哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

《模式识别》课程实验报告

实验二：字符识别

院　　系：机电工程学院自动化系

姓　　名：朱颖聪

学　　号：200320708

班　　级：自动化 7 班

日　　期：2023 年 12 月 30 日

目录

1 项目背景.....2
1.1 背景介绍.....2
1.2 数据集说明.....2
2 项目目标.....2
3 理论介绍.....2
4 实验过程及结果.....3
任务一：卷积神经网络.....3
任务二：K-L 变换+MLP 4
附录：关键代码注释.....7

字符识别

1 项目背景

1.1 背景介绍

字符识别作为计算机视觉领域的重要任务，涵盖了许多实际应用，如自然场景中的车牌识别、手写文字识别等。本次实验旨在比较并探索基于 K-L 变换和多层感知机（MLP）的特征提取与分类方法，以及基于卷积神经网络（CNN）的字符识别。这将帮助我们深入了解不同方法在字符识别任务中的性能和适用性。

1.2 数据集说明

本次实验所用数据集为 Chars74K 系列数据集。Chars74K 系列数据集是一个用于字符识别的综合数据集，包含英文大写字母和数字 0~9 的图像，图像大小为 28x28 像素，字符为白色，背景为黑色。该数据集由基础数据集（包括 Chars74K 数据集和网络数据集）以及经过旋转、缩小还原等操作的变换数据集构成。数据集分为训练集（Chars74K_train）和测试集（Chars74K_test）。训练集共计 74700 个图像，测试集则包含 6294 个独立的图像。

2 项目目标

任务一：

基于 K-L 变换和 MLP 的字符识别：

- **特征提取：** 利用 K-L 变换从原始图像中提取高效的特征向量。
- **分类方法：** 使用 MLP 进行字符分类。

任务二：

基于卷积神经网络的字符识别：

- **特征提取：** 利用卷积神经网络（CNN）自动学习图像特征。
- **分类方法：** 使用 CNN 进行字符分类。

3 理论介绍

1. K-L 变换：

K-L 变换是一种特征提取方法，通过线性变换将数据映射到一个新的坐标系，使得在新坐标系中数据的方差最大化。在字符识别中，K-L 变换可用于将原始图

像转换为更紧凑的特征表示，有助于提高后续分类器的性能，并且可以大大简化训练成本。

2. 多层感知机（MLP）：

多层感知机是一种前馈神经网络，由输入层、隐藏层和输出层组成。每个神经元都与上一层的所有神经元连接，具有权重和激活函数。

在字符识别中，MLP 可用于学习从 K-L 变换后的特征到字符类别的映射。通过训练，模型能够自动学习字符的抽象表示。

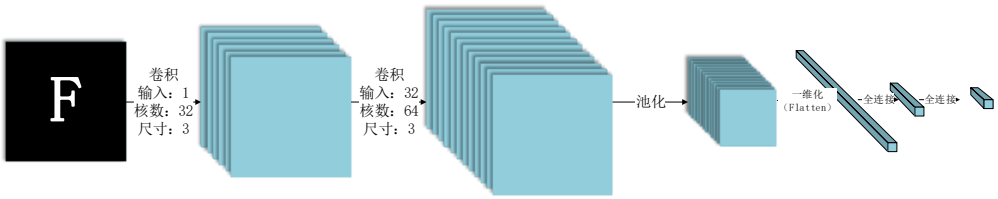
3. 卷积神经网络（CNN）：

卷积神经网络是一类包含卷积计算且具有深度结构的前馈神经网络，是深度学习的代表算法之一。卷积神经网络具有表征学习能力，能够按其阶层结构对输入信息进行平移不变分类，因此也被称为“平移不变人工神经网络”。

4 实验过程及结果

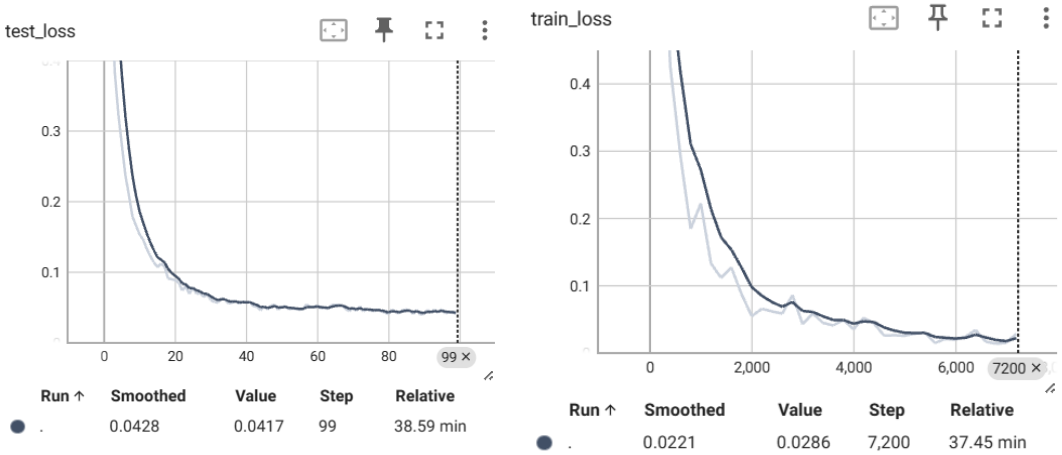
任务一：卷积神经网络

1. 网络结构：本次实验使用的卷积神经网络一共包含两个卷积层、一个池化层以及两个全连接层。



2. 选择优化器为 Adadelta，损失函数为 CrossEntropyLoss，设置 batch_size = 1024，训练 100 epochs，载入训练数据，训练网络。
3. 根据训练结果选取最优网络，测试并评估结果。

   以下为训练曲线：



训练损失和测试损失随着训练次数的增加不断下降，选取最后一个回合的模型进行测试。

```
Wrong! label: 7, prediction: 1, image_name: img008-002.png
Wrong! label: 7, prediction: 1, image_name: img008-013.png
Wrong! label: 7, prediction: 9, image_name: img008-018.png
Wrong! label: 8, prediction: 5, image_name: img009-053.png
Wrong! label: 9, prediction: 3, image_name: img010-047.png
测试图像数量: 6294, 误分类数量: 62, 分类准确率: 99.01%
```

准确率达到 99.01%

在该网络结构下，不需要额外的参数调整即可获得很好结果。

任务二：K-L 变换+MLP

首先，对训练集的图像计算协方差矩阵，计算特征值和特征向量，根据累计贡献率选取前 N 大的特征向量，对图像进行降维处理。（代码见附录）

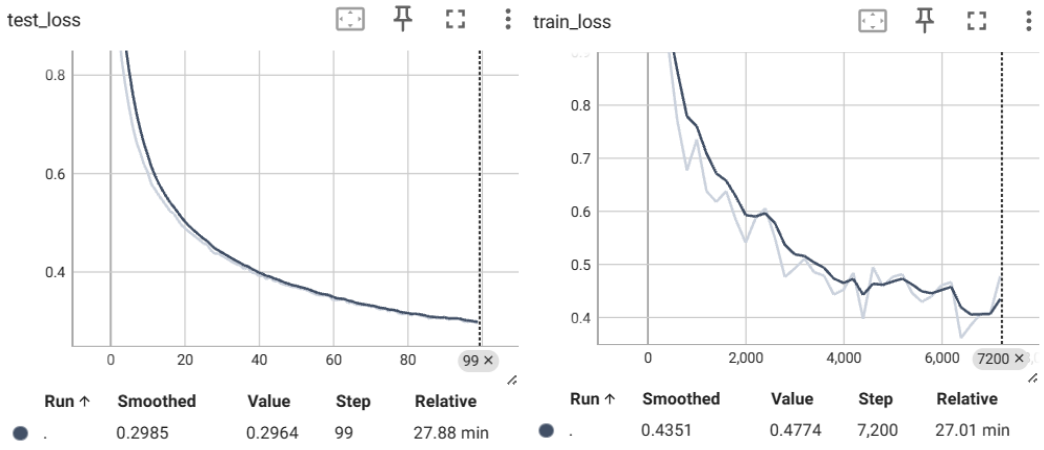
（1）选取累计贡献率 e=1，即相当于不使用 K-L 转换,输入向量维度为 784

设计网络结构为

```python
# 定义构造函数
def __init__(self):
    super(Classification, self).__init__()
    # 定义网络结构
    self.output = nn.Sequential(
        nn.Flatten(),
        nn.Linear(784, 128),
        nn.ReLU(True),
        nn.Dropout(0.5),
        nn.Linear(128, number_classes),
    )
# 定义前向传播函数
def forward(self, x):
    output = self.output(x)
    return output
```

选择优化器为 Adadelta，损失函数为 CrossEntropyLoss，设置 batch_size = 1024, 训练 100 epochs。

训练过程损失函数如下所示：



训练损失和测试损失随着训练次数的增加不断下降，选取最后一个回合的模

型进行测试。

```
Wrong! label: 9, prediction: T, image_name: img010-038.png
Wrong! label: 9, prediction: 0, image_name: img010-040.png
Wrong! label: 9, prediction: A, image_name: img010-046.png
Wrong! label: 9, prediction: J, image_name: img010-047.png
Wrong! label: 9, prediction: Y, image_name: img010-052.png
Wrong! label: 9, prediction: Y, image_name: img010-053.png
测试图像数量: 6294, 误分类数量: 510, 分类准确率: 91.90%
```

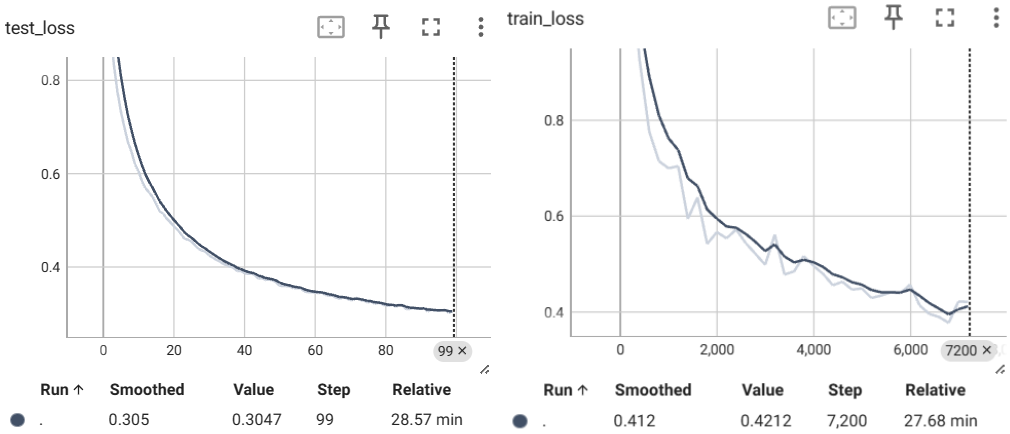
可以看到准确率仅 91.9%，与 CNN 网络差距较大。

（2）选取累计贡献率 e=0.99，降维后输入向量维数为 170

网络结构为

```
# 定义构造函数
def __init__(self,num,vec):
    super(Classification_KLT, self).__init__()
    # 定义网络结构
    self.flatten = nn.Flatten()
    self.vec = vec.t()
    self.output = nn.Sequential(
        nn.Linear(num, 128),
        nn.ReLU(True),
        nn.Dropout(0.5),
        nn.Linear(128, number_classes),
    )
# 定义前向传播函数
def forward(self, x):
    x = self.flatten(x)
    x = torch.mm(x,self.vec)
    output = self.output(x)
    return output
```

网络中的 self.vec 为特征向量，除输入外，其余网络结构与 e=1 时的基本一致。选择优化器为 Adadelta，损失函数为 CrossEntropyLoss，设置 batch_size = 1024，训练 100 epochs。



训练损失和测试损失随着训练次数的增加不断下降，选取最后一个回合的模型进行测试。

```
Wrong! label: 9, prediction: T, image_name: img010-038.png
Wrong! label: 9, prediction: 0, image_name: img010-040.png
Wrong! label: 9, prediction: A, image_name: img010-046.png
Wrong! label: 9, prediction: 1, image_name: img010-047.png
Wrong! label: 9, prediction: Y, image_name: img010-052.png
测试图像数量: 6294, 误分类数量: 522, 分类准确率: 91.71%
```

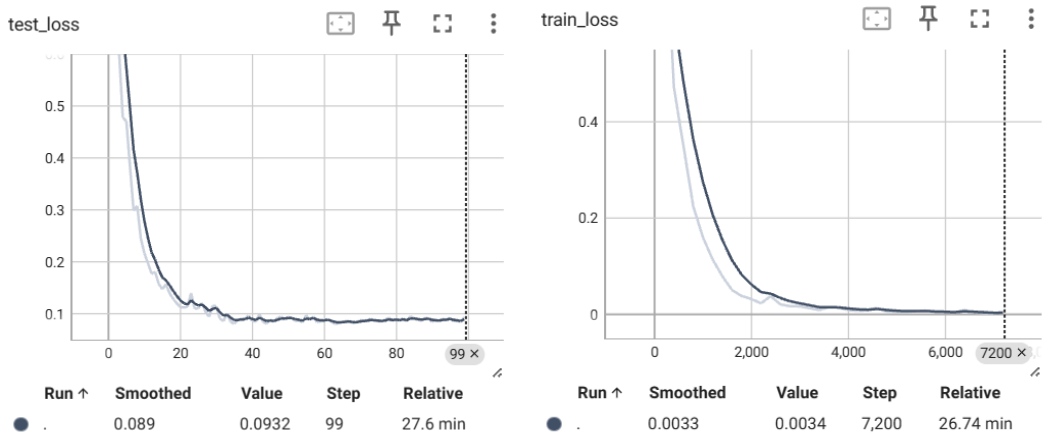
准确率为 91.71%，与贡献率 e=1 时的准确率基本一致，可以看出 K-L 转换降低特征向量维度的同时基本保留特征向量的所有信息，这样可以提高训练的效率，节省时间。

（3）选取累计贡献率 e=0.99，增加网络复杂度

网络结构为

```python
# 定义构造函数
def __init__(self,num,vec):
    super(Classification_KLT, self).__init__()
    # 定义网络结构
    self.flatten = nn.Flatten()
    self.vec = vec.t()
    self.output = nn.Sequential(
        nn.Linear(num, 512),
        nn.ReLU(True),
        nn.Linear(512, 256),
        nn.ReLU(True),
        nn.Linear(256, 128),
        nn.ReLU(True),
        nn.Dropout(0.5),
        nn.Linear(128, number_classes),
    )
# 定义前向传播函数
def forward(self, x):
    x = self.flatten(x)
    x = torch.mm(x,self.vec)
    output = self.output(x)
    return output
```

选择优化器为 Adadelta，损失函数为 CrossEntropyLoss，设置 batch_size = 1024，训练 100 epochs。



训练损失和测试损失随着训练次数的增加很快达到收敛，在第 60 回合中训练损失最低，故选择第 60 回合作为测试模型。

```
Wrong! label: 8, prediction: 3, image_name: img009-032.png
Wrong! label: 8, prediction: 3, image_name: img009-047.png
Wrong! label: 9, prediction: F, image_name: img010-021.png
Wrong! label: 9, prediction: S, image_name: img010-022.png
Wrong! label: 9, prediction: J, image_name: img010-047.png
测试图像数量: 6294, 误分类数量: 105, 分类准确率: 98.33%
```

准确率为 98.33%，可以看出提高模型复杂度可以大大提高准确率，与 CNN 网络的准确率相差不大。

附录：关键代码注释

K-L 转换代码

```python
def getX(dataset):
    X = []
    for i in range(1, len(dataset) + 1):
        img, label = dataset[i - 1]
        img = np.array(img)
        img = img.flatten()
        X.append(img)
    mean = np.mean(X, axis=0)
    X = np.array(X) - mean  # 去均值
    return X

def getEigenVector(X, e):
    Y = np.dot(np.transpose(X), X) / X.shape[0]  # 样本矩阵降维：m*wh * wh*m -> m*m
    eigenvalue, feature_vector = np.linalg.eig(Y)  # 计算特征值和特征向量(得到的是列向量构成的特征矩阵)
    feature_vector = np.transpose(feature_vector)  # 将特征向量转置一下
    eigen = []
    for i in range(0, eigenvalue.shape[0]):  # 将特征值和特征向量绑到一起，方便排序
        eigen.append([eigenvalue[i], feature_vector[i]])
    eigen = sorted(eigen, key=lambda eigen: eigen[0], reverse=True)  # 根据特征值大小，从大到小排序
    eigen_num = 0  # 根据重构精度确定特征脸数目
    eigenvalue_sum = 0
    for i in range(0, eigenvalue.shape[0]):
        eigenvalue_sum += eigen[i][0]
        if eigenvalue_sum / eigenvalue.sum() >= e:
            eigen_num = i + 1
            break
    eigen_vector = []
    for i in range(0, eigen_num):
        vec = eigen[i][1]
        eigen_vector.append(vec)
    return eigen_num, np.array(eigen_vector)
```

主要训练过程

```python
print('开始训练……')
for epoch in range(epochs):
    CC.train()
    loss_train = []
    for index, (images, labels) in enumerate(data_loader_train):
        images = images.to(device)
        labels = labels.to(device)
        CC_output = CC(images)
        loss = lossFun(CC_output, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        loss_train.append(loss.data.item())
        total_train_step += 1
        print('epoch:{}, batch:{}, total_train_step:{}, loss:{:.6f}'.format(epoch + 1, index + 1,
                                                                            total_train_step, loss.data.item()))
        if total_train_step % 200 == 0:  # 每训练100步保存一次模型
            writer.add_scalar("train_loss", loss.item(), total_train_step)
    if (epoch + 1) % 10 == 0:
        torch.save(CC.state_dict(), './models'+suffix+'/CC_' + str(epoch + 1) + '.pth')
        torch.save(optimizer.state_dict(), './models'+suffix+'/CC_optimizer_' + str(epoch + 1) + '.pth')
    CC.eval()
    loss_test = []
    print('开始验证……')
    for index, (images, labels) in enumerate(data_loader_test):
        images = images.to(device)
        labels = labels.to(device)
        CC_output = CC(images)
        loss = lossFun(CC_output, labels)
        loss_test.append(loss.data.item())
    print('平均损失：', np.mean(loss_test))
    # 记录损失
    loss_epochs_train.append(np.mean(loss_train))
    loss_epochs_test.append(np.mean(loss_test))
    writer.add_scalar("test_loss", np.mean(loss_test), epoch)
    epochs_x.append(epoch + 1)
```