计算机视觉

期末大作业

姓名：郝裕玮

班级：计科1班

学号：18329015

目录

[1 实验环境 3](#_Toc108206353)

[1.1 Python版本 3](#_Toc108206354)

[1.2 开发环境 3](#_Toc108206355)

[2 Seam Carving 3](#_Toc108206356)

[2.1 问题内容 3](#_Toc108206357)

[2.2 算法思路&代码分析 4](#_Toc108206358)

[2.3 实验结果 10](#_Toc108206359)

[3 Segmentation 13](#_Toc108206360)

[3.1 问题内容 13](#_Toc108206361)

[3.2 算法思路&代码分析 14](#_Toc108206362)

[3.3 实验结果 21](#_Toc108206363)

[4 Dimensionality Reduction—PCA 23](#_Toc108206364)

[4.1 问题内容 23](#_Toc108206365)

[4.2 问题1——算法思路&代码分析 24](#_Toc108206366)

[4.3 问题1结果展示 28](#_Toc108206367)

[4.4 问题2结果展示 30](#_Toc108206368)

[5 实验感想 30](#_Toc108206369)

# 1 实验环境

## 1.1 Python版本

Python版本：3.8.5

## 1.2 开发环境

开发工具：Jupyter Notebook (anaconda 3)

# 2 Seam Carving

## 2.1 问题内容

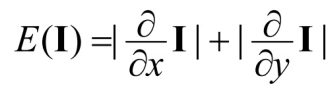
结合 “Lecture 6 Resizing“ 的 Seam Carving 算法，设计并实现前景保持的图像缩放，前景由 gt 文件夹中对应的标注给定。要求使用“Forward Seam Removing”机制，X，Y 方向均要进行压缩。压缩比例视图像内容自行决定（接近 1-前景区域面积/（2\*图像面积）即可）。每一位同学从各自的测试子集中任选两张代表图，将每一步的 seam removing 的删除过程记录，做成 gif 动画格式提交，测试子集的其余图像展示压缩后的图像结果。

## 2.2 算法思路&代码分析

Seam Carving的算法思路如下：

（1）计算图像能量图

能量图一般是图像像素的梯度模值，为了简化计算可先转换成灰度图像，然后直接采用如下公式（直接用x、y方向上的差分取绝对值，然后相加）



该部分代码如下（包含分析）：

#计算梯度能量

def energy(img):

    #原图转为灰度图

    img\_gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

    #利用Sobel算子进行图像梯度计算

    #分别计算x方向和y方向的一阶导数

    x = cv2.Sobel(img\_gray,cv2.CV\_64F,1,0,ksize=3)

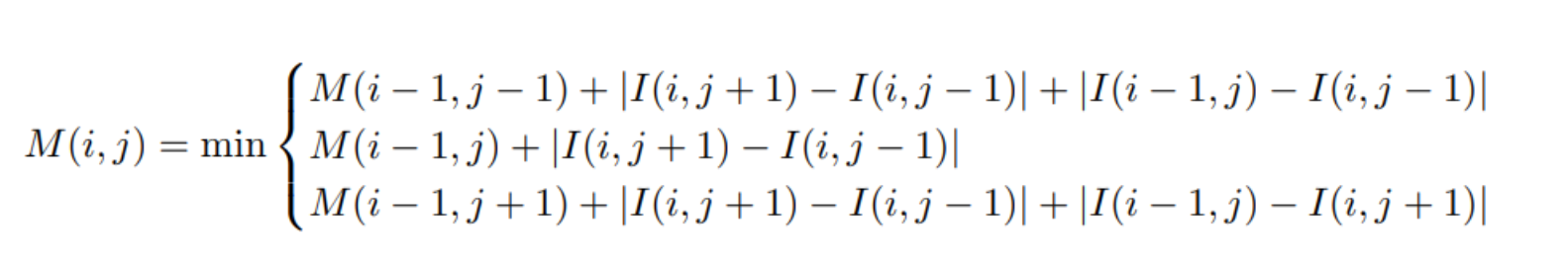
    y = cv2.Sobel(img\_gray,cv2.CV\_64F,0,1,ksize=3)

    #根据公式可知，能量值为x方向和y方向的一阶梯度的绝对值之和

    return cv2.add(np.absolute(x), np.absolute(y))

（2）寻找最小能量线

题目要求使用Forward Seam Removing，公式如下：



根据如上公式，可对第一问的能量图进行更新，代码如下所示（包含分析，见下页）：

#从垂直方向上计算每点的能量

def Calculate\_Horizontal\_Energy(energy,img\_gt):

    #图像高度和宽度

    height, width = energy.shape[:2]

    #用于存储每点能量

    energy\_map = np.zeros((height, width))

    #若当前坐标在前景范围内则设置其能量为1e6

    for i in range(height):

        for j in range(width):

            if img\_gt[i,j,0] > 200:

                energy\_map[i,j] = 1e6

    #根据Forward Seam Removing机制计算能量

    for i in range(1, height):

        for j in range(width):

            up, down = (i-1) % height, (i+1) % height

            left, right = (j-1) % width, (j+1) % width

            #若当前坐标在前景中为黑色像素点(避开白色像素点)，则直接使用公式更新能量

            if img\_gt[up,j,0] == 0:

                cost\_middle = np.abs(energy[i,right] - energy[i,left])

            #若为白色像素点则设置为1e6防止删除接缝线时接触到白色像素点

            else:

                cost\_middle = 1e6

            if img\_gt[up,left,0] == 0:

                cost\_left = np.abs(energy[up,j] - energy[i,left]) + cost\_middle

            else:

                cost\_left = 1e6

            if img\_gt[up,right,0] == 0:

                cost\_right = np.abs(energy[up,j] - energy[i,right]) + cost\_middle

            else:

                cost\_right = 1e6

            #根据公式找出能量最小的组合，并更新能量图

            cost = np.array([cost\_middle, cost\_left, cost\_right])

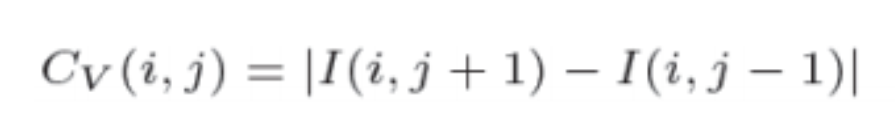
            M = cost + np.array([energy\_map[up][j], energy\_map[up][left], energy\_map[up][right]])

            min\_index = np.argmin(M)

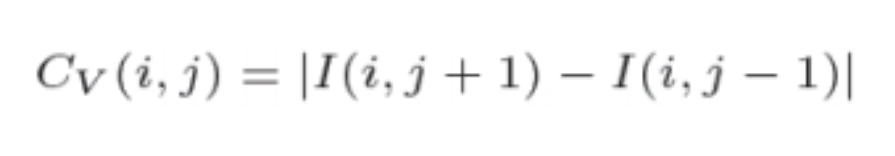
            energy\_map[i,j] = M[min\_index]

    return energy\_map

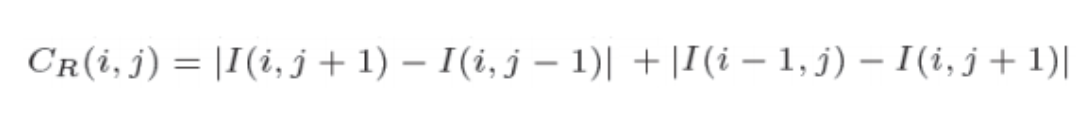
其中cost\_middle对应公式为：



cost\_left对应公式为：



cost\_right对应公式为：



（3）找出最小能量线

以垂直方向能量线为例：

从最后一列开始往前回溯，先从最后一列能量最小的开始回溯，且下一列需要选择的位置当前列能量最小的位置的八连通区域（垂直方向即当前位置的左边一列的上中下三个位置），依次回溯，即可找出能量最小线。

代码如下所示（包含分析）：

#连接垂直方向最小能量线

def Find\_Vertical\_EnergyLine(img,energy\_map):

    #图像高度和宽度

    height, width = energy\_map.shape[0], energy\_map.shape[1]

     #当前行的最小能量值所在位置(height)

    cur = 0

    SeamLine = []

    #从最后一行开始往前回溯

    for i in range(height - 1, -1, -1):

        #current\_row存储当前所在行的所有元素(每个元素存储该点的能量值)

        current\_row = energy\_map[i, :]

        #如果处于最后一行，则直接找出最低能量值所在位置即可

        #argmin:找出最小值的下标

        if i == height - 1:

            cur = np.argmin(current\_row)

        #若不在最后一行，则需要从上一行最小能量值的八连通区域中找出能量最小值

        #需要筛选的八连通区域为上一行最小能量值的上边一行(即当前行)的左、中、右三个位置

        else:

            #更新左中右三个位置的能量值

            if cur - 1 >= 0:

                left = current\_row[cur - 1]

            else:

                left = 1e6

            middle = current\_row[cur]

            if cur + 1 <= width - 1:

                right = current\_row[cur + 1]

            else:

                right = 1e6

             #比较三者大小，根据最小值来对当前列的最小能量值的位置进行更新

            if left == min(left, middle, right):

                if cur == 0:

                    cur = 0

                else:

                    cur = cur - 1

            if middle == min(left, middle, right):

                cur = cur

            if right == min(left, middle, right):

                if cur == width - 1:

                    cur = cur

                else:

                    cur = cur + 1

        SeamLine.append([cur, i])

    return SeamLine

（4）删除Seam接缝线（以（3）中的垂直方向为例）

将（3）中的最小能量线对应的像素删除即可，代码如下所示（包含分析）

#移除垂直方向最小seam线

def Remove\_Vertical\_EnergyLine(img, img\_gt,seam):

    #移除原图的Seam线

    height, width, depth = img.shape

    removed\_img = np.zeros((height, width - 1, depth), np.uint8)

    for (y, x) in seam:

        removed\_img[x, 0:y] = img[x, 0:y]

        removed\_img[x, y:width - 1] = img[x, y + 1:width]

    #在移除原图的Seam线时，对前景图img\_gt也要进行同样的操作

    height, width, depth = img\_gt.shape

    removed\_gt = np.zeros((height, width - 1, depth), np.uint8)

    for (y, x) in seam:

        removed\_gt[x, 0:y] = img\_gt[x, 0:y]

        removed\_gt[x, y:width - 1] = img\_gt[x, y + 1:width]

    return removed\_img,removed\_gt

（5）找出水平方向上的最小能量线并删除（（3）—（4）找出并删除了垂直方向上的最小能量线）

水平方向只需要将原图和前景图转置，即可继续使用Calculate\_Horizontal\_Energy( )，Find\_Vertical\_EnergyLine( )，Remove\_Vertical\_EnergyLine( )这三个函数。

    #对原图和前景图进行转置，即可继续调用垂直方向上的相关函数

    #水平方向进行Seam Carving

    img = img.transpose((1, 0, 2))

    img\_gt = img\_gt.transpose((1, 0, 2))

    for i in range(row\_cnt):

        energy\_map = Calculate\_Horizontal\_Energy(energy(img),img\_gt)

        SeamLine = Find\_Vertical\_EnergyLine(img,energy\_map)

        Plot\_Horizontal(img, SeamLine,image\_list)

        img,img\_gt = Remove\_Vertical\_EnergyLine(img, img\_gt,SeamLine)

    #最终保存的图片需要再转置回来

    img = img.transpose((1, 0, 2))

同时在主函数中输入图像压缩比例，即可计算出水平方向和垂直方向上需要删除的行数和列数，分别执行for循环即可。代码如下所示（包含分析）：

#主函数

if \_\_name\_\_ == "\_\_main\_\_":

    #图片编号，可修改

    img\_number="15"

    #读取原图和前景图

    img = cv2.imread("C:/Users/93508/Desktop/Final/data/imgs/" + img\_number + ".png")

    img\_gt = cv2.imread("C:/Users/93508/Desktop/Final/data/gt/" + img\_number + ".png")

    #存储原图和前景图的尺寸

    img\_height, img\_width = img.shape[0], img.shape[1]

    img\_gt\_height, img\_gt\_width = img\_gt.shape[0], img\_gt.shape[1]

    #压缩比例，可修改

    ratio = 0.8

    #压缩后的原图尺寸

    width = int(ratio \* img\_width)

    height = int(img\_height \* ratio)

    # 在目标目录保存原图和前景图用于对比

    cv2.namedWindow('Seam Carving', cv2.WINDOW\_NORMAL)

    cv2.resizeWindow('Seam Carving', 500, 500)

    cv2.imwrite("C:/Users/93508/Desktop/Final/data/result/SeamCarving/" + img\_number + "\_origin.png", img)

    cv2.imwrite("C:/Users/93508/Desktop/Final/data/result/SeamCarving/" + img\_number + "\_gt.png", img\_gt)

    # col\_cnt：需要删除的列数

    # row\_cnt：需要删除的行数

    col\_cnt = img\_width - width

    row\_cnt = img\_height - height

    #用于存放生成GIF的各帧图片

    image\_list=[]

    #先在垂直方向进行Seam Carving

    for i in range(col\_cnt):

        energy\_map = Calculate\_Horizontal\_Energy(energy(img),img\_gt)

        SeamLine = Find\_Vertical\_EnergyLine(img,energy\_map)

        Plot\_Vertical(img, SeamLine,image\_list)

        img,img\_gt = Remove\_Vertical\_EnergyLine(img, img\_gt,SeamLine)

    #对原图和前景图进行转置，即可继续调用垂直方向上的相关函数

    #水平方向进行Seam Carving

    img = img.transpose((1, 0, 2))

    img\_gt = img\_gt.transpose((1, 0, 2))

    for i in range(row\_cnt):

        energy\_map = Calculate\_Horizontal\_Energy(energy(img),img\_gt)

        SeamLine = Find\_Vertical\_EnergyLine(img,energy\_map)

        Plot\_Horizontal(img, SeamLine,image\_list)

        img,img\_gt = Remove\_Vertical\_EnergyLine(img, img\_gt,SeamLine)

    #最终保存的图片需要再转置回来

    img = img.transpose((1, 0, 2))

    #保存结果

    cv2.imwrite("C:/Users/93508/Desktop/Final/data/result/SeamCarving/" + img\_number + '\_result.png', img)

    cv2.imshow('Seam Carving', img)

    #保存GIF

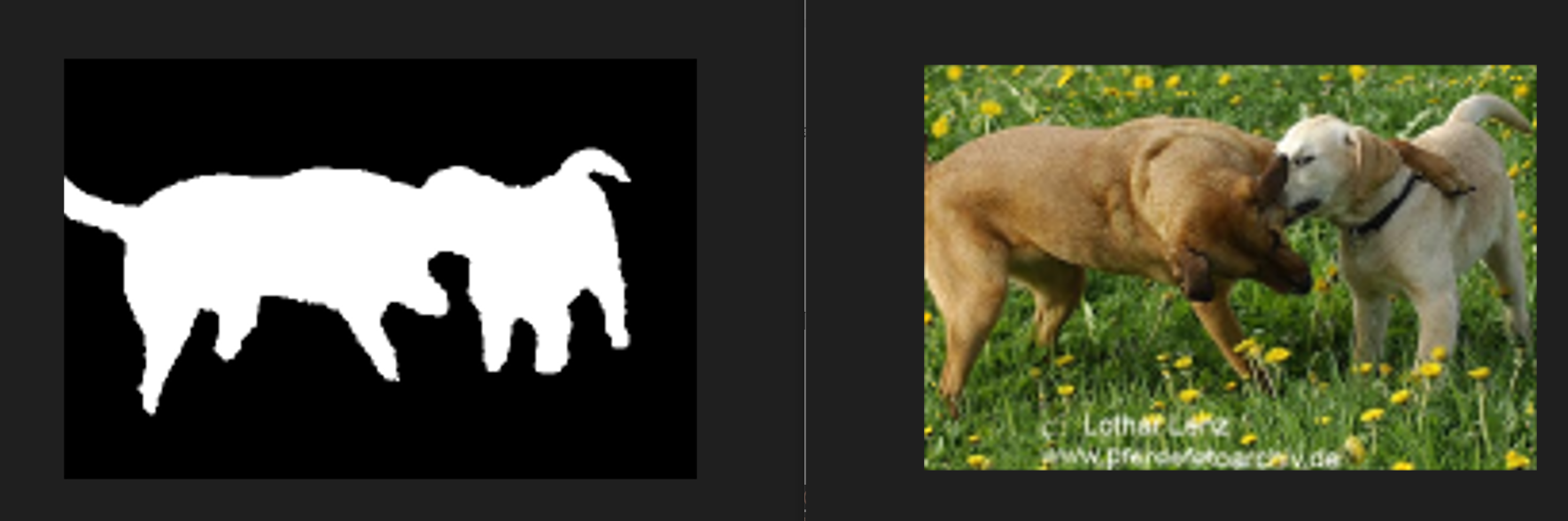
    imageio.mimsave("C:/Users/93508/Desktop/Final/data/result/SeamCarving/" + img\_number + '.gif', image\_list, 'GIF', duration=0.1)

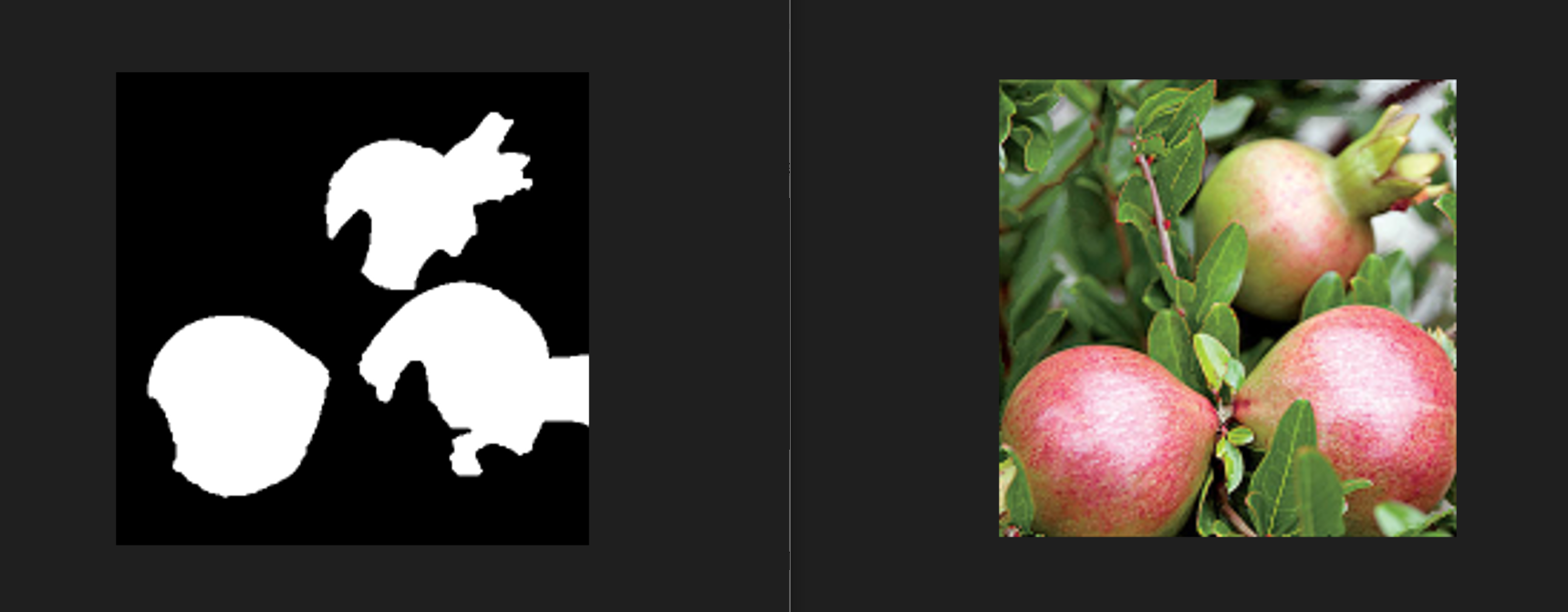
    cv2.waitKey(0)

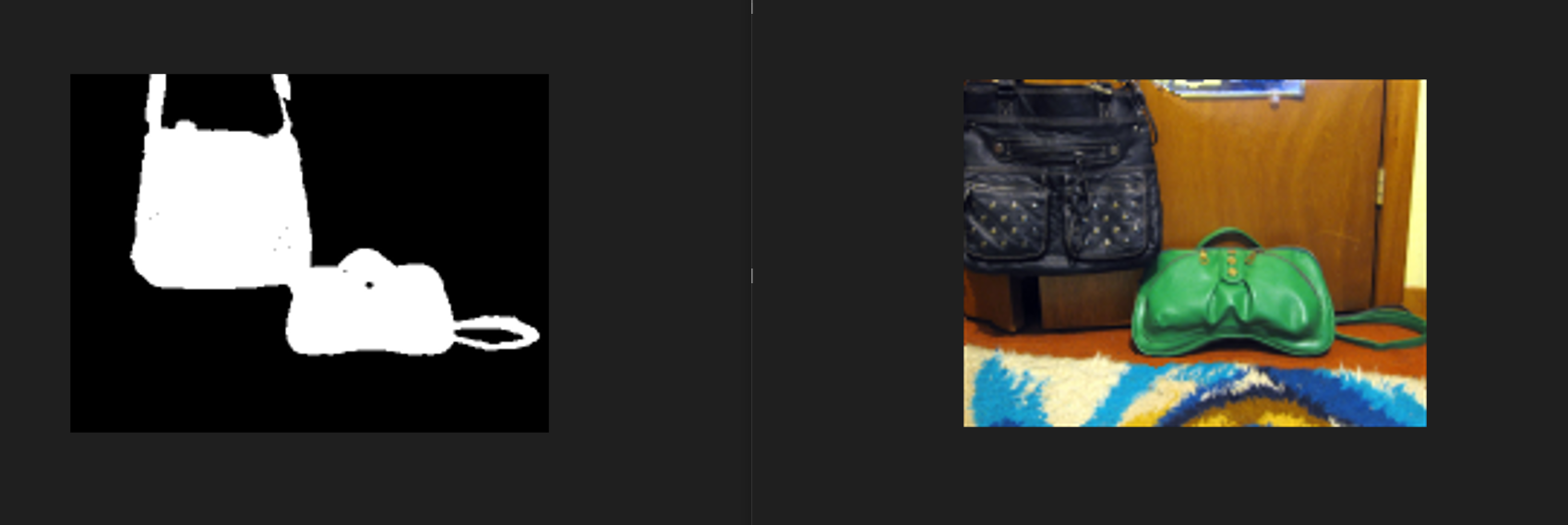
    cv2.destroyAllWindows()

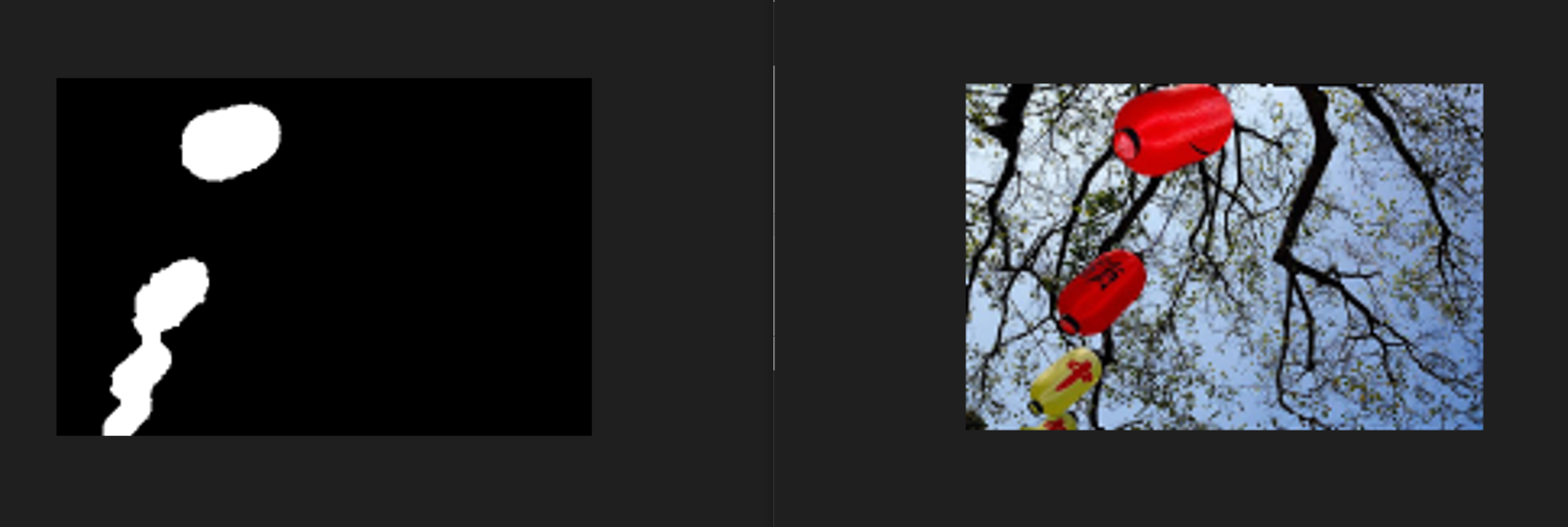
## 2.3 实验结果

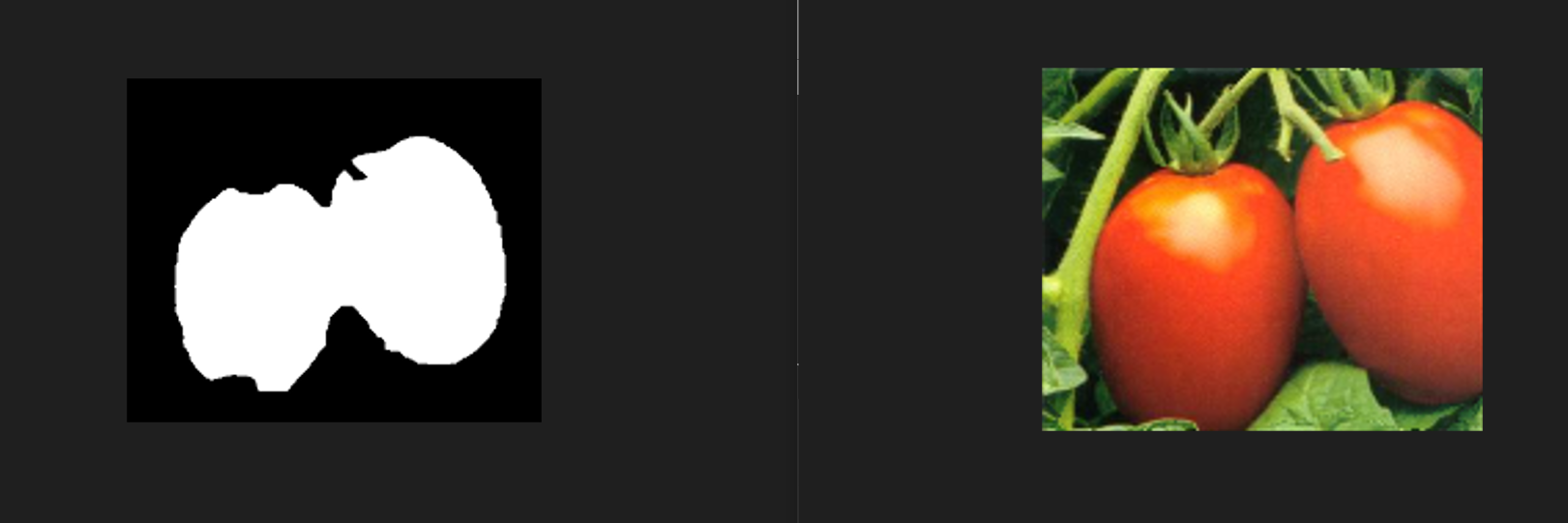
我的学号为18329015，实验结果和前景图对比如下所示：

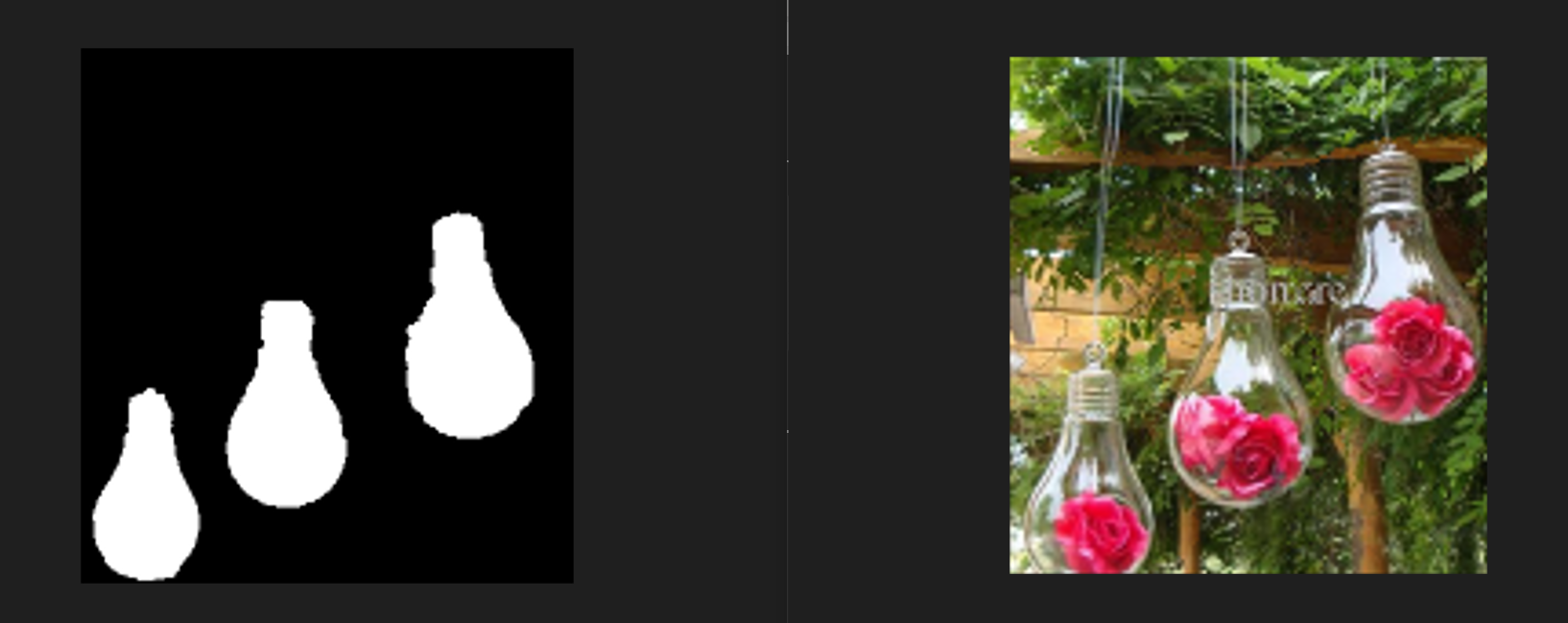


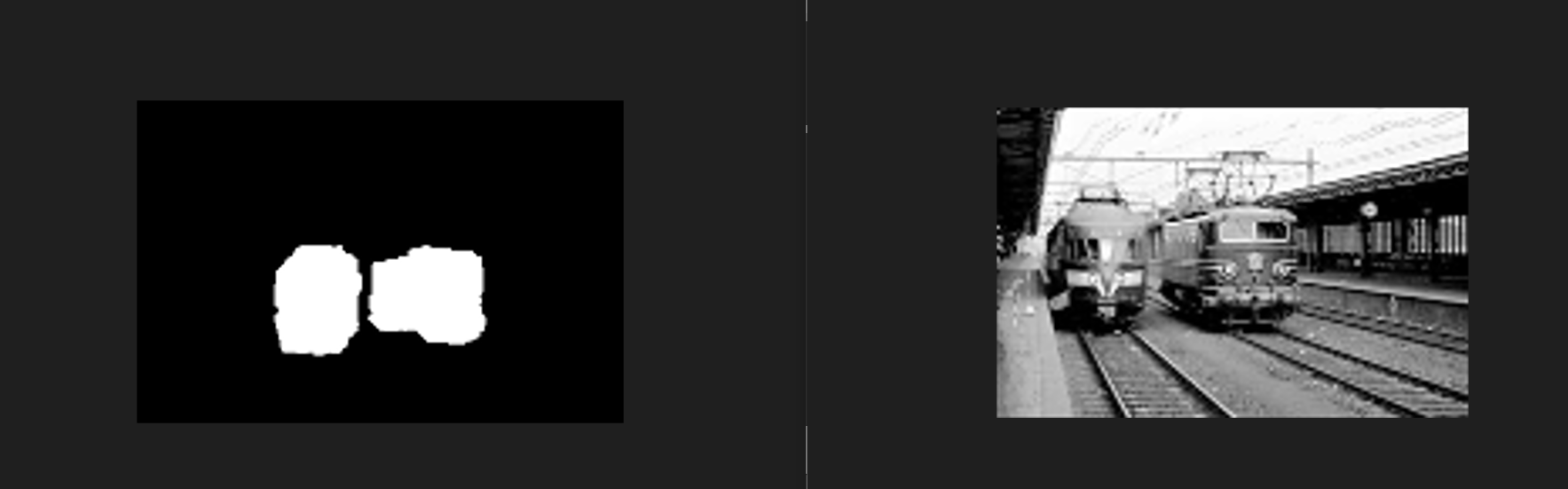


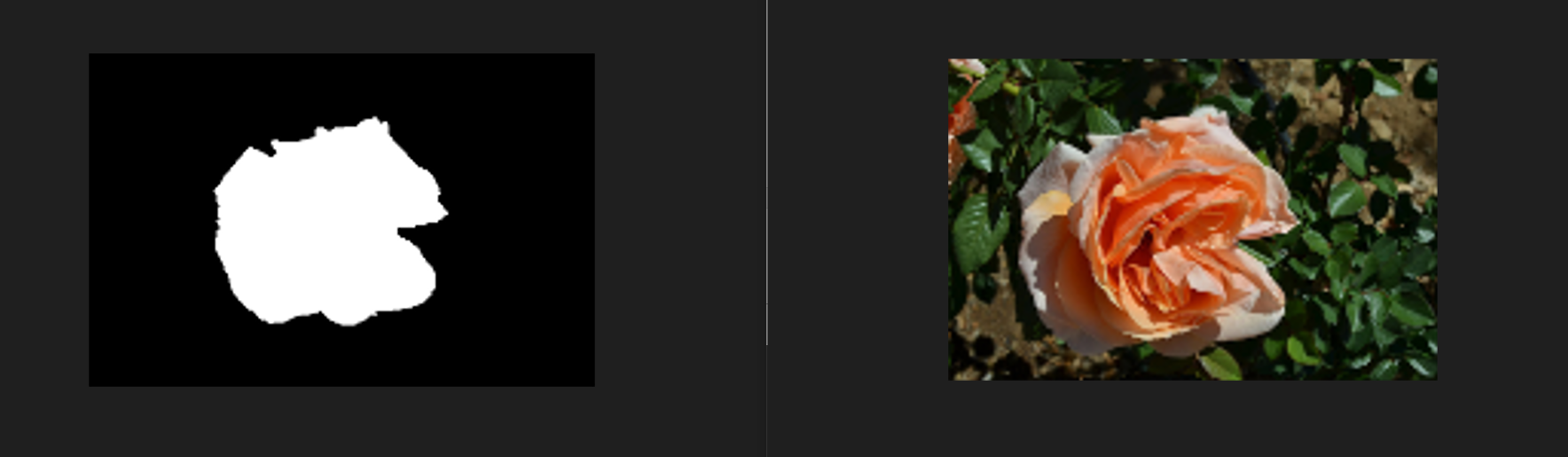


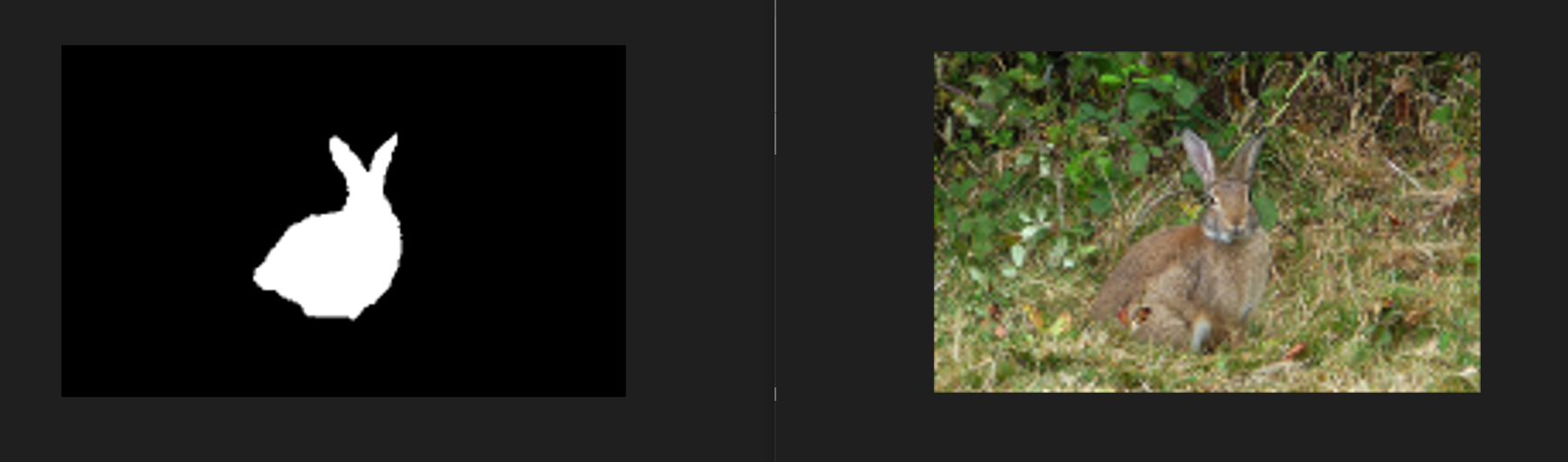


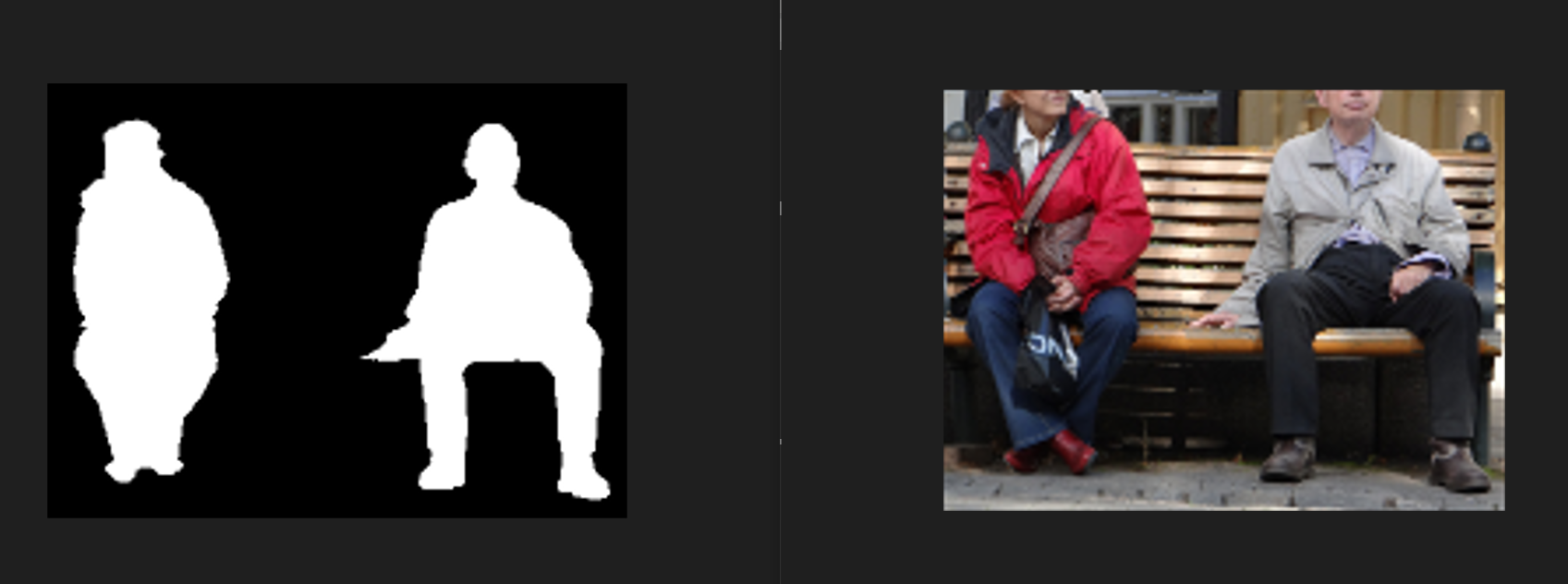












其中715.png和815.png生成了gif，详见文件夹result/SeamCarving

# 3 Segmentation

## 3.1 问题内容

结合“Lecture 7 Segmentation”内容及参考文献[1]，实现基于 Graphbased image segmentation 方法（可以参考开源代码，建议自己实现），通过设定恰当的阈值将每张图分割为 50~70 个区域，同时修改算法要求任一分割区域的像素个数不能少于 50 个（即面积太小的区域需与周围相近区域合并）。结合GT 中给定的前景 mask，将每一个分割区域标记为前景（区域 50%以上的像素在 GT 中标为 255）或背景（50%以上的像素被标为 0）。区域标记的意思为将该区域内所有像素置为 0 或 255。要求对测试图像子集生成相应处理图像的前景标注并计算生成的前景 mask 和 GT 前景 mask 的 IOU 比例。假设生成的前景区域为 R1，该图像的 GT 前景区域为 R2，则



[1] Felzenszwalb P F, Huttenlocher D P. Efficient graph-based image

segmentation[J]. International journal of computer vision, 2004, 59(2): 167-181. <http://people.cs.uchicago.edu/~pff/papers/seg-ijcv.pdf>

## 3.2 算法思路&代码分析

（1）将图片的所有像素点存储到矩阵中，连接边，建立无向图

代码如下所示，具体思路及分析可见注释：

#建图

def build\_graph(img):

    #存储图的尺寸

    height, width = img.shape[:2]

    #edges：存储边

    #edges\_dis：存储边的长度

    edges = []

    edges\_dis = []

    #开始遍历整张图片建图

    for i in range(height):

        for j in range(width):

            #对当前点向正右方的点连接一条边

            if j+1 <= width-1:

                edges.append(np.array([i\*width+j, i\*width+(j+1)]))

                edges\_dis.append(dist(img[i][j], img[i][j+1]))

            #对当前点向正下方的点连接一条边

            if i+1 <= height-1:

                edges.append(np.array([i\*width+j, (i+1)\*width+j]))

                edges\_dis.append(dist(img[i][j], img[i+1][j]))

    #将数组设置为垂直向下存储每条边

    edges = np.vstack(edges).astype(int)

    edges\_dis = np.array(edges\_dis).astype(float)

    #考虑到我们需要根据边的大小来对edges和edges\_dis两个数组进行相同的排序

    #而edges本身只存储边，无法用大小来进行排序

    #所以我们使用argsort来提取edges\_dis数组从大到小排序的下标数组，再应用到edges数组中返回

    id = np.argsort(edges\_dis)

    return edges[id], edges\_dis[id]

（2）使用并查集的思想对图进行find和join操作

代码如下所示，具体思路及详细分析可见注释：

对于并查集的find和join操作，我们封装一个类DisJointSet

#并查集

class DisjointSet:

    #初始化

    def \_\_init\_\_(self, num\_vertices):

        '''

            num\_sets:簇的数量

            num\_vertices:像素点数量

            data:每个簇自身所带的特征信息

        '''

        self.num\_sets = num\_vertices

        self.num\_vertices = num\_vertices

        self.data = np.empty((num\_vertices, 4), dtype=np.int\_)

        #像素点特征信息初始化

        for i in range(num\_vertices):

            '''

                rank:合并时的从属关系(谁合并谁)

                size:每个簇的大小

                parent:用于判断属于哪个簇

                gt\_num:簇内前景点的数量

            '''

            self.data[i, 0] = 0 # rank

            self.data[i, 1] = 1 # size

            self.data[i, 2] = i # parent

            self.data[i, 3] = 0 # gt\_num

    #判断当前像素点所属的簇

    #并查集中的find操作

    def find(self, id):

        #递归回溯寻找所属的簇

        parent = id

        while parent != self.data[parent, 2]:

            parent = self.data[parent, 2]

        self.data[id, 2] = parent

        return parent

    #并查集中的join操作

    def join(self, id1, id2):

        #若合并的从属关系为：左>右，则右边的簇合并到左边

        if self.data[id1, 0] > self.data[id2, 0]:

            #合并需将左边的簇的像素点数量更新为二者之和

            #同时将右边的簇所属的父节点id修改为左边的

            self.data[id1, 1] += self.data[id2, 1]

            self.data[id2, 2] = id1

        #同理，将合并的关系调换即可

        else:

            self.data[id2, 1] += self.data[id1, 1]

            self.data[id1, 2] = id2

        self.num\_sets -= 1

#计算两个像素点之间的距离

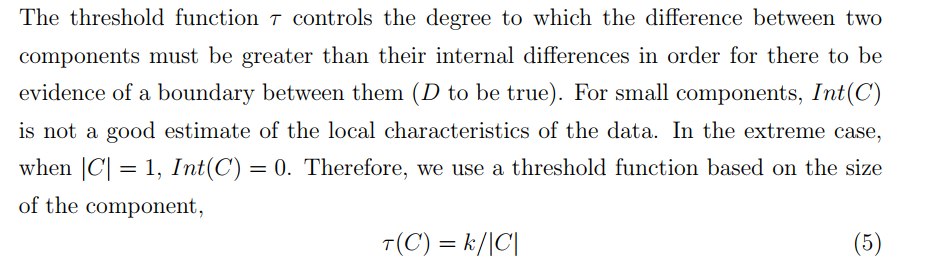
def dist(p1, p2):

    diff = pow((p1-p2),2)

    return np.sqrt(np.sum(diff))

（3）对图的不同区域分割和合并（同时需要根据参考文献中的公式进行find和join操作）

根据参考论文可知首先要确立一个阈值函数threshold



论文中提到为了便于计算，可假设|C| = 1，这样阈值函数可化简为*k*。

    #根据参考文献设立阈值函数

    for i in range(height \* width):

        threshold[i] = k

遍历图中的所有边，若某条边上的两个顶点不属于同一个簇且不满足阈值函数，则需要进行join合并操作。

    #开始合并

    for i in range(len(edges)):

        v1\_parent = djs.find(edges[i, 0])

        v2\_parent = djs.find(edges[i, 1])

        #若某条边的两点不属于同一个簇

        if v1\_parent != v2\_parent:

            #且不满足参考文献的判定标准

            if (edges\_dis[i] <= threshold[v1\_parent]) and (edges\_dis[i] <= threshold[v2\_parent]):

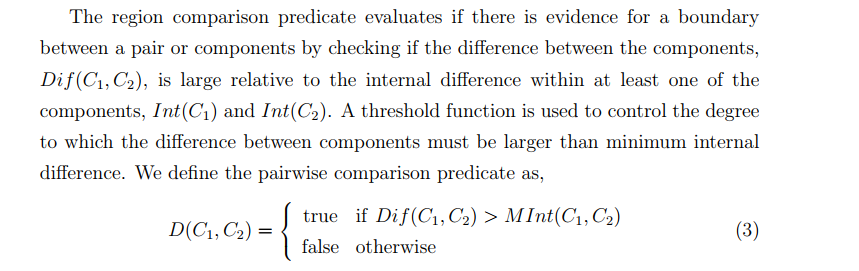
                #则对这两个簇进行合并

                djs.join(v1\_parent, v2\_parent)

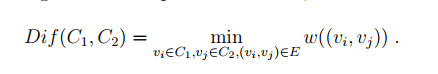
                v1\_parent = djs.find(v1\_parent)

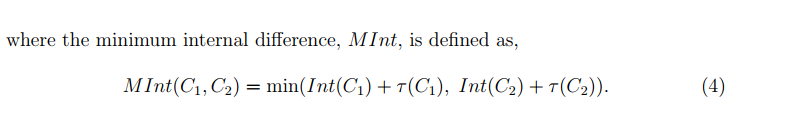
                #更新阈值函数

                threshold[v1\_parent] = edges\_dis[i] + k / djs.data[v1\_parent, 1]



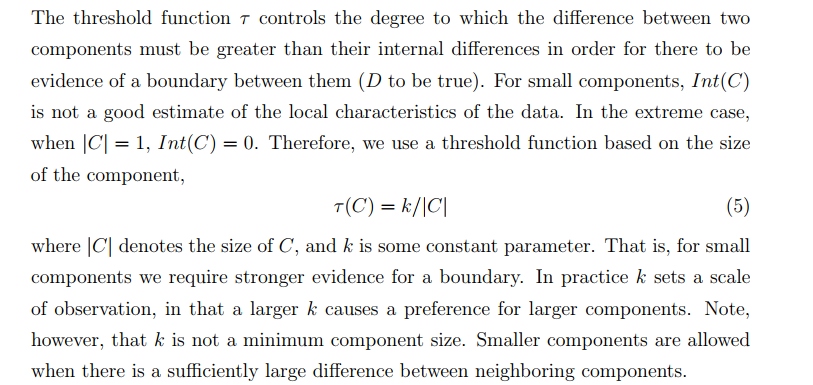






由参考论文可知，代表两簇之间的最短距离（代码中可简化为边的长度，无需找最小，因为我们会遍历所有的边）。代表边的距离加上阈值函数。

同时根据论文所说（见下页）：



我们可以将*Int(C)*默认为0，所以可简化为两个不同簇的阈值函数的最小值，体现在代码中为：

        #若某条边的两点不属于同一个簇

        if v1\_parent != v2\_parent:

            #且不满足参考文献的判定标准

            if (edges\_dis[i] <= threshold[v1\_parent]) and (edges\_dis[i] <= threshold[v2\_parent]):

但是在后续更新阈值函数时不能将*Int(C)*默认为0，所以代码如下所示：

   threshold[v1\_parent] = edges\_dis[i] + k / djs.data[v1\_parent, 1]

即阈值函数为（边的长度+，其中代表簇内的顶点个数）

同时还需要合并较小的簇（像素点数量少于50），代码如下所示（详细分析见注释）：

    #对较小的簇也进行合并

    while True:

        flag = True

        for i in range(len(edges)):

            v1\_parent = djs.find(edges[i, 0])

            v2\_parent = djs.find(edges[i, 1])

            #若该边的两个点不属于同一个簇且其中一个簇的像素点数量小于阈值50

            #则对这两个簇进行合并

            if (v1\_parent != v2\_parent) and ((djs.data[v1\_parent, 1] < min\_num) or (djs.data[v2\_parent, 1] < min\_num)):

                flag = False

                djs.join(v1\_parent, v2\_parent)

        if flag:

            break

（4）区域标记

代码如下所示，具体分析详见注释：

    #区域标记

    for i in range(height):

        for j in range(width):

            #若当前像素点位置为白色(前景区域)

            #则为该像素点所属的簇对其内部属性gt\_num = gt\_num + 1

            if img\_gt[i][j] > 200:

                djs.data[djs.find(i\*width+j), 3] += 1

    #根据之前的区域标记生成我们自己前景图

    res = np.zeros((height, width))

    for i in range(height):

        for j in range(width):

            parent = djs.find(i\*width+j)

            #若当前像素点所属的簇内部gt\_num超过一半，则设置当前点为白色(对于所有这个簇的像素点都会变为白色)

            #即 gt\_num / size >= 0.5

            if djs.data[parent, 3] / djs.data[parent, 1] >= 0.5:

                res[i][j] = 255

    return res, djs

（5）调用主函数（包含计算IOU）

代码如下所示，具体分析详见注释：

#主函数

if \_\_name\_\_ == "\_\_main\_\_":

    #需要遍历的文件名

    a = ["15","115","215","315","415","515","615","715","815","915"]

    cnt = 0

    cnt2 = 0

    correct = 0

    # k代表每个簇的阈值

    k = 130

    #遍历测试图片

    for img\_number in a:

        #每个簇的像素点数量不得少于min\_num

        min\_num = 50

        img = cv2.imread("C:/Users/93508/Desktop/Final/data/imgs/" + img\_number + ".png",cv2.IMREAD\_COLOR)

        img\_gt = cv2.imread("C:/Users/93508/Desktop/Final/data/gt/" + img\_number + ".png",cv2.IMREAD\_GRAYSCALE)

        res, djs = segmentation(img, img\_gt, k, min\_num)

        #将前景图写入文件夹用于对比

        cv2.imwrite("C:/Users/93508/Desktop/Final/data/result/Segmentation/" + img\_number + "\_gt.png", img\_gt)

        #将生成的前景图保存至对应目录

        smt\_output\_path = "C:/Users/93508/Desktop/Final/data/result/Segmentation"

        filename = img\_number + "\_result.png"

        cv2.imwrite(os.path.join(smt\_output\_path, filename), res)

        #保存前景图尺寸

        height, width = img\_gt.shape

        #用于计算IOU

        intersection = 0

        union = 0

        #遍历前景图每个像素点

        for i in range(height):

            for j in range(width):

                #若生成图和前景图的当前像素点都是白色则 intersection + 1

                if img\_gt[i][j] > 200 and res[i][j] > 200:

                    intersection += 1

                #若生成图和前景图的当前像素点不全是白色则 union + 1

                if img\_gt[i][j] > 200 or res[i][j] > 200:

                    union += 1

        #统计合并后簇的数量

        cnt += djs.num\_sets

        if djs.num\_sets >= 50 and djs.num\_sets <= 70:

            cnt2 = cnt2 + 1

        print("%s   %d   %.2f%%" %(filename, djs.num\_sets, intersection / union\*100))

        #用于统计平均正确率

        correct = correct + intersection / union

    print("\nK = %d\nAverage num\_sets = %.1f\nCorrect num\_sets = %d\nAverage Correctness = %.2f%%\n" %(k, cnt/10, cnt2, correct/10\*100))

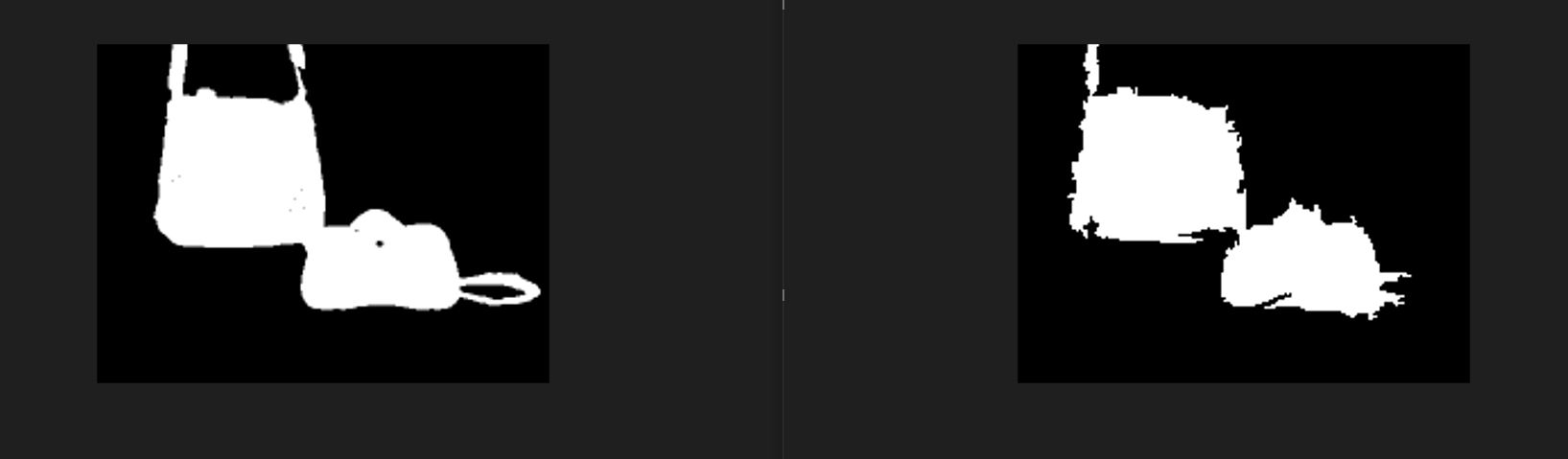
## 3.3 实验结果

经过对k值的循环遍历，我得出最佳的k值为130.（其中Average num\_sets代表10张图的平均簇数，Correct num\_sets代表满足簇的个数在50-70之间的图片数量，Average Correctness代表10张图的平均IOU）

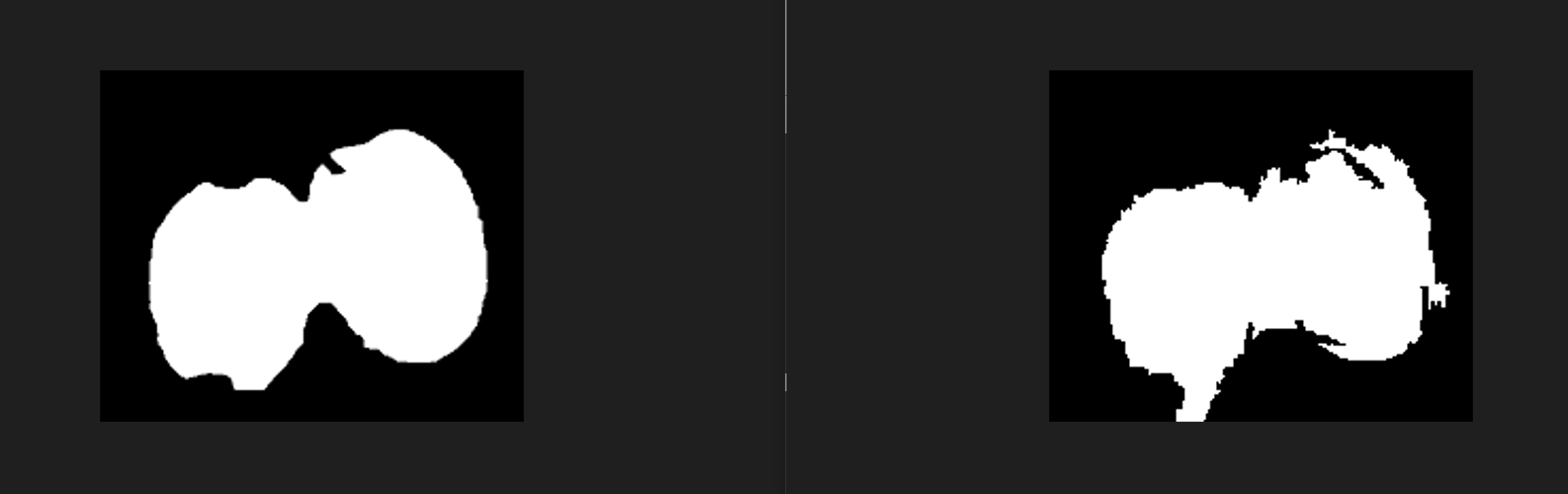




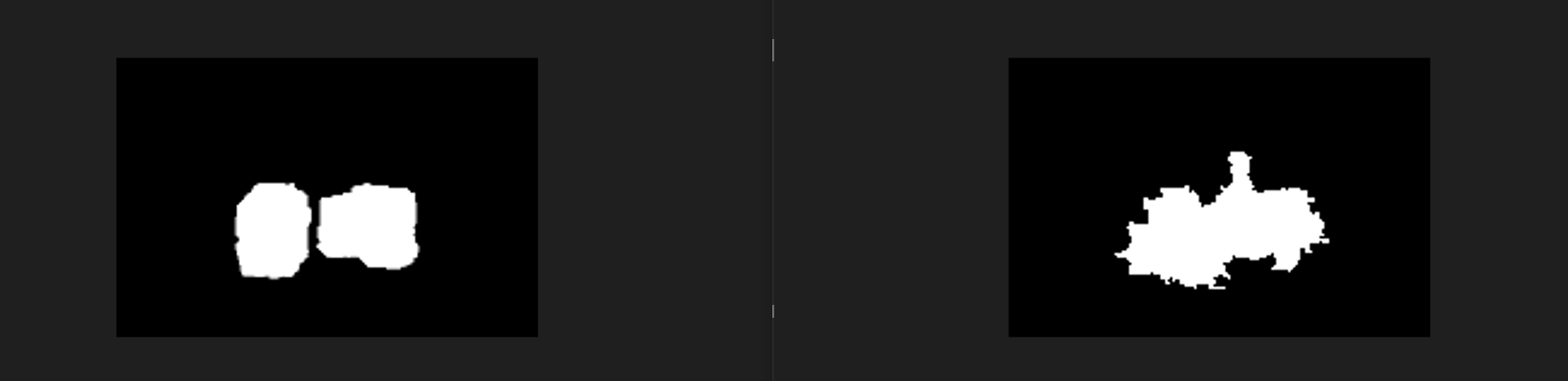


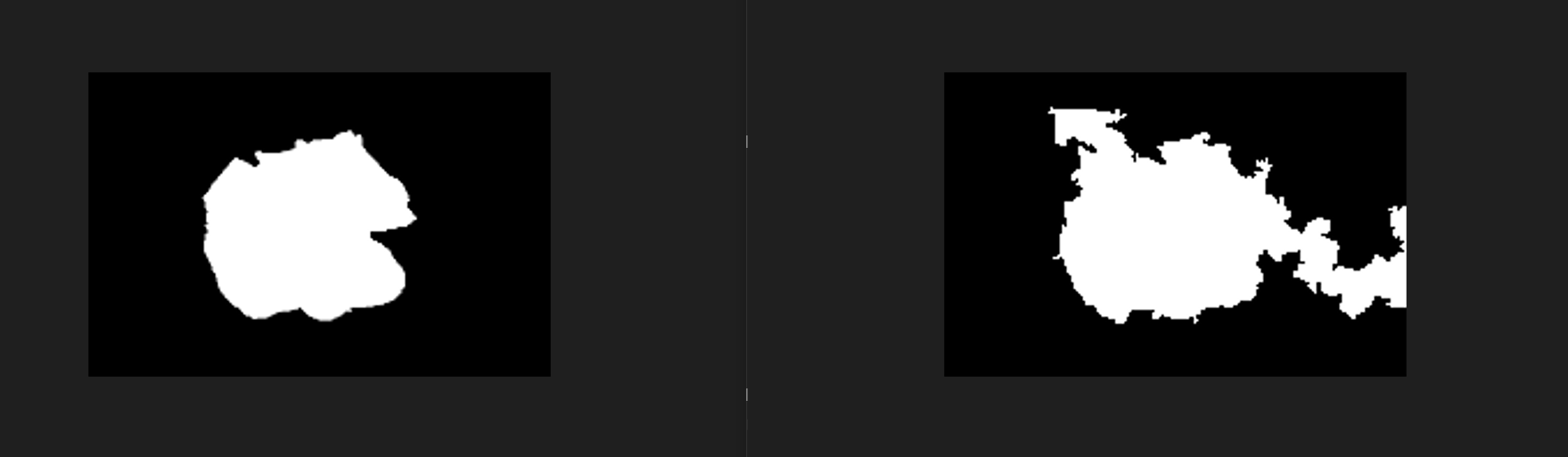


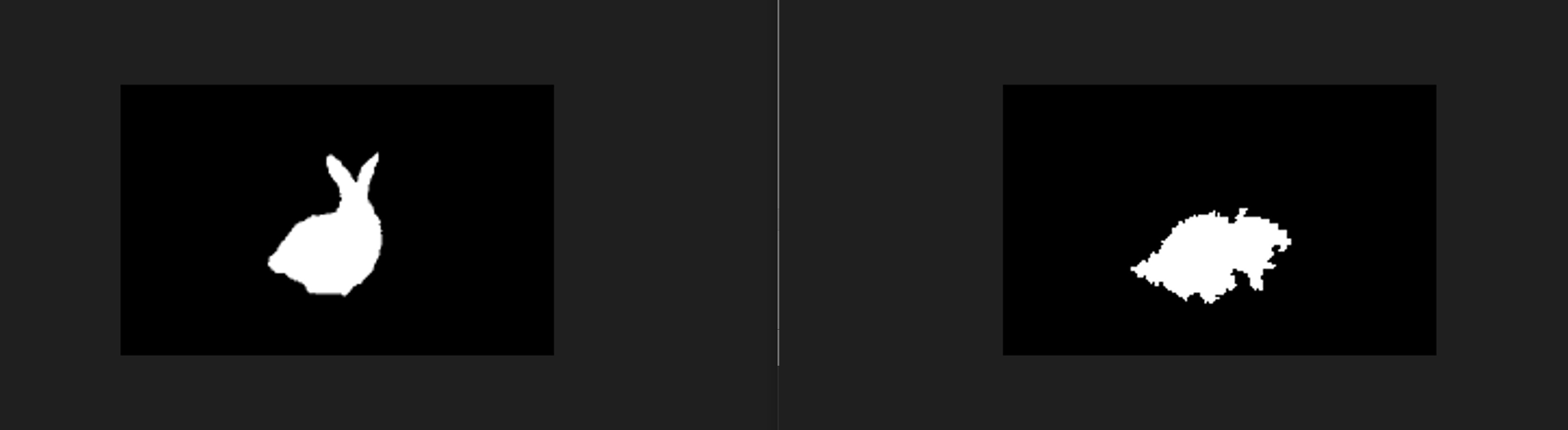














# 4 Dimensionality Reduction—PCA

## 4.1 问题内容

结合“Lecture 10. Dimensionality Reduction”中学习的方法，每一位同学从各自的测试子集中任选一张代表图，执行 PCA 压缩。先将图片尺寸缩放或裁减为 12的倍数，以 12\*12 patch 为单位执行 PCA 压缩，1）展示 16 个最主要的特征向量的可视化图，2）展示 144D，60D，16D 和 6D 的压缩结果。需要介绍算法流程和对应的结果展示。

## 4.2 问题1——算法思路&代码分析

PCA算法流程为：

设有*m*条*n*维数据。

（1）将原始数据按列组成*n*行*m*列矩阵X

这里我选取的图片为915.jpg，原图尺寸为200\*150，根据题目要求我将其裁剪为144\*144，代码如下所示，详细分析见注释：



#存储裁剪后图片的每个子图(12\*12)

X = np.zeros((12,12),dtype = int)

#存储所有子图，组成训练集

X\_new = np.zeros((144,144),dtype = int)

#读取代表

#img\_num可修改为自己需要读取的图片编号

img\_number = "915"

img = cv2.imread("C:/Users/93508/Desktop/Final/data/imgs/" + img\_number + ".png")

#以灰度图的方式读取，去除图片的三通道RGB信息

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

#将图片尺寸裁剪为12的倍数

img = cv2.resize(img,(144,144))

#将图片拆分12\*12的子图并全部保存至X\_new

#每个12\*12的子图被视为144维的向量，将其一维扁平化存入X\_new的每一行中

#由于图片为144\*144，子图为12\*12，所以X\_new为144\*144

#即144条数据，每条数据有144维的特征

for i in range(0,144):

    X = img[int(i/12)\*12:int(i/12)\*12+12,i%12\*12:i%12\*12+12]

    X\_new[i,:] = X.reshape(1,-1)

（2）将X的每一行（代表一个特征）进行零均值化，即减去这一行的均值

PS：（2）—（6）的操作均在函数pca中实现：

    #矩阵行数代表样本数量n\_samples，矩阵列数代表每个样本的特征数n\_features(即特征向量)

    n\_samples, n\_features = X.shape

    #对矩阵进行零均值化，即减去这一行的均值

    mean = np.array([np.mean(X[:,i]) for i in range(n\_features)])

    norm\_X = X - mean

（3）求出协方差矩阵C = 

    #散度矩阵scatter\_matrix

    #散度矩阵就是协方差矩阵\*(总数据量-1),因此他们的特征根和特征向量是一样的

    scatter\_matrix = np.dot(np.transpose(norm\_X),norm\_X)

这里我们求的是散度矩阵，易看出散度矩阵就是协方差矩阵\*(总数据量-1)，所以他们的特征根和特征向量是一样的。（后续调用sklearn库中的PCA函数进行对比也可以看出我们手工实现的pca是正确的）

（4）求出协方差矩阵C的特征值及对应的特征向量

    #计算特征向量和特征值

    eig\_val, eig\_vec = np.linalg.eig(scatter\_matrix)

    eig\_pairs = [(np.abs(eig\_val[i]), eig\_vec[:,i]) for i in range(n\_features)]

（5）将特征向量按对应特征值大小从上到下按行排列成矩阵，取前*k*行组成矩阵P

    #根据特征值大小，从大到小对(特征值,特征向量)的多对pair进行排序

    eig\_pairs.sort(reverse=True)

    #选出前k个特征向量

    feature=np.array([ele[1] for ele in eig\_pairs[:k]])

（6）Y = PX即为降维到*k*维后的数据

    #Y = PX即为降维到k维后的数据

    data=np.dot(norm\_X,np.transpose(feature))

    return data

为了验证手工实现的PCA功能正确，我们又写了一份调用sklearn库中自带的PCA函数的代码用于结果对比，代码如下所示，详细分析见注释：

import numpy as np

import os

import cv2

import shutil

import math

import random

from PIL import Image

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

from sklearn.decomposition import PCA

import warnings

warnings.filterwarnings("ignore")

#存储裁剪后图片的每个子图(12\*12)

X = np.zeros((12,12),dtype = int)

#存储所有子图，组成训练集

X\_new = np.zeros((144,144),dtype = int)

#读取代表图片

#img\_num可修改为自己需要读取的图片编号

img\_number = "915"

img = cv2.imread("C:/Users/93508/Desktop/Final/data/imgs/" + img\_number + ".png")

#以灰度图的方式读取，去除图片的三通道RGB信息

img = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

#将图片尺寸裁剪为12的倍数

img = cv2.resize(img,(144,144))

#将图片拆分12\*12的子图并全部保存至X\_new

#每个12\*12的子图被视为144维的向量，将其一维扁平化存入X\_new的每一行中

#由于图片为144\*144，子图为12\*12，所以X\_new为144\*144

#即144条数据，每条数据有144维的特征

for i in range(0,144):

    X = img[int(i/12)\*12:int(i/12)\*12+12,i%12\*12:i%12\*12+12]

    X\_new[i,:] = X.reshape(1,-1)

# 可修改n的数量来选择需要保留的特征数量（即降维后的结果）

n = 16

#调用sklearn库中的PCA

#n\_components: 需要保留的特征数量（即降维后的结果）

pca = PCA(n\_components = n) #实例化

newX = pca.fit\_transform(X\_new) #用已有数据训练PCA模型，并返回降维后的数据

#PCA中的属性components\_为降维后的特征向量

print("降维后的特征向量为：\n")

for k in range(16):

    for i in range(12):

        for j in range(12):

            print("%.3f  "%(pca.components\_[k][i\*12+j]),end="")

        print("\n")

    print("\n")

print("-----------------------------------------------------------------------------")

#将特征向量rescale到0——255之间

array = pca.components\_

ymax = 255

ymin = 0

xmax = max(map(max,array))

xmin = min(map(min,array))

for i in range(16):

    for j in range(144):

        array[i][j] = int(round(((ymax-ymin)\*(array[i][j]-xmin)/(xmax-xmin))+ymin))

print("rescale到0—255范围内的特征向量为：\n")

for k in range(16):

    for i in range(12):

        for j in range(12):

            print("%d  "%(array[k][i\*12+j]),end="")

        print("\n")

    print("\n")

print("-----------------------------------------------------------------------------")

print("压缩结果为：\n")

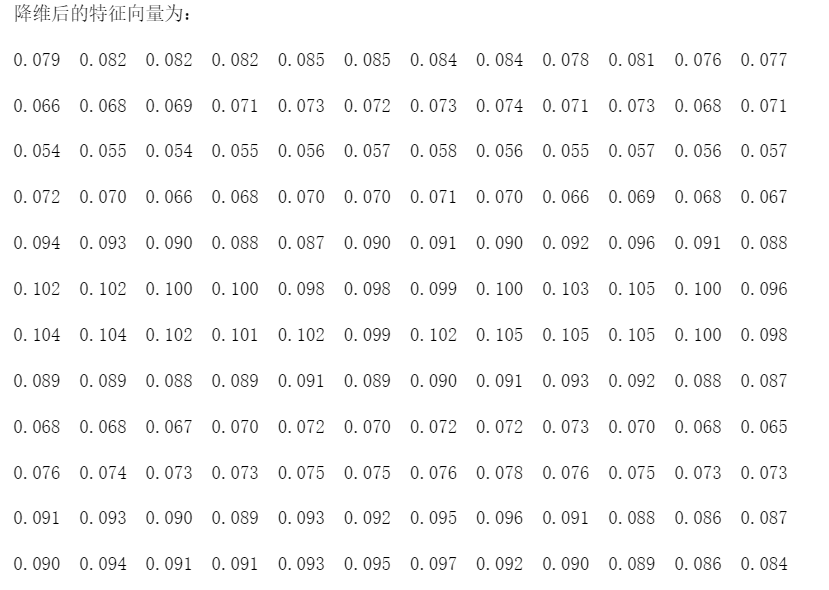
print(newX)

## 4.3 问题1结果展示

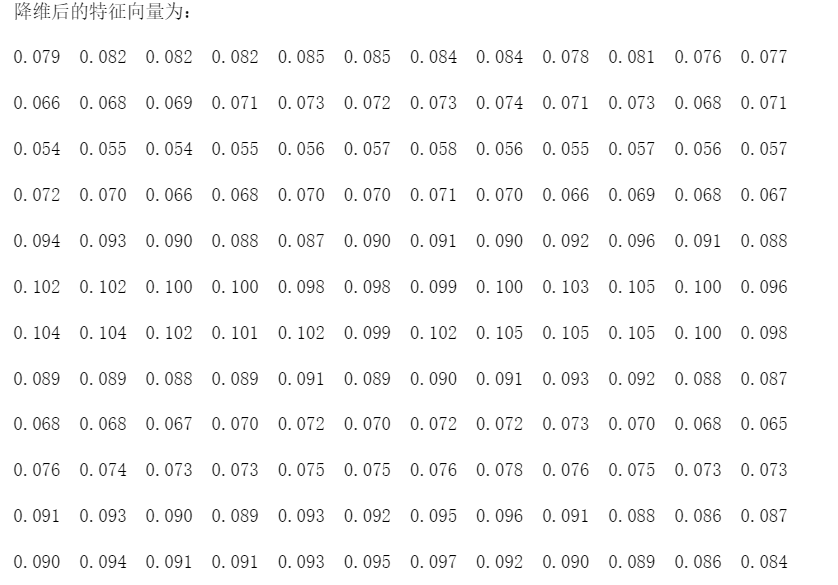
首先我们通过对比手工PCA和sklearn库中PCA函数的二者降维后的特征向量可知前者实现正确（可详见CV\_Question4.ipynb）。

PS：这里的保留下来的16个特征向量是1\*144维的，根据老师在群里所要求的，为了使得特征向量可视化，我在对特征向量的输出上改为以12\*12的矩阵形式输出。

手工PCA：



Sklearn库的PCA：



易知实现正确。

但在二者部分特征向量矩阵中会出现数字相同，少数数字符号相反的情况，导致最终的压缩结果不同。在我反复检查代码之后，发现原因如下：

sklearn中的PCA是通过svd\_flip函数实现的，sklearn对奇异值分解结果进行了一个处理，因为，也就是*u*和*v*同时取反得到的结果是一样的，而这会导致通过PCA降维得到不一样的结果（虽然都是正确的）。

所以通过二者特征向量矩阵数值相同，仍然可以验证我手工实现PCA的正确性。

剩余的特征向量可视化可详见CV\_Question4.ipynb，不在这里依次展示（16张矩阵图片过于占篇幅）。

## 4.4 问题2结果展示

144D，60D，16D 和 6D的降维后的特征向量和最终压缩结果仍可详见CV\_Question4.ipynb，不在这里依次展示。

# 5 实验感想

这次的期末大作业让我收获颇多，对于本学期所学的知识完成了一次系统的汇总应用：对于Seam Carving，Segmentation以及降维算法PCA有了更深层次的理解。由于时间关系和个人学艺不精，未能完成第三问，这里向老师和助教表示歉意，同时也想对您们说一句：辛苦了！谢谢老师和助教平日的耐心，让我从这门课学到了许多，非常感谢老师和助教这学期的努力付出！