

# Project report

---

刘双铖

521021910904

[lsc2021@sjtu.edu.cn](mailto:lsc2021@sjtu.edu.cn)

## Mandatory Task

---

### MyNet

#### python 代码实现

```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, padding=1)
        self.bn1 = nn.BatchNorm2d(16)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, padding=1)
        self.bn2 = nn.BatchNorm2d(32)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.bn3 = nn.BatchNorm2d(64)
        self.fc1 = nn.Linear(64 * 8 * 8, 512)
        self.bn4 = nn.BatchNorm1d(512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, x):
        x = F.relu(self.bn1(self.conv1(x)))
        x = self.pool1(x)
        x = F.relu(self.bn2(self.conv2(x)))
        x = self.pool2(x)
        x = F.relu(self.bn3(self.conv3(x)))
        x = x.view(-1, 64 * 8 * 8)
        x = F.relu(self.bn4(self.fc1(x)))
        x = self.fc2(x)
        return x, x.view(x.size(0), -1)

    def get_mid(self, x, str):
        x = self.conv1(x)
        if str == "conv1":
            return x
        x = F.relu(self.bn1(x))
        if str == "bn1":
            return x
        x = self.pool1(x)

        x = self.conv2(x)
        if str == "conv2":
```

```

        return x
    x = F.relu(self.bn2(x))
    if str == "bn2":
        return x
    x = self.pool2(x)

    x = self.conv3(x)
    if str == "conv3":
        return x
    x = F.relu(self.bn3(x))
    if str == "bn3":
        return x
    x = x.view(-1, 64 * 8 * 8)

    x = self.fc1(x)
    if str == "fc1":
        return x
    x = F.relu(self.bn4(x))
    if str == "bn4":
        return x
    x = self.fc2(x)
    if str == "final":
        return x

```

其中我自己定义的get\_mid()函数是为了更方便提取中层输出，只需要指定str就可以任意提取中层输出特征

## 网络结构

```

input: 1000x1x32x32
|
conv1: 3x3, s=1, p=1, output: 1000x16x32x32
|
bn1: num_features=16, output: 1000x16x32x32
|
relu
|
pool1: 2x2, s=2, output: 1000x16x16x16
|
conv2: 3x3, s=1, p=1, output: 1000x32x16x16
|
bn2: num_features=32, output: 1000x32x16x16
|
relu
|
pool2: 2x2, s=2, output: 1000x32x8x8
|
conv3: 3x3, s=1, p=1, output: 1000x64x8x8
|
bn3: num_features=64, output: 1000x64x8x8
|
relu
|
view: output: 1000x4096
|


```

```
fc1: (4096, 512), output: 1000x512
|
bn4: num_features=512, output: 1000x512
|
relu
|
fc2: (512, 10), output: 1000x10
|
output: 1000x10
```

在设计这个网络的过程中我的思路是比较简单的，最初想法就是在Lenet-5的基础上增加个卷积层，然后把卷积核变小，看看能不能提取特征效果更好，但是实验下来效果一般，准确率甚至会有所下降。于是我又在网络里添加BN层来提高模型泛化能力并且缓解梯度消失。这样的好处是显而易见的，在我的训练中添加BN层以后我的训练效果会迅速地收敛，并且准确率也有所提高。

最终我的神经网络包含3个卷积层，2个池化层和2个全连接层，中间使用Relu激活函数和BN层来提高泛化能力并加速收敛（不知道3-7层的要求中是否包含BN层，但是当时实验中添加BN层效果显著因此也没有再修改网络结构），最终准确率大概提高了6%，收敛速率加快了10倍左右。

## Loss 曲线

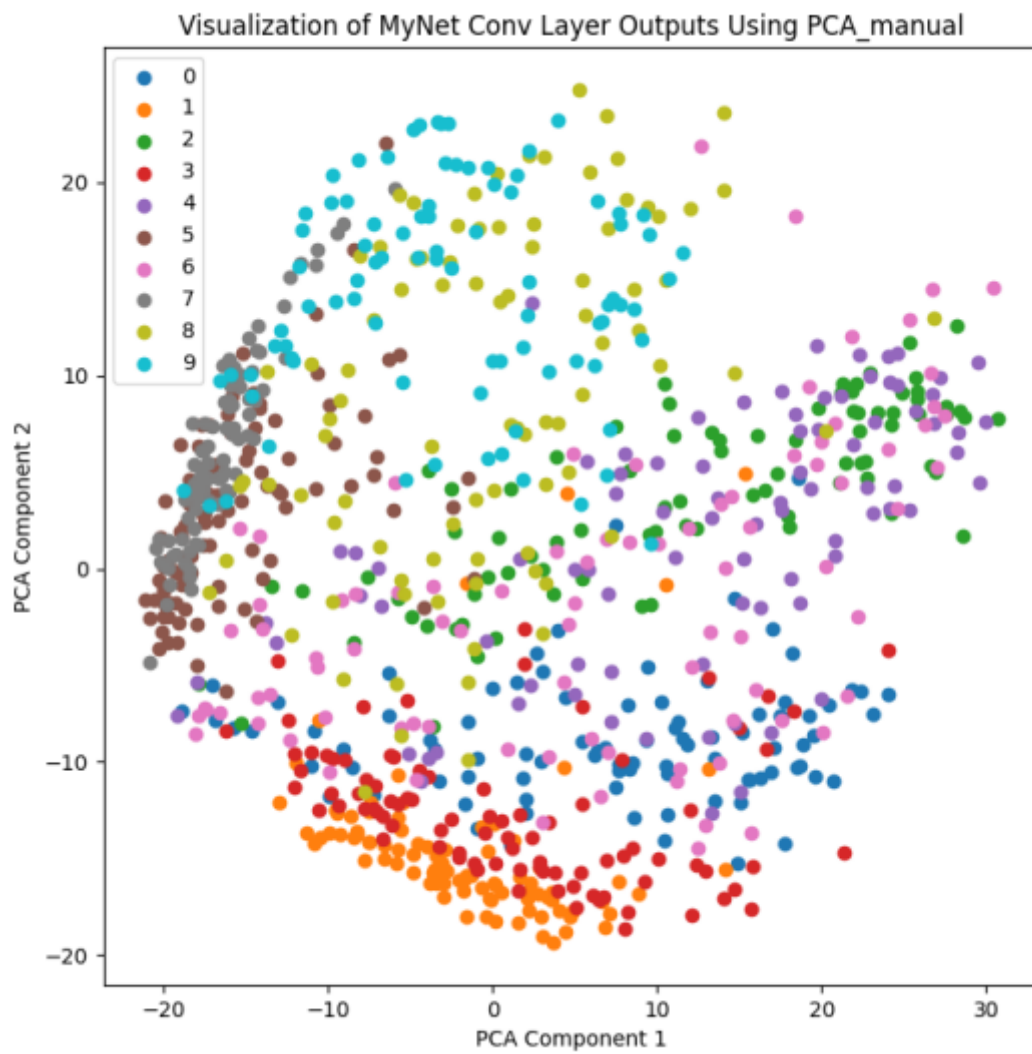
 mynet loss 100

## Accuracy 曲线

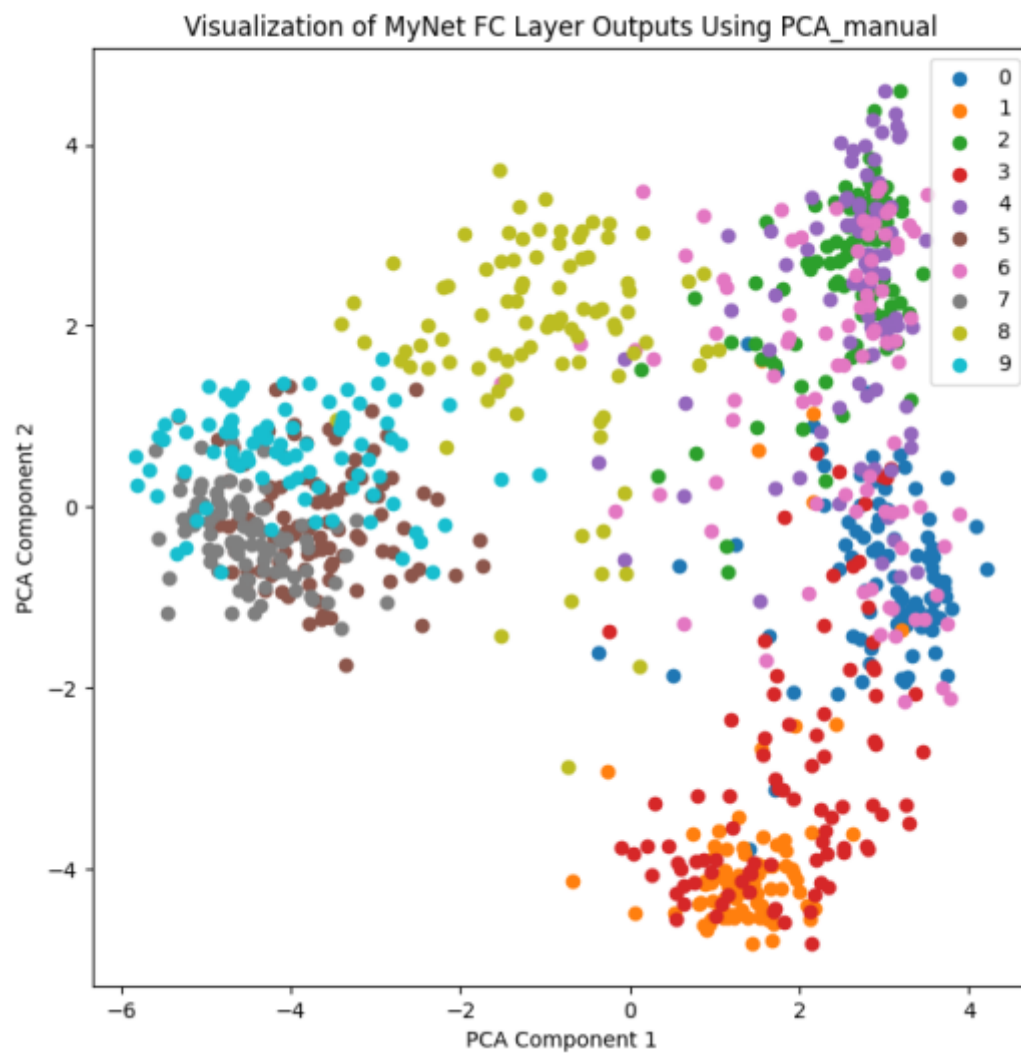
 My net acc 100

## PCA 可视化

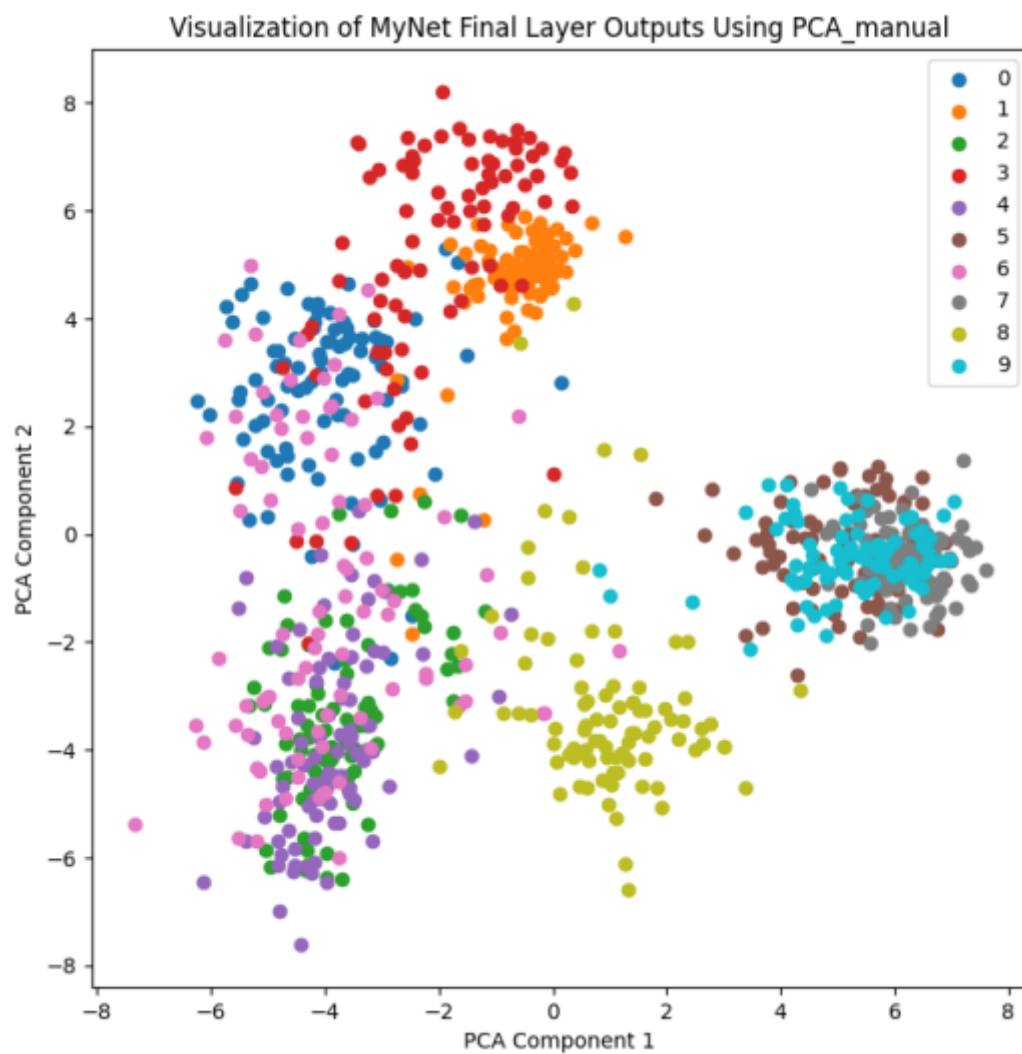
- 第二个卷积层：



- 第一个全连接层:

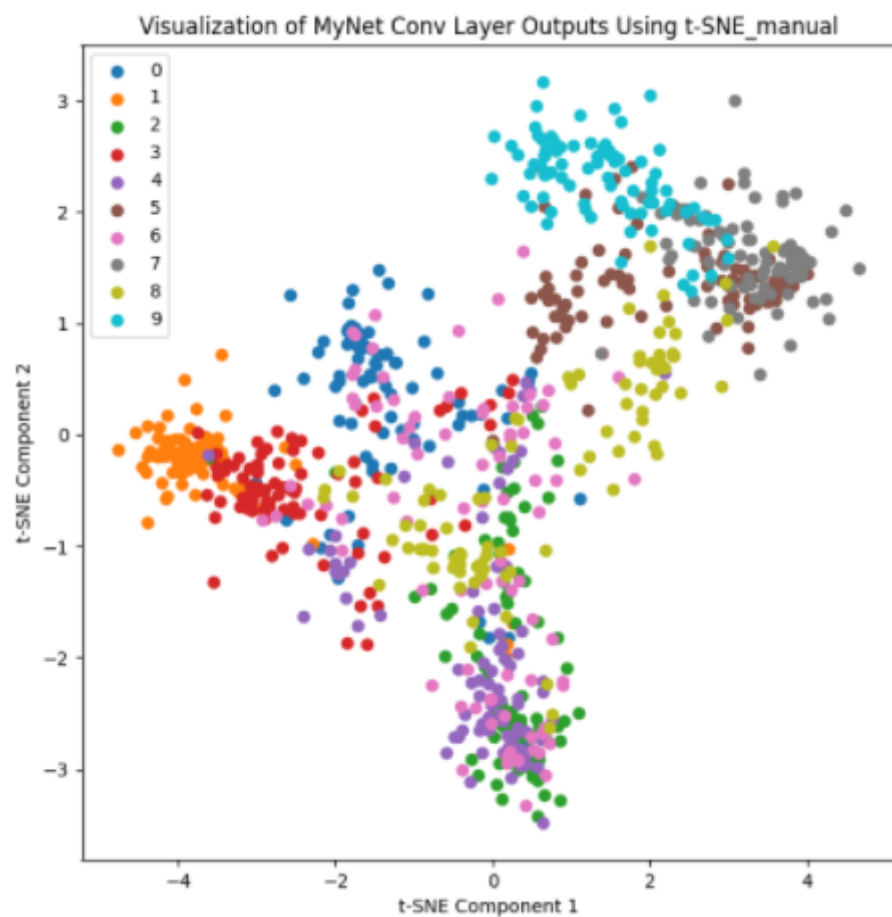


- 最终层:

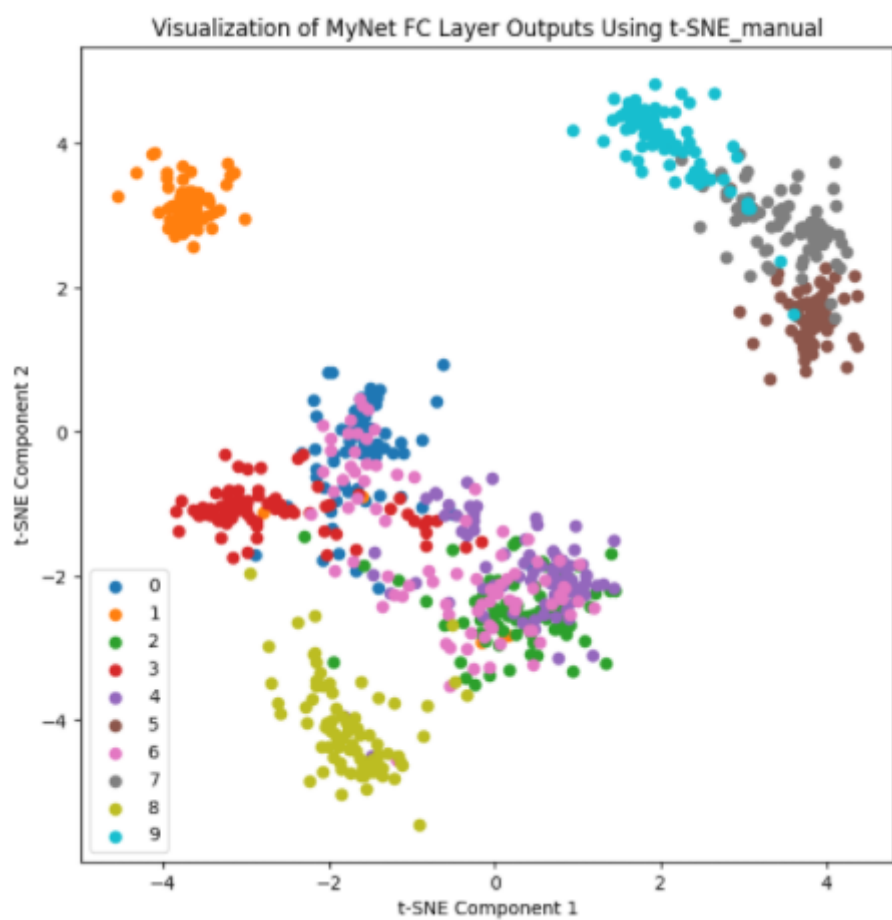


## t-SNE 可视化

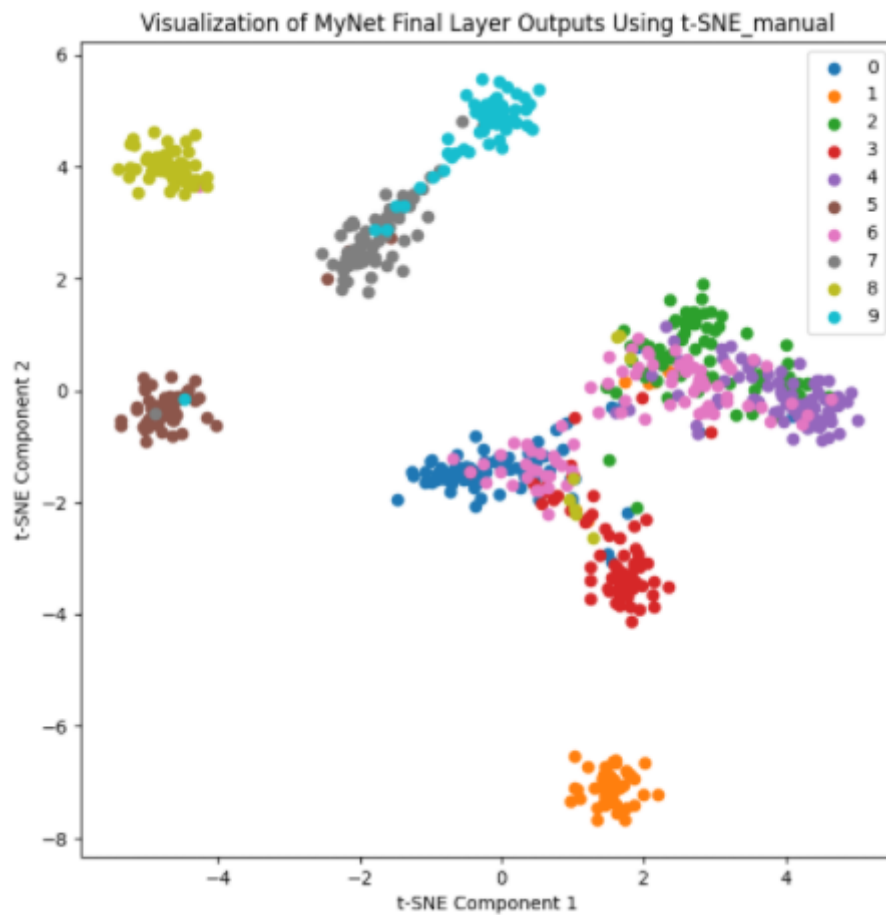
- 第二个卷积层:



- 第一个全连接层:




- 最终层:



## LeNet

### Loss 曲线


 Lenet loss 400

### Accuracy 曲线


 Lenet acc 400

### PCA 可视化


- 第二个卷积层:

 fb5edbb2957b5941ee6dedfd3e547b0

- 第一个全连接层:

 f8807090961b2b59104a5d3cdc5aba5

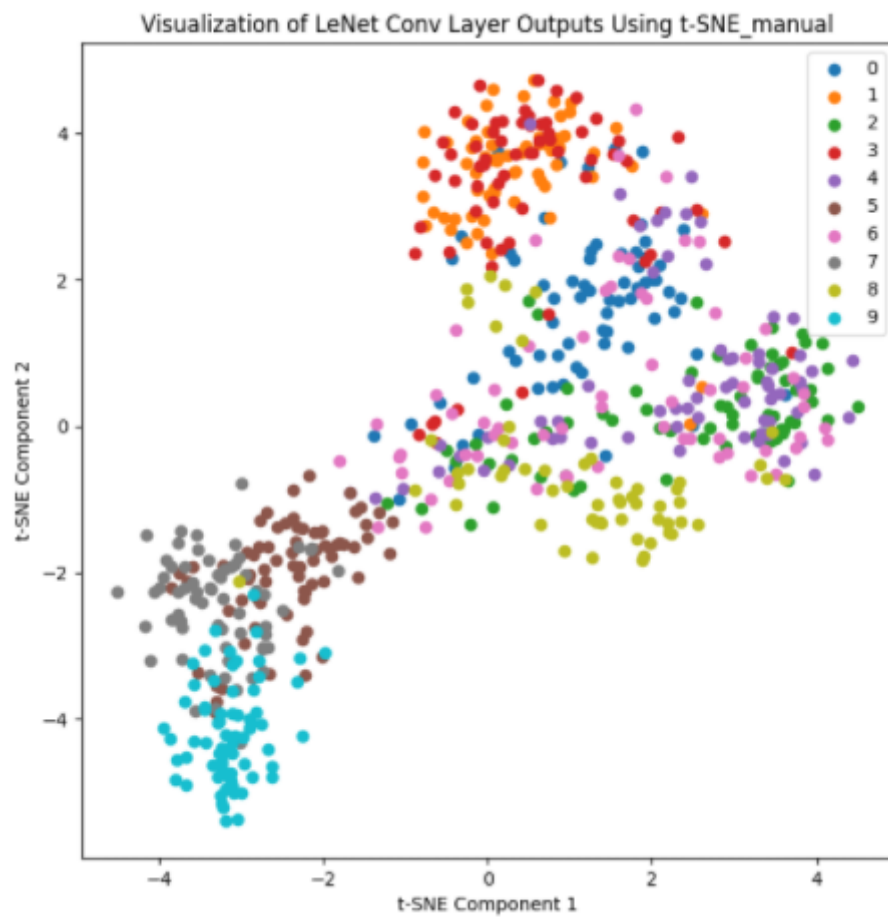
- 最终层:

 fd0ce8cb43d64f34bb93e2c59e77627

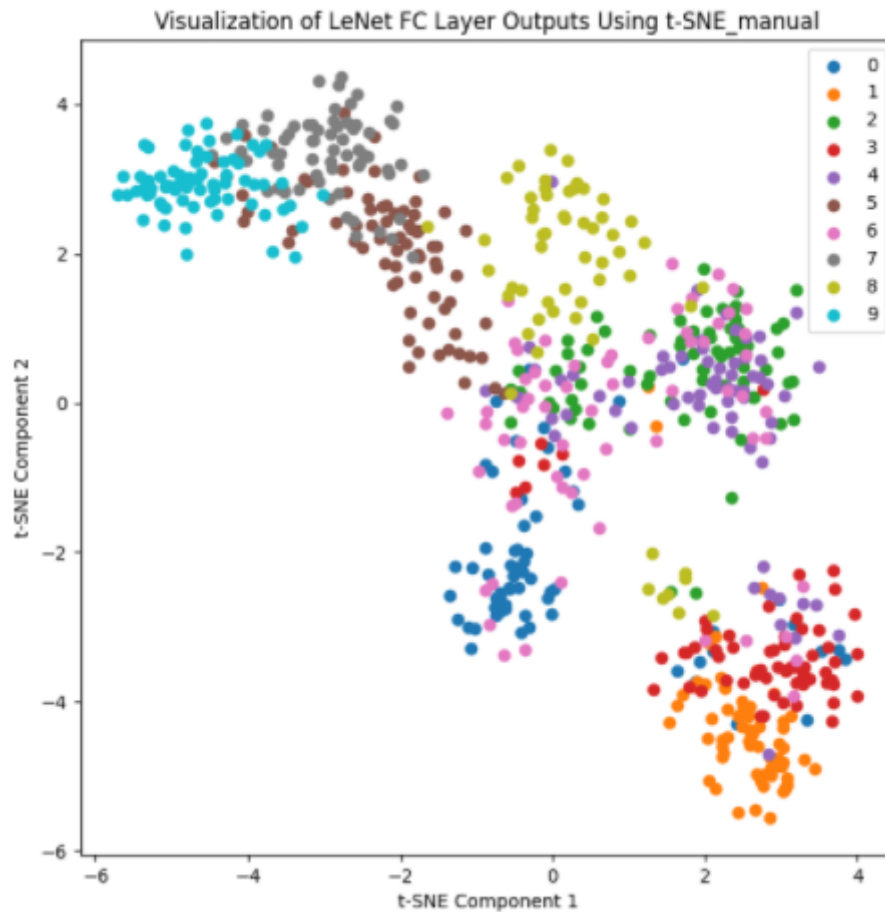
### t-SNE 可视化

- 第二个卷积层:

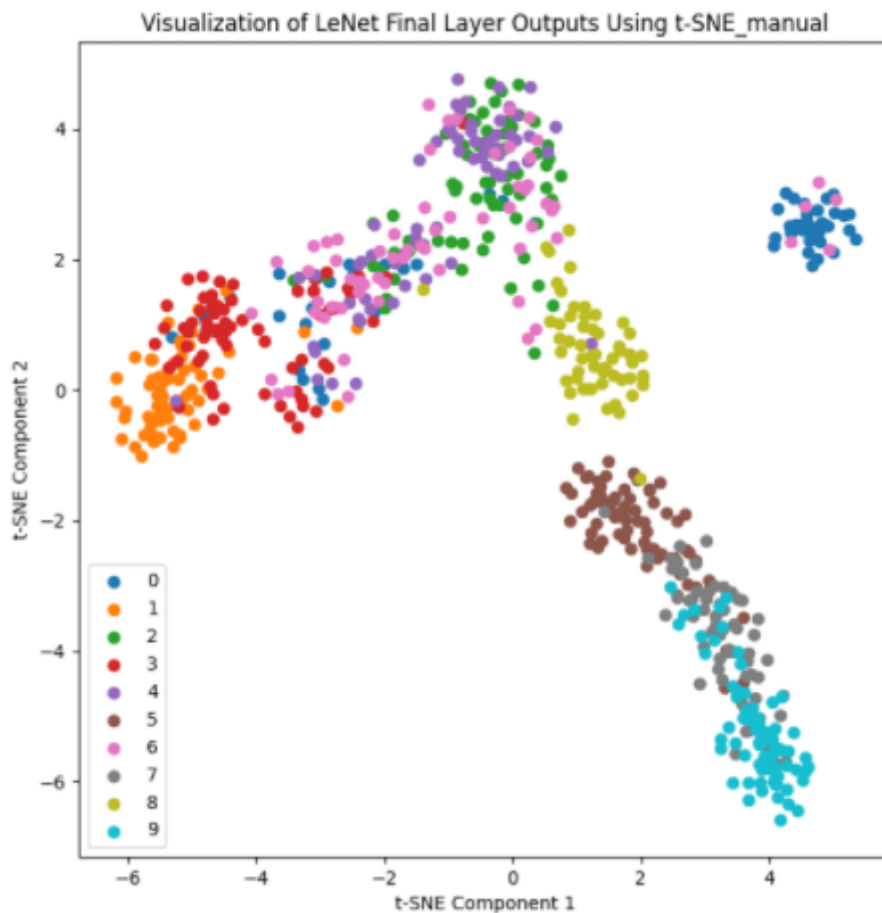




- 第一个全连接层:



- 最终层:



## 结论

- PCA通过计算数据的协方差矩阵，找到数据中最重要的主成分，然后用主成分来表示数据，因此只有在处理线性关系时效果才比较好，这点可以从可视化的效果图中看出来。
- 而 t-SNE 通过优化数据点间的相似度，将高维数据映射到低维空间中，可以把高维数据中的复杂结构和聚类关系保留下来，从而提供更好的聚类效果，这点也可以从可视化的效果图中看出来。
- 此外在可视化过程中无论是LeNet还是MyNet，对于粉色类别的聚类效果都很差，已经尝试过修改 t-SNE中迭代次数和perplexity等超参来试图优化可视化效果但是收效甚微，猜测可能是数据本身的特征过于复杂或者噪声过多，可能导致相似度计算不准确，从而影响t-SNE的降维效果。

## Optional Task1

---

### VAE 结构

```

input: image (3,32x32)
|
encoder: conv2d (32,16x16) -> relu (32,16x16) -> conv2d (64,8x8) -> relu
(64,8x8) -> flatten (4096,1) -> linear (128,1) -> (mu, logvar)
|
reparameterize(64,1)
|
decoder: linear (128,1) -> relu (128,1) -> linear (4096,1) -> relu (4096,1) ->
unflatten (64,8x8) -> conv_transpose2d (32,16x16) -> relu (32,16x16) ->
conv_transpose2d (3,32x32)-> sigmoid (将输出的像素值缩放到0到1之间)
|
output: reconstructed image (3,32x32)

```

## python 代码实现

```

class VAE(nn.Module):
    def __init__(self, z_dim):
        super(VAE, self).__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.Flatten(),
            nn.Linear(64 * 8 * 8, 128),
            nn.ReLU(),
        )

        self.mu = nn.Linear(128, z_dim)
        self.logvar = nn.Linear(128, z_dim)

        self.decoder = nn.Sequential(
            nn.Linear(z_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 64 * 8 * 8),
            nn.ReLU(),
            nn.Unflatten(1, (64, 8, 8)),
            nn.ConvTranspose2d(64, 32, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 3, kernel_size=4, stride=2, padding=1),
            nn.Sigmoid(),
        )

    def encode(self, x):
        h = self.encoder(x)
        return self.mu(h), self.logvar(h)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std

    def decode(self, z):

```

```

        return self.decoder(z)

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

    def loss_function(recon_x, x, mu, logvar):
        BCE = nn.functional.binary_cross_entropy(recon_x, x, reduction='sum')
        KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
        return BCE + KLD

```

我设计的 VAE 中 encoder 包括 2 个卷积层和 3 个全连接层，用于将输入图像映射到  $z\_dim$  维度的空间中。encoder 的输出包括均值和方差，用于后面的 reparameterize。reparameterize 从标准正态分布中采样，生成潜在变量。decoder 包括三个全连接层和两个反卷积层，用于将潜在变量映射回数据空间中，从而生成重构图像。VAE 的 loss\_function 包括重构误差(这里采用 Binary Cross Entropy)和 KL 散度。

训练过程中我使用 Adam 优化器来最小化损失函数，在训练的每个 epoch 循环里将 data 传递给 VAE 模型进行前向计算，得到重构图像、潜在变量的均值和方差。接下来，通过 loss\_function 计算总损失，然后通过反向传播来计算并更新模型参数，同时累计每个批次的 loss。

我通过 `torch.randn(z_dim)` 从标准正态分布中随机生成潜在变量维度的张量，重构图片，并且线性拟合两个张量来重构图片，下面为效果图

## 实验效果可视化





# 参考资料

<https://zhuanlan.zhihu.com/p/32183010>

<https://zhuanlan.zhihu.com/p/148170862>

<https://zhuanlan.zhihu.com/p/108262170>

<https://zhuanlan.zhihu.com/p/64863315>