

机器学习结课报告

|  |  |
| --- | --- |
| 学年学期： | 2025~2026学年第一学期 |
| 学 院： | 计算机科学与工程学院 |
| 报告题目： | 卷积神经网络在人脸识别中的应用 |
| 专业班级： | 计科23-1、2 |
| 姓 名： | 王千龙、闫凯杰、吴学喆 |
| 考核等级： | □优秀 □良好 □中等 □及格 □不及格 |
| 完成日期： | 二零二五年十月 |

**目 录**

**第1章 灵感来源及实现过程 1**

1.1 灵感来源 1

1.2 实现过程 1

**卷积神经网络在人脸识别中的应用**

**第1章 灵感来源及实现过程**

**1.1** **灵感来源**

在机器学习的课程中，接收到老师上传的资料，试图运行2DPCA.py文件，但是代码存在着一些问题，比如路径问题，这可以通过自己修改路径来改正。源代码是mac编写的，不兼容windows，有更好的兼容编写方式。另外，opencv不支持中文路径，解压人脸识别文件后有中文路径，image解析后会返回None，直接报错。并且由于numpy版本太老，矩阵乘法为\*，不兼容@，而且有些方法在当前版本numpy下废弃了，比如np.mat。

在对上述问题进行修改过后，运行后得到的结果为降到K（10/20/30/40）维时识别准确率的折线图。代码是通过PCA算法来对传入的图片数据进行训练，将人脸图像看作高维空间中的点，使用PCA找到最能代表人脸变化的主要方向（主成分），最后用少量主成分来近似表示所有人脸。

但是在这个程序中，如果要添加新人，需要重新训PCA，而且在我们私下进行添加数据比对时，发现背景也会对比对产生影响，这让我们想到了学校的门禁系统，学校的门禁系统只需要我们提供一张照片就可以进行比对识别，而且准确率也相当高，同时受到背景的影响也比较低，这让我们意识到学校的门禁系统不会是使用PCA算法实现的，经过查阅，现代的人脸识别技术大多是使用卷积神经网络（cnn）实现的，这让我们产生了使用cnn来实现人脸识别的想法。

.......

**1.2** **实现过程**

仍然是使用目前存有的40\*10的人脸数据进行训练

class ORLDataset(Dataset):

def \_\_init\_\_(self, root, train=True, k=5, transform=None, n=40):

root: 数据集根目录，train: 是否为训练模式k: 训练集图片数量（每人前k张，这里的k只是先赋予了初始值，之后会对k进行更改）n: 总人数（40人）。数据组织：每人10张图片，前k张训练，剩余测试

def init\_data\_labels(self, root, train, k, n):

data = [] # 存储图片路径的列表

labels = [] # 存储对应标签的列表

# 遍历每个人物

for i in range(n):

# 生成人物文件夹名，如 s1, s2, s3, ..., sn

person = f"s{i+1}"

# 生成该人物所有的图片文件名列表: ['1.BMP', '2.BMP', ..., '10.BMP']

imgs = [f"{j}.BMP" for j in range(1, 11)]

# 根据训练/测试模式选择图片

if train:

# 训练模式：选择前k张图片

selected = imgs[:k]

else:

# 测试模式：选择后10-k张图片

selected = imgs[k:]

# 遍历选中的图片文件

for img\_name in selected:

# 构建完整的图片路径：root/person/img\_name

img\_path = os.path.join(root, person, img\_name)

# 将图片路径添加到数据列表

data.append(img\_path)

# 将人物标签添加到标签列表（i从0开始，对应第1个人）

labels.append(i)

# 返回图片路径列表和对应的标签列表

return data, labels

def \_\_len\_\_(self):

return len(self.data)

然后对数据进行预处理

transform = transforms.Compose([

transforms.Resize((112, 92)), *# 统一尺寸*

transforms.ToTensor(),*# 转为Tensor*

transforms.Normalize((0.5, ), (0.5, )) *# 归一化*

])

获取图片并且将其转换为灰度图。

def \_\_getitem\_\_(self, idx):

img\_path = self.data[idx] # 获取图片路径

image = Image.open(img\_path).convert('L') # 打开并转为灰度图

if self.transform:

image = self.transform(image) # 应用数据变换

label = self.labels[idx] # 获取对应标签

return image, label

在训练模型检查一遍数据，确保一切正常后再开始训练。

def print\_data(self, num = 5):

infos = "查看内部存储的data信息:"

print(infos)

pprint(self.data[:num])

print(len(self))

def print\_label(self, num = 5):

infos = "查看内部储存的label信息:"

print(infos)

pprint(self.labels[:num])

print(len(self.labels))

def analysis\_image\_label(self, idx):

image, label = self[idx]

pprint(image)

print(image.shape)

print(f"这是第{label}人.")

# 训练集: 每人前6张图片

train\_dataset = ORLDataset(root, k=6, transform=transform, n=45)

# 测试集: 每人后4张图片

test\_dataset = ORLDataset(root, train=False, k=6, transform=transform, n=45)

创建训练集和测试集实例

train\_dataset = ORLDataset(root, k=6, transform=transform, n=45)

test\_dataset = ORLDataset(root, train=False, k=6, transform=transform, n=45)

train\_loader = DataLoader(train\_dataset, batch\_size=16, shuffle=True)

test\_loader = DataLoader(test\_dataset, batch\_size=16, shuffle=False)

# batch\_size=16: 每批16个样本

# shuffle=True: 每个epoch打乱数据顺序，用以防止模型记忆样本顺序，提高泛化能力

def \_\_init\_\_(self, num\_classes=40):

super(FaceCNN, self).\_\_init\_\_()

# 第一卷积层: 1个输入通道 -> 32个输出通道

self.conv1 = nn.Conv2d(1, 32, kernel\_size=3, padding=1)

# 池化层: 2x2最大池化，步长为2

self.pool = nn.MaxPool2d(2,2)

# 第二卷积层: 32个输入通道 -> 64个输出通道

self.conv2 = nn.Conv2d(32, 64, kernel\_size=3, padding=1)

# 第一个全连接层: 64\*28\*23 -> 256个神经元

self.fc1 = nn.Linear(64\*28\*23, 256)

# 第二个全连接层: 256 -> num\_classes个输出

self.fc2 = nn.Linear(256, num\_classes)

之后首先通过卷积、ReLU激活和池化操作从输入图像中逐步提取层次化特征，将112×92的输入图像压缩为28×23的特征图；然后通过展平操作和两个全连接层将空间特征转换为分类得分，最终输出每个样本在所有类别上的预测分数。

def forward(self, x):

x = self.pool(F.relu(self.conv1(x))) # 112x92 -> 56x46

x = self.pool(F.relu(self.conv2(x))) # 56x46 -> 28x23

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

x = F.relu(self.fc1(x))

x = self.fc2(x)

return x

之后便是对数据进行训练、测试以及绘制并输出学习损失曲线和测试集的准确性曲线。

class Train:

def \_\_init\_\_(

self,

num\_epochs = 20

):

*# 初始化训练类，自动执行训练流程*

self.init\_infos = self.\_prepare\_all() *# 准备训练所需的所有组件*

self.model = self.init\_infos[1] *# 获取模型实例*

self.train\_losses = self.train(self.init\_infos, num\_epochs) *# 执行训练并记录损失*

self.accuracy = self.train\_accuracy(self.init\_infos) *# 计算测试集准确率*

def \_prepare\_all(self):

*# 准备训练环境：设备、模型、损失函数和优化器*

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu') *# 自动选择GPU或CPU*

model = FaceCNN(num\_classes=45).to(device) *# 创建人脸识别模型并移至设备*

criterion = nn.CrossEntropyLoss() *# 使用交叉熵损失函数*

optimizer = optim.Adam(model.parameters(), lr = 0.001) *# 使用Adam优化器，学习率0.001*

init\_infos = [device, model, criterion, optimizer] *# 打包训练信息*

return init\_infos

def train(self, init\_infos, num\_epochs = 20):

*# 模型训练主循环*

device, model, criterion, optimizer = init\_infos *# 解包训练信息*

train\_losses = [] *# 记录每个epoch的平均损失*

for epoch in range(num\_epochs):

model.train() *# 设置模型为训练模式*

running\_loss = 0.0 *# 累计损失*

*# 使用tqdm创建进度条，显示训练进度*

loader = tqdm(train\_loader, desc=f"Train Epoch {epoch+1}", unit="batch", ncols=100)

for images, labels in loader:

*# 将数据移至设备（GPU/CPU）*

images, labels = images.to(device), labels.to(device)

optimizer.zero\_grad() *# 清空梯度*

outputs = model(images) *# 前向传播，获取预测结果*

loss = criterion(outputs, labels) *# 计算损失*

loss.backward() *# 反向传播，计算梯度*

optimizer.step() *# 更新模型参数*

running\_loss += loss.item() *# 累计损失值*

loader.set\_postfix(loss=loss.item()) *# 在进度条中显示当前batch的损失*

*# 计算该epoch的平均损失并记录*

train\_losses.append(f"{running\_loss/len(train\_loader):.4f}")

return train\_losses

def get\_train\_plot(self):

*# 绘制训练损失曲线图*

train\_losses\_float = [float(x) for x in self.train\_losses] *# 将损失转换为浮点数*

plt.figure(figsize=(8,5))

plt.plot(train\_losses\_float, marker='o', linestyle='-', color='blue') *# 绘制损失曲线*

plt.title("学习损失曲线")

plt.xlabel("轮次")

plt.ylabel("平均损失")

plt.grid(True) *# 显示网格*

plt.tight\_layout()

plt.show()

def train\_accuracy(self, init\_infos):

*# 在测试集上评估模型准确率*

device, model, \_, \_ = init\_infos

model.eval() *# 设置模型为评估模式*

correct = 0 *# 正确预测数量*

total = 0 *# 总样本数量*

accuracy = [] *# 记录准确率*

with torch.no\_grad(): *# 禁用梯度计算，节省内存和计算资源*

for images, labels in test\_loader:

images, labels = images.to(device), labels.to(device)

outputs = model(images) *# 前向传播*

\_, predicted = torch.max(outputs.data, 1) *# 获取预测类别（最大值的索引）*

total += labels.size(0) *# 累计总样本数*

correct += (predicted == labels).sum().item() *# 累计正确预测数*

accuracy.append(f"{100\*correct / total:.2f}") *# 记录当前准确率*

return accuracy

def get\_train\_accuracy\_plot(self):

*# 绘制测试准确率曲线图*

accuracy\_float = [float(x) for x in self.accuracy] *# 将准确率转换为浮点数*

plt.figure(figsize = (8, 5))

plt.plot(accuracy\_float, marker='o', color = 'red') *# 绘制准确率曲线*

plt.xlabel("测试数据集")

plt.ylabel("准确性 (%)")

plt.title("测试集的准确性")

plt.grid(True) *# 显示网格*

plt.ylim(50, 100) *# 设置y轴范围*

plt.show()

因为一开始是想要实现可以实时对摄像头拍摄到的图像进行检测的程序，所以同时编辑程序，调用默认的摄像头来进行比对。可以在开始的训练的人脸集合中，加入我们现实拍摄的照片进行训练.虽然我们开始是想使用cnn来实现人脸识别，但是数据样本不足，所以导致最终实现的效果其实与使用PCA实现的效果类似，仍然是对摄像头识别到的人脸与之前训练的模型进行比对，只能称作为PCA算法的“改进版”。

class RealTimeDetection:

def \_\_init\_\_(self, infos, using\_real\_name = False):

*# 初始化实时人脸检测类*

real\_names = ["wly", 'wql', 'wwd', 'wxz', 'ykj']  *# 真实姓名列表*

person\_names = [f"Person {i+1}" for i in range(40)] *# 默认人名列表*

if using\_real\_name:

person\_names.extend(real\_names) *# 如果使用真实姓名，扩展人名列表*

cap = cv2.VideoCapture(0) *# 打开默认摄像头*

*# 加载人脸检测器*

face\_cascade=cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade\_frontalface\_default.xml')

self.start(person\_names, cap, face\_cascade, infos) *# 开始实时检测*

def start(self, person\_names, cap, face\_cascade, infos):

*# 实时人脸识别主循环*

device, model, \_, \_ = infos *# 解包模型信息*

model.eval() *# 设置模型为评估模式*

with torch.no\_grad(): *# 禁用梯度计算*

while True:

ret, frame = cap.read() *# 读取摄像头帧*

if not ret:

break

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY) *# 转换为灰度图*

*# 检测人脸，返回人脸位置坐标*

faces = face\_cascade.detectMultiScale(gray, 1.3, 5)

for (x, y, w, h) in faces:

*# 提取人脸区域*

face\_img = gray[y:y+h, x:x+w]

*# 转换为PIL图像并预处理*

face\_img = Image.fromarray(face\_img).convert('L')

face\_img = transform(face\_img).unsqueeze(0).to(device) *# 添加批次维度并移至设备*

outputs = model(face\_img) *# 模型预测*

\_, predicted = torch.max(outputs.data, 1) *# 获取预测结果*

name = person\_names[predicted.item()] *# 根据预测索引获取人名*

*# 在图像上绘制人脸框*

cv2.rectangle(frame, (x, y), (x+w, y+h), (0,255,0), 2)

*# 在图像上显示人名*

cv2.putText(frame,name,(x,y-10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (0,0,255), 2)

cv2.imshow("Face Recognition", frame) *# 显示处理后的帧*

if cv2.waitKey(1) & 0xFF == ord('q'):  *# 按'q'键退出*

break

cap.release() *# 释放摄像头*

cv2.destroyAllWindows() *# 关闭所有OpenCV窗口*

如果提前导入了真实的人脸图片进行训练，在这一步的调用摄像头进行比对时，可以识别出摄像头所拍摄到的人像对应的名称。

*# 创建Train类的实例，自动执行整个训练流程*

train = Train()

*# 绘制并显示训练损失曲线图*

*# 展示模型在训练过程中损失值随epoch的变化趋势*

train.get\_train\_plot()

*# 绘制并显示测试集准确率曲线图*

*# 展示模型在测试集上的准确率变化*

train.get\_train\_accuracy\_plot()

*# 创建实时人脸检测实例并启动摄像头识别*

*# train.init\_infos: 传递训练好的模型和设备信息*

*# using\_real\_name=False: 不使用真实姓名显示识别结果，如果要使用真实的姓名显示识别结果，则将其改为Ture*

*# 程序会打开摄像头，实时检测人脸并进行识别标注*

realtimedetection = RealTimeDetection(train.init\_infos, using\_real\_name=False)

**1.2.1** **运行结果**

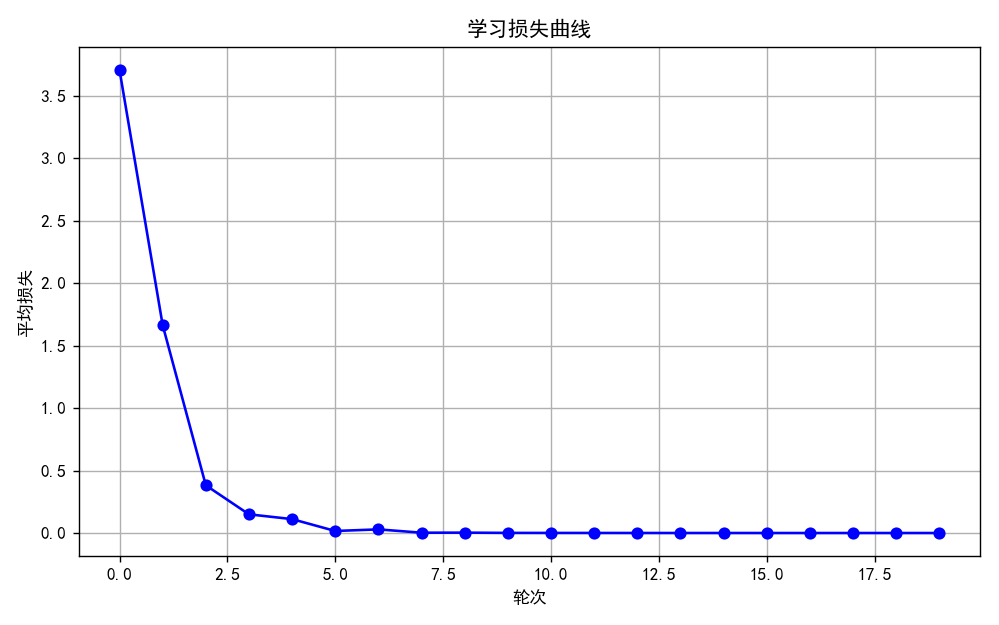


图1.1 学习损失曲线

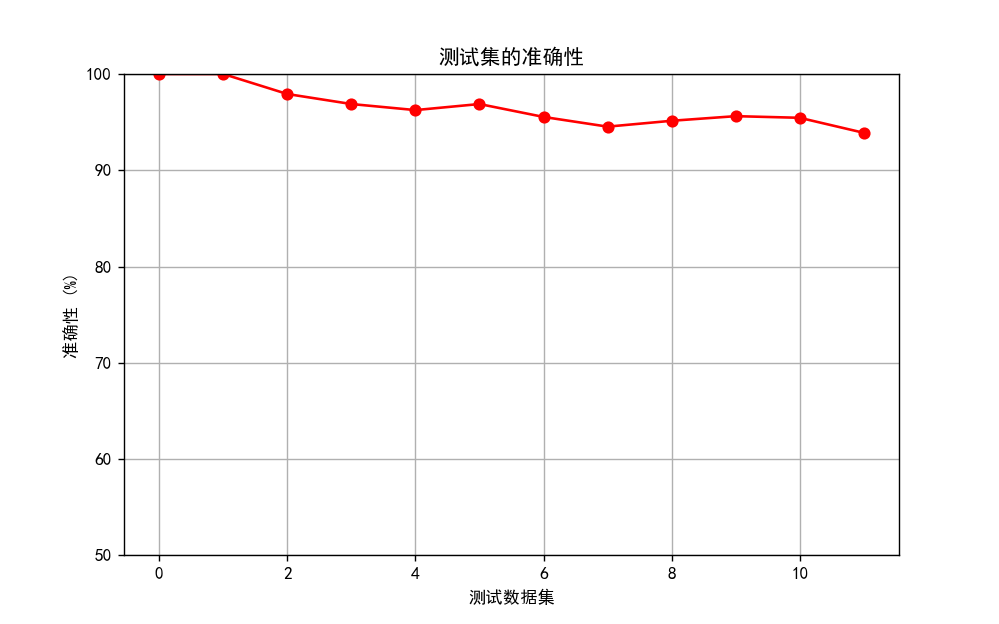
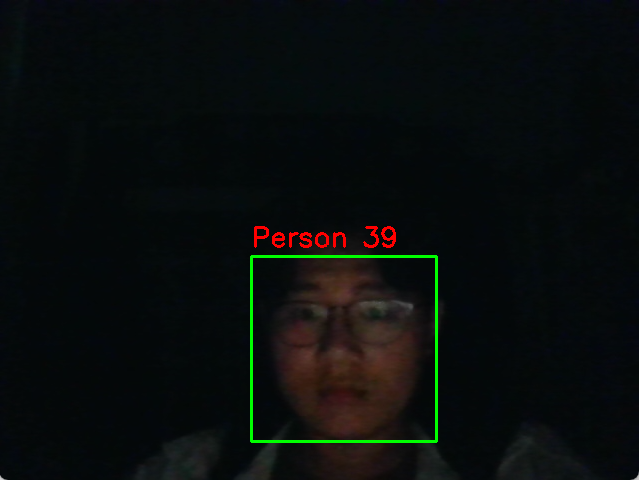


图1.2 测试集准确性曲线



调用摄像头进行人脸比对，由于没有提前导入真人图片进行训练，这里的会识别为特征相近的Person 39

参考文献 (黑体小二)

[1]周志华.机器学习

[2]张伟楠，赵寒烨，俞勇.动手学机器学习 北京：人民邮电出版社，2023.8