廣東工業大學

课 程 报 告

课程名称	模式识别
题目名称	人脸图像识别
专业班级	集成电路设计与集成系统创新班
学号姓名	谭家骏 3122009754
	张扬 3222009770
	李智欣 3222009768
	游垚明 3122009761
指导教师	邢延

2024 年 12 月

目录

1 模式识别系统目标与任务

1.1 设计背景与任务

生物特征识别技术是利用人体生物特征进行身份认证的一种技术。生物特征是人的内在属性，具有很强的自身稳定性和个体差异性。一般来说，人类的身份识别分为特征物品、特征知识和人类生物特征三类。人脸识别属于人体生物特征。基于人脸面部特征的识别具有主动性、非侵犯性和用户友好性等许多有点，是一种更方便、更直接、更容易被人们接受的识别方法。

在本次任务中，我们分别采用了 PCV-SVM、***、***和***方法，完成了对人脸图像的特征识别，如性别、年龄、表情和种族等。对各特征提取、特征降维、机器学习等算法进行分析和优化，最后进行分类效果对比。

1.2 数据来源与数据特点

数据来源于麻省理工学院 MIT2004 秋季的模式识别课程。课程名称为 MAS622J/1.126J: Pattern Recognition and Analysis。

原始数据：faceDR、faceDS、rawdata

数据说明：

表 1.2 数据说明

文件名	文件内容
faceDR	每一个人脸的数据说明，2000 个人脸，即类别卷标
faceDS	每一个人脸的数据说明，1000 个人脸，即类别卷标
rawdata	各个人类的原始图像数据，用编号关联卷标

经过数据预处理，数据存在缺失、图像存在尺寸不匹配和图像质量较差等问题。数据集包含 3993 个人脸图像，其中 3991 个大小为 16kb，2 个大小为 256kb，均为二维的单通道灰度数据，每个像素用 8bit 处理；特征维度为 128*128，即 16384。卷标包含 sex（male、female）、age（child、teen、adult、senior）、race（white、black、asian、hispanic）、face（smiling、serious、funny）、prop。类别上，sex 和 age、face 较为均衡，但 race 中 white 和 black 占比极高，prop 数据较少，不纳入模式识别中。数据量可能不足以训练一个复杂的深度学习模型，但是对于简单的分类器或传统的机器学习方法是足够的。

1.3 开发环境说明

算法测试环境基于 Windows 11 ， CPU 型号 Intel(R) Core(TM) i7-11800H @ 2.30GHz@2.30 GHz。编程语言为 python，版本为 python3.9.13，运行平台为 Visual Studio Code。

2 PCA+SVM 算法

2.1 设计框架

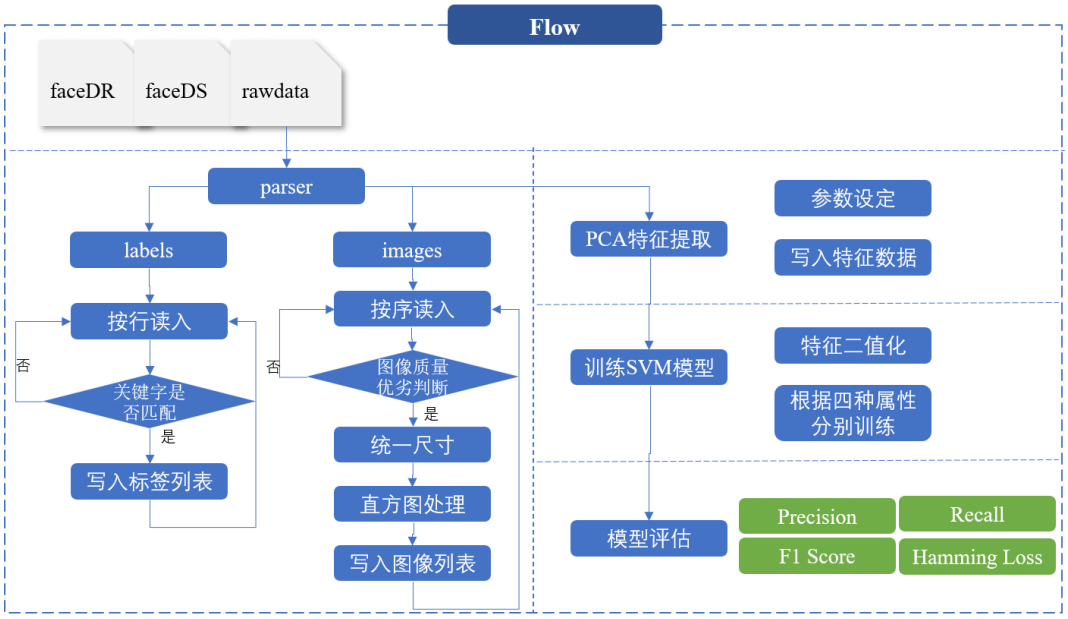


图 2.1.1 设计框架

2.2 图像和卷标文件解析

2.2.1 图像文件解析与预处理

按文件顺序读入图片后，将数据尺寸统一整理为 128*128 的 unit8 的二维数组，在直方图均衡后再展平为一维数组，写入 images 列表。其中，每一行代表一个图片，每一列代表一个维度。

数据集中包含了大小为 256kb 的图片，无法统一格式。由于仅有编号为 2412 和 2416 两张图，错误率小，手动删除。

直方图均衡化是一种图像增强技术，旨在增强图像的全局对比度。通过重新分配图像的灰度直方图，使得输出图像的直方图分布更加均匀、细节更加明显，使图像更容易分析处理。图像的直方图均衡化是通过离散形式的累计分布函数求解的，映射方法是：

$$s_k = \sum_{j=0}^{k} \frac{n_j}{n} \quad k = 0,1,2,\ldots,L-1 \tag{1}$$

其中，s_k 指当前灰度级经过累计分布函数映射后的值，n 是图像中像素的总和，n_j 是当前灰度级的像素个数，L 是图像中的灰度级总数。

从图 2.2.1--2.24 四张图片可以看出，直方图也增强了图像的对比度。

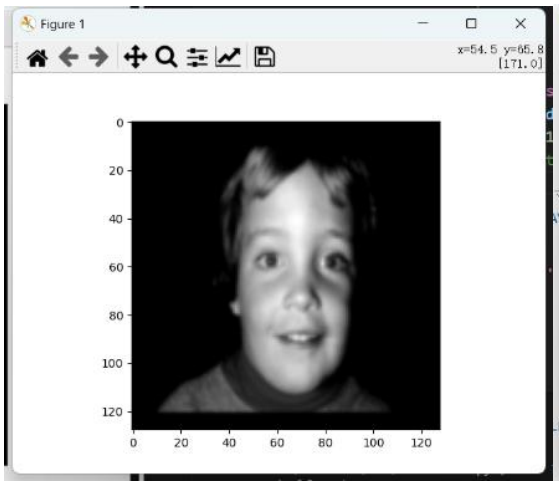


图 2.2.1 1223 直方图前

图 2.2.2 1223 直方图后

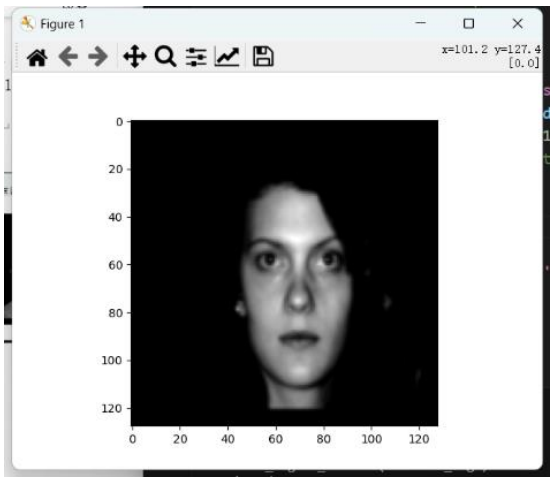


图 2.2.3 2783 直方图前

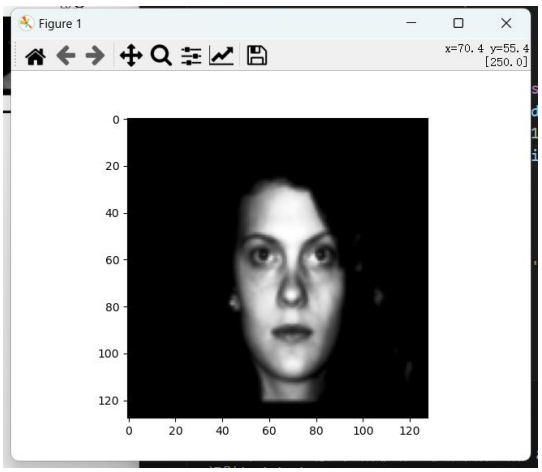


图 2.2.4 2783 直方图后

2.2.2 卷标文件解析与预处理

卷标文件分别为 faceDC 和 faceDR，内容格式为序号-性别-年龄-种族-表情-其他，共有五个维度，例如：3223 (_sex female) (_age senior) (_race black) (_face smiling) (_prop '(hat))。解析器采用以空格分离字符，按关键字匹配，将卷标数字化。存在数据缺失或单词拼写错误的，标记为-1，在后续正常训练分类器，在评估时忽略负值即可避免错误标签影响。卷标属性数字化如下表：

表 2.2.2 卷标属性对照表

属性	卷标				
sex	male 0	female 1	error -1		
age	child 0	teen 1	adult 2	senior 3	error -1
race	white 0	black 1	asian 2	hispanic 3	error -1
face	smiling 0	serious 1	funny 2	error -1	

按行读入卷标，关键字匹配后写入对应卷标值，当关键字为 missing 时，跳过这一行。最后按行写入列表 labels，一个元素代表一张图像的属性卷标，不储

存卷标序号。faceDR 和 faceDS 分别写入后，合并标签，再删除图像预处理时标记的错误图像的标签。

2.3 主成成分分析（PCA）

2.3.1 算法原理

主成成分分析（PCA）是一种常用的无监督学习方法，这一方法利用对原坐标系做旋转变换，即正交变换，把由线性相关变量表示的观测数据转换为少数几个由线性无关变量的数据，线性无关的变量称作主成分。通过 PCA 可以完成数据的降维和特提取。

坐标变换后的坐标轴具有两个性质，一是最大可分性，即样本点在这个坐标轴上的投影尽可能分开，对应 PCA 最大方差理论；二是最近重构性，即样本点到这个坐标轴的距离都足够近，对应 PCA 最小平均误差理论。两种理论方法是等价的，下面分析 PCA 最小平均误差理论的数学原理。

考虑整个训练集，原样本点 x_i 与基于投影重构的样本点 x_i'之间的距离为

$$\sum_{i=1}^{m} || \sum_{j=1}^{d'} z_{ij} w_j - x_i|_2^2 = \sum_{i=1}^{m} z_i^T z_i - 2 \sum_{i=1}^{m} z_i^T W^T x_i + const\alpha - tr\left(W^T \left(\sum_{i=1}^{m} x_i x_i^T \right) W \right) \quad (2)$$

根据最近重构式，上式应被最小化，考虑到w_j式标准正交基，$\sum_i x_i$是协方差矩阵，有

$$min_W - tr(W^T X X^T W) \quad s.t. \quad W^T W = I \quad (3)$$

这就是主成成分分析的优化目标。

2.3.2 算法实现

在 sklearn 中，求解 PCA 的方法正是采用奇异值分解法，它大大提高了计算效率，使用 PCA 默认使用 svd_solver 对数据进行降维。

对 n*m 矩阵 X 进行奇异值分解：

$$X = U \sum V^T \quad (4)$$

对矩阵 X 进行阶段奇异值分解，保留前 d 个奇异值、起义向量，V 的每一列对应一个主成分，得到 d*m 样本主成成分矩阵 Y。

$$Y = V^T X \quad (5)$$

代码主要部分如下：

```
# PCA 函数：主成成分特征提取
# input          图像的二维矩阵（images）
# output         pca 处理后的特征数据,pca
def algorithm_PCA(images):
        #  创建 pca 对象
        pca = PCA(n_components=150,whiten=True)
        pca.fit(images)
        features = pca.transform(images)
        return features, pca
```

2.3.3 参数确定

n_components=150，whiten=True。将高维数据集转换为 150 维，可以显著减少数据的存储空间和计算资源需求，PCA 通过保留最重要的 150 个主成分，可以提取出图像中最优信息量的特征，方便后续分类器的训练。白化的数据可以提高机器学习算法的性能，尤其是算法对特征尺度敏感时，如支持向量机 SVM 和线性判别分析 LDA，后续的分类器用到了 SVM。

2.4 支持向量机（SVM）

2.4.1 算法原理

支持向量机（SVM）是一种经典的监督学习算法，用于解决二分类多分类问题，其核心思想是通过在特征空间中找到一个最优的超平面来进行分类，并且间隔最大。

SVM 通过优化一个凸二次规划问题来求解最佳的超平面，其中包括最小化模型的复杂度（即最小化权重的平方和），同时限制训练样本的误分类情况。这个优化问题可以使用拉格朗日乘子法来求解。对于非线性可分的情况，SVM 可以通过核函数（Kernel Function）将输入特征映射到高维空间，使得原本线性不可分的数据在高维空间中变得线性可分。常用的核函数包括线性核、多项式核、高斯核等。

2.4.2 算法实现

实验函数是一个用于训练多标签分类模型的算法，它通过 OneVsRest 策略将多标签问题分解为多个二分类问题，并使用支持向量机（SVM）作为基础分类器。该函数接受特征数据和标签数据作为输入，然后对每个属性的标签进行二值化处理，以便 SVM 可以处理。接着，它为每个属性创建一个包含线性核 SVM 的 OneVsRestClassifier，并在二值化后的标签上进行训练。最终，函数返回一个包含所有属性的标签二值化器和对应 SVM 分类器的列表。

这种方法灵活且可扩展，适用于需要处理多个标签的场景，如本次实验中，

人脸特征具有多个特征，每个特征有多个属性。但由于是对每个特征分别做训练，对时间和内存要求较高。

代码主要部分如下：

```
# 训练 SVM 模型
# svm 只能处理二进制的数据，需要先二值化
def train_svm(features, labels):
    classifiers = []
    for i in range(labels.shape[1]):    # 遍历每个属性
        lb = LabelBinarizer()
        y_bin = lb.fit_transform(labels[:, i])    # 对每个属性进行二值化
        clf = OneVsRestClassifier(SVC(kernel='linear', probability=True))
        clf.fit(features, y_bin)    # 训练每个属性的分类器
        classifiers.append((lb, clf))    # 保存标签二值化器和分类器
    return classifiers
```

2.5 实验结果与分析

完成 SVM 模型训练后，使用 Precision、Recall、F1 Score 和 Hamming Loss 分别对四个模型进行评估，结果如下：

表 2.5.2 实验结果

	Precision	Recall	F1 Score	Hamming Loss
model 0	0.787	0.782	0.785	0.168
model 1	0.452	0.237	0.311	0.077
model 2	0.974	0.987	0.981	0.023
model 3	0.777	0.875	0.823	0.132

根据评估结果，我们可以看到模型在不同维度上的表现存在显著差异。维度 0 的性能较好，具有较高的精确度、召回率和 F1 分数，以及相对较低的汉明损失，表明模型在这一维度上能够有效地区分正负样本。维度 1 的性能则不尽如人意，精确度和召回率都较低，F1 分数也不理想，汉明损失相对较低，暗示模型在这一维度上存在大量的漏报和误报。维度 2 的表现最为出色，无论是精确度、召回率还是 F1 分数都非常高，汉明损失也非常低，显示出模型在这一维度上具有极高的分类准确性。维度 3 的性能介于维度 0 和维度 1 之间，虽然精确度和召回率较高，但汉明损失相对较高，说明模型在这一维度上仍有改进空间。

四个维度中，对年龄的分类器效果最差。对年龄的分类可能需要更精细的特

征数据来提高性能。例如，可以采用同一个人不同年龄段的人脸数据加入模型训练。

代码详细部分见附录。

3 L1 正则化 + Lasso 回归 + 深度神经网络 + K 折交叉验证

3.1 设计框架

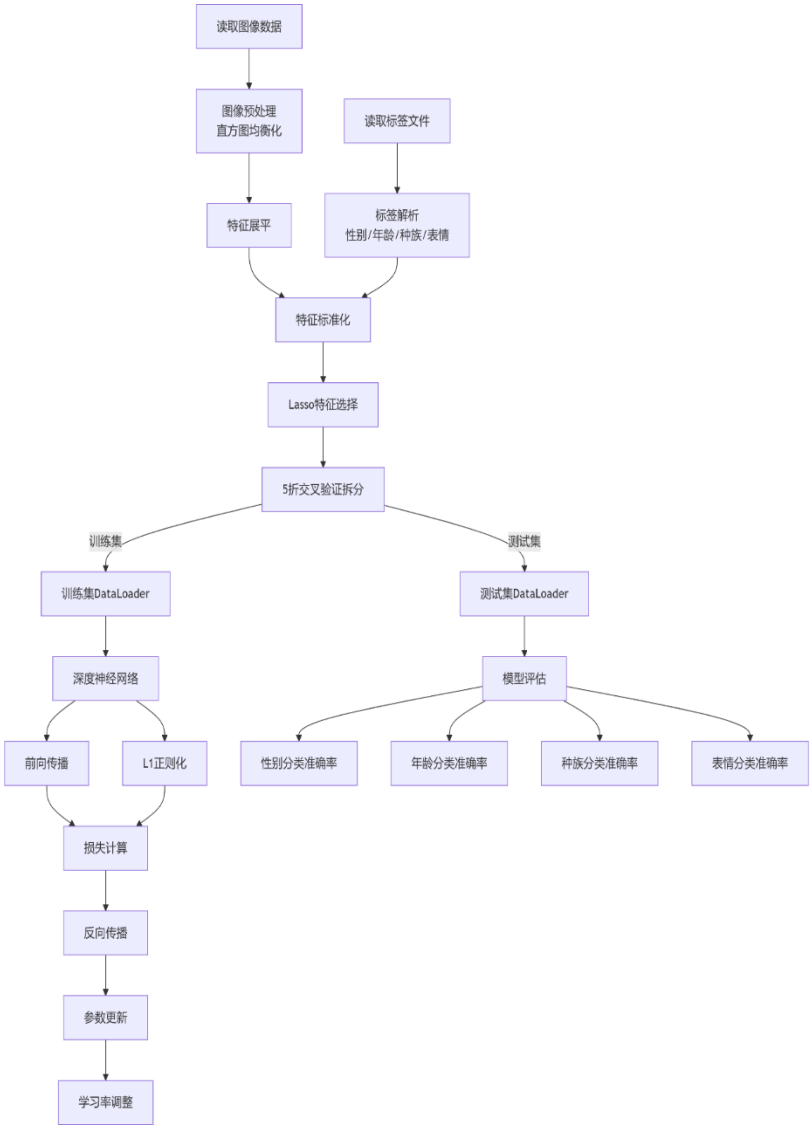


图 3.1.1 设计框架

3.2 L1 正则化与 Lasso 回归

3.2.1 L1 正则化原理

首先，让我们回顾标准线性回归问题。在线性回归中，我们试图找到一组参数 w，使得预测值与实际值之间的平方误差最小。其数学表达式为：

$$minimize J(w) = ||y - Xw||^2 = \sum(y_i - \sum w_j \cdot x_{\{ij\}})^2 \quad (6)$$

这就引出了正则化的需求。L1 正则化通过在上述目标函数中添加一个惩罚项来改变优化问题的性质：

$$J\_L1(w) = ||y - Xw||^2 + \lambda||w||_1 = \sum(y\_i - \sum w\_j \cdot x\_\{ij\})^2 + \lambda\sum|w\_j| \quad (7)$$

这里的 λ 是正则化参数，控制着正则化的强度。L1 正则化之所以能够产生稀疏解（很多参数变为零），可以从几何和数学的角度来理解：

1. 几何解释： 在二维参数空间中，L1 正则化项定义了一个菱形约束区域。最小二乘项定义的等值线（椭圆）与这个菱形的交点很可能发生在坐标轴上，这意味着有一个参数会变为零。这种几何特性在高维空间中同样存在。
2. 数学解释：

考虑简化的一维情况，目标函数可以写作：

$$J(w) = (w - \mu)^2 + \lambda|w| \quad (8)$$

对这个函数求导（在 w≠0 处）：

$$\partial J/\partial w = 2(w - \mu) + \lambda \cdot sign(w) \quad (9)$$

在最优解处，导数应该为零：

$$2(w - \mu) + \lambda \cdot sign(w) = 0 \quad (10)$$

这个方程的解具有以下性质：

- 当|μ| ≤ λ/2 时， w = 0
- 当 μ > λ/2 时， w = μ - λ/2
- 当 μ < -λ/2 时， w = μ + λ/2

这就解释了为什么 L1 正则化会产生稀疏解：当原始解（μ）不够大时，正则化会将参数压缩到零。

3.2.2 Lasso 回归原理

LASSO 回归正是将这种 L1 正则化应用到线性回归中。它的特殊之处在于它同时具备了回归和特征选择的双重功能。从优化理论的角度来看，LASSO 问题可以重写为带约束的形式：

$$minimize\ ||y - Xw||^2\ subject\ to \sum|w\_j| \le t \quad (11)$$

LASSO 解的独特性质可以通过次梯度（subgradient）分析来理解。在 w_j = 0 处，L1 范数的次梯度是[-1,1]区间内的任意值。这意味着在最优解处，对于零系数，满足:

$$-2X_j^T(y - Xw) \in [-\lambda, \lambda] \quad (12)$$

这个条件说明了为什么某些系数会精确地等于零，而不是接近零的小值。这种数学性质使得 LASSO 特别适合处理高维数据，因为它能自动识别最相关的特征，同时将不重要的特征的系数置为零。这不仅提高了模型的可解释性，还降低了过拟合的风险。

3.2.3 L1 正则化与 Lasso 回归在人脸识别中的应用

让我从模式识别和人脸识别的角度，来解释 L1 正则化和 LASSO 回归的实际应用价值。

在人脸识别这个具体场景中，我们面临着一个基本的挑战：图像数据维度极高。想象一张 128×128 像素的灰度图像，它就有 16384 个像素点，每个像素点都可以被视为一个特征。然而，并非所有像素对于识别人脸特定属性（如性别、

年龄、表情等）都同等重要。这就是 L1 正则化和 LASSO 回归发挥作用的地方。让我们以性别识别这个任务为例来详细说明：

当我们获取一张人脸图像时，某些区域（如眉毛、下巴轮廓、发际线等）对于判断性别可能比其他区域（如耳朵、脸颊等）更重要。LASSO 回归通过其特征选择能力，能够自动识别出这些关键区域。具体来说，它会：

1. 评估每个像素点对性别判断的贡献度。
2. 将不重要区域的权重降为零，保留重要区域的非零权重。
3. 最终可能会从 16384 个像素中筛选出 2000 个最具辨识度的像素点。

这个过程就像是一个经验丰富的人类专家在告诉我们："要判断一个人的性别，你主要要看这些特定的面部区域"。

再比如在年龄识别任务中，LASSO 可能会：

- 重点关注眼角和嘴角的皱纹区域
- 保留额头纹路的特征
- 忽略掉与年龄关系不大的区域，如耳朵形状

L1 正则化在深度神经网络中的应用更为微妙。它不仅能帮助选择重要特征，还能：

1. 在网络训练过程中自动调整神经元之间的连接强度
2. 将不重要的连接权重降为零，形成一个更稀疏的网络结构
3. 这种稀疏结构不仅减少了计算量，还提高了模型的鲁棒性

这就像是在训练一个专业的人脸识别专家，我们不是告诉他"看所有地方"，而是引导他"关注重要特征，忽略无关信息"。这种方法的优势在于：

1. 提高计算效率：不需要处理所有 16384 个像素，只需关注 2000 个关键像素
2. 增强泛化能力：减少对无关特征的依赖，使模型在新数据上表现更好
3. 提升可解释性：我们可以通过查看非零权重的分布，理解模型在关注脸部的哪些区域

这种特征选择和权重稀疏化的过程，实际上模拟了人类视觉认知的某些特点：我们在识别人脸时也是优先关注一些关键特征，而不是平等地处理所有视觉信息。

3.2.4 算法实现

```
        #L1 正则化
    def l1_regularization(self):
        l1_loss = 0
        for param in self.parameters():
            l1_loss += torch.sum(torch.abs(param))
        return self.l1_lambda * l1_loss
```

```
# Lasso 特征选择
def select_features(X, y, n_features=2000):
    lasso = Lasso(alpha=0.01, max_iter=10000)
    lasso.fit(X, y)
    importance = np.abs(lasso.coef_)
    selected_indices = np.argsort(importance)[-n_features:]
```

```
    return selected_indices
```

3.3 深度神经网络

首先，让我们从神经网络的基本结构开始。神经网络是由多层神经元组成的，每一层的数学表达式为：

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)} \tag{13}$$

$$a^{(l)} = f\left(z^{(l)}\right) \tag{14}$$

其中，W^(l)是权重矩阵，b^(l)是偏置向量，f 是激活函数，a^(l)是该层的输出。

3.3.1 前向传播算法与 ReLU 激活函数

算法原理：

前向传播是神经网络中信息流动的基本过程。每个神经元接收输入，进行加权求和，然后通过激活函数产生输出。

1. 单个神经元的数学模型：

$$z = w^1x^1 + w^2x^2 + \cdots + w_{\mathrm{n}}x_{\mathrm{n}} + ba = f(z) \tag{15}$$

这里 z 是加权和，f 是激活函数，a 是神经元输出。

2. ReLU（Rectified Linear Unit）激活函数的数学定义：

$$f(x) = max(0, x) \tag{16}$$

其导数为：

$$f'(x) = \{1, if\ x > 0\ \{0, if\ x < 0\ \{undefined, if\ x = 0 \tag{17}$$

3. 对于一个 L 层的神经网络，前向传播的完整过程是：
   - 输入层：$a^{(0)} = x$
   - 隐藏层和输出层(l=1 到 L)：

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}a^{(l)} = ReLU\left(z^{(l)}\right) \tag{18}$$

ReLU 函数的数学特性使其在深度学习中表现出色：

- 解决了传统 sigmoid 函数的梯度消失问题
- 计算简单，只需判断是否大于零
- 具有稀疏激活性质，有助于网络学习稀疏表示

在人脸识别中的应用：

实际应用： 在人脸识别系统中，前向传播和 ReLU 的组合发挥着关键作用。以面部特征提取为例：

1. 第一层神经元：
   - 接收原始像素值作为输入
   - 通过加权求和检测基本特征，如边缘和纹理
   - ReLU 将负值置零，保留有用的特征响应
2. 中间层：
   - 组合低层特征形成更复杂的模式
   - 例如，多个边缘检测器的输出可能组合成"眼睛"或"鼻子"的检测器

- ReLU 帮助网络聚焦于最显著的特征组合

3. 高层：
   - 整合全局信息形成抽象概念
   - 比如"年轻面孔"或"微笑表情"等高级特征
   - ReLU 的稀疏激活确保每个神经元只对特定模式响应

在实际的人脸属性识别中，这种层次化处理表现为：

- 底层可能对应简单的明暗变化和轮廓
- 中层可能识别出眼睛、鼻子等局部特征
- 高层则能够捕捉性别、年龄等抽象属性

ReLU 的非线性特性尤其重要：

- 允许网络学习复杂的非线性决策边界
- 帮助网络适应不同光照条件下的人脸识别
- 提供了计算效率和表示能力的良好平衡

3.3.2 反向传播算法

算法原理：

反向传播算法的核心思想是利用链式法则高效计算损失函数对网络参数的梯度。让我们一步步理解这个过程：

1. 首先定义前向传播过程。对于神经网络中的第 l 层：

$$z^{(l)} = W^{(l)}a^{(l-1)} + b^{(l)}a^{(l)} = f\left(z^{(l)}\right) \tag{19}$$

   其中 W^(l)是权重矩阵，b^(l)是偏置向量，f 是激活函数。

2. 定义损失函数 L，例如均方误差：

$$L = (1/2)\left|\left|y - a^{(L)}\right|\right|^2 \tag{20}$$

   其中 y 是目标输出，a^(L)是网络最后一层的输出。

3. 要更新网络参数，我们需要计算损失函数对每层参数的偏导数。这里引入一个关键概念：误差项

$$\delta^{(l)}\delta_j^{(l)} = \partial L/\partial z_j^{(l)} \tag{21}$$

   表示损失函数对第 l 层第 j 个神经元加权输入的偏导数。

4. 使用链式法则，我们可以得到误差的反向传播方程：

$$\delta^{(l)} = \left(\left(W^{(l+1)}\right)^T\delta^{(l+1)}\right) \odot f'\left(z^{(l)}\right) \tag{22}$$

   其中⊙表示逐元素乘法。

5. 最后，我们可以计算参数的梯度：

$$\partial L/\partial W^{(l)} = \delta^{(l)}\left(a^{(l-1)}\right)^T \partial L/\partial b^{(l)} = \delta^{(l)} \tag{23}$$

在人脸识别中的应用：

在人脸识别任务中，反向传播算法发挥着关键作用。以年龄识别为例：

1. 当网络预测某人年龄出现误差时，比如将一个 30 岁的人预测为 20 岁：
   - 损失函数会计算出预测值与真实值之间的差距
   - 反向传播算法会追踪这个误差，找出导致误判的原因
2. 通过误差反向传播，网络会学习到：

- 哪些面部特征对年龄判断最重要（如眼角纹路、额头皱纹）
- 哪些特征权重需要调整（比如可能过分依赖肤色而忽视了面部结构）

3. 在实际训练过程中：
   - 早期阶段，网络可能主要关注明显的特征，如白发
   - 随着训练进行，反向传播帮助网络逐渐学习到更细微的年龄特征
   - 最终形成一个层次化的特征提取系统，能够综合考虑多个年龄相关的视觉线索

这个算法的巧妙之处在于：

- 它能自动发现对任务最重要的特征，无需人工指定具体的特征点
- 通过多层反向传播，网络可以学习到从简单到复杂的特征层次
- 它提供了一种高效的方式来调整网络中数百万个参数

3.3.3 Adam(Adaptive Moment Estimation)优化算法

算法原理：

Adam 算法结合了动量法和自适应学习率的优点，通过计算梯度的一阶矩估计（均值）和二阶矩估计（未中心化的方差）来自适应地调整每个参数的学习率。

1. 参数更新的基本框架： 对于参数 θ，在时间步 t，更新规则为：

$$\theta_t = \theta_t^{-1} - \alpha \cdot \hat{m}_t / \left(\sqrt{\hat{v}_t} + \varepsilon\right) \tag{24}$$

2. 一阶矩估计（动量）：

$$m_t = \beta^1 \cdot m_t^{-1} + (1 - \beta^1) \cdot g_t \tag{25}$$

其中：

- g_t是当前梯度
- β_1是动量衰减率（通常取 0.9）
- m_t追踪梯度的指数移动平均

3. 二阶矩估计：

$$v_t = \beta^2 \cdot v_t^{-1} + (1 - \beta^2) \cdot g_t^2 \tag{26}$$

其中：

- β_2是二阶矩衰减率（通常取 0.999）
- v_t追踪梯度平方的指数移动平均

4. 偏差修正： 由于 m_1和 v_1初始化为 0，需要进行偏差修正：

$$\hat{m}_t = m_t/(1 - \beta^{1t}) \hat{v}_t = v_t/(1 - \beta^{2t}) \tag{27}$$

这个算法的数学设计体现了几个关键思想：

- 动量项帮助克服鞍点和局部最小值
- 自适应学习率允许不同参数有不同的更新步长
- 偏差修正确保早期估计的准确性

在人脸识别中的应用

在人脸识别系统的训练中，Adam 算法的这些特性带来了显著优势。以表情识别任务为例：

1. 早期训练阶段：

- 网络需要快速学习明显的表情特征（如大笑 vs 皱眉）
- 动量项帮助网络快速找到大致正确的参数方向
- 自适应学习率确保不同层的参数能够合适地更新

2. 中期训练阶段：
   - 网络开始学习更细微的表情差别
   - 一阶矩估计帮助平滑参数更新
   - 二阶矩估计自动降低频繁变化参数的学习率
3. 后期微调阶段：
   - 网络需要对微妙的表情变化进行精确判断
   - 自适应学习率机制使得微小但重要的参数调整得以实现
   - 动量帮助克服训练后期的参数振荡

在实际训练过程中，Adam 的优势表现为：

- 不同面部特征检测器能够以适合的速度学习
  - 眼睛区域的参数可能需要较大更新
  - 而细微的嘴角变化可能需要较小更新
- 自动处理不同尺度的梯度
  - 浅层的边缘检测器和深层的抽象特征提取器都能得到合适的更新
- 较少需要手动调整学习率
  - 算法能自适应地为不同参数选择合适的更新步长

3.3.4 Dropout 正则化算法

算法原理：

Dropout 是一种通过在训练过程中随机"关闭"部分神经元来防止过拟合的正则化技术。其数学描述如下：

1. 训练时的数学模型： 对于第 l 层的输出 a^(l)，引入随机变量 r^(l)：

$$r^{(l)} \sim Bernoulli(p) \tilde{a}^{(l)} = r^{(l)} \odot a^{(l)} \tag{28}$$

   其中：
   - p 是保留神经元的概率（通常取 0.5~0.7）
   - r^(l)是一个与 a^(l)同形状的掩码
   - ⊙表示逐元素乘法
   - ã^(l)是实际传递给下一层的输出
2. 测试时的数学期望调整： 为了保持输出期望值不变，测试时的输出需要缩放：

$$a_t^{(l)} est = p \cdot a^{(l)} \tag{29}$$

3. 每个训练小批量都会生成新的随机掩码：

$$P\left(r_i^{(l)} = 1\right) = pP\left(r_i^{(l)} = 0\right) = 1 - p \tag{30}$$

这种随机失活的数学原理可以从以下几个角度理解：

- 防止特征共适应：通过随机断开神经元连接
- 实现模型集成：每次 dropout 相当于训练不同的子网络
- 引入噪声：增强模型对输入扰动的鲁棒性

在人脸识别中的应用

在人脸识别任务中，Dropout 的应用体现了其独特价值。以性别识别为例：

1. 特征学习阶段：
    - 网络可能过度依赖某些显著特征（如长发）
    - Dropout 强制网络学习多样化的特征组合
    - 例如：
        - 当头发特征被随机关闭时，网络必须学会利用脸型
        - 当脸型特征被关闭时，网络需要利用其他特征如眉骨
2. 鲁棒性提升：
    - 面对遮挡或部分特征缺失时仍能正确识别
    - 比如：
        - 即使部分面部被口罩遮挡
        - 或者在不同角度下观察人脸
        - 网络依然能够进行可靠判断
3. 实际训练效果：
    - 提高模型泛化能力
    - 减少过度拟合训练数据的风险
    - 例如：
        - 避免仅依赖训练集中的特定模式
        - 提高在新数据上的表现
        - 增强对不同环境条件的适应性

在实际部署中的优势：

- 提高系统可靠性：不过分依赖单一特征
- 增强适应性：能处理各种现实场景
- 改善泛化：在新用户数据上表现更好

3.3.5 学习率调度算法

算法原理：

学习率调度是一种动态调整学习率的策略，在本例中使用的是 ReduceLROnPlateau（在验证指标停止改善时降低学习率）。其数学描述如下：

1. 基本调度规则： 当验证损失在连续 patience 次迭代中没有改善时：

$$\alpha_new = \alpha_old \times factor \tag{31}$$

其中：

- α 是学习率
- factor 是学习率衰减因子（通常取 0.1 或 0.5）
- patience 是等待改善的迭代次数

2. 改善判定标准： 定义最小改善阈值 ε，对于当前验证损失 L_t： 如果

$$L_t > min\left(L_{t-patience:t-1}\right) - \varepsilon \tag{32}$$

则触发学习率调整

3. 学习率更新的数学约束：
    - 上界约束：α_new ≤ α_initial
    - 下界约束：α_new ≥ α_min 这确保学习率保持在合理范围内
4. 早停机制： 如果 α_new < α_min 则可以选择停止训练

在人脸识别的应用：

在人脸识别系统的训练过程中，学习率调度的作用体现在不同训练阶段：

1. 初始训练阶段：
    - 使用较大的学习率快速接近最优解
    - 例如在学习基本的面部特征时：
        - 快速学习边缘检测
        - 快速建立面部区域的初步映射
2. 中期训练阶段：
    - 当基本特征已经学习到位，但精度仍在提升时：
        - 学习率可能第一次降低
        - 帮助网络更细致地调整参数
        - 例如：更精确地定位眼角、嘴角等关键点
3. 精调阶段：
    - 当网络性能接近饱和时：
        - 使用更小的学习率
        - 对参数进行微调
        - 例如：优化不同光照条件下的识别准确率
        - 提升对细微表情变化的敏感度

实际训练中的优势：

1. 自适应学习过程：
    - 早期快速收敛
    - 后期精细调整
    - 避免错过最优解
2. 处理学习停滞：
    - 在性能平台期自动调整学习策略
    - 帮助逃离局部最优
    - 提高最终模型性能
3. 提高训练效率：
    - 减少人工调参需求
    - 自动寻找合适的学习节奏
    - 在不同训练阶段保持适当的更新步长

这种动态调整策略的效果体现在：

- 训练更稳定：避免后期震荡
- 收敛更可靠：降低过拟合风险
- 性能更优：达到更好的最终效果

3.3.5 学习率调度算法

深度神经网络主体代码：

```
        # 定义深度神经网络
class FaceClassifier(nn.Module):
    def __init__(self, input_size, num_classes):
        super(FaceClassifier, self).__init__()
```

```
        self.layers = nn.Sequential(
            nn.Linear(input_size, 1024),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, num_classes)
        )
        self.l1_lambda = 0.01

    def forward(self, x):
        return self.layers(x)

    #L1 正则化
    def l1_regularization(self):
        l1_loss = 0
        for param in self.parameters():
            l1_loss += torch.sum(torch.abs(param))
        return self.l1_lambda * l1_loss


# Lasso 特征选择
def select_features(X, y, n_features=2000):
    lasso = Lasso(alpha=0.01, max_iter=10000)
    lasso.fit(X, y)
    importance = np.abs(lasso.coef_)
    selected_indices = np.argsort(importance)[-n_features:]
    return selected_indices


# 训练单个模型
def train_single_model(X_train, y_train, X_test, y_test, num_classes,
device):
    # 创建数据加载器
    train_dataset = FaceDataset(X_train, y_train)
    test_dataset = FaceDataset(X_test, y_test)
    train_loader = DataLoader(train_dataset, batch_size=64,
shuffle=True)
    test_loader = DataLoader(test_dataset, batch_size=64)

    # 初始化模型
```

```
    model = FaceClassifier(X_train.shape[1], num_classes).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=0.001,
weight_decay=0.0001)
    scheduler = optim.lr_scheduler.ReduceLROnPlateau(optimizer,
mode='min', factor=0.5, patience=5)

    # 训练
    best_acc = 0
    for epoch in range(50):
        model.train()
        total_loss = 0
        for batch_X, batch_y in train_loader:
            batch_X, batch_y = batch_X.to(device), batch_y.to(device)
            optimizer.zero_grad()
            outputs = model(batch_X)
            loss = criterion(outputs, batch_y) +
model.l1_regularization()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        # 验证
        model.eval()
        correct = 0
        total = 0
        with torch.no_grad():
            for batch_X, batch_y in test_loader:
                batch_X, batch_y = batch_X.to(device),
batch_y.to(device)
                outputs = model(batch_X)
                _, predicted = torch.max(outputs.data, 1)
                total += batch_y.size(0)
                correct += (predicted == batch_y).sum().item()

        accuracy = correct / total
        if accuracy > best_acc:
            best_acc = accuracy

        scheduler.step(total_loss)

    return best_acc
```

3.4 代码与结果

```
开始训练 性别 分类器...
正在训练第 1 折...
第 1 折准确率: 0.6095
正在训练第 2 折...
第 2 折准确率: 0.5940
正在训练第 3 折...
第 3 折准确率: 0.5852
正在训练第 4 折...
第 4 折准确率: 0.6216
正在训练第 5 折...
第 5 折准确率: 0.6278
性别 分类器的平均准确率: 0.6076 ± 0.0161
```

图 3.4.1 性别结果

```
开始训练 年龄 分类器...
正在训练第 1 折...
第 1 折准确率: 0.8026
正在训练第 2 折...
第 2 折准确率: 0.8444
正在训练第 3 折...
第 3 折准确率: 0.8495
正在训练第 4 折...
第 4 折准确率: 0.8089
正在训练第 5 折...
第 5 折准确率: 0.8364
年龄 分类器的平均准确率: 0.8284 ± 0.0190
```

图 3.4.2 年龄结果

```
开始训练 种族 分类器...
正在训练第 1 折...
第 1 折准确率: 0.8942
正在训练第 2 折...
第 2 折准确率: 0.8942
正在训练第 3 折...
第 3 折准确率: 0.8841
正在训练第 4 折...
第 4 折准确率: 0.8892
正在训练第 5 折...
第 5 折准确率: 0.8929
种族 分类器的平均准确率: 0.8909 ± 0.0039
```

图 3.4.4 种族结果

```
开始训练 表情 分类器...
正在训练第 1 折...
第 1 折准确率: 0.4894
正在训练第 2 折...
第 2 折准确率: 0.5013
正在训练第 3 折...
第 3 折准确率: 0.4837
正在训练第 4 折...
第 4 折准确率: 0.5276
正在训练第 5 折...
第 5 折准确率: 0.5201
表情 分类器的平均准确率: 0.5044 ± 0.0170
```

图 3.4.5 表情结果

结果分析：

有分类器准确率上看，该人脸识别系统在种族和年龄上有较高的正确率，而对表情和年龄的识别的准确率较低。

为了能够尽可能准确地找到准确率不高的原因并且排除可能训练次数不足的情况出现，又对代码做出了以下优化：

1.在全连接层面添加批量归一化层，加快模型收敛；

2.增加神经元深度；

3.讲原来的激活函数 ReLU 换为 LeakyReLU，避免在训练深度网络的过程中出现梯度消失的情况；

4.讲训练轮次从 50 提升到 500

5.K 折交叉验证从 5 折提高到 10 折。

以下对代码做出上述优化后的结果：

```
开始训练 性别 分类器...
正在训练第 1 折...
第 1 折准确率: 0.6075
正在训练第 2 折...
第 2 折准确率: 0.6165
正在训练第 3 折...
第 3 折准确率: 0.6216
正在训练第 4 折...
第 4 折准确率: 0.5789
正在训练第 5 折...
第 5 折准确率: 0.6391
正在训练第 6 折...
第 6 折准确率: 0.5915
正在训练第 7 折...
第 7 折准确率: 0.6316
正在训练第 8 折...
第 8 折准确率: 0.6115
正在训练第 9 折...
第 9 折准确率: 0.6516
正在训练第 10 折...
第 10 折准确率: 0.6316
性别 分类器的平均准确率: 0.6181 ± 0.0208
```

图 3.4.6 性别结果（优化后）

```
开始训练 年龄 分类器...
正在训练第 1 折...
第 1 折准确率: 0.7833
正在训练第 2 折...
第 2 折准确率: 0.8198
正在训练第 3 折...
第 3 折准确率: 0.8272
正在训练第 4 折...
第 4 折准确率: 0.8639
正在训练第 5 折...
第 5 折准确率: 0.8717
正在训练第 6 折...
第 6 折准确率: 0.8272
正在训练第 7 折...
第 7 折准确率: 0.8115
正在训练第 8 折...
第 8 折准确率: 0.8063
正在训练第 9 折...
第 9 折准确率: 0.8272
正在训练第 10 折...
第 10 折准确率: 0.8455
年龄 分类器的平均准确率: 0.8284 ± 0.0251
```

图 3.4.7 年龄结果（优化后）

```
开始训练 种族 分类器...
正在训练第 1 折...
第 1 折准确率: 0.8892
正在训练第 2 折...
第 2 折准确率: 0.8992
正在训练第 3 折...
第 3 折准确率: 0.8992
正在训练第 4 折...
第 4 折准确率: 0.8892
正在训练第 5 折...
第 5 折准确率: 0.8992
正在训练第 6 折...
第 6 折准确率: 0.8690
正在训练第 7 折...
第 7 折准确率: 0.8690
正在训练第 8 折...
第 8 折准确率: 0.9093
正在训练第 9 折...
第 9 折准确率: 0.9018
正在训练第 10 折...
第 10 折准确率: 0.8841
种族 分类器的平均准确率: 0.8909 ± 0.0129
```

图 3.4.8 种族结果（优化后）

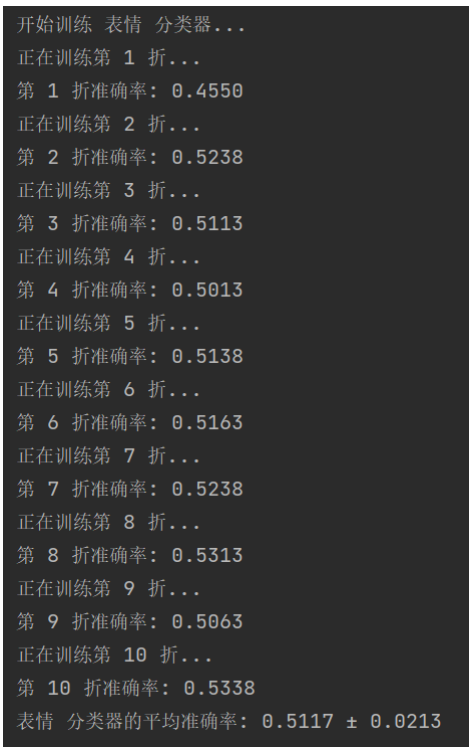


图 3.4.9 表情结果（优化后）

分析：

从对代码优化后的结果来看，训练出来的模型在交叉验证中的平均准确率都有小幅度的提高，但在表情和性别的训练效果还是不理想，接着继续查阅相关资料并进行排查，有以下猜测：

1. 用 Lasso 回归进行特征提取时只选择了前 2000 个特征这种方法假设了像素的重要性是独立的，但实际上面部特征往往是由多个像素群组成的。比如说，判断一个人是否在微笑，需要看嘴角、眼角等多个区域的像素共同作用，而不是独立的像素点。

2. 然后当前神经网络架构还是一个全连接的神经网络，这样的结构不太适合用来处理图像数据，虽然有多个全连接层学习特征，但是难以有效捕捉到图像的局部特征和空间关系，用一句话来说就是相当于“盲人摸象”，增加了学习的难度。

因此在此基础上再对代码进行了一次优化：

1. 在 Lasso 回归进行特征提取时的特征数量增加到了 5000 个；

2. 降低了初始学习率，增加了权重衰减，同时增加耐心值，给模型更多的时间找到最优解。

以下是本次优化后的结果：

```
开始训练 性别 分类器...
正在训练第 1 折...
第 1 折准确率: 0.6750
正在训练第 2 折...
第 2 折准确率: 0.6466
正在训练第 3 折...
第 3 折准确率: 0.6316
正在训练第 4 折...
第 4 折准确率: 0.6140
正在训练第 5 折...
第 5 折准确率: 0.6291
正在训练第 6 折...
第 6 折准确率: 0.6441
正在训练第 7 折...
第 7 折准确率: 0.6316
正在训练第 8 折...
第 8 折准确率: 0.6341
正在训练第 9 折...
第 9 折准确率: 0.6767
正在训练第 10 折...
第 10 折准确率: 0.6065
性别 分类器的平均准确率: 0.6389 ± 0.0217
```

图 3.4.10 表情结果（再优化）

```
开始训练 年龄 分类器...
正在训练第 1 折...
第 1 折准确率: 0.7885
正在训练第 2 折...
第 2 折准确率: 0.8277
正在训练第 3 折...
第 3 折准确率: 0.8272
正在训练第 4 折...
第 4 折准确率: 0.8639
正在训练第 5 折...
第 5 折准确率: 0.8717
正在训练第 6 折...
第 6 折准确率: 0.8272
正在训练第 7 折...
第 7 折准确率: 0.8115
正在训练第 8 折...
第 8 折准确率: 0.8168
正在训练第 9 折...
第 9 折准确率: 0.8403
正在训练第 10 折...
第 10 折准确率: 0.8482
年龄 分类器的平均准确率: 0.8323 ± 0.0235
```

图 3.4.10 年龄结果（再优化）

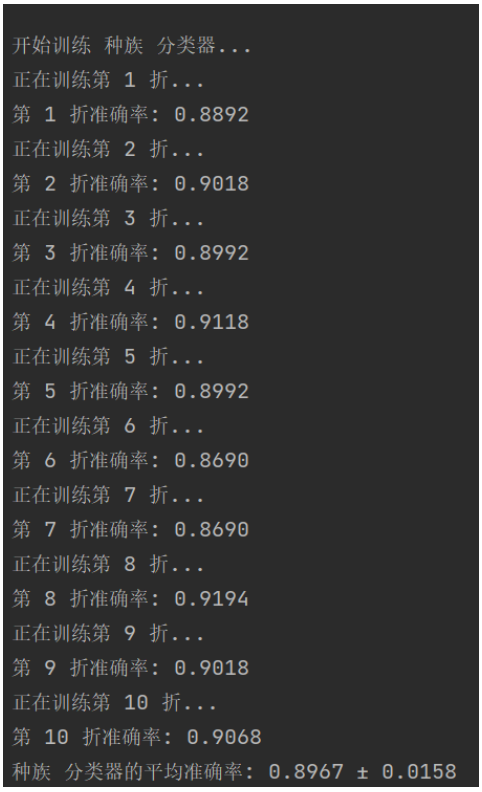


图 3.4.11 种族结果（再优化）

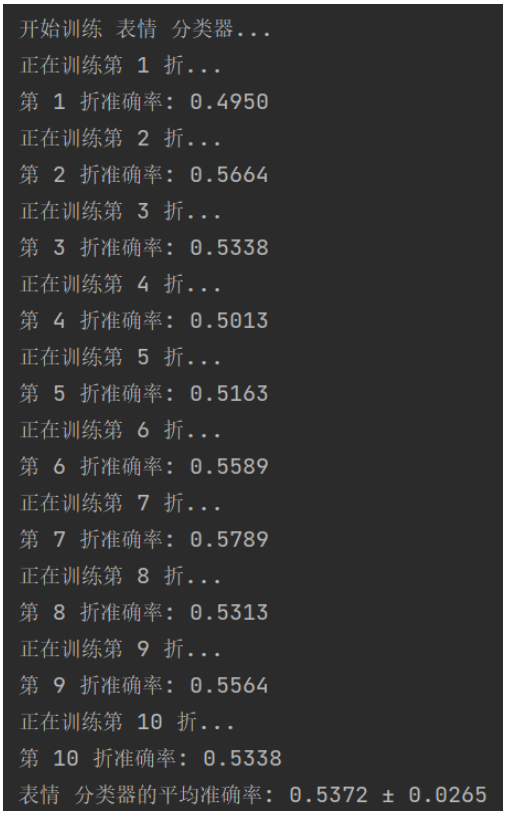


图 3.4.12 表情结果（再优化）

表 3.4.1 三次代码的结果对比

	第一次	第二次	第三次
性别	0.6076±0.0161	0.6181±0.0208	0.6389±0.0217
年龄	0.8284±0.0190	0.8284±0.0251	0.8323±0.0235
种族	0.8909±0.0039	0.8909±0.0129	0.8967±0.0158
表情	0.5044±0.0170	0.5117±0.0213	0.5372±0.0265

对三次结果数据进行分析可知，每次训练的准确率都在提高，但在表情上的准确率还是不忍直视，于是在 Google Scholar 上查阅相关论文后有以下的猜测：

1.完成该模式识别系统的设计我本人是希望更充分地体验整个设计过程，因此更加注重在算法上的堆砌，而忽略了算法与算法间，参数与参数间模型的适配性，导致算法和算法的搭配可能并不合理从而影响了训练效果；

2.在前面的数据预处理中将图像展平为一维数组，导致面部特征空间分布信息丢失，不利于特征学习。

```
img_mat = img.reshape(128, 128)  # 统一 128*128 尺寸
img_mat = cv2.equalizeHist(img_mat)  # 直方图均衡
img_flattened = img_mat.flatten()  # 展平为一维数组
```

4 基于朴素贝叶斯的 Adaboost 算法

4.1 设计框架

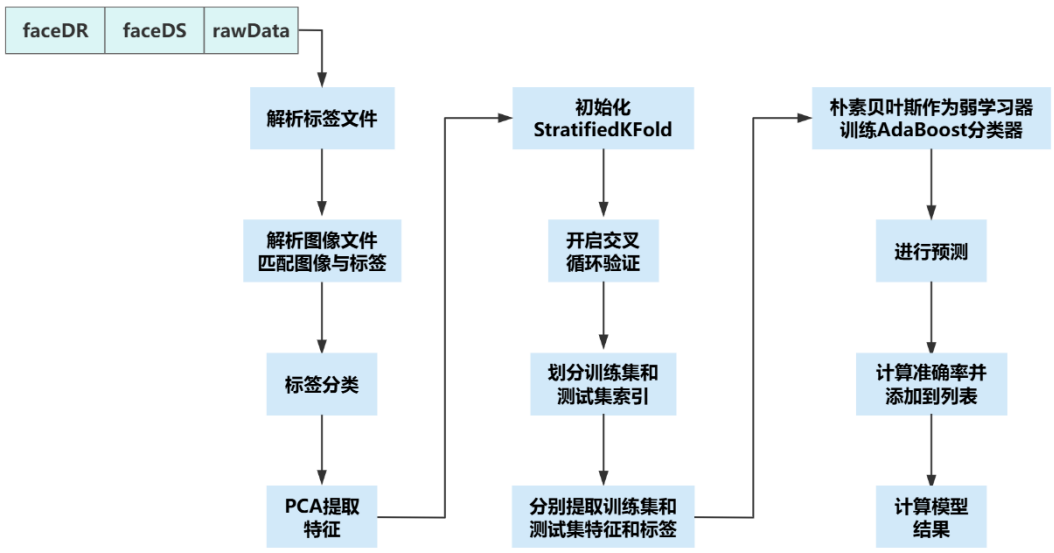


图 4.1.1 设计框架

4.2 特征提取

人脸图像包含大量的像素信息，直接处理这些高纬度的数据会导致计算复杂度高，影响运算速度。进行特征提取可以将图像的数据转换为低维的、有代表性

的特征向量，减少计算资源的消耗。人脸的特征提取可以突出脸部的重要信息，比如眼睛、鼻子、嘴巴等部位的位置和形状，忽略对识别过程不重要的信息。此外，通过提取脸部关键信息，可以减少噪声和不想管特性的干扰，从而提高识别算法的准确性和鲁棒性。

在实验过程中，我尝试了两种特征提取算法，分别是 LBP 算法和 PCA 算法，经过对比发现。通过实验结果发现，两种算法的结果差异较大，基于 PCA 算法提取特征的模型训练结果正确率高于基于 LBP 算法提取特征的结果。以下是对两种算法的分析介绍。

4.2.1 Local Binary Pattern(LBP)算法

LBP(Local Binary Pattern)即局部二值模式，它是先由 T.Ojala、D.Harwood 等人在 1994 年提出。该算法主要用于描述图像中的局部纹理模式，能够有效地捕捉图像中地纹理特征，在如人脸识别、纹理分类等许多计算机视觉和图像处理任务中有广泛应用。

LBP 提取特征地原理，首先选取一个大小为 3×3 的窗口，以该窗口的中心像素为阈值，与相邻的 8 个像素的灰度值进行比较，如果周围的像素值大于中心像素值，那么该位置会被标记为 1，反之标记为 0。接着顺时针拼接所有相邻的像素点的标记值，就可获得一个二进制数，将该二进制数转换为十进制，即 LBP 码（共 256 种），转换之后的值即作为该窗口的中心像素点的 LBP 值，以此来反应这个 3×3 区域的纹理信息。

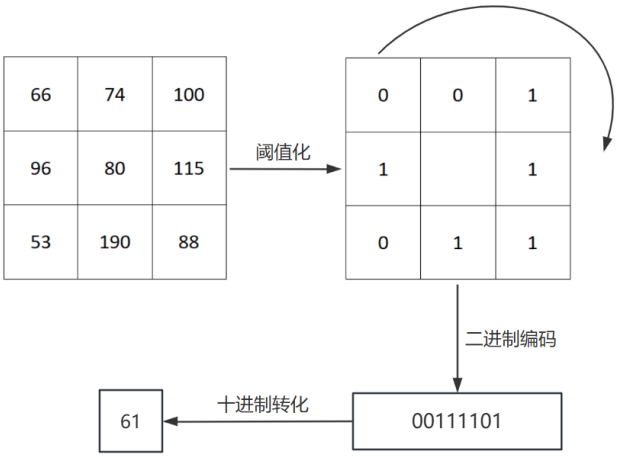


图 4.2.1 LBP 特征提取示意图

算法公式为

$$LBP(x_c, y_c) = \sum_{i=0}^{p-1} 2^p s(g_i - g_c) \tag{33}$$

其中：(x_c, y_c) 代表 LBP 中心像素点的位置；g_c 代表中心像素点的灰度值；g_i 代表相邻像素点的灰度值，$s(x)$ 为二值函数，定义如下：

$$s(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases} \tag{33}$$

传统 LBP 算法只是覆盖了很小的一个 3×3 区域范围，较难满足对于不同尺寸纹理特征的提取需求。为适应不同尺度的纹理特征，并达到灰度和旋转不变性的要求，T.Ojala 等人提出了改进 LBP 算法，即圆形局部二进制模式(Circular Local Binary Pattern,CLBP)，该算法引入圆形邻域代替正方形邻域。Circular LBP 算法允许在半径为 r 的圆形邻域内有任意 n 个像素点。

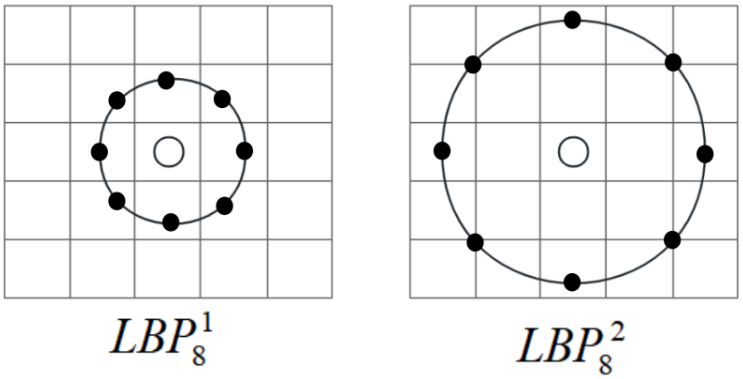


图 4.2.2 Circular LBP 特征提取示意图

算法公式为

$$LBP_{N,R}(x_c, y_c) = \sum_{n=1}^{N} 2^p s(g_n - g_c) \tag{34}$$

其中 n 表示圆形区域中共计 N 个采样点中的第 n 个，R 表示采样圆形半径，$s(x)$ 公式与传统 LBP 公式中的相同。圆形边界上的点坐标计算如下：

$$\begin{cases} x_p = x_c + R \bullet cos\left(\frac{2\pi p}{P}\right) \\ y_p = y_c - R \bullet sin\left(\frac{2\pi p}{P}\right) \end{cases} \tag{35}$$

Circular LBP 算法的代码如下：

```
def lbp(image, n, r):
    img = image.astype(np.uint8)
    lbp_image = sk.local_binary_pattern(img, n, r, method='uniform')
    hist, _ = np.histogram(lbp_image.ravel(), bins=np.arange(0, n+10), range=(0,
            n+10), density=True)
    hist = hist / hist.sum()
    return hist
```

参数处理：image 为输入图像的数据，这里为灰度图像。n 为圆形邻域的点的数量，这个参数决定了每个像素点周围有多少个点参与计算局部二值模式。r 为圆形领域的半径。在训练过程中尝试了多种（n，r）的数据搭配，比如（2，6）、（3，8）、（2，10）等，对不同的数据进行训练。

对图像处理的结果如下所示：

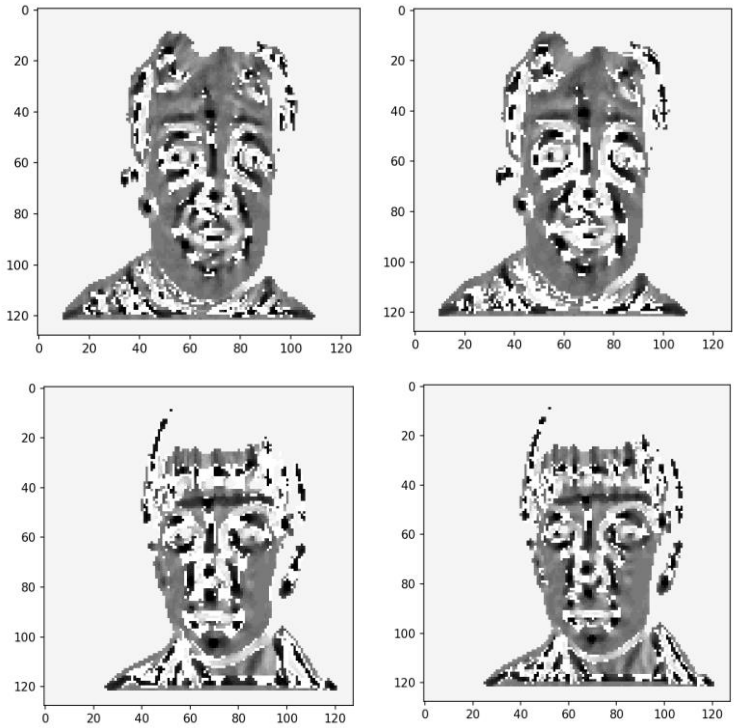


图 4.2.3 CLBP 图像处理结果图（左列为 R=3，N=3×R；右列为 R=4，N=6×R）

4.2.2 PCA 算法

主成成分分析(Principal Component Analysis,PCA)，也称为卡尔胡宁-勒夫变换(Karhunen-Loeve Transform)，是一种给用于探索高维数据结构的技术，是一种使用广泛的数据降维方法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征，也称为主成成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。方差是衡量数据分布离散程度的指标，方差越大，数据的变异性越强。PCA 的核心是通过寻找数据中方差最大的方向来提取主要特征，其试图找到一个新的坐标系，是的数据在这个坐标系中的投影（即主成分）具有最大的方差，从而保留尽可能多的信息。PCA 通过线性变换将原始数据投影到新的特征空间中，其首先计算数据的协方差矩阵，然后通过特征值分解或奇异值分解(SVD)来获取特征值和特征向量，特征值表示主成分的重要程度，而特征向量则表示主成分的方向。

数据标准化是 PCA 的关键步骤之一，通过标准化，消除不同特征之间的量纲差异，使每个特征在同一尺度上进行比较，标准化公式如下：

$$X' = \frac{X - \mu}{\sigma} \tag{36}$$

其中为均 μ 值，σ 为标准差，X' 是标准化后的数据。

标准化后下一步是计算协方差矩阵，用于了解不同特征之间的关系，协方差矩阵计算如下：

$$C = \frac{1}{n-1} X^T X' \tag{37}$$

其中 n 是样本数量，C 是一个 $n \times n$ 的协方差矩阵。

特征值和特征向量的计算是 PCA 的核心步骤，识别出数据中的主成分。对协方差矩阵进行特征分解，得到特征值 λ 和特征向量 ν。选择主成成分后，将原

始数据投影到新的特征空间中。

PCA 的算法代码如下：

```
def pca_feature(images, n_components):
    img_flatten = np.array([img.flatten() for img in images])  #展平
    # 创建 PCA 对象并设置要保留的主成分数量
    pca = PCA(n_components=n_components)
    # 对展平后的图像数据应用 PCA 进行特征提取和降维
    pca_features = pca.fit_transform(img_flatten)
    return pca_features  # 将二维的特征矩阵（n_samples, n_components）转换为一维向量返回
```

特征提取：

提取特征前先对图像数据进行处理，原始的图像数据中存在图像大小不一的情况，这会对特征提取的结果造成影响，先将所有的图像统一重塑为 128×128 的矩阵。循环检查每个图像，如果元素数量为 16384（即 128x128 像素的图像），则直接将其重塑为 128x128 的矩阵。如果元素数量为 262144（即 512x512 像素的图像），则将其重塑为 512x512 的矩阵。对于 512x512 像素的图像，计算中心点坐标，然后裁剪出中心 128x128 的区域，通过计算中心点坐标，然后从中心点向四周扩展 64 个像素，将裁剪出的中心区域调整为 128x128 大小，以保持图像尺寸的一致性。

n_components 参数指定了要保留的主成分数量，在实验过程中，我使用交叉验证来评估不同 n_components 值对模型性能的影响，从而选择最佳的主成分数量，最终经过结果比较，设定 n_components=150.

```
def extrat_features(images_arr):
    feature = []
    img_arr = []
    for i in range(len(images_arr)):
        # 重新塑形为 128x128 的矩阵
        if len(images_arr[i]) == 16384:
            img = images_arr[i].reshape(128, 128)
        if len(images_arr[i]) == 262144:
            img_reshaped = images_arr[i].reshape(512, 512)
            # 计算裁剪的起始位置，以获取中心 128x128 的区域
            center_x = img_reshaped.shape[1] // 2
            center_y = img_reshaped.shape[0] // 2
            x_start = center_x - 64  # 128/2
            y_start = center_y - 64  # 128/2
            # 裁剪中心的 128x128 区域
            cropped_image = img_reshaped[y_start:y_start + 128, x_start:x_start + 128]
            img = resize(cropped_image, (128, 128), anti_aliasing=True)
        img_arr.append(img)
    img_ = np.array(img_arr)
    features = pca_feature(img_, n_components=150)
```

```
    return features
```

4.3 朴素贝叶斯

朴素贝叶斯(Native Bayes,NB)算法，是一种基于贝叶斯定理与特征条件独立假设的分类方法。“朴素”表示特征条件独立，“贝叶斯”即基于贝叶斯定理，所谓朴素就是在整个形式化过程中只做最原始的假设。朴素贝叶斯模型来自概率论，有着坚实的数学理论基础。贝叶斯分类算法以贝叶斯公式作为理论基础计算每一个属性属于某一类别的概率。朴素贝叶斯是基于各个属性之间相互独立假设的分类方法。

朴素贝叶斯的训练样本由 n 维特征向量 $X=\{X_1,X_2,X_3,\Lambda\ X_n\}$ 表示，描述具有 n 个属性 A 的 n 维度，则有 $A=\{A_1,A_2,A_3,\Lambda\ A_n\}$，表示样本 n 个属性。已有 m 个类别，则类别集合 $C=\{c_1,c_2,\Lambda\ c_n\}$。$P(C_i)$ 是先验概率，当有一个未知数据样本向量 X 时，朴素贝叶斯方法计算在 X 条件下后验概率的最大类别，贝叶斯公式如下：

$$P(C_i|X)=\frac{P(X|C_i)P(C_i)}{P(X)} \tag{38}$$

由于 $P(X)$ 是常数，所以在计算后验概率时只需计算 $P(X|C_i)P(C_i)$ 即可，先验概率 $P(C_i)=\frac{N_{c_i}}{N}$，其中 N 是训练样本数量，C_i 是训练样本中类别的数量。

朴素贝叶斯因为属性间独立性假设，所以计算 $P(X|C_i)$ 公式如下：

$$P(X|C_i)=P(x_1,x_2,\cdots x_n|C_i)=\prod_{j=1}^{n}P(x_j|C_i) \tag{39}$$

当属性是连续属性时，A_i 是连续值，则假设属性服从正态分布，即

$$P(x_j|C_i)=\frac{1}{\sqrt{2\pi}\sigma_{c_i}}e^{\frac{(x_j-\mu_{c_i})^2}{2\sigma_i^2}} \tag{40}$$

其中 μ_{c_i} 和 σ_{c_i} 分别是类别中 C_i 的均值和标准差。对于样本 X 类别，计算每个类别的 $P(X|C_i)P(C_i)$，最后概率最大的那个类别则是样本 X 的预测类别，即：

$$C=argmax\,P(C_i)\prod_{j=1}^{n}P(x_j|C_i) \tag{41}$$

4.4 Adaptive Boosting 算法(AdaBoost)

4.4.1 算法介绍

AdaBoost 算法是典型的 Boosting 算法。Boosting 算法是将弱分类器提升为强

分类器的过程。弱分类器是指在某个学习任务上略优于随机猜测的模型，准确率略高于 50%，通常，弱分类器的性能不足以独立解决整个问题，但通过组合多个弱分类器，可以得到一个强大的模型。AdaBoost 算法的核心思想是对所有样本初始化一个权重，在算法的一开始，每个样本的权重是一样的，即每个样本被选择到的概率相同。然后选择一个特征，只用这个特征来进行分类，得到一个弱分类器，下一步对样本的权重进行重新分配，对于那些被识别错的样本分配更高的权重，对于识别正确的样本给予低的权重，在此基础上选择另一个特征进行分类，得到一个新的弱分类器，如此反复循环，最后对弱分类器进行加权平均，得到最终的分类器。随着弱分类器数量的增加，AdaBoost 算法的最终分类器在训练集上的错误率会越来越小。

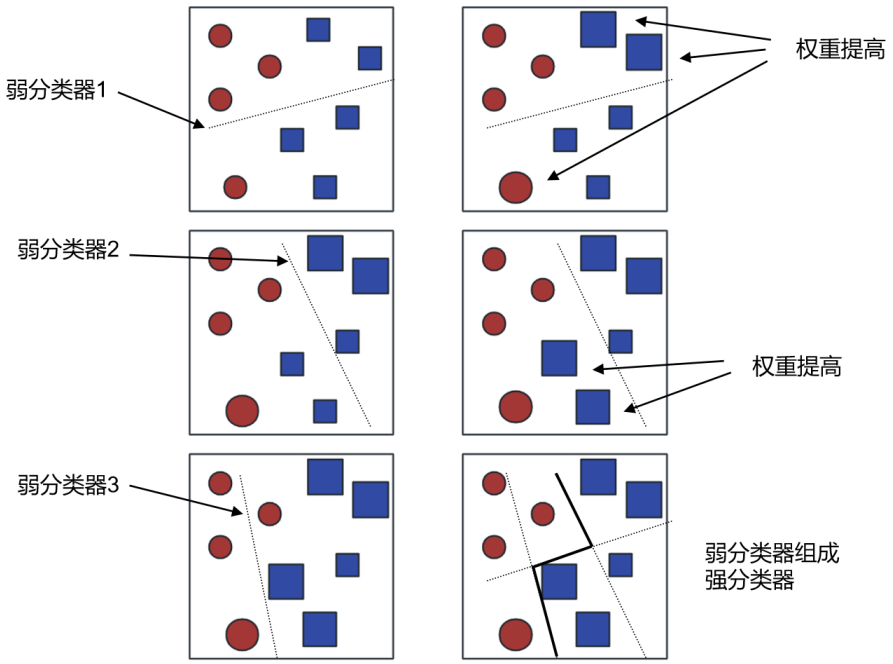


图 4.4.1 AdaBoost 算法原理图

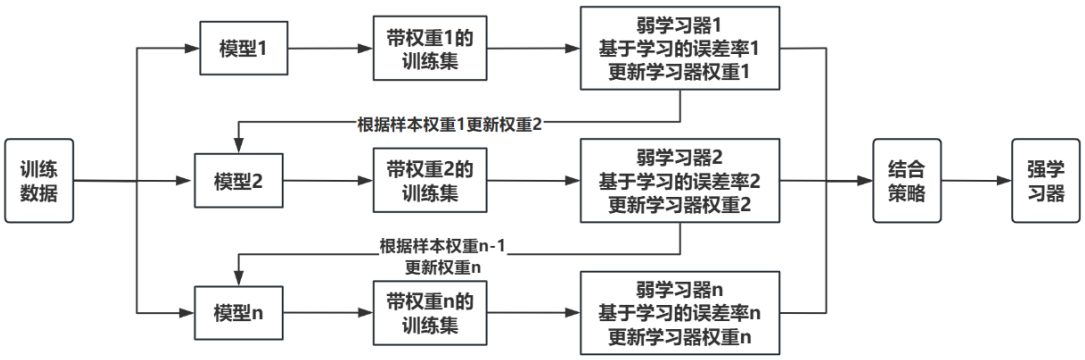


图 4.4.2 AdaBoost 算法步骤

4.4.2 算法原理

假设一个二分类训练样本集（以本项目中的性别分类为例）：

$$T = (x_1, y_1), (x_2, y_2), \cdots, (x_m, y_m) \tag{42}$$

训练集在第 k 个弱学习器的输出权重为：

$$D(k) = (w_{k1}, w_{k2}, \cdots, w_{km}); w_{1i} = \frac{1}{m}; i = 1,2, \cdots m \tag{43}$$

第 k 个弱分类器 $G_k(x)$ 在训练集上的加权误差率为：

$$e_k = P(G_k(x_i) \neq y_i) = \sum_{i=1}^{m} w_{ki} I(G_k(x_i) \neq y_i) \tag{44}$$

第 k 个弱分类器 $G_k(x)$ 的权重系数为：

$$\partial_k = \frac{1}{2} log \frac{1 - e_k}{e_k} \tag{45}$$

更新样本权重 D，假设第 k 个弱分类器的样本集权重系数为 $D(k) = (w_{k1}, w_{k2}, \Lambda, w_{km})$，则对应的第 $k+1$ 个弱分类器的样本集权重系数为：

$$w_{k+1,i} = \frac{w_{ki}}{Z_k} exp(-\partial_k y_i G_k(x_i)) \#(46)$$

这里 Z_k 是规范化因子：

$$Z_k = \sum_{i=1}^{m} w_{ki}\, exp(-\partial_k y_i G_k(x_i)) \tag{46}$$

从 $w_{k+1,i}$ 计算公式可见，如果第 i 个样本分类错误，则 $y_i G_k(x_i) < 0$，导致样本的权重在第 $k+1$ 个弱分类器中增大，如果分类正确，则权重在第 $k+1$ 个弱分类器中减少。

最后是集合策略，AdaBoost 算法采用的是加权表决法，构建基本分类器的线性组合：

$$f(x) = \sum_{k=1}^{K} \partial_k\, G_k(x) \tag{47}$$

最终的强分类器为：

$$G(x) = sign(f(x)) = sign(\sum_{k=1}^{K} \partial_k G_k(x)) \tag{48}$$

为防止过拟合，通常会有正则化，定义为 ν 对于前面的学习器的迭代：

$$f_k(x) = f_{k-1}(x) + \partial_k G_k(x) \tag{49}$$

如果加上正则化项则有：

$$f_k(x) = f_{k-1}(x) + \nu \partial_k G_k(x) \tag{50}$$

ν 的取值范围为 $0 < \nu \leq 1$。对于同样步长的训练集学习结果，较小的 ν 意味着需要更多的若学习器的迭代次数。

模型训练（以性别分类为例）：

在模型训练过程中，使用了分层 K 折交叉验证（StratifiedKFold），这是一种

在机器学习中常用的交叉验证方法，特别是在分类问题中。它的核心目的是确保每次划分的训练集和测试集都能保持原始数据集中的类别比例。参数 n_splits=k：指定了 K 折交叉验证中的“K”，即数据将被分成 K 份，经过多次分类结果比较，最终选择 K=15 的分类准确率是较高的。使用 train_index 和 test_index 从特征数组 features_arr 和标签数组 gender_label 中划分出训练集和测试集，在每一轮交叉验证中，使用训练集数据训练模型，并在测试集上评估模型的性能。

将 4 中标签分类进行训练，使用 Scikit-learn 库中实现 AdaBoost 算法的类 AdaBoostClassifier，GaussianNB()是高斯朴素贝叶斯分类器，将其作为 AdaBoost 算法中的弱学习器。参数 n_estimators 指定了要使用的弱学习器的数量。参数 random_state=42 是用来设置随机种子以确保结果的可重复性。在 AdaBoost 的训练过程中，每个弱学习器会根据前一个弱学习器的错误对样本权重进行调整，然后再次训练。这个过程会重复 n_estimators 次，每次都会关注那些之前被错误分类的样本，AdaBoost 分类器会结合所有弱学习器的预测结果来做出最终的预测。每个弱学习器的预测结果会根据其准确性被赋予不同的权重。经过多次训练结果比较，最终将模型的参数 n_estimators 设置为 50。

```
def gender_bys_ada(gender_label, features_arr, k, n):
    features_arr = np.array(features_arr)
    skf = StratifiedKFold(n_splits=k, shuffle=True, random_state=42)
    all_accuracies = []
    for train_index, test_index in skf.split(features_arr, gender_label):
        # 划分训练集和测试集特征、标签
        x_train_gender, x_test_gender = features_arr[train_index],
features_arr[test_index]
        y_train_gender, y_test_gender = np.array(gender_label)[train_index],
np.array(gender_label)[test_index]
        # 创建 Adaboost 分类器对象，使用高斯朴素贝叶斯作为弱分类器
        gender_clf = AdaBoostClassifier(GaussianNB(), n_estimators=n,
random_state=42)
        # 使用划分好的训练集数据训练模型
        gender_clf.fit(x_train_gender, y_train_gender)
        # 使用训练好的模型对测试集进行预测
        gender_pred = gender_clf.predict(x_test_gender)
        accuracy = accuracy_score(y_test_gender, gender_pred)
        all_accuracies.append(accuracy)
    average_accuracy = np.mean(all_accuracies)
    print("gender average(fold={}):".format(k), average_accuracy)
    #在全部数据上重新训练模型
    final_gender_clf = AdaBoostClassifier(GaussianNB(), n_estimators=n,
random_state=42)
    final_gender_clf.fit(features_arr, np.array(gender_label))
    final_gender_pred = final_gender_clf.predict(features_arr)
    gender_label = np.array(gender_label)
    print_result(gender_label, final_gender_pred)
```

4.5 实验结果分析

对每个标签分类进行训练，得到的结果分析如下，包括了 15 折交叉验证的平均准确率（gender average、age average、race average、expression average），以及总的准确率（total），还有宏平均（macro average）和加权平均（weighted average）的精确度（precision）、召回率（recall）和 F1 分数（f1）：

- 性别分类结果

15 折分层交叉验证的平均准确率为 0.7617，总准确率为 0.8423。

表 4.5.1 性别分类结果

	precision	recall	f1
macro average	0.8393	0.8369	0.8380
weighted average	0.841	0.8423	0.8420

性别分类的准确率相对较高，特别是在总准确率上。宏平均和加权平均的指标也显示出模型在性别分类上表现良好。性别是一个二分类问题，所以高精确度和召回率表明模型在区分两个类别上做得不错。

- 年龄分类结果

15 折分层交叉验证的平均准确率为 0.7722，总准确率为 0.8553。

表 4.5.2 年龄分类结果

	precision	recall	f1
macro average	0.8131	0.7828	0.7897
weighted average	0.8735	0.8553	0.8605

年龄分类的平均准确率略低于性别分类，但总准确率较高，表明模型在整体上对年龄的预测较为准确。宏平均和加权平均的精确度、召回率和 F1 分数都较高，尤其是加权平均的精确度非常高，这可能意味着模型对某些年龄组的预测非常准确。

- 种族分类结果

15 折分层交叉验证的平均准确率为 0.9199，总准确率为 0.9574。

表 4.5.3 种族分类结果

	precision	recall	f1
macro average	0.7954	0.8318	0.8122
weighted average	0.9602	0.9574	0.9586

种族分类的平均准确率和总准确率都非常高，显示出模型在种族分类任务上表现非常好。加权平均的精确度和召回率都非常高，这表明模型在不同种族的分类上有很好的性能。

- 表情分类结果

  15 折分层交叉验证的平均准确率为 0.6310，总准确率为 0.7196。

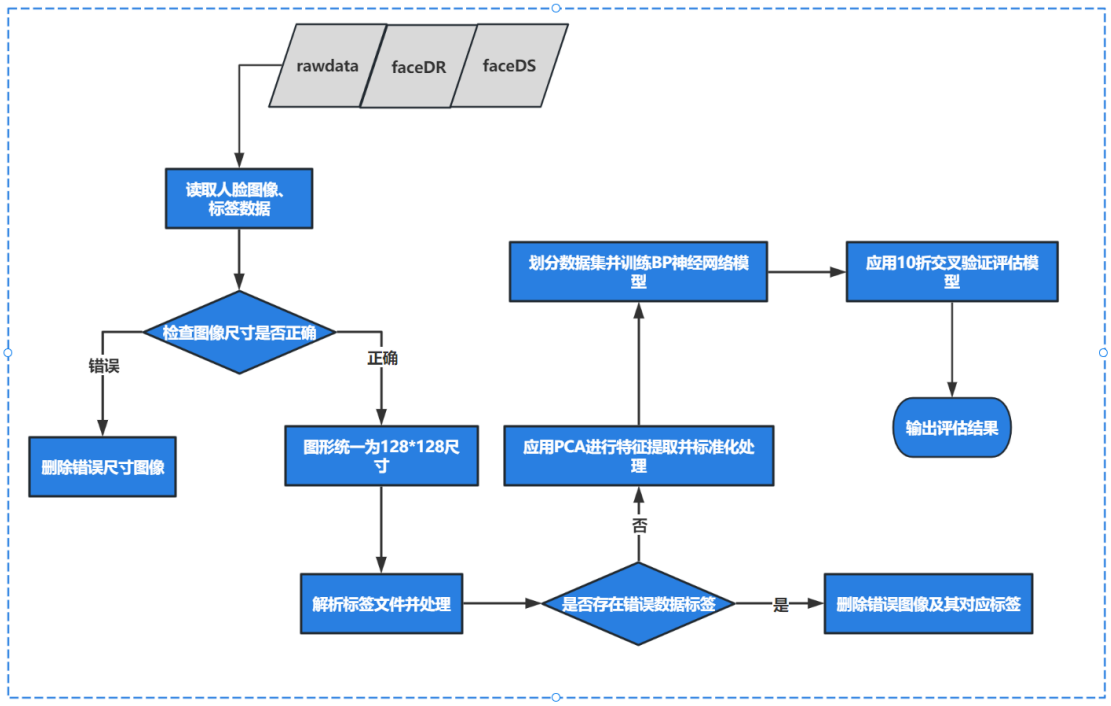
表 4.5.4 表情分类结果

	precision	recall	f1
macro average	0.7605	0.7823	0.7709
weighted average	0.7191	0.7196	0.7192

表情分类的平均准确率和总准确率相对较低，这可能表明表情分类是一个更具挑战性的任务。宏平均和加权平均的精确度、召回率和 F1 分数也相对较低，表明模型在表情分类上的性能不如其他任务。

5 PCA + BP 神经网络算法

5.1 设计框架



5.1.1 设计框架

5.2 数据预处理

5.2.1 人脸图像预处理

加载、统一尺寸、展平图像数据，同时记录尺寸不正确的文件信息。

首先定义列表 images 用于存储处理后的图像数据，以及列表 error 用于记录尺寸错误的文件信息。然后，遍历目标文件夹中的所有图像文件，对每个图像文件进行如下处理：构建文件完整路径，以二进制只读模式打开文件，将文件数据读取为无符号 8 位整数的 NumPy 数组。如果图像数据的大小不是 16384（128x128），则将文件名记录到 error 列表并继续处理下一个文件。接着，将图

像数据整理成 128x128 的二维数组，并展平为一维数组，将展平后的数组加入 images 列表中。最后，将 images 列表转换为 NumPy 数组，并返回处理后的图像数据数组以及尺寸错误的文件信息列表。

5.2.2 人脸标签预处理

根据标签文件中的内容提取性别、年龄、种族和表情等属性信息，并将其转换为数字形式，存储在属性列表中。

首先，将行内容按空格分割并去除首尾空格，以获得各部分内容。然后，创建一个长度为 4 的属性列表 attributes，用于存储解析后的属性信息。接着，如果第二部分为'(_missing'，表示丢失数据标签，则跳过当前行；如果第一部分在错误列表 error 中，表示错误数据标签，则跳过当前行。接下来，将文字信息以数字形式存储，解析信息如下。解析性别信息：如果第三部分为'male)'，将性别属性设为 0（代表男性）；如果第三部分为'female)'，将性别属性设为 1（代表女性）；否则，将性别属性设为-1（代表未知）。然后，解析年龄信息：根据第五部分内容判断年龄信息，并赋予相应值（0 代表儿童，1 代表青少年，2 代表成年人，3 代表老年人）。接着，解析种族信息：根据第七部分内容判断种族信息，并赋予相应值（0 代表白种人，1 代表黑种人，2 代表亚洲人，3 代表西班牙裔）。最后，解析表情信息：根据第九部分内容判断表情信息，并赋予相应值（0 代表微笑，1 代表认真，2 代表有趣）。将解析后的属性列表 attributes 添加到 labels 列表中。最终，返回包含解析后属性信息的 labels 列表。

5.3 特征提取

利用主成分分析（Principal Component Analysis）算法对输入的图像数据进行降维处理，提取出主要特征。

首先，利用 PCA 算法对输入的图像数据 images 进行处理。在初始化 PCA 对象时，设定主成分数量为 150（n_components=150），启用白化处理（whiten=True），并采用随机化方法进行奇异值分解（svd_solver="randomized"）。接着，对输入的图像数据进行 PCA 模型的拟合，以便后续的特征提取。随后，将拟合后的 PCA 模型应用于输入数据，将其转换为具有更少特征维度的特征表示。最后，返回经 PCA 处理后的特征数据 features。

5.3.1 PCA 算法原理

PCA(Principal Component Analysis)，即主成分分析方法，是一种使用最广泛的数据降维算法。PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。PCA 的工作就是从原始的空间中顺序地找一组相互正交的坐标轴，新的坐标轴的选择与数据本身是密切相关的。其中，第一个新坐标轴选择是原始数据中方差最大的方向，第二个新坐标轴选取是与第一个坐标轴正交的平面中使得方差最大的，第三个轴是与第 1,2 个轴正交的平面中方差最大的。依次类推，可以得到 n 个这样的坐标轴。通过这种方式获得的新的坐标轴，我们发现，大部分方差都包含在前面 k 个坐标轴中，后面的坐标轴所含的方差几乎为 0。于是，我们可以忽略余下的坐标轴，只保留前面 k 个含有绝大部分方差的坐标轴。事实上，这相当于只保留包含绝大部分方差的维度特征，而忽略包含方差几乎为 0 的特征维

度，实现对数据特征的降维处理。

5.3.2 参数解析

1、n_component 这个参数指定了希望从原始数据中提取的主成分（特征）的数量。在这里，设置为 150，意味着 PCA 将提取 150 个主成分。

2、whiten 参数白化处理是一种数据预处理步骤，它通过减去均值并除以标准差来使数据具有零均值和单位方差。这有助于消除不同特征量纲的影响，使得 PCA 能够更公平地处理所有特征。

3、svd_solver 参数指定了用于计算奇异值分解（SVD）的算法。参数"randomized"使用随机抽样来估计协方差矩阵的奇异值，在该大数据集上，"randomized" 方法通常比默认的 "full" 方法更快。

5.4 BP 神经网络训练与验证

对给定的属性标签和特征数据进行多层感知器神经网络的训练和评估，以预测不同属性的分类结果，并输出各属性的性能指标。

通过循环遍历每个属性标签（性别、年龄、种族、表情），将特征数据 features 与当前属性的标签数据 labels[:, i]分割为训练集和测试集，保持类别分布的一致性。接着，对训练数据进行标准化处理，利用 StandardScaler 进行数据缩放，确保数据具有相似的尺度。然后，建立多层感知器分类器（MLPClassifier），设定模型参数如求解器（solver）、学习率等，并进行训练。在此过程中采用了交叉验证（10 折）来评估模型的性能，输出当前属性的准确率（Accuracy）。训练完整个神经网络后，计算并输出精确率（Precision）、召回率（Recall）和 F1 分数（F1 Score）等性能指标。随后，对每个属性重复上述步骤进行训练和评估。

5.4.1 BP 神经网络原理

BP 神经网络是一种按误差反向传播训练的多层前馈网络，其算法称为 BP 算法，它的基本思想是梯度下降法，利用梯度搜索技术，以期使网络的实际输出值和期望输出值的误差均方差为最小。

基本 BP 算法包括信号的前向传播和误差的反向传播两个过程。即计算误差输出时按从输入到输出的方向进行，而调整权值和阈值则从输出到输入的方向进行。正向传播时，输入信号通过隐含层作用于输出节点，经过非线性变换，产生输出信号，若实际输出与期望输出不相符，则转入误差的反向传播过程。

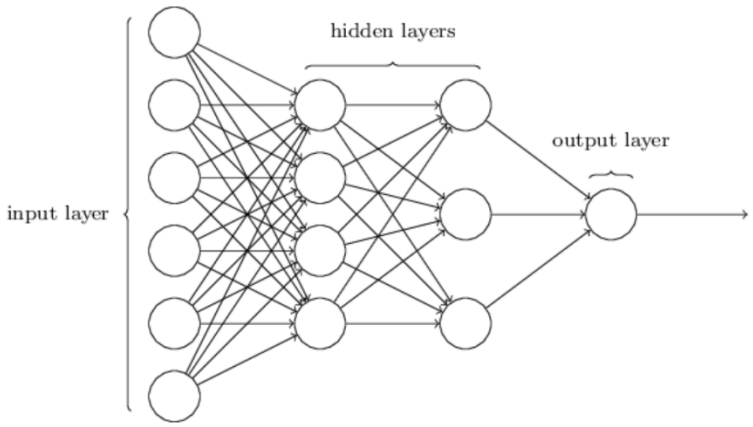


图 5.4.1 神经网络传输示意图

误差反传是将输出误差通过隐含层向输入层逐层反传，并将误差分摊给各层所有单元，以从各层获得的误差信号作为调整各单元权值的依据。通过调整输入节点与隐层节点的联接强度和隐层节点与输出节点的联接强度以及阈值，使误差沿梯度方向下降，经过反复学习训练，确定与最小误差相对应的网络参数(权值和阈值)，训练即告停止。此时经过训练的神经网络即能对类似样本的输入信息，自行处理输出误差最小的经过非线形转换的信息。

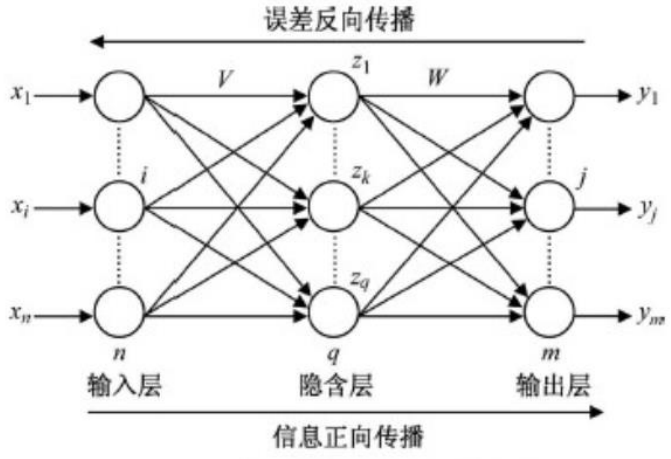


图 5.4.2 误差与信息传播方向示意图

在 BP 神经网络中，误差信号反向传递过程比较复杂，它是基于 Widrow-Hoff 学习规则的。假设输出层的所有结果为 djdj，则误差函数为

$$E(w,b)=\frac{1}{2}\sum_{j=0}^{n-1}\left(d_j-y_j\right)^2 \tag{51}$$

而 BP 神经网络的主要目的是反复修正权值和阀值，使得误差函数值达到最小。Widrow-Hoff 学习规则是通过沿着相对误差平方和的最速下降方向，连续调整网络的权值和阈值，根据梯度下降法，权值矢量的修正正比于当前位置上 E(w,b)的梯度，对于第 j 个输出节点有：

$$\Delta w(i,j) = -\eta \frac{\partial E(w,b)}{\partial w(i,j)} \tag{52}$$

隐含层和输出层中对 E 求 w 和 b 的偏导：

$$\frac{\partial E(w,b)}{\partial w_{ij}} = \frac{1}{\partial w_{ij}} \cdot \frac{1}{2} \sum_{j=0}^{n-1} \left(d_j - y_j\right)^2 = \delta_{ij} \cdot x_i \tag{53}$$

$$\frac{\partial E(w,b)}{\partial b_j} = \delta_{ij} \tag{54}$$

其中：

$$\delta_{ij} = \left(d_j - y_j\right) \cdot \frac{f(S_j)[A - f(S_j)]}{AB} \tag{55}$$

有了上述公式，根据梯度下降法，那么对于隐含层和输出层之间的权值和阀值调整如下：

$$w_{ij} = w_{ij} - \eta_1 \cdot \frac{\partial E(w,b)}{\partial w_{ij}} = w_{ij} - \eta_1 \cdot \delta_{ij} \cdot x_i \tag{56}$$

$$b_j = b_j - \eta_2 \cdot \frac{\partial E(w,b)}{\partial b_j} = b_j - \eta_2 \cdot \delta_{ij} \tag{57}$$

5.4.2 K 折交叉验证

K 折交叉验证（K-Fold Cross-Validation）是一种统计学方法，用于评估并比较机器学习模型的性能。它通过将数据集分成 K 个不相交的子集（或“折”），然后使用 K-1 个子集作为测试集，其余 K-1 个子集作为训练集，重复这个过程 K 次，每次选择不同的子集作为测试集。这种方法可以更全面地评估模型的性能，因为它确保了每个数据点都被用于训练和测试。

5.5 实验结果与分析

模型评估平均结果如下：

属性	准确率 (Accuracy)	精确率 (Precision)	召回率 (Recall)	F1分数 (F1 Score)
性别	81%	97.17%	97.14%	97.12%
年龄	81%	99.43%	99.43%	99.43%
种族	92%	99.82%	99.82%	99.82%
表情	72%	99.64%	99.64%	99.64%

由表格可见该模型在年龄和种族的识别上表现最佳，精确率、召回率和 F1 分数都接近或达到 99%。性别识别的表现也非常好，但略低于年龄和种族。表情识别的准确率相对较低，尽管精确率和召回率仍然很高，这可能表明模型在表情识别上存在一定的误差，可能需要进一步优化。这也可能是由于表情的微妙变化较大所导致。

6 总结

本报告通过四种不同的算法方法对人脸识别技术进行了深入研究:PCA+SVM、基于 L1 正则化的 Lasso 回归和深度神经网络、基于朴素贝叶斯的 AdaBoost 算法,以及 PCA 与 BP 神经网络的组合。通过这些多样化的方法,我们获得了若干关于不同人脸属性分类方法有效性的重要见解。

在所有实现方案中都观察到一个一致的模式:种族和年龄分类普遍获得较高的准确率,而表情分类则在所有方法中都得不尽人意。这表明某些具有更显著和稳定特征的面部特征可能更容易被机器识别,而其他特征(特别是表情)则蕴含更微妙的变化,让分类难上加难。

PCA+SVM 方法在种族分类方面表现出色,F1 分数高达 0.981,但在年龄分类方面表现较差,F1 分数仅为 0.311。采用 L1 正则化和 Lasso 回归的深度神经网络实现通过多次优化迭代显示出改进,最终版本在性别分类上达到 0.6389 的准确率,年龄分类达到 0.8323,种族分类达到 0.8967,表情分类达到 0.5372。

基于朴素贝叶斯的 AdaBoost 实现在种族分类上实现了 0.9199 的 15 折交叉验证平均准确率,在性别和年龄分类上也表现厉害,但表情识别的准确率仍然只有 0.6310。PCA 与 BP 神经网络的方法在所有类别中都表现出色,特别是在年龄和种族分类方面,准确率接近 99%。

通过本课程项目设计我们对模式识别这门课程有了一些新的理解。首先,特征提取方法的选择对分类性能有显著影响,这一点在 AdaBoost 实现中 LBP 和 PCA 方法的对比中得到了证明。其次,数据预处理步骤,特别是图像标准化和直方图均衡化,对提高模型性能至关重要。第三,集成方法和混合方法的有效性很明显,这表明结合多种技术可以帮助克服单一方法的局限性。

在该课程项目的设计与实现过程中我们也遇到了许多的难题,包括但不限于捕捉微妙面部表情的难度、适当的数据预处理的重要性,以及在不同属性类别之间需要平衡的训练数据。这些挑战指出了未来研究的潜在方向,例如开发更复杂的表情识别特征提取方法,或探索先进的数据增强技术来解决类别不平衡问题。

总的来说,虽然每种算法方法都展示出独特的优势和局限性,但整体研究为不同机器学习技术在面部属性分类中的有效性提供了有价值的见解。结果表明,混合方法将多种方法的优势相结合可能是最有效的综合人脸识别系统方案,特别是在处理多样化的属性分类时。

7 参考文献

1. Cunha, J.C., O.F. Rana, and P.D. Medeiros, *Future Trends in Distributed Applications and Problem-solving Environments.* 2005.

2. Ahonen T, Hadid A, Pietikäinen M. Face description with local binary patterns: Application to face recognition[J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006, 28(12): 2037-2041.

3. Zhang W, Shan S, Gao W, et al. Local gabor binary pattern histogram sequence (LGBPHS): A novel non-statistical model for face representation and recognition[C]//Tenth IEEE International Conference on Computer Vision. IEEE, 2005: 786-791.

4. Turk M, Pentland A. Eigenfaces for recognition[J]. Journal of cognitive neuroscience, 1991, 3(1): 71-86.

5. Wu X, Kumar V, Quinlan J R, et al. Top 10 algorithms in data mining[J]. Knowledge and information systems, 2008, 14(1): 1-37.

6. Freund Y, Schapire R E. A decision-theoretic generalization of on-line learning and an application to boosting[J]. Journal of computer and system sciences, 1997, 55(1): 119-139.

7. Lecun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

8 附录

表 A1 成员分工及贡献度自评

姓名	个人分工	小组任务贡献度
谭家骏	“L1正则化+Lasso回归+深度神经网络+K折交叉验证”报告攥写以及算法实现	25
张扬	“模式识别目标与任务”“PCA+SVM算法”报告攥写以及算法实现	30
李智欣	“基于朴素贝叶斯的Adaboost算法”报告攥写以及算法实现	24
游垚明	“PCA+BP神经网络算法”报告攥写以及算法实现	21

课程体会与建议

内容丰富，挑战性强，课程项目设计很有参与感，大家都收获颇丰。

Github 地址：
https://github.com/Zxin66/Pattern_Recognition_work.git