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

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）

$$E(\mathbf{I}) = \left|\frac{\partial}{\partial x}\mathbf{I}\right| + \left|\frac{\partial}{\partial y}\mathbf{I}\right|$$

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：

$$M(i,j) = \min \begin{cases} M(i-1,j-1) + |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j-1)| \\ M(i-1,j) + |I(i,j+1) - I(i,j-1)| \\ M(i-1,j+1) + |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j+1)| \end{cases}$$

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：

$$C_V(i,j) = |I(i,j+1) - I(i,j-1)|$$

cost_left 对应公式为：

$$C_V(i,j) = |I(i,j+1) - I(i,j-1)|$$

cost_right 对应公式为：

$$C_R(i,j) = |I(i,j+1) - I(i,j-1)| + |I(i-1,j) - I(i,j+1)|$$

（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))
```

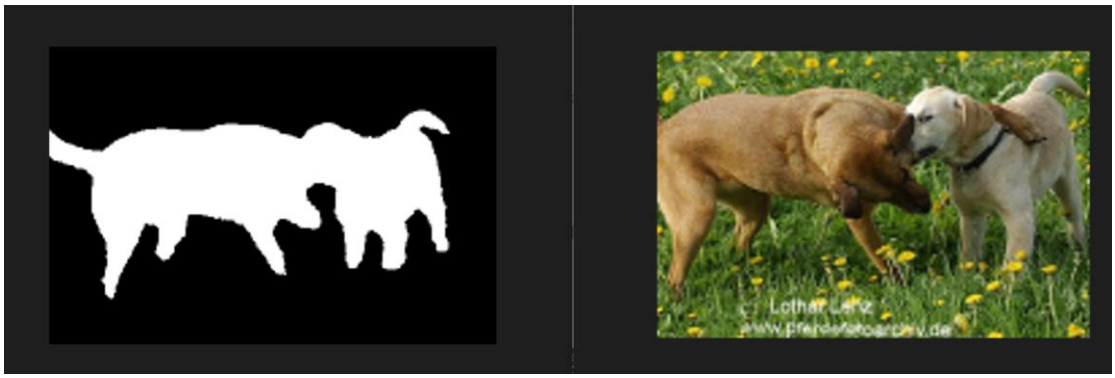
同时在主函数中输入图像压缩比例，即可计算出水平方向和垂直方向上需要删除的行数和列数，分别执行 for 循环即可。代码如下所示（包含分析）：

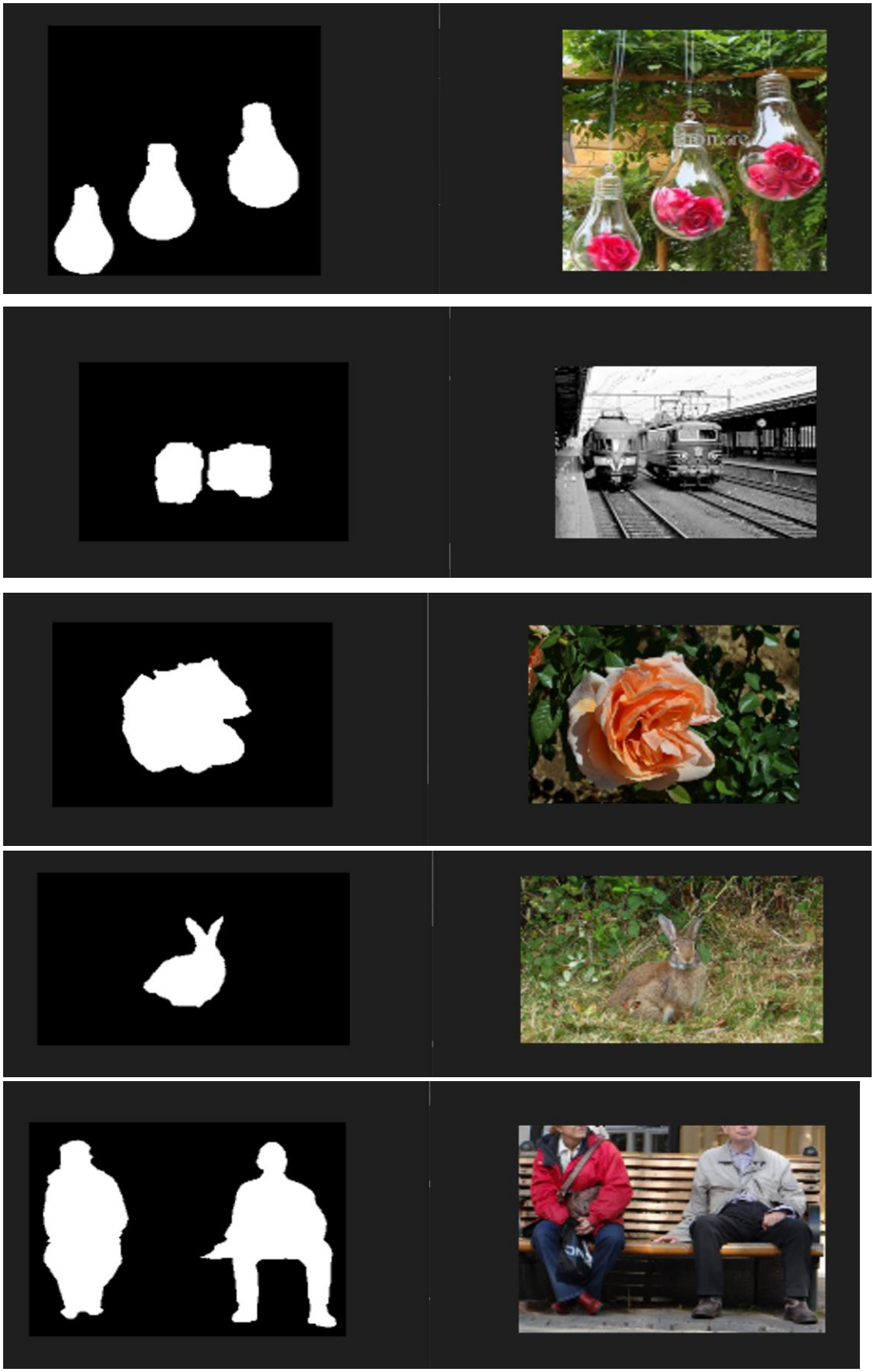
```
#主函数
if __name__ == "__main__":
    #图片编号，可修改
    img_number="15"

    #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/SeamCarvi
ng/" + 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

$$IOU = \frac{R1 \cap R2}{R1 \cup R2}$$

[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：

①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）

①根据参考论文可知首先要确立一个阈值函数 threshold

The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them (D to be true). For small components, $Int(C)$ is not a good estimate of the local characteristics of the data. In the extreme case, when $|C| = 1$, $Int(C) = 0$. Therefore, we use a threshold function based on the size of the component,

$$\tau(C) = k/|C| \tag{5}$$

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
```

②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]
```

The region comparison predicate evaluates if there is evidence for a boundary between a pair or components by checking if the difference between the components, $Dif(C_1, C_2)$, is large relative to the internal difference within at least one of the components, $Int(C_1)$ and $Int(C_2)$. A threshold function is used to control the degree to which the difference between components must be larger than minimum internal difference. We define the pairwise comparison predicate as,

$$D(C_1, C_2) = \begin{cases} \text{true} & \text{if } Dif(C_1, C_2) > MInt(C_1, C_2) \\ \text{false} & \text{otherwise} \end{cases} \tag{3}$$

$$Int(C) = \max_{e \in MST(C,E)} w(e) .$$

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j)) .$$

where the minimum internal difference, $MInt$, is defined as,

$$MInt(C_1, C_2) = \min(Int(C_1) + \tau(C_1),\ Int(C_2) + \tau(C_2)). \tag{4}$$

由参考论文可知，$Dif(C_1,C_2)$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）。$MInt(C_1,C_2)$代表边的距离加上阈值函数。

同时根据论文所说（见下页）：

The threshold function τ controls the degree to which the difference between two components must be greater than their internal differences in order for there to be evidence of a boundary between them (D to be true). For small components, $Int(C)$ is not a good estimate of the local characteristics of the data. In the extreme case, when $|C| = 1$, $Int(C) = 0$. Therefore, we use a threshold function based on the size of the component,

$$\tau(C) = k/|C| \tag{5}$$

where $|C|$ denotes the size of C, and k is some constant parameter. That is, for small components we require stronger evidence for a boundary. In practice k sets a scale of observation, in that a larger k causes a preference for larger components. Note, however, that k is not a minimum component size. Smaller components are allowed when there is a sufficiently large difference between neighboring components.

我们可以将 *Int(C)*默认为 0，所以$MInt(C_1, C_2)$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$\mathrm{Int}(A)+\tau(A)$（边的长度+$\frac{k}{|C|}$，其中$|C|$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:
```

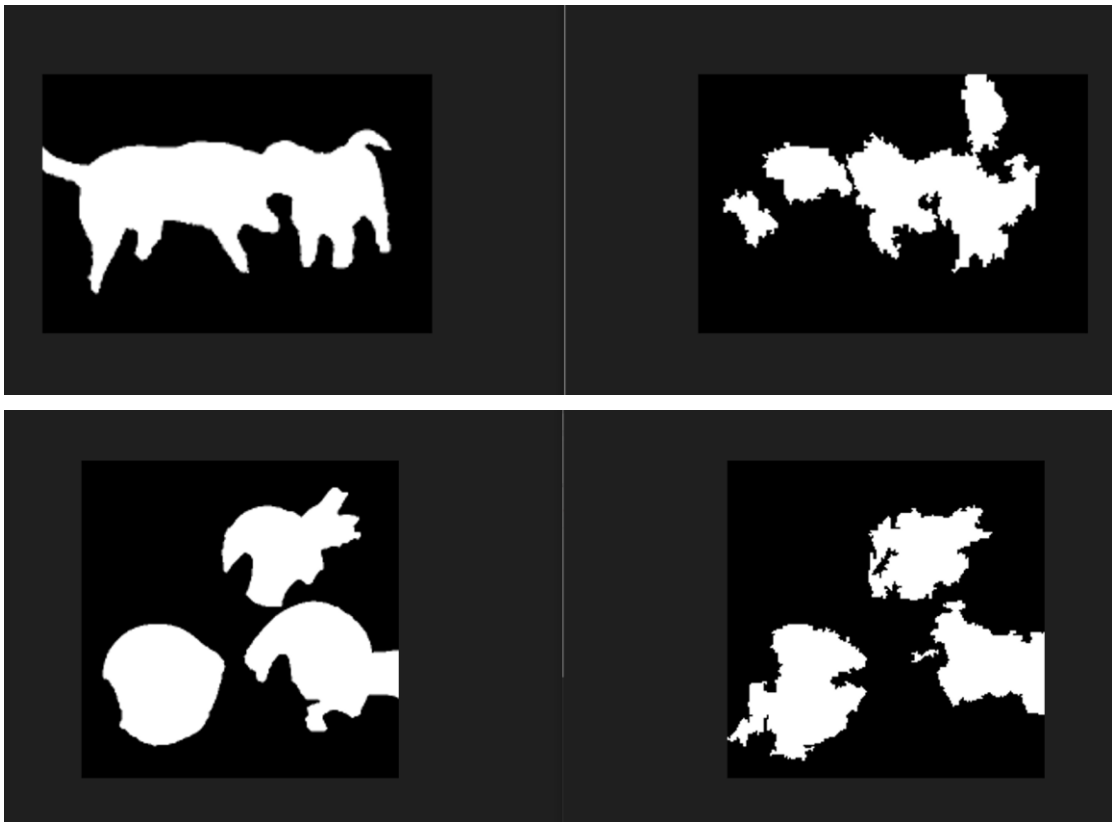
```
        #每个簇的像素点数量不得少于 min_num
        min_num = 50
        img = cv2.imread("C:/Users/93508/Desktop/Final/data/imgs/" +
img_number + ".png",cv2.IMREAD_COLOR)
        img_gt = cv2.imread("C:/Users/93508/Desktop/Final/data/gt/" +
img_number + ".png",cv2.IMREAD_GRAYSCALE)

        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/Segmentat
ion/" + 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))
```

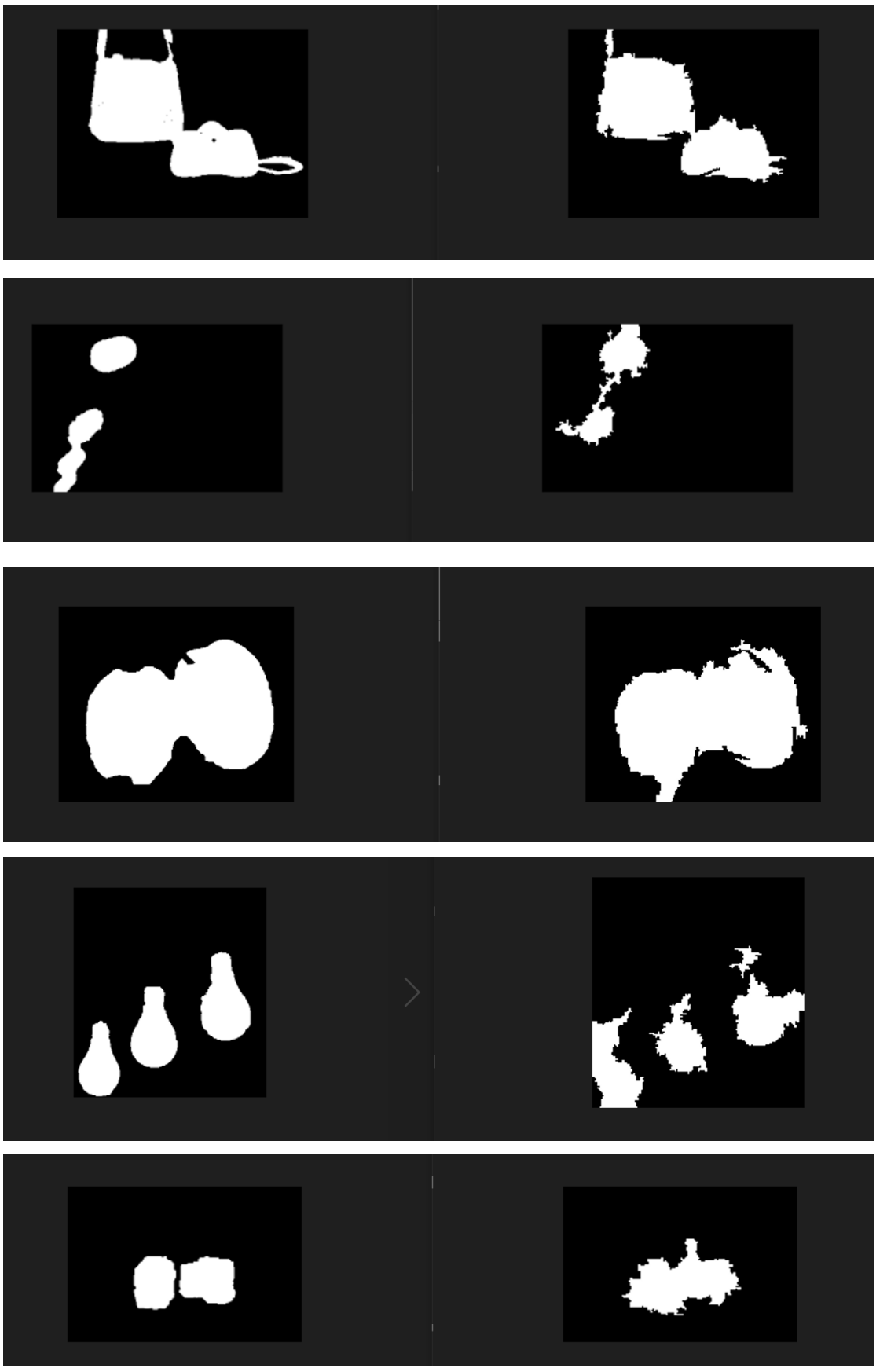
3.3 实验结果

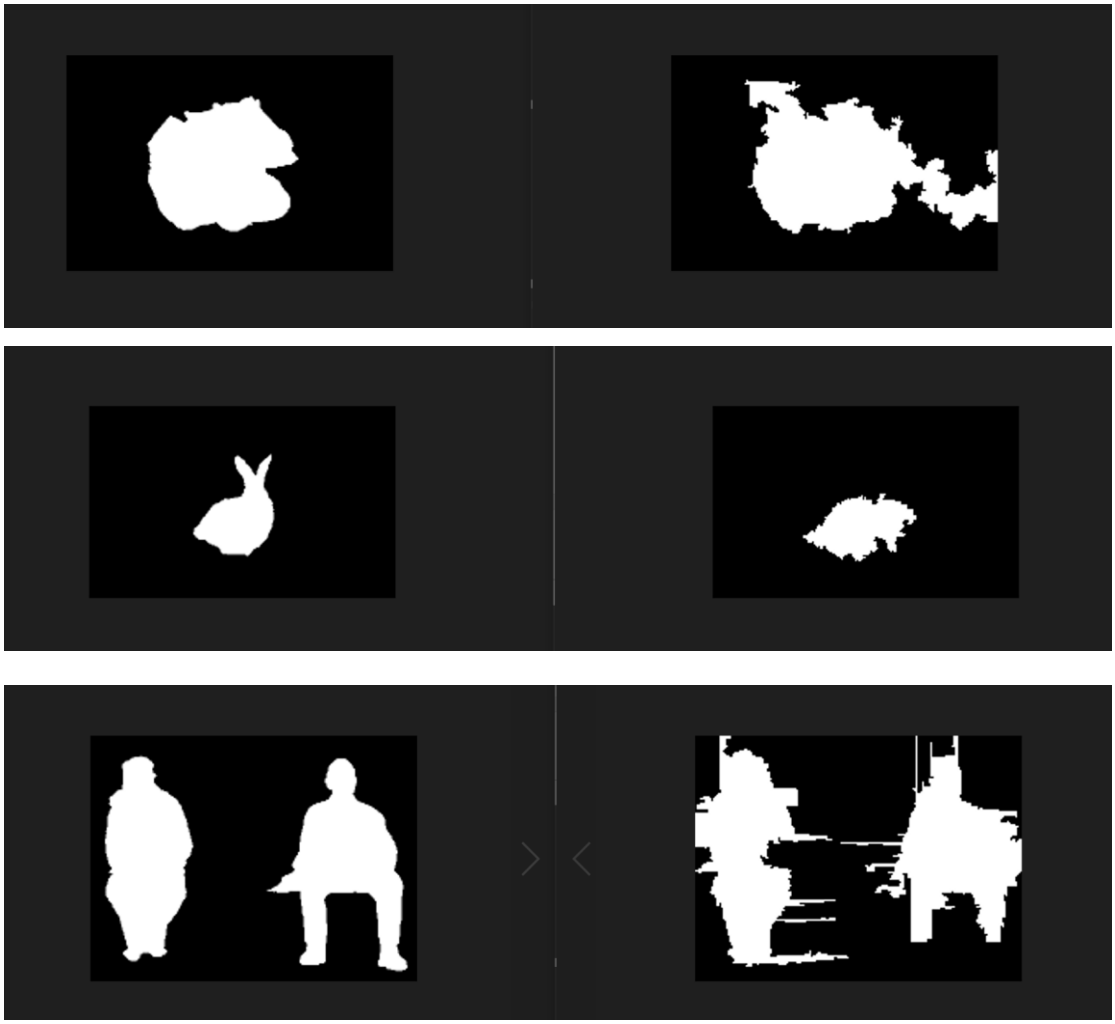
经过对 k 值的循环遍历，我得出最佳的 k 值为 130.（其中 Average num_sets 代表 10 张图的平均簇数，Correct num_sets 代表满足簇的个数在 50-70 之间的图片数量，Average Correctness 代表 10 张图的平均 IOU）

```
        #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))
```

```
15_result.png    37    54.79%
115_result.png    61    66.20%
215_result.png    58    83.51%
315_result.png    35    49.74%
415_result.png    59    88.08%
515_result.png    47    58.97%
615_result.png    110    71.11%
715_result.png    60    68.53%
815_result.png    30    60.15%
915_result.png    86    70.91%

K = 130
Average num_sets = 58.3
Correct num_sets = 4
Average Correctness = 67.20%
```







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 = $\frac{1}{m}XX^T$

```
    #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XX^T，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：

降维后的特征向量为：

0.079 0.082 0.082 0.082 0.085 0.085 0.084 0.084 0.078 0.081 0.076 0.077

0.066 0.068 0.069 0.071 0.073 0.072 0.073 0.074 0.071 0.073 0.068 0.071

0.054 0.055 0.054 0.055 0.056 0.057 0.058 0.056 0.055 0.057 0.056 0.057

0.072 0.070 0.066 0.068 0.070 0.070 0.071 0.070 0.066 0.069 0.068 0.067

0.094 0.093 0.090 0.088 0.087 0.090 0.091 0.090 0.092 0.096 0.091 0.088

0.102 0.102 0.100 0.100 0.098 0.098 0.099 0.100 0.103 0.105 0.100 0.096

0.104 0.104 0.102 0.101 0.102 0.099 0.102 0.105 0.105 0.105 0.100 0.098

0.089 0.089 0.088 0.089 0.091 0.089 0.090 0.091 0.093 0.092 0.088 0.087

0.068 0.068 0.067 0.070 0.072 0.070 0.072 0.072 0.073 0.070 0.068 0.065

0.076 0.074 0.073 0.073 0.075 0.075 0.076 0.078 0.076 0.075 0.073 0.073

0.091 0.093 0.090 0.089 0.093 0.092 0.095 0.096 0.091 0.088 0.086 0.087

0.090 0.094 0.091 0.091 0.093 0.095 0.097 0.092 0.090 0.089 0.086 0.084

Sklearn 库的 PCA：

降维后的特征向量为：

0.079	0.082	0.082	0.082	0.085	0.085	0.084	0.084	0.078	0.081	0.076	0.077
0.066	0.068	0.069	0.071	0.073	0.072	0.073	0.074	0.071	0.073	0.068	0.071
0.054	0.055	0.054	0.055	0.056	0.057	0.058	0.056	0.055	0.057	0.056	0.057
0.072	0.070	0.066	0.068	0.070	0.070	0.071	0.070	0.066	0.069	0.068	0.067
0.094	0.093	0.090	0.088	0.087	0.090	0.091	0.090	0.092	0.096	0.091	0.088
0.102	0.102	0.100	0.100	0.098	0.098	0.099	0.100	0.103	0.105	0.100	0.096
0.104	0.104	0.102	0.101	0.102	0.099	0.102	0.105	0.105	0.105	0.100	0.098
0.089	0.089	0.088	0.089	0.091	0.089	0.090	0.091	0.093	0.092	0.088	0.087
0.068	0.068	0.067	0.070	0.072	0.070	0.072	0.072	0.073	0.070	0.068	0.065
0.076	0.074	0.073	0.073	0.075	0.075	0.076	0.078	0.076	0.075	0.073	0.073
0.091	0.093	0.090	0.089	0.093	0.092	0.095	0.096	0.091	0.088	0.086	0.087
0.090	0.094	0.091	0.091	0.093	0.095	0.097	0.092	0.090	0.089	0.086	0.084

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 $u_i * \sigma_i * v_i = (-u_i) * \sigma_i * (-v_i)$，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！