# 大众GP优化任务--脸型优化

# 1.反馈问题：

原图： 生成效果：



客户反馈以上生成**棱型脸**更合适但此处生成了**钻石型脸**：

资源示例如下：



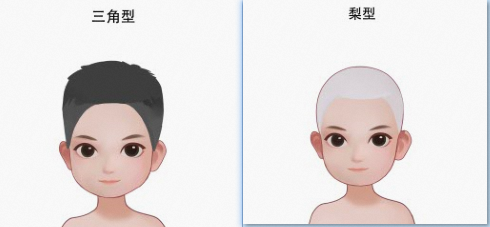
# 2.优化工作：

## 1）标签类别优化：

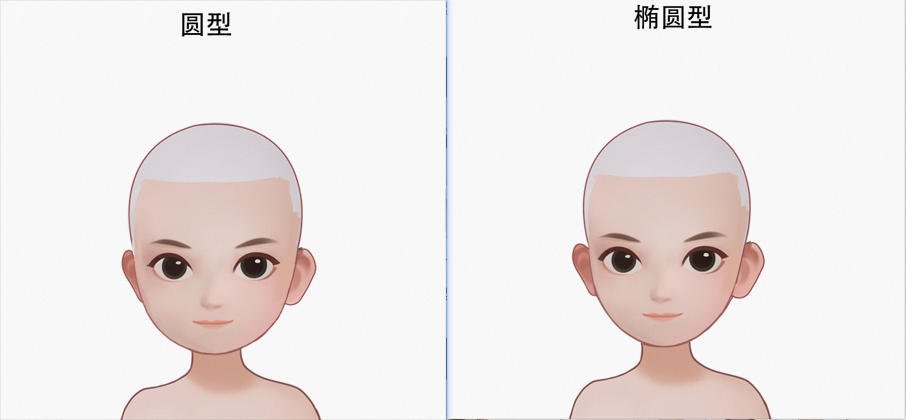
根据反馈问题和现有资产，由于传统算法在进行算法匹配时只会应用到**脸部下半部分的特征**，这主要是由于人有头发会导致脸的**上半部分有不同程度的遮挡**，将以下脸部下半部分的特征**相似度很高**的标签进行合并，具体如下：



**倒三角型**和**心型**合并为一种标签：**心型**



**三角形**和**梨型**合并为一种标签：**梨型**



**圆型**和**椭圆型**合并为一种标签：**圆型**

合并标签后，需要分类的脸型由十种变为以下7种，图示如下：



## 2）测试集制作：

根据融合后的标签，在网上寻找了不同类型的资源图片作为各个label的测试集，具体数量情况如下表：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 长方型 | 国字型 | 棱型 | 梨型 | 心型 | 圆型 | 钻石型 |
| 数量 | 22 | 15 | 11 | 23 | 20 | 14 | 18 |

测试集：[📎test\_data.zip](https://yuque.antfin.com/attachments/lark/0/2023/zip/23456869/1686122186318-93c140f4-117a-4d89-83bb-07042dbc6174.zip)

其中各个标签示例如下：

国字型



棱型



钻石型



梨型



长方型



心型



圆型



## 3）算法优化：

### a.解决bug

首先看下代码细节，脸型匹配为**遗留代码**，识别效果差可能是算法有一些问题，深究python版算法和其对应的数学方法后，发现以下问题：

public NDArray getFaceDistance(NDArray landmarks) {  
 ...   
 float Wface = computeDistance(lan1, lan31); //面宽对应的点  
 float Hface = computeDistance(land16, land51); //面高对应的点  
 float Wmidd = computeDistance(land8, land24); //下额的宽度  
 float Wxia = computeDistance(land12, land20); // 下巴的宽度  
 float Iface = Math.round(Hface / Wface); // 面部的比例,面宽/面高  
 float Imiddle = Wmidd / Wface;  
 float Ixia = Wxia / Wface;  
 NDArray arrayFaceRatio = manager.create(new float[]{Iface, Imiddle, Ixia});  
 NDArray consLeft = getFaceCosV3(faceBase = new int[]{0, 16},landFaceList = new int[]{1, 3, 5, 7, 9, 11, 13});  
 return NDArrays.concat(new NDList(arrayFaceRatio, consLeft), 0);

def get\_Face\_distance(self,img):  
 ...  
 WFace = compute\_distance(land1, land31) # 面宽  
 Hface = compute\_distance(land16, land51) # 面高  
 Wmidd = compute\_distance(land8, land24) # 下额的宽度  
 Wxia = compute\_distance(land12, land20) # 下巴的宽度  
 Iface = round(Hface / WFace,4) # 面部的比例,面宽/面高  
 Imiddle = round(Wmidd / WFace,4)  
 Ixia = round(Wxia/WFace,4)  
  
 cons\_zuo = get\_consin\_V3(landmarks\_parts,base\_=[0,16],land\_list = [1,3,5,7,8,9,11,13,14,15])  
 # 判断是否为尖下巴  
 face\_angle1 = cal\_ang(landmarks\_parts[15],landmarks\_parts[16],landmarks\_parts[17],cos=True)  
 face\_angle2 = cal\_ang(landmarks\_parts[14], landmarks\_parts[16], landmarks\_parts[18], cos=True)  
 face\_angle3 = cal\_ang(landmarks\_parts[13], landmarks\_parts[16], landmarks\_parts[19], cos=True)  
  
 return [Iface, Imiddle,Ixia,round(face\_angle1,4),round(face\_angle2,4),round(face\_angle3,4)] + cons\_zuo

#### 1）bug（java第七行，python第七行）

首先有一个明显bug，在java工程代码中，对于第七行的iface变量使用了math.round函数，直接将变量结果取整，导致参数几乎恒为1.0。对比移植前python代码使用的是round(Wmidd / WFace,4)，会保留4位有效数字，而不是恒为1.0，此时的iface变量才是有意义的，所以在java算法中**将此处进行对齐**

#### 2）细节差异a（java第11行，python第11行）

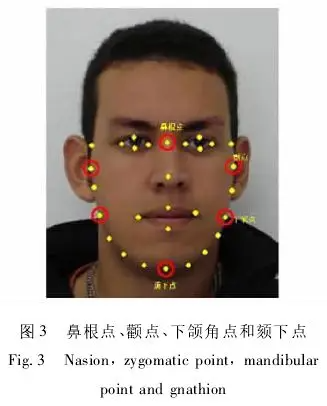
在python原版代码中，计算左脸关键点角度是利用了[1,3,5,7,8,9,11,13,14,15]等关键点，相比java工程中的[1, 3, 5, 7, 9, 11, 13]中多了8，14，15这三个关键点，**此处造成多大差异存疑，暂时未进行对齐**

#### 3）细节差异b（python第13，14，15行）

在python原版代码中，额外计算了下巴处关键点的三个角度值，但是在java工程中，移植者省去了这块内容，**此处造成多大差异存疑，暂时未进行对齐**

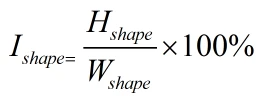
#### 4）细节差异c

在原版knn确定脸型的算法中，描述如下：

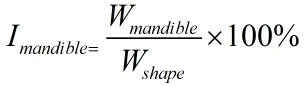


利用图中红色圈出来的点可以计算面宽 Wshape、下颌宽 Wmandible、形态面高 Hshape3 个测量参数。面宽去左右颧点的欧氏距离，下颌宽取左右下颌角点欧氏距离，形态面高取鼻根点与颏下点间欧氏距离【3】【4】。

形态面指数：



颧下颌宽指数：



将以上这些参数作为脸型分类的特征进行分类。

分类过程：1、利用 ASM 获取形状特征点集；2、计算形态面指数和颧下颌宽指数，若形态面指数大于 0.915 归为长脸；3、归为长脸的样本若颧下颌宽大于 0.875 归为方圆下巴长脸，否则归为尖下巴长脸。面型指数小于 0.915 的样本，需要将形态面指数和颧下颌宽指数和样本库中的样本做比对，求取 k 个近邻得到脸型类别。

形态面指数指代码中的Iface变量，颧下颌宽指代码中的Imiddle，这两个值在原文中是用于**根据阈值做分类**的，但是无论在python版代码和java工程代码中，其用法都是通过**欧氏距离的值进行匹配，**这就意味着这两种参数和后面的半边脸角度参数的权重是同一水平，**此处造成多大差异存疑，暂时未进行对齐**

### b.策略优化

根据反馈的问题，对比棱形脸和钻石型脸的主要差距来自于下巴部分，算法侧修改了策略，弱化了先验信息为male时的尖下巴类脸型的匹配得分。其中圆下巴和尖下巴的分类如下：

|  |  |
| --- | --- |
|  | labels |
| 尖下巴 | 心型、钻石型、国字型 |
| 圆下巴 | 圆型、长方型、棱型、三角型 |

### c.定量测试

优化后和优化前测试指标如下所示(top-1)：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 长方型 | 国字型 | 棱型 | 梨型 | 心型 | 圆型 | 钻石型 |
| 优化前 | 4/22 = 0.182 | 2/15 = 0.133 | 1/11 = 0.091 | 7/23= 0.304 | 14/20 = 0.70 | 7/14 = 0.500 | 9/18 = 0.500 |
| 优化后 | 14/22 = 0.636 | 6/15 = 0.400 | 8/11 = 0.727 | 17/23 = 0.739 | 9/20 = 0.45 | 7/14 = 0.500 | 12/18 = 0.667 |

### d.定性测试

优化前后定性指标：

测试背景：主要使用合肥斑马同学自采的数据集，对边前后脸型差异

|  |  |  |
| --- | --- | --- |
| 当前照片 | 优化前预测 | 优化后预测 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |