School of Data and Computer Science, Sun Yat-sen University

Report of Final Project

18340013 陈琮昊

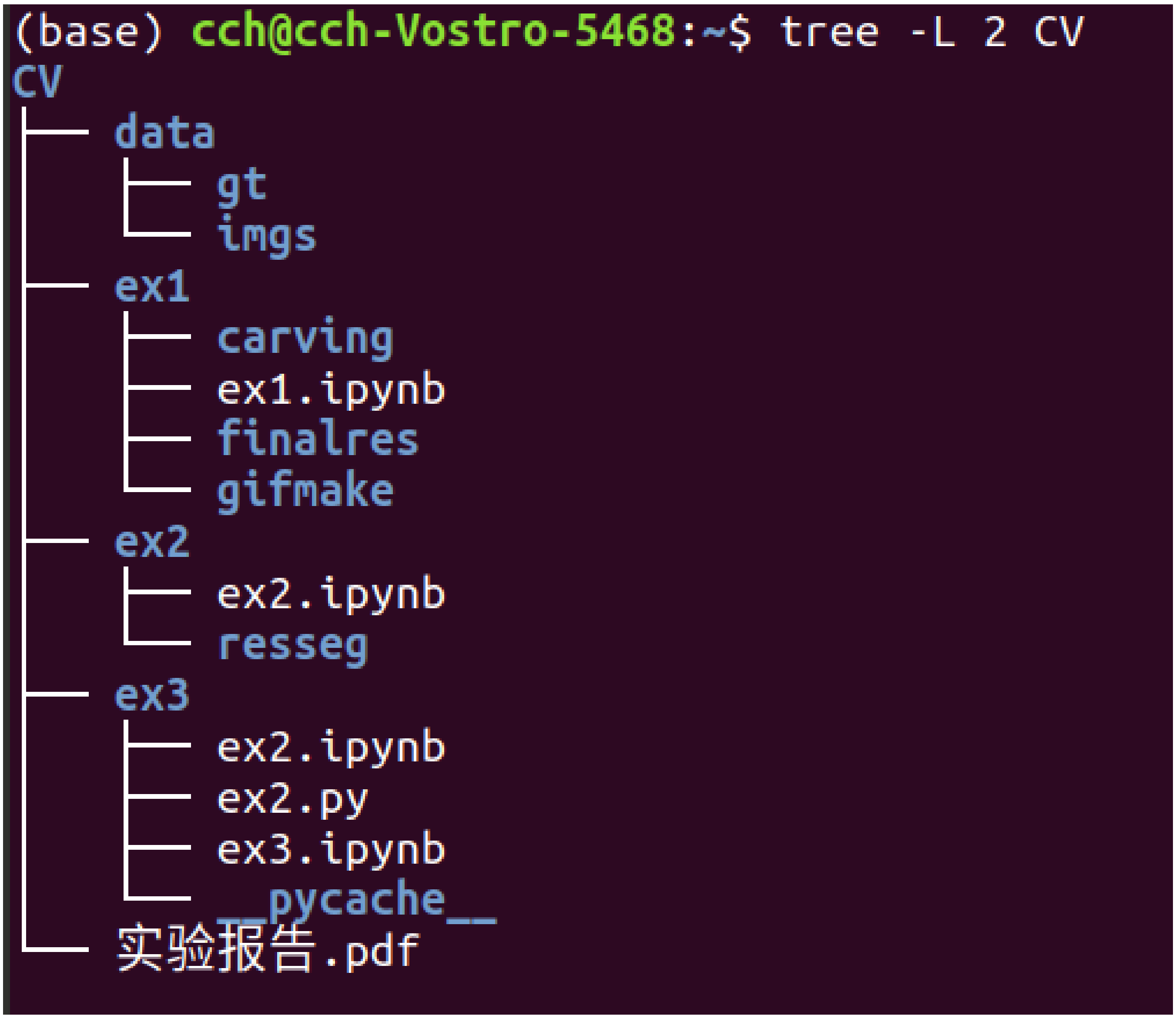
July 12, 2021

Contents

1 任务介绍

本次大作业需完成如下三道题，撰写实验报告，需提交可运行的源代码及实验测试结果图（建议用 python）。作业包含数据集 data.zip 一份，数据集包含 imgs 和 gt 两个文件夹。gt 中的图片为 imgs 对应图像的前景区域标注，图片名称为 XXXX.png，其中 XXXX 为图像 id。每位同学的实验测试图像为图像子集，该集合满足图像命名后两位与自己学号后两位相同，其余图像作为训练图像。

本次实验的文件目录结构如下；



```
(base) cch@cch-Vostro-5468:~$ tree -L 2 CV
CV
├── data
│   ├── gt
│   └── imgs
├── ex1
│   ├── carving
│   ├── ex1.ipynb
│   ├── finalres
│   └── gifmake
├── ex2
│   ├── ex2.ipynb
│   └── resseg
├── ex3
│   ├── ex2.ipynb
│   ├── ex2.py
│   ├── ex3.ipynb
│   └── __pycache__
└── 实验报告.pdf
```

Figure 1: 文件结构

为了方便调试，三道题目我都使用了 jupyter notebook 来编写代码，所以三道题目的代码文件分别为 ex1.ipynb，ex2.ipynb，ex3.ipynb。

注：由于我的学号后两位为 13，因此 13.png，113.png......913.png 是我的测试图像。

2 任务 1

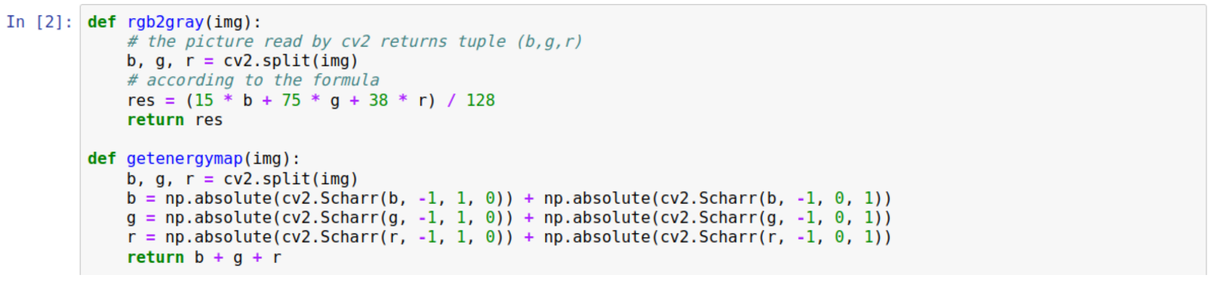
2.1 问题描述

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 求解过程

本次实验只实现了“缩”的过程，并未实现“放”的过程。由于已经给出了前景，因此我们的实现是前景保持的 Seam Carving。

- 将图像转为灰度图，计算能量值：



```
In [2]: def rgb2gray(img):
            # the picture read by cv2 returns tuple (b,g,r)
            b, g, r = cv2.split(img)
            # according to the formula
            res = (15 * b + 75 * g + 38 * r) / 128
            return res

        def getenergymap(img):
            b, g, r = cv2.split(img)
            b = np.absolute(cv2.Scharr(b, -1, 1, 0)) + np.absolute(cv2.Scharr(b, -1, 0, 1))
            g = np.absolute(cv2.Scharr(g, -1, 1, 0)) + np.absolute(cv2.Scharr(g, -1, 0, 1))
            r = np.absolute(cv2.Scharr(r, -1, 1, 0)) + np.absolute(cv2.Scharr(r, -1, 0, 1))
            return b + g + r
```

Figure 2: 转灰度图、得到能量值

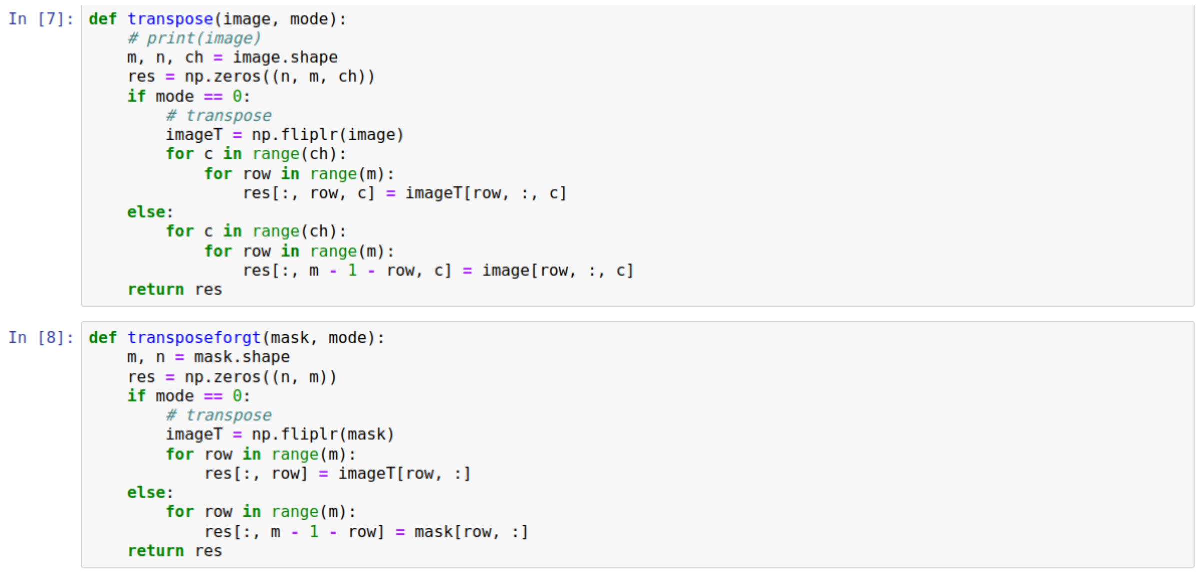
- 使用动态规划的方法找到最小的能量线：首先执行 Forward Seam Removing 算法，用动态规划的思想找到最后一行最小的能量值；在寻找过程中我们用一个数组记录下来，方便进行回溯；通过回溯就可以得到最小的能量线并将其删除，此时行数（列数）会减 1（见 reduce_col 函数）. 这部分的代码如下：

```
def dynamicforward(energymap, img):
    # forward seam removing(dynamic programming):
    I = rgb2gray(img)
    m, n = energymap.shape
    res = np.copy(energymap)
    for row in range(1, m):
        for col in range(n):
            if col == 0:
                rightup = res[row - 1, col + 1] + np.abs(I[row, col
                    + 1] - I[row, col]) + np.abs(I[row - 1, col] -
                    I[row, col + 1])
                middleup = res[row - 1, col] + np.abs(I[row, col +
                    1] - I[row, col])
                res[row, col] = energymap[row, col] + min(rightup,
                    middleup)
            elif col == n - 1:
                upleft = res[row - 1, col - 1] + np.abs(I[row, col]
                    - I[row, col - 1]) + np.abs(I[row - 1, col] -
                    I[row, col - 1])
                middleup = res[row - 1, col] + np.abs(I[row, col] -
                    I[row, col - 1])
                res[row, col] = energymap[row, col] + min(upleft,
                    middleup)
            else:
                upleft = res[row - 1, col - 1] + np.abs(I[row, col
                    + 1] - I[row, col - 1]) + np.abs(I[row - 1, col]
                    - I[row, col - 1])
                rightup = res[row - 1, col + 1] + np.abs(I[row, col
                    + 1] - I[row, col - 1]) + np.abs(I[row - 1, col]
                    - I[row, col + 1])
                middleup = res[row - 1, col] + np.abs(I[row, col +
                    1] - I[row, col - 1])
                res[row, col] = energymap[row, col] + min(upleft,
                    rightup, middleup)
    return res


def minpath(finalmap):
    m, n = finalmap.shape
    res = np.zeros((m,), dtype=np.uint32)
    res[-1] = np.argmin(finalmap[-1])
    for i in range(m - 2, -1, -1):
        if res[i + 1] == 0:
            res[i] = np.argmin(finalmap[i, : 2])
        elif res[i + 1] == (n - 1):
            res[i] = np.argmin(finalmap[i, n - 2: n]) + n - 2
        else:
            res[i] = np.argmin(finalmap[i, res[i + 1] - 1: res[i +
               1] + 2]) + res[i + 1] - 1
    return res

def deleteminpath(input, seamline):
    m, n = input.shape[: 2]
    res = np.zeros((m, n - 1, 3))
    for row in range(m):
        col = seamline[row]
        res[row, :, 0] = np.delete(input[row, :, 0], [col])
        res[row, :, 1] = np.delete(input[row, :, 1], [col])
        res[row, :, 2] = np.delete(input[row, :, 2], [col])
    input = np.copy(res)
    return input

def reduce_col(num, mode, gt, dynamic, resize):
    for i in range(resize):
        energymap = getenergymap(dynamic)
        energymap[np.where(gt > 0)] *= 1000
        finalmap = dynamicforward(energymap, dynamic)
        seamline = minpath(finalmap)
        # print(len(seamline))
        if num == 6 or num == 9:
            temp = np.copy(dynamic)
            gifpre(temp, seamline, mode, num, i)
        dynamic = deleteminpath(dynamic, seamline)
        gt = deleteminpathforgt(gt, seamline)
    return dynamic, gt
```

注意：由于我们实现的是前景保持的 `Seam Carving` 算法，为了能够更好的让前景作为参照，因此在这个过程中，前景图像也要跟着进行变化：

```
def deleteminpathforgt(gt, seamline):
    m, n = gt.shape
    res = np.zeros((m, n - 1))
    for row in range(m):
        col = seamline[row]
        res[row, :] = np.delete(gt[row, :], [col])
    gt = np.copy(res)
    return gt
```

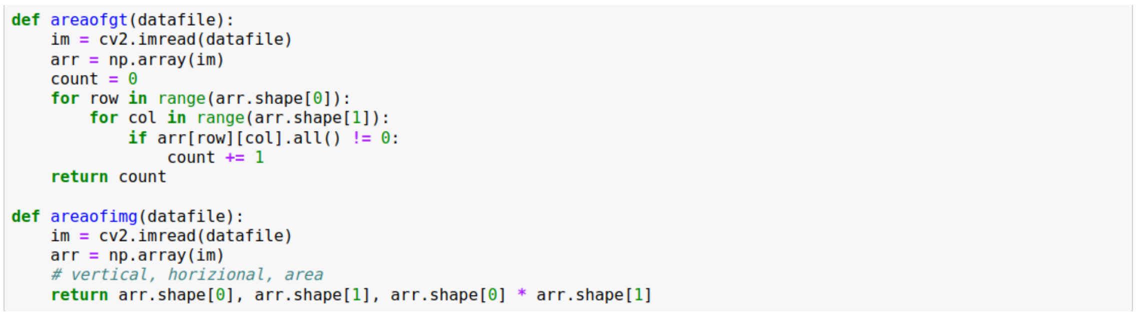
在删除最小能量线的过程中，我们首先只对列进行删除，至于对行删除，则可以通过将图片转置，对转置后的图片的列进行删除，就达到了对行删除的效果。最后再将图片转置回来即可：



```
In [7]: def transpose(image, mode):
            # print(image)
            m, n, ch = image.shape
            res = np.zeros((n, m, ch))
            if mode == 0:
                # transpose
                imageT = np.fliplr(image)
                for c in range(ch):
                    for row in range(m):
                        res[:, row, c] = imageT[row, :, c]
            else:
                for c in range(ch):
                    for row in range(m):
                        res[:, m - 1 - row, c] = image[row, :, c]
            return res

In [8]: def transposeforgt(mask, mode):
            m, n = mask.shape
            res = np.zeros((n, m))
            if mode == 0:
                # transpose
                imageT = np.fliplr(mask)
                for row in range(m):
                    res[:, row] = imageT[row, :]
            else:
                for row in range(m):
                    res[:, m - 1 - row] = mask[row, :]
            return res
```

Figure 3: 转置操作

- 重复上述步骤。根据压缩比例的要求，经计算可以得到新图片的长和宽。因此我们可以计算出需要删去的行数与列数，将这两个参数传入 reduce_col 函数中，通过执行该函数得到最终要的图片。



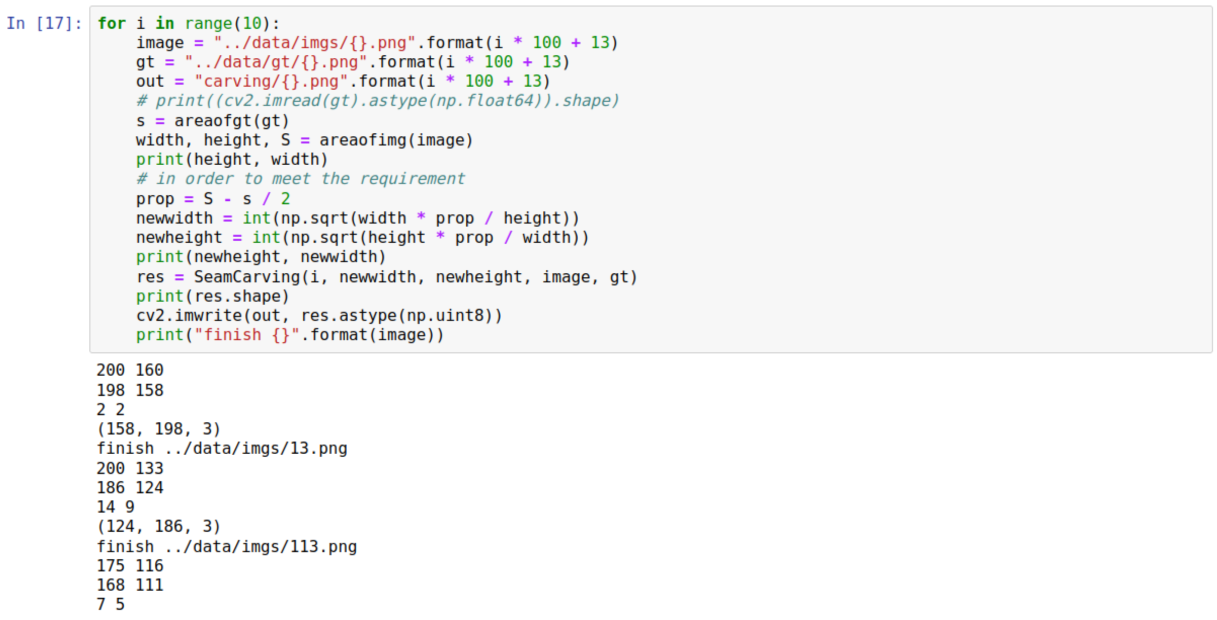
```
def areaofgt(datafile):
    im = cv2.imread(datafile)
    arr = np.array(im)
    count = 0
    for row in range(arr.shape[0]):
        for col in range(arr.shape[1]):
            if arr[row][col].all() != 0:
                count += 1
    return count

def areaofimg(datafile):
    im = cv2.imread(datafile)
    arr = np.array(im)
    # vertical, horizional, area
    return arr.shape[0], arr.shape[1], arr.shape[0] * arr.shape[1]
```

Figure 4: 计算面积得到压缩比例

整个 Seam Carving 的代码如下：

```
def SeamCarving(th, nwidth, nheight, input, mask):
    imgs = cv2.imread(input).astype(np.float64)
    gt = cv2.imread(mask, 0).astype(np.float64)
    cal = np.copy(imgs)
   # print(type(cal))
    widthreduce = imgs.shape[0] - nwidth
    highreduce = imgs.shape[1] - nheight
    print(highreduce, widthreduce)
   # exception, return original
    if highreduce < 0 or widthreduce < 0:
        print(imgs.shape[0], imgs.shape[1], nwidth, nheight,
            highreduce, widthreduce)
        cv2.imwrite(input, imgs.astype(np.uint8))
   # column delete
    cal, gt = reduce_col(th, 0, gt, cal, highreduce)
   # print(type(cal))
   # transpose img and gt
    cal = transpose(cal, 0)
    gt = transposeforgt(gt, 0)
   # row delete
    cal, gt = reduce_col(th, 1, gt, cal, widthreduce)
   # transpose to return original shape
    cal = transpose(cal, 1)
    gt = transposeforgt(gt, 1)
    return cal
```

主函数如下，调用 SeamCarving 函数来完成压缩：



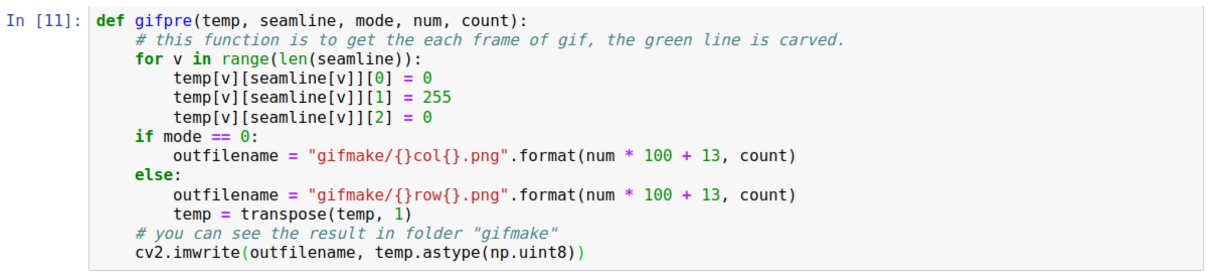
```
In [17]: for i in range(10):
             image = "../data/imgs/{}.png".format(i * 100 + 13)
             gt = "../data/gt/{}.png".format(i * 100 + 13)
             out = "carving/{}.png".format(i * 100 + 13)
             # print((cv2.imread(gt).astype(np.float64)).shape)
             s = areaofgt(gt)
             width, height, S = areaofimg(image)
             print(height, width)
             # in order to meet the requirement
             prop = S - s / 2
             newwidth = int(np.sqrt(width * prop / height))
             newheight = int(np.sqrt(height * prop / width))
             print(newheight, newwidth)
             res = SeamCarving(i, newwidth, newheight, image, gt)
             print(res.shape)
             cv2.imwrite(out, res.astype(np.uint8))
             print("finish {}".format(image))
```

```
200 160
198 158
2 2
(158, 198, 3)
finish ../data/imgs/13.png
200 133
186 124
14 9
(124, 186, 3)
finish ../data/imgs/113.png
175 116
168 111
7 5
```

Figure 5: 主函数

最后就是生成动图来展示他们的压缩过程，我选择了两张图片分别是 `613.png` 和 `913.png`：

首先将剪裁过程中每删除一行/列时得到的结果存储起来：



```
In [11]: def gifpre(temp, seamline, mode, num, count):
             # this function is to get the each frame of gif, the green line is carved.
             for v in range(len(seamline)):
                 temp[v][seamline[v]][0] = 0
                 temp[v][seamline[v]][1] = 255
                 temp[v][seamline[v]][2] = 0
             if mode == 0:
                 outfilename = "gifmake/{}col{}.png".format(num * 100 + 13, count)
             else:
                 outfilename = "gifmake/{}row{}.png".format(num * 100 + 13, count)
                 temp = transpose(temp, 1)
             # you can see the result in folder "gifmake"
             cv2.imwrite(outfilename, temp.astype(np.uint8))
```

然后再将这些图片作为每一帧拼接起来即可得到 gif：

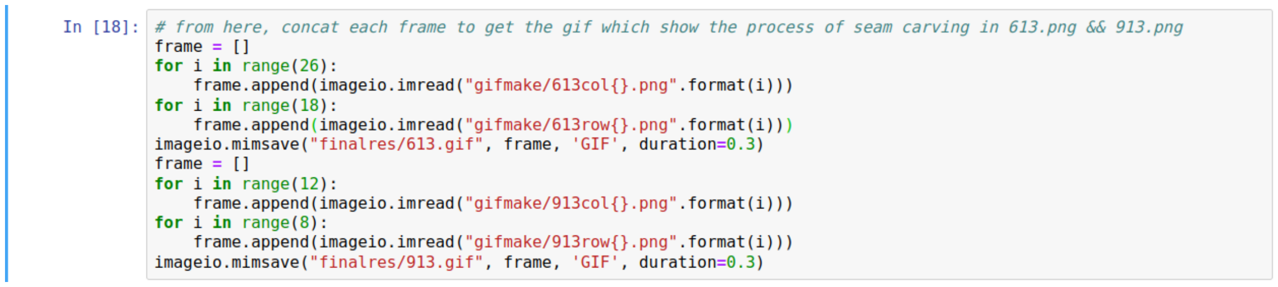

```
In [18]: # from here, concat each frame to get the gif which show the process of seam carving in 613.png && 913.png
frame = []
for i in range(26):
    frame.append(imageio.imread("gifmake/613col{}.png".format(i)))
for i in range(18):
    frame.append(imageio.imread("gifmake/613row{}.png".format(i)))
imageio.mimsave("finalres/613.gif", frame, 'GIF', duration=0.3)
frame = []
for i in range(12):
    frame.append(imageio.imread("gifmake/913col{}.png".format(i)))
for i in range(8):
    frame.append(imageio.imread("gifmake/913row{}.png".format(i)))
imageio.mimsave("finalres/913.gif", frame, 'GIF', duration=0.3)
```

Figure 6: 获得动图

2.3 结果分析

选取了 `613.png`、`913.png` 来制作动图观察 `Seam Carving` 的过程，结果在 `ex1/finalres` 文件夹内。在对这两幅图像进行 `Seam Carving` 时，每压缩一次就将压缩后的图片保留在在 `ex1/gifmake` 文件夹内。`gif` 动图的制作过程是在网上找的代码，所以也没对其进行封装，直接将文件夹内的静态图片当作每一帧拼在一起了；所以这段代码的可移植性不强（不过这也应该不是本次实验重点）。对于测试集的 10 张图像，其最终结果存放在 `ex1/carving` 内。由于实现时压缩比例是严格按照文档所给，并不是动态调整，所以压缩率较小，对于有的图像来说，可能还可以继续压缩；但最后的效果还是不错的，对于重要信息保留的很好。

3 任务 2

3.1 问题描述

结合 `Lecture 7 Segmentation` 内容及如下的参考文献，实现基于 Graph-based 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 至 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（区域 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，该图像的前景区域为 R2，则 IOU=$\frac{R1 \cap R2}{R1 \cup R2}$。
参考文献：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 求解过程

论文中提出的基于图的分割方法的主要思想如下：该函数主要构造一个并查集森林（并查集：按一定顺序将所属于同一组的元素所在的集合合并），该函数首先对边矩阵中的元素按边权值进行降序排序，并初始化并查集 u 和各区域阈值 threshold。随后，函数遍历每一条边，如果由这条边连接的两个区域的阈值均大于该边的权重，则将两个区域进行合并。

其中阈值（可动态调整，见后面 segment_image 函数）和相似性的定义如下：

```
def criteria(img, x1, y1, x2, y2):
    r = np.square(img[0][y1, x1] - img[0][y2, x2])
    g = np.square(img[1][y1, x1] - img[1][y2, x2])
    b = np.square(img[2][y1, x1] - img[2][y2, x2])
    return np.sqrt(r + g + b)


def THRESHOLD(size, c):
    return c / size
```

该任务代码参考了论文的开源代码，并做了些修改，整个代码所做的工作流程如下：

- 并查集森林定义于 universe 类中。universe 主要维护一个 shape 为 (num_vertices, 3) 的 elts 矩阵（num_vertices 为像素节点数量）。对节点 i，elts[i,0] 记录它在所属区域在并查集中所处的层数，elts[i,1] 记录其所属区域的大小，elts[i,2] 记录它的父节点。universe 主要包括三个方法：find 方法寻找一个节点 x 所属区域的并查集的根节点，用来表示它所属的区域；join 方法合并根节点 x 和 y 所代表的两个区域；comps 方法是为了查找当前记录下来的所有区域，返回一个记录所有区域根节点的列表。

```
# disjoint set
class universe():
    def __init__(self, elements):
        self.num = elements
        self.elts = []
        for i in range(elements):
            rank = 0
            size = 1
            p = i
            self.elts.append((rank, size, p))
```

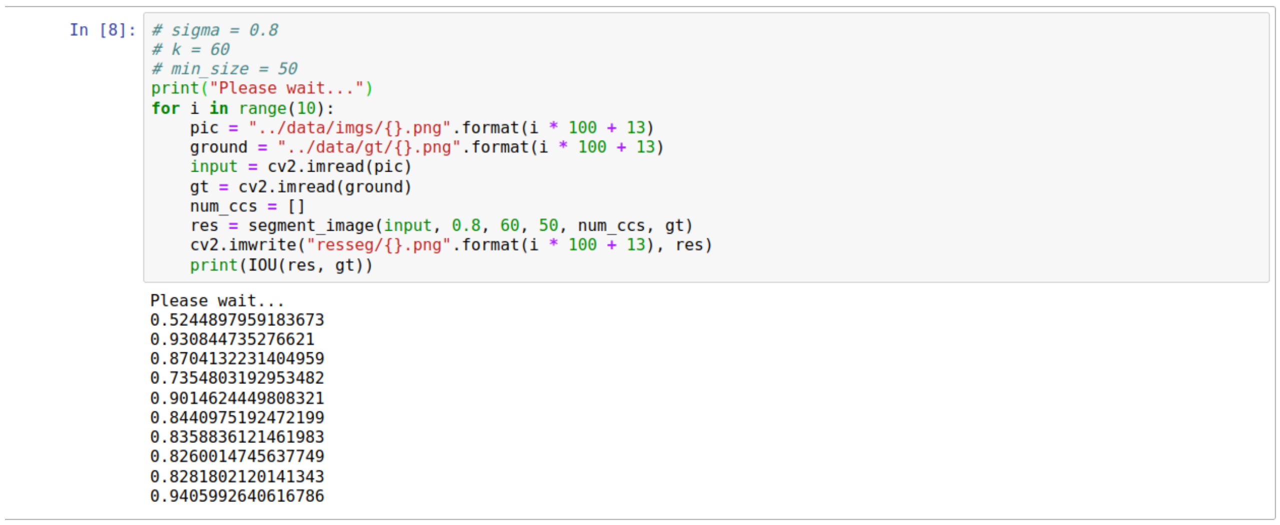
```
def find(self, u):
    if self.elts[u][2] == u:
        return u
    self.elts[u] = (self.elts[u][0], self.elts[u][1],
       self.find(self.elts[u][2]))
    return self.elts[u][2]

def join(self, x, y):
    if self.elts[x][0] > self.elts[y][0]:
        self.elts[y] = (self.elts[y][0], self.elts[y][1],
           self.elts[x][2])
        self.elts[x] = (self.elts[x][0], self.elts[x][1] +
           self.elts[y][1], self.elts[x][2])
    else:
        self.elts[x] = (self.elts[x][0], self.elts[x][1],
           self.elts[y][2])
        self.elts[y] = (self.elts[y][0], self.elts[y][1] +
           self.elts[x][1], self.elts[y][2])
        if self.elts[x][0] == self.elts[y][0]:
            self.elts[y] = (self.elts[y][0] + 1,
               self.elts[y][1], self.elts[y][2])
    self.num -= 1

def size(self, x):
    return self.elts[x][1]

def num_sets(self):
    return self.num

def comps(self):
    ret = []
    i, n = 0, 0
    while n != self.num:
            if self.elts[i][2] == i:
                ret.append(i)
                n += 1
            i += 1
        return ret
```

- 区域标记。实现于 segment_graph 和 segment_image 中。函数首先对输入图像进行分割，得到分割后的并查集森林。随后它统计森林中每一个区域在 gt 文件夹中对应图像被标记为前景 (255) 的像素数量，超过一半则属于前景，否则为后景。判断出来是前景/后景之后，将前景区域的所有像素置为 255，后景区域所有像素置为 0。

```
# Segment a graph
# Returns a disjoint-set forest representing the segmentation.
def segment_graph(num_vertices, num_edges, graph, c):
    # make a disjoint-set forest
    u = universe(num_vertices)
    # init thresholds
    threshold = np.zeros(num_vertices, dtype=float)
    for i in range(num_vertices):
        threshold[i] = THRESHOLD(1, c)
    # for each edge, in non-decreasing weight order...
    for i in range(num_edges):
        a = u.find(graph[i][0])
        b = u.find(graph[i][1])
        if a != b:
            if (graph[i][2] <= threshold[a]) and graph[i][2] <=
                threshold[b]:
                u.join(a, b)
                a = u.find(a)
                threshold[a] = graph[i][2] + THRESHOLD(u.size(a), c)
    return u


# Segment an image
# This function is used to solve homework ex2
# Returns the final gt(after mark)
def segment_image(im, sigma, c, min_size, num_ccs, gt):
    height, width, channel = im.shape
    im = np.array(im, dtype=float)
    gaussian_img = cv2.GaussianBlur(im, (5, 5), sigma)
    b, g, r = cv2.split(gaussian_img)
    smooth_img = (r, g, b)
    graph = []
    num = 0
    for y in range(height):
        for x in range(width):
            if x < width - 1:
                a = y * width + x
                b = y * width + (x + 1)
                w = criteria(smooth_img, x, y, x + 1, y)
                num += 1
                graph.append((a, b, w))
            if y < height - 1:
                a = y * width + x
                b = (y + 1) * width + x
                w = criteria(smooth_img, x, y, x, y + 1)
                num += 1
                graph.append((a, b, w))
            if x < width - 1 and y < height - 1:
                a = y * width + x
                b = (y + 1) * width + (x + 1)
                w = criteria(smooth_img, x, y, x + 1, y + 1)
                num += 1
                graph.append((a, b, w))
            if x < width - 1 and y > 0:
                a = y * width + x
                b = (y - 1) * width + (x + 1)
                w = criteria(smooth_img, x, y, x + 1, y - 1)
                num += 1
                graph.append((a, b, w))
    # sort according to the similarity
    graph = sorted(graph, key=lambda x: (x[2]))
    u = segment_graph(width * height, num, graph, c)
    for i in range(num):
        a = u.find(graph[i][0])
        b = u.find(graph[i][1])
        # if the number of pixel in each area < 50, concat.
        if (a != b) and ((u.size(a) < min_size) or u.size(b) <
            min_size):
            u.join(a, b)
    # dynamic adjust to keep the number of area is [50,70]
    while u.num_sets() < 50 or u.num_sets() > 70:
        if u.num_sets() < 50:
            c = c / 2
            u = segment_graph(width * height, num, graph, c)
        if u.num_sets() > 70:
            c = c * 1.5
            u = segment_graph(width * height, num, graph, c)
        for i in range(num):
            a = u.find(graph[i][0])
            b = u.find(graph[i][1])
            if (a != b) and ((u.size(a) < min_size) or u.size(b) <
                min_size):
                u.join(a, b)
    num_ccs.append(u.num_sets())
    comps = u.comps()
    # calculate the number of white pixel(gt) in each area
    cnt = [0 for i in range(u.num_sets())]
    # used to mark the segmentation area
    mark = [0 for i in range(u.num_sets())]
    for y in range(height):
        for x in range(width):
            comp = u.find(y * width + x)
            # Here I use data/gt as reference, if the current pixel
                in data/gt is gt, +1
            if (gt[y, x, :] == [255, 255, 255]).all():
                cnt[comps.index(comp)] += 1
    # if the white pixel is more than half, then mark the area into
        gt.
    for i in range(u.num_sets()):
        if 2 * cnt[i] >= u.size(comps[i]):
            mark[i] = 1
        else:
            mark[i] = 0
    # final gt after mark
    for y in range(height):
        for x in range(width):
            comp = u.find(y * width + x)
            if mark[comps.index(comp)] == 1:
                gaussian_img[y, x, :] = [255, 255, 255]
            else:
                gaussian_img[y, x, :] = [0, 0, 0]

    return gaussian_img
```

- 主函数。执行图像分割，将区域标记后的结果写入 `ex2/resseg` 文件夹内，并计算 IOU：

```
def IOU(resseg, gt):
    height, width, channel = resseg.shape
    I, U = 0, 0
    for y in range(height):
        for x in range(width):
            if (gt[y, x, :] == [255, 255, 255]).all() and
                (resseg[y, x, :] == [255, 255, 255]).all():
                I += 1
            if (gt[y, x, :] == [255, 255, 255]).all() or (resseg[y,
                x, :] == [255, 255, 255]).all():
                U += 1
    return float(I) / float(U)
```



```
In [8]: # sigma = 0.8
        # k = 60
        # min_size = 50
        print("Please wait...")
        for i in range(10):
            pic = "../data/imgs/{}.png".format(i * 100 + 13)
            ground = "../data/gt/{}.png".format(i * 100 + 13)
            input = cv2.imread(pic)
            gt = cv2.imread(ground)
            num_ccs = []
            res = segment_image(input, 0.8, 60, 50, num_ccs, gt)
            cv2.imwrite("resseg/{}.png".format(i * 100 + 13), res)
            print(IOU(res, gt))

        Please wait...
        0.5244897959183673
        0.930844735276621
        0.8704132231404959
        0.7354803192953482
        0.9014624449808321
        0.8440975192472199
        0.8358836121461983
        0.8260014745637749
        0.8281802120141343
        0.9405992640616786
```

Figure 7: 主函数与运行结果

- 注：由于任务 3 中还要用到图像分割，为了能够解决任务 3 的问题，我们需要对 segment_image 略微进行改写，改写后的函数名为 segment_image_forthree，实现于 `ex2.ipynb` 中。增加的部分就是获得每个分割区域的直方图以及返回值有所不同。在任务 3 的代码中我们只需要在文件头部加上 `from ex2 import segment_image_forthree`，并将 `ex2.ipynb` 导出至 `ex2.py` 放入 `ex3` 文件夹内即可。该函数代码如下：

```
# Segment an image
# This function is only for ex3 in homework
# return: gt(after mark), every little area(segmentation), the
    count of area, label(0:bg, 1:gt)
def segment_image_forthree(im, sigma, c, min_size, num_ccs, gt):
    height, width, channel = im.shape
    im = np.array(im, dtype=float)
    gaussian_img = cv2.GaussianBlur(im, (5, 5), sigma)
    b, g, r = cv2.split(gaussian_img)
    smooth_img = (r, g, b)
    graph = []
```

```
num = 0
for y in range(height):
    for x in range(width):
        if x < width - 1:
            a = y * width + x
            b = y * width + (x + 1)
            w = criteria(smooth_img, x, y, x + 1, y)
            num += 1
            graph.append((a, b, w))
        if y < height - 1:
            a = y * width + x
            b = (y + 1) * width + x
            w = criteria(smooth_img, x, y, x, y + 1)
            num += 1
            graph.append((a, b, w))
        if x < width - 1 and y < height - 1:
            a = y * width + x
            b = (y + 1) * width + (x + 1)
            w = criteria(smooth_img, x, y, x + 1, y + 1)
            num += 1
            graph.append((a, b, w))
        if x < width - 1 and y > 0:
            a = y * width + x
            b = (y - 1) * width + (x + 1)
            w = criteria(smooth_img, x, y, x + 1, y - 1)
            num += 1
            graph.append((a, b, w))
# sort according to the similarity
graph = sorted(graph, key=lambda x: (x[2]))
u = segment_graph(width * height, num, graph, c)
for i in range(num):
    a = u.find(graph[i][0])
    b = u.find(graph[i][1])
```

```
        # if the number of pixel in each area < 50, concat.
        if (a != b) and ((u.size(a) < min_size) or u.size(b) <
            min_size):
             u.join(a, b)
    # dynamic adjust to keep the number of area is [50,70]
    while u.num_sets() < 50 or u.num_sets() > 70:
        if u.num_sets() < 50:
             c = c / 2
             u = segment_graph(width * height, num, graph, c)
        if u.num_sets() > 70:
             c = c * 1.5
             u = segment_graph(width * height, num, graph, c)
        for i in range(num):
             a = u.find(graph[i][0])
             b = u.find(graph[i][1])
             if (a != b) and ((u.size(a) < min_size) or u.size(b) <
                 min_size):
                  u.join(a, b)
    num_ccs.append(u.num_sets())
    comps = u.comps()
    # calculate the number of white pixel(gt) in each area
    cnt = [0 for i in range(u.num_sets())]
    # used to mark the segmentation area
    mark = [0 for i in range(u.num_sets())]
    for y in range(height):
        for x in range(width):
             comp = u.find(y * width + x)
             # Here I use data/gt as reference, if the current pixel
                 in data/gt is gt, +1
             if (gt[y, x, :] == [255, 255, 255]).all():
                  cnt[comps.index(comp)] += 1
    # if the white pixel is more than half, then mark the area into
        gt.
```

```
for i in range(u.num_sets()):
    if 2 * cnt[i] >= u.size(comps[i]):
        mark[i] = 1
    else:
        mark[i] = 0
block = {}
label = {}
# get the histogram of each little area after segmentation
for y in range(height):
    for x in range(width):
        comp = u.find(y * width + x)
        if comp not in block:
            block[comp] = np.zeros((8,8,8))
            block[comp][int(smooth_img[0][y][x] //
                32)][int(smooth_img[1][y][x] //
                32)][int(smooth_img[2][y][x] // 32)] += 1
        else:
            block[comp][int(smooth_img[0][y][x] //
                32)][int(smooth_img[1][y][x] //
                32)][int(smooth_img[2][y][x] // 32)] += 1
# flatten:(8 * 8 * 8) -> (1 * 512)
for comp in block:
    block[comp] = block[comp].flatten()
    sum = np.sum(block[comp])
    for i in range(len(block[comp])):
        block[comp][i] = block[comp][i] / sum
    # print(block[comp].shape)
# print(len(block))
# final gt after mark
for y in range(height):
    for x in range(width):
        comp = u.find(y * width + x)
        if mark[comps.index(comp)] == 1:
                gaussian_img[y, x, :] = [255, 255, 255]
                label[comp] = 1
            else:
                gaussian_img[y, x, :] = [0, 0, 0]
                label[comp] = 0

    return gaussian_img, block, u.num_sets(), label
```

3.3 结果分析

可以看到第一张图（13.png）的交并比较低，这幅图的确比较难获得理想的效果。我试着不断调参得到的最好结果就是上图所放的 0.5244。这里展示一下不同参数对应的结果图：

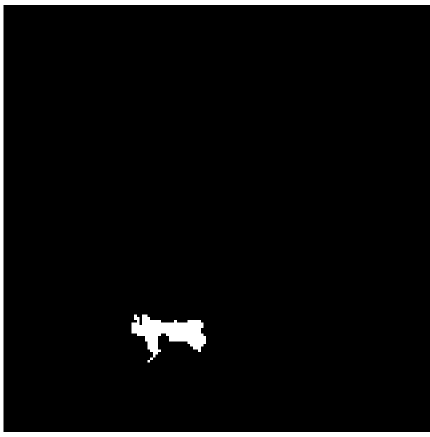


Figure 8: 其他参数下

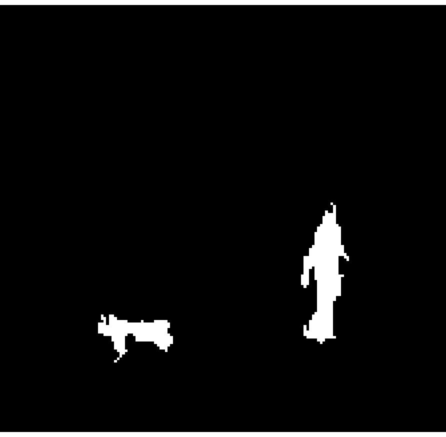


Figure 9: 该参数下

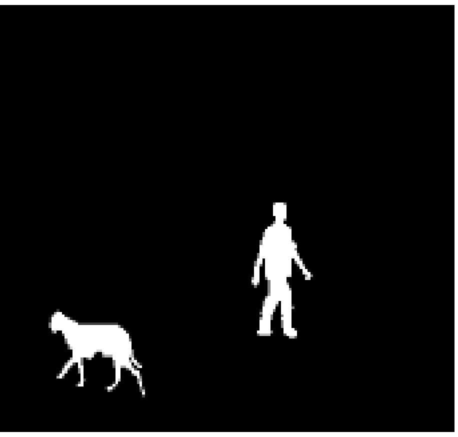


Figure 10: 所提供 gt

可以看到最左边那张图片完全没有识别出来原图片中的人；中间那张图片识别出来了人，之所以交并比较低是因为对动物和人的识别都不是很理想（不仔细看绝对看不出来是什么）。

4 任务 3

4.1 问题描述

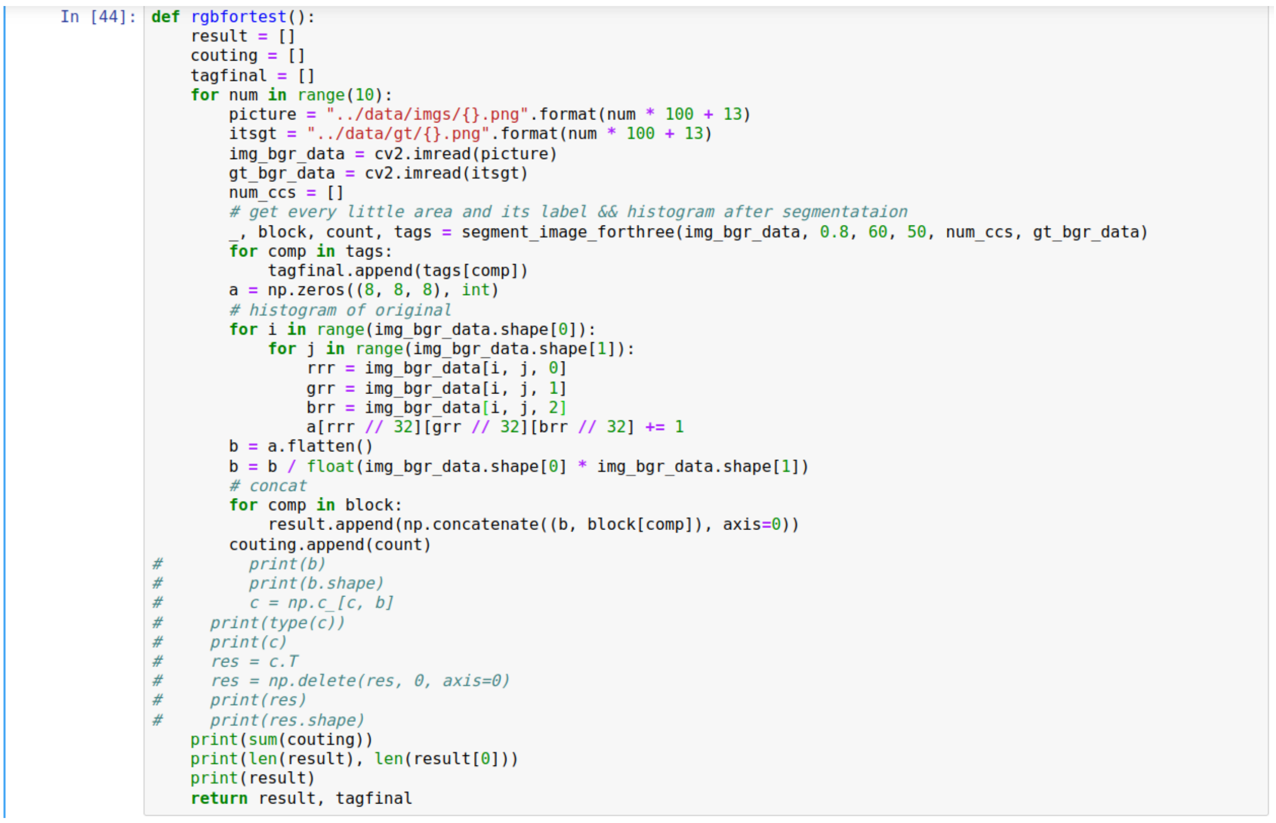
从训练集中随机选择 200 张图用以训练，对每一张图提取归一化 RGB 颜色直方图（维度为 8*8*8=512），同时执行任务 2 对其进行图像分割（分割为 50 至 70 个区域），对得到的每一个分割区域提取归一化 RGB 颜色直方图特征（维度为 8*8*8=512），将每一个区域的颜色对比度特征定义为区域颜色直方图和全图颜色直方图的拼接，因此区域颜色区域对比度特征的维度为 2*512=1024 维，采用 PCA 算法对特征进行降维取前 20 维。利用选择的 200 张图的所有区域（每个区域 20 维特

征）构建 Visual Bag of Words Dictionary（参考 Lecture 12. Visual Bag of Words 内容），单词数（聚类数）设置为 50 个，visual word 的特征设置为聚簇样本的平均特征，每个区域降维后颜色对比度特征（20 维）和各个 visual word 的特征算点积相似性得到 50 个相似性值形成 50 维。将得到的 50 维特征和前面的 20 维颜色对比度特征拼接得到每个区域的 70 维特征表示。根据任务 2，每个区域可以被标注为类别 1（前景：该区域 50% 以上像素为前景）或 0（背景：该区域 50% 以上像素为背景），选用任意分类算法（SVM，Softmax，随机森林，KNN 等）进行学习得到分类模型。最后在测试集上对每一张图的每个区域进行测试（将图像分割为 50 至 70 个区域，对每个区域提取同样特征并分类），根据测试图像的 GT，分析测试集区域预测的准确率。

4.2 求解过程

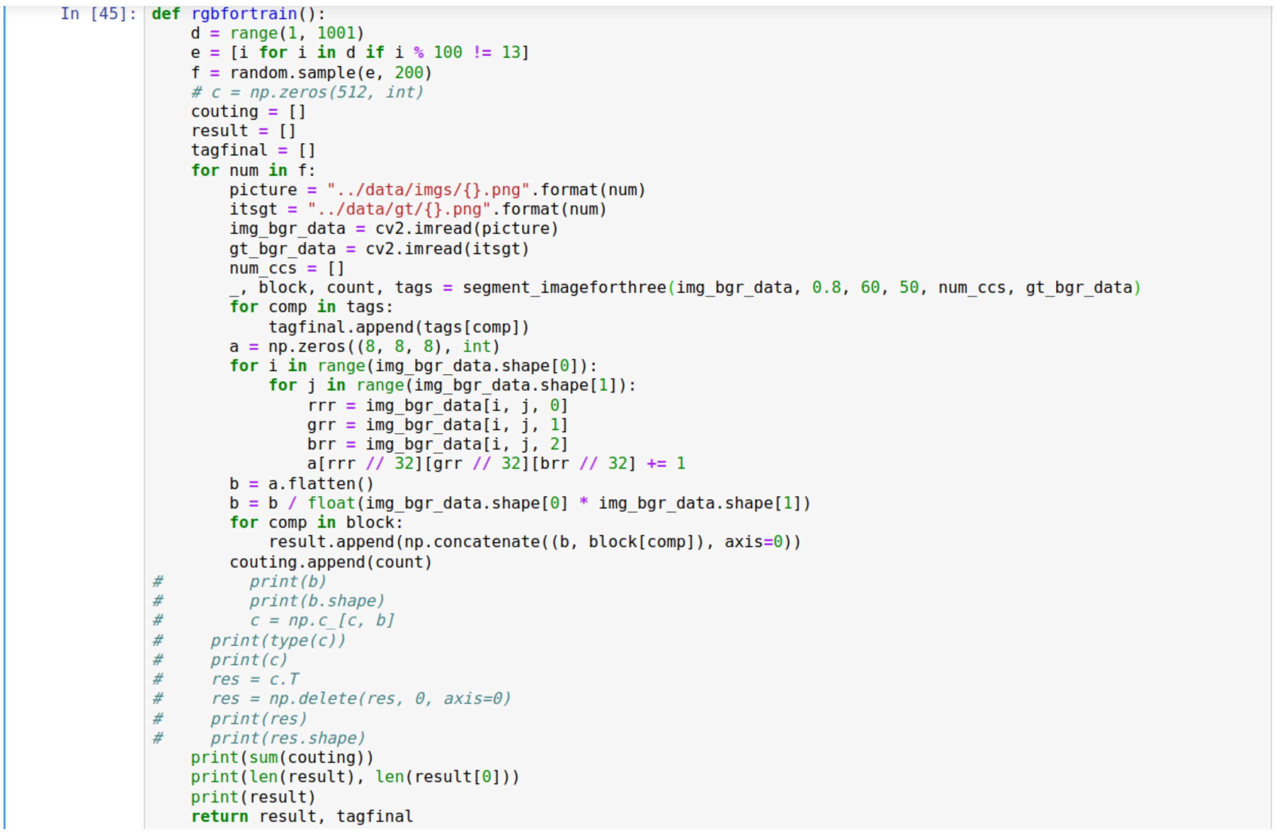
本任务需要在任务 2 的基础上进行，整个过程的流程如下：

- 获得 RGB 直方图。由于要求的维度为 8*8*8，因此对于每个通道可取值均为 0-255 的图片来说，我们需要将 32 个合并为一组，这样最终的直方图就是 8*8*8。然后我们将其拉伸成一维数组，数组长度为 512。至于后面的 512 维，一副图对应多张分割区域，我们通过略微改写任务 2 中的函数即可获得每个区域的 RGB 直方图，再将这两个长度为 512 的数组连接起来即可。代码如下：



```
In [44]: def rgbfortest():
             result = []
             couting = []
             tagfinal = []
             for num in range(10):
                 picture = "../data/imgs/{}.png".format(num * 100 + 13)
                 itsgt = "../data/gt/{}.png".format(num * 100 + 13)
                 img_bgr_data = cv2.imread(picture)
                 gt_bgr_data = cv2.imread(itsgt)
                 num_ccs = []
                 # get every little area and its label && histogram after segmentataion
                 _, block, count, tags = segment_image_forthree(img_bgr_data, 0.8, 60, 50, num_ccs, gt_bgr_data)
                 for comp in tags:
                     tagfinal.append(tags[comp])
                 a = np.zeros((8, 8, 8), int)
                 # histogram of original
                 for i in range(img_bgr_data.shape[0]):
                     for j in range(img_bgr_data.shape[1]):
                         rrr = img_bgr_data[i, j, 0]
                         grr = img_bgr_data[i, j, 1]
                         brr = img_bgr_data[i, j, 2]
                         a[rrr // 32][grr // 32][brr // 32] += 1
                 b = a.flatten()
                 b = b / float(img_bgr_data.shape[0] * img_bgr_data.shape[1])
                 # concat
                 for comp in block:
                     result.append(np.concatenate((b, block[comp]), axis=0))
                 couting.append(count)
         #         print(b)
         #         print(b.shape)
         #         c = np.c_[c, b]
         #     print(type(c))
         #     print(c)
         #     res = c.T
         #     res = np.delete(res, 0, axis=0)
         #     print(res)
         #     print(res.shape)
             print(sum(couting))
             print(len(result), len(result[0]))
             print(result)
             return result, tagfinal
```

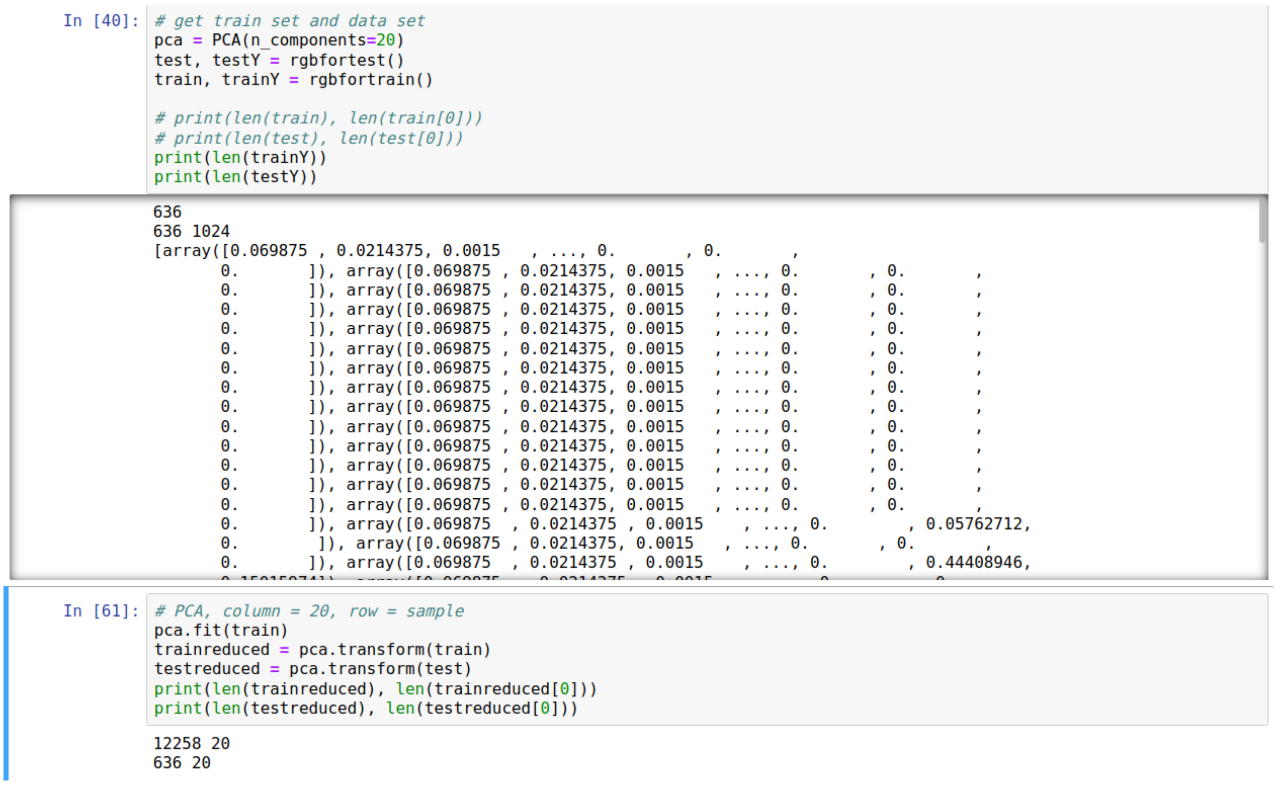
Figure 11: 测试集



```
In [45]: def rgbfortrain():
             d = range(1, 1001)
             e = [i for i in d if i % 100 != 13]
             f = random.sample(e, 200)
             # c = np.zeros(512, int)
             couting = []
             result = []
             tagfinal = []
             for num in f:
                 picture = "../data/imgs/{}.png".format(num)
                 itsgt = "../data/gt/{}.png".format(num)
                 img_bgr_data = cv2.imread(picture)
                 gt_bgr_data = cv2.imread(itsgt)
                 num_ccs = []
                 _, block, count, tags = segment_imageforthree(img_bgr_data, 0.8, 60, 50, num_ccs, gt_bgr_data)
                 for comp in tags:
                     tagfinal.append(tags[comp])
                 a = np.zeros((8, 8, 8), int)
                 for i in range(img_bgr_data.shape[0]):
                     for j in range(img_bgr_data.shape[1]):
                         rrr = img_bgr_data[i, j, 0]
                         grr = img_bgr_data[i, j, 1]
                         brr = img_bgr_data[i, j, 2]
                         a[rrr // 32][grr // 32][brr // 32] += 1
                 b = a.flatten()
                 b = b / float(img_bgr_data.shape[0] * img_bgr_data.shape[1])
                 for comp in block:
                     result.append(np.concatenate((b, block[comp]), axis=0))
                 couting.append(count)
         #         print(b)
         #         print(b.shape)
         #         c = np.c_[c, b]
         #     print(type(c))
         #     print(c)
         #     res = c.T
         #     res = np.delete(res, 0, axis=0)
         #     print(res)
         #     print(res.shape)
             print(sum(couting))
             print(len(result), len(result[0]))
             print(result)
             return result, tagfinal
```

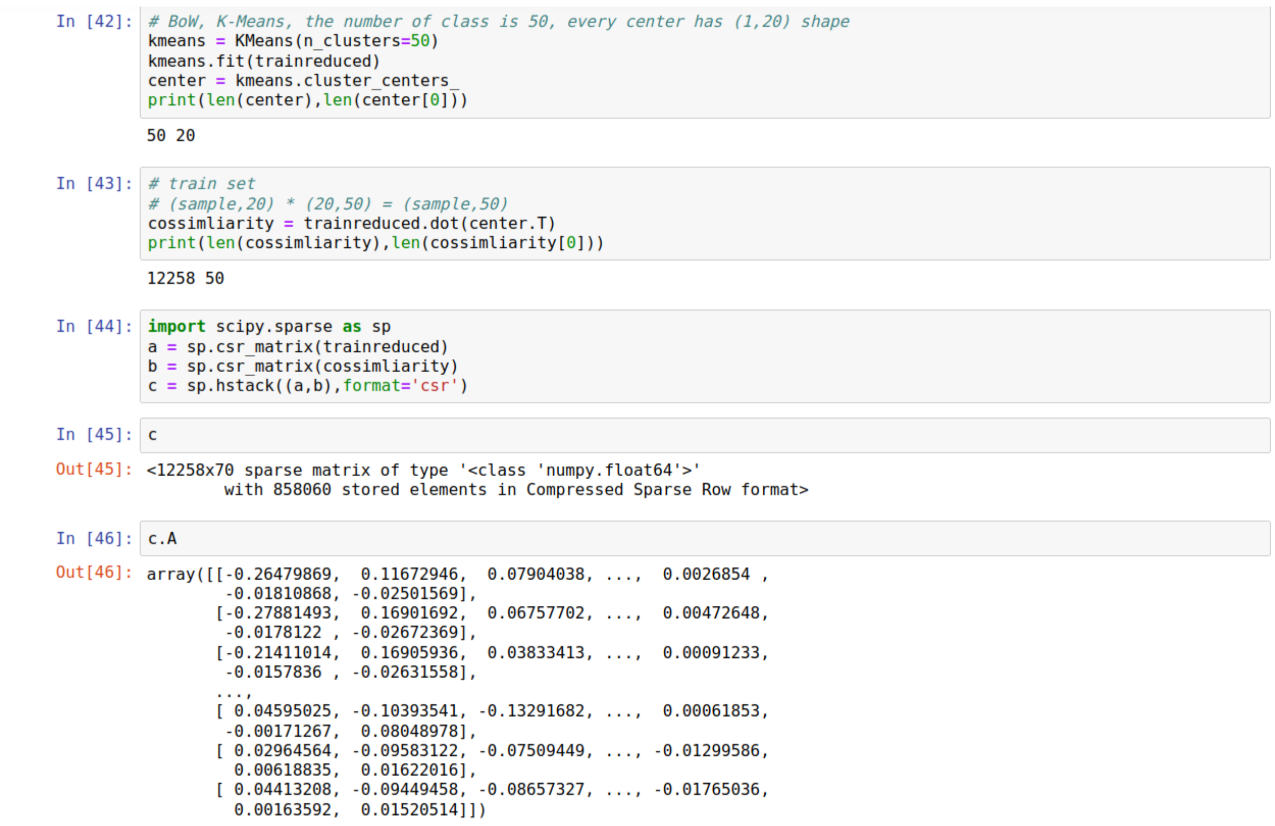
Figure 12: 训练集

- 在上一部分实现中，我们返回的是 1024 维的样本以及标签（0 代表背景，1 代表前景）。接下来是进行 PCA 降维和构建 `Visual Bag of Words Dictionary` 计算点积相似性，这一部分主要通过调取 `sklearn` 库完成，这里列出了代码、中间结果：

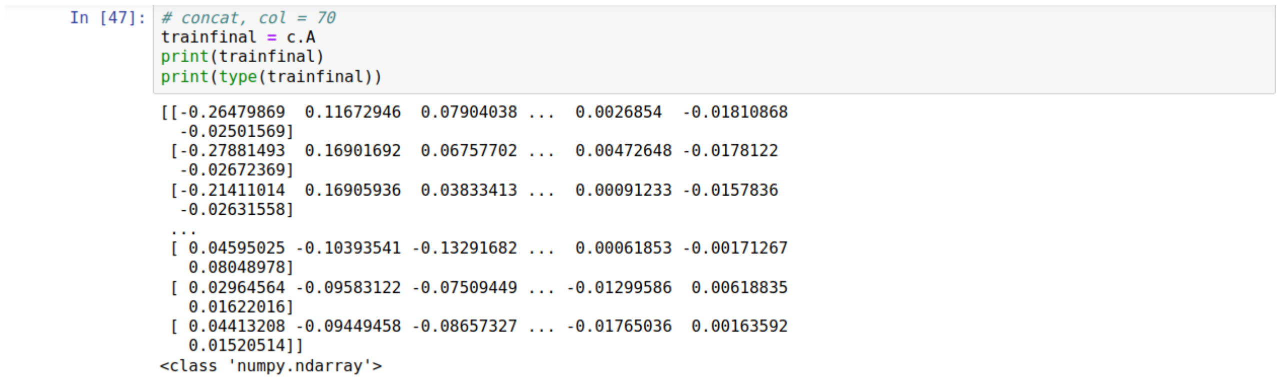


```
In [40]: # get train set and data set
         pca = PCA(n_components=20)
         test, testY = rgbfortest()
         train, trainY = rgbfortrain()

         # print(len(train), len(train[0]))
         # print(len(test), len(test[0]))
         print(len(trainY))
         print(len(testY))

         636
         636 1024
         [array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875  , 0.0214375 , 0.0015    , ..., 0.        , 0.05762712,
                0.        ]), array([0.069875 , 0.0214375, 0.0015   , ..., 0.        , 0.        ,
                0.        ]), array([0.069875  , 0.0214375 , 0.0015    , ..., 0.        , 0.44408946,

In [61]: # PCA, column = 20, row = sample
         pca.fit(train)
         trainreduced = pca.transform(train)
         testreduced = pca.transform(test)
         print(len(trainreduced), len(trainreduced[0]))
         print(len(testreduced), len(testreduced[0]))

         12258 20
         636 20
```

Figure 13: 降维

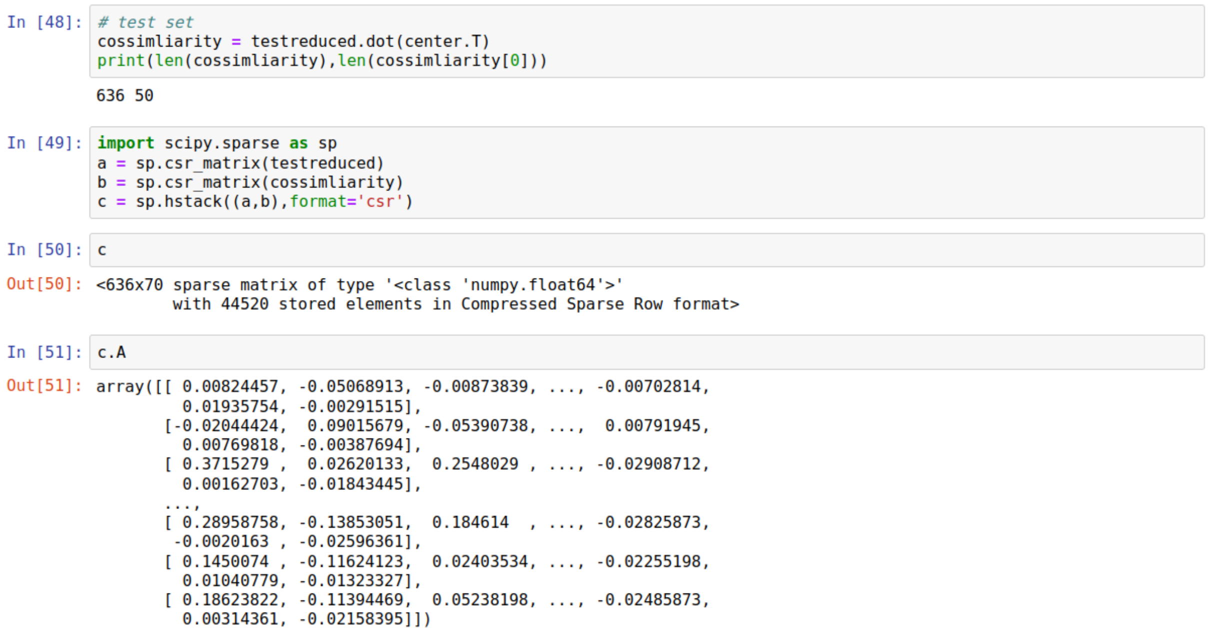


```
In [42]: # BoW, K-Means, the number of class is 50, every center has (1,20) shape
         kmeans = KMeans(n_clusters=50)
         kmeans.fit(trainreduced)
         center = kmeans.cluster_centers_
         print(len(center),len(center[0]))

         50 20

In [43]: # train set
         # (sample,20) * (20,50) = (sample,50)
         cossimliarity = trainreduced.dot(center.T)
         print(len(cossimliarity),len(cossimliarity[0]))

         12258 50

In [44]: import scipy.sparse as sp
         a = sp.csr_matrix(trainreduced)
         b = sp.csr_matrix(cossimliarity)
         c = sp.hstack((a,b),format='csr')

In [45]: c

Out[45]: <12258x70 sparse matrix of type '<class 'numpy.float64'>'
                 with 858060 stored elements in Compressed Sparse Row format>

In [46]: c.A

Out[46]: array([[-0.26479869,  0.11672946,  0.07904038, ...,  0.0026854 ,
                 -0.01810868, -0.02501569],
                [-0.27881493,  0.16901692,  0.06757702, ...,  0.00472648,
                 -0.0178122 , -0.02672369],
                [-0.21411014,  0.16905936,  0.03833413, ...,  0.00091233,
                 -0.0157836 , -0.02631558],
                ...,
                [ 0.04595025, -0.10393541, -0.13291682, ...,  0.00061853,
                 -0.00171267,  0.08048978],
                [ 0.02964564, -0.09583122, -0.07509449, ..., -0.01299586,
                  0.00618835,  0.01622016],
                [ 0.04413208, -0.09449458, -0.08657327, ..., -0.01765036,
                  0.00163592,  0.01520514]])
```

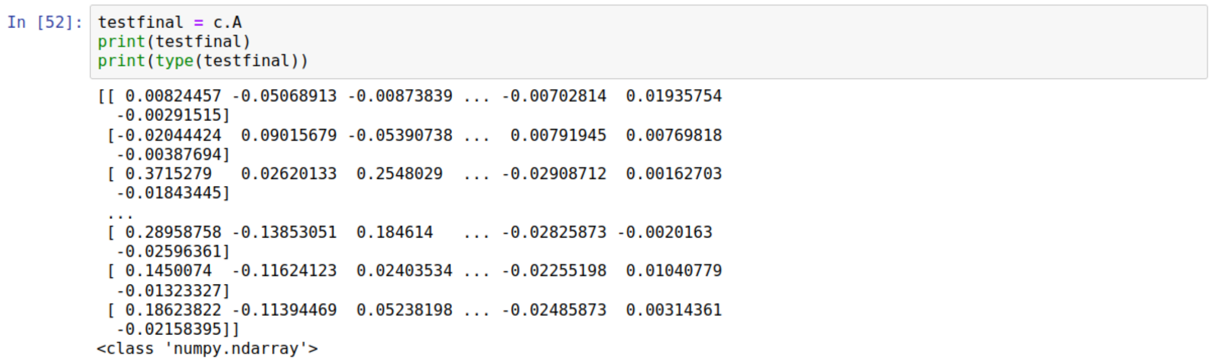


```
In [47]: # concat, col = 70
         trainfinal = c.A
         print(trainfinal)
         print(type(trainfinal))

         [[-0.26479869  0.11672946  0.07904038 ...  0.0026854  -0.01810868
           -0.02501569]
          [-0.27881493  0.16901692  0.06757702 ...  0.00472648 -0.0178122
           -0.02672369]
          [-0.21411014  0.16905936  0.03833413 ...  0.00091233 -0.0157836
           -0.02631558]
          ...
          [ 0.04595025 -0.10393541 -0.13291682 ...  0.00061853 -0.00171267
            0.08048978]
          [ 0.02964564 -0.09583122 -0.07509449 ... -0.01299586  0.00618835
            0.01622016]
          [ 0.04413208 -0.09449458 -0.08657327 ... -0.01765036  0.00163592
            0.01520514]]
         <class 'numpy.ndarray'>
```

Figure 14: 训练集最终 70 维数据



```
In [48]: # test set
         cossimliarity = testreduced.dot(center.T)
         print(len(cossimliarity),len(cossimliarity[0]))

         636 50

In [49]: import scipy.sparse as sp
         a = sp.csr_matrix(testreduced)
         b = sp.csr_matrix(cossimliarity)
         c = sp.hstack((a,b),format='csr')

In [50]: c

Out[50]: <636x70 sparse matrix of type '<class 'numpy.float64'>'
                 with 44520 stored elements in Compressed Sparse Row format>

In [51]: c.A

Out[51]: array([[ 0.00824457, -0.05068913, -0.00873839, ..., -0.00702814,
                  0.01935754, -0.00291515],
                [-0.02044424,  0.09015679, -0.05390738, ...,  0.00791945,
                  0.00769818, -0.00387694],
                [ 0.3715279 ,  0.02620133,  0.2548029 , ..., -0.02908712,
                  0.00162703, -0.01843445],
                ...,
                [ 0.28958758, -0.13853051,  0.184614  , ..., -0.02825873,
                 -0.0020163 , -0.02596361],
                [ 0.1450074 , -0.11624123,  0.02403534, ..., -0.02255198,
                  0.01040779, -0.01323327],
                [ 0.18623822, -0.11394469,  0.05238198, ..., -0.02485873,
                  0.00314361, -0.02158395]])
```

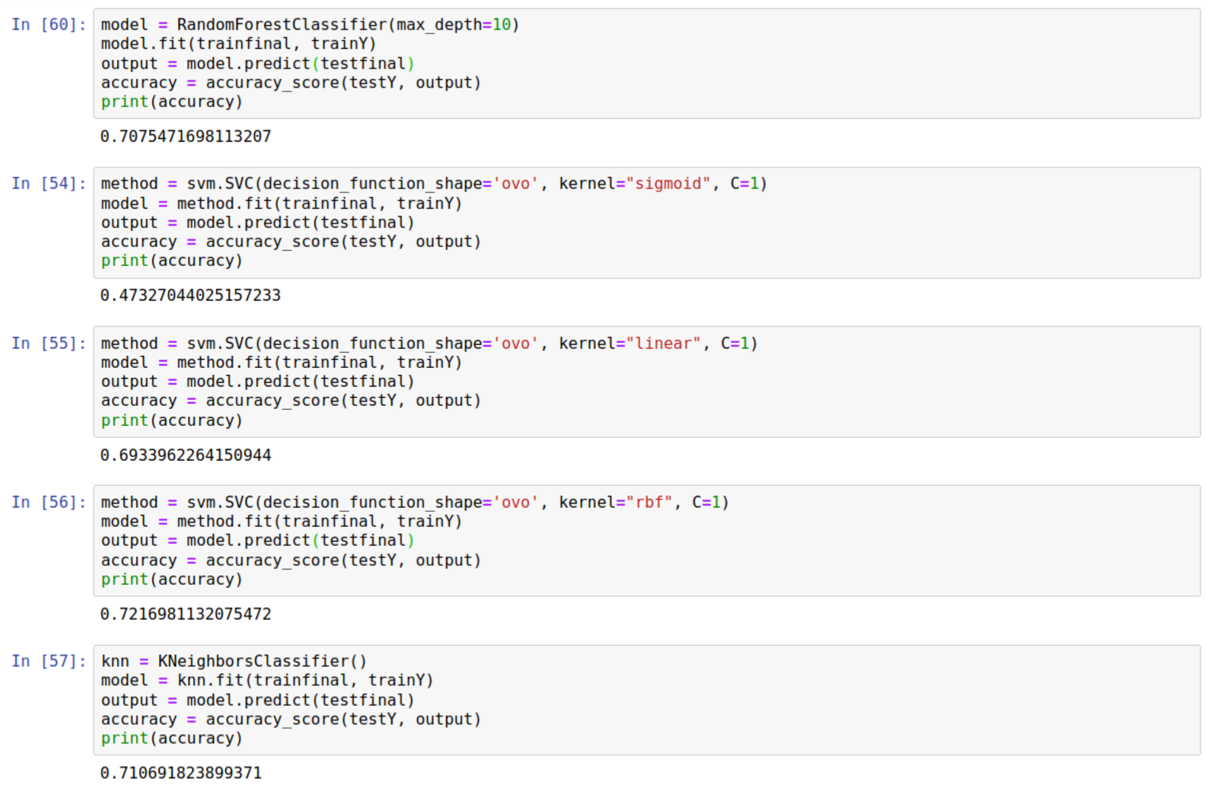


```
In [52]: testfinal = c.A
         print(testfinal)
         print(type(testfinal))

         [[ 0.00824457 -0.05068913 -0.00873839 ... -0.00702814  0.01935754
           -0.00291515]
          [-0.02044424  0.09015679 -0.05390738 ...  0.00791945  0.00769818
           -0.00387694]
          [ 0.3715279   0.02620133  0.2548029  ... -0.02908712  0.00162703
           -0.01843445]
          ...
          [ 0.28958758 -0.13853051  0.184614   ... -0.02825873 -0.0020163
           -0.02596361]
          [ 0.1450074  -0.11624123  0.02403534 ... -0.02255198  0.01040779
           -0.01323327]
          [ 0.18623822 -0.11394469  0.05238198 ... -0.02485873  0.00314361
           -0.02158395]]
         <class 'numpy.ndarray'>
```

Figure 15: 测试集最终 70 维数据

- 至此，训练集和测试集的数据都已拿到，接下来就是对测试集进行预测并获得准确率，我分别

测试了不同 kernel 下的 SVM、随机森林和 KNN 这三种分类算法，准确率如下：



```
In [60]: model = RandomForestClassifier(max_depth=10)
         model.fit(trainfinal, trainY)
         output = model.predict(testfinal)
         accuracy = accuracy_score(testY, output)
         print(accuracy)

         0.7075471698113207

In [54]: method = svm.SVC(decision_function_shape='ovo', kernel="sigmoid", C=1)
         model = method.fit(trainfinal, trainY)
         output = model.predict(testfinal)
         accuracy = accuracy_score(testY, output)
         print(accuracy)

         0.47327044025157233

In [55]: method = svm.SVC(decision_function_shape='ovo', kernel="linear", C=1)
         model = method.fit(trainfinal, trainY)
         output = model.predict(testfinal)
         accuracy = accuracy_score(testY, output)
         print(accuracy)

         0.6933962264150944

In [56]: method = svm.SVC(decision_function_shape='ovo', kernel="rbf", C=1)
         model = method.fit(trainfinal, trainY)
         output = model.predict(testfinal)
         accuracy = accuracy_score(testY, output)
         print(accuracy)

         0.7216981132075472

In [57]: knn = KNeighborsClassifier()
         model = knn.fit(trainfinal, trainY)
         output = model.predict(testfinal)
         accuracy = accuracy_score(testY, output)
         print(accuracy)

         0.710691823899371
```

Figure 16: 准确率

4.3 结果分析

可以看到，使用 sigmoid 作为核函数的 SVM 方法分类得到的准确率明显较低，除此之外其他几个分类器效果均较好，准确率较接近。