MNIST(root="./dataset",train=True,transform=t
test_dataset=torchvision.datasets.MNIST(root="./dataset", transform=transform,tr

pca_train_datasets = []
train_label = []
pca_test_datasets = []
test_label = []

for i in range(len(train_dataset)):
    # print(train_dataset[i][0].cpu().numpy().reshape(-1).shape)
    pca_train_datasets.append(train_dataset[i][0].cpu().numpy().reshape(-1))
    y_onehot = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    y_onehot[train_dataset[i][1]] = 1
    train_label.append(y_onehot)

for i in range(len(test_dataset)):
    pca_test_datasets.append(test_dataset[i][0].cpu().numpy().reshape(-1))
    y_onehot = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
    y_onehot[test_dataset[i][1]] = 1
    test_label.append(y_onehot)


# 调用 SKlearn的 PCA 方法
n_components = 2
pca = PCA(n_components=n_components).fit(pca_train_datasets)

eigenvalues = pca.components_.reshape(n_components, 28, 28)

#提取PCA主成分（特征值），仔细想想应该是特征向量
eigenvalues = pca.components_

#画图
n_row = 1
n_col = 2

```

```
# Plot the first 8 eignenvalues
plt.figure(figsize=(13,12))
for i in list(range(n_row * n_col)):
    offset =0
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(eigenvalues[i].reshape(28,28), cmap='jet')
    title_text = 'Eigenvalue ' + str(i + 1)
    plt.title(title_text, size=6.5)
    plt.xticks(())
    plt.yticks(())
plt.show()
plt.savefig('./pca_minst.png')
```

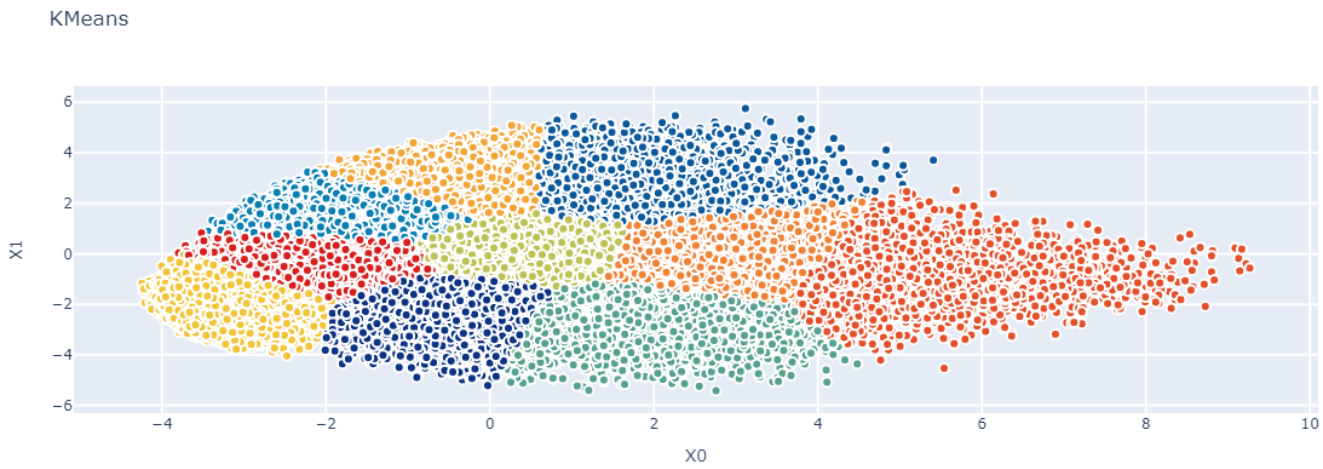
有可视化结果如下：




再利用前二维度特征向量绘制降维后数据分布图有（点颜色由标签类型决定）：




由于PCA无法对其进行分类，我们先尝试用Kmeans对二维特征进行10分类并绘制：




但由于Kmeans为无监督的学习方法，不易确认其真实分类效果。故接下来构建了使用PCA降维后数据进行训练MLP分类器的验证：

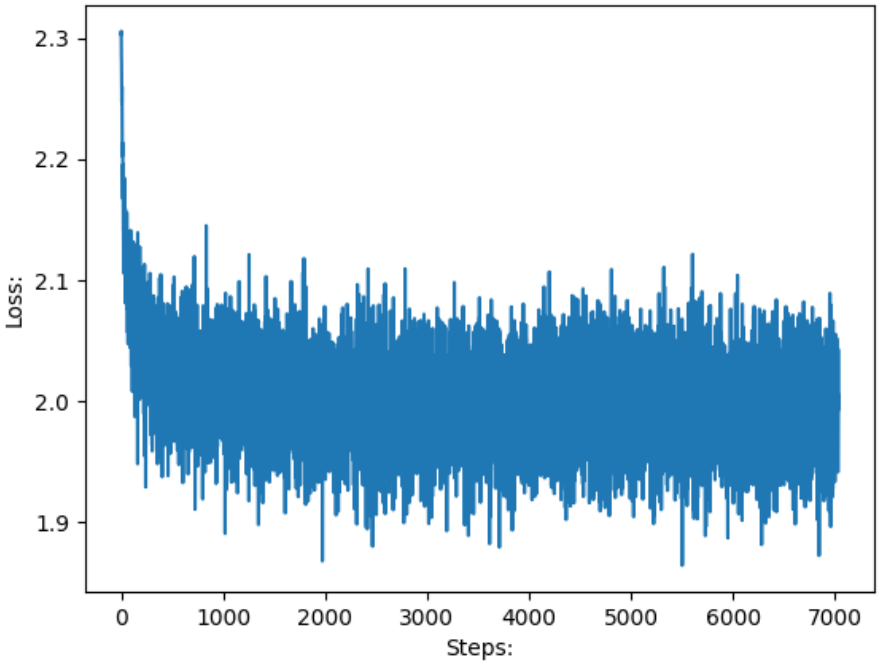
搭建一个简易的MLP模型(2-256-10)进行分类，有代码如下：

```
1
2  pca_train_datasets = pca.transform(pca_train_datasets)
3  pca_test_datasets = pca.transform(pca_test_datasets)
4
5  train_dataloader = []
6  test_dataloader = []
7  batch_size = 128
8
9  train_dataset = Data.TensorDataset(torch.tensor(pca_train_datasets, dtype=torch.
10 test_dataset = Data.TensorDataset(torch.tensor(pca_test_datasets, dtype=torch.fl
11
12 train_dataloader = Data.DataLoader(dataset=train_dataset, batch_size=batch_size,
13                                    num_workers=8)
```

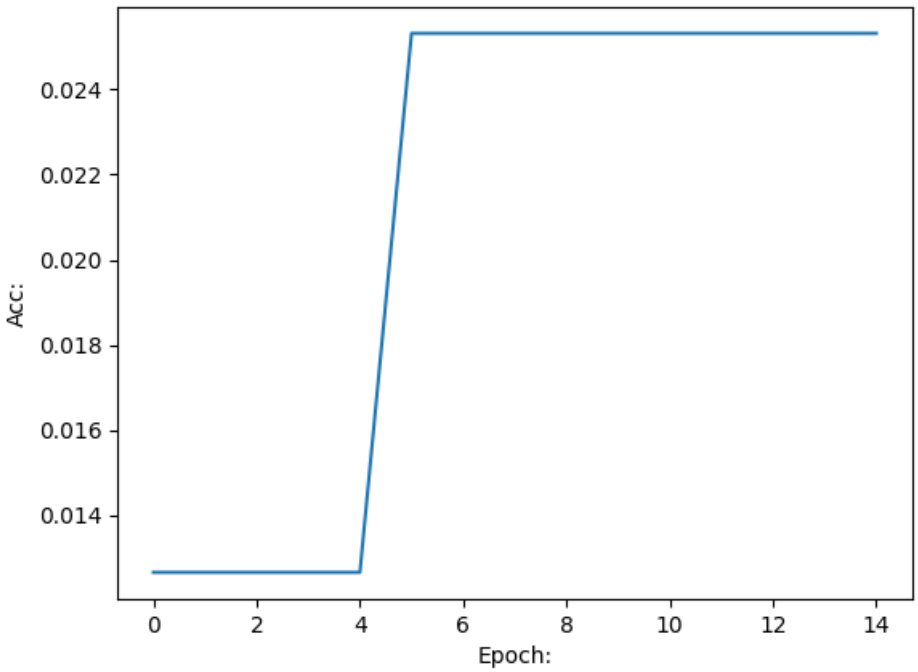
```
14
15 test_dataloader = Data.DataLoader(dataset=test_dataset, batch_size=batch_size,sh
16                                     num_workers=8)
17
18 class SimpleClassifier(nn.Module):
19     def __init__(self, *args, **kwargs) -> None:
20         super().__init__(*args, **kwargs)
21         self.fc = nn.Sequential(
22             nn.Linear(2, 256),
23             nn.ReLU(True),
24             nn.Linear(256, 10),
25             nn.ReLU(True),
26             nn.Softmax(dim=1)
27         )
28
29         self.loss_fn = nn.CrossEntropyLoss()
30
31     def forward(self, x):
32         return self.fc(x)
33
34     def train_step(self, X, y):
35         outputs = self.forward(X)
36         loss = self.loss_fn(outputs, y)
37         return loss
38
39     def predict(self, X, y):
40         outputs = self.forward(X)
41         i = torch.argmax(outputs)
42         if i == torch.argmax(y):
43             return 1
44         else:
45             return 0
46
47
48 model = SimpleClassifier().to('cuda')
49 Epochs = 15
50 LR = 5e-4
51 optimizer = torch.optim.Adam(model.parameters(), lr=LR)
52
53 train_losses = []
54 acc = []
55 for i in range(Epochs):
56     model.train()
57     with tqdm(total=len(train_dataloader)) as t:
58         for idx, (X, y) in enumerate(train_dataloader):
59             optimizer.zero_grad()
60             loss = model.train_step(X.to('cuda'), y.to('cuda'))
```

```
            loss.backward()
            optimizer.step()
            train_losses.append(loss.item())
            t.set_description("Epoch: %i" %i)
            t.set_postfix(train_loss='%.4f'%loss.item())
            t.update(1)

    model.eval()
    score = 0
    with tqdm(total=len(test_dataloader)) as t:
        for idx, (X, y) in enumerate(test_dataloader):
            score += model.predict(X.to('cuda'), y.to('cuda'))
            t.set_description("Test: %i" %i)
            t.set_postfix(test_acc='%.4f'%(score/float(idx+1)))
            t.update(1)

    acc.append(score/float(len(test_dataloader)))

plt.figure()
plt.plot(train_losses)
plt.ylabel("Loss:")
plt.xlabel("Steps:")
plt.show()
plt.savefig("./pca_classifier_loss.png")

plt.figure()
plt.plot(acc)
plt.ylabel("Acc:")
plt.xlabel("Epoch:")
plt.show()
plt.savefig("./pca_classifier_acc.png")
```

有损失图如下：

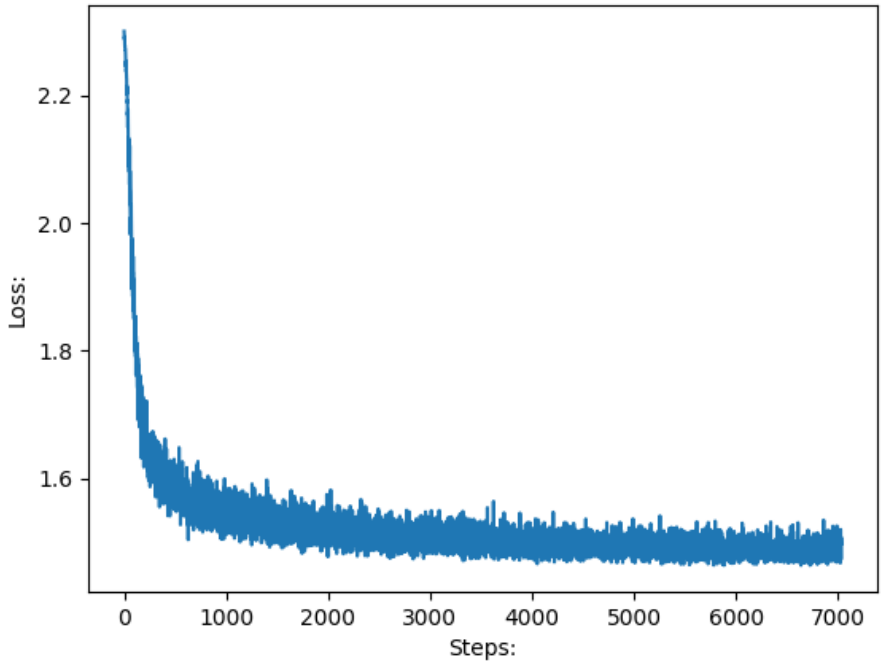


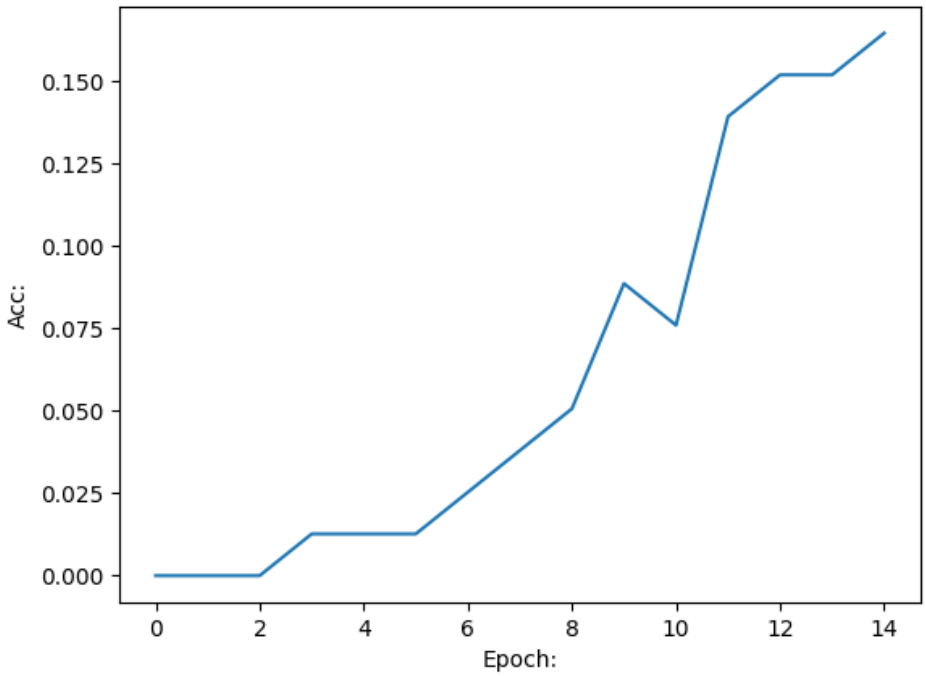

有测试集上预测精度：




从结果上容易得知，只依靠二维特征向量难以对输入进行合理的分类。

尝试扩大特征维度，以下为降维至30维后数据效果：




特征维度扩大后，分类效果变好很多，再进行适当的扩大维度和训练也许可以让数据降维后分类效果有进一步提升。

## ② AutoEncoder基本训练测试

### 1. 基本AutoEncoder模型结构搭建

```
class AE(nn.Module):
    def __init__(self, encoded_space_dim=32):
        super().__init__()
```

```
4
5           self.encoder = nn.Sequential(
6           nn.Conv2d(1, 8, 3, stride=2, padding=1),
7           nn.BatchNorm2d(8),
8           nn.ReLU(True),
9
10          nn.Conv2d(8, 16, 3, stride=2, padding=1),
11          nn.BatchNorm2d(16),
12          nn.ReLU(True),
13
14          nn.Conv2d(16, 32, 3, stride=2, padding=0),
15          nn.BatchNorm2d(32),
16          nn.ReLU(True),
17
18          nn.Flatten(start_dim=1),
19          nn.Linear(3 * 3 * 32, 128),
20          nn.ReLU(True),
21          nn.Linear(128, encoded_space_dim)
22          )
23
24
25          self.decoder = nn.Sequential(
26            nn.Linear(encoded_space_dim, 128),
27            nn.ReLU(True),
28            nn.Linear(128, 3 * 3 * 32),
29            nn.ReLU(True),
30            nn.Unflatten(dim=1,unflattened_size=(32, 3, 3)),
31            nn.ConvTranspose2d(32, 16, 3,
32                               stride=2, output_padding=0),
33            nn.BatchNorm2d(16),
34            nn.ReLU(True),
35            nn.ConvTranspose2d(16, 8, 3, stride=2,
36                               padding=1, output_padding=1),
37            nn.BatchNorm2d(8),
38            nn.ReLU(True),
39            nn.ConvTranspose2d(8, 1, 3, stride=2,
40                               padding=1, output_padding=1)
41          )
42
43      def forward(self, x):
44          """
45          :param [b, 1, 28, 28]:
46          :return [b, 1, 28, 28]:
47          """
48          batchsz = x.size(0)
49          # encode
50          x = self.encoder(x)
```

```
        # decode
        x = self.decoder(x)
        # reshape
        x = x.view(batchsz, 1, 28, 28)
        return x
```

模型整体结构采用了编码器解码器对称的经典设计方法，在构建模型的过程中结合使用了卷积层和全连接层，对于解码器而言先使用卷积层逐渐对图像进行特征压缩与升维的处理，之后使用全连接层得到中间的隐藏层特征作为对解码器的输入，模型的编码器输出为一32维的特征张量。解码器总体而言则是编码器的逆过程，逐渐从特征张量还原成原图。

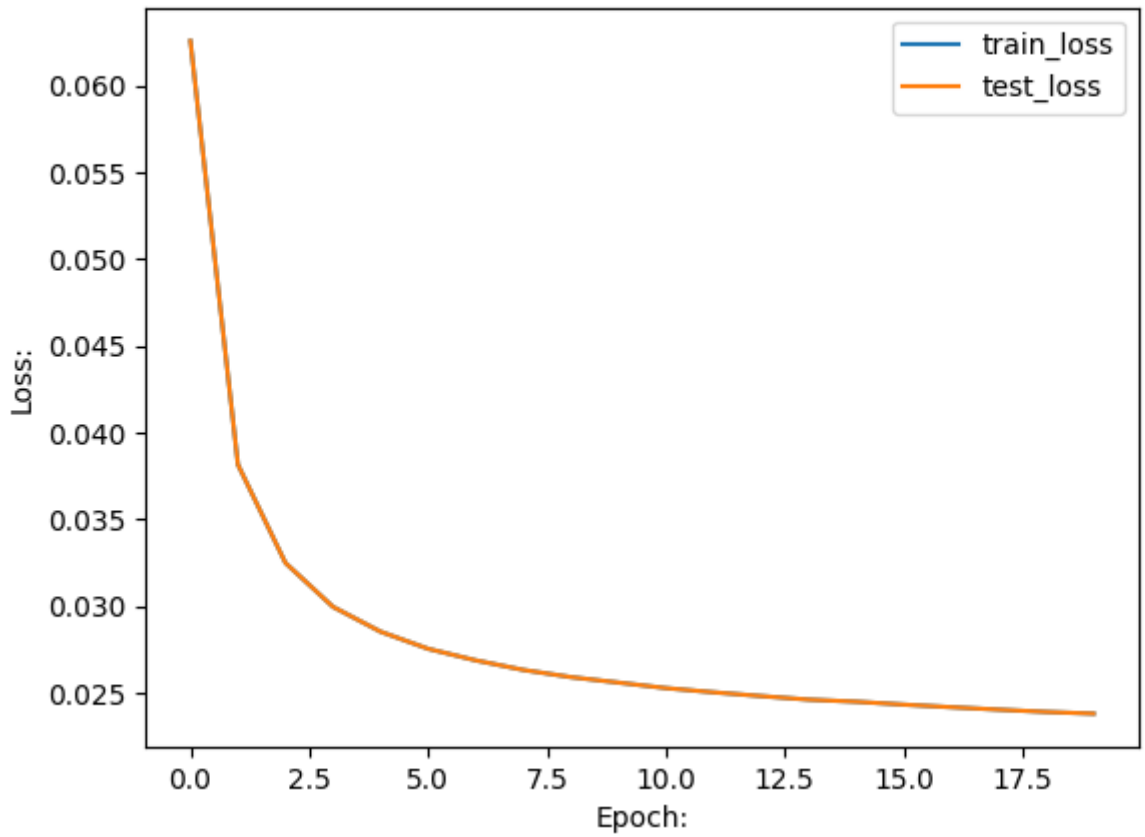
2. 普适训练接口定义

```
def add_noise(inputs,noise_factor=0.3):
  noisy = inputs+torch.randn_like(inputs) * noise_factor
  noisy = torch.clip(noisy,0.,1.)
  return noisy


def Train(img:torch.Tensor):
    optimizer.zero_grad()
    y = torch.clone(img)

    if train_mode == 'Denoise':
        img = add_noise(img).to(img.device)

    if isinstance(model, AE):
        re_img = model(img)
        loss = loss_func(re_img, y)
    elif isinstance(model, VAE):
        re_img, kld = model(img)
        loss = loss_func(re_img, y) + kld * 1.0

    # loss = loss_func(re_img, y)
    loss.backward()
    optimizer.step()
    return loss_func(re_img, y)
```

对于普通的训练过程，其训练较为简单，只需比较输出的内容和原图，求MSELoss或者MAELoss等即可。

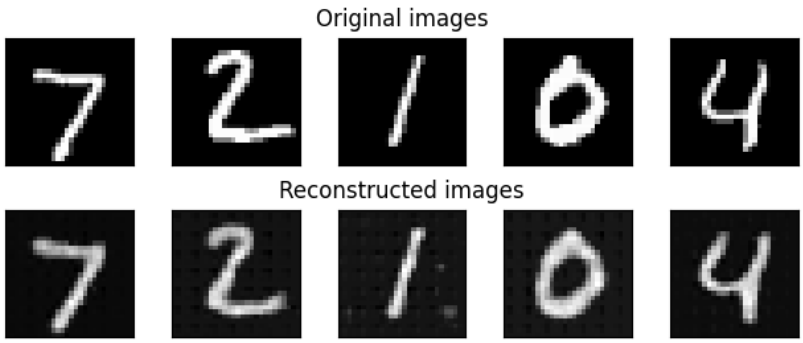
但为之后附加题铺垫，也进行了VAE，DAE的训练支持，针对DAE增加了对图像加噪这一步，针对VAE增加了KL散度损失的计算，但在绘制损失时，所使用的损失都是与原图进行比较的损失，确保了损失图意义一致。

### 3. AE训练损失结果及可视化效果




该模型训练过程进行了20个epoch，在学习率为5e-4，batch-size为32大小情况下使用Adam优化器进行训练，从损失上来看收敛效果良好。

同样有可视化效果图：



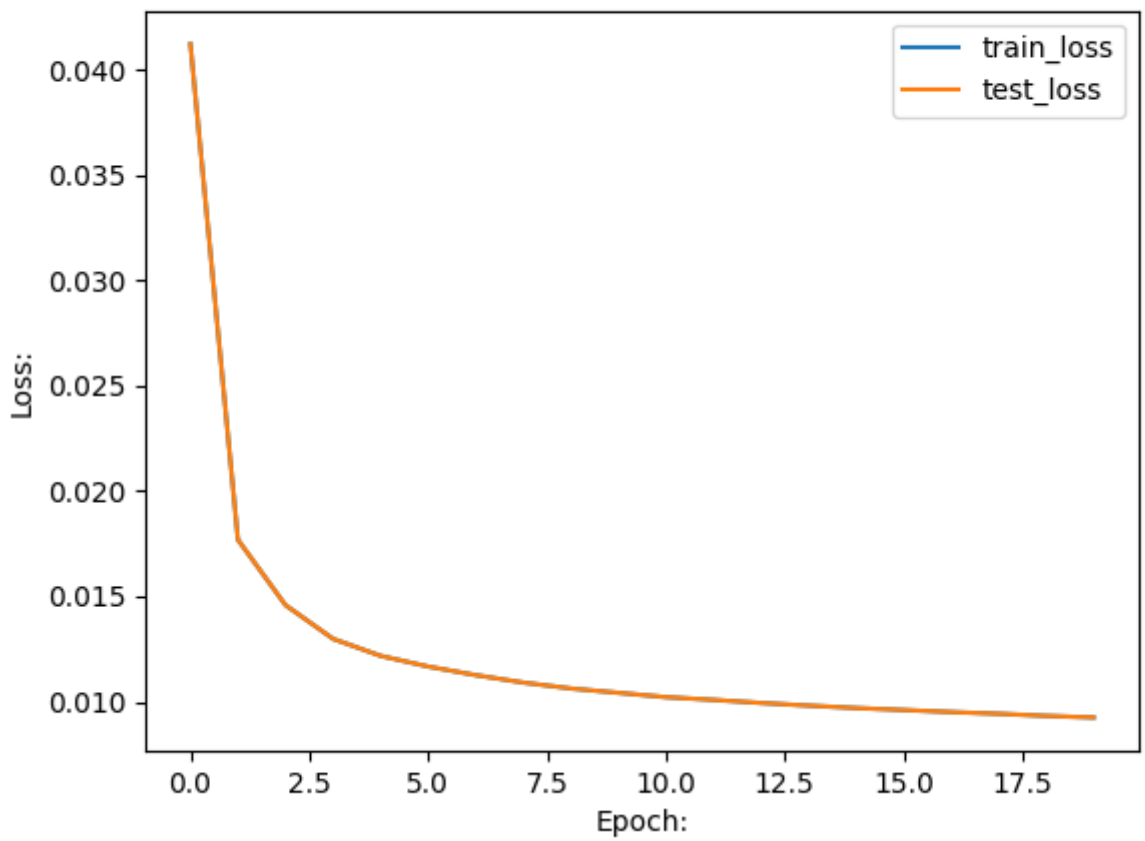

对图像的压缩，恢复情况较为良好，精度损失并不多。

## ③ 引入DenoiseAutoEncoder

1. 去噪自动编码器的修改与结果分析

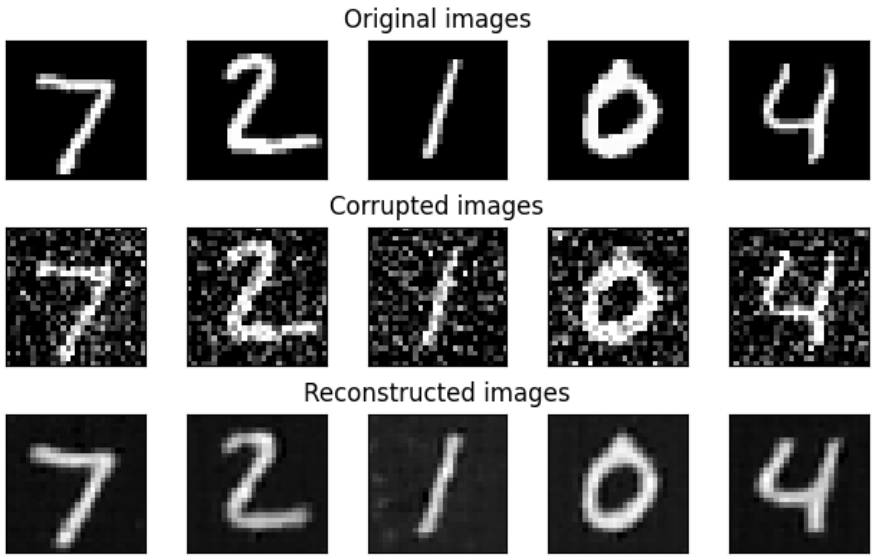
根据调查了解，我们可以利用自动编码器对特征的无监督学习能力，进行图像去噪的尝试，为强化去噪能力，我们在训练过程中对原图像进行了噪声添加的步骤，模型结构并不需要进行修改。

有损失图如下：



收敛效果较为良好，但因为在原图像有噪声添加，破坏了部分图像特征，其与原图像比较必然损失精度更大。

可有去噪自编码器效果图：




观察可知模型的去噪能力较佳，能够进行对加噪的图像进行精度恢复。

## ④ 引入VariationalAutoEncoder

同时通过调研，得知一类重要的生成模型由自编码器演化而来，其通过KL散度对编码器得出的特征向量进行约束。

### 1. VAE模型构建

```
class VAE(nn.Module):
    def __init__(self, encoded_space_dim=20):
        super(VAE, self).__init__()
        self.encoder = nn.Sequential(
        nn.Conv2d(1, 8, 3, stride=2, padding=1),
        nn.BatchNorm2d(8),
        nn.ReLU(True),

        nn.Conv2d(8, 16, 3, stride=2, padding=1),
        nn.BatchNorm2d(16),
        nn.ReLU(True),

        nn.Conv2d(16, 32, 3, stride=2, padding=0),
        nn.BatchNorm2d(32),
        nn.ReLU(True),

```

```
        nn.Flatten(start_dim=1),### Linear section
        nn.Linear(3 * 3 * 32, 128),
        nn.ReLU(True),
        nn.Linear(128, encoded_space_dim)
        )


        self.decoder = nn.Sequential(
          nn.Linear(encoded_space_dim // 2, 128),
          nn.ReLU(True),
          nn.Linear(128, 3 * 3 * 32),
          nn.ReLU(True),
          nn.Unflatten(dim=1,unflattened_size=(32, 3, 3)),
          nn.ConvTranspose2d(32, 16, 3,
                             stride=2, output_padding=0),
          nn.BatchNorm2d(16),
          nn.ReLU(True),
          nn.ConvTranspose2d(16, 8, 3, stride=2,
                             padding=1, output_padding=1),
          nn.BatchNorm2d(8),
          nn.ReLU(True),
          nn.ConvTranspose2d(8, 1, 3, stride=2,
                             padding=1, output_padding=1)
        )

    def forward(self, x):
        """
        :param [b, 1, 28, 28]:
        :return [b, 1, 28, 28]:
        """
        batchsz = x.size(0)
        q = self.encoder(x)

        mu, sigma = q.chunk(2, dim=1)
        q = mu + sigma * torch.randn_like(sigma)

        x_hat = self.decoder(q)
        x_hat = x_hat.view(batchsz, 1, 28, 28)

        # KL
        kld = 0.5 * torch.sum(
            torch.pow(mu, 2) +
            torch.pow(sigma, 2) -
            torch.log(1e-8 + torch.pow(sigma, 2)) - 1
        ) / (batchsz*28*28)

        return x_hat, kld
```

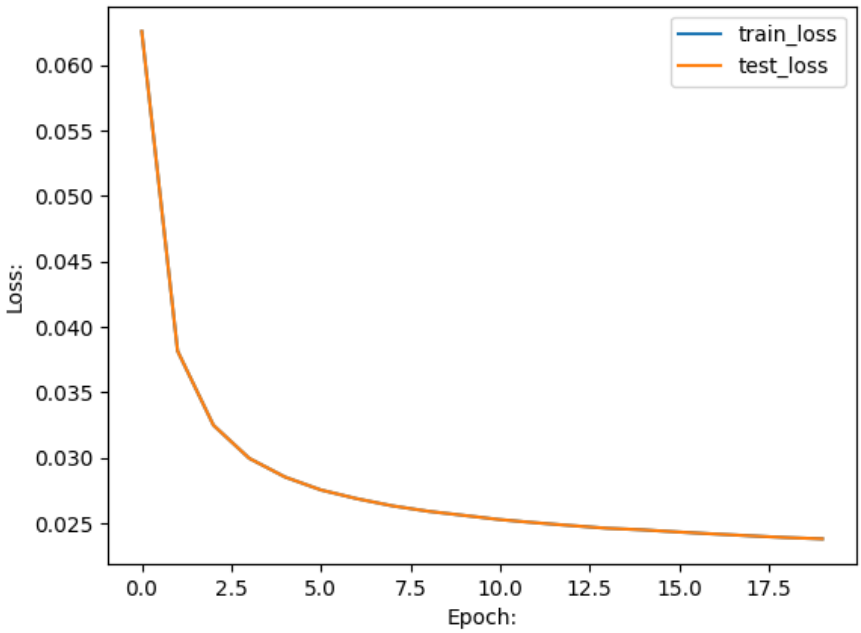
```
64
```

## 2. VAE训练效果（非去噪与去噪）

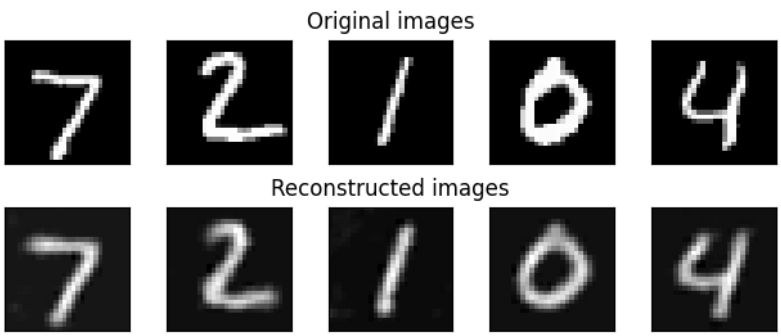
对于VAE进行了两种训练方式的尝试，先是正常的进行自编码器训练后尝试了将去噪方法应用在VAE上得到DVAE的方法

**Normal VAE：**

有损失图：

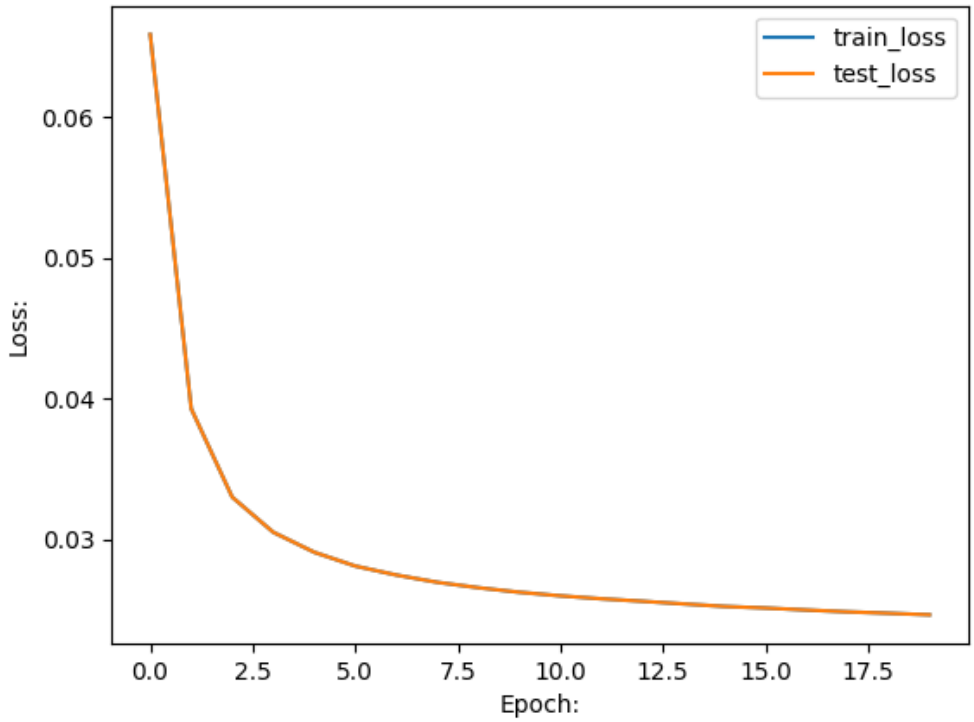



效果图：

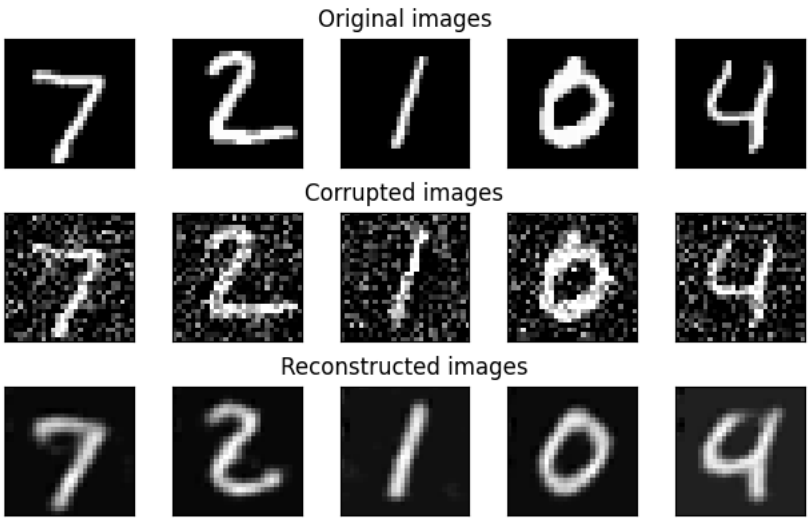


**Denoise VAE：**

有损失图：




效果图：




观察可知去噪变分自编码器其生成效果风格与原本的数字差异性更大，风格更多变。

# 三、结语：

🎉 在本次实验中，先进行了PCA的数据降维尝试，对特征张量进行了可视化，并绘制了数据在R2分布上的嵌入分布图，尝试了仅利用二维特征张量进行分类，其难以起到分类效果。

之后搭建了自编码器，对图像进行压缩后恢复，使用了卷积搭配全连接网络的模型结构，并进行对称设计，效果较好。

为加强模型去噪能力，引入了去噪自编码器，成功能在受一定噪声污染的情况下恢复原图像。

为加强模型生成能力，生成风格不同的数据，尝试了变分自编码器，其中去噪变分自编码器效果最佳。