CS159-Programming-ZH

Report for
“Artist’s Meme”
（艺术家的表情包）

Group name: 我们是表情包的生产者

Members: 刘嘉宜，千芊，李中华

Teacher: Bao Yang

08/06/2019

目录

1 背景

1.1 选题原因

随着网络即时聊天软件的不断普及与发展，人们逐渐对“体现自己风格”的头像与有趣美观的表情包有了越来越大的需求。但目前网上流行的表情包重复性强，风格趋同，缺乏趣味性；而头像如果直接来自于照片，美感上就略有欠缺，如果直接采用现成图片，就有无法体现个人特色。因此我们寻求能够对照片深入加工的方法，使用户有机会便捷地亲自制作表情包，或得到有特点、有艺术感的的头像，以期取得惊艳效果。

1.2 项目介绍

如图1所示，首先识别一张照片是否为人脸，如果不是，则显示“无人！”；如果是，则继续通过opencv识别该人脸是否微笑。如果微笑，则配上系统自带或用户自行输入的表情包文字制作搞笑表情包；如果不微笑，则基于keras搭建风格转移平台,实现照片与经典艺术作品的图像风格迁移，制作艺术风格头像。

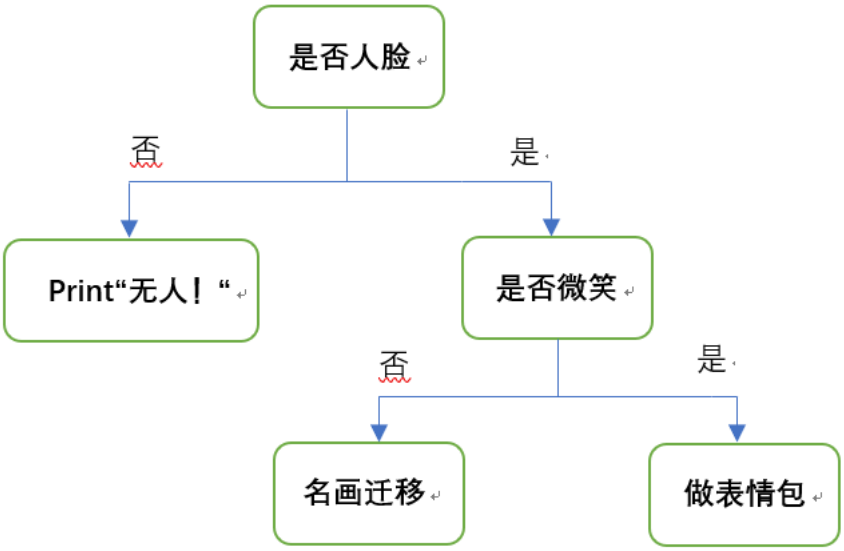


图1：项目流程示意图

1.3 Library介绍

1.3.1 OpenCV（Haar分类器原理）

Haar分类器又称Viola-Jones识别器，是Viola和Jones分别在2001年的《Rapid Object Detection using a Boosted Cascade of Simple Features》和2004年的《Robust Real-Time Face Detection》中提出并改进的。Haar分类器由 Haar 特征提取、离散强分类器、强分类级联器组成。核心思想是提取人脸的 Haar 特征，使用积分图对特征进行快速计算，然后挑选出少量关键特征，送入由强分类器组成的级联分类器进行迭代训练[1]。

Haar矩形特征是用于物体检测的数字图像特征。这类矩形特征模板由两个或多个全等的黑白矩形相邻组合而成，而矩形特征值是白色矩形的灰度值的和减去黑色矩形的灰度值的和，矩形特征对一些简单的图形结构，如线段、边缘比较敏感。如果把这样的矩形放在一个非人脸区域，那么计算出的特征值应该和人脸特征值不一样，所以这些矩形就是为了把人脸特征量化，以区分人脸和非人脸。

如图2所示，Haar特征有3种基本类型，分别是边缘特征、线性特征、中心特征和对角线特征，组合成特征模板。Haar特征值反映了图像的灰度变化情况。例如：脸部的一些特征能由矩形特征简单的描述，如：眼睛要比脸颊颜色要深，鼻梁两侧比鼻梁颜色要深，嘴巴比周围颜色要深等。但矩形特征只对一些简单的图形结构，如边缘、线段较敏感，所以只能描述特定走向（水平、垂直、对角）的结构。

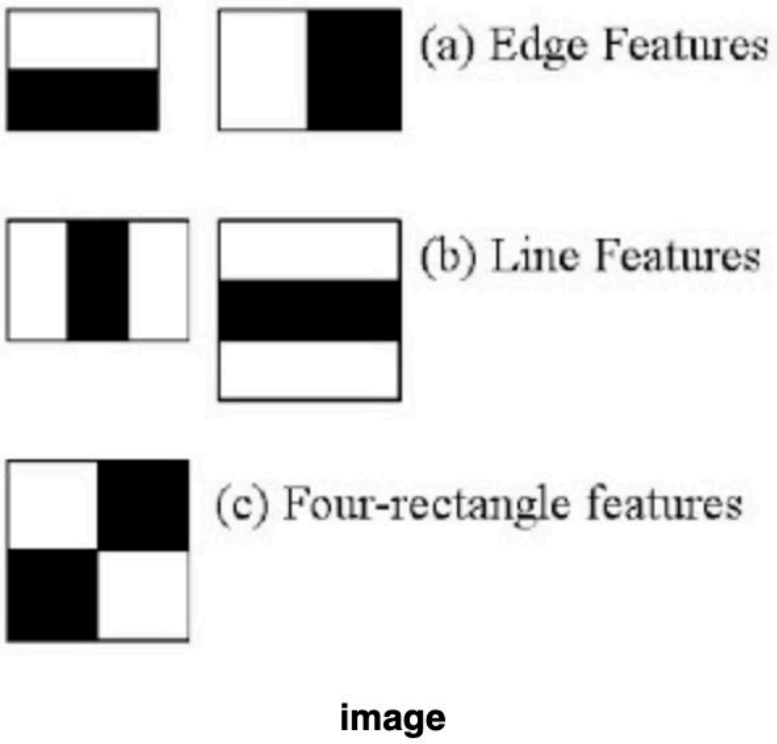


图2：Haar特征的基本类型示意图

确定 Haar 特征之后，在用积分图对图片 Haar-like 特征进行计算，求出图像中所有区域像素和。积分图主要的思想是将图像从起点开始到各个点所形成的矩形区域像素之和作为一个数组的元素保存在内存中，当要计算某个区域的像素和时可以直接索引数组的元素，不用重新计算这个区域的像素和，从而加快了计算。[2]

1.3.2 Keras风格迁移（卷积神经网络）

风格迁移则是借助深度学习技术实现，背后基础是卷积网络超强的图像特征提取能力。基于格拉姆（Gram）矩阵定义内容与风格损失函数，提取图片特征值并进行匹配。

这里我们所用的内容损失函数为：

$$\mathcal{L}_c(p,x,l) = \frac{1}{2}\sum_{i,j}(F_{ij}^l - P_{ij}^l)^2$$

其中，F和P是两个矩阵，包含N个行和M个列N是给定层L中的过滤器数量，M是给定层I的特征图谱（高度乘以宽度）中空间元素的数量，F包含给定层L中X的特征表示，P包含给定层L中p的特征表示。

对于风格损失，我们引用格拉姆函数，定义如下：

$$G^l = F^l(F^l)^T.$$

$$E_l = \frac{1}{4N_l^2M_l^2}\sum_{i,j}(G_{ij}^l - A_{ij}^l)^2,$$

其中A是风格照片a的格拉姆矩阵，G为生成照片x的格拉姆矩阵。

2 代码解释

2.1 人脸识别

首先导入所需的包和模块文件（事先已pip install），主要有opencv（用来实现人脸识别和笑脸识别），numpy，pandas，PIL（用来实现向照片添加中文字），keras（用来实现风格迁移的机器学习），time（用来计时），random（用来实现随机从语料库中选取配字）

```
import cv2
import numpy as np
import pandas as pd
from PIL import ImageFont, ImageDraw, Image#用PIL实现添加中文字
from keras import backend as K
from keras.preprocessing.image import load_img, img_to_array
from keras.applications import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.layers import Input
from scipy.optimize import fmin_l_bfgs_b
import time
import random
```

然后定义人脸检测器，如下图所示。使用opencv中自带的模块文件，设置特征的最大、最小检测窗口，最后返回框出人脸的矩形坐标。其中，cv2.cvtColor()函数将图片转换为灰度图，这是因为图像处理时，有些图像可能在 RGB 颜色空间信息不如转换到其它颜色空间更清晰，所以要先转换成灰度图；而detectMultiScale()函数用来检测人脸，它可以检测出图片中所有的人脸，并将人脸用vector保存各个人脸的坐标、大小（用矩形表示），函数由分类器对象调用。

```
# 人脸检测器 定位haarcascade_frontalface_alt.xml
faceCascade = cv2.CascadeClassifier("/Users/maggi/Anaconda3/Lib/site-packages/cv2/data/haarcascade_frontalface_alt.xml")
def detectFaces(image_name):
  img = cv2.imread(image_name)
  #定位haarcascade_frontalface_default.xml
  face_cascade = cv2.CascadeClassifier("/Users/maggi/Anaconda3/Lib/site-packages/cv2/data/haarcascade_frontalface_default.xml")
  if img.ndim == 3:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  else:
    gray = img #if语句：如果img维度为3，说明不是灰度图，先转化为灰度图gray，如果不为3，也就是2，原图就是灰度图

  faces = face_cascade.detectMultiScale(gray, 1.2, 5)#1.2和5是特征的最小、最大检测窗口，它改变检测结果也会改变
  result = []
  for (x,y,width,height) in faces:
    result.append((x,y,x+width,y+height))
  return result
```

然后调用定义好的人脸检测函数对目标图片1.jpg进行检测，如果len(result)>0，也就是说框出的矩形存在，说明检测到人脸。

```
if __name__ == '__main__':
    result=detectFaces('1.jpg')
if len(result)>0:
```

如果没有检测到人脸，则显示“无人！”

```
else:
    print("无人!")
```

2.2 笑脸识别

在确认照片中存在人脸后，进行笑脸识别。首先调用笑脸检测模块文件，读取目标图片，将图片转换为灰度图后，用detectMultiScale() 这个函数来对图像进行多尺度检测。

```
if len(result)>0:
    # 笑脸检测器
    smileCascade = cv2.CascadeClassifier("/Users/maggi/Anaconda3/Lib/site-packages/cv2/data/haarcascade_smile.xml")
    img = cv2.imread("1.jpg")
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # 首先检测人脸
    faces = faceCascade.detectMultiScale(gray,1.3,5,minSize=(55,55))
```

然后对人脸进行笑脸检测，类似于之前的人脸检测，如果len(result)>0，就说明检测到笑脸，则进入制作表情包环节；如果不是这样，则说明未检测到笑脸，则进入风格迁移环节。

```
    # 找出每一个人脸，提取出人脸所在区域
    for (x, y, w, h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        # 对人脸进行笑脸检测
        smile = smileCascade.detectMultiScale(
            roi_gray,
            scaleFactor= 1.16,
            minNeighbors=35,
            minSize=(25, 25)
            )
        # 检测笑脸
        result=[]
        for (x2, y2, w2, h2) in smile:
            result.append((x2,y2,x2+w2,y2+h2))
        if len(result)>0:####################如果检测出笑脸，则制成表情包，配文字############
```

2.3 制作表情包

如果检测出笑脸，则制成表情包配上文字。首先询问用户是想要自行输入表情包配字还是使用我们预设的文字。如果用户输入1，则用户自行输入表情包配字；如果用户输入2，则从我们预设的20句表情包常用语中，通过random随机选取一句给图片配上文字。如下图所示。

```
if len(result)>0:##################如果检测出笑脸，则制成表情包，配文字#####################
    tag_if = input("请问您是否想要自行输入表情包配字？“是”请输入1，“否”请输入2：")
    if tag_if == "1":
        word = input("请输入您想要的表情包配字：")
    elif tag_if == "2":
        word_lib=[
            '无与伦比的快乐',
            '我就笑笑不说话',
            '？？？？？',
            '不要欺负我这种年轻又好看的人',
            '你是魔鬼吗',
            '我可能就是来大学搞笑的',
            '神秘的微笑',
            '我没有笑',
            '偷偷开心',
            '你这只猪蛮有意思的',
            '蛮好玩惹',
            '露出本宝宝八颗牙齿的微笑',
            '哈哈，我不想活啦',
            '好气哦，但还是要保持微笑',
            '哈哈哈哈哈哈嗝',
            '拜拜了您嘞',
            '你想把我笑死 好继承我的QQ黄钻吗',
            '写程序真开心',
            '你怎么这个亚子',
            '你的脑回路里净是路障'
            ]#语句库，可更改
        word=word_lib[random.randint(0,19)]
```

接着设置字体格式与大小、位置。在具体实施过程中，原本使用cv2.putText方法，但是发现这样无法使用中文字体配字，所以改成用PIL包中ImageFont等模块来实现中文配字；设置文字位置时，采用(x-50,y-100)的坐标，这样使得配字大约在图片上方中间的位置；采用白色(255,255,255)作为文字颜色；然后采用cv2.imshow展示配完文字的图片，注意在imshow后加入cv2.waitkey(0)使得图片可以一直处于展示状态。如下图所示。

```
                                                #原来的代码cv2.putText#cv2无法添加中文文字和导入字体文件，所以用PIL
fontpath = "font/msyh.ttc"#设置字体
font = ImageFont.truetype(fontpath, 36)# 36为字体大小
img_pil = Image.fromarray(img)
draw = ImageDraw.Draw(img_pil)
#绘制文字信息<br># (x,y-7)为字体的位置，(255,255,255)为白色，(0,0,0)为黑色
draw.text((x-50,y-100),  word, font = font, fill = (255, 255, 255))
img = np.array(img_pil)



cv2.imshow('Smile', img)
cv2.waitKey (0)
```

如下图所示窗口。

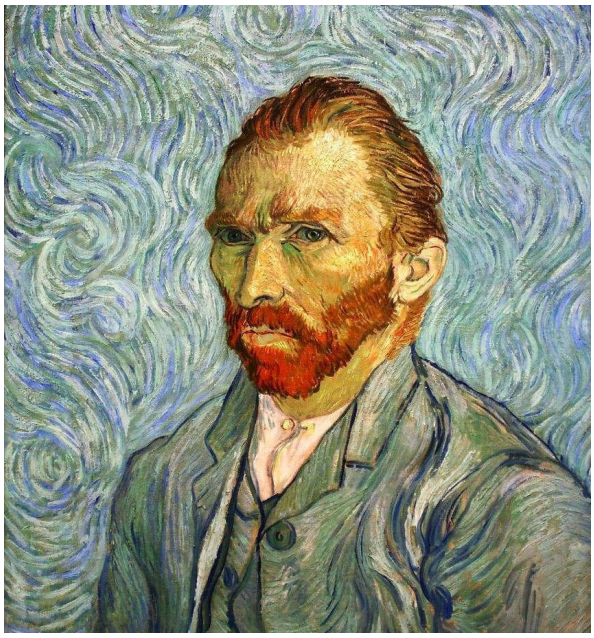
此时询问用户是否想要保存这张表情包。如果用户输入1，则保存表情包为smile_emoji.jpg在设定路径中；如果用户输入2，则不保存表情包。上图所示窗口可以直接按x关闭。

```
save_if = input("请问您是否想要保存这张图片？“是”请输入1，“否”请输入2：")
if save_if == "1":
    cv2.imwrite("C:/Users/maggi/Desktop/Python Project/outcome/smile_emoji.jpg",img)
    cv2.waitKey (0)
    cv2.destroyAllWindows()
elif save_if == "2":
    cv2.waitKey (0)
    cv2.destroyAllWindows()
```

最后得到的表情包如下图所示。

2.4 风格迁移

如果在2.3的笑脸识别中没有检测出笑脸，即len(result)<=0，则进行风格迁移。这里我们使用梵高的一幅画像作为目标迁移图片，命名为fangao.jpg；使用游乐王子的一张照片作为原图片，命名为1.jpg。



确定要进行分割迁移的原图、风格图、以及生成图的目标路径与文件名：（这里已将文件保存在当前工作路径中）

```
cImPath = '1.jpg' #（jpg格式图片，原图）
sImPath = 'fangao.jpg' #（jpg格式图片，风格图）
genImOutputPath = 'output.jpg' #（输出图片）
```

然后首先对图片做预处理：

目标图片的大小调整为长512，宽512。

```
targetHeight = 512
targetWidth = 512
targetSize = (targetHeight, targetWidth)
```

我们这里为了后面的优化，将glm0初始化为 float64。而且为了避免GPU的内存错误，我们将cImArr和slmArr保持为float32。

```
cImage0rig = Image.open(cImPath)
cImageSize0rig = cImage0rig.size
cImage = load_img(path=cImPath, target_size=targetSize)
cImArr = img_to_array(cImage)
cImArr = K.variable(preprocess_input(np.expand_dims(cImArr, axis=0)), dtype='float32')

sImage = load_img(path=sImPath, target_size=targetSize)
sImArr = img_to_array(sImage)
sImArr = K.variable(preprocess_input(np.expand_dims(sImArr, axis=0)), dtype='float32')

gIm0 = np.random.randint(256, size=(targetWidth, targetHeight, 3)).astype('float64')
gIm0 = preprocess_input(np.expand_dims(gIm0, axis=0))

gImPlaceholder = K.placeholder(shape=(1, targetWidth, targetHeight, 3))
```

接下来是风格迁移中内容损失函数的定义，内容损失的目标是确保生成的照片仍能保留原内容照片的“全局”风格。首先使用梯度下降算法定义层级特征：

```
def get_feature_reps(x, layer_names, model):
    featMatrices = []
    for ln in layer_names:
        selectedLayer = model.get_layer(ln)
        featRaw = selectedLayer.output
        featRawShape = K.shape(featRaw).eval(session=tf_session)
        N_l = featRawShape[-1]
        M_l = featRawShape[1]*featRawShape[2]
        featMatrix = K.reshape(featRaw, (M_l, N_l))
        featMatrix = K.transpose(featMatrix)
        featMatrices.append(featMatrix)
    return featMatrices
```

计算内容损失：

```
def get_content_loss(F, P):
    cLoss = 0.5*K.sum(K.square(F - P))
    return cLoss
```

计算风格损失：

```
def get_content_loss(F, P):
    cLoss = 0.5*K.sum(K.square(F - P))
    return cLoss

def get_Gram_matrix(F):
    G = K.dot(F, K.transpose(F))
    return G

def get_style_loss(ws, Gs, As):
    sLoss = K.variable(0.)
    for w, G, A in zip(ws, Gs, As):
        M_l = K.int_shape(G)[1]
        N_l = K.int_shape(G)[0]
        G_gram = get_Gram_matrix(G)
        A_gram = get_Gram_matrix(A)
        sLoss= sLoss+w*0.25*K.sum(K.square(G_gram - A_gram))/ (N_l**2 * M_l**2)
    return sLoss

def get_total_loss(gImPlaceholder, alpha=1.0, beta=10000.0):
    F = get_feature_reps(gImPlaceholder, layer_names=[cLayerName], model=gModel)[0]
    Gs = get_feature_reps(gImPlaceholder, layer_names=sLayerNames, model=gModel)
    contentLoss = get_content_loss(F, P)
    styleLoss = get_style_loss(ws, Gs, As)
    totalLoss = alpha*contentLoss + beta*styleLoss
    return totalLoss
```

加载训练模型：

```
tf_session = K.get_session()
cModel = VGG16(include_top=False, weights='imagenet', input_tensor=cImArr)
sModel = VGG16(include_top=False, weights='imagenet', input_tensor=sImArr)
gModel = VGG16(include_top=False, weights='imagenet', input_tensor=gImPlaceholder)
cLayerName = 'block4_conv2'
sLayerNames = [
                'block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                #'block5_conv1'
                ]

P = get_feature_reps(x=cImArr, layer_names=[cLayerName], model=cModel)[0]
As = get_feature_reps(x=sImArr, layer_names=sLayerNames, model=sModel)
ws = np.ones(len(sLayerNames))/float(len(sLayerNames))
```

由于时间关系，这里我们采用预先训练过的卷积神经网络模型VGG16而不是自己创建模型来匹配迁移函数。

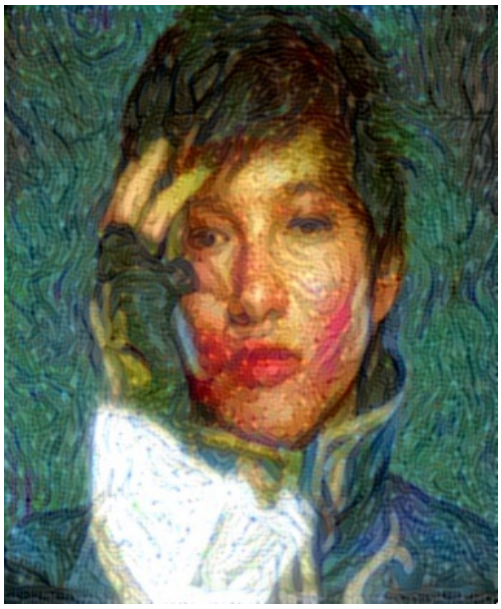
确定迭代次数（iteration），由于keras迁移算法是根据风格图片用损失与保留函数对原图进行一次次迭代而成，每一次迭代过程的耗时大约在230秒左右，而较少的迭代次数又使得迁移后的图片风格并不显著，因此我们综合考虑以上因素最终将迭代次数定为100次。

```
iterations = 100
```

最后是对迭代过程耗时的统计：（实际运行中一位组员的电脑上平均迭代1次230秒左右，另一位组员的电脑上平均迭代1次100秒左右）

```
x_val = gIm0.flatten()
start = time.time()
xopt, f_val, info= fmin_l_bfgs_b(calculate_loss, x_val, fprime=get_grad,
                                 maxiter=iterations, disp=True)
x0ut = postprocess_array(xopt)
xIm = save_original_size(x0ut)
end = time.time()
```

结果如下图所示。



3 局限与改进

3.1 局限性

代码略显繁琐。在笑脸检测中实际上包含人脸识别这一步，前面的一大段代码实际上可以合并到笑脸检测这里。

图片制作时间过长。一张图片进行风格迁移全过程需要200次迭代，一次花费近300秒，也即完成一张图片的名画风格迁移需要16小时左右，效率较低。

功能较为单一。仅以“是否微笑”作为表情包的判断标准过于武断，我们的制作初衷是根据微笑与否判断图片的严肃性，防止给严肃的图片自动匹配搞笑文字做成表情包，但事实上有些不微笑的图片也很适合制作表情包。

只能在固定的设备上进行图片加工。由于程序内涉及图片存储位置，无法满足每一位用户在自己的电脑上直接加工图片的要求。而且在未识别出人脸后更改图片的同时也需要更改路径。

制作表情包时，由于图片大小格式不同，导致配字可能偏离正中间，美观性较差。如图3所示，表情包配字出现在图片的左上角，并且窗口的画面比例失调。

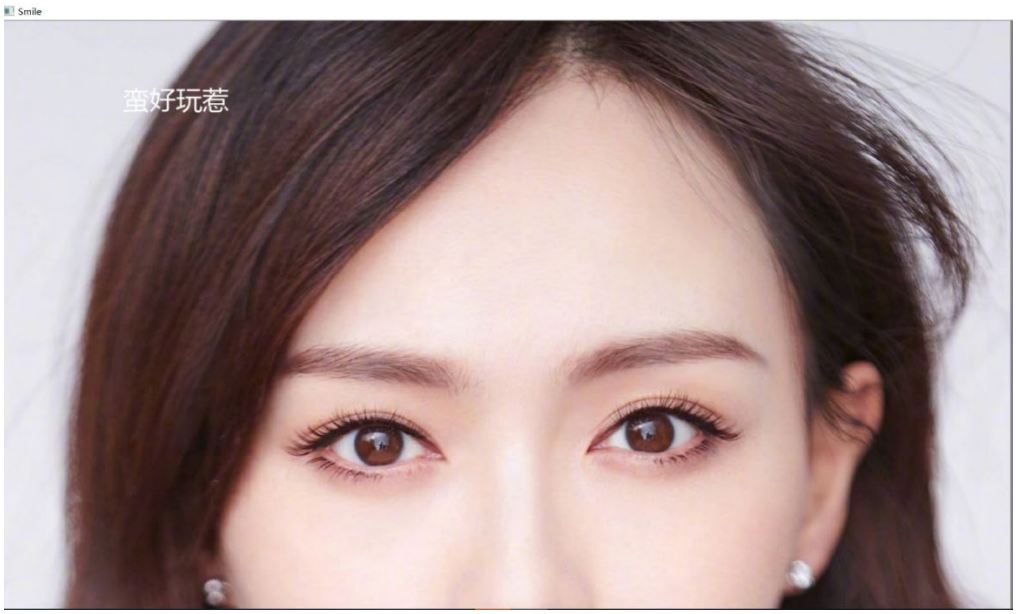


图3：配字位置示意图

3.2 优化设想

简化代码：将开头一大段人脸识别的代码并入笑脸检测中detectMultiScale()这一步之后，加入if，else结构。

缩减图片制作时间：加强电脑硬件设备。

功能的具体化、多样化：可以增加面部表情识别来使功能更加完善，分别检测不同的五官表情，并建立数据库判断照片中人的心情（例如快乐、惊讶、悲伤等），据此选择图片适配的表情包文字或进行相关风格迁移。

在不同设备进行图片加工：把图片存储的地址设置为用户输入的input，让用户可以输入自己电脑里图片的存储地址，这样即可直接复制程序实现多设备图片加工。

4 小结

我们根据自己使用社交软件的经验发现了制作表情包和艺术风格头像的需求，并通过python代码将其实现。值得注意的是，我们通过Github实现了组员共同产出、修改代码，方便快捷。组员在各种论坛、网站上寻找需要的相关代码并进行代码阅读与快速自学，不断调整方案，使问题既在我们力所能及的解决范围内，又使它有较强趣味性、完整度，尽可能地提高用户体验。虽然费了一番周折，但有机会通过自己所学的内容实现一个有趣的程序给我们带来了很大的成就感，也大大提高了我们对复杂程序的编程能力。总而言之，在这次作业中我们小组成员收获颇丰，大家都很期望在日后不断提升python编程能力，完成更多有趣、实用的程序。

感谢鲍杨老师的指导！

参考文献

[1] DUffiE D, SCHEICHER M, VUILLEMEY G. Central Clearing and Collateral Demand[J]. : 48.

[2] Justin Johnson, Alexandre Alahi, Li Fei-Fei，Perceptual Losses for Real-Time Style Transfer and Super-Resolution[J].