廣東工業大學

课 程 报 告

课程名称 模式识别

题目名称 基于人脸图像的年龄识别

专业班级 17 级自动化创新班

学号姓名 3117000961 廖睿翔

3117000819 陈漪皓

指导教师 邢延

2020 年 06 月 15 日

目录

1 模式识别系统设计

1.1 目标类别

在该人脸识别课程设计中，本小组针对人脸图像，进行类别为年龄的有监督学习。其中，年龄具体分为 4 类，分别为，adult(成年人)、child(儿童)、teen(青少年)以及 senior(年长者)；

1.2 数据特点

(1) 数据类别方面，在本实验数据集中，共有 4000 张原始数据，其中无效数据数量为 7，因此，在剔除无效数据后，有效数据总量为 3993。每一类别所对应的数据量以及数据总量如表 1 所示。

表 1 数据类别及数据量

数据类别	adult	child	teen	senior	total
数据量	3168	312	344	169	3993
类型占比	79.34%	7.81%	8.62%	4.23%	100%

由表 1 可知，adult 类型的数据占实验数据集绝大部分比重(79.34%)，而其余三种数据均占较小比重，故该数据集中，数据类别及其数量没有达到绝对均衡。

(2) 数据颜色通道方面，数据集中所有图片均为单通道，即只有 8 位二进制灰度信息，并不具备 RGB 三通道。

(3) 数据维度方面，在 3993 张有效图片中，编号为 2412 及 2416 的两张图片的文件大小为 256KB，即图像尺寸大小为 512×512=262144(像素)；其余编号的图片的文件大小均为 16KB，即图像尺寸大小为 128×128=16384(像素)。因此，在数据清洗、变换等过程中，需要对这一细节进行判断。

(4) 数据量方面，本数据集中共有 3993 张有效人脸数据，而在基于机器学习、深度学习技术背景的实验下，图像类型的数据数量达到万级别甚至十万级别时较为理想，因此，本数据集总量并未达到完全理想的情况。

1.3 实验环境及开发工具

本实验所采取的实验环境以及开发工具如表 2 所示。

表 2 实验环境以及开发工具

操作系统	Windows 10
集成开发环境 (IDE)	VSCode 编辑器 + Python 插件
NVIDIA 通用并行计算架构	CUDA 10.0
版本控制系统	Git

1.4 Python 依赖及其版本

本实验所安装的 Python 依赖及其版本如表 3 所示。

表 3　Python 依赖及相应版本

依赖	版本	依赖	版本
Numpy	1.18.5	Scikit-image	0.17.1
Matplotlib	3.2.1	Scikit-learn	0.19.2
Pandas	1.0.3	Keras	2.3.1
XGBoost	1.1.0	TensorFlow	2.2.0
OpenCV2	4.2.0	Torch	1.4.0
OpenCV3 (专利)	4.0.0.21	Torchvision	0.5.0

1.5　模式识别方法

本小组实现将原始数据进行特征提取、特征降维，最终使用机器学习分类器，结合评估指标，进行的完整图像分类流程[1]。

1.5.1　特征提取算法

本小组两位成员最终确定采用的特征提取算法如表 4 所示。

表 4 特征提取算法

	经典算法	改进算法
廖睿翔	LBP (局部二值模式，LocalBinaryPattern)	DenseNet201
陈漪皓	HOG (方向梯度直方图，Histogram of Oriented Gradient)	ResNet50

1.5.2　特征降维算法

由于原始数据经过特征提取后，数据维度仍然较大，导致后续分类时间较长。因此，为了降低数据维度，本小组添加特征降维算法，与未进行特征降维的特征提取与分类过程进行对比，如表 5 所示。

表 5 特征降维算法

经典算法	改进算法
PCA (主成分分析， Principal Component Analysis)	KPCA (基于核函数的主成分分析， Kernel Principal Component Analysis)

1.5.3　分类算法

本小组两位成员最终确定采用的分类算法如表 6 所示。

表 6 分类算法

	经典算法	智能/改进算法
廖睿翔	Adaboost	CNN (卷积神经网络， Convolutional Neural Network)
陈漪皓	SVM (支持向量机， Support Vector Machine)	XGBoost

1.6　评估指标

本小组采用的评估指标及其对应中文术语如表 7 所示。需要注意的是，在

Scikit-learn 框架 metrics 模块中，precision、recall、f1-score 已经设置‘weighted’参数，即**考虑了数据不平衡**的情况。

表 7 评估指标

英文术语	中文术语
k-fold cross validation	k 折交叉验证
accuracy_score	准确率得分
precision_score	精确度
recall_score	召回率
f1_score	——
confusion_matrix	混淆矩阵
error_matrix	犯错矩阵

2 准备工作

2.1 数据集划分

原始数据中包含以下三个部分：(1) 所有人脸图片的原始文件，以二进制形式存放；(2) 已经过 PCA (主成分分析)进行特征降维的图片数据，以矩阵形式存放，并按照 0.5：0.5 的比例划分训练集与测试集；(3) 与(2)中数据相对应的类别标签及索引序号。

为了体现模式识别系统过程的完整性，本小组将所有数据标签进行汇总，并将所有以二进制形式存放的图片文件转化为矩阵形式，生成“图片索引序号-所对应的矩阵形式的图片文件-类别标签”的待分类数据。

得到全部未处理过的图片矩阵文件以及对应类别标签后，采用 Scikit-learn 框架中 model_selection 模块下的 train_test_split()函数，按照训练集数据量：测试集数据量=0.7：0.3 的比例对数据集进行划分，并规定随机种子，以确保每次需要划分数据集时执行函数的结果相同，便于复现实验。

完成划分数据集后，训练集与测试集中数据类别及对应数量如表 8 所示。

表 8 训练集与测试集数据类别及数量

	数据类别	adult	child	teen	senior	total
训练集	数据量	2215	221	242	117	2795
	类型占比	79.25%	7.91%	8.66%	4.19%	100%
测试集	数据量	953	91	102	52	1198
	类型占比	79.55%	7.60%	8.51%	4.34%	100%

2.2 算法选择

在最终确定表 4、表 5 及表 6 中的模式识别算法之前，本小组进行了多种算法的组合仿真进行性能比较。由于所采取的分类算法基本针对二分类问题进行设计，因此，针对本小组所研究的多类别分类任务，采用 Scikit-learn 框架 multiclass 模块中的 OneVsOneClassifier()类进行“一对一(OVO)”多分类任务学习，与 Scikit-learn 框架 multiclass 模块中的 OneVsRestClassifier()类进行“一对其余(OVR)”多分类任务学习。评估指标采取 accuracy_score(准确率得分)与完整耗时，结果如表

9 所示。

注：初步对比算法时，均采用默认参数设定。其中：

(1) SIFT：特征提取算法(尺度不变特征变换，Scale-invariant feature transform)；

(2) KNN：分类算法(k 最近邻，k-NearestNeighbour)；

(3) RF：分类算法(随机森林，Random Forest)；

(4) Softmax：分类算法，CNN 网络内部设置。

表 9 实验算法组合

方法组合	score	耗时(s)	方法组合	score	耗时(s)
SIFT+KNN (OVO)	0.795		LBP+KNN (OVO)	0.795	
SIFT+KNN (OVR)	0.795	310.01	LBP+KNN (OVR)	0.795	922.66
SIFT+PCA+KNN (OVO)	0.795		LBP+PCA+KNN (OVO)	0.795	
SIFT+PCA+KNN (OVR)	0.795	261.41	LBP+PCA+KNN (OVR)	0.795	310.01
SIFT+SVM (OVO)	0.795		LBP+RF (OVO)	0.802	
SIFT+SVM (OVR)	0.795	623.57	LBP+RF (OVR)	0.811	929.27
SIFT+PCA+SVM (OVO)	0.795		LBP+PCA+RF (OVO)	0.856	
SIFT+PCA+SVM (OVR)	0.795	365.24	LBP+PCA+RF (OVR)	0.855	791.96
LBP+SVM (OVO)	0.795		LBP+Adaboost (OVO)	0.911	
LBP+SVM (OVR)	0.795	13983	LBP+Adaboost (OVR)	0.920	3812.51
LBP+PCA+Adaboost(OVO)	0.910		HOG+SVM (OVO)	0.928	
LBP+PCA+Adaboost(OVR)	0.908	797.81	HOG+SVM (OVR)	0.927	652.90
HOG+PCA+SVM (OVO)	0.945		ResNet50+XGBoost(OVO)	0.920	
HOG+PCA+SVM (OVR)	0.945	513.99	ResNet50+XGBoost(OVR)	0.926	265.04
ResNet50+KPCA+XGBoost (OVO)	0.923		DenseNet201+CNN (Softmax)	0.885	575.62
ResNet50+KPCA+XGBoost (OVR)	0.924	236.71	DenseNet201+KPCA +CNN (Softmax)	0.913	313.45
LBP+CNN (Softmax)	0.795		10800		

观察表 9 可知，(1) 将特征提取算法 SIFT、LBP，特征降维算法 PCA，与分类算法 KNN、SVM、CNN 分别进行组合后，所得准确率得分均为 0.795；(2) 将特征提取算法 LBP，特征降维算法 PCA 与分类算法 RandomForest 分别进行组合后，所得准确率得分小幅超过 0.795，分别超过 0.9%(LBP+RF (OVO))、2.0%(LBP+RF (OVR))、7.7%(LBP+PCA+RF (OVO))与 7.5%(LBP+PCA+RF (OVR))；(3) 将特征提取算法 LBP、HOG、ResNet50、DenseNet201，特征降维算法 PCA、KPCA 与分类算法 Adaboost、SVM、XGBoost、CNN 分别进行组合后，所得的准确率得分均大幅超过 0.795，超出比例至少为 14.2%(LBP+PCA+Adaboost (OVR))；(4) 在采用相同的特征提取算法与分类算法的情况下，添加特征降维过程能够明显降低图像分类所消耗时间。

因此，针对上述初步实验结果，本实验最终以 LBP+Adaboost、HOG+SVM、ResNet50+XGBoost 及 DenseNet201+CNN 为 baseline，采取特征降维算法 PCA 与 KPCA 作为评估指标以及消耗时间的对照，并对上述 Adaboost、SVM、XGBoost、CNN 方法进行参数调优，旨在找到使得分类效果最佳的参数。

2.3 图片还原

为了更好地运用所提供的数据，加深对完整模式识别系统流程的理解，将原始二进制形式的图片文件，结合 OpenCV2 模块将二进制格式存放的图片文件转换为 jpg 格式。从 adult、child、teen、senior 四类图片中分别取出 1 张进行展示，结果如图 1 所示。

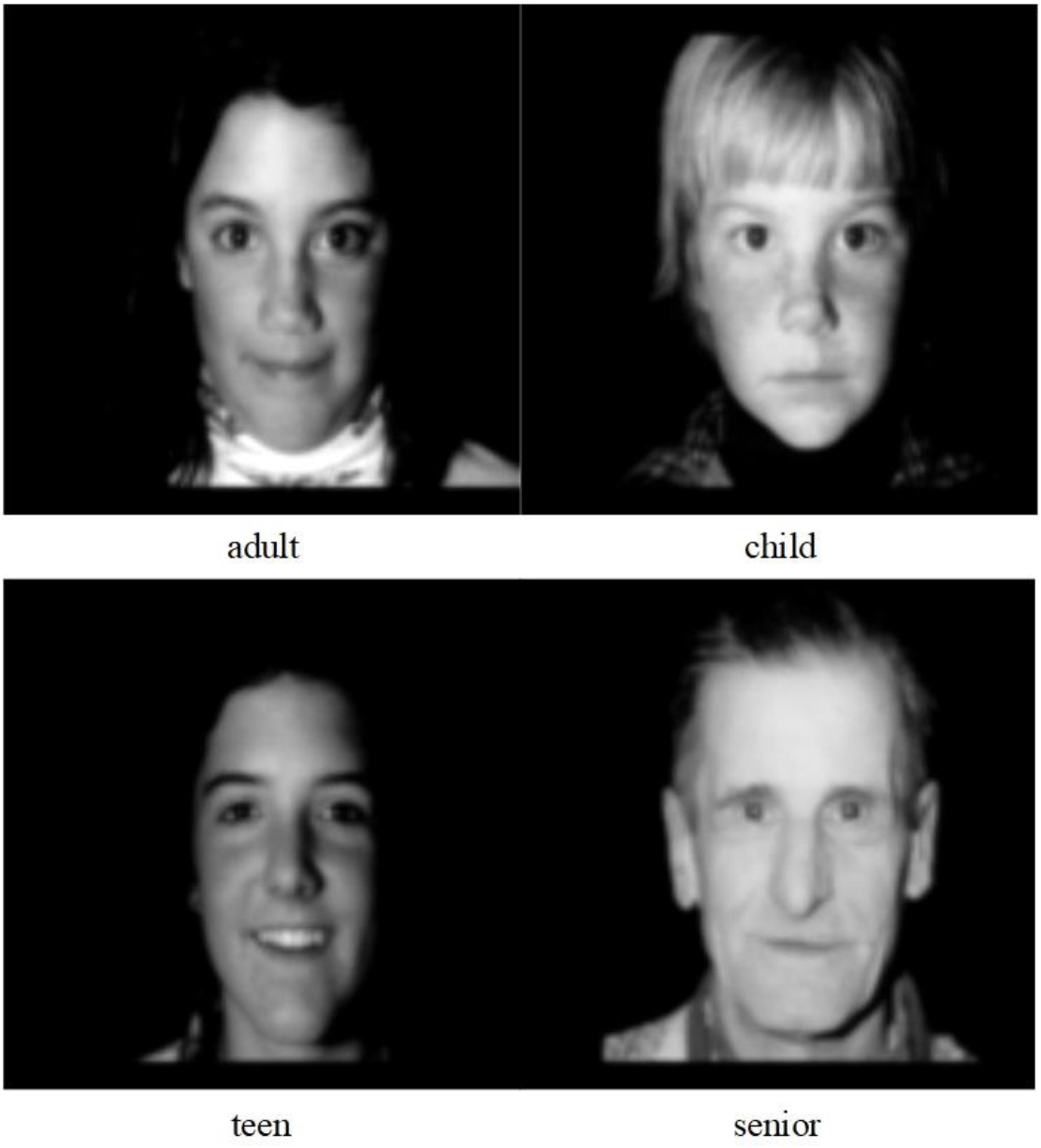


图 1 四种类别人脸图片代表

3 基于 Adaboost 的模式识别分类方法

3.1 特征提取算法

3.1.1 经典算法：LBP

3.1.1.1 算法原理

LBP(Local Binary Pattern，局部二值模式)是一种用来描述图像局部纹理特征的算子；它具有旋转不变性和灰度不变性等显著的优点[2]。原始的 LBP 算子定义为在 3*3 的窗口内，以窗口中心像素为阈值，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。这样，3*3 邻域内的 8 个点经比较可产生 8 位二进制数(通常转换为十进制数即 LBP 码，共 256 种)，即得到该窗口中心像素点的 LBP 值，并用这个值来反映该区域的纹理信息。

对 LBP 特征向量进行提取的步骤如下：

(1) 首先，将检测窗口划分为 16×16 的小区域(cell)；

(2) 对于每个 cell 中的一个像素，将相邻的 8 个像素的灰度值与其进行比较，若周围像素值大于中心像素值，则该像素点的位置被标记为 1，否则为 0。这样，3*3 邻域内的 8 个点经比较可产生 8 位二进制数，即得到该窗口中心像素点的 LBP 值；

(3) 然后，计算每个 cell 的直方图，即每个数字(假定是十进制数 LBP 值)出现的频率；然后对该直方图进行归一化处理；

(4) 最后，将得到的每个 cell 的统计直方图进行连接成为一个特征向量，也就是整幅图的 LBP 纹理特征向量。

3.1.1.2 参数调整与设定

进行 LBP 特征提取时，基于 Scikit-image 框架 feature 模块，调用 local_binary_pattern()函数进行 LBP 算法特征提取。部分参数说明及设定如表 10 所示。

表 10 LBP 算法参数设定

参数	中文说明	设定值
image	灰度图像	——
P	圆对称邻接集点的数目(角空间的量化)	24
R	圆的半径(操作符的空间分辨率)	8
method	确定图案的方法	default

由于该函数参数设定较为固定，因此 LBP 算法最终参数设定如表 10 所示。

3.1.2 智能算法：DenseNet201

3.1.2.1 算法原理

ResNet 中的路层连接设计引申出了数个后续工作。DenseNet(稠密连接网络)与 ResNet 的主要区别如图 2 所示。

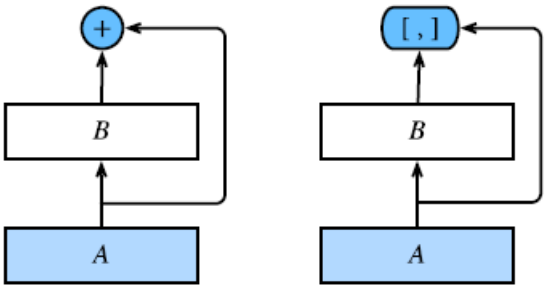


图 2　ResNet(左)和 DenseNet(右)在跨层连接上区别

图 2 中将部分前后相邻的运算抽象为模块 A 和模块 B。与 ResNet 的主要区别在于，DenseNet 里模块 B 的输出并不像 ResNet 那样和模块 A 的输出相加，而是在通道维上连接。因此，模块 A 的输出可以直接传入模块 B 后面的层。在这个设计里，模块 A 直接和模块 B 后面的所有层连接在了一起。这也是它被称为“稠密连接”的原因[3]。DenseNet 的主要构建模块是稠密块和过渡层。前者定义了输入和输出是如何连接的，后者则用来控制通道数，使之不过大。DenseNet 系列网络结构如图 3 所示。

3.1.2.2　参数调整与设定

进行 DenseNet 特征提取时，基于 Pytorch 框架，调用 torchvision 模块中 models 子类下已训练并集成完毕的 DenseNet201 网络结构进行 DenseNet 算法特征提取，并调用 CUDA 进行并行运算。由于数据集中图片为单通道，而 DenseNet201 网络需要输入三通道数据，因此需要通过 numpy 模块将二维数据转换为三维，进而将单通道数据变换为三通道。查看源码，得部分参数设定如图 4 所示。

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

图 3　DenseNet 系列网络结构

```
__constants__ = ['features']

    def __init__(self, growth_rate=32, block_config=(6, 12, 24, 16),
                 num_init_features=64, bn_size=4, drop_rate=0, num_classes=
1000, memory_efficient=False):

        super(DenseNet, self).__init__()

        # First convolution
        self.features = nn.Sequential(OrderedDict([
            ('conv0', nn.Conv2d(3, num_init_features, kernel_size=7, stride=2,
                                padding=3, bias=False)),
            ('norm0', nn.BatchNorm2d(num_init_features)),
            ('relu0', nn.ReLU(inplace=True)),
            ('pool0', nn.MaxPool2d(kernel_size=3, stride=2, padding=1)),
        ]))

        # Each denseblock
        num_features = num_init_features
        for i, num_layers in enumerate(block_config):
            block = _DenseBlock(
                num_layers=num_layers,
                num_input_features=num_features,
                bn_size=bn_size,
                growth_rate=growth_rate,
                drop_rate=drop_rate,
                memory_efficient=memory_efficient
            )
            self.features.add_module('denseblock%d' % (i + 1), block)
            num_features = num_features + num_layers * growth_rate
            if i != len(block_config) - 1:
                trans = _Transition(num_input_features=num_features,
                                    num_output_features=num_features // 2)
                self.features.add_module('transition%d' % (i + 1), trans)
                num_features = num_features // 2
```

```
        # Final batch norm
        self.features.add_module('norm5', nn.BatchNorm2d(num_features))

        # Linear layer
        self.classifier = nn.Linear(num_features, num_classes)

        # Official init from torch repo.
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
            elif isinstance(m, nn.Linear):
                nn.init.constant_(m.bias, 0)

    def forward(self, x):
        features = self.features(x)
        out = F.relu(features, inplace=True)
        out = F.adaptive_avg_pool2d(out, (1, 1))
        out = torch.flatten(out, 1)
        out = self.classifier(out)
        return out
```

图 4　DenseNet201 网络参数

3.2　特征降维算法

3.2.1　经典算法：PCA

3.2.1.1　算法原理

PCA(主成分分析，Principal Component Analysis)基本思想是从一组特征中计算出一组按照重要性的大小从大到小依次排列的新特征，它们是原有特征的线性组合，并且新特征之间不相关。计算出原有特征在新特征上的映射值，即得到新的降维后的样本。因此，PCA 的目标是用一组正交向量来对原特征进行变换得到新特征，新特征是原有特征的线性组合[4]。

假设数据未进行中心化，输入数据$X = \{x^1, x^2, \dots, x^n\}_{p\times n}$，n 个 p 维的列向量。

(1) 数据中心化：$\mu = \frac{1}{n}\sum_{i=1}^{n} x^i, x^i = x^i - \mu, i = 1,2,\dots,n$；

(2) 计算协方差矩阵$\sum \frac{1}{n} XX^T$，(X 为中心化后的)，$\sum$为$p \times p$维对称矩阵；

(3) 对协方差矩阵$\sum$进行特征值分解，得到特征值$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p$，选择前

k 个特征值所对应的特征向量，构成投影矩阵$A = [a_1, a_2, \dots, a_k]$，$a_i$为$p$维特征向量；

(4) 将原样本投影到新的特征空间，得到新的降维样本，$X' = W^T X$，X'为$p \times n$维矩阵。

3.2.1.2 参数调整与设定

进行 PCA 特征降维时，基于 Scikit-image 框架 feature 模块，调用 PCA()函数进行 PCA 算法特征降维。部分参数说明及设定如表 11 所示。

表 11 PCA 算法参数设定

参数	中文说明	设定值
n_components	PCA 降维后的特征维度数目	99
copy	是否将原始数据复制	True
whiten	白化，对降维后的数据的每个特征进行标准化，使方差都为 1	False
svd_solver	奇异值分解 SVD 的方法	auto

由于该函数参数设定较为固定，因此 PCA 算法最终参数设定如表 11 所示。

3.2.2 改进算法：KPCA

3.2.2.1 算法原理

KPCA (基于核函数的主成分分析，Kernel Principal Component Analysis)在人脸识别领域中是一种重要的非线性数据降维方法。它的主要原理为：首先，将输入空间中的样本数据 $X = \{x_i | i = 1,2,\cdots,n; x_i \in R^d\}$ 通过非线性函数H 映射到高维特征空间G中，则有 $x_i \rightarrow H(x_i)$；然后在这个高维特征空间 G 中做主元分析，提取其中的非线性特征信息[5]。假设已提前将映射数据做了零均值处理，则经过映射过程之后可以得到数据的协方差矩阵为：

$$C = \frac{1}{n}\sum_{i} H^T(x_i) \times H(x_i) \tag{1}$$

对协方差矩阵 C 进行特征矢量分析，定义其特征值为 λ，对应的特征向量为 v，则可以用 $H(xk)$的线性组合来表示 C 的特征向量，即有：

$$v = \sum_{n}^{j=1} a_j H(x_j) \tag{2}$$

式中，$a_j(j = 1,2,\dots,n)$是常数。接下来引入核函数$K(x_i, x_j) = H(x_i) \times H(x_j)$与核矩阵$K = (K_{ij})$，$K_{ij} = K(x_i, x_j)$，进一步分析化简得：

$$\bar{\lambda} a = Ka \tag{3}$$

式中，$\bar{\lambda} = n\lambda$，那么最终将问题转化为了求 K 的特征值与特征向量。

在高维特征空间 G 中计算 v的投影，将会得到其第 t个核主元，即有：

$$T_t = [v_t \times H(x)] = \sum_{i=1}^{n} a_{t,i} K(x_i, x) \tag{4}$$

由上式得到的T_t就是对X做 KPCA 处理后得到的非线性主元.

3.2.2.2 参数调整与设定

进行 KPCA 特征降维时，基于 Scikit-image 框架 feature 模块，调用 KPCA() 函数进行 KPCA 算法特征降维。部分参数说明及设定如表 11 所示。

表 11 KPCA 算法参数设定

参数	中文说明	设定值
n_components	KPCA 降维后的特征维度数目	99
kernel	核	linear
max_iter	最大迭代次数	None
copy_X	是否将原始数据复制	True
tol	收敛惩罚系数	0
eigensolver	特征分解方法	auto

由于该函数参数设定较为固定，因此 KPCA 算法最终参数设定如表 11 所示。

3.3 分类算法

3.3.1 经典算法：Adaboost

3.3.1.1 算法原理

(1) 初始化训练数据的权值分布：

$$W = (w_{11}, w_{12}, \dots, w_{1N}), w_{11} = \frac{1}{N}, i = 1,2, \dots N \tag{5}$$

(2) 使用具有权值分布的D_m训练数据集学习，得到基本分类器$G_m(x)$，计算$G_m(x)$在训练集上的分类误差率：

$$e_m = P(G_m(x_i) \neq y_i) = \frac{\sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i)}{\sum_{i=1}^{N} w_{mi}} = \sum_{i=1}^{N} w_{mi} I(G_m(x_i) \neq y_i) \tag{6}$$

(3) 计算$G_m(x)$的系数：

$$\alpha_m = \frac{1}{2} log \frac{1-e_m}{e_m} \tag{7}$$

(4) 更新训练数据集的权值分布(Z_m是规范化因子，它使$D_m + 1$成为一个概率分布)：

$$W_{m+1,i} = \frac{w_{mi}}{Z_m} exp\left(-\alpha_m y_i G_m(x_i)\right) \tag{8}$$

$$Z_m = \sum_{i=1}^{N} w_{mi} exp\left(-\alpha_m y_i G_m(x_i)\right) \tag{9}$$

(5) 构建分类器的线性组合，得到最终的分类器[6]：

$$f(x) = \sum_{m=1}^{N} \alpha_m G_m(x) \tag{10}$$

$$G(x) = sign\big(f(x)\big) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right) \tag{11}$$

3.3.1.2　参数调整与设定

进行 Adaboost 分类时，基于 Scikit-learn 框架 ensemble 模块，调用 AdaBoostClassifier()类进行 Adaboost 算法分类。由于 Adaboost 主要针对二分类问题，因此，针对本实验所探究的多年龄段分类问题，采用 Scikit-learn 框架 multiclass 模块中的 OneVsOneClassifier()类进行“一对一(OVO)”多分类任务学习，与 Scikit-learn 框架 multiclass 模块中的 OneVsRestClassifier()类进行“一对其余(OVR)”多分类任务学习。

调整参数时，借助 Scikit-learn 框架 model_selection 模块，调用 GridSearchCV 类进行参数网格搜索。同时，设置模型调参评估方法为’precision’，在训练集上做 10 折交叉验证，找到最优参数后对测试集进行分类、预测。设置待调整参数及其对应值程序如图 5 所示，调参结果如图 6 所示。

```
# 设置gridsearch的参数
    tuned_parameters = [
        {
            'learning_rate': [0.01, 0.1, 1],
            'algorithm': ['SAMME', 'SAMME.R'],
            'n_estimators': [50, 100, 200],
        }
    ]

    # 模型调参评估方法
    score = 'precision'

    ada_clf = GridSearchCV(
        AdaBoostClassifier(),
        tuned_parameters,
        cv=10,
        scoring='%s_weighted' % score,
    )

    # 只在训练集上面做k-fold,然后返回最优的模型参数
    ada_clf.fit(X_train, y_train.values.ravel())

    print('Best parameters set found on development set:')
    print(ada_clf.best_params_)
    print('Grid scores on development set:')
    for params, mean_score, scores in ada_clf.grid_scores_:
        print('%0.3f (+/-%0.03f) for %r'
              % (mean_score, scores.std() * 2, params))
```

图 5　待调整参数程序

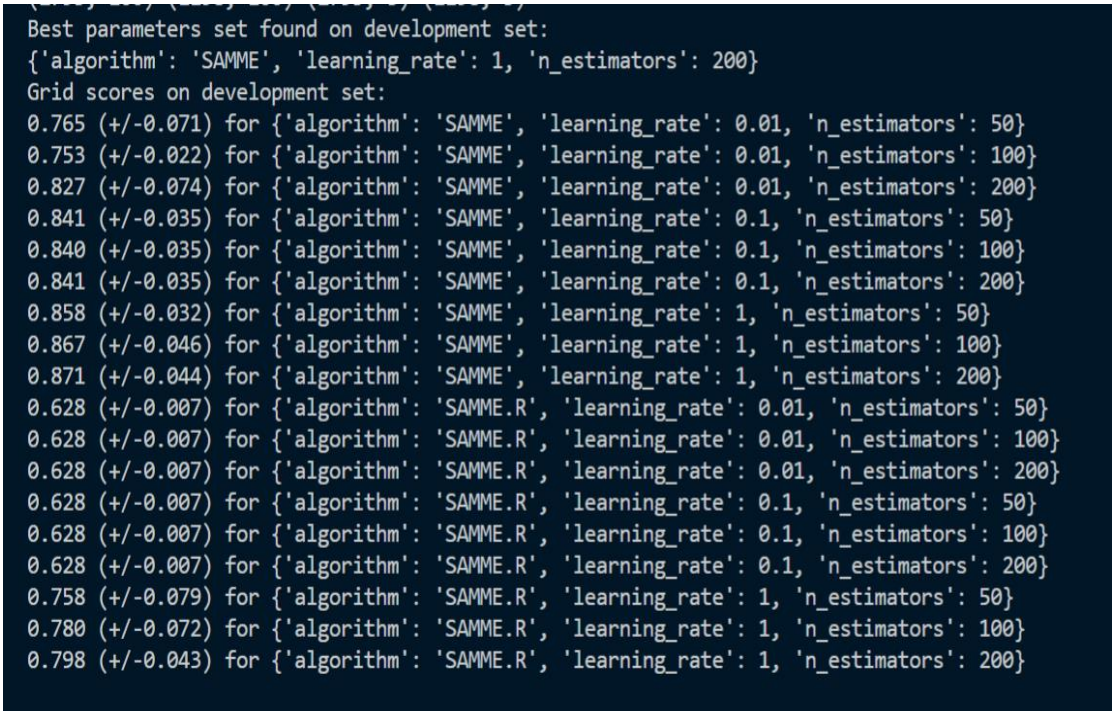


图 6 调参结果

由图 6 可知，在训练过程中，寻找到局部最优解时所对应的待调整参数为：algorithm: ‘SAMME’，learning_rate: 1，n_estimators: 200，取得 0.871±0.044 的准确率。故最终 Adaboost 分类器所采用的参数设定如表 12 所示。

表 12 Adaboost 参数设定

参数	中文说明	设定值
base_estimator	弱分类器	DecisionTreeClassifier
n_estimators	最大的弱学习器的个数	200
learning_rate	每个弱学习器的权重缩减系数	1
algorithm	Adaboost 分类算法	SAMME

3.3.2 智能算法：CNN

3.3.2.1 算法原理

CNN (卷积神经网络，Convolutional Neural Network)在结构上由卷积层网络、池化层网络、全连接层网络、Softmax 层网络组成，区别于 BP 网络的输入-多隐含-输出三类网络层结构[7]，如图 7 所示。

卷积层是构成 CNN 网络的基本结构之一，每层的卷积层需要定义卷积核大小与计算步长，卷积层上一层的输出作为它的输入，其输入可以是原始数据输入层、池化层输出与卷积层输出。卷积层中卷积核所包含的参数都与传统神经网络中的权值参数相当，图 8 为卷积一般计算过程。

如图 8 所示，输入为 6*6 大小的二维矩阵特征图，其中具体数字代表的是像素的灰度值(0-255)，Cov Kernel 卷积核为大小选择 3*3，之间“*”号代表卷积关系，设置卷积步长为 1。通过滑动卷积核窗口，得到最后的 Cov Layer。

卷积层的计算是区别于一般数学意义上卷积，两个矩阵对应位置的元素相乘后相加，最后还要加上一个偏置，一般数学表达式为：

$$y^{j(r)} = f(b^{j(r)} + \sum_i k^{ij(r)} * x^{i(r)} \quad (12)$$

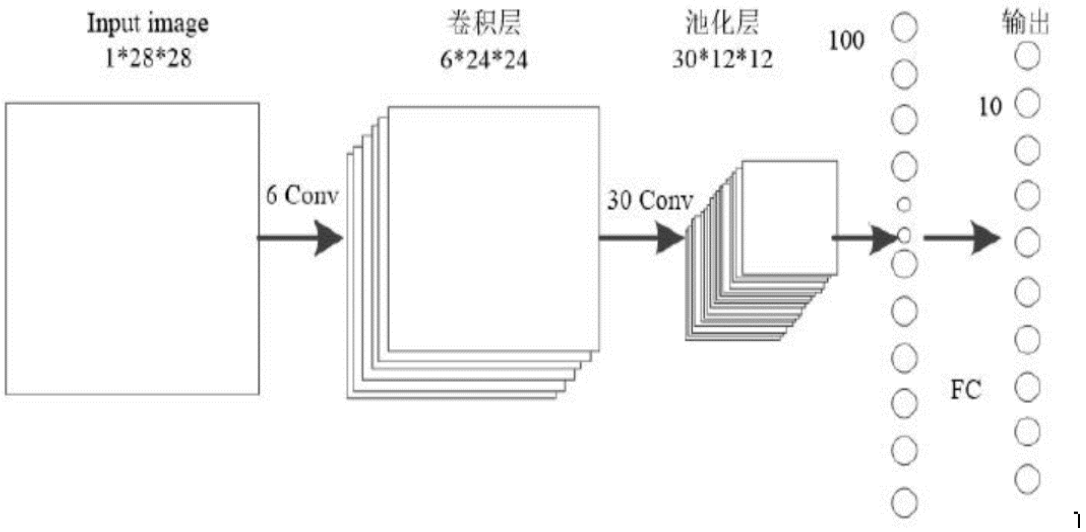


图 7　CNN 网络基本结构

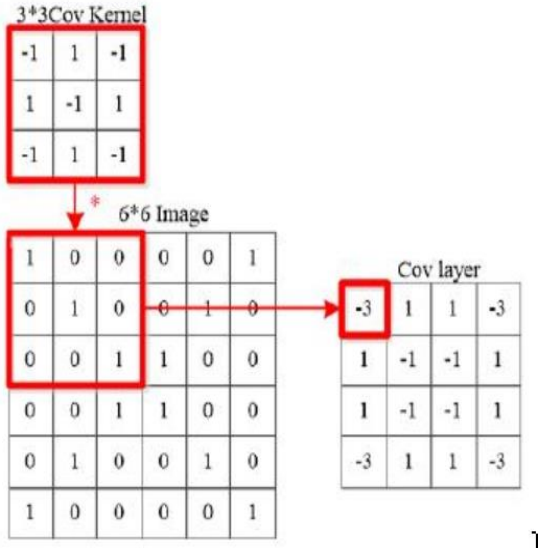


图 8 卷积计算步骤

其中，符号 r 代表当前的层数，符号$x^{i(r)}$代表第 i 个输入特征图，符号$k^{ij(r)}$代表为该层的卷积核，符号 $y^{j(r)}$代表第 j 个输出特征图，符号$b^{j(r)}$代表卷积核对应的偏置值，激活函数保证卷积层的非线性输出，以增强整个神经网络的表达能力。

卷积计算中使用共同的卷积核参与计算，是一种权值共享方式，区别于全连接方式，为局部连接，该连接计算方式减少了学习参数量与计算量。卷积层也可

设置多通道卷积计算，即使用多个卷积核。不同的卷积核大小能获得的不同尺度的表征信息，现在卷积核大小一般取 1*1、3*3、5*5 或 7*7。

在设计 CNN 网络结构中，卷积层与池化层往往成对存在，先完成对特征的提取后，再输入到池化层。设计池化层的目的，对特征向量进行降维处理，获得较低维度表征信息，减少计算量。根据图像各特征位置相对不变性原理，采用聚合统计方法处理图像信息，降低特征对象的维度，用得到的图像概要特征代替全部图像特征，保留有效特征信息，池化层的一般数学公式为：

$$X_j^l = f\left(down\left(X_j^{l-1}\right)\right) \tag{13}$$

其中，符号X_j^{l-1}表示卷积网络上一层的输出，符号$f(.)$ 是该池化层激励函数，符号$down(.)$也是个函数，是下采样函数即池化函数，池化方法主要有 Average Pooling、Max Pooling、Stochastic-pooling。最常用的 Max Pooling 具体操作步骤如图 9 所示。

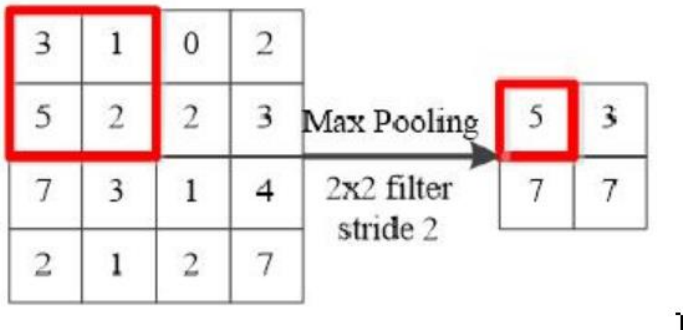


图 9　Max Pooling 具体操作图

全连接层可简称为 FC 层，一般设置在 CNN 的最后。全连接方式指的是本层的神经元点都与上一层的每个神经元点相连，FC 层结构如图 10 所述。

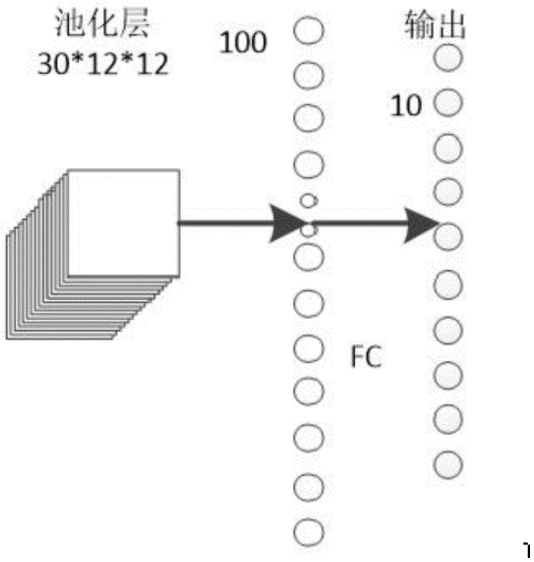


图 10 全连接层工作图

基于端到端 CNN 网络模型的输出，就是分类，即获得所要识别对象的的概率，也就是一个概率值对应一个类别号。全连接层的作用是将高度提纯的特征，高度浓缩为一个具体数值，以便于交给最后的分类器。全连接层一般计算公式为：

$$y_i = f(\sum_i x_i \cdot \omega_{i,j} + b_i) \tag{14}$$

其中，x_i和y_i分别代表该层的输入和输出，ω和b 表示权重和偏移量，激活函数$f(.)$可以为 Sigmoid 激活函数或选择修正线性单元(ReLu)等。全连接层的权值不共享，是一种全连接方法，该层占整个卷积神经网络绝大多数参数量。

CNN 用于分类时，能够抽取高级特征，使得仅使用简单的分类器就能获得满意的各类识别精确率，因而传统上大多数网络最后的输出采用 Softmax 分类器，卷积神经网络在解决多类辨别问题需要用到 Softmax 函数，建立的 Softmax 模型是 Logistic 回归模型的延伸，Softmax 模型所用的概率分布公式如下：

$$p(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{15}$$

利用概率分布知识点，解决多类分类问题，Softmax 用于多分类具体实现步骤如图 11 所示。

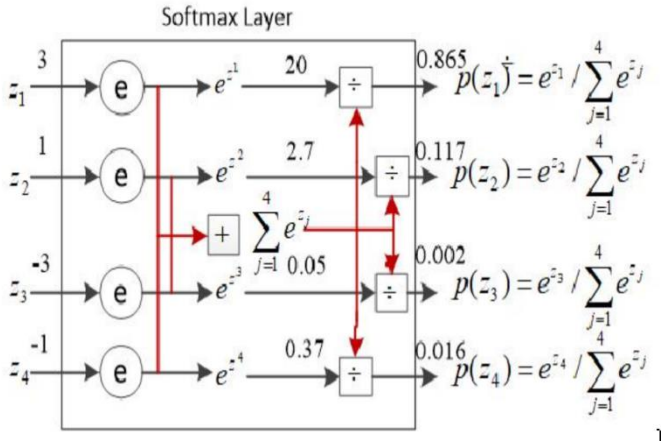


图 11　Softmax 分类步骤图

3.3.2.2　参数调整与设定

进行 CNN 分类时，基于 TensorFlow.Keras 框架，进行卷积神经网络搭建。调整参数时，由于 TensorFlow 及 Keras 框架较难实现 Scikit-learn 中与 GridSearchCV 类相同的网格搜索参数功能。因此，进行手动调整参数。寻找到局部最优解时，网络中的每一层结构及其部分参数设定如表 13 所示。

表 13　CNN 参数设定

	参数	中文说明	设定值
卷积层	filters	卷积中输出滤波器的数量	64
	kernel_size	2D 卷积窗口的高度和宽度	(5, 5)
	padding	零填充方式	same
池化层	pool_size	池化窗口的最大值	(2, 2)
	strides	每个合并步骤的合并窗口移动的距离	(2, 2)
	padding	零填充方式	same
	activation	激活函数	ReLu
卷积层	filters	卷积中输出滤波器的数量	64
	padding	零填充方式	same
	kernel_size	2D 卷积窗口的高度和宽度	(5, 5)
池化层	activation	激活函数	ReLu
	pool_size	池化窗口的最大值	(2, 2)
	strides	每个合并步骤的合并窗口移动的距离	(2, 2)
Flatten	dense	输出空间维数	128
	activation	激活函数	ReLu
全连接层	dense	输出空间维数	128
	activation	激活函数	softmax
训练	epoch	迭代轮数	100
	batch_size	每次训练时送入的样本数	32

3.4　实验结果

3.4.1　LBP-Adaboost

3.4.1.1　实验数据

将原始数据经过 LBP 进行特征提取后，对 Adaboost 分类器分别进行 OVO 及 OVR 的封装。再将特征提取得到的结果送往分类器，得到分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)，分别如表 14、表 15、表 16 所示。

表 14　Classification Report (LBP-Adaboost)

method	class/metric	precision	recall	f1-score	support
LBP-Adaboost (OVO)	adult	0.96	0.97	0.96	953
	child	0.8	0.76	0.78	91
	teen	0.86	0.85	0.85	52
	senior	0.63	0.58	0.61	102
	weighted avg	0.91	0.92	0.91	1198
LBP-Adaboost (OVR)	adult	0.95	0.97	0.96	953
	child	0.82	0.78	0.8	91
	teen	0.85	0.79	0.82	52
	senior	0.68	0.58	0.62	102
	weighted avg	0.91	0.91	0.91	1198

表 15　Confusion Matrix (LBP-Adaboost)

method	class	adult	child	teen	senior
LBP-Adaboost (OVO)	adult	926	4	6	17
	child	4	69	1	17
	teen	8	0	44	0
	senior	30	13	0	59
LBP-Adaboost (OVR)	adult	925	4	6	18
	child	9	71	1	10
	teen	11	0	41	0
	senior	31	12	0	59

表 16　Error Matrix (LBP-Adaboost)

method	class	adult	child	teen	senior
LBP-Adaboost (OVO)	adult	0	0.044	0.115	0.167
	child	0.004	0	0.019	0.167
	teen	0.008	0	0	0
	senior	0.031	0.143	0	0
LBP-Adaboost (OVR)	adult	0	0.044	0.115	0.176
	child	0.009	0	0.019	0.098
	teen	0.011	0	0	0
	senior	0.032	0.132	0	0

3.4.1.2　可视化

将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图 12 所示。

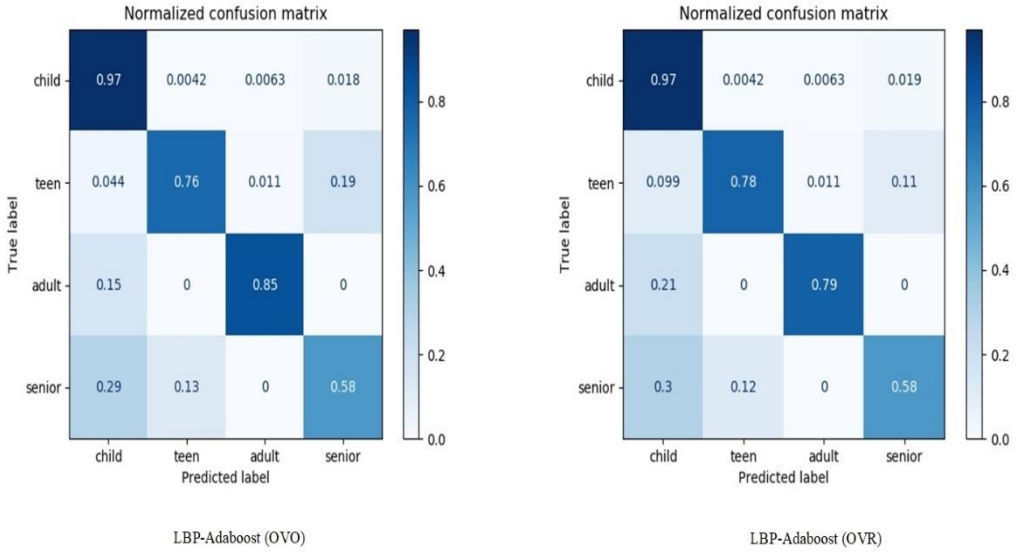


图 12　混淆矩阵　(LBP-Adaboost)

将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 13 所示。

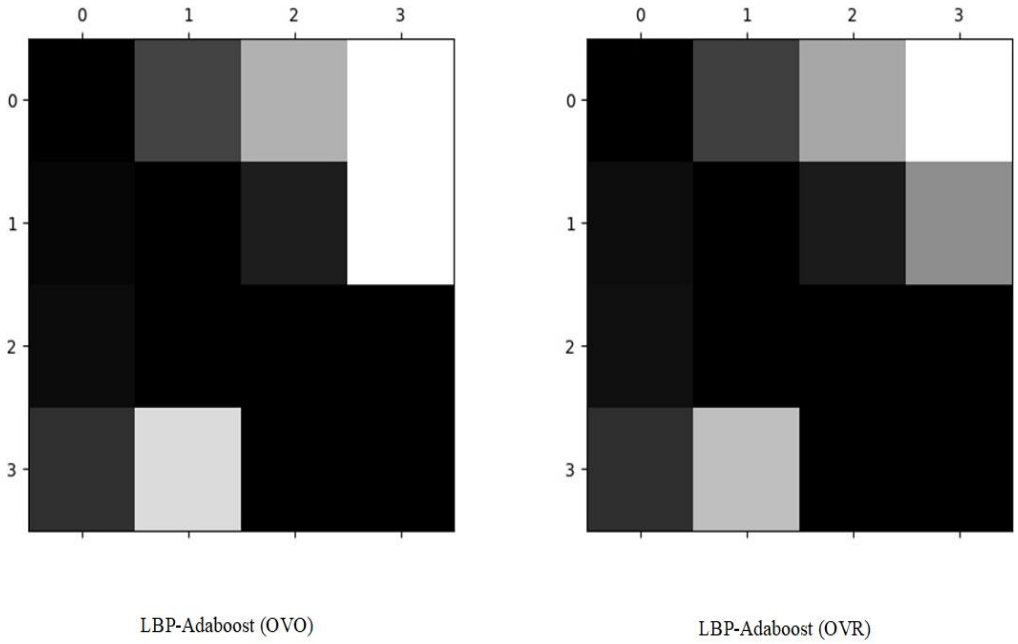


图 13 犯错矩阵 (LBP-Adaboost)

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。其次，Scikit-learn 框架 metrics 模块的 roc_curve()函数仅支持“OVR(一对多)”分类任务，并不支持“OVO(一对一)”分类任务，故此处仅展示 OVR 方法下得到的 ROC 曲线，如图 14 所示。

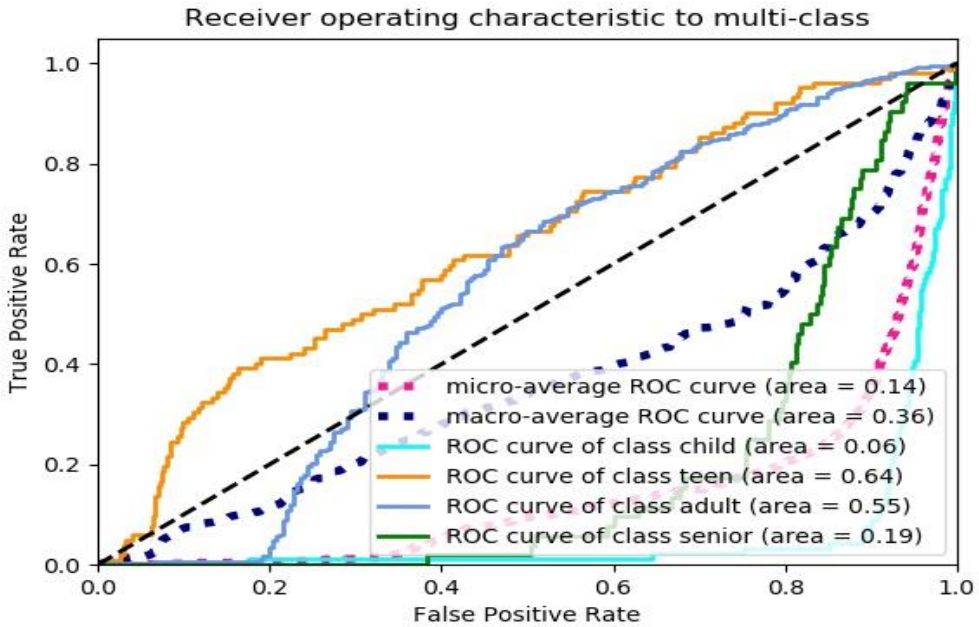


图 14 ROC 曲线 (LBP-Adaboost (OVR))

3.4.2 LBP-PCA-Adaboost

3.4.2.1 实验数据

将原始数据经过 LBP 进行特征提取后，对 Adaboost 分类器分别进行 OVO 及 OVR 的封装。再将特征提取得到的结果送往分类器，得到分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)，分别如表 17、表 18、表 19 所示。

表 17 Classification Report (LBP-PCA-Adaboost)

method	class/metric	precision	recall	f1-score	support
LBP-PCA-Adaboost (OVO)	adult	0.95	0.97	0.96	953
	child	0.74	0.64	0.69	91
	teen	0.84	0.71	0.72	52
	senior	0.56	0.53	0.55	102
	weighted avg	0.89	0.9	0.9	1198
LBP-PCA-Adaboost (OVR)	adult	0.94	0.97	0.95	953
	child	0.79	0.66	0.72	91
	teen	0.84	0.81	0.82	52
	senior	0.58	0.47	0.52	102
	weighted avg	0.89	0.9	0.89	1198

表 18 Confusion Matrix (LBP-PCA-Adaboost)

method	class	adult	child	teen	senior
LBP-PCA-Adaboost (OVO)	adult	927	4	6	16
	child	6	58	1	26
	teen	15	0	37	0
	senior	32	16	0	54
LBP-PCA-Adaboost (OVR)	adult	925	2	7	19
	child	13	60	1	16
	teen	10	0	42	0
	senior	40	14	0	48

表 19 Error Matrix (LBP-PCA-Adaboost)

method	class	adult	child	teen	senior
LBP-PCA-Adaboost (OVO)	adult	0	0.044	0.115	0.157
	child	0.006	0	0.019	0.255
	teen	0.016	0	0	0
	senior	0.033	0.176	0	0
LBP-PCA-Adaboost (OVR)	adult	0	0.022	0.135	0.186
	child	0.014	0	0.019	0.157
	teen	0.01	0	0	0
	senior	0.042	0.154	0	0

3.4.2.2　可视化

将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图15所示。

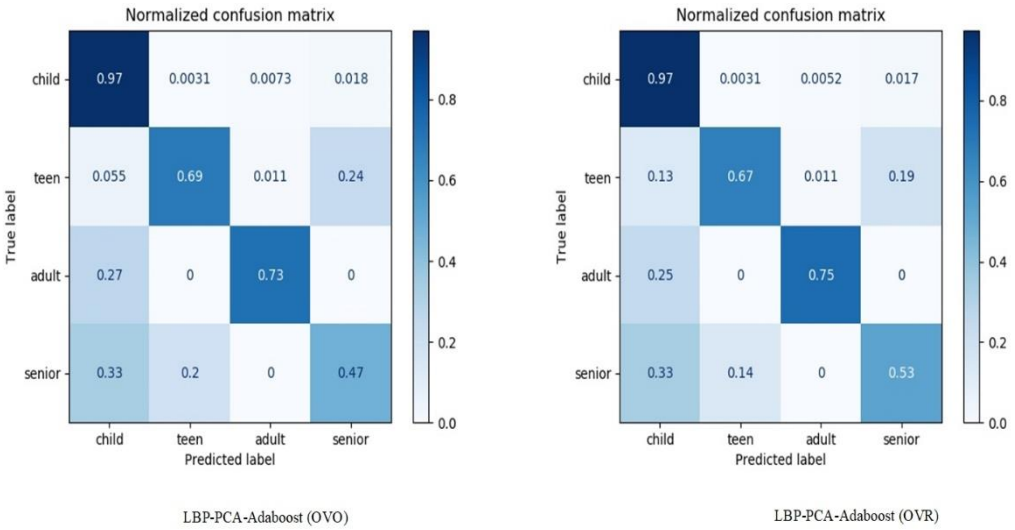


图 15　混淆矩阵　(LBP-PCA-Adaboost)

将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图16所示。

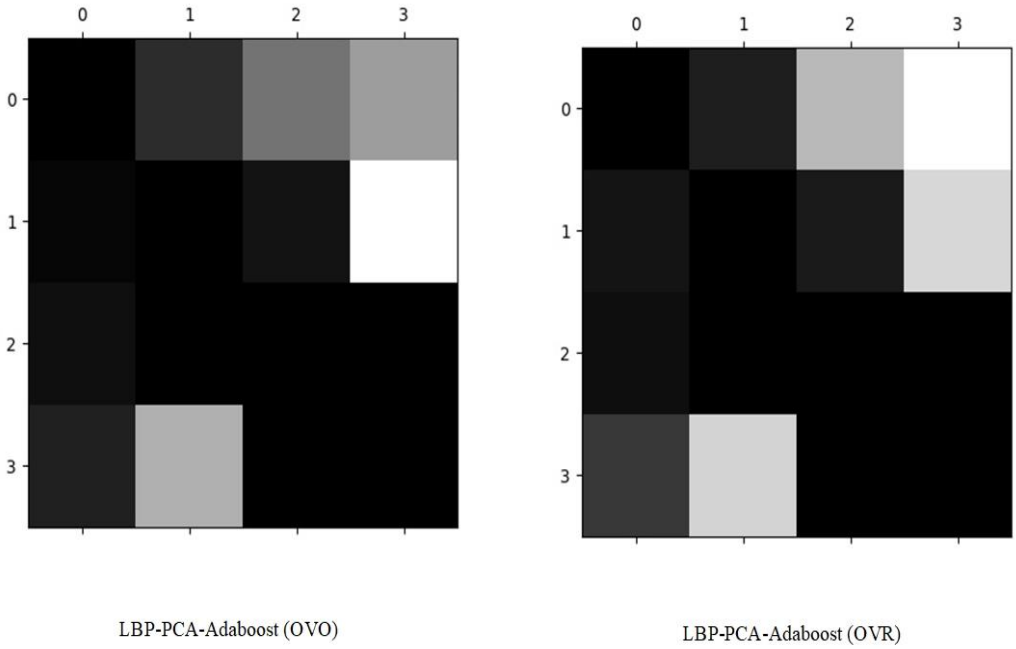


图 16　犯错矩阵　(LBP-PCA-Adaboost)

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。其次，Scikit-learn 框架 metrics 模块的 roc_curve()函数仅支持“OVR(一对多)”分类任务，并不支持“OVO(一对一)”分类任务，故此处仅展示 OVR 方法下得到的 ROC 曲线，如图 17 所示。

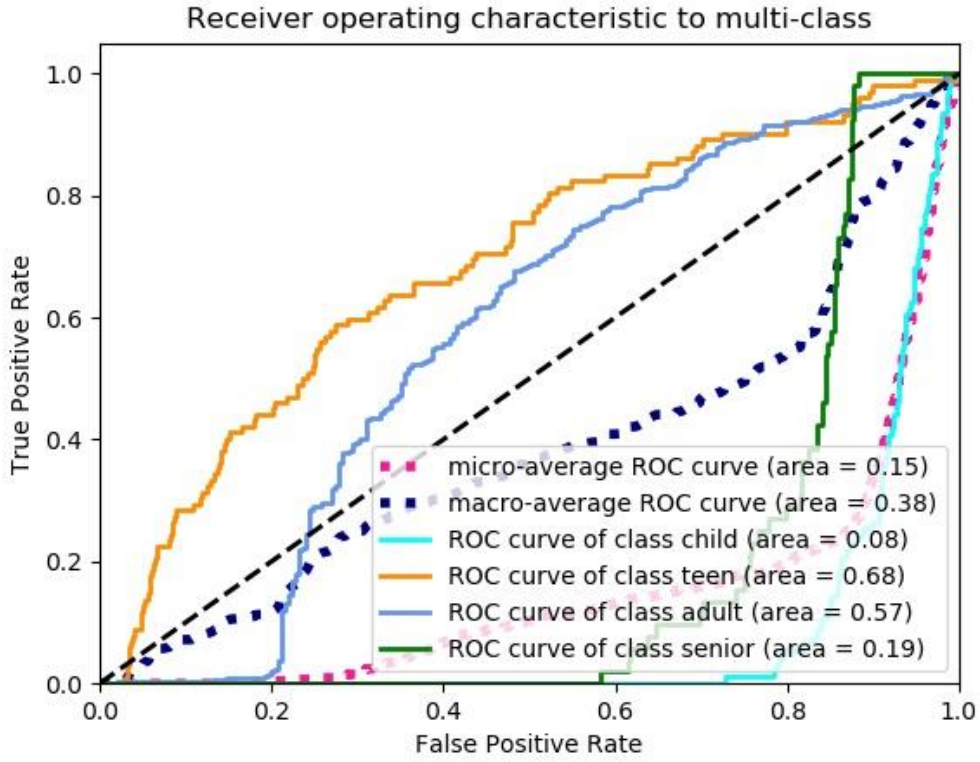


图 17 ROC 曲线 (LBP-PCA-Adaboost)

3.4.3 DenseNet201-CNN

3.4.3.1 实验数据

将原始数据经过 DenseNet201 进行特征提取后，将特征提取得到的结果送往 CNN (Softmax)分类器。由于 TensorFlow 及 Keras 并没有像 Scikit-learn 框架集成分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)等指标，因此需要手动构造函数。评估指标函数编程思路如图 18 所示，评估结果分别如表 20、表 21、表 22 所示。

表 20 Classification Report (DenseNet201-CNN)

method	class/metric	precision	recall	f1-score	support
DenseNet201 -CNN (Softmax)	adult	0.95	0.96	0.96	953
	child	0.8	0.77	0.75	91
	teen	0.84	0.83	0.83	52
	senior	0.65	0.62	0.61	102
	weighted avg	0.9	0.91	0.91	1198

表 21　Confusion Matrix (DenseNet201-CNN)

method	class	adult	child	teen	senior
DenseNet201-CNN (Softmax)	adult	924	4	6	21
	child	5	71	0	12
	teen	9	0	46	0
	senior	33	15	0	52

表 22　Error Matrix (DenseNet201-CNN)

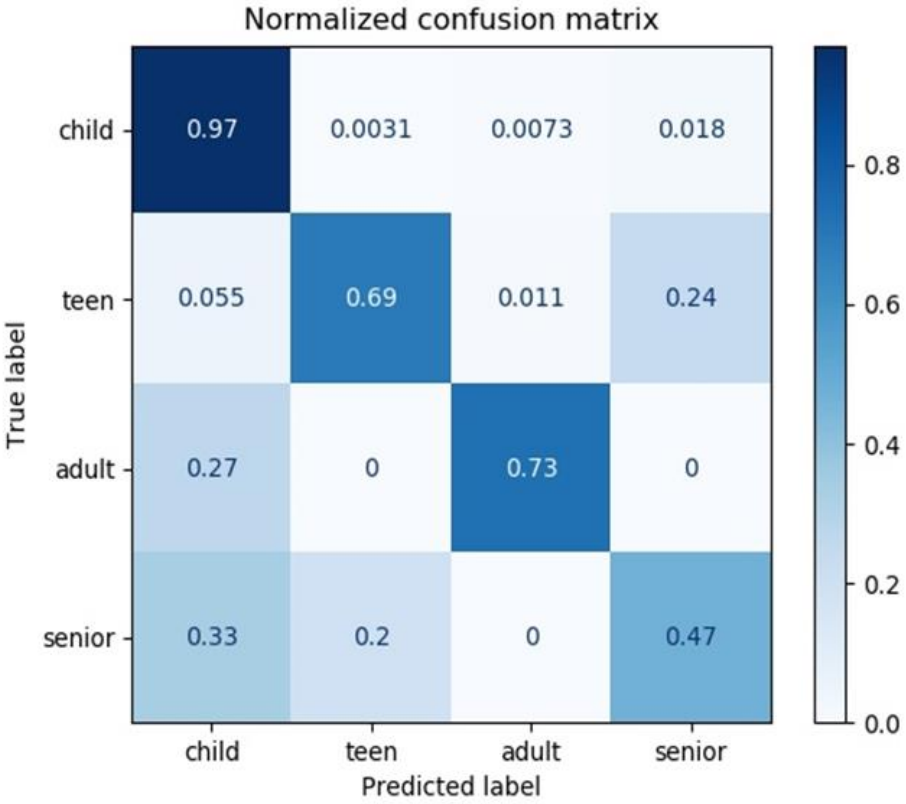
method	class	adult	child	teen	senior
DenseNet201-CNN (Softmax)	adult	0	0.042	0.115	0.183
	child	0.008	0	0.017	0.165
	teen	0.006	0	0	0
	senior	0.028	0.14	0	0

```
def metric_fn(label_ids, predicted_labels):
    accuracy = tf.metrics.accuracy(label_ids, predicted_labels)
    f1_score = tf.contrib.metrics.f1_score(
        label_ids,
        predicted_labels)
    auc = tf.metrics.auc(
        label_ids,
        predicted_labels)
    recall = tf.metrics.recall(
        label_ids,
        predicted_labels)
    precision = tf.metrics.precision(
        label_ids,
        predicted_labels)
    true_pos = tf.metrics.true_positives(
        label_ids,
        predicted_labels)
    true_neg = tf.metrics.true_negatives(
        label_ids,
        predicted_labels)
    false_pos = tf.metrics.false_positives(
        label_ids,
        predicted_labels)
    false_neg = tf.metrics.false_negatives(
        label_ids,
        predicted_labels)
    return {
        'eval_accuracy': accuracy,
        'f1_score': f1_score,
        'auc': auc,
        'precision': precision,
        'recall': recall,
        'true_positives': true_pos,
        'true_negatives': true_neg,
        'false_positives': false_pos,
        'false_negatives': false_neg
    }
```

图 18　TensorFlow 构造评估指标

3.4.3.2　可视化

将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图19 所示。



DenseNet201-CNN

图 19　混淆矩阵 (DenseNet201-CNN)

将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 20 所示。

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。故得到的 ROC 曲线，如图 21 所示。

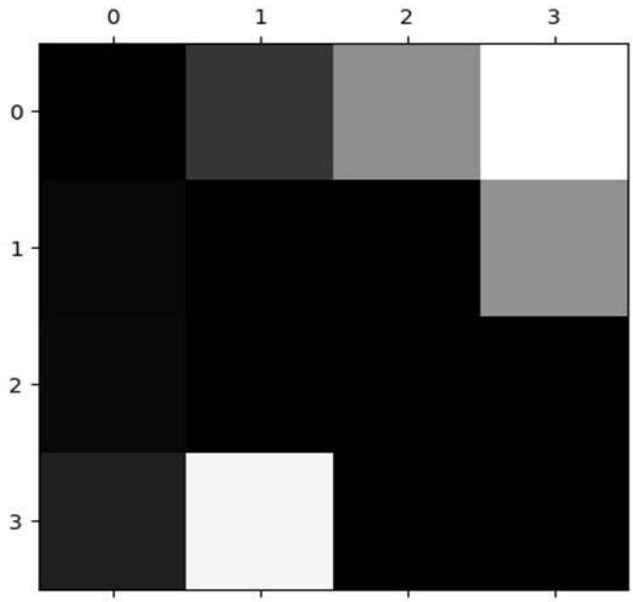


图 20 犯错矩阵 (DenseNet201-CNN)

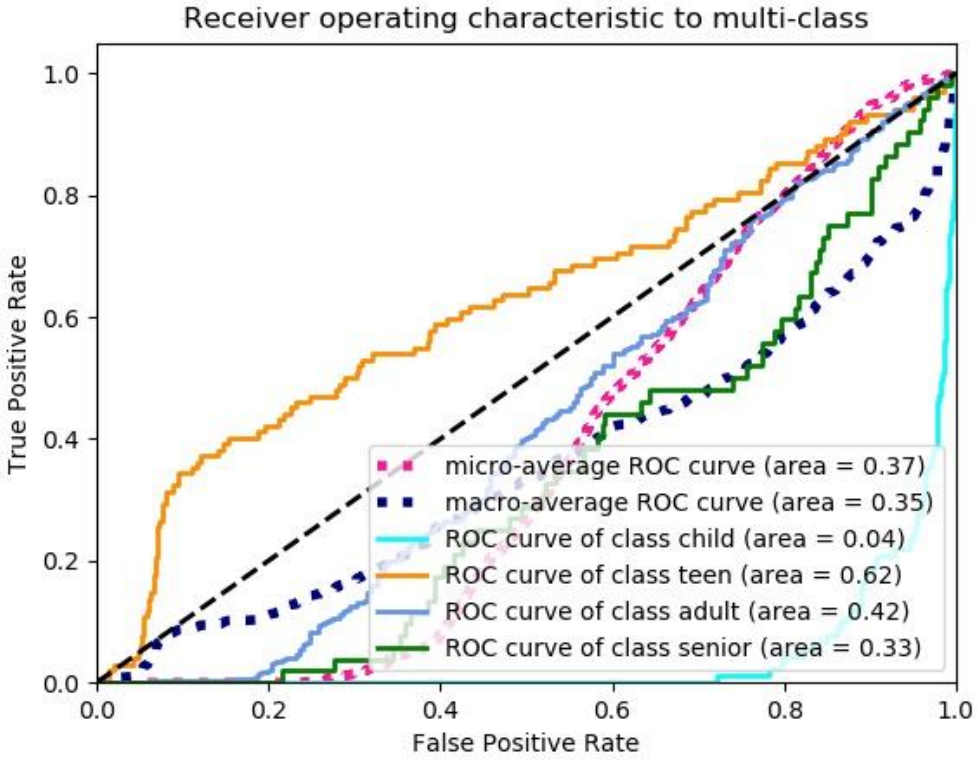


图 21 ROC 曲线 (DenseNet201-CNN)

3.4.4 DenseNet201-KPCA-CNN

3.4.4.1 实验数据

将原始数据经过 DenseNet201 进行特征提取后，将特征提取得到的结果送往 CNN (Softmax)分类器。评估结果分别如表 23、表 24、表 25 所示。

表 23 Classification Report (DenseNet201-KPCA-CNN)

method	class/metric	precision	recall	f1-score	support
DenseNet201 -CNN (Softmax)	adult	0.96	0.95	0.96	953
	child	0.81	0.82	0.79	91
	teen	0.86	0.85	0.85	52
	senior	0.72	0.64	0.63	102
	weighted avg	0.92	0.92	0.92	1198

表 24 Confusion Matrix (DenseNet201- KPCA-CNN)

method	class	adult	child	teen	senior
DenseNet201- CNN (Softmax)	adult	926	5	6	13
	child	9	76	0	9
	teen	6	0	48	0
	senior	20	21	0	59

表 25 Error Matrix (DenseNet201- KPCA-CNN)

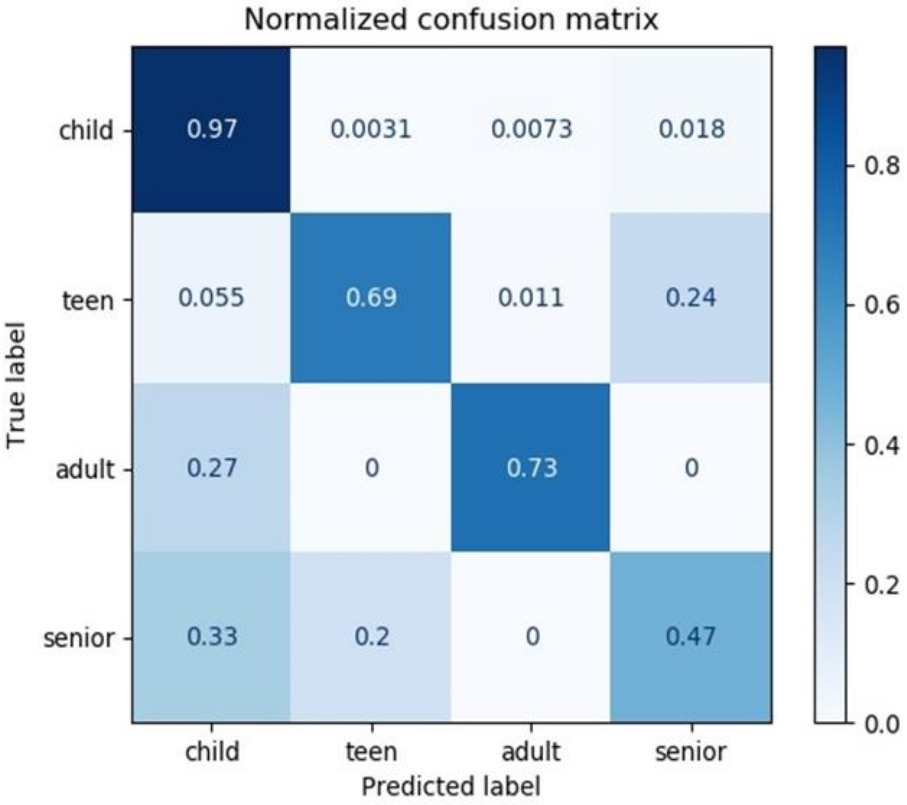
method	class	adult	child	teen	senior
DenseNet201- CNN (Softmax)	adult	0	0.044	0.135	0.127
	child	0.009	0	0	0.156
	teen	0.018	0	0	0
	senior	0.033	0.184	0	0

3.4.4.2 可视化

将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图 22 所示。

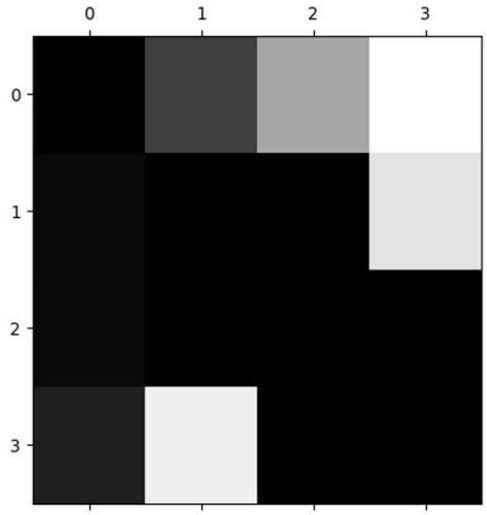
将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 23 所示。

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。故得到的 ROC 曲线，如图 24 所示。



DenseNet201-KPCA-CNN

图 22 混淆矩阵 (DenseNet201-KPCA-CNN)



DenseNet201-KPCA-CNN

图 23 犯错矩阵 (DenseNet201-KPCA-CNN)

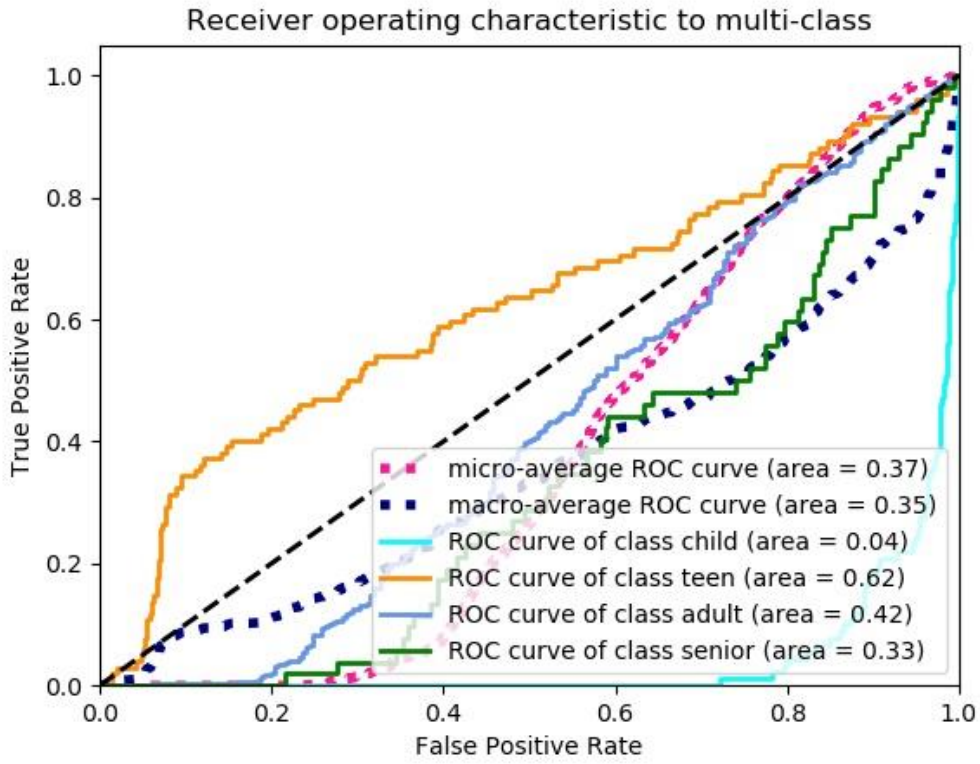


图 24　ROC 曲线 (DenseNet201-KPCA-CNN)

3.5　结果分析

(1) 精确率指模型预测为正的样本中实际也为正的样本占被预测为正的样本的比例。对于精确度 precision_score 指标，通过经典算法 LBP-Adaboost 以及智能算法 DenseNet201-(KPCA)-CNN 所得出的结果至少为 0.9，而 LBP-PCA-Adaboost 所得出的结果为 0.89。因此，添加了特征降维 PCA 环节尽管会使数据维度从较高维度降低到较低维度，但会使经典算法在该指标下表现稍逊于未添加 PCA 模块，而智能算法在有无添加改进特征降维算法 KPCA 前后均表现优秀；

(2) 召回率指实际为正的样本中被预测为正的样本所占实际为正的样本的比例。对于召回率 recall_score 指标，通过经典算法 LBP-(PCA)-Adaboost 以及智能算法 DenseNet201-(KPCA)-CNN 所得出的结果至少为 0.9，故经典、改进多种算法组合在该指标下均取得优秀表现；

(3) f1_score 是精确率和召回率的调和平均值。precision_score 体现了模型对负样本的区分能力，precision_score 越高，模型对负样本的区分能力越强；recall_score 体现了模型对正样本的识别能力，recall 越高，模型对正样本的识别能力越强。f1-score 是两者的综合，f1_score 越高，说明模型越稳健。对于 f1_score 指标，通过经典算法 LBP-(PCA)-Adaboost 以及智能算法 DenseNet201-(KPCA)-CNN 所得出的结果绝大部分至少为 0.9，只有通过 LBP-PCA-Adaboost(OVR)得到的结果为 0.89。故总体来说，经典、改进多种算法组合在该指标下均取得优秀表现；

(4) 混淆矩阵是数据科学、数据分析和机器学习中总结分类模型预测结果的情形分析表，以矩阵形式将数据集中的记录按照真实的类别与分类模型作出的分

类判断两个标准进行汇总。将混淆矩阵转化为误差矩阵，并进行可视化，观察可知，①对于经典算法 LBP-(PCA)-Adaboost 以及智能算法 DenseNet201-(KPCA)-CNN，其混淆矩阵大体类似，最主要的区别在于(adult, senior)以及(child, senior)两个区域，即将 adult、child 两类图片错分为 senior 类；②对于 LBP-Adaboost(OVO)，(adult, senior)区域几乎为白色，(child, senior)区域亮度也非常高，因此该算法在这两类数据错分为 senior 类时犯错率较高；③LBP-Adaboost(OVR)在(child, senior)区域犯错率较低；④LBP-PCA-Adaboost(OVO)与 LBP-PCA-Adaboost(OVR)分别在(child, senior)区域以及(adult, senior)区域进行区分时较容易犯错；⑤DenseNet201-(KPCA)-CNN 在(senior, child)区域，即将 senior 类数据预测为 child 类时犯错率明显较高；

(5) ROC 曲线图是反映敏感性与特异性之间关系的曲线。每一条曲线的所在位置，把曲线的对应区域划分成了两部分，曲线下方部分的面积被称为 AUC(Area Under Curve)，用来表示预测准确性，AUC 值越高，也就是曲线下方面积越大，说明预测准确率越高。曲线越接近左上角(X 越小，Y 越大)，预测准确率越高。观察四个算法组合的 ROC 曲线可知，①LBP-(PCA)-Adaboost 和 DenseNet201-(KPCA)-CNN 方法 AUC 值均在 teen 类上取得最大，在 child 类上取得最小，即预测 teen 类效果最好，预测 child 类效果最差；②微平均 AUC 值均小于宏平均 AUC 值，即设置微平均权重分类准确率低于宏平均权重分类准确率；③对于 teen 类和 adult 类的 AUC 值较为接近，即对于这两类数据的预测效果接近；

(6) 在算法耗时方面，由于本实验所采取模式识别算法均基于机器学习框架以及已经过海量数据训练调优的模型，故对于算法本身的时间复杂度和空间复杂度我们并无法直接进行计算求得。因此，采用完整耗时进行评估。由表 9 可知，经典算法 LBP-Adaboost 与 LBP-PCA-Adaboost 分别耗时 3812.51s 和 797.81s，而智能算法 DenseNet201-CNN 与 DenseNet201-KPCA-CNN 分别耗时 575.62s 和 313.45s。显然，针对本实验数据量并不大的情况下，经典算法在不添加 PCA 进行特征降维时，耗费时间非常漫长，而添加 PCA 模块能够为经典算法缩短 79.07%的时间，提升较高运行速率。而智能算法在耗时长短表现上明显超越经典算法，并且，添加 KPCA 环节能够为智能算法缩短 45.55%的时间，优化效果显著。

4 基于 SVM 的模式识别分类方法

4.1 特征提取算法

4.1.1 经典算法：HOG

4.1.1.1 算法原理

方向梯度直方图(Histogram of Oriented Gradient, HOG)特征是一种在计算机视觉和图像处理中用来进行物体检测的特征描述子。它通过计算和统计图像局部区域的梯度方向直方图来构成特征。其主要思想是：在一副图像中，局部目标的表象和形状(appearance and shape)能够被梯度或边缘的方向密度分布很好地描述。

大致实现方法为：首先，将图像分成小的连通区域，称为细胞单元。接下来，采集细胞单元中各像素点的梯度的或边缘的方向直方图。最后，把这些直方图组合构成特征描述器。把这些局部直方图在图像的更大的范围内(称为区间或 block)进行对比度归一化(contrast-normalized)，该过程所采用的方法是：先计算各直方图在这个区间(block)中的密度，然后根据这个密度对区间中的各个细胞单元做归一化。通过这个归一化后，能对光照变化和阴影获得更好的效果。

与其他的特征描述方法相比，HOG 最显著的优点是，由于 HOG 是在图像的局部方格单元上操作，所以它对图像几何的和光学的形变都能保持很好的不变性，这两种形变只会出现在更大的空间领域上[8]。

具体计算过程如下：

(1) 标准化 gamma 空间和颜色空间

为了减少光照因素的影响，首先需要将整个图像进行规范化(归一化)。在图像的纹理强度中，局部的表层曝光贡献的比重较大，所以，这种压缩处理能够有效地降低图像局部的阴影和光照变化。因为颜色信息作用不大，通常先转化为灰度图；

(2) 计算图像梯度

计算图像横坐标和纵坐标方向的梯度，并据此计算每个像素位置的梯度方向值；求导操作不仅能够捕获轮廓，人影和一些纹理信息，还能进一步弱化光照的影响。

图像中像素点(x,y)的梯度为：

$$G_x(x,y) = H(x+1,y) - H(x-1,y) \tag{16}$$

$$G_y(x,y) = H(x,y+1) - H(x,y-1) \tag{17}$$

其中，$G_x(x,y)$、$G_y(x,y)$与$H(x,y)$分别表示输入图像中像素点(x,y)处的水平方向梯度、垂直方向梯度和像素值。像素点(x,y)处的梯度幅值和梯度方向分别为：

$$G(x,y) = \sqrt{G_x(x,y)^2 + G_y(x,y)^2} \tag{18}$$

$$\alpha(x,y) = tan^{-1}\left(\frac{G_y(x,y)}{G_x(x,y)}\right) \tag{19}$$

(3) 为每个细胞单元构建梯度方向直方图

将图像分成若干个“单元格 cell”，例如每个 cell 为 6*6 个像素。将 cell 的梯度方向 360 度分成 9 个方向块。若该像素的梯度方向是 20-40 度，直方图第 2 个 bin 的计数就加一。因此，对 cell 内每个像素用梯度方向在直方图中进行加权投影(映射到固定的角度范围)，即可得到这个 cell 的梯度方向直方图。

(4) 把细胞单元组合成大的块(block)，块内归一化梯度直方图

由于局部光照的变化以及前景-背景对比度的变化，这使得梯度强度的变化范围非常大。因此，这需要对梯度强度进行归一化，从而进一步对光照、阴影和边缘进行压缩。将各个细胞单元组合成大的、空间上连通的区间(blocks)，故一个 block 内所有 cell 的特征向量串联后便得到该 block 的 HOG 特征。这些区间互相重叠，意味着每一个单元格的特征会以不同的结果多次出现在最后的特征向量中。因此，归一化之后的块描述符(向量)成为 HOG 描述符。

(5) 收集 HOG 特征

将检测窗口中所有重叠的块进行 HOG 特征的收集，结合成最终的特征向量。

4.1.1.2 参数调整与设定

进行 HOG 特征提取时，基于 Scikit-image 框架 feature 模块，调用 hog()函数进行 HOG 算法特征提取。部分参数说明及设定如表 26 所示。

原始数据经过统一转换为矩阵形式后，每张图片的维度为 16384。而经过 HOG 算法进行特征提取后，维度升高到 32448。

调整参数设定，如表 27 所示。在该参数设定下，经过 HOG 算法进行特征提取后，数据维度由原始 16384 降低到 15876。

考虑到后续降维及分类过程中，在较低维度下进行能够加快拟合，因此，HOG 算法最终参数设定如表 27 所示。

表 23 HOG 算法参数设定

参数	中文说明	设定值
image	输入图像(jpg、png 格式或矩阵)	——
orientation	指定 bin 的个数	12
pixels_per_cell	每个 cell 的像素数	(8, 8)
cell_per_block	每个 block 内有多少个 cell	(4, 4)
block_norm	block 内部采用的 norm 类型	L2-Hys
transform_sqrt	是否进行 power law compression，减少阴影和光照变化对图片的影响	True
visualise	是否显示 HOG 图像	False
multichannel	图像是否为三通道	False

表 24　HOG 算法参数调整

参数	中文说明	设定值
image	输入图像(jpg、png 格式或矩阵)	——
orientation	指定 bin 的个数	6
pixels_per_cell	每个 cell 的像素数	(5, 5)
cell_per_block	每个 block 内有多少个 cell	(2, 2)
block_norm	block 内部采用的 norm 类型	L1
transform_sqrt	是否进行 power law compression，减少阴影和光照变化对图片的影响	True
visualise	是否显示 HOG 图像	False
multichannel	图像是否为三通道	False

4.1.2　智能算法：ResNet50

4.1.2.1　算法原理

ResNet50(残差网络)算法原理如下。如图 25 所示，设输入为 x。假设期望理想映射为 $f(x)$，从而作为图 26 上方激活函数的输入。左图虚线框中的部分需要直接拟合出该映射 $f(x)$，而右图虚线框中的部分则需要拟合出有关恒等映射的残差映射 $f(x)-x$。残差映射在实际中往往更容易优化。因此，只需将图 25 中右图虚线框内上方的加权运算(如仿射)的权重和偏差参数学成 0，那么 $f(x)$即为恒等映射。图 25 右图是 ResNet 的基础块，即残差块。在残差块中，输入可通过跨层的数据线路更快地向前传播[9]。

ResNet 沿用了 VGG 全 3×3 卷积层的设计。残差块里首先由 2 个有相同输出通道数的 3×3 卷积层。每个卷积层后接一个批量归一化层和 ReLu 激活函数。然后将输入调过这两个卷积运算后直接加在最后的 ReLu 激活函数前。这样的设计要求两个卷积层的输出与输入形状一样，从而可以相加。如果想改变通道数，就需要引入一个额外的 1×1 卷积层来将输入变换成需要的形状后再做相加运算。ResNet 的前两层结构为：在输出通道数为 64、步幅为 2 的 7×7 卷积层后接步幅为 2 的 3×3 最大池化层。并且，在每个卷积层后增加批量归一化层[10]。ResNet 系列网络结构如图 26 所示。

4.1.2.2　参数调整设定

进行 ResNet 特征提取时，基于 Pytorch 框架，调用 torchvision 模块中 models 子类下已训练并集成完毕的 ResNet50 网络结构进行 ResNet 算法特征提取，并调用 CUDA 进行并行运算。由于数据集中图片为单通道，而 ResNet50 网络需要输入三通道数据，因此需要通过 numpy 模块将二维数据转换为三维，进而将单通道数据变换为三通道。查看源码，得部分参数设定如图 27 所示。

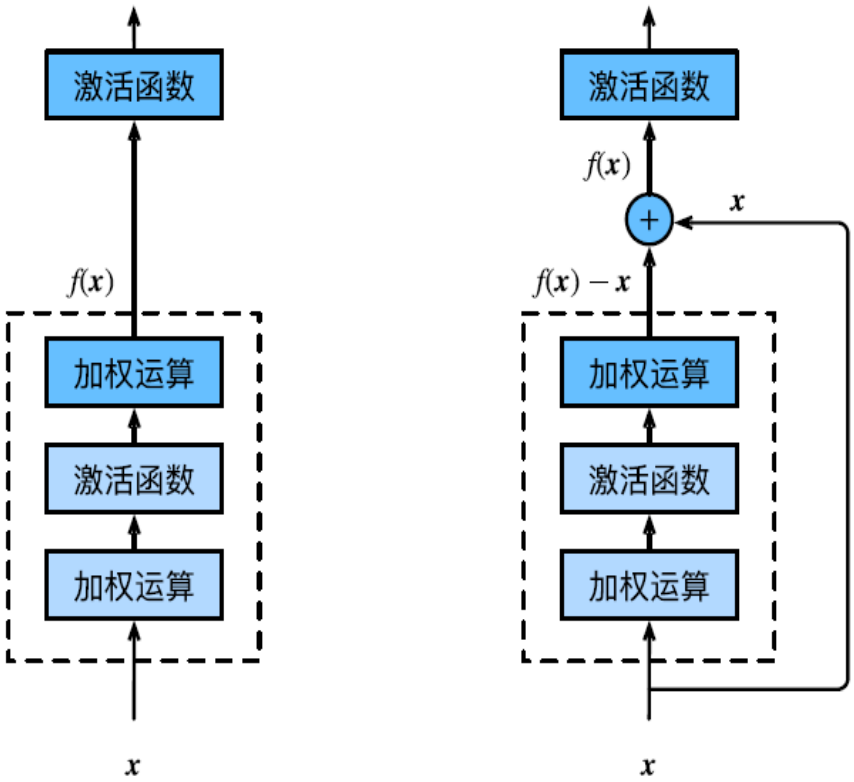


图 25　残差块结构

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

图 26　ResNet 系列网络结构

4.2 特征降维算法

基于 SVM 的模式识别分类方法与本报告 3.2 节采取相同的特征降维算法，即 PCA(经典)以及 KPCA(改进)，故此处不再赘述。

4.3 分类算法

4.3.1 经典算法：SVM

4.3.1.1 算法原理

支持向量机基于线性函数$w^T x + b$，但不同于逻辑回归的是，支持向量机不输出概率，只输出类别。当$w^T x + b$为正时，支持向量机预测属于正类，反之，支持向量机预测属于负类。

支持向量机的一个重要创新是核技巧[11]。核技巧观察到许多机器学习算法都可以写成样本间点积的形式。例如，支持向量机中的线性函数可以重写为：

$$w^T x + b = b + \sum_{i=1}^{m} \alpha_i x^T x^{(i)} \tag{20}$$

其中，$x^{(i)}$是训练样本，α是系数向量。学习算法重写为这种形式允许我们将x替

```python
class ResNet(nn.Module):

    def __init__(self, block, layers, num_classes=1000, zero_init_residual=
False,
                 groups=1, width_per_group=64, replace_stride_with_dilation=
None,
                 norm_layer=None):
        super(ResNet, self).__init__()
        if norm_layer is None:
            norm_layer = nn.BatchNorm2d
        self._norm_layer = norm_layer

        self.inplanes = 64
        self.dilation = 1
        if replace_stride_with_dilation is None:
            # each element in the tuple indicates if we should replace
            # the 2x2 stride with a dilated convolution instead
            replace_stride_with_dilation = [False, False, False]
        if len(replace_stride_with_dilation) != 3:
            raise ValueError("replace_stride_with_dilation should be None "
                             "or a 3-element tuple, got {}"
.format(replace_stride_with_dilation))
        self.groups = groups
        self.base_width = width_per_group
        self.conv1 = nn.Conv2d(3, self.inplanes, kernel_size=7, stride=2,
padding=3,
                               bias=False)
        self.bn1 = norm_layer(self.inplanes)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)
        self.layer1 = self._make_layer(block, 64, layers[0])
        self.layer2 = self._make_layer(block, 128, layers[1], stride=2,
                                       dilate=replace_stride_with_dilation[0])
        self.layer3 = self._make_layer(block, 256, layers[2], stride=2,
                                       dilate=replace_stride_with_dilation[1])
        self.layer4 = self._make_layer(block, 512, layers[3], stride=2,
                                       dilate=replace_stride_with_dilation[2])
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)

        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out', nonlinearity
='relu')
            elif isinstance(m, (nn.BatchNorm2d, nn.GroupNorm)):
                nn.init.constant_(m.weight, 1)
                nn.init.constant_(m.bias, 0)
```

```python
def _forward_impl(self, x):
    # See note [TorchScript super()]
    x = self.conv1(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.maxpool(x)

    x = self.layer1(x)
    x = self.layer2(x)
    x = self.layer3(x)
    x = self.layer4(x)

    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    x = self.fc(x)

    return x

def forward(self, x):
    return self._forward_impl(x)
```

图 27　ResNet50 网络参数

换为特征函数$\varphi(x)$的输出，点积替换为被称为核函数的函数$k(x, x^{(i)}) = \varphi(x) \cdot \varphi(x^{(i)})$。使用核估计替换点积之后，可以使用如下函数进行预测：

$$f(x) = b + \sum_i \alpha_i \, k(x, x^{(i)}) \tag{21}$$

该函数关于x非线性，关于$\varphi(x)$线性，α和$f(x)$之间的关系也是线性的。核函数完全等价于用$\varphi(x)$预处理所有的输入，然后在新的转换空间学习线性模型。

核技巧强大的原因是：其一，它使我们能够使用保证有效收敛的凸优化技术来学习非线性模型。其二，核函数k的实现方法通常比直接构建$\varphi(x)$再算点积高效得多。

4.3.1.2 参数调整与设定

进行 SVM 分类时，基于 Scikit-learn 框架 svm 模块，调用 SVC()类进行 SVM 算法分类。由于 SVM 主要针对二分类问题，因此，针对本实验所探究的多年龄段分类问题，采用 Scikit-learn 框架 multiclass 模块中的 OneVsOneClassifier()类进行“一对一(OVO)”多分类任务学习，与 Scikit-learn 框架 multiclass 模块中的 OneVsRestClassifier()类进行“一对其余(OVR)”多分类任务学习。

调节参数时，借助 Scikit-learn 框架 model_selection 模块，调用 GridSearchCV 函数进行参数网格搜索。同时，设置模型调参评估方法为'precision'，在训练集上做 10 折交叉验证，找到最优参数后对测试集进行分类、预测。设置待调整参数及其对应值程序如图 28 所示，调参结果如图 29 所示。

注：由于测试过程中发现 kernel 设置为‘linear’‘poly’时模型无法收敛，即便设置最大迭代次数 max_iter 为 1000、10000、20000 等数值时仍然无法收敛且预测效果非常差，因此调整参数时 kernel 仅设置为‘rbf’。

由图 29 可知，在训练过程中，寻找到局部最优解时所对应的待调整参数为：C: 1，gamma: 1，kernel: rbf，取得 0.871±0.044 的准确率。故最终 Adaboost 分类器所采用的参数设定如表 25 所示。

表 25　SVM 参数设定

参数	中文说明	设定值
kernel	核函数类型	rbf
C	惩罚系数	1
gamma	核函数系数	0.1
probability	是否按概率输出每种类型	False
tol	停止训练的误差精度	0.001

```python
# 设置gridsearch的参数
    tuned_parameters = [
        {
            'kernel': ['rbf'],
            'C': [0.1, 1, 10],
            'gamma': [0.01, 0.1, 1],
        }
    ]

    # 模型调参评估方法
    score = 'precision'

    svm_clf = GridSearchCV(
        SVC(),
        tuned_parameters,
        cv=10,
        scoring='%s_weighted' % score,
    )

    # 只在训练集上面做k-fold,然后返回最优的模型参数
    svm_clf.fit(X_train, y_train.values.ravel())

    print('Best parameters set found on development set:')
    print(svm_clf.best_params_)
    print('Grid scores on development set:')
    for params, mean_score, scores in svm_clf.grid_scores_:
        print('%0.3f (+/-%0.03f) for %r'
              % (mean_score, scores.std() * 2, params))
```

图 28 待调整参数程序

```
Best parameters set found on development set:
{'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
Grid scores on development set:
0.730 (+/-0.129) for {'C': 0.1, 'gamma': 0.01, 'kernel': 'rbf'}
0.628 (+/-0.007) for {'C': 0.1, 'gamma': 0.1, 'kernel': 'rbf'}
0.628 (+/-0.007) for {'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
0.947 (+/-0.023) for {'C': 1, 'gamma': 0.01, 'kernel': 'rbf'}
0.673 (+/-0.075) for {'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
0.628 (+/-0.007) for {'C': 1, 'gamma': 1, 'kernel': 'rbf'}
0.939 (+/-0.027) for {'C': 10, 'gamma': 0.01, 'kernel': 'rbf'}
0.815 (+/-0.054) for {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
0.628 (+/-0.007) for {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

图 29 调参结果

4.3.2 智能算法：XGBoost

4.3.2.1 算法原理

(1) 正则化学习目标

对于一个给定的有n个样本和m个特征的数据集$D = (x_i, y_i)(|D| = n, x_i \in R^m, y_i \in R)$。一个整体的树模型使用K个累加的函数来预测输出：

$$\hat{y}_i = \varphi(x_i) = \sum_{k=1}^{K} f_k(x_i) \tag{22}$$

其中，$f_k \in F$，$F = f(x) = w_q(x)$ 是CART(回归树)的空间。其中q代表树的结构，可以将每个样本映射到对应的叶节点中，T是树中叶子节点的个数。每个f_k对应于一个独立的树结构q和叶子权重w。不同于决策树，每个回归树在每一个叶节点上包含一个连续分数值，w_i代表第i个结点的分数。$w_q(x)$是对样本x的评分，即模型预测值。对于每个样本，使用多个树中我们将使用多个树中决策规则来将它分类到叶节点中，并且通过累加对应叶子中的分数w来获得最终的预测(每个样本的预测结果就是每棵树预测分数的总和)。最小化下列正则化目标：

$$L(\varphi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{23}$$

其中，$\Omega(f) = \gamma T + \frac{1}{2}\lambda||w||^2$，$L$是一个可微凸损失函数，度量测量值与目标值之间的差。第二项Ω惩罚模型的复杂度(树的复杂度之和)。该项中包含了两个部分，一个是叶子结点的总数，一个是叶子结点得到的L2正则化项。这个额外的正则化项能够平滑每个叶节点的学习权重来避免过拟合。直观地，正则化的目标将倾向于选择采用简单和预测函数的模型。当正则化参数为零时，这个函数就变为传统的Gradient Tree Boosting[12]。

(2) Gradient Tree Boosting

树集合模型以函数作为参数，所以不能直接使用传统的优化方法进行优化。而是采用加法学习方式(Additve training)训练，开始于一个常数预测，每次增加的一个新的函数学习当前的树，找到当前最佳的树模型加入带整体模型中。因此，关键在于学习第 t 棵树，寻找最佳的f_t，增加f_t并最小化目标函数，其中是在第 t 次迭代时样本 i 的预测值：

$$\hat{y}_i^t = \hat{y}_i^{t-1} + f_t(x_i) \tag{24}$$

在损失函数中，g_i和h_i是一阶和二阶梯度统计。这就是 XGBoost 的特点。通过这种近似，可以自行定义一些损失函数(例如，平方损失、逻辑损失)，只要保证二阶可导。将常数项移除得到了第 t 步的简化的目标函数。使用从单个叶结点开始，并迭代地向树添加分支的贪心算法。假设I_L和I_R是拆分后左右节点的实例集。$I = I_L \cup I_R$，拆分之后的损失如下：

$$L_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in I_L} g_i\right)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{\left(\sum_{i\in I_R} g_i\right)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}\right] - \gamma \tag{25}$$

该公式通常被用于评估分割候选集(选择最优分割点)，其中前两项分别是切分后左右子树的的分支之和，第三项是未切分前该父节点的分数值，最后一项是

引入额外的叶节点导致的复杂度。

4.3.2.2 参数调整与设定

进行 XGBoost 分类时，基于 XGBoost 模块，调用 XGBoostClassifier()类进行 XGBoost 算法分类。由于 XGBoost 主要针对二分类问题，因此，针对本实验所探究的多年龄段分类问题，采用 Scikit-learn 框架 multiclass 模块中的 OneVsOneClassifier()类进行“一对一(OVO)”多分类任务学习，与 Scikit-learn 框架 multiclass 模块中的 OneVsRestClassifier()类进行“一对其余(OVR)”多分类任务学习。

调节参数时，借助 Scikit-learn 框架 model_selection 模块，调用 GridSearchCV 函数进行参数网格搜索。同时，设置模型调参评估方法为'precision'，在训练集上做 10 折交叉验证，找到最优参数后对测试集进行分类、预测。设置待调整参数及其对应值程序如图 30 所示，调参结果如图 31 所示。

由图 31 可知，在训练过程中，寻找到局部最优解时所对应的待调整参数为：colsample_bytree: 1，eta: 0.01，lambda: 1，取得 0.935±0.031 的准确率。故最终 XGBoost 分类器所采用的参数设定如表 26 所示。

```
# 设置gridsearch的参数
    tuned_parameters = [
        {
            'eta': [0.01, 0.1, 1],
            'colsample_bytree': [0.1, 0.5, 1],
            'lambda': [0.1, 0.5, 1],
        }
    ]

    # 模型调参评估方法
    score = 'precision'

    xgb_clf = GridSearchCV(
        xgb(),
        tuned_parameters,
        cv=10,
        scoring='%s_weighted' % score,
    )

    # 只在训练集上面做k-fold,然后返回最优的模型参数
    xgb_clf.fit(X_train, y_train.values.ravel())

    print('Best parameters set found on development set:')
    print(xgb_clf.best_params_)
    print('Grid scores on development set:')
    for params, mean_score, scores in xgb_clf.grid_scores_:
        print('%0.3f (+/-%0.03f) for %r'
              % (mean_score, scores.std() * 2, params))
```

图 30 待调整参数程序

```
Best parameters set found on development set:
{'colsample_bytree': 1, 'eta': 0.01, 'lambda': 1}
Grid scores on development set:
0.628 (+/-0.007) for {'colsample_bytree': 0.1, 'eta': 0.01, 'lambda': 0.1}
0.628 (+/-0.007) for {'colsample_bytree': 0.1, 'eta': 0.01, 'lambda': 0.5}
0.628 (+/-0.007) for {'colsample_bytree': 0.1, 'eta': 0.01, 'lambda': 1}
0.740 (+/-0.137) for {'colsample_bytree': 0.1, 'eta': 0.1, 'lambda': 0.1}
0.732 (+/-0.153) for {'colsample_bytree': 0.1, 'eta': 0.1, 'lambda': 0.5}
0.735 (+/-0.112) for {'colsample_bytree': 0.1, 'eta': 0.1, 'lambda': 1}
0.852 (+/-0.037) for {'colsample_bytree': 0.1, 'eta': 1, 'lambda': 0.1}
0.857 (+/-0.038) for {'colsample_bytree': 0.1, 'eta': 1, 'lambda': 0.5}
0.860 (+/-0.035) for {'colsample_bytree': 0.1, 'eta': 1, 'lambda': 1}
0.916 (+/-0.038) for {'colsample_bytree': 0.5, 'eta': 0.01, 'lambda': 0.1}
0.917 (+/-0.044) for {'colsample_bytree': 0.5, 'eta': 0.01, 'lambda': 0.5}
0.914 (+/-0.045) for {'colsample_bytree': 0.5, 'eta': 0.01, 'lambda': 1}
0.933 (+/-0.031) for {'colsample_bytree': 0.5, 'eta': 0.1, 'lambda': 0.1}
0.932 (+/-0.030) for {'colsample_bytree': 0.5, 'eta': 0.1, 'lambda': 0.5}
0.934 (+/-0.033) for {'colsample_bytree': 0.5, 'eta': 0.1, 'lambda': 1}
0.924 (+/-0.031) for {'colsample_bytree': 0.5, 'eta': 1, 'lambda': 0.1}
0.928 (+/-0.030) for {'colsample_bytree': 0.5, 'eta': 1, 'lambda': 0.5}
0.928 (+/-0.037) for {'colsample_bytree': 0.5, 'eta': 1, 'lambda': 1}
0.933 (+/-0.030) for {'colsample_bytree': 1, 'eta': 0.01, 'lambda': 0.1}
0.933 (+/-0.032) for {'colsample_bytree': 1, 'eta': 0.01, 'lambda': 0.5}
0.935 (+/-0.031) for {'colsample_bytree': 1, 'eta': 0.01, 'lambda': 1}
0.934 (+/-0.033) for {'colsample_bytree': 1, 'eta': 0.1, 'lambda': 0.1}
0.933 (+/-0.036) for {'colsample_bytree': 1, 'eta': 0.1, 'lambda': 0.5}
0.934 (+/-0.034) for {'colsample_bytree': 1, 'eta': 0.1, 'lambda': 1}
0.927 (+/-0.035) for {'colsample_bytree': 1, 'eta': 1, 'lambda': 0.1}
0.929 (+/-0.031) for {'colsample_bytree': 1, 'eta': 1, 'lambda': 0.5}
0.927 (+/-0.032) for {'colsample_bytree': 1, 'eta': 1, 'lambda': 1}
```

图 31 调参结果

表 26 XGBoost 参数设定

参数	中文说明	设定值
booster	基分类器	gbtree
silent	静默模式，不输出中间过程	True
eta	学习率，减小每一步权重	0.01
min_child_weight	最小叶子节点样本权重和	1
max_depth	树的最大深度	6
gamma	损失阈值	0
max_delta_step	每个学习器树的最大深度	0
subsample	对于每棵树随机采样的比例	1
colsample_bytree	每棵树随机选取的特征的比例	1
colsample_bylevel	每个层级分裂时子样本的特征所占的比例	1
lambda	权重的 L2 正则化项	1
scale_pos_weight	处理样本不平衡问题	1

4.4 实验结果

4.4.1 HOG-SVM

4.4.1.1 实验数据

将原始数据经过 HOG 进行特征提取后，对 SVM 分类器分别进行 OVO 及 OVR 的封装。再将特征提取得到的结果送往分类器，得到分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)，分

别如表 32、表 33、表 34 所示。

表 32　Classification Report (HOG-SVM)

method	class/metric	precision	recall	f1-score	support
HOG-SVM (OVO)	adult	0.96	0.96	0.96	953
	child	0.82	0.80	0.81	91
	teen	0.88	0.81	0.84	52
	senior	0.65	0.69	0.67	102
	weighted avg	0.92	0.92	0.92	1198
HOG-SVM (OVR)	adult	0.96	0.96	0.96	953
	child	0.86	0.71	0.78	91
	teen	0.62	0.72	0.67	102
	senior	0.62	0.72	0.67	102
	weighted avg	0.92	0.92	0.92	1198

表 33　Confusion Matrix (HOG-SVM)

method	class	adult	child	teen	senior
HOG-SVM (OVO)	adult	919	3	6	25
	child	6	73	0	12
	teen	10	0	42	0
	senior	19	13	0	70
HOG-SVM (OVR)	adult	919	2	6	26
	child	8	65	0	18
	teen	9	0	43	0
	senior	20	9	0	73

表 34　Error Matrix (HOG-SVM)

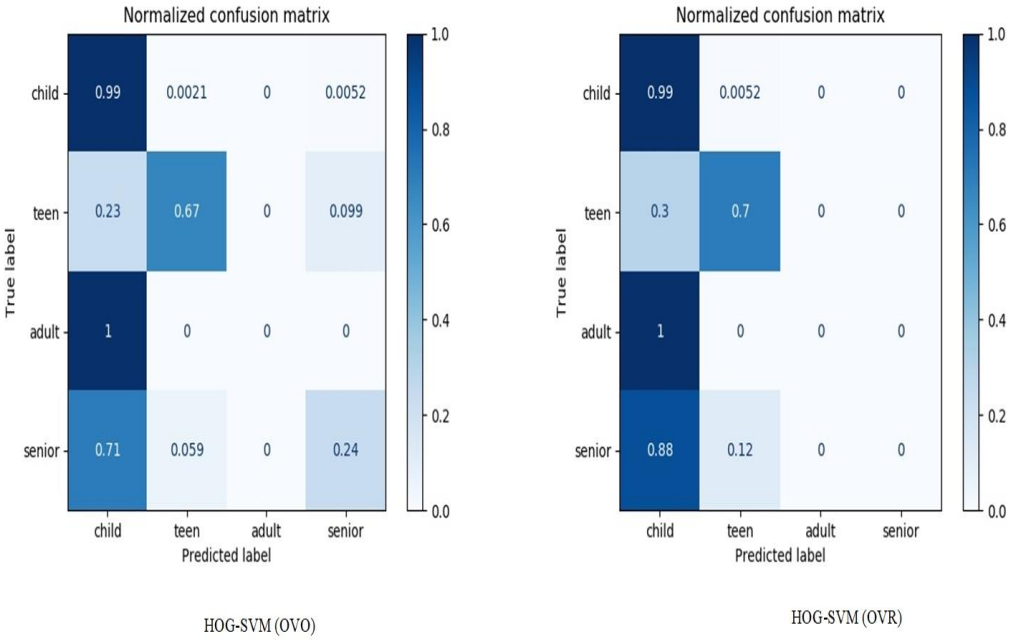
method	class	adult	child	teen	senior
HOG-SVM (OVO)	adult	0	0.033	0.115	0.245
	child	0.006	0	0	0.118
	teen	0.010	0	0	0
	senior	0.020	0.143	0	0
HOG-SVM (OVR)	adult	0	0.022	0.115	0.225
	child	0.008	0	0	0.176
	teen	0.009	0	0	0
	senior	0.021	0.099	0	0

4.4.1.2　可视化

将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图 32 所示。

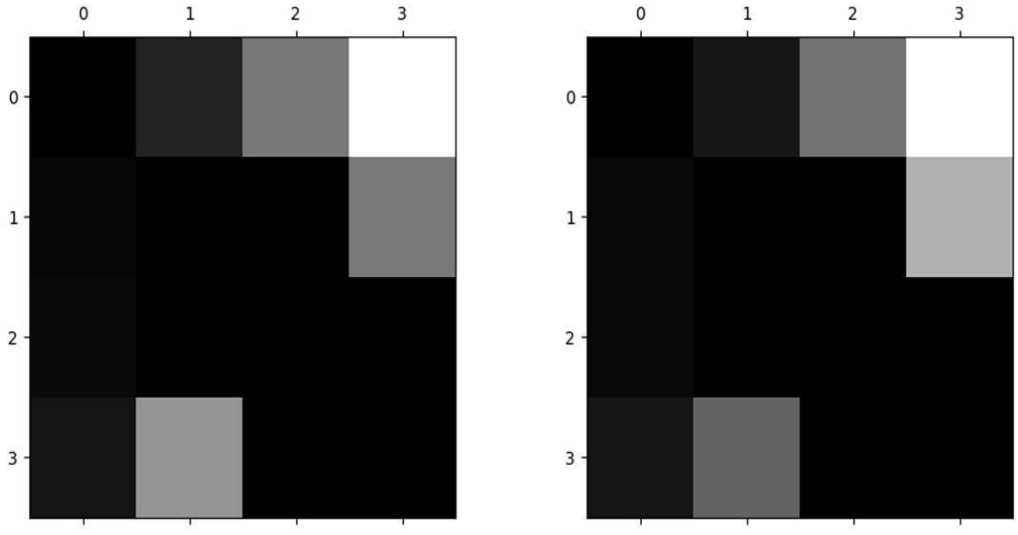
将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 33 所示。

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。其次，Scikit-learn 框架 metrics 模块的 roc_curve()函数仅支持“OVR(一对多)”分类任务，并不支持“OVO(一对一)”分类任务，故此处仅展示 OVR 方法下得到的 ROC 曲线，如图 34 所示。



HOG-SVM (OVO)　　HOG-SVM (OVR)

图 32　混淆矩阵　(HOG-SVM)



HOG-SVM (OVO)　　HOG-SVM (OVR)

图 33　犯错矩阵　(HOG-SVM)

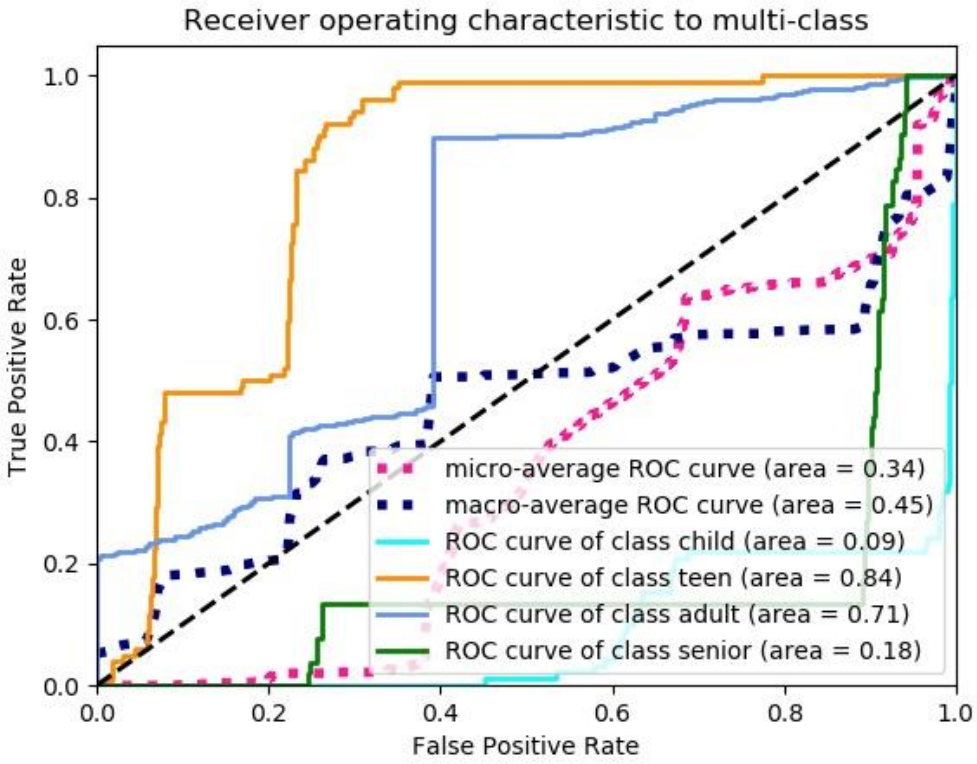


图 34　ROC 曲线 (HOG-SVM)

4.4.2　HOG-PCA-SVM

4.4.2.1　实验数据

将原始数据经过 HOG 进行特征提取后，将所得结果进行 PCA 作特征降维处理，对 SVM 分类器分别进行 OVO 及 OVR 的封装。再将特征降维得到的结果送往分类器，得到分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)，分别如表 35、表 36、表 37 所示。

表 35　Classification Report (HOG-PCA-SVM)

method	class/metric	precision	recall	f1-score	support
HOG-PCA-SVM (OVO)	adult	0.97	0.98	0.97	953
	child	0.86	0.86	0.86	91
	teen	0.88	0.87	0.87	52
	senior	0.74	0.73	0.73	102
	weighted avg	0.94	0.94	0.94	1198
HOG-PCA-SVM (OVR)	adult	0.97	0.97	0.97	953
	child	0.86	0.86	0.86	91
	teen	0.88	0.87	0.87	102
	senior	0.73	0.73	0.73	102
	weighted avg	0.94	0.94	0.94	1198

表 36　Confusion Matrix (HOG-PCA-SVM)

method	class	adult	child	teen	senior
HOG-PCA-SVM (OVO)	adult	930	3	6	14
	child	1	78	0	12
	teen	7	0	45	0
	senior	18	10	0	74
HOG-PCA-SVM (OVR)	adult	929	3	6	15
	child	1	78	0	12
	teen	7	0	45	0
	senior	18	10	0	74

表 37　Error Matrix (HOG-PCA-SVM)

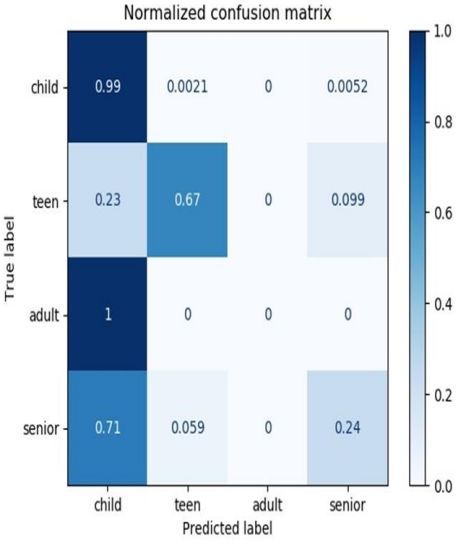
method	class	adult	child	teen	senior
HOG-PCA-SVM (OVO)	adult	0	0.033	0.115	0.137
	child	0.001	0	0	0.118
	teen	0.007	0	0	0
	senior	0.189	0.110	0	0
HOG-PCA-SVM (OVR)	adult	0	0.022	0.115	0.147
	child	0.001	0	0	0.118
	teen	0.007	0	0	0
	senior	0.189	0.110	0	0

4.4.2.2　可视化

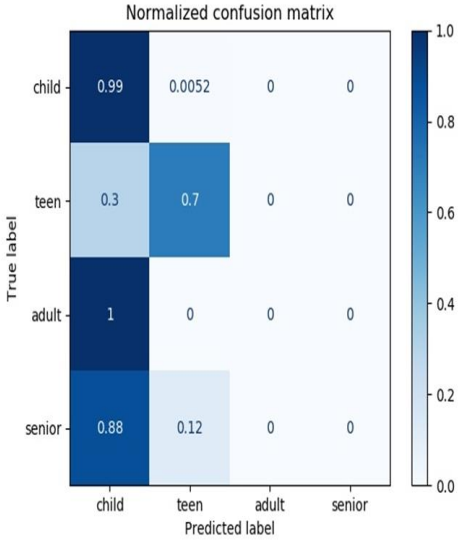
将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图 35 所示。

将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 36 所示。

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。其次，Scikit-learn 框架 metrics 模块的 roc_curve()函数仅支持“OVR(一对多)”分类任务，并不支持“OVO(一对一)”分类任务，故此处仅展示 OVR 方法下得到的 ROC 曲线，如图 37 所示。

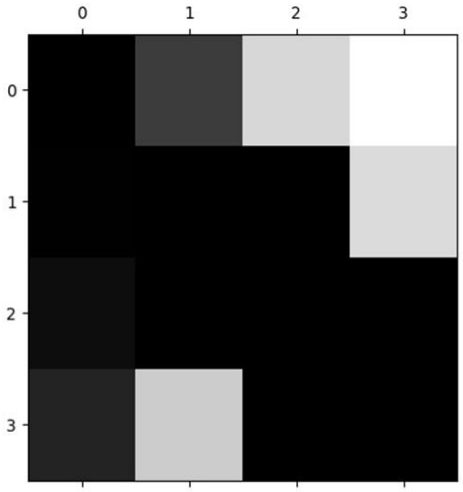


HOG-PCA-SVM (OVO)

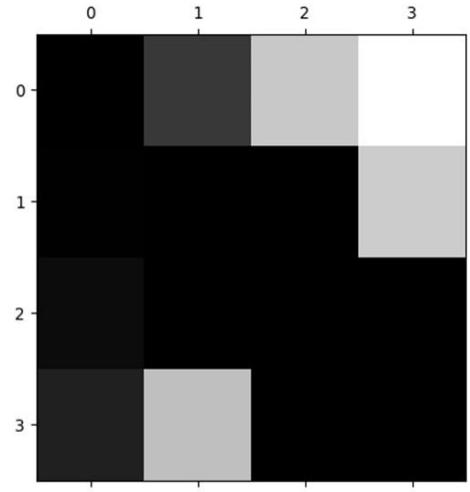


HOG-PCA-SVM (OVR)

图 35 混淆矩阵 (HOG-PCA-SVM)



HOG-PCA-SVM (OVO)



HOG-PCA-SVM (OVR)

图 36 犯错矩阵 (HOG-PCA-SVM)

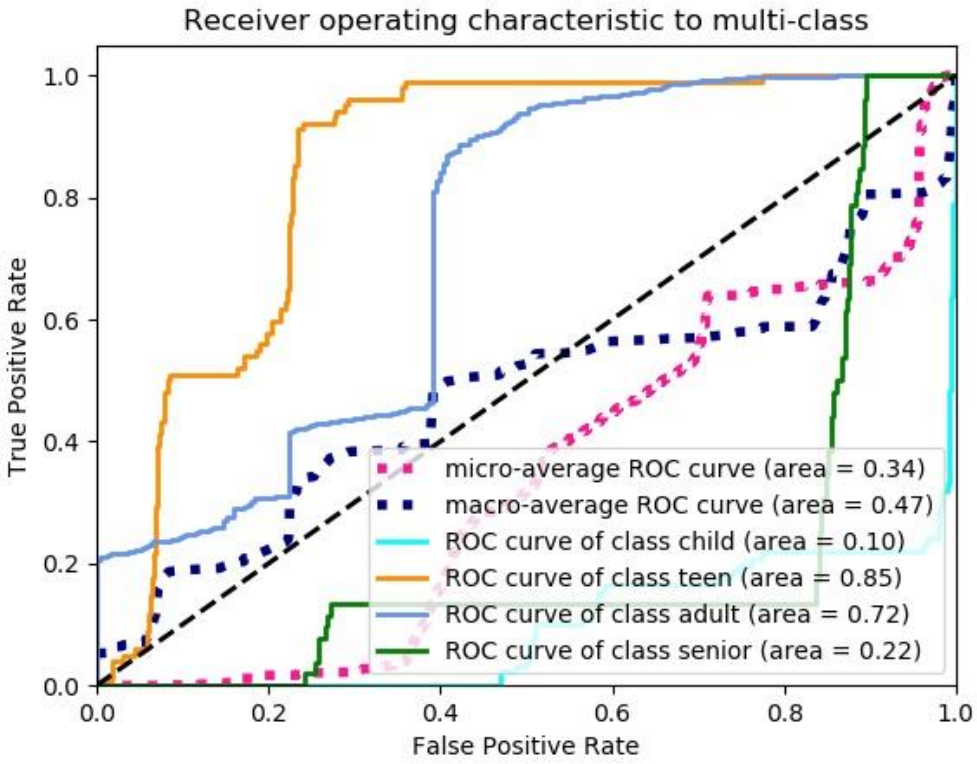


图 37 ROC 曲线 (HOG-PCA-SVM)

4.4.3 ResNet50-XGBoost

4.4.3.1 实验数据

将原始数据经过 ResNet50 进行特征提取后，对 XGBoost 分类器分别进行 OVO 及 OVR 的封装。再将特征提取得到的结果送往分类器，得到分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)，分别如表 38、表 39、表 40 所示。

表 38 Classification Report (ResNet50-XGBoost)

method	class/metric	precision	recall	f1-score	support
ResNet50-XGBoost (OVO)	adult	0.97	0.97	0.97	953
	child	0.80	0.76	0.78	91
	teen	0.87	0.87	0.87	52
	senior	0.65	0.69	0.67	102
	weighted avg	0.92	0.92	0.92	1198
ResNet50-XGBoost (OVR)	adult	0.96	0.97	0.96	953
	child	0.77	0.80	0.78	91
	teen	0.88	0.87	0.87	102
	senior	0.65	0.59	0.62	102
	weighted avg	0.92	0.92	0.92	1198

表 39　Confusion Matrix (ResNet50-XGBoost)

method	class	adult	child	teen	senior
ResNet50-XGBoost (OVO)	adult	922	4	6	21
	child	5	69	1	16
	teen	7	0	45	0
	senior	19	13	0	70
ResNet50-XGBoost (OVR)	adult	922	4	6	21
	child	6	73	0	12
	teen	7	0	45	0
	senior	24	18	0	60

表 40　Error Matrix (ResNet50-XGBoost)

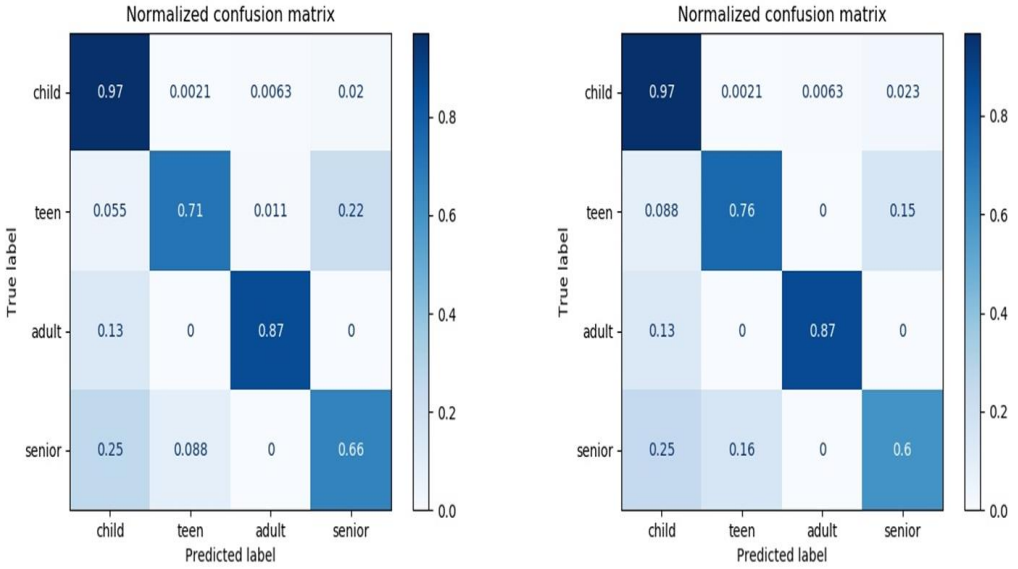
method	class	adult	child	teen	senior
ResNet50-XGBoost (OVO)	adult	0	0.044	0.115	0.206
	child	0.005	0	0.019	0.157
	teen	0.007	0	0	0
	senior	0.020	0.143	0	0
ResNet50-XGBoost (OVR)	adult	0	0.044	0.115	0.206
	child	0.006	0	0	0.118
	teen	0.007	0	0	0
	senior	0.025	0.198	0	0

4.4.3.2　可视化

将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图 38 所示。

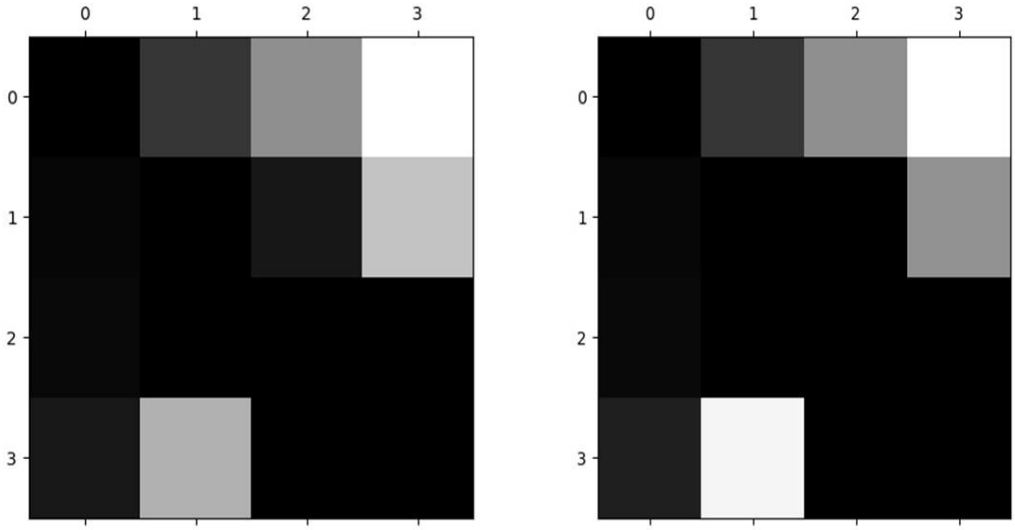
将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 39 所示。

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。其次，Scikit-learn 框架 metrics 模块的 roc_curve()函数仅支持“OVR(一对多)”分类任务，并不支持“OVO(一对一)”分类任务，故此处仅展示 OVR 方法下得到的 ROC 曲线，如图 40 所示。



ResNet50-XGBoost (OVO)　　ResNet50-XGBoost (OVR)

图 38　混淆矩阵　(ResNet50-XGBoost)



ResNet50-XGBoost (OVO)　　ResNet50-XGBoost (OVR)

图 39　犯错矩阵　(ResNet50-XGBoost)

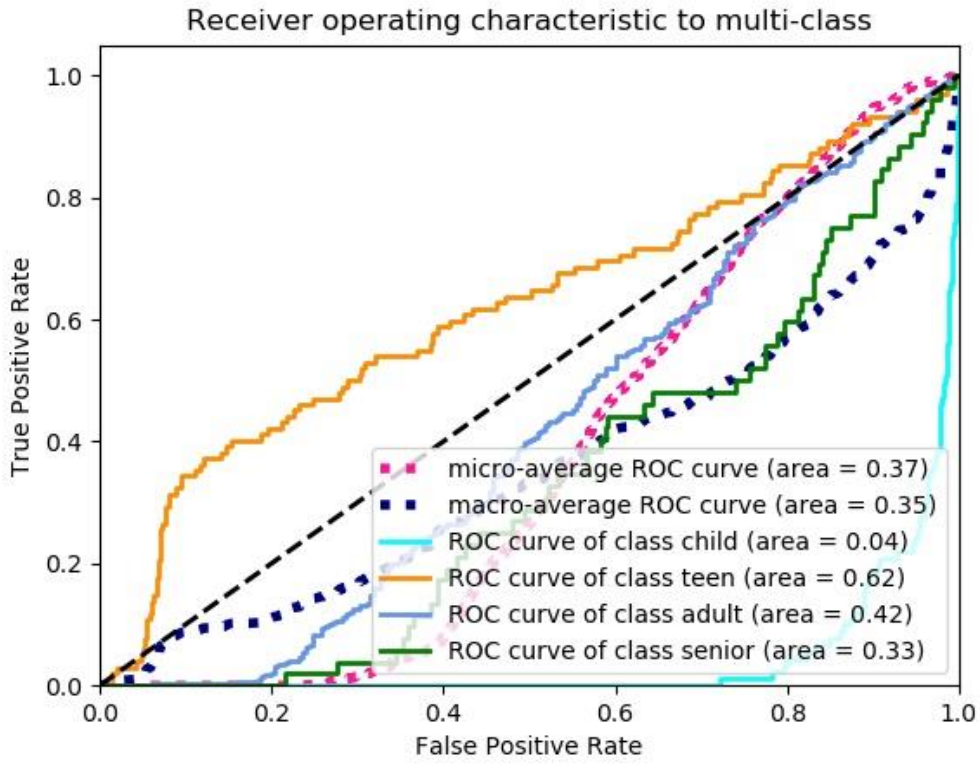


图 40 ROC 曲线 (ResNet50-XGBoost)

4.4.4 ResNet50-KPCA-XGBoost

4.4.4.1 实验数据

将原始数据经过 ResNet50 进行特征提取后，将所得结果进行 KPCA 作特征降维处理，对 XGBoost 分类器分别进行 OVO 及 OVR 的封装。再将特征降维得到的结果送往分类器，得到分类报告(classifcation_report)、混淆矩阵(confusion_matrix)以及犯错矩阵(error_matrix)，分别如表 41、表 42、表 43 所示。

表 41 Classification Report (ResNet50-KPCA-XGBoost)

method	class/metric	precision	recall	f1-score	support
ResNet50-XGBoost (OVO)	adult	0.96	0.97	0.97	953
	child	0.78	0.76	0.77	91
	teen	0.88	0.87	0.87	52
	senior	0.66	0.64	0.65	102
	weighted avg	0.92	0.92	0.92	1198
ResNet50-XGBoost (OVR)	adult	0.96	0.98	0.97	953
	child	0.77	0.80	0.78	91
	teen	0.88	0.87	0.87	102
	senior	0.72	0.60	0.65	102
	weighted avg	0.92	0.93	0.92	1198

表 42　Confusion Matrix (ResNet50- KPCA-XGBoost)

method	class	adult	child	teen	senior
ResNet50-XGBoost (OVO)	adult	925	4	6	18
	child	6	69	0	16
	teen	7	0	45	0
	senior	22	15	0	65
ResNet50-XGBoost (OVR)	adult	930	4	6	13
	child	7	73	0	11
	teen	7	0	45	0
	senior	23	18	0	61

表 43　Error Matrix (ResNet50- KPCA-XGBoost)

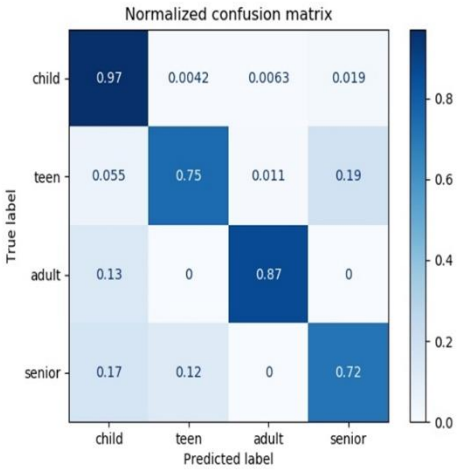
method	class	adult	child	teen	senior
ResNet50-XGBoost (OVO)	adult	0	0.044	0.115	0.176
	child	0.006	0	0	0.157
	teen	0.007	0	0	0
	senior	0.023	0.165	0	0
ResNet50-XGBoost (OVR)	adult	0	0.044	0.115	0.127
	child	0.007	0	0	0.108
	teen	0.007	0	0	0
	senior	0.024	0.198	0	0

4.4.4.2　可视化

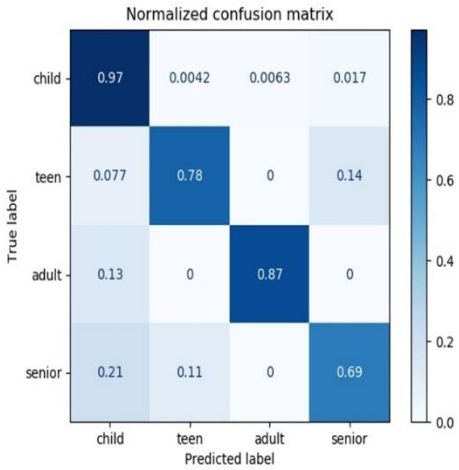
将混淆矩阵中的各个元素通过“标准化”操作压缩至[0, 1]区间，以便观察混淆矩阵中每一类(真假、阴阳)的概率。将“标准化”混淆矩阵进行可视化，如图 41 所示。

将犯错矩阵进行可视化，对于犯错越多的情况，所对应的像素值越高，用灰度图像展示出来表现为亮度越亮，如图 42 所示。

绘制 ROC 曲线时，由于 ROC 曲线仅支持二分类任务。因此，针对本研究的多分类任务，首先需要借用 pandas 模块的 get_dummy()函数，将数据标签转化为 one-hot 编码。其次，Scikit-learn 框架 metrics 模块的 roc_curve()函数仅支持“OVR(一对多)”分类任务，并不支持“OVO(一对一)”分类任务，故此处仅展示 OVR 方法下得到的 ROC 曲线，如图 43 所示。

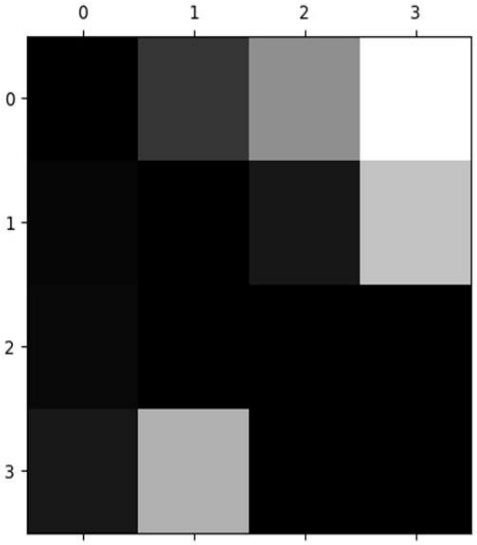


ResNet50-KPCA-XGBoost (OVO)

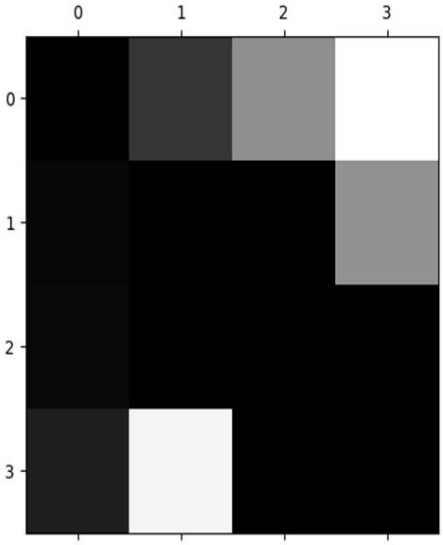


ResNet50-KPCA-XGBoost (OVR)

图 41 混淆矩阵 (ResNet50-KPCA-XGBoost)



ResNet50-KPCA-XGBoost (OVO)



ResNet50-KPCA-XGBoost (OVR)

图 42 犯错矩阵 (ResNet50-KPCA-XGBoost)

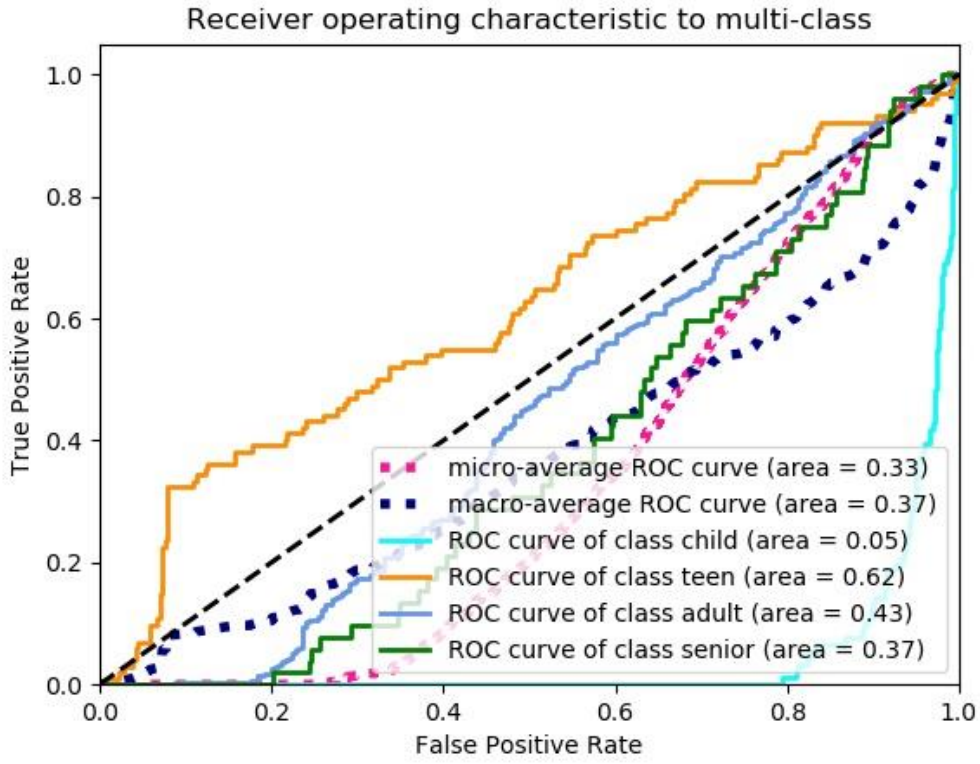


图 43　ROC 曲线(ResNet-KPCA-XGBoost)

4.5　结果分析

(1) 精确率指模型预测为正的样本中实际也为正的样本占被预测为正的样本的比例。对于精确度 precision_score 指标，通过经典算法 HOG-SVM 以及智能算法 ResNet50-(KPCA)-XGBoost 所得出的结果均超过 0.9，且至少为 0.92。此外，HOG-PCA-SVM 取得了精确度最大值 0.94。因此，添加了特征降维 PCA 环节会使数据维度从较高维度降低到较低维度，加快拟合，同时提升经典算法的分类性能，而智能算法在有无添加改进特征降维算法 KPCA 前后均表现优秀；

(2) 召回率指实际为正的样本中被预测为正的样本所占实际为正的样本的比例。对于召回率 recall_score 指标，通过经典算法 HOG-(PCA)-SVM 以及智能算法 ResNet50-(KPCA)-XGBoost 所得出的结果均超过 0.9，且至少为 0.92。此外，HOG-(PCA)-SVM 取得了召回率最大值 0.94。ResNet50-KPCA-XGBoost 取得召回率次大值 0.93。故经典、改进多种算法组合在该指标下均取得优秀表现；

(3) f1_score 是精确率和召回率的调和平均值。precision_score 体现了模型对负样本的区分能力，precision_score 越高，模型对负样本的区分能力越强；recall_score 体现了模型对正样本的识别能力，recall 越高，模型对正样本的识别能力越强。f1-score 是两者的综合，f1_score 越高，说明模型越稳健。对于 f1_score 指标，通过经典算法 HOG-(PCA)-SVM 以及智能算法 ResNet50-(KPCA)-XGBoost 所得出的结果均超过 0.9，且至少为 0.92。故总体来说，经典、改进多种算法组合在该指标下均取得优秀表现；

(4) 混淆矩阵是数据科学、数据分析和机器学习中总结分类模型预测结果的情形分析表，以矩阵形式将数据集中的记录按照真实的类别与分类模型作出的分

类判断两个标准进行汇总。将混淆矩阵转化为误差矩阵，并进行可视化，观察可知，①对于经典算法 HOG-(PCA)-SVM 以及智能算法 ResNet50-(KPCA)- XGBoost，其混淆矩阵大体类似，最主要的区别在于(adult, teen)，(senior, child)，(child, senior)以及(adult, senior)四个区域，即将 adult 类图片错分为 teen 类，将 senior 类错分为 child 类，以及将 adult、child 类图片错分为 senior 类；②除了 ResNet50-KPCA-XGBoost(OVR)方法下(adult, senior)区域亮度较暗，其余方法所得出的犯错矩阵(adult, senior)区域亮度均几乎到达白色。故除 ResNet50-KPCA-XGBoost(OVR)以外的方法较容易将 adult 类图片错分为 senior 类；③ResNet50-KPCA-XGBoost 方法在(senior, child)区域表现较差，有较大的可能性将 senior 类图片错分为 child 类；

(5) ROC 曲线图是反映敏感性与特异性之间关系的曲线。每一条曲线的所在位置，把曲线的对应区域划分成了两部分，曲线下方部分的面积被称为 AUC(Area Under Curve)，用来表示预测准确性，AUC 值越高，也就是曲线下方面积越大，说明预测准确率越高。曲线越接近左上角(X 越小，Y 越大)，预测准确率越高。观察四个算法组合的 ROC 曲线可知，①HOG-(PCA)-SVM 和 ResNet50-(KPCA)-XGBoost 方法 AUC 值均在 teen 类上取得最大，在 child 类上取得最小，即预测 teen 类效果最好，预测 child 类效果最差；②HOG-(PCA)-SVM 方法在 AUC 指标上表现极为优秀，是否添加 PCA 特征降维环节，均取得较大 AUC 值，分别为 0.84 和 0.85，远远大于智能算法，在预测 teen 类上 AUC 值平均高于智能算法 36.29%，在预测 adult 类上 AUC 值远高于智能算法 69.05%；③微平均 AUC 值均小于宏平均 AUC 值，即设置微平均权重分类准确率低于宏平均权重分类准确率；

(6) 在算法耗时方面，由于本实验所采取模式识别算法均基于机器学习框架以及已经过海量数据训练调优的模型，故对于算法本身的时间复杂度和空间复杂度我们并无法直接进行计算求得。因此，采用完整耗时进行评估。由表 9 可知，经典算法 HOG-SVM 与 HOG-PCA-SVM 分别耗时 652.90s 和 513.99s，而智能算法 ResNet50-XGBoost 与 ResNet50-KPCA-XGBoost 分别耗时 265.04s 和 236.71s。显然，经典算法平均耗时长于经典算法一倍左右，但智能算法中有无添加 KPCA 进行特征降维对于缩短分类时间作用较小，添加 KPCA 仅为智能算法运行速度提升 10.69%，约为添加 PCA 环节对经典算法时间性能提升的百分比 21.28%的一半。

5 结论

5.1 工作总结

本小组对所提供的原始数据及其对应标签进行系统性整合，自定义数据接口，重新划分数据集，并进行“特征提取-特征降维-分类”三个步骤，实现了较为完整的模式识别图像有监督分类任务。在初步对比 17 种算法组合之后，本小组最终确定了适合本数据集有监督学习任务的最优算法——经典：LBP-(PCA)-Adaboost、HOG-(PCA)-SVM；智能：DenseNet201-(KPCA)-CNN、ResNet50-(KPCA)-XGBoost，并将上述算法结合有监督分类任务进行功能扩展、性能优化等工作。最终，上述四种算法组合，在分类准确率、精确度等分类结果评估指标上取得明显优于同类算法的成绩。

在上述算法组合中，从分类准确率角度上分析，HOG-(PCA)-SVM 算法组合取得最为优秀的表现，远远优于其他任何一种经典算法与改进算法的组合。因此本小组认为，对于该实验数据集，从分类准确率、精确程度等“硬性指标”上出发，HOG-(PCA)-SVM 算法组合最应该被采纳。即便模式识别分类时间不如其他部分算法组合，但算法耗时方面可以通过提升计算机配置，提高计算机性能，从而缩短耗时。

实验过程中，对于同一套算法，本小组发现不进行特征降维的情况下，即便取得较高的分类准确程度，但计算机运行效率较低、内存开销较大。因此，本小组在特征提取与分类环节之间，分别采用经典与改进的特征降维算法，成功地在不丢失分类精度的前提下，减少图像分类执行过程所耗费的时间，降低运行所消耗的内存。这对于目前大数据背景下的机器学习、数据挖掘等技术至关重要。

通过网络上优秀学生的学习总结得知，“数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。”在本次实验中对比了多种算法组合，并最终确定最优算法组合之后，本小组对这一点更有了进一步的体会与理解。

此外，对于工作过程中的代码管理环节，考虑到对算法进行调试、修改、添加模块等操作，以及完成算法后需要进行版本标记时，都需要一套严格的代码版本控制系统，因此，本小组采用 Git 技术实现整洁且有序的代码管理，对于不同提交记录的程序均能够进行回退查看、标记修改等操作，实现程序的高效梳理。

本小组在进行该实验之前几乎没有接触过图像处理领域技术的情况下，通过不断翻阅资料、浏览优秀代码，以及多次进行语音交流、共享屏幕等，良好地完成了任务，扩充了自身知识储备。在实验过程中，通过一次又一次地发现问题-解决问题-提出新想法-落地实现，本小组在获得较为完备且理想的实验结果的情况下，对当前人工智能发展所带动的机器学习、深度学习等火热技术加深了理解。

5.2 不足之处

与此同时，本小组在该模式识别系统设计流程中还存在一些值得改进的地方。

(1) 由于计算机的算力有限，在调整参数时并没有办法将更多的参数组合纳入考量范围进行调参，因此找到的最优参数组合可能只是局部最优解，并不是全局最优解；

(2) 在进行模式识别系统设计前期，本小组没有进行良好的时间管理，疏于管理工作进度，导致前期准备工作造成了一部分不必要的时间花费；

(3) 由于本小组内部成员所采用的模式识别算法彼此不能重复，而在前期阅

读文献、搜寻适合算法时，小组内部成员没有做到及时且高效的沟通与协调，从而导致在进行算法准备时遇到所选择方法互相重复，需要重新选择算法的情况，造成工作效率较低；

(4) 由于缺乏部分数据结构与算法、计算机底层原理等计算机科学相关基础知识，本小组在进行模式识别系统设计时，尚未做到算法运行效率最优化。本小组成员在工作前期曾经尝试“手写”特征提取程序，但提取效果较差，运行效率低，从而借用机器学习框架进行系统设计；

(5) 此外，本小组采取多种算法进行对比，尽管已经采取优化程序，从而使代码结构清晰、整洁等操作，如将路径封装成类，将方法进行封装、暴露外部接口便于完整执行时调用，但仍存在冗余之处。

因此，针对上述不足之处，本小组成员在日后的升学、工作阶段，会牢记在此次模式识别课程设计中所遇到的问题、汲取的教训，不断扩充自身知识储备，增加编程练习，拓宽国际视野，掌握行业先进算法，树立“终身学习”观念，从而在计算机、互联网等相关行业紧跟前沿技术的脚步，让自己的眼光更为长远，提升自己、完善自己。

6 问题与解决

(1) 问题：多种算法组合在该数据集上只能得到 0.795 的准确率评分，而只有个别算法组合能够突破 0.795，达到 0.85 甚至 0.9 及以上。

解决：测试集中 adult 数量 953 与测试集总量 1198 之比恰好等于 0.795，因此猜想在效果并不是最优的算法组合下，让模式识别系统任意进行预测，都能够预测到概率最大的数据，即预测到 953 张 adult 类别的图片中的任意一张。而想要突破这个限制，就要寻求更优的算法组合。

(2) 问题：HOG 算法即便是封装在了 Scikit-learn 框架里，但并不能像 PCA、KPCA 一样直接输入全部的数据矩阵进行拟合。

解决：循环读入一个个图片矩阵，但是速度很慢。

(3) 问题：进行 HOG、LBP 特征提取后，数据维度仍然很大，送往分类器进行训练、测试的时候，执行效率非常低。

解决：添加 PCA、KPCA 模块降维至 99。

(4) 问题：读取数据矩阵和标签时，标签文件少了第一列索引列，导致标签文件不完整自己矩阵维数丢失。

解决：统一用 pandas 模块读取文件，返回 ndarray 格式的文件。如果不用 ndarray 格式返回，就要增加.values()函数才能输出 pandas 格式的文件。

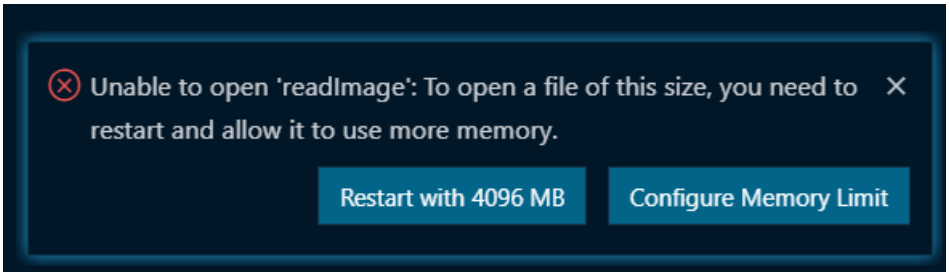
(5) 问题：SVM 用于分类的 SVC()类在 kernel=’linear’时极难收敛，在设置最大迭代数为 20000 时，还是未能收敛，且分类效果非常差。

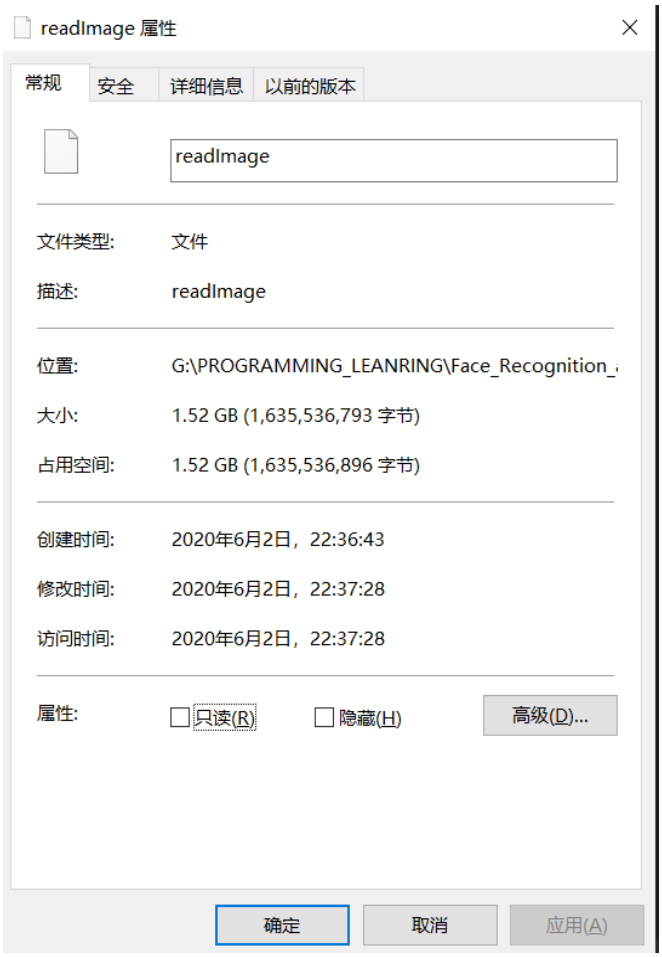
解决：采用 rbf 作为 kernel，运行速度快，分类准确率高。

(6) 问题：过程中经常出现矩阵维度不匹配的问题，导致矩阵连接、运算等操作时经常出错。

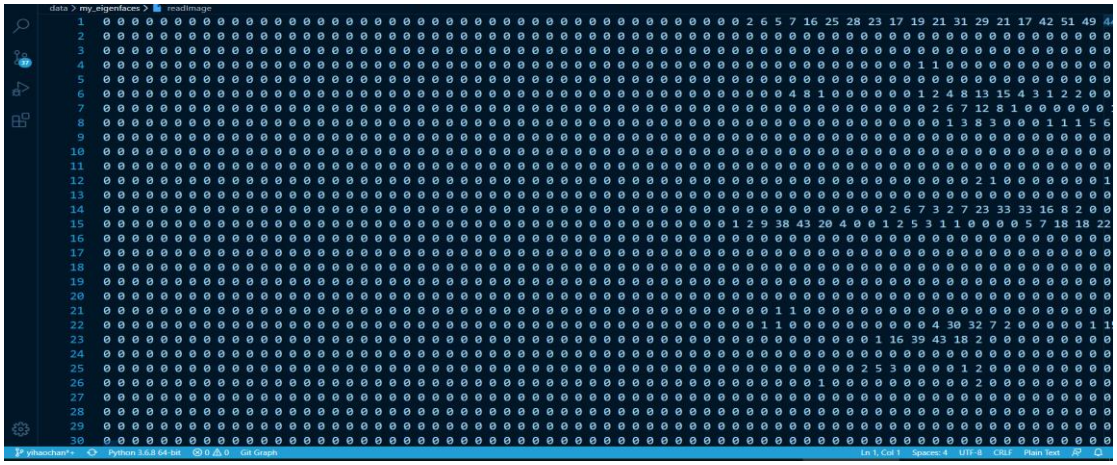
解决：在关键步骤处打印矩阵的 shape，便于调试、改错。

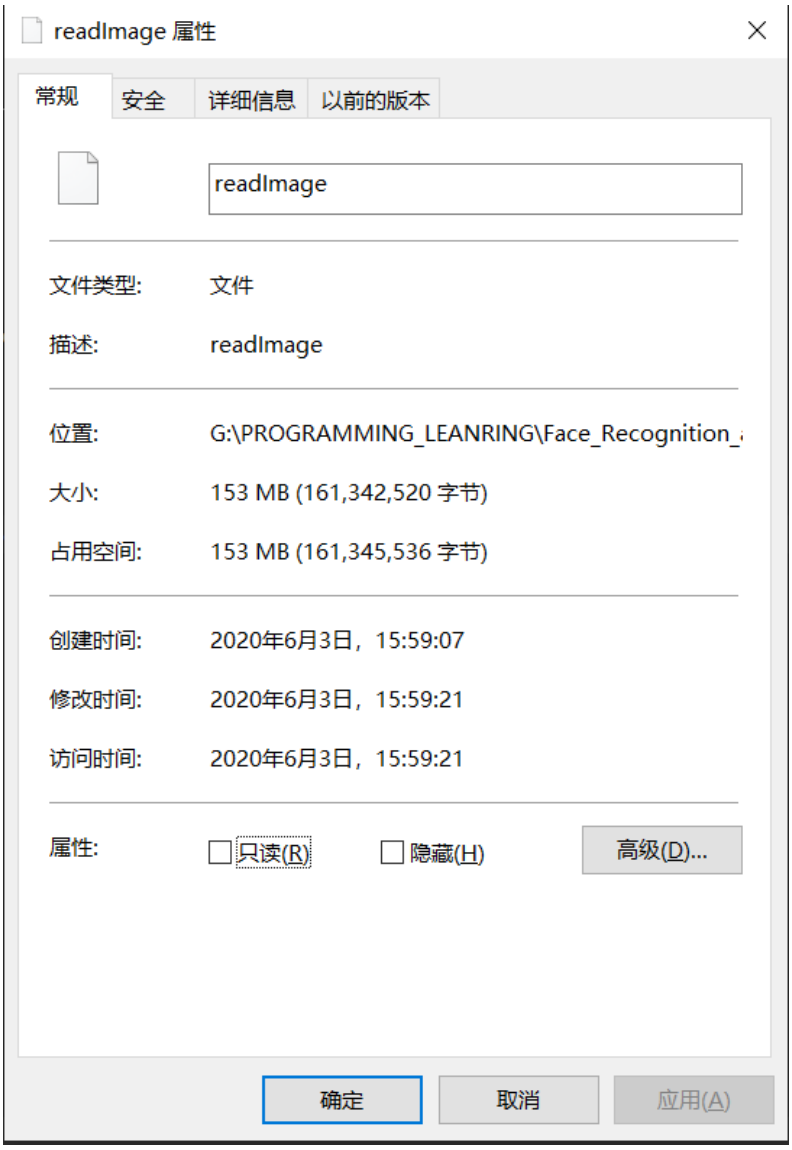
(7) 问题：一开始将 rawdata 转换为矩阵文件时太多庞大，完全打不开，如下所示。





解决：将矩阵各个元素转换为整数形式进行存放，文件大小由 1.52GB 降低至 153MB，结果如下。





(8) 问题：同一套方法，在两位成员不同的电脑上运行出的结果不同。

解决：发现可能是不同版本的机器学习框架和模块采用不同的优化方式，因此统一所有 Python 依赖版本，最终看到相同的结果。

参考文献

[1] 周志华. 机器学习 : = Machine learning[M]. 清华大学出版社, 2016.

[2] Shelton J , Dozier G V , Bryant K S , et al. Genetic based LBP feature extraction and selection for facial recognition[C]// Southeast Regional Conference. ACM, 2011.

[3] Iandola F , Moskewicz M , Karayev S , et al. DenseNet: Implementing Efficient ConvNet Descriptor Pyramids[J]. Eprint Arxiv, 2014.

[4] Shen H , Huang J Z . Sparse principal component analysis via regularized low rank matrix approximation[J]. Journal of Multivariate Analysis, 2008, 99(6):1015-1034.

[5] Kim, Kwang In, Jung, Keechul, Kim, Hang Joon. Face recognition using kernel principal component analysis[J]. ieee signal processing letters, 2002, 9(2):40-42.

[6] Song E , Huang D , Ma G , et al. Semi-supervised multi-class Adaboost by exploiting unlabeled data[J]. Expert Systems with Applications, 2011, 38(6):6720-6726.

[7] Shin H C , Roth H R , Gao M , et al. Deep Convolutional Neural Networks for Computer-Aided Detection: CNN, Architectures, Dataset Characteristics and Transfer Learning[J]. IEEE Transactions on Medical Imaging, 2016, 35(5):1285-1298.

[8] Mizuno K , Terachi Y , Takagi K , et al. Architectural Study of HOG Feature Extraction Processor for Real-Time Object Detection[C]// Signal Processing Systems (SiPS), 2012 IEEE Workshop on. IEEE, 2012.

[9] 阿斯顿·张，李沐，[美] 扎卡里·C. 立顿，[德] 亚历山大·J. 斯莫拉. 动手学深度学习[M]. 人民邮电出版社，2019.

[10] Reddy A S B , Juliet D S . Transfer Learning with ResNet-50 for Malaria Cell-Image Classification[C]// 2019 International Conference on Communication and Signal Processing (ICCSP). 2019.

[11] Reddy A S B , Juliet D S . Transfer Learning with ResNet-50 for Malaria Cell-Image Classification[C]// 2019 International Conference on Communication and Signal Processing (ICCSP). 2019.

[12] Mavroforakis M E , Theodoridis S . A geometric approach to Support Vector Machine (SVM) classification[J]. IEEE Transactions on Neural Networks, 2006, 17(3):p.671-682.

[13] Chen T , Guestrin C . XGBoost: A Scalable Tree Boosting System[J]. 2016.

附录

表 39 成员分工及贡献度自评

姓名	个人分工	贡献度
廖睿翔	课程报告第1、2、3、4节	50%
陈漪皓	课程报告第1、2、3、5节	50%

完整程序仓库地址：
https://github.com/YihaoChan/Pattern_Recognition_Assignment_Face_Supervised_Classification_Age
https://github.com/LuckyXixi/Face_Recognition_Age_public