****

课程设计报告书

**题目：基于深度学习的颜值预测**

**学 院 计算机科学与工程学院**

**专 业 计算机科学与技术专业**

**学生姓名 陈浩鑫**

**学生学号 201530311042**

**指导教师 张艳青**

**课程编号**

**课程学分**

**起始日期**

|  |  |
| --- | --- |
| 教  师  评  语 | 教师签名：  日期： |
| 成  绩  评  定 |  |
| 备  注 |  |

**基于深度学习的颜值预测**

**一、选题背景**

颜值是表示一个人面容英俊或美丽的数值，用以评价人物的样貌。颜值的判定是比较主观的，但也有某一些普适性的评判标准，所以这就有一个潜在的规则供我们去寻找。颜值的判定，主要是看脸部是否协调，如黄金分割在人脸上也同样适用。

正因人脸颜值的共性且遵循一定的客观规律，这为人脸颜值计算提供了依据。人脸识别的问题在机器学习领域也可找到对应的应用算法，适应的算法有：特征提取与推理预测。特征提取关注于将人脸高维图片转化为低维特征，如：几何特征。推理预测关注于将特征进行推理，常见有分类与回归算法。分类是将颜值分为多类，回归是用数值表示预测的分数。

颜值的计算，也可同样应用于其他对人脸进行分别的领域。这种思想是普适性的，即通过算法寻找出最优的目标。

传统的人脸颜值算法有：

基于几何特征的方法，通过提取人脸的特征点，计算各个特征点间的距离，然后根据特征点间的距离计算五官协调度，即颜值。

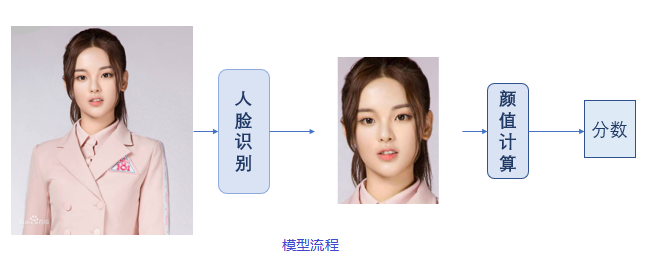
基于纹理的方法，通过检测人脸皮肤的状况对人脸的颜值进行计算，光滑程度与颜值成正比。

提取特征，进行特征的预测分类的端到端学习。这一任务在深度学习领域得到很好的解决。自从2012年Alex在ImageNet竞赛中提出的五层卷积AlexNet以来，神经网络得到研究人员的广泛关注。不需要人工特诊的提取，与特征的选择，通过网络学习特征的表示与推理是深度学习的一大特点。

使用热门的深度学习去解决一个有趣的问题，是本次实验的出发点。

**二、方案论证**

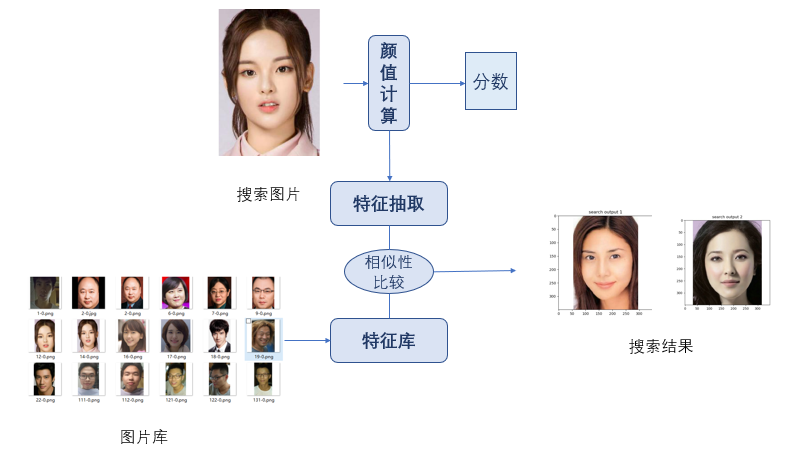
本次实验的目标是对输入图片的人脸进行评分。

故实验框架分为两部分：人脸识别与颜值评价。思路是对一张输入图片，识别出图片中的人脸部分；再将识别出的人脸进行颜值预测。

人脸识别输入为一张图片，算法识别出图片中的人脸后进行抠图输出，并输入到颜值计算模块，通过训练一个特征提取、回归预测算法对人脸进行打分。

这两部分的实现由我和队友洪浩强。

洪浩强负责：人脸识别模块 ； 我负责：颜值计算模块

此外，通过深度学习的好处是：可以抽出中间层的特征进行相似性比较，实现相似性搜索。所以颜值计算模块可以进行扩展：

首先训练一个预测模型，抽取其中的特征输出作为样本的特征，将图片库中的所有样本进行特征提取输出为特征库。在线算法每次输入一张搜索图片，对图片进行特征提取后，与特征库进行相似性比较，将得分最高的几张图进行输出。

**三、过程论述**

我负责的是颜值预测与相似搜索部分。

颜值预测部分：

主要的思路是对采用卷积网络进行特征提取，并接入全连接网络回归预测。

卷积网络：一个简单的卷积网络由以下层组成：

a数据输入层/ Input layer

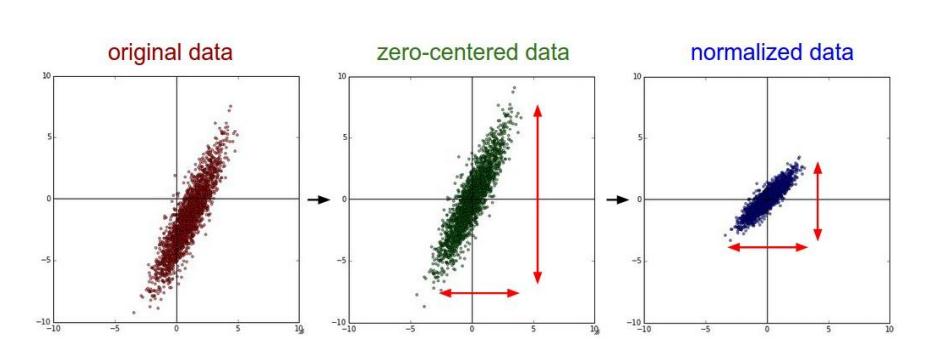
b卷积计算层/ CONV layer

c激励层 / Active layer

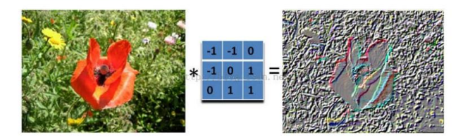
d池化层 / Pooling layer

e 全连接层 / FC layer

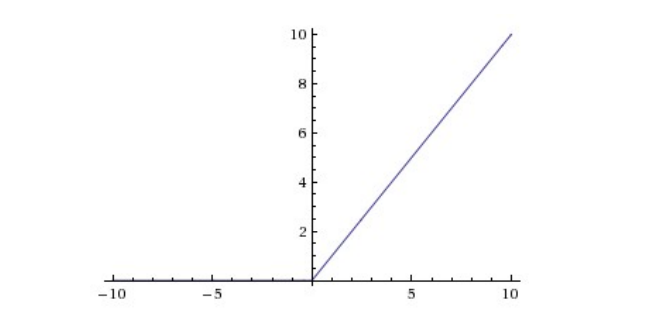
数据输入层，该层要做的处理主要是对原始图像数据进行预处理。包括：去均值、归一化、PCA/白化。

取均值是把输入数据各个维度都中心化为0，归一化是幅度归一化到同样的范围，PCA是将维度降低到方差最大的几个不相关维度。

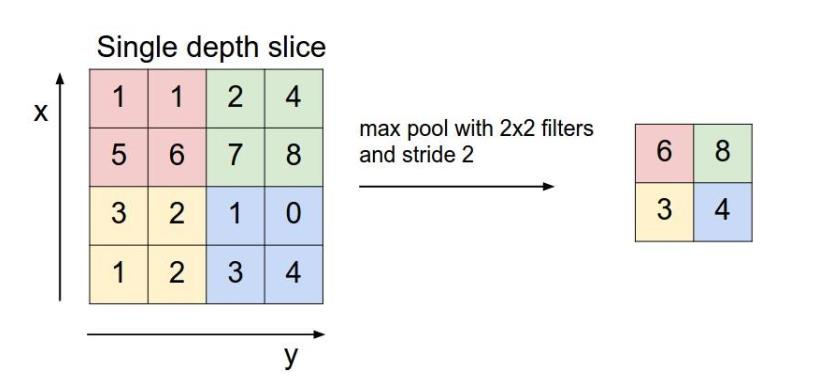
上图是将数据去均值归一化的处理。

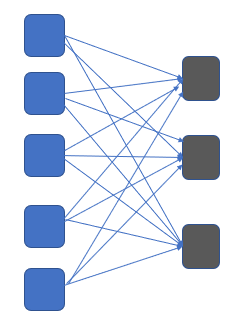
卷积计算层，该层要做的处理是用一个卷积核对输入进行卷积操作。通过卷积核窗口的滑动计算出各个局部的卷积值。

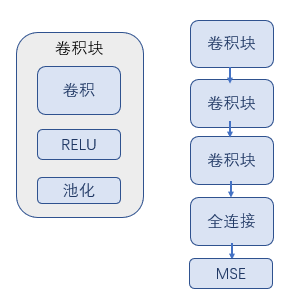
主要功能：对数据的特征学习与选取。

激励层的计算，该层要做的处理是对卷积层的输出进行非线性变换。将线性空间的组合映射到非线性变换中。

上图为有效且主流的Relu函数。

池化层是为卷积网络提供了对图像平移，伸缩变换带来影响的一个层，将一个局部的输出取均值或取最大值。

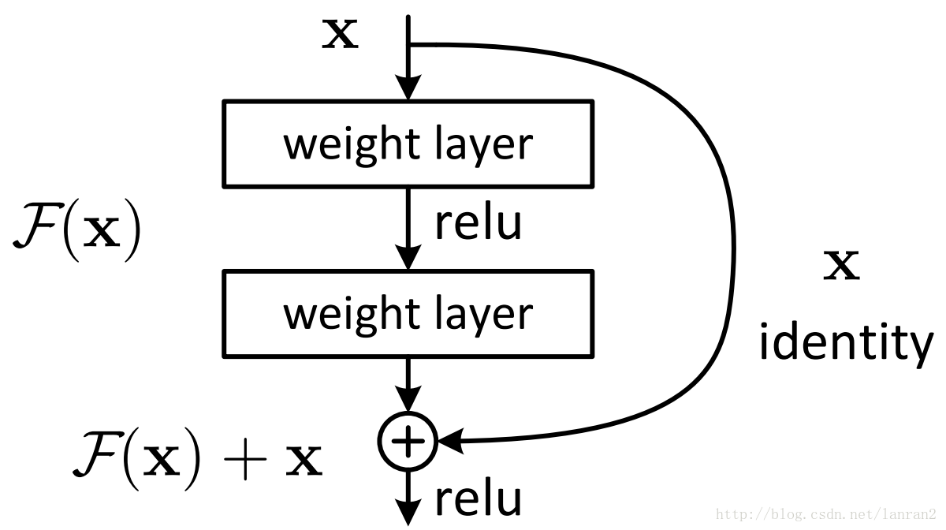
全连接层的计算是将最后的卷积输出用权重进行组合，匹配到分类或者预测问题。

本次实验一开始使用以下的网络：

但由于效果不佳，无法选取到合适参数使得误差降低到合适程度。

而后采用了迁移训练ResNet网络。

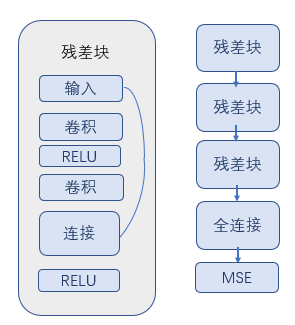
ResNet是2015年微软提出的残差网络，ResNet50深度达到152层卷积的深度网络。

它使用了一种连接方式叫做“shortcut connection”

使得原来阻止神经网络层数加深的梯度消失问题得到解决。

残差网络在将输入与输入的卷积计算组成输出，使得梯度回传时，通过原始输入的这一通路进行回传。

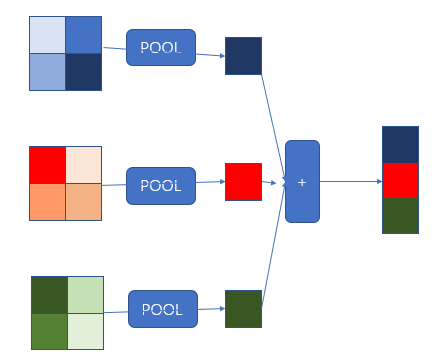
这个简单的加法并不会给网络增加额外的参数和计算量，同时却可以大大增加模型的训练速度、提高训练效果，并且当模型的层数加深时，这个简单的结构能够很好的解决退化问题。

迁移训练是将一个针对原域的网络通过微调进而对目标域起到作用的过程。我采用了ResNet的预训练网络，并通过将ResNet的后面层的去除，接入适应预测目标的一维数值，接入了对应的全连接层。

Loss是计算预测值与颜值间的MSE（均方差误差）进行估量的。

通过Adam算法在loss function上寻找优解。

相似搜索部分是

在全连接层之前，拉取卷积的特征进行通道的对比。

上图是将提取特征组成特征向量的示意图。

相似性比较是通过点积直接计算得到，类似的方法也有余弦相似度。

通过将待搜索图片的特征与特征库的特征做相似性计算后，得出的得分数组进行排序，得到相似性的排序。

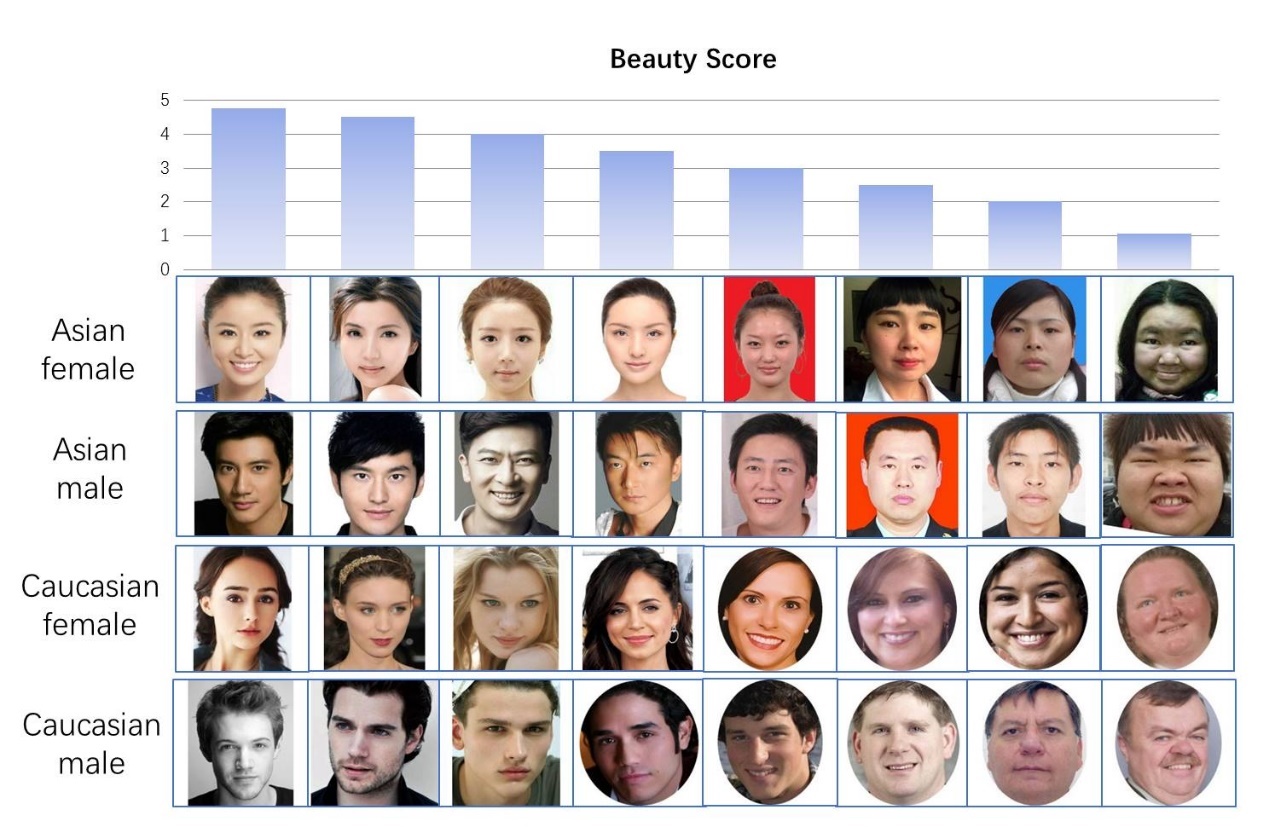
数据集描述：

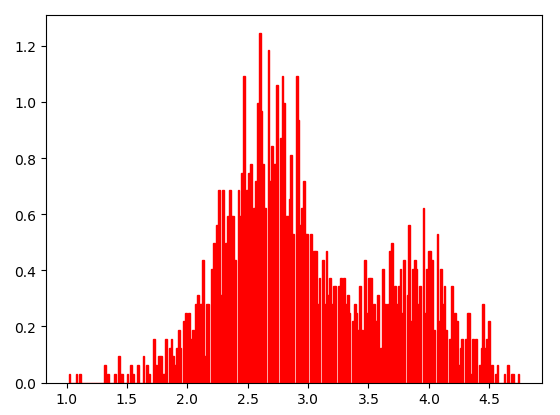
本模块数据是基于华南理工大学SCUT-FBP5500-Database-Release数据集的。

数据集描述：

共5500张图片，按60%训练集40%测试集划分。

每张图片有1-5分的颜值评分范围。



数据集分布：

代码实现部分：

本模块是在keras平台实现的：

1、数据输入部分：

Getdata.py

def saveDataToNpy():

X\_Train\_Name,train\_Y = get\_data\_name('train')

X\_Test\_Name,test\_Y = get\_data\_name('test')

N\_train = X\_Train\_Name.shape[0]

N\_test = X\_Test\_Name.shape[0]

train\_X = np.array([imread(DataName+X\_Train\_Name[i]) for i in range(N\_train)])

np.save('train\_X',train\_X)

np.save('train\_Y',train\_Y)

test\_X = np.array([imread(DataName+X\_Test\_Name[i]) for i in range(N\_test)])

np.save('test\_X',test\_X)

np.save('test\_Y',test\_Y)

负责将训练集与测试集的图片读取，并存为numpy\_array格式

2、卷积网络实现：

Face\_cnn.py

def main():

if K.image\_data\_format() == 'channels\_first':

input\_shape = (3, img\_width, img\_height)

else:

input\_shape = (img\_width, img\_height, 3)

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=input\_shape))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))

model.add(Activation('relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(64))

model.add(Activation('relu'))

model.add(Dropout(0.5))

model.add(Dense(1))

model.compile(loss='mean\_squared\_error',

optimizer=optimizers.SGD(lr=1e-5, momentum=0.9),

#optimizer=optimizers.Adam(),

metrics=['accuracy'])

model.load\_weights('face\_cnn\_epochs\_2.h5')

train\_X, train\_Y ,test\_X,test\_Y= get\_data()

3层卷积+全连接

3、残差网络实现：

def main():

input\_tensor = Input(shape=(350, 350, 3))

resnet = ResNet50(include\_top=False, input\_tensor=input\_tensor)

model = Sequential()

model.add(resnet)

model.add(Flatten(input\_shape=resnet.output\_shape[1:]))

model.add(Dense(1))

print(model.summary())

model.load\_weights('model\_res\_nopool\_train\_e7.h5')

train\_X, train\_Y, test\_X, test\_Y = get\_data()

model.compile(loss='mean\_squared\_error', optimizer=Adam(), metrics=['accuracy'])

model.fit(batch\_size=32,

x=train\_X, y=train\_Y,

epochs=3,

validation\_data=(test\_X, test\_Y))

model.save\_weights('model\_res\_false\_train\_e3.h5')

残差网络+全连接

x = ZeroPadding2D(padding=(3, 3), name='conv1\_pad')(img\_input)

x = Conv2D(64, (7, 7), strides=(2, 2), padding='valid', name='conv1')(x)

x = BatchNormalization(axis=bn\_axis, name='bn\_conv1')(x)

x = Activation('relu')(x)

x = MaxPooling2D((3, 3), strides=(2, 2))(x)

x = conv\_block(x, 3, [64, 64, 256], stage=2, block='a', strides=(1, 1))

x = identity\_block(x, 3, [64, 64, 256], stage=2, block='b')

x = identity\_block(x, 3, [64, 64, 256], stage=2, block='c')

x = conv\_block(x, 3, [128, 128, 512], stage=3, block='a')

x = identity\_block(x, 3, [128, 128, 512], stage=3, block='b')

x = identity\_block(x, 3, [128, 128, 512], stage=3, block='c')

x = identity\_block(x, 3, [128, 128, 512], stage=3, block='d')

x = conv\_block(x, 3, [256, 256, 1024], stage=4, block='a')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='b')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='c')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='d')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='e')

x = identity\_block(x, 3, [256, 256, 1024], stage=4, block='f')

x = conv\_block(x, 3, [512, 512, 2048], stage=5, block='a')

x = identity\_block(x, 3, [512, 512, 2048], stage=5, block='b')

x = identity\_block(x, 3, [512, 512, 2048], stage=5, block='c')

x = AveragePooling2D((7, 7), name='avg\_pool')(x)

残差网络：keras有预训练的resnet50。此处使用预训练模型。

3、测试集验证部分

test\_X\_Result = np.array([])

for i in range(220):

temp = model.predict(test\_X[i\*10:(i+1)\*10])

temp = temp.reshape(1,-1)

if i == 0:

test\_X\_Result = temp

else:

test\_X\_Result = np.hstack((test\_X\_Result,temp))

np.savetxt('result\_nopool\_0.txt',test\_X\_Result,fmt='%.6f')

4、数据误差分析与分布展示

Test.py

def PC(X,Y):

n = X.shape[0]

EXY = np.sum(X\*Y)/n

EX = np.sum(X)/n

EY = np.sum(Y)/n

EX2 = np.sum(X\*X)/n

EY2 = np.sum(Y\*Y)/n

pc = (EXY-EX\*EY)/(np.sqrt(EX2-EX\*\*2)\*np.sqrt(EY2-EY\*\*2))

print(X,Y,EXY,EX,EY,EX2,EY2)

return pc

print('PC',PC(test,testY))

print('MAE',np.sum(abs(test-testY))/2200)

print('RMSE',np.sqrt(np.sum(np.square(test-testY))/2200))

print('MSE',np.sum(np.square(test-testY))/2200)

import matplotlib.pyplot as plt

plt.figure("lena")

plt.hist(testY, bins=256, normed=1,facecolor='r',edgecolor='r',hold=1)

#plt.hist(test, bins=256, normed=1,facecolor='g',edgecolor='g',hold=1)

#plt.hist(t, bins=256, normed=1,facecolor='b',edgecolor='b',hold=1)

plt.show()

5、特征提取：

Imagesearch下：

class RESNet:

def \_\_init\_\_(self):

self.input\_shape = (350, 350, 3)

self.model = ResNet50(include\_top=False, pooling='avg')

self.model.predict(np.zeros((1, 350, 350 , 3)))

def extract\_feat(self, img\_path):

img = image.load\_img(img\_path, target\_size=(self.input\_shape[0], self.input\_shape[1]))

img = image.img\_to\_array(img)

img = np.expand\_dims(img, axis=0)

img = preprocess\_input(img)

feat = self.model.predict(img)

norm\_feat = feat[0]/LA.norm(feat[0])

return norm\_feat

特征写入到h5文件

model = RESNet()

for i, img\_path in enumerate(img\_list):

norm\_feat = model.extract\_feat(img\_path)

img\_name = os.path.split(img\_path)[1]

feats.append(norm\_feat)

names.append(img\_name.encode())

feats = np.array(feats)

output = args["index"]

h5f = h5py.File(output, 'w')

h5f.create\_dataset('dataset\_1', data = feats)

h5f.create\_dataset('dataset\_2', data = names)

h5f.close()

6、颜值计算与相似性搜索的在线预测

model = RESNet()

predict = Predict()

def query(queryDir):

queryImg = mpimg.imread(queryDir)

plt.title("Query Image")

plt.imshow(queryImg)

plt.show()

print('颜值:',predict.predict(queryDir,weight))

queryVec = model.extract\_feat(queryDir)

scores = np.dot(queryVec, feats.T)

rank\_ID = np.argsort(scores)[::-1]

rank\_score = scores[rank\_ID]

maxres = 3

imlist = [imgNames[index] for i,index in enumerate(rank\_ID[0:maxres])]

for i,im in enumerate(imlist):

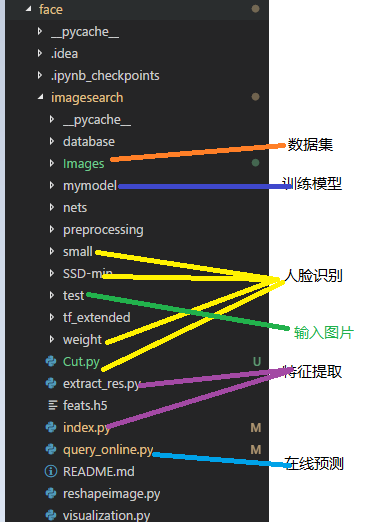
im = im.decode()

image = mpimg.imread(args["result"]+"/"+str(im))

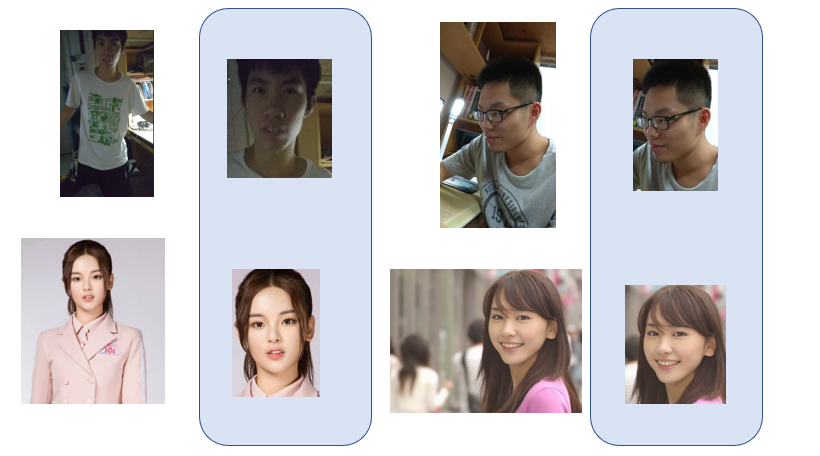
plt.title("search output %d" %(i+1))

plt.imshow(image)

plt.show()

项目架构：

**四、结果分析**

人脸识别：

将输入图片中的人脸输出：

ymin = int(bboxes[i, 0] \* height\*0.9)

xmin = int(bboxes[i, 1] \* width\*0.85)

ymax = int(bboxes[i, 2] \* height\*1.2)

xmax = int(bboxes[i, 3] \* width\*1.1)

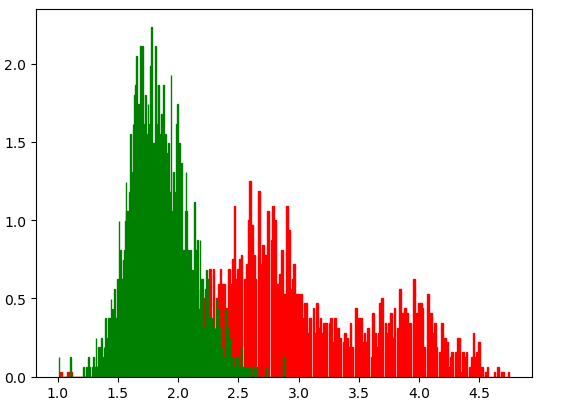
输出的边框放大。

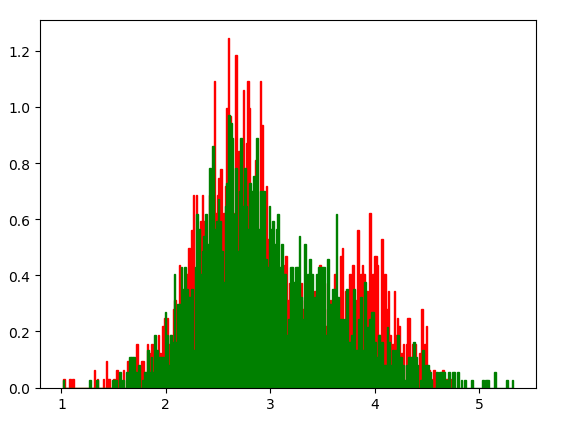
缩放到350\*350大小，空缺处填白色。

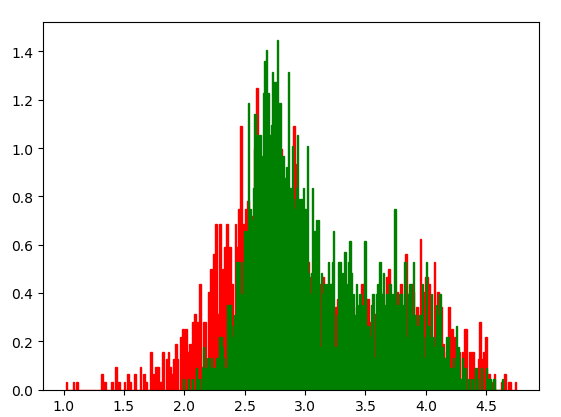
颜值预测：

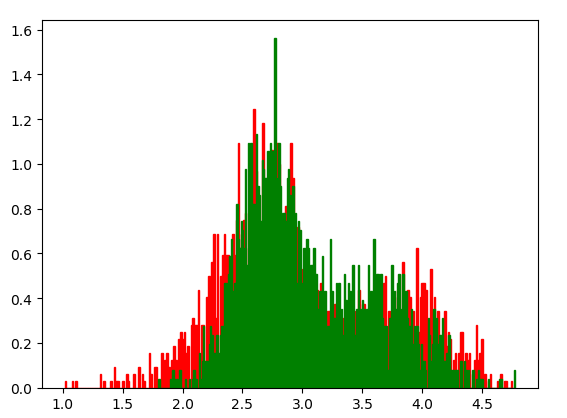
|  |  |  |
| --- | --- | --- |
|  | MSE-LOSS | PC（皮尔逊） |
| 3层CNN | 1.652 | 0.582 |
| ResNet+Pool+FC | 0.136 | 0.855 |
| ResNet+FC | 0.187 | 0.786 |
| 集成两个ResNet(1：1) | 0.121 | 0.864 |

CNN预测结果：

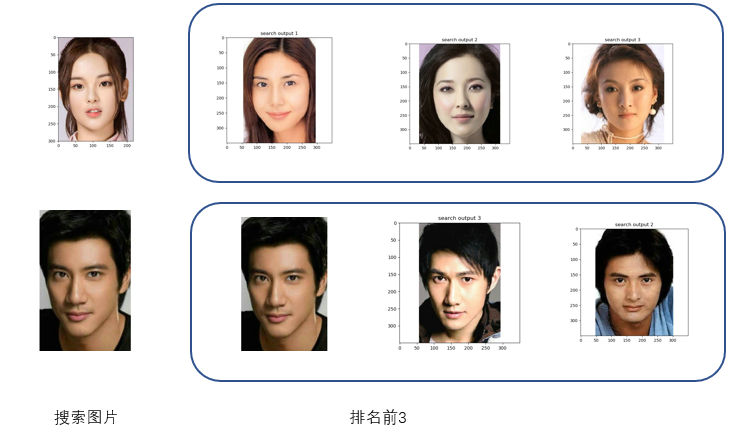
3层CNN

ResNet+FC 

ResNet+Pool+FC 

集成两个ResNet(1：1)

可见集成了两个resNet的模型拟合数据分布最佳。

相似性搜索：

**五、课程设计总结**

本次课程的实验对于我来说，更多是一种对深度学习的实践，而不是研究与创新。所以，在模型构造方面采用了传统卷积+迁移训练，没有采用自己搭建更加深层的网络。

从实践角度上，这次的课程作业，让我接触了数据集获取，数据集输入，模型的搭建，模型训练，模型预测，结果可视化，特征抽取等流程。

数据集获取：人脸识别部分是通过获取别的数据集，加以适配，转换得到的，这种方法也是可以用于自己打标签。颜值预测的数据集则是从github上找到的。

数据集输入：使用图片库函数，读取样本-标签后，将样本路径的图片读取并转化为数组。

模型的搭建：在不同平台搭建的方式是不同，keras上使用序列化的搭建，依次堆叠网络层，加入损失函数后，自动求导，后向传播。

模型训练，batch训练能加快训练速度，且有效地防止结果到较差的局部最优。Epochs是训练周期，对于大数据下，一个epochs可能需要几个小时，所以训练时，为了防止后续出错。需要用小数据进行训练，再正式训练。

模型预测，将模型的权重导出之后便可以随时加载，用于预测。

结果可视化，使用中间层输出和结果展示，可以显示中间层的激活图与结果的直方图。

特征抽取，将网络中的任意层抽取出输出，可以实现对图片的特征描述。

最后，深度学习的易于上手，使得对于现实的一些图像处理问题都能简单去实践。