作业六 图像的空间变换

赵心怡 19307110452

开发环境介绍

开发环境： python 3.9

需要的库： opencv-python,

numpy,

matplotlib

可执行的代码在jupyter notebook文件中，逐行运行前请保证第三方包已全部安装。

（1）空间变换算法

空间变换算法主要包括：图像平移、镜像、缩放和旋转，通过线性代数中的齐次坐标变换来实现。

我们以图像的平移变换为例，变换后的坐标和原坐标的关系是：

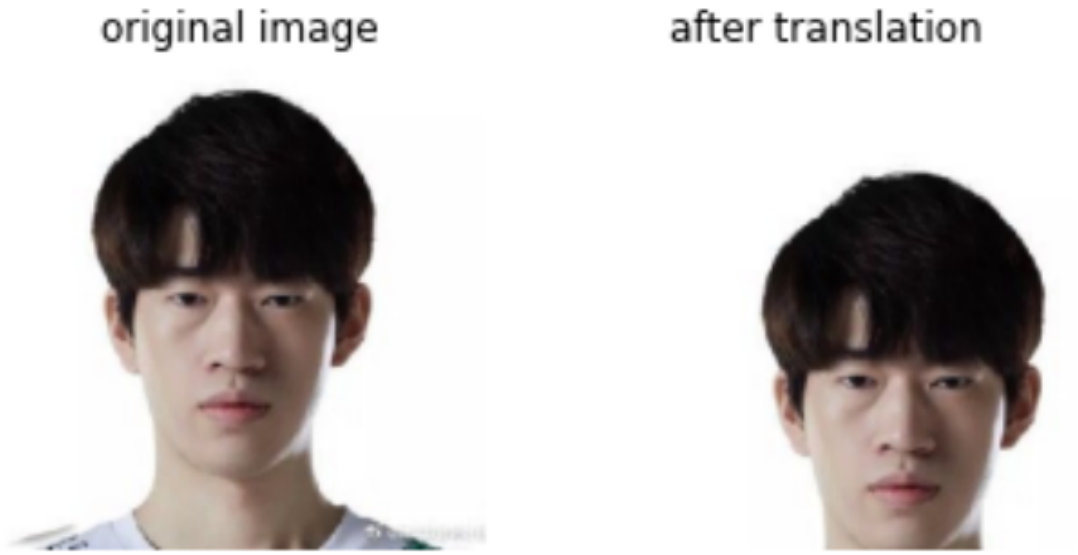
$$\begin{bmatrix} a(x,y) \\ b(x,y) \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

平移变换的代码如下：

```
def translation(img, X, Y):#平移变换代码，X，Y为竖直和水平方向平移的坐标
    height, width, channel = img.shape
    G_inv =np.eye(3)  # 初始化Gi为3*3的初始单位矩阵
    G_inv[0:2,2] = [X,Y] # 变换矩阵利用控制点在两张图上的位移得到
    newImg = img.copy()
    for x in range(height):
        for y in range(width): # 遍历每个点
            newImg[x,y]=255 #初始颜色设为白色
    for x in range(height):
        for y in range(width): # 遍历每个点
            Gi_X = np.dot(G_inv, np.array([x,y,1]))
            if Gi_X[0]<0 or Gi_X[1]<0 or Gi_X[0]>=height or Gi_X[1]>=width:#
如果不在图像范围内就不继续
                continue
            newImg[int(Gi_X[0]),int(Gi_X[1])]=img[x,y]#取整
    return newImg

```

设置X和Y 为50像素，变换结果如下：



全局仿射变换和局部仿射变换的区别在于，在局部仿射变换过程中，求解仿射变换矩阵的参考点是局部的。

为了优化第i个特征点的仿射矩阵，我们有

$$\Psi_i = \sum_{j=1}^{K} \omega_{ij} ||\vec{\epsilon}_j||^2$$
$$= \sum_{j=1}^{K} \omega_{ij} ||\vec{s}_j - \mathbf{A}_j \vec{r}_j||^2$$

和最优化过程：

$$\min_{\mathbf{A}_i} \Psi_i$$

其中，权重系数w由下面的式子决定

$$w_{ij} = exp(-||\vec{r}_i - \vec{r}_j||^2 / e^2)$$

e决定了特征点之间距离和权重系数之间的放缩关系。e等于无穷大时等效全局仿射变换。

局部仿射变换的代码：

```
def localAffine(target, U, target_U, epsilon):
    '''局部仿射函数，寻找X=T^{-1} (Y)的函数关系，返回一个字典position_map'''

    height, width, channel = target.shape
    position_map = defaultdict(lambda: np.zeros(3))  # 构造position_map映射字典，初始为三维零向量

    # 得到Gi
    G_inv = defaultdict(lambda: np.eye(3))  # 初始化Gi，字典内是3*3的初始单位矩阵
    num_control = len(U) # 选择的控制点个数
    for each_point in range(num_control):#改变（1，3）和（2，3）位置的元素为位移量
```

```
        G_inv[each_point][0:2,2] = U[each_point] - target_U[each_point] # 变换矩阵利用控制点在两张图上的位移得到

    for x in range(height):
        for y in range(width): # 遍历每个个点
            Y = (x,y)
            distance = np.sqrt(((np.array(Y) - target_U) ** 2).sum(axis=1)) # Y到target图中每一个控制点的距离
            w = 1 / (distance ** epsilon) if distance.all() else np.ones(num_control)
            # 如果不是控制点正常更新w，否则先随便赋一个值，下面再统一更新控制点
            w = w / w.sum()
            for i in range(num_control):
                Gi_X = np.dot(G_inv[i], np.array([x,y,1]))
                position_map[Y] += w[i] * Gi_X    # 利用w对Gi加权
            position_map[Y] = position_map[Y][:2]     # 把向量最后的1去除

    for i in range(num_control):
        position_map[tuple(target_U[i])] = U[i] # 控制点Y的映射就是X
    return position_map
```

（2）将上述空间变换应用于实现基于反向的图像变换过程，从而实现图像甲到图像乙的图像变换。

作业的基本算法内容参考课堂上讲解和课件。

反向变换是从图像的坐标（x,y)出发，经过矩阵一系列变换，得到在原始图像中的对应坐标（u，v), 后向映射中，x,y是整数而u,v一般是浮点数。

遍历变换后图像的坐标，变换到原始图像空间中的某一位置，通过插值获得具体的像素值。

之前已经实现过双线性插值，因此这里就直接写在函数里。

代码如下：

```
# 反向变换
def backwardTrans(source, target, position_map):
    height,width = target.shape[0],target.shape[1]

    # 对每个像素点标志是否已经填过
    is_paint = np.zeros((height,width),dtype='uint8')
    print(is_paint.shape)
    '''根据变换函数position_map完成反向图变换'''

    newImg = np.zeros_like(target)   # 变换后的target
    for i in range(height):
        for j in range(width): # 遍历target每一个像素点
            X = position_map[i, j] # source中对应的点（精确位置）
            u, v = X - X.astype(int)
            neigh_i, neigh_j = X.astype(int) # 得到附近像素点和垂直水平距离u,v
            # 进行插值，得到X位置的像素值，也就是newImg在Y处像素值
            if neigh_i < 0 or neigh_j < 0 or neigh_i+1 >source.shape[0]-1 or neigh_j+1 >source.shape[1]-1:
                is_paint[i,j]=1
                continue #如果这个点的位置出界了就不填
            newImg[i,j] = (1 - u) * (1 - v) * source[neigh_i, neigh_j] + (1 - u) * v * source[neigh_i,neigh_j + 1] +\
```

```
                    u * (1 - v) * source[neigh_i +1,neigh_j] +  u * v *
source[neigh_i + 1,neigh_j + 1]#用双线性插值的方法获得点的像素


    # 对没有填的区域进行修复
    newImg = cv2.inpaint(newImg, is_paint, 13, cv2.INPAINT_NS)

    return newImg.astype(np.uint8)  # 变成像素需要的int
```

有一个细节是选取图像点的时候，坐标默认是（横坐标，纵坐标）即对应的是列和行，与图像的索引（行，列）是反过来的，所以在选图像的特征点的时候需要转置一下。

图像预处理

前后的两张图片可能会存在尺寸不一致的问题，可以先用resize将模板图改变大小，这个操作可能会导致比例失调，但是在选取控制点做反向变换后的影响并不大。

我们的代码包含了少量GUI方便与用户进行交互，方便更好地选取控制点的位置，将选点行为如选取眼睛，鼻子，嘴巴等部位进行记录保存在数组中。特征点的数量取决于使用者自己的偏好。

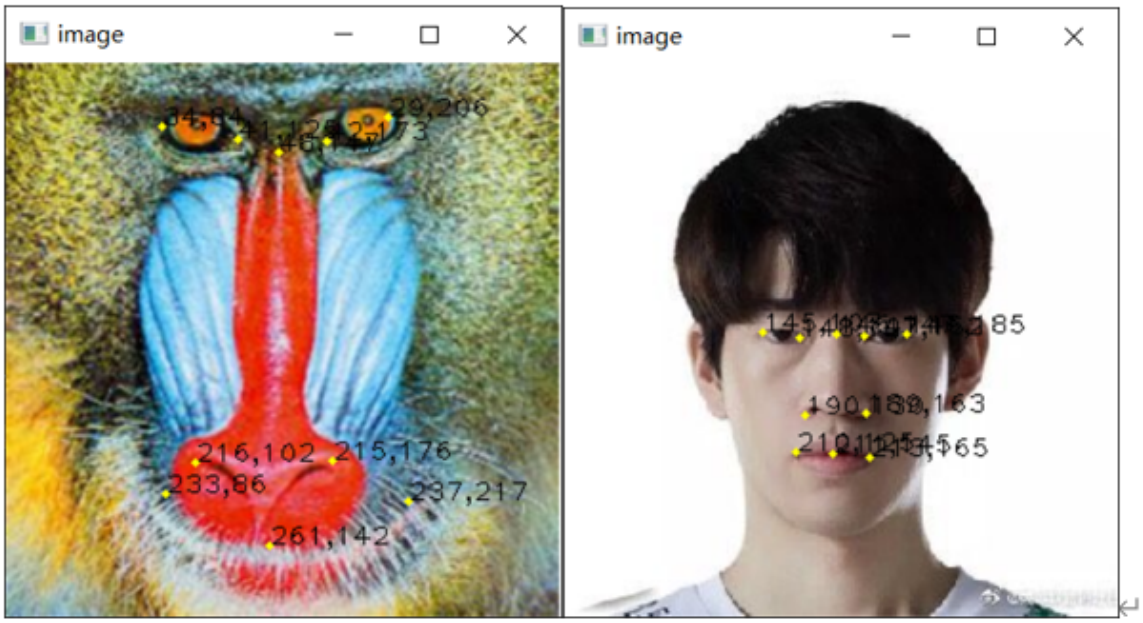
具体代码如下：

```
#获取坐标轴
source = cv2.imread("xiaohu.jpg")     # 人脸
img = cv2.imread('dog.jpg')
target_list=[]
img =cv2.resize(img, source.shape[:2])
def on_click(event, x, y):#定义鼠标点击事件
    if event == cv2.EVENT_LBUTTONDOWN:
        xy = "%d,%d" % (y,x)
        target_list.append([y,x])#坐标转置后存在列表里
        cv2.circle(img, (x, y), 1, (0, 0, 255), thickness=-1)#标出来
        cv2.putText(img, xy, (x, y), cv2.FONT_HERSHEY_PLAIN,
                    1.0, (0, 0, 0), thickness=1)#print出具体的坐标
        print(y,x)
        cv2.imshow("image", img)


cv2.namedWindow("image")
cv2.setMouseCallback("image", on_click)
cv2.imshow("image", img)
cv2.waitKey(0)
print(target_list)
```

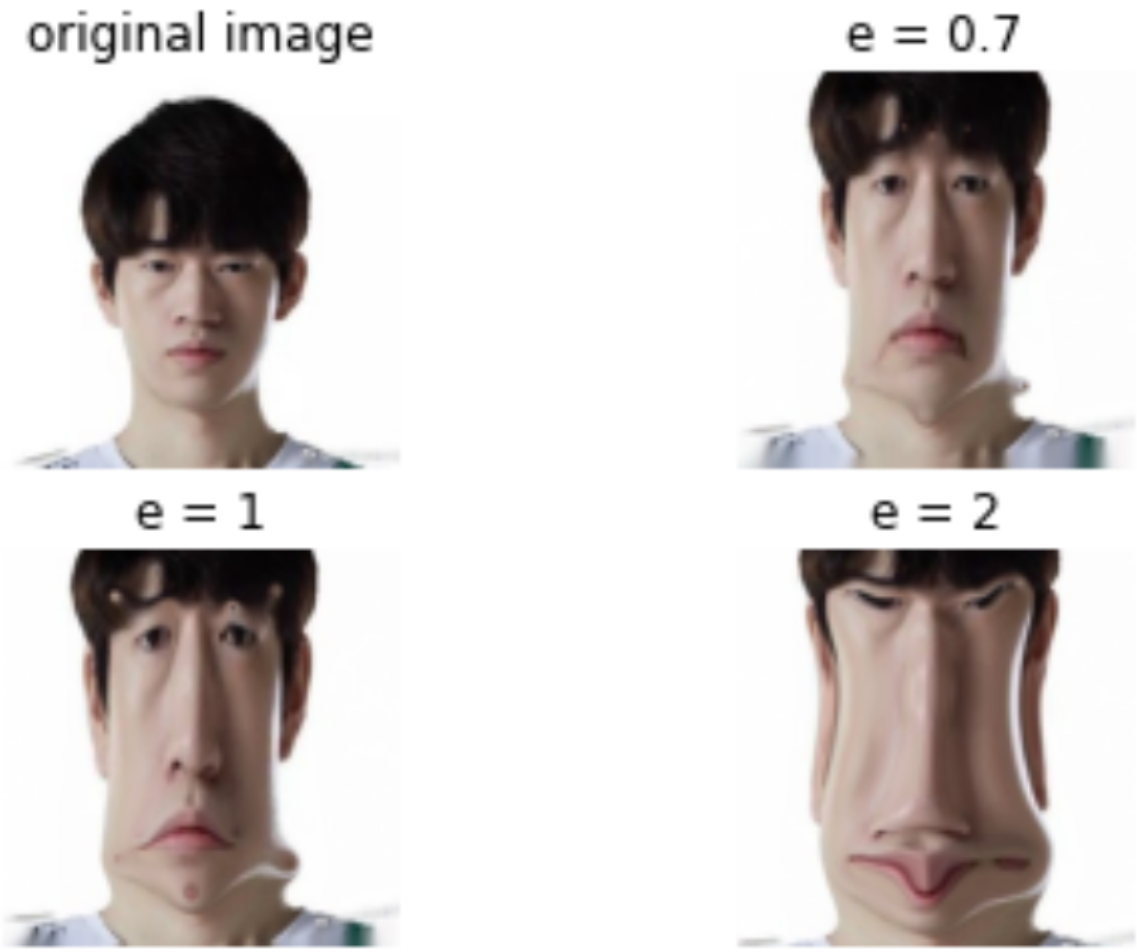
我们用对比度高的控制点的颜色让它和背景颜色的对比度更清晰。浮动数字可能在图像不是很清晰，但代码可以自动保存标记的点位置，不需要从图中读点。

实现的效果如下：



我们人工选取了控制点的坐标，保存在target_list列表中。

最终的图像处理结果：



可以看到，将e修改成不同大小会很大影响最后的输出结果。可以看到衰减指数越大，图像扭曲越明显。可能与选控制点的不准确有关，最终的图像五官会有略微的不对称。