基于深度学习的人脸年龄预测

Deep learning-based face age prediction

Modern methods of AI

小组成员：封兆欣、颜昊、曹英凡

于濠铭、崔磊、熊锐成

任务目录

TASK　CATALOG

1

实验任务与数据处理

人脸年龄预测

基于人脸图像的年龄预测是指机器根据面部图像推测出人的大概年龄或所属的年龄范围(年龄段)。该系统一般分为人脸检测与定位，年龄特征提取，年龄估计，系统性能评价几个部分。根据提取特征方式的不同又分为传统方法和深度学习方法

传统的方法自然是手动提取特征进行估计，这种方法通常预测不准确并且工作量大，较为困难。

基于深度学习的人脸年龄预测通过向CNN输入海量人脸图像，自动学习各种面部特征。

Adience数据集数据集来源为Flickr相册，共26k张图片，目标是正确预测照片中特定人物的年龄，将年龄划分为了8个区间：(0-2, 4-6, 8-13, 15-20, 25-32, 38-43, 48-53, 60-)

数据的预处理

- aligned
- faces
- Folds

在完成任务之前，需要对数据集进行预处理，给出的数据集已经完成的步骤包括：人脸检测，人脸特征点检测，基于人脸特征点的对齐。

我们的选择？为什么？

代码

```
class Age_train_set(data.Dataset):
    def __init__(self, age_train,transforms=None):
        self.aligned_path = './data/aligned/aligned'
        self.transforms = transform
        # self.imgs = [] #包含所有图片文件路径
        f=open(age_train,'r',encoding='utf8')
        #  '7890646@N03/landmark_aligned_face.1391.11079646666_04f28d301b_o.jpg 0\n',
        self.imgs=f.readlines()
        f.close()
    def __getitem__(self, index):
        img_path = os.path.join(self.aligned_path,self.imgs[index].strip()[:-2])
        age_class=self.imgs[index].strip()[-1]
        # [0, 0, 0, 1, 0, 0, 0, 0]
        #label=[0 if i !=int(age_class) else 1  for i in range(8)]
        #label=torch.FloatTensor(label)
        label = int(age_class)
        data = Image.open(img_path)
        # transform方法很规则
        if self.transforms:
            data = self.transforms(data)
        return data, label
    def __len__(self):
        return len(self.imgs)
```

10069023@N00/landmark_aligned_face.1924.10335948845_0d22490234_o.jpg 5

定义Age_train_set、Age_val_set读入训练集和验证集，经过对文本的strip，返回数据和其对应的标签

```
transform = T.Compose([
    T.ToTensor(),  # 将图片(Image)转成Tensor，归一化至[0, 1]
    T.Normalize(mean=[.5, .5, .5], std=[.5, .5, .5])  # 标准化至[-1, 1]，规定均值和标准差
])
```

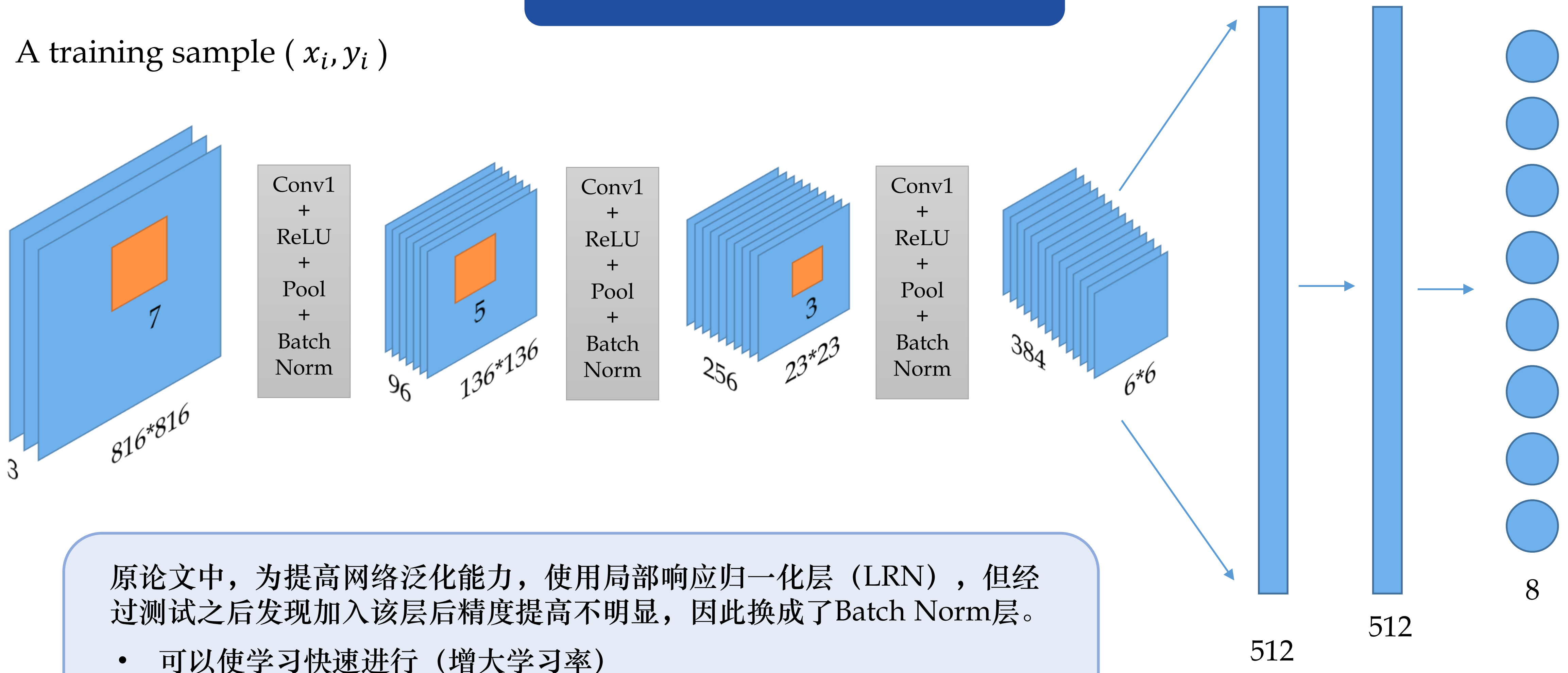
对输入数据的类型转换和标准化处理

2

网络结构

网络结构

A training sample (x_i, y_i)



原论文中，为提高网络泛化能力，使用局部响应归一化层（LRN），但经过测试之后发现加入该层后精度提高不明显，因此换成了Batch Norm层。

- 可以使学习快速进行（增大学习率）
- 不那么依赖初始值（对于初始值不用那么神经质）
- 抑制过拟合（降低Dropout的必要性）

代码及参数

```
self.conv1 = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=3, out_channels=96, kernel_size=7, stride=3, padding=2),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2,stride=2),
    torch.nn.BatchNorm2d(96)
)

self.conv2 = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=96, out_channels=256,kernel_size=5, stride=3, padding=2),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2,stride=2),
    torch.nn.BatchNorm2d(256),
)

self.conv3 = torch.nn.Sequential(
    torch.nn.Conv2d(in_channels=256, out_channels=384,kernel_size=3, stride=2, padding=1),
    torch.nn.ReLU(),
    torch.nn.MaxPool2d(kernel_size=2,stride=2),
    torch.nn.BatchNorm2d(384)

)
```

第一层卷积层：
卷积核7*7，步长为3，padding为2

第二层卷积层：
卷积核5*5，步长为3，padding为2

第三层卷积层：
卷积核3*3，步长为2，padding为1

代码及参数

```
self.fc1 = torch.nn.Sequential(
    torch.nn.Flatten(),
    torch.nn.Linear(6*6*384,512),
    torch.nn.ReLU(),
    #torch.nn.Dropout()
)

self.fc2 = torch.nn.Sequential(
    torch.nn.Linear(512, 512),
    torch.nn.ReLU(),
    #torch.nn.Dropout()
)

self.fc3 = torch.nn.Sequential(
    torch.nn.Linear(512, 8),
    torch.nn.Softmax(dim=1)
)
```

第四层全连接层：
512个神经元

第五层全连接层：
512个神经元

第六层输出层：
使用Softmax函数
8个神经元

3

准确率与误差评估

交叉熵

交叉熵主要是用来判定实际的输出与期望的输出的接近程度，它主要刻画的是实际输出（概率）与期望输出（概率）的距离，也就是交叉熵的值越小，两个概率分布就越接近。

```
criterion=nn.CrossEntropyLoss()
```

```
loss = criterion(probs, target)
```

$$\text{loss}(x, class) = \text{weight}[class]\left(-x[class] + \log\left(\sum_j \exp(x[j])\right)\right)$$

准确率

```
def accuracy(probs,target):
    correct = 0
    probs = probs.tolist()
    target = target.tolist()
    for i in range(len(probs)):
        if probs[i].index(max(probs[i])) == target[i]:
            correct += 1
    accuracy = correct/batch_size

    return accuracy
```

主要分为exact和one-off两类评估方式。

```
pred=model(face)

pred_age=torch.argmax(pred,1).item()

matrix[int(age_class), pred_age] = matrix[int(age_class), pred_age] + 1

if pred_age == int(age_class):
#if abs(pred_age - int(age_class))<=1:
    #print('预测正确   预测分类' + str(pred_age) + ' ; 真实分类' + str(age_class))
    correct+=1

else:
    #print('预测失败   预测分类' + str(pred_age) + ' ; 真实分类' + str(age_class))
    wrong+=1
```

4

训练过程

环境配置&编程框架

torch -- 1.8.1+cu111
pillow -- 8.4.0
torchvision -- 0.9.1+cu111
numpy -- 1.18.5
matplotlib -- 3.3.4

Project Interpreter -- Python 3.8 (pytorch)
IDE -- PyCharm

```
In[14]: torch.cuda.get_device_name(0)
Out[14]: 'NVIDIA GeForce GTX 1650'
```

```
net = Age_CNN()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
net.to(device)
```

参数设置&优化器

batch_size = 16
epochs = 25
learning_rate = 0.01

```
# Loss function
criterion=nn.CrossEntropyLoss()

#optimizer = optim.Adam(net.parameters(), lr=learning_rate)
optimizer=optim.SGD(net.parameters(), lr=learning_rate)
```

```
# 划分训练集、验证集
fold_path = './data/Folds/Folds/train_val_txt_files_per_fold/test_fold_is_0'
# aligned_path = './data/aligned/aligned'

age_train = os.path.join(fold_path, 'age_train.txt')
age_val = os.path.join(fold_path, 'age_val.txt')
# age_test = os.path.join(fold_path, 'age_test.txt')

train_set = Age_train_set(age_train, transforms=Age_transform)
val_set = Age_val_set(age_val, transforms=Age_transform)

train_loader=DataLoader(train_set, batch_size=batch_size, shuffle=True, num_workers=2, drop_last=False)
val_loader=DataLoader(val_set, batch_size=batch_size, shuffle=True, num_workers=2, drop_last=False)
```

Train - Val

```python
if __name__ == '__main__':

    f_train=open('train.txt','w',encoding='utf8',buffering=1)
    f_val=open('val.txt','w',encoding='utf8',buffering=1)

    for epoch in range(epochs):
        net.train()
        print("正在进行第{}轮训练:".format(epoch + 1))

        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(device), target.to(device)
            probs = net(data)
            loss = criterion(probs, target)

            train_batch_accuracy = accuracy(probs, target)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            print("batch:{} loss={}".format(batch_idx,loss))

            f_train.write(str(batch_idx)+'     '+'loss='+str(loss.item())+'    '+
                          'accuracy='+str(train_batch_accuracy)+'\n')
```

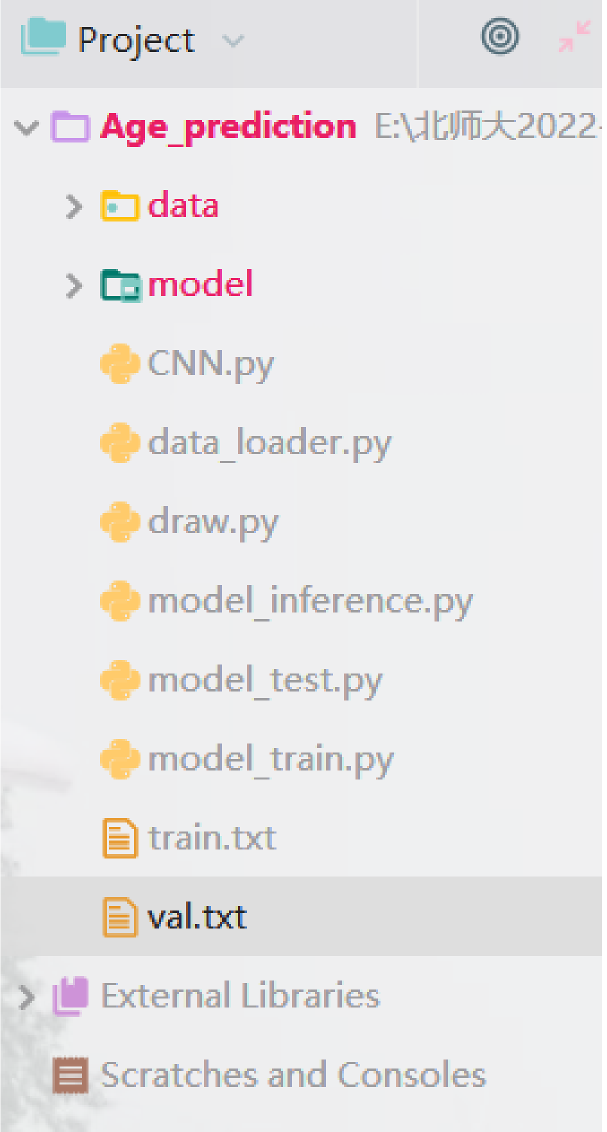
```python
for epoch in range(epochs):
    net.train()
    print("正在进行第{}轮训练:".format(epoch + 1))

    for batch_idx, (data, target) in enumerate(train_loader):...

    net.eval()
    val_accuracy = []
    val_loss=[]

    print('----------val----------')

    for (data, target) in val_loader:...
    mean_var_loss=np.mean(np.array(val_loss))
    mean_val_accuracy=np.mean(np.array(val_accuracy))
    print("loss={:.7f} accuracy={}".format(mean_var_loss, mean_val_accuracy))
    f_val.write(str(epoch) + '     ' + 'loss=' + str(mean_var_loss) + '     ' +
                'accuracy=' + str(mean_val_accuracy) + '\n')

    torch.save(net, './model/net_epoch{}.pth'.format(epoch))

f_train.close()
f_val.close()

torch.save(net,"./model/final_model.pth")
```

Val Record

val.txt - 记事本

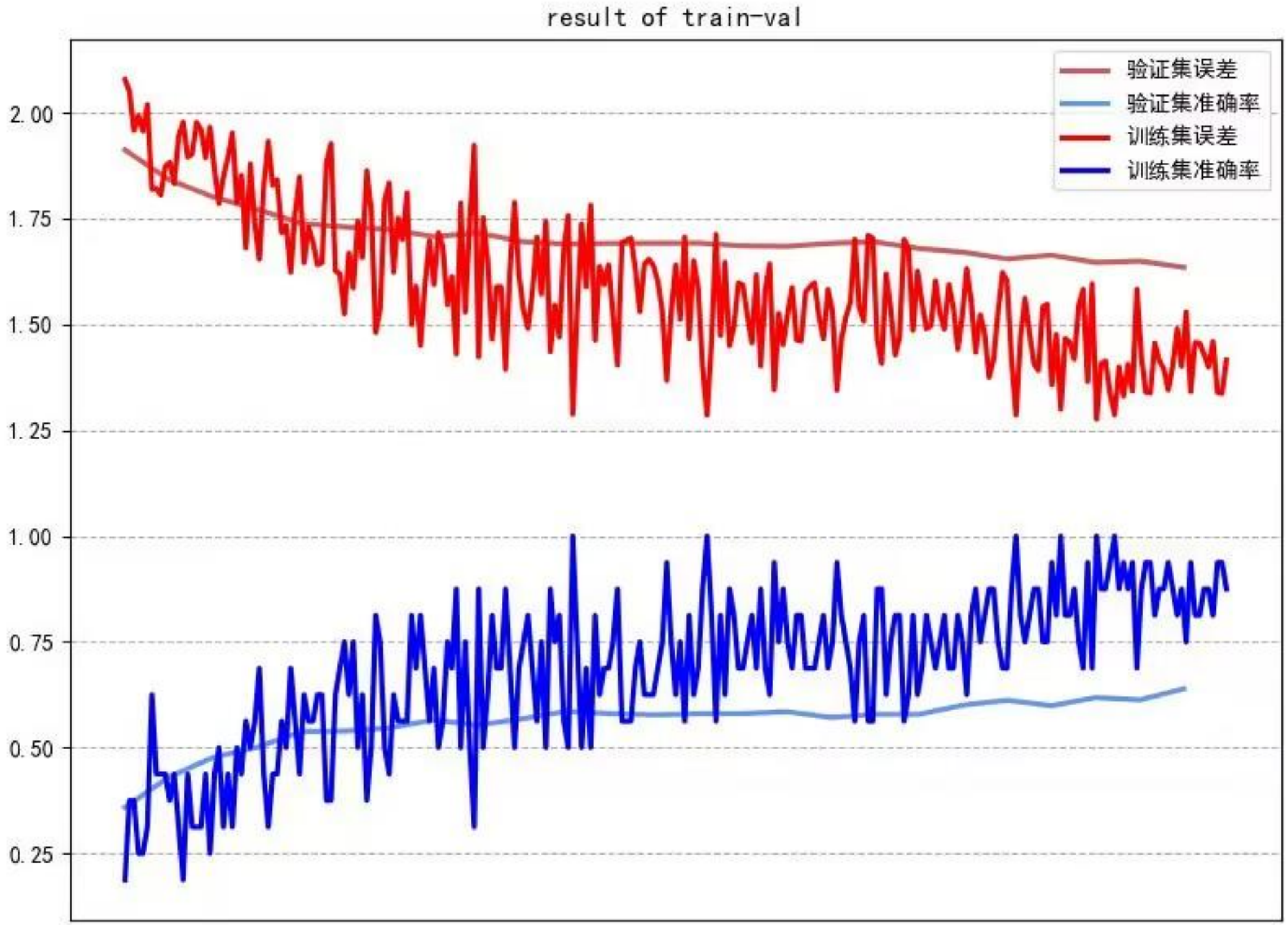
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

0	loss=1.912176270543793	accuracy=0.3587962962962963
1	loss=1.841764720869653	accuracy=0.4305555555555556
2	loss=1.8011470606297622	accuracy=0.47685185185185186
3	loss=1.7721086460867046	accuracy=0.5007716049382716
4	loss=1.7386572861377103	accuracy=0.5378086419753086
5	loss=1.7303466105166776	accuracy=0.5393518518518519
6	loss=1.7246608881302823	accuracy=0.5462962962962963
7	loss=1.7073046807889585	accuracy=0.5655864197530864
8	loss=1.7162714622638844	accuracy=0.5532407407407407
9	loss=1.6955189984521748	accuracy=0.5679012345679012
10	loss=1.6899730099572077	accuracy=0.5856481481481481
11	loss=1.6915786722559987	accuracy=0.5802469135802469
12	loss=1.6918290353115695	accuracy=0.5771604938271605
13	loss=1.69221603281704	accuracy=0.5802469135802469
14	loss=1.6856708070378246	accuracy=0.5802469135802469
15	loss=1.68439147501816	accuracy=0.5848765432098766
16	loss=1.6917073255703774	accuracy=0.5709876543209876
17	loss=1.6949014236897597	accuracy=0.5787037037037037
18	loss=1.679313709706436	accuracy=0.5787037037037037
19	loss=1.670586833247432	accuracy=0.6003086419753086
20	loss=1.6547577307548051	accuracy=0.6118827160493827
21	loss=1.663629601031174	accuracy=0.5987654320987654
22	loss=1.6462659041086833	accuracy=0.6180555555555556
23	loss=1.6496582369745514	accuracy=0.6126543209876543
24	loss=1.6347758284321539	accuracy=0.6388888888888888



Loss & Accuracy

```python
with open(val) as file_object:
    lines = file_object.readlines()
    epoch = list(range(0, 18475, 739))
    loss_val = []
    accuracy_val = []
    for line in lines:
        loss_val.append(float(line.split()[1]))
        accuracy_val.append(float(line.split()[2]))

with open(train) as file_object:
    lines = file_object.readlines()
    batch = list(range(0, 18475))
    loss_train = []
    accuracy_train = []
    for line in lines:
        loss_train.append(float(line.split()[1]))
        accuracy_train.append(float(line.split()[2]))
plt.plot(epoch,loss_val,linewidth =2.3,
         label='验证集误差',color='indianred')
plt.plot(epoch,accuracy_val,linewidth =2.3,
         label='验证集准确率',color='cornflowerblue')
plt.plot(batch[0:18475:75],loss_train[0:18475:75],
         linewidth =2.3,label='训练集误差',color='red')
plt.plot(batch[0:18475:75],accuracy_train[0:18475:75],
         linewidth =2.3,label='训练集准确率',color='blue')
plt.xticks([])
plt.legend(loc='upper right')
plt.grid(ls='--')
plt.rcParams['font.sans-serif']=['SimHei'] #用来正常显示中文
plt.title('result of train-val')
```



5

实验结果

部分代码

```
# torch.Size([1, 3, 816, 816])

pred=model(face)

pred_age=torch.argmax(pred,1).item()

matrix[int(age_class), pred_age] = matrix[int(age_class), pred_age] + 1

if pred_age == int(age_class):
#if abs(pred_age - int(age_class))<=1:
    #print('预测正确  预测分类' + str(pred_age) + ' ; 真实分类' + str(age_class))
    correct+=1

else:
    #print('预测失败  预测分类' + str(pred_age) + ' ; 真实分类' + str(age_class))
    wrong+=1
```

实验结果

Exact

```
当前进度  49.815%  累计正确率: 47.581%
当前进度  52.132%  累计正确率: 47.467%
当前进度  54.449%  累计正确率: 47.064%
当前进度  56.766%  累计正确率: 47.510%
当前进度  59.082%  累计正确率: 48.000%
当前进度  61.399%  累计正确率: 47.698%
当前进度  63.716%  累计正确率: 47.709%
当前进度  66.033%  累计正确率: 47.684%
当前进度  68.350%  累计正确率: 47.763%
当前进度  70.667%  累计正确率: 47.770%
当前进度  72.984%  累计正确率: 47.778%
当前进度  75.301%  累计正确率: 47.692%
当前进度  77.618%  累计正确率: 47.582%
当前进度  79.935%  累计正确率: 47.449%
当前进度  82.252%  累计正确率: 47.296%
当前进度  84.569%  累计正确率: 47.342%
当前进度  86.886%  累计正确率: 47.093%
当前进度  89.203%  累计正确率: 46.961%
当前进度  91.520%  累计正确率: 47.139%
当前进度  93.837%  累计正确率: 47.235%
当前进度  96.154%  累计正确率: 47.012%
当前进度  98.471%  累计正确率: 46.894%
--------------
--------------
最终准确率: 47.08063021316033%

Process finished with exit code 0
```

1-off

```
当前进度  56.766%  累计正确率: 82.082%
当前进度  59.082%  累计正确率: 82.039%
当前进度  61.399%  累计正确率: 81.962%
当前进度  63.716%  累计正确率: 81.964%
当前进度  66.033%  累计正确率: 82.035%
当前进度  68.350%  累计正确率: 82.068%
当前进度  70.667%  累计正确率: 82.066%
当前进度  72.984%  累计正确率: 82.032%
当前进度  75.301%  累计正确率: 82.185%
当前进度  77.618%  累计正确率: 82.209%
当前进度  79.935%  累计正确率: 82.232%
当前进度  82.252%  累计正确率: 82.225%
当前进度  84.569%  累计正确率: 82.164%
当前进度  86.886%  累计正确率: 82.160%
当前进度  89.203%  累计正确率: 82.078%
当前进度  91.520%  累计正确率: 81.949%
当前进度  93.837%  累计正确率: 81.852%
当前进度  96.154%  累计正确率: 81.759%
当前进度  98.471%  累计正确率: 81.718%
--------------
--------------
最终准确率: 81.81186283595923%

Process finished with exit code 0
```

结果对比

Exact

最终准确率：47.08063021316033%

1-off

最终准确率：81.81186283595923%

Method	Exact	1-off
Best from [10]	45.1 ± 2.6	79.5 ± 1.4
Proposed using single crop	49.5 ± 4.4	84.6 ± 1.7
Proposed using over-sample	**50.7 ± 5.1**	**84.7 ± 2.2**

Table 3. **Age estimation results on the Adience benchmark.** Listed are the mean accuracy $\pm$ standard error over all age categories. Best results are marked in bold.

6

原因分析

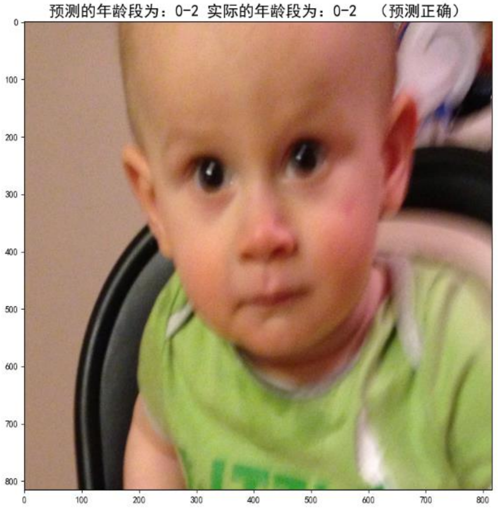
混淆矩阵

矩阵的每一列表达了分类器对于样本的类别预测，二矩阵的每一行则表达了版本所属的真实类别之所以叫做’ 混淆矩阵 ‘，是因为能够很容易的看到机器学习有没有将样本的类别给混淆了。

Age		Predict							
		0-2	4~6	8~13	15~20	25~32	38~43	48~53	60~
Real	0-2	665	232	28	3	8	24	0	0
	4~6	90	284	83	9	9	20	0	0
	8~13	3	14	99	28	32	37	0	0
	15~20	2	4	7	29	77	36	0	0
	25~32	31	65	121	148	725	490	0	0
	38~43	10	6	25	49	235	230	0	0
	48~53	3	4	13	21	83	95	0	0
	60~	1	4	23	19	23	69	0	0

Age		Predict							
		0-2	4~6	8~13	15~20	25~32	38~43	48~53	60~
Real	0-2	0.826	0.378	0.07	0.01	0.007	0.024	0	0
	4~6	0.112	0.463	0.208	0.029	0.008	0.02	0	0
	8~13	0.004	0.023	0.248	0.092	0.027	0.037	0	0
	15~20	0.002	0.007	0.018	0.095	0.065	0.036	0	0
	25~32	0.039	0.106	0.303	0.484	0.608	0.49	0	0
	38~43	0.012	0.01	0.063	0.16	0.197	0.23	0	0
	48~53	0.004	0.007	0.033	0.069	0.07	0.095	0	0
	60~	0.001	0.007	0.058	0.062	0.019	0.069	0	0

预测的年龄段为：0-2 实际的年龄段为：0-2 （预测正确）

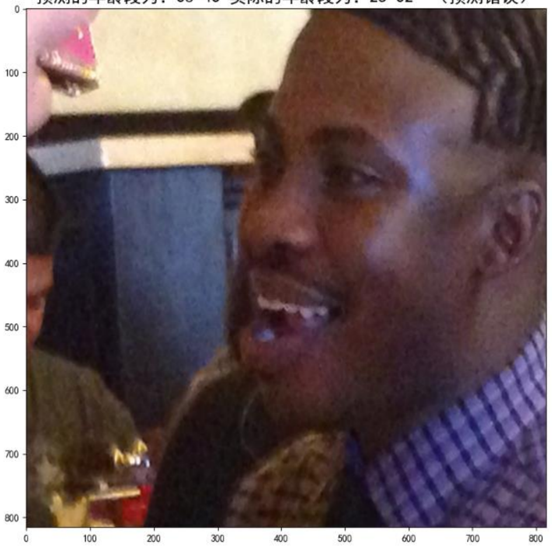


预测的年龄段为：4-6 实际的年龄段为：4-6 （预测正确）

预测的年龄段为：0-2 实际的年龄段为：8-13 （预测错误）

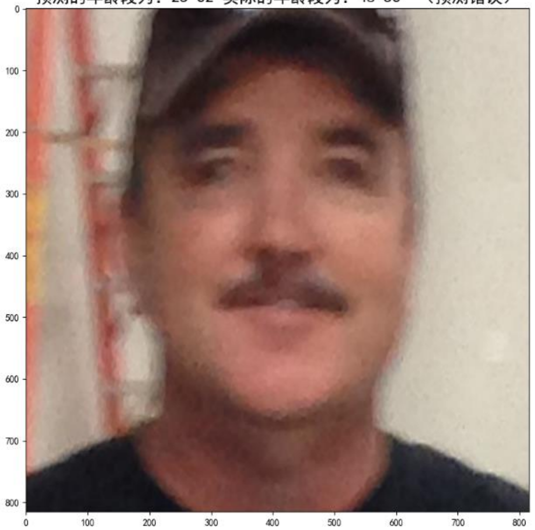
预测的年龄段为：25-32 实际的年龄段为：15-20 （预测错误）

预测的年龄段为：38-43 实际的年龄段为：25-32 （预测错误）

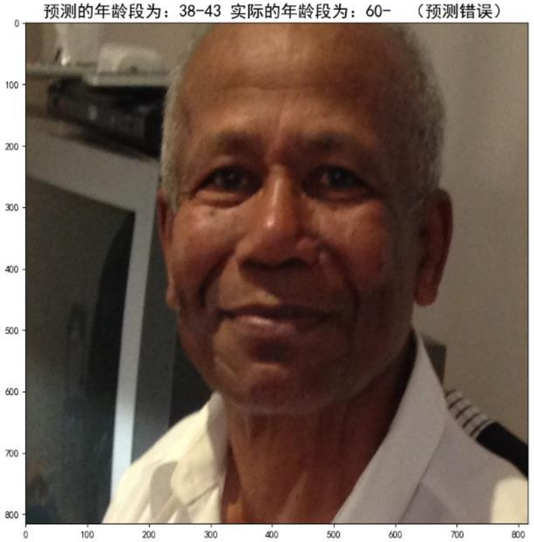


预测的年龄段为：25-32 实际的年龄段为：38-43 （预测错误）

预测的年龄段为：25-32 实际的年龄段为：48-53 （预测错误）



预测的年龄段为：38-43 实际的年龄段为：60- （预测错误）

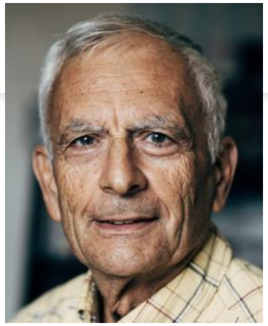


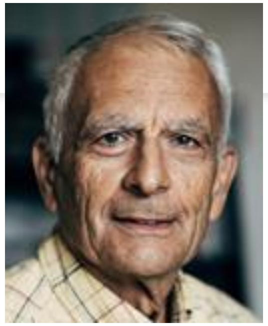
可行的改进

1 **增加高龄人群的面部数据以平衡数据集的分布。**

2 **对数据集中高龄人群数据进行数据增强。**

对图像进行几何变换，包括翻转，旋转，裁剪，变形，缩放等各类操作，从而得到更多可用数据。





Thanks

小组成员：熊锐成、颜昊、曹英凡

于濠铭、崔磊、封兆欣