# Lab\_2 人脸颜值打分

## 一、实习目的与要求

1、结合实际应用理解监督学习分类过程；

2、理论结合实践，采用多种分类方法实现人脸颜值打分；

3、通过进一步查阅文献，了解相关研究方向的最新研究进展。

## 二、实习题目

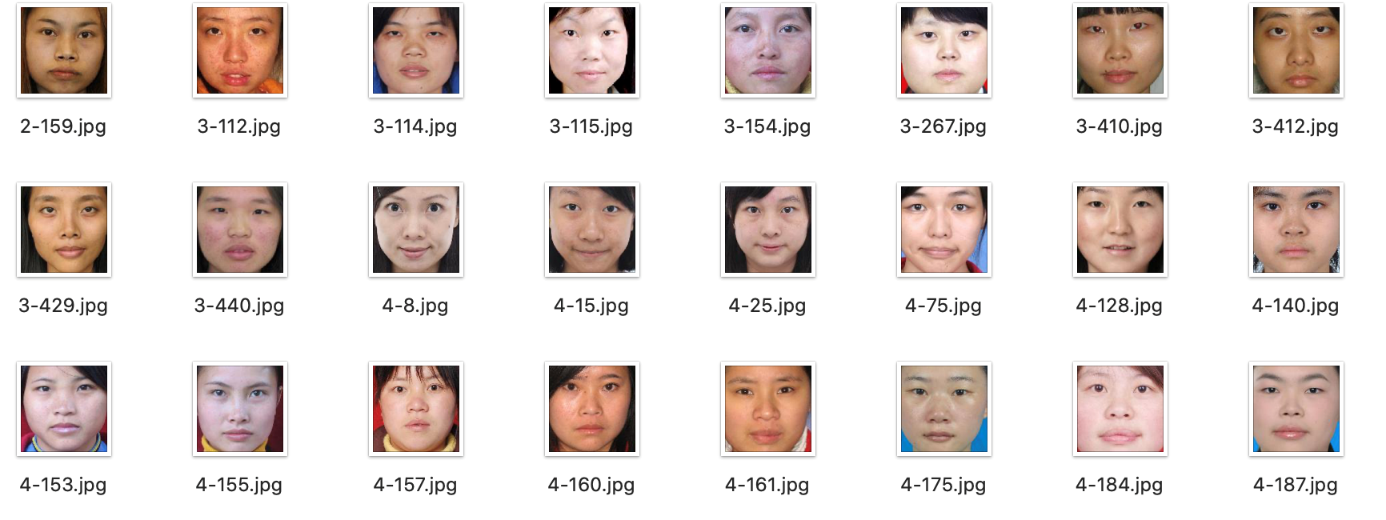
基于VGG分类方法的人脸颜值打分。

#### 【实验数据】

数据中包含500张女生图片，分别由70人进行打分，最终取平均值即为该图片的打分情况。

我们在实践中将图片分值设定为1-5。

500张图片中，450张用于训练，50张用于验证。



#### 【过程描述】

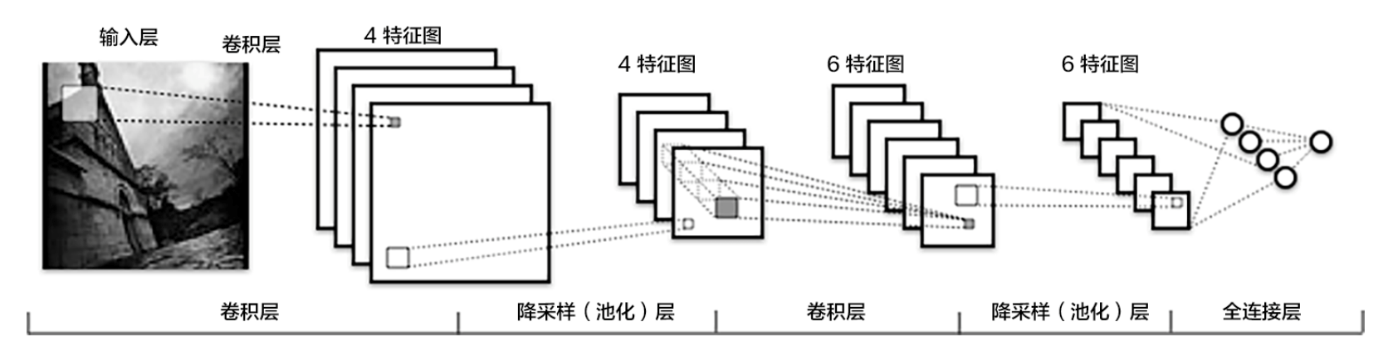
* **step1.数据准备**

1、定义数据提供器 data\_r

2、定义train\_reader、test\_reader

1. def data\_mapper(data):
2. img, label = data
3. img = paddle.dataset.image.load\_image(img)
4. *#将img数组进行进行归一化处理，得到0到1之间的数值*
6. img= img.flatten().astype('float32')/255.0
7. return img, int(label)
8. def data\_reader(data\_path, buffered\_size=512):
9. print(data\_path)
10. def reader():
11. for image in os.listdir(data\_path):
12. label = image.split('-')[0]
13. img = os.path.join(data\_path+ '/' + image)
14. yield img, label
15. return paddle.reader.xmap\_readers(data\_mapper, reader, cpu\_count(), buffered\_size)
16. *#构造训练、测试数据提供器*
17. BATCH\_SIZE = 64
18. train\_r = data\_reader(data\_path='/home/aistudio/face\_image\_train')
19. train\_reader = paddle.batch(paddle.reader.shuffle(reader=train\_r,buf\_size=128),batch\_size=BATCH\_SIZE)
20. test\_r= data\_reader(data\_path='/home/aistudio/face\_image\_test')
21. test\_reader = paddle.batch(test\_r, batch\_size=BATCH\_SIZE)
22. *# print(next(test\_reader()))*

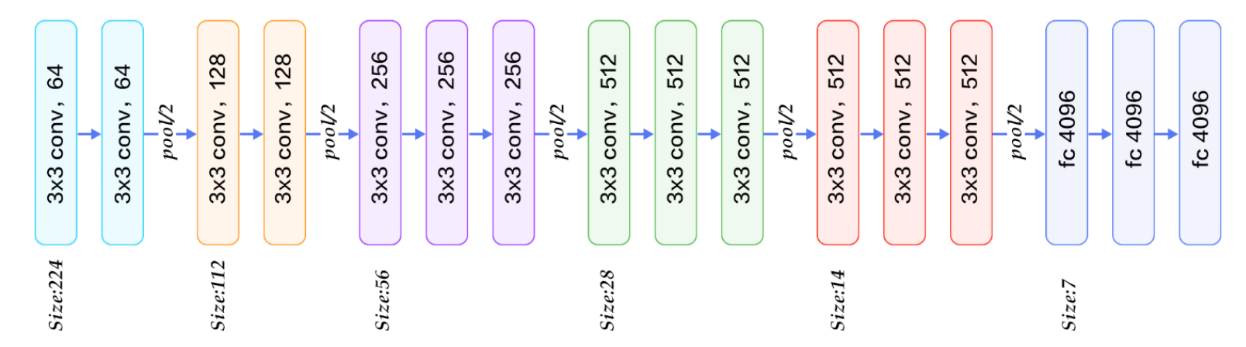
* **Step2.网络配置**



* 卷积层(convolution layer): 执行卷积操作提取底层到高层的特征，发掘出图片局部关联性质和空间不变性质。
* 池化层(pooling layer): 执行降采样操作。通过取卷积输出特征图中局部区块的最大值(max-pooling)或者均值(avg-pooling)。降采样也是图像处理中常见的一种操作，可以过滤掉一些不重要的高频信息。
* 全连接层(fully-connected layer，或者fc layer): 输入层到隐藏层的神经元是全部连接的。

1. def convolutional\_neural\_network(img):
2. *# 第一个卷积-池化层*
3. conv1=fluid.layers.conv2d(input=img,       #输入图像
4. num\_filters=50,  #卷积核数量，它与输出的通道相同
5. filter\_size=5,   #卷积核大小
6. act="relu")      #激活函数
8. drop1 = fluid.layers.dropout(x=conv1, dropout\_prob=0.5)
9. pool1 = fluid.layers.pool2d(input=conv1,                     #输入
10. pool\_size=2,                     #池化核大小
11. pool\_type='max',                 #池化类型
12. pool\_stride=2)                   #池化步长
13. conv\_pool\_1 = fluid.layers.batch\_norm(pool1)   #将该层的输出正则化
14. *# 第二个卷积-池化层*
15. conv2=fluid.layers.conv2d(input=conv\_pool\_1,
16. num\_filters=50,
17. filter\_size=5,
18. act="relu")
19. drop2 = fluid.layers.dropout(x=conv2, dropout\_prob=0.5)
20. pool2 = fluid.layers.pool2d(
21. input=conv2,
22. pool\_size=2,
23. pool\_type='max',
24. pool\_stride=2)
25. conv\_pool\_2 = fluid.layers.batch\_norm(pool2)
26. *# 第三个卷积-池化层*
27. conv3=fluid.layers.conv2d(input=conv\_pool\_2,
28. num\_filters=50,
29. filter\_size=5,
30. act="relu")
31. pool3 = fluid.layers.pool2d(input=conv3,
32. pool\_size=2,
33. pool\_type='max',
34. pool\_stride=2)
35. *# 以softmax为激活函数的全连接输出层*
36. prediction = fluid.layers.fc(input=pool3,
37. size=5,
38. act='softmax')
39. return prediction

搭建VGG网络



**1.**首先定义了一组卷积网络，即conv\_block。卷积核大小为3x3，池化窗口大小为2x2，窗口滑动大小为2，groups决定每组VGG模块是几次连续的卷积操作，dropouts指定Dropout操作的概率。所使用的img\_conv\_group是在paddle.networks中预定义的模块，由若干组 Conv->BN->ReLu->Dropout 和一组 Pooling 组成。

**2.**五组卷积操作，即 5个conv\_block。 第一、二组采用两次连续的卷积操作。第三、四、五组采用三次连续的卷积操作。每组最后一个卷积后面Dropout概率为0，即不使用Dropout操作。

**3.**最后接两层512维的全连接。

**4.**通过上面VGG网络提取高层特征，然后经过全连接层映射到类别维度大小的向量，再通过Softmax归一化得到每个类别的概率，也可称作分类器。

1. def vgg\_bn\_drop(image, type\_size):
2. def conv\_block(ipt, num\_filter, groups, dropouts):
3. return fluid.nets.img\_conv\_group(
4. input=ipt, # 具有[N，C，H，W]格式的输入图像
5. pool\_size=2,
6. pool\_stride=2,
7. conv\_num\_filter=[num\_filter] \* groups, # 过滤器个数
8. conv\_filter\_size=3, # 过滤器大小
9. conv\_act='relu',
10. conv\_with\_batchnorm=True, # 表示在 Conv2d Layer 之后是否使用 BatchNorm
11. conv\_batchnorm\_drop\_rate=dropouts,# 表示 BatchNorm 之后的 Dropout Layer 的丢弃概率
12. pool\_type='max') # 最大池化
13. conv1 = conv\_block(image, 64, 2, [0.0, 0])
14. conv2 = conv\_block(conv1, 128, 2, [0.0, 0])
15. conv3 = conv\_block(conv2, 256, 3, [0.0, 0.0, 0])
16. conv4 = conv\_block(conv3, 512, 3, [0.0, 0.0, 0])
17. conv5 = conv\_block(conv4, 512, 3, [0.0, 0.0, 0])
18. drop = fluid.layers.dropout(x=conv2, dropout\_prob=0.5)
19. fc1 = fluid.layers.fc(input=drop, size=512, act=None)
21. bn = fluid.layers.batch\_norm(input=fc1, act='relu')
22. drop2 = fluid.layers.dropout(x=bn, dropout\_prob=0.0)
23. fc2 = fluid.layers.fc(input=drop2, size=512, act=None)
24. *# predict = fluid.layers.fc(input=fc1, size=type\_size, act='softmax')*
25. predict = fluid.layers.fc(input=fc1, size=type\_size)
26. return predict
27. *#定义两个张量*
28. image = fluid.layers.data(name='image', shape=[3, 224, 224], dtype='float32')
29. label = fluid.layers.data(name='label', shape=[1], dtype='int64')

定义损失函数、优化方法

交叉熵损失函数在分类任务上比较常用。

定义了一个损失函数之后，还要对它求平均值，因此定义的是一个Batch的损失值。

同时我们还可以定义一个准确率函数，这个可以在我们训练的时候输出分类的准确率。

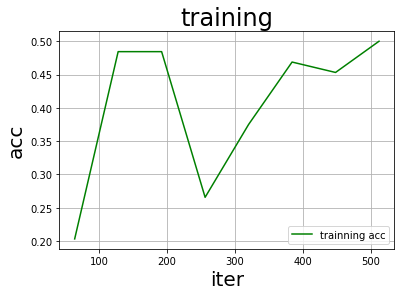
1. *#定义损失函数*
2. *# cost = fluid.layers.cross\_entropy(input=predict, label=label)*
3. cost = fluid.layers.softmax\_with\_cross\_entropy(logits=predict, label=label)
4. avg\_cost = fluid.layers.mean(cost)
5. accuracy = fluid.layers.accuracy(input=predict, label=label)
6. *# print(fluid.default\_main\_program().to\_string(True))*
7. *#克隆main\_program得到test\_program，使用参数for\_test来区分该程序是用来训练还是用来测试，该api请在optimization之前使用.*
8. test\_program = fluid.default\_main\_program().clone(for\_test=True)
9. *#优化方法*
10. optimizer = fluid.optimizer.Adam(learning\_rate=0.0002)    *# Adam是一阶基于梯度下降的算法，基于自适应低阶矩估计该函数实现了自适应矩估计优化器*
11. optimizer.minimize(avg\_cost)
12. *#use\_cuda为False,表示运算场所为CPU;use\_cuda为True,表示运算场所为GPU*
13. use\_cuda = True
14. place = fluid.CUDAPlace(0) if use\_cuda else fluid.CPUPlace()
15. *#创建一个Executor实例exe*
16. exe = fluid.Executor(place)
17. *#正式进行网络训练前，需先执行参数初始化*
18. exe.run(fluid.default\_startup\_program())
19. *#数据映射器*
20. feeder = fluid.DataFeeder(feed\_list=[image, label], place = place)

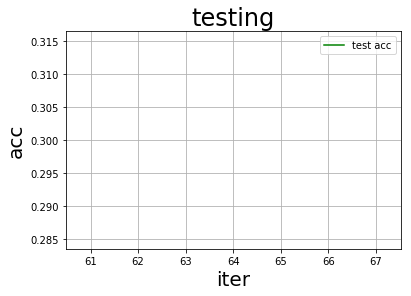
定义数据映射器

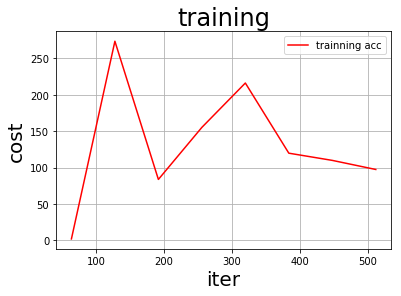
DataFeeder负责将数据提供器（train\_reader,test\_reader）返回的数据转成一种特殊的数据结构，使其可以输入到Executor中。

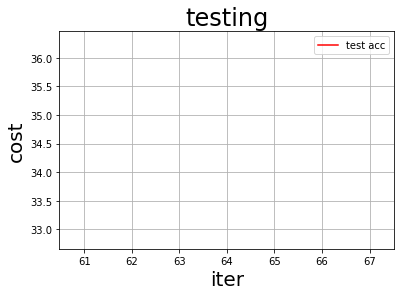
feed\_list设置向模型输入的向变量表或者变量表名

* **Step3.模型训练**
* **Step4.模型评估**
* *#展示模型训练曲线*
* all\_train\_iter=0
* all\_train\_iters=[]
* all\_train\_costs=[]
* all\_train\_accs=[]
* all\_test\_iter=0
* all\_test\_iters=[]
* all\_test\_costs=[]
* all\_test\_accs=[]
* *# def draw\_process(title,iters,costs,accs,label\_cost,lable\_acc):*
* *#     plt.title(title, fontsize=24)*
* *#     plt.xlabel("iter", fontsize=20)*
* *#     plt.ylabel("cost/acc", fontsize=20)*
* *#     plt.plot(iters, costs,color='red',label=label\_cost)*
* *#     plt.plot(iters, accs,color='green',label=lable\_acc)*
* *#     plt.legend()*
* *#     plt.grid()*
* *#     plt.show()*
* def draw\_cost\_process(title,iters,costs,label\_cost):
* plt.title(title, fontsize=24)
* plt.xlabel("iter", fontsize=20)
* plt.ylabel("cost", fontsize=20)
* plt.plot(iters, costs,color='red',label=label\_cost)
* plt.legend()
* plt.grid()
* plt.show()
* def draw\_acc\_process(title,iters,acc,label\_acc):
* plt.title(title, fontsize=24)
* plt.xlabel("iter", fontsize=20)
* plt.ylabel("acc", fontsize=20)
* plt.plot(iters, acc,color='green',label=label\_acc)
* plt.legend()
* plt.grid()
* plt.show()
* model\_save\_dir = "/home/aistudio/work"
* EPOCH\_NUM = 1
* for pass\_id in range(EPOCH\_NUM):
* train\_cost = 0
* for batch\_id, data in enumerate(train\_reader()):                         *#遍历train\_reader的迭代器，并为数据加上索引batch\_id*
* train\_cost, train\_acc = exe.run(program=fluid.default\_main\_program(),                            *#运行主程序*
* feed=feeder.feed(data),                                          *#喂入一个batch的数据*
* fetch\_list=[avg\_cost, accuracy])                                 *#fetch均方误差和准确率*
* all\_train\_iter=all\_train\_iter+BATCH\_SIZE
* all\_train\_iters.append(all\_train\_iter)
* all\_train\_costs.append(train\_cost[0])
* all\_train\_accs.append(train\_acc[0])
* if batch\_id % 10 == 0:                                               *#每50次batch打印一次训练、进行一次测试*
* print("\nPass %d, Step %d, Cost %f, Acc %f" % (pass\_id, batch\_id, train\_cost[0], train\_acc[0]))
* test\_accs = []                                                            *#测试的损失值*
* test\_costs = []                                                           *#测试的准确率*
* *# 每训练一轮 进行一次测试*
* for batch\_id, data in enumerate(test\_reader()):                           *# 遍历test\_reader*
* test\_cost, test\_acc = exe.run(program=test\_program,  *# #运行测试主程序*
* feed=feeder.feed(data),                *#喂入一个batch的数据*
* fetch\_list=[avg\_cost, accuracy])       *#fetch均方误差、准确率*
* test\_accs.append(test\_acc[0])                                        *#记录每个batch的误差*
* test\_costs.append(test\_cost[0])                                      *#记录每个batch的准确率*
* all\_test\_iter=all\_test\_iter+BATCH\_SIZE
* all\_test\_iters.append(all\_test\_iter)
* all\_test\_costs.append(test\_cost[0])
* all\_test\_accs.append(test\_acc[0])
* test\_cost = (sum(test\_costs) / len(test\_costs))                           *# 每轮的平均误差*
* test\_acc = (sum(test\_accs) / len(test\_accs))                              *# 每轮的平均准确率*
* print('Test:%d, Cost:%0.5f, ACC:%0.5f' % (pass\_id, test\_cost, test\_acc))
* if not os.path.exists(model\_save\_dir):
* os.makedirs(model\_save\_dir)
* *# 保存训练的模型，executor 把所有相关参数保存到 dirname 中*
* fluid.io.save\_inference\_model(model\_save\_dir,  *#保存推理model的路径*
* ['image'],       *#推理（inference）需要 feed 的数据*
* [predict],       *#保存推理（inference）结果的 Variables*
* exe)             *#executor 保存 inference model*
* print('训练模型保存完成！')
* *#调用绘制曲线*
* draw\_acc\_process("training", all\_train\_iters, all\_train\_accs, "trainning acc")
* draw\_acc\_process("testing", all\_test\_iters, all\_test\_accs, "test acc")
* draw\_cost\_process("training", all\_train\_iters, all\_train\_costs, "trainning acc")
* draw\_cost\_process("testing", all\_test\_iters, all\_test\_costs, "test acc")

****

****

****

****

* **Step5.模型预测**
* *#模型预测#*
* infer\_exe = fluid.Executor(place)
* inference\_scope = fluid.core.Scope()*#要想运行一个网络，需要指明它运行所在的域，确切的说： exe.Run(&scope)*
* *#图片预处理*
* def load\_image(file):
* im = Image.open(file)
* im = im.resize((224, 224), Image.ANTIALIAS)                 *#resize image with high-quality 图像大小为128\*128*
* im = np.array(im).reshape(1, 3, 224, 224).astype(np.float32)*#返回新形状的数组,把它变成一个 numpy 数组以匹配数据馈送格式。*
* im = im / 255.0                            *#归一化到【-1~1】之间*
* return im
* infer\_img='/home/aistudio/1.jpg'
* *#获取训练好的模型*
* *#从指定目录中加载 推理model(inference model)*
* [inference\_program,*# 预测用的program*
* feed\_target\_names,*# 是一个str列表，它包含需要在推理 Program 中提供数据的变量的名称。*
* fetch\_targets] = fluid.io.load\_inference\_model(model\_save\_dir, infer\_exe)*#fetch\_targets：是一个 Variable 列表，从中我们可以得到推断结果。*
* img = Image.open(infer\_img)
* *# plt.imshow(img)   #根据数组绘制图像*
* *# plt.show()        #显示图像*
* image=load\_image(infer\_img)
* *# 开始预测*
* results = infer\_exe.run(
* inference\_program,                      *#运行预测程序*
* feed={feed\_target\_names[0]: image},*#喂入要预测的数据*
* fetch\_list=fetch\_targets)               *#得到推测结果*
* print('results',results)
* label\_list = ["1", "2", "3", "4", "5"]
* print("infer results: %s" % label\_list[np.argmax(results[0])])
* for batch\_id,data in enumerate(train\_reader()):
* print(batch\_id)
* print(len(data))
* print(all\_test\_iters,all\_test\_iters, all\_test\_costs, all\_test\_accs)

#### 【源代码】

见附件

#### 【改进设想】

根据人脸图像的特点，定义并提取了一系列人脸美学分析特征，包括点特征、几何特征和比例特征。通过提取这些特征作为美丽指数的研究基础，然后对上述特征按照一定准则进行归一化，以保证特征的稳定性。定义了美丽指数主观分析准则：在美丽指数的研究中，需要定义一些主观的评价准则，一方面来指导客观准则的建立，另一方面可以验证客观分析模型的有效性。在此我们提出了美丽指数分级打分的思想，首先建立主观评价准则，然后邀请大量的志愿者对一定规模的人脸美学研究数据集进行主观评价。对打分结果进行统计分析，通过定义不同的美丽指数等级和统计结果来寻找合适的美丽指数等级。人脸美丽指数客观分析模型的建立：在主观评价的基础上，建立了基于最近邻分类算法的人脸美丽指数客观分析模型，在该模型下对不同人脸特征以及多种美学特征进行融合分类，并结合主观评价结果，来检验该分析模型的有效性。

## 三、本次实习小结

经过对颜值、美丽值或者是魅力值打分相关的资料整理。方法大致有三类，非监督方法、监督方法和深度学习相关的方法。

在这些方法中，使用的判别标准或者说是参考特征主要有：人脸五官大小和相对位置特征，例如，眼宽、瞳孔距，脸宽，鼻宽，鼻长，嘴宽，上唇厚度，下唇厚度，眉心距以及比例特征等；人脸纹理特征，例如肤色、皮肤光滑程度等。

利用深度学习和监督方法需要大量样本，通过对样本的学习对新的样本进行预测；非监督方法中，也需要一些样本，根据提取的标准按照一定的方法建立评测标准来进行打分。

=======================================================

我的联系方式：

电话：13349830890

邮箱：liuyy@cug.edu.cn