CS3612-01 机器学习　Final Project

Classification, Image Reconstruction
and Model Analysis

周　易　520030910269

目 录

第一章　Mandatory Task: Fashion-MNIST clothing classification

本次实验使用了简化的 fashion-MNIST 数据集，使用了 LeNet 神经网络和一个由自己设计的浅层网络 MyNet 对任务进行预测。

1.1　LeNet 网络

经典 LeNet 是一种最初用于手写数字识别任务的卷积神经网络（Convolutional Neural Network，CNN），由 Yann LeCun 等人在 1998 年提出。它是第一个成功应用于数字识别任务的卷积神经网络，也是深度学习领域的里程碑之一。

1.1.1　LeNet 网络结构

经典 LeNet 的网络架构由 7 层组成，包括输入层、卷积层、池化层和全连接层。

LeNet 中有两个卷积层，分别是网络中的第一层和第三层在两个卷积层中，每个滤波器与输入图像进行卷积操作，分别生成 6 和 16 个特征图（feature map）。这些特征图代表了输入图像的不同局部特征。

两个卷积层后面都是是池化层。池化层的目的是减小特征图的空间尺寸，并保留主要的特征。之后是三个全连接层。

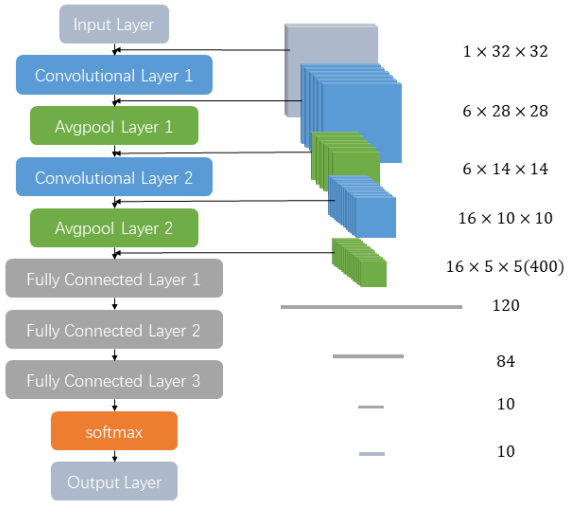


图 1–1　使用 PowerPoint 绘制的 LeNet 网络结构

LeNet 网络 pytorch 实现如下：

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.avgpool1 = nn.AvgPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.avgpool2 = nn.AvgPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = self.avgpool1(x)
        x = torch.relu(self.conv2(x))
        x = self.avgpool2(x)

        x = x.view(x.size(0), -1)

        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)

        return F.softmax(x, dim=0)
```

1.1.2 LeNet 网络训练过程及其准确率

简化的 fashion-MNIST 数据集共具有 1000 个数据样本，在本次实验中，分为单个大小为 32 的 batch 进行训练，每轮训练中，batch 都会被重新打乱，从而让模型取得更好的泛化效果。同样为了减少过拟合，梯度下降在每个 batch 上进行，缺点是模型的收敛性可能会受到影响。所使用的损失函数为 CrossEntropyLoss()，优化算法为 Adam。

模型的性能主要由预测的准确率进行衡量。训练过程中，在训练集上，最好的准确率为 0.943；在测试集上，最好的准确率为 0.817。

训练了 500 个 epoch 后，最终保存的模型在训练集和测试集上的准确率分别为 0.932 和 0.812，略低于模型训练过程中的历史最高值。

模型训练过程中的 curve 如下：

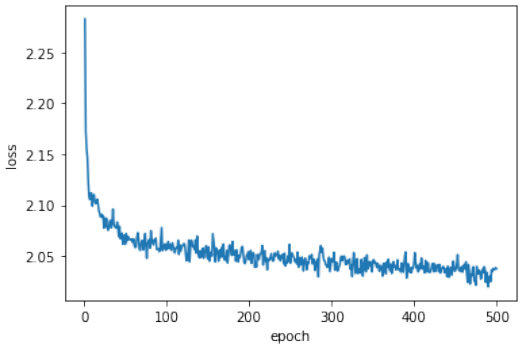


图 1–2 训练集上的 loss curve

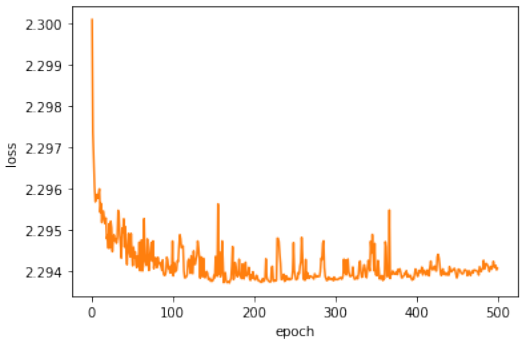


图 1–3 测试集上的 loss curve

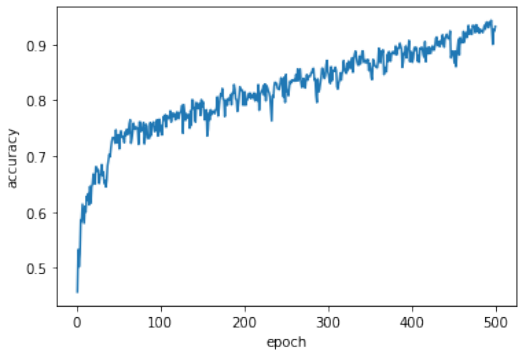


图 1–4 训练集上的 accuracy curve

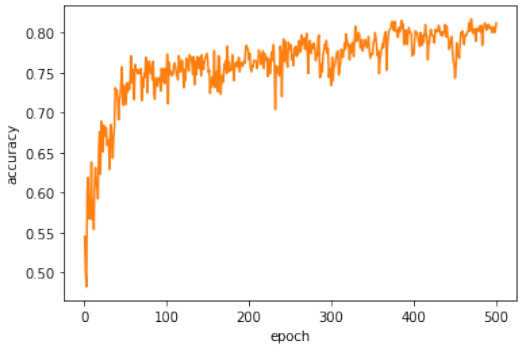


图 1–5 测试集上的 accuracy curve

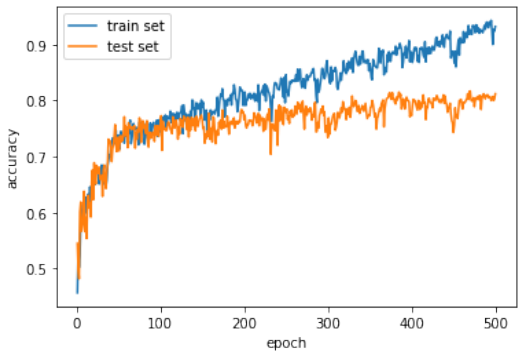


图 1–6 accuracy 对比

1.2　个人设计的浅层网络 MyNet

LeNet 大致由两层卷积层、两层平均池化层和三层全连接层构成。其中，卷积层用于提取图像的局部特征，池化层用于减小特征图的尺寸并保留主要特征，全连接层用于将提取的特征映射到不同的类别上。

在个人设计的网络 MyNet 中，以 LeNet 为基础，将两个平均池化层替换成了最大池化层对模型进行改良，因为最大池化层有更好的保存图像显著特征和平移不变性等特征，是一种更流行的池化层选择。

1.2.1　MyNet 网络架构

MyNet 网络结构示意图：

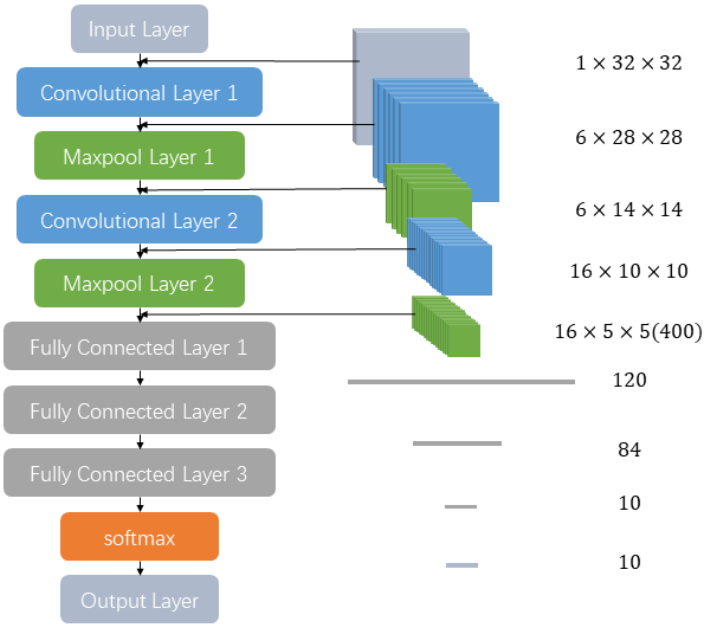


图 1–7　使用 PowerPoint 绘制的 MyNet 网络结构

MyNet 网络的 pytorch 实现如下：

```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=5)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=5)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2)

        self.fc1 = nn.Linear(16 * 5 * 5, 120)
```

```
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = self.pool1(x)
        x = torch.relu(self.conv2(x))
        x = self.pool2(x)

        x = x.view(x.size(0), -1)

        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        x = self.fc3(x)

        return F.softmax(x, dim=0)
```

1.2.2 MyNet 网络训练过程及其准确率

为了横向对比 MyNet 网络相对于 LeNet 网络的效果改进，采用了完全一致的训练过程，此处不赘述。

MyNet 在训练集上历史最高的准确率为 0.974；在测试集上历史最高的准确率为 0.823。

训练了 450 个 epoch 后，最终保存的模型在训练集和测试集上的准确率分别为 0.952 和 0.815，略低于模型训练过程中的历史最高值。在该任务中上，MyNet 的准确率指标在训练集和测试集上都略高于 LeNet，但由于 MyNet 中选用最大池化层，其训练曲线也显得更为不平稳，随机性更大。

MyNet 训练过程中的 curve 如下：

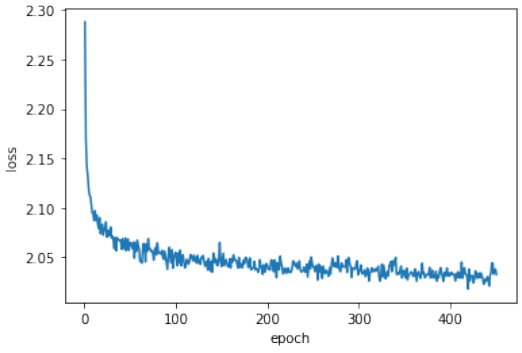


图 1–8 训练集上的 loss curve

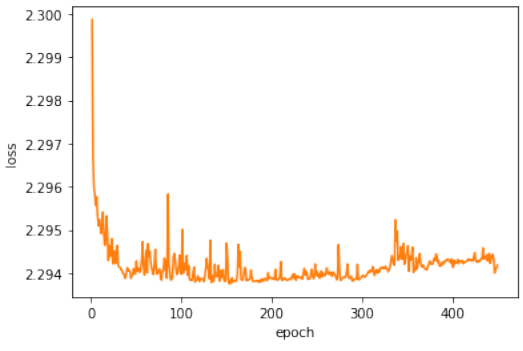


图 1–9 测试集上的 loss curve

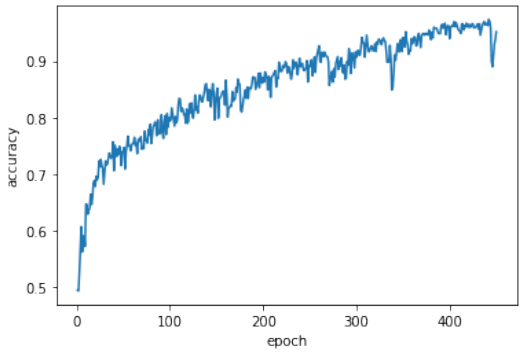


图 1–10 训练集上的 accuracy curve

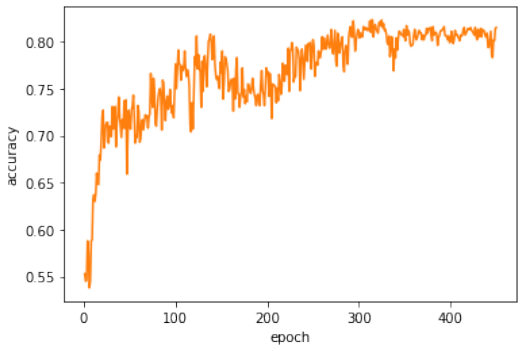


图 1–11 测试集上的 accuracy curve

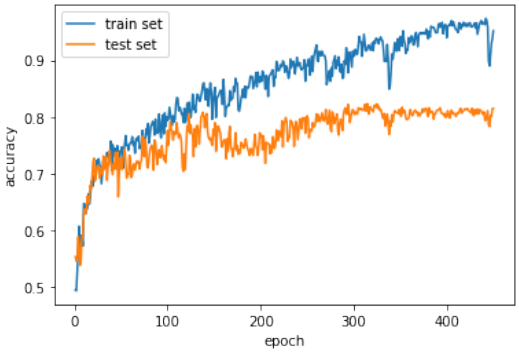


图 1–12 accuracy 对比

1.3 Principal component analysis (PCA)

1.3.1 PCA 原理解释

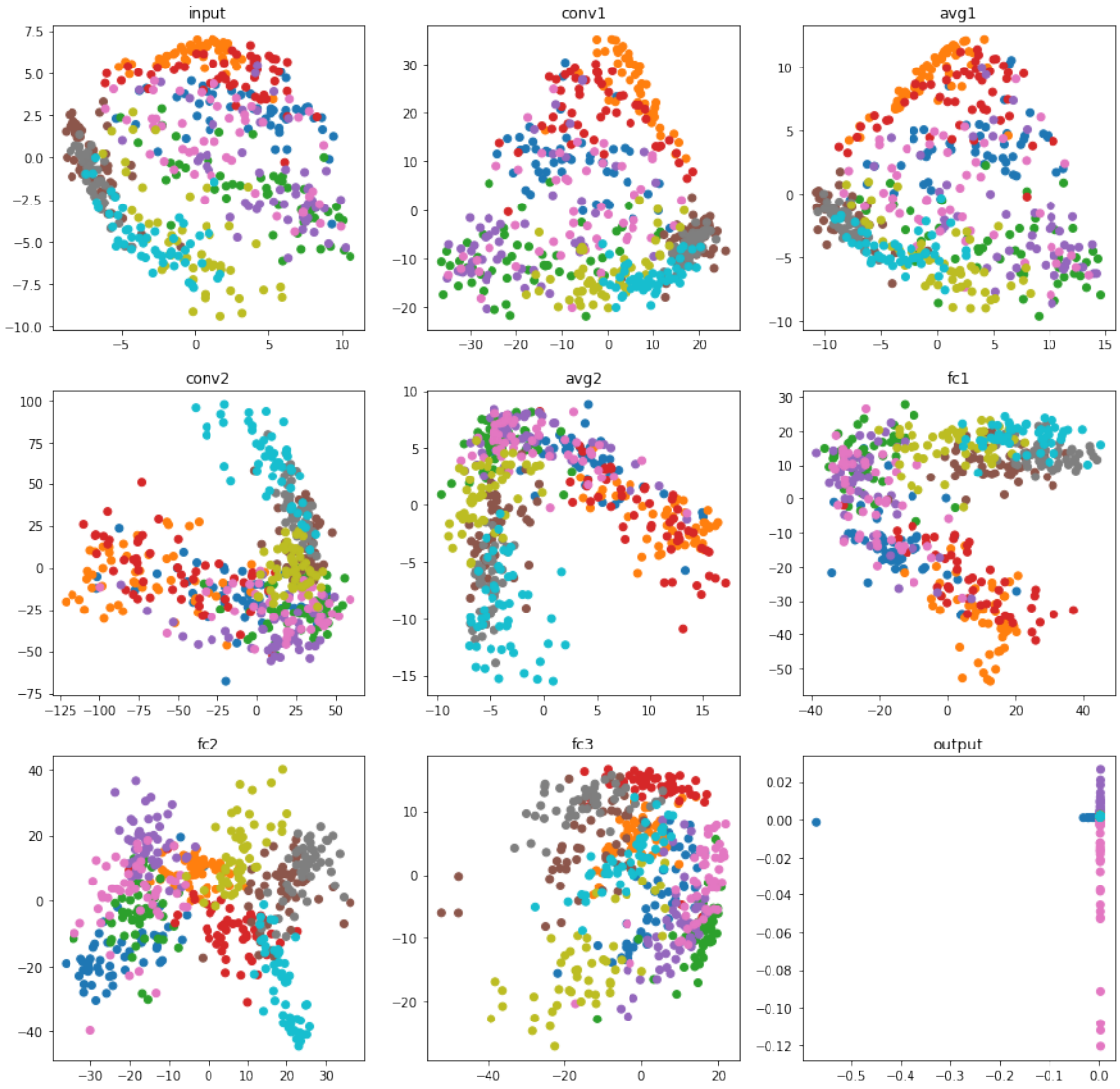
主成分分析（Principal Component Analysis，PCA）是一种常用的数据降维和特征提取技术。通过线性变换将原始数据投影到一个新的特征空间，使得投影后的数据具有最大的方差。PCA 的目标是找到数据中最重要的特征方向，即主成分，以便能够用较少的维度来表示数据。

在可视化任务中，通常选择将原始数据降至二维或三维；本次实验中，将测试集及其在神经网络中输出的隐藏值进行 PCA 处理，输出特征为二维，并以此为基础构建平面图。

原始测试集共含有 10 种类型的样例，每种 100 个，对每种类型的数据各采样 50 个，在输出图中用不同的颜色进行标识。观测颜色的数据点聚集情况和不同颜色的数据点分开情况，可在一定程度上反映神经网络前向传递过程中，对特征提取能力的好坏。

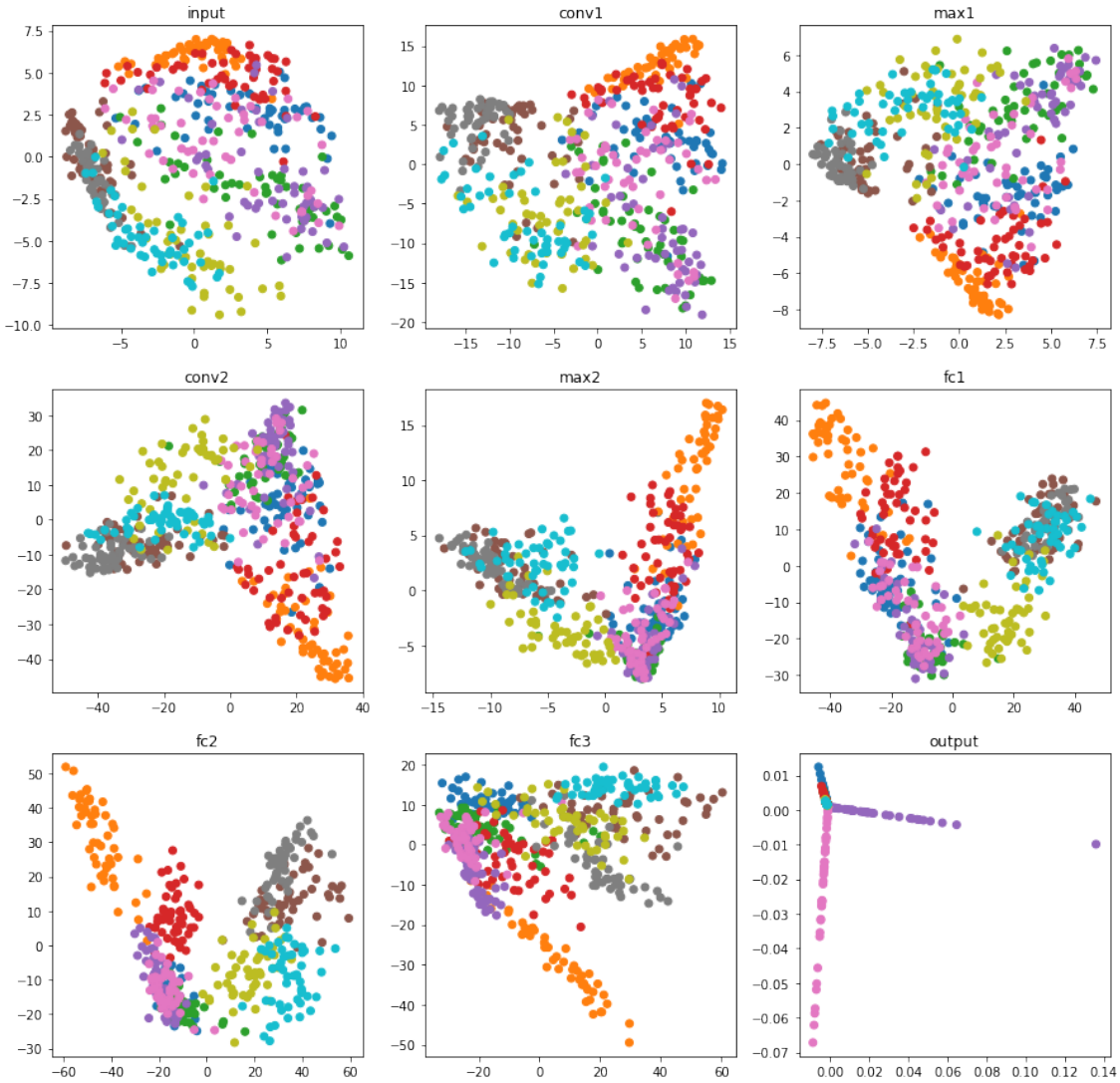
1.4 Stochastic Neighbor Embedding (t-SNE)

类似于 PCA，t-SNE（t-Distributed Stochastic Neighbor Embedding）是一种常用的非线性降维和数据可视化算法，可以帮助我们在低维空间中展示高维数据的结构和相似性。



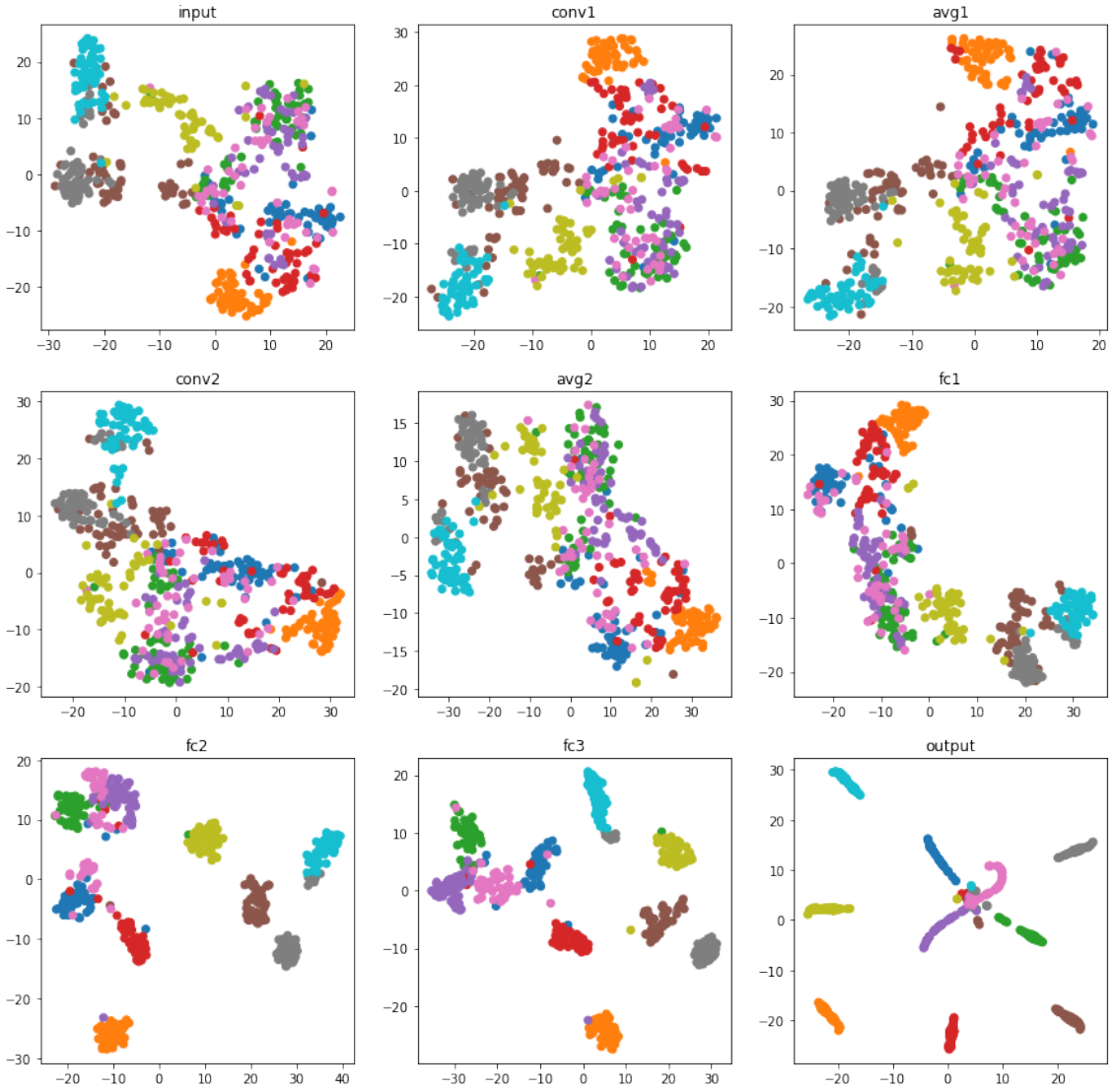
从左到右，从上到下依次为:
输入值，卷积层 1 输出，池化层 1 输出，
卷积层 2 输出，池化层 2 输出，全连接层 1 输出，
全连接层 2 输出，全连接层 3 输出，输出值

图 1–13 PCA 可视化 LeNet 各层隐藏值



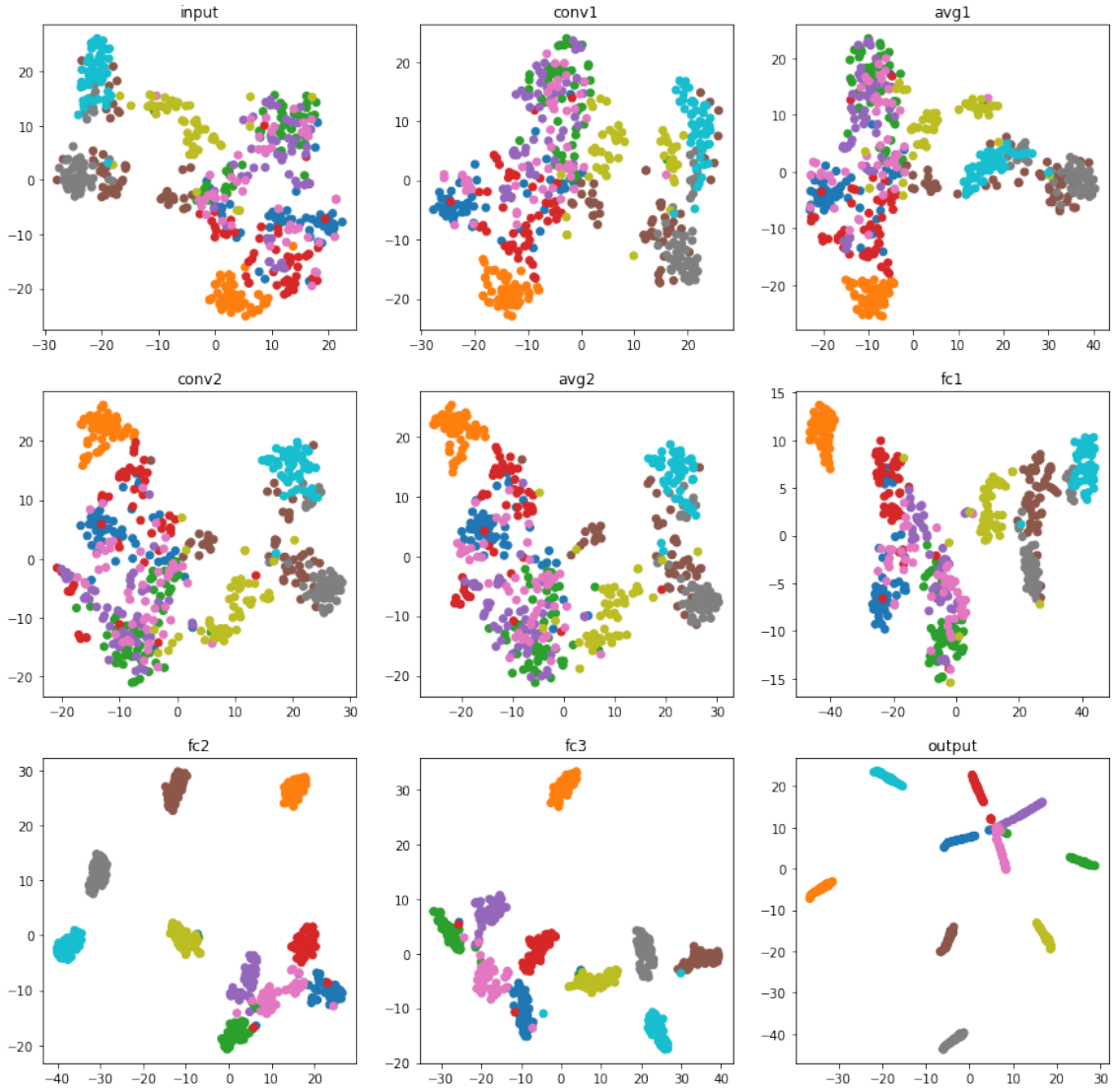
从左到右，从上到下依次为:
输入值，卷积层 1 输出，池化层 1 输出，
卷积层 2 输出，池化层 2 输出，全连接层 1 输出，
全连接层 2 输出，全连接层 3 输出，输出值

图 1–14 PCA 可视化 MyNet 各层隐藏值



从左到右，从上到下依次为:
输入值，卷积层 1 输出，池化层 1 输出，
卷积层 2 输出，池化层 2 输出，全连接层 1 输出，
全连接层 2 输出，全连接层 3 输出，输出值

图 1–15 t-SNE 可视化 LeNet 各层隐藏值



从左到右，从上到下依次为:
输入值，卷积层 1 输出，池化层 1 输出，
卷积层 2 输出，池化层 2 输出，全连接层 1 输出，
全连接层 2 输出，全连接层 3 输出，输出值

图 1–16 t-SNE 可视化 MyNet 各层隐藏值

第二章 Optional task 1: Image reconstruction

本次实验使用了经简化的数据集 Labeled Faces in the Wild (LFW)，包含 1000 张 rgb 人脸样本。实验旨在通过 VAE（变分自编码器）模型完成图像重建任务。并对 vae 编码器输出的特征向量进行线性插值以得到具有特定性质的图片。

2.1 Variational Autoencoder，VAE

变分自编码器（Variational Autoencoder，VAE）是一种生成模型，结合了自编码器（Autoencoder）和概率推断的思想。VAE 也是一种无监督学习算法，常用于生成新的数据样本。

2.1.1 VAE 架构

VAE 的主要由编码器（Encoder）和解码器（Decoder）两个组件组成。其中，编码器的任务是将输入数据映射到一个低维空间，解码器的任务是将低维空间中的向量重新映射回原始数据空间。特别的是，VAE 的编码器具有两个输出，均值向量和方值向量，用于表示数据特征的概率分布，对分布进行采样和重参数化（reparameterize）后得到传统意义上的潜在向量 Z。

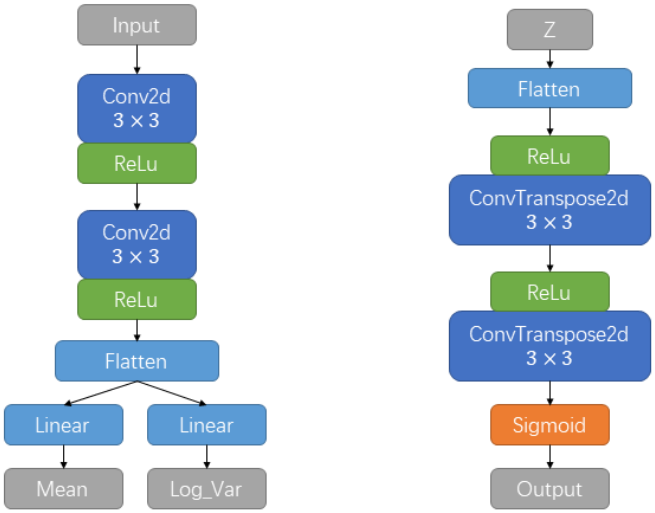
VAE 的损失函数由重构损失和 KL 散度损失两部分组成。重构损失度量了解码器生成的输出与原始输入之间的差异，而 KL 散度损失则度量了潜在向量的分布与预定义的高斯分布之间的差异。通过最小化这两个损失，VAE 得以学习到一个能够生成接近原始数据分布的模型。

实验任务涉及到对图像数据集进行操作，故使用卷积层以提取图像向量中的空间信息。具体而言，本次实验在编码器中使用了两层卷积层，在译码器中使用了两层转置卷积层。

VAE 在 pytorch 中的定义如下：

```
# Define the VAE model and its loss function

class VAE(nn.Module):
    def __init__(self, input_channel, height, width, latent_dim):
        super(VAE, self).__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(input_channel, 16, kernel_size=3, stride=2, padding=1),
```

```
            nn.ReLU(),
            nn.Conv2d(16, 32, kernel_size=3, stride=2, padding=1),
            nn.ReLU(),
            nn.Flatten()
        )

        self.fc_mean = nn.Linear(32 * (height // 4) * (width // 4), latent_dim)
        self.fc_log_var = nn.Linear(32 * (height // 4) * (width // 4), latent_dim)

        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, 32 * (height // 4) * (width // 4)), # 32 * 8 * 8
            nn.Unflatten(1, (32, height // 4, width // 4)),
            nn.ReLU(),
            nn.ConvTranspose2d(32, 16, kernel_size=4, stride=2, padding=1),
            nn.ReLU(),
            nn.ConvTranspose2d(16, input_channel, kernel_size=4, stride=2, padding=1),
            nn.Sigmoid()
        )

    def encode(self, x):
        hidden = self.encoder(x) # 2048
        mean = self.fc_mean(hidden)
        log_var = self.fc_log_var(hidden)
        return mean, log_var

    def reparameterize(self, mean, log_var):
        std = torch.exp(0.5 * log_var)
        epsilon = torch.randn_like(std)
        z = mean + std * epsilon
        return z

    def decode(self, z):
        return self.decoder(z)

    def forward(self, x):
        mean, log_var = self.encode(x)
        z = self.reparameterize(mean, log_var) # 20
        reconstruction = self.decode(z) # 3, 32, 32
        return reconstruction, mean, log_var


def vae_loss(reconstruction, x, mean, log_var):
    reconstruction_loss = F.binary_cross_entropy(reconstruction, x, reduction='sum')
    kl_loss = -0.5 * torch.sum(1 + log_var - mean.pow(2) - log_var.exp())
    return reconstruction_loss + kl_loss
```



编码器（左），译码器（右）

图 2–1 使用 PowerPoint 绘制的 VAE 结构

2.1.2 VAE 训练过程

为提高训练效率和模型的泛化能力，单个 epoch 中，将原始的数据集拆分为多个 batchsize 为 32 的 batch 进行训练，梯度下降在单个 batch 上进行以提高模型的泛化能力。训练在 gpu 上进行，选用的优化器为 Adam，500 个 epoch 后，VAE 基本收敛。

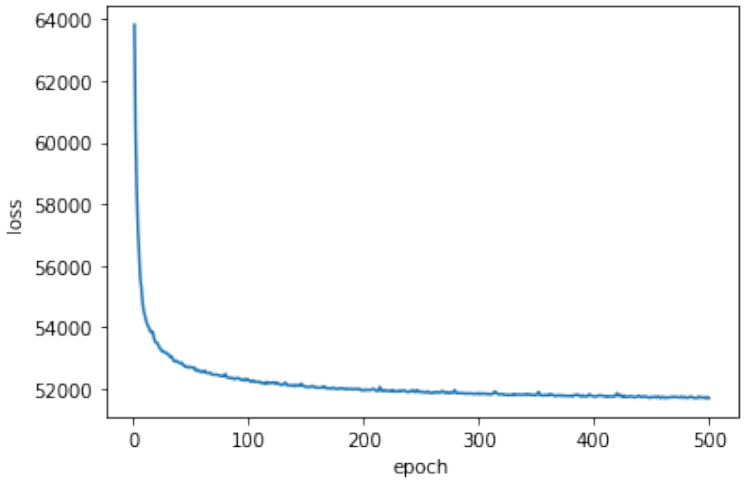


图 2–2 VAE 训练过程中的 loss curve

图 2–3 VAE 在训练集上的重构图像（前 400 个）

2.1.3 对编码器的输出进行线性插值

理论上而言，编码器输出的任一特征向量均可以通过解码器解码得到和输入同性质的输出，在本实验中，解码器应该能够输出大小为 32×32 的人脸照片。那么，如果对编码器输出的两个特征向量 z_1，z_2 进行线性插值得到的新向量 $(1-\alpha)z_1 + \alpha z_2$，解码器也应该能够生成具有特定性质的图像，我们将选用不同的 α 值构建具有不同性质的图像输出。随着 α 的变化，新生成的图像也会随之发生连续的变化。

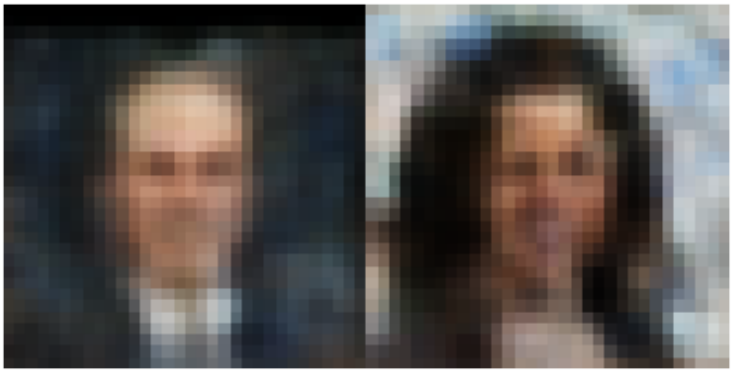
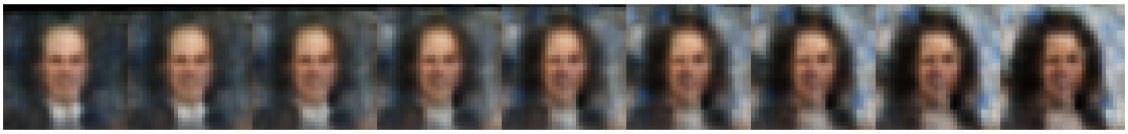


图 2–4 z_1，z_2 通过解码器生成的人脸



从左到右 α 分别为 $0.1, 0.2, ..., 0.9$
α 值越小，图像越接近 z_1，α 值越大，图像越接近 z_2，

图 2–5 z_1，z_2 在不同插值下通过解码器生成的人脸