《计算机视觉》期末大作业报告

18342066 鲁沛

实验环境

Ananconada 4.5.11
Python 3.7
opencv-python 3.4.2.16
opencv-contrib-python 3.4.2.16

任务一

问题描述

结合第七次课内容及参考文献[1]，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 50~100 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 20 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 = $R1 \cap R2$ / $R1 \cup R2$.

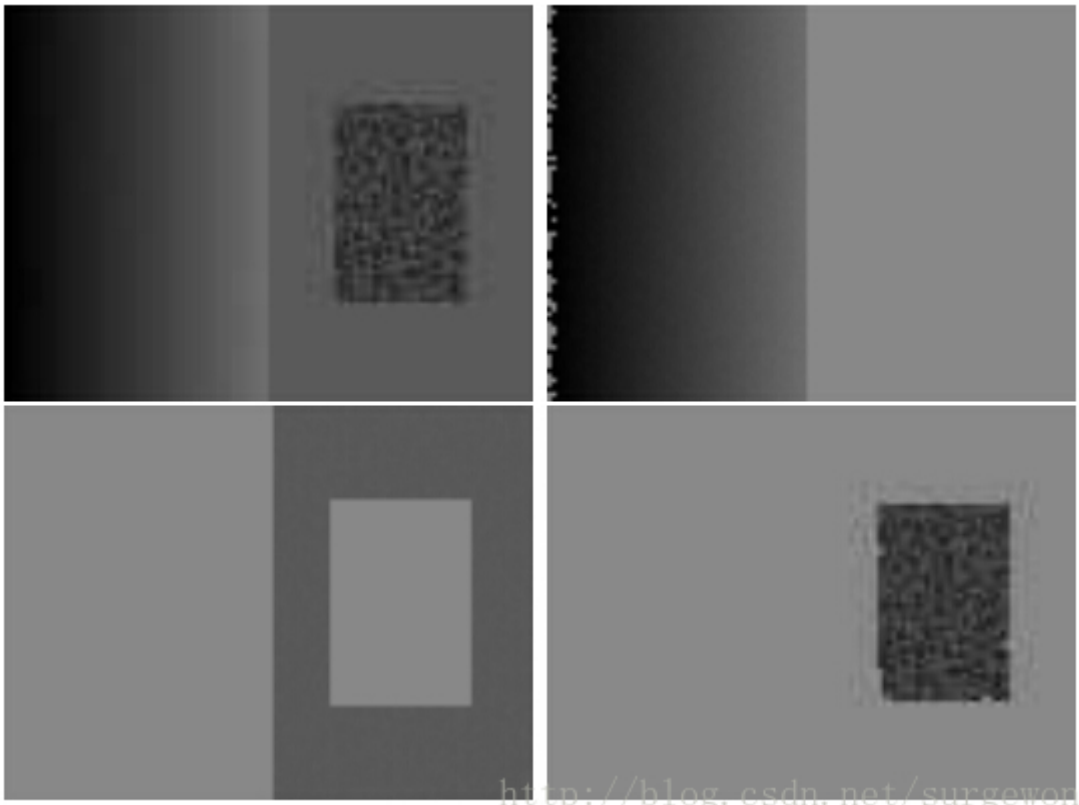
算法简介

2004年 Felzenszwalb 在 IJCV 上发表了一篇文章，主要介绍了一种基于图表示（graph-based）的图像分割方法。图像分割的主要目的也就是将图像分割成若干个特定的、具有独特性质的区域，然后从中提取出感兴趣的目标。而图像区域之间的边界定义是图像分割算法的关键，论文给出了一种在图表示（graph-based）下图像区域之间边界的定义的判断标准，其分割算法就是利用这个判断标准使用贪心选择来产生分割。该算法在时间效率上，基本上与图像的图表示的边数量成线性关系，而图像的图表示的边与像素点成正比，也就说图像分割的时间效率与图像的像素点个数成线性关系。这个算法有一个非常重要的特性，它能保持低变化（low-variability）区域的细节，同时能够忽略高变化（high-variability）区域的细节。这个性质很特别也很重要，对图像有一个很好的分割效果（能够找出视觉上一致的区域，简单讲就是高变化区域有一个很好聚合，能够把它们分在同一个区域）。

基本概念

1. 图像区域

做图像分割，首先需要理解如何定义一个区域，这对我们人眼来说并不难，但是使用数字化语言定义就没有那么容易了。比较直观的区域划分方法有，区域的颜色，边缘，纹理。我们来看一个例子：



人眼很容易判断出有三个区域：左半部分是灰度渐进变化的，右半部分外层灰度均匀变化，内层灰度变化较大。我们当然希望算法也能像人眼一样区分，上图的例子告诉我们不能使用灰度的变化作为分割依据，也不能使用单一的灰度阈值来作为分割的评判标准。能够捕捉视觉上重要的区域（perceptually important regions）对于一个分割算法来说是非常有意义的。

2. 图的概念

图：图由顶点集（vertices）和边集（edges）组成，连接一对顶点的边具有权重，本次实验中的权重是指顶点之间的不相似度，所用的是无向图。

树：特殊的图，图中任意两个顶点，都有路径相连接，但是没有回路。如果看成一团乱连的珠子，只保留树中的珠子和连线，那么随便选个珠子，都能把这棵树中所有的珠子都提起来。

最小生成树：特殊的树，给定需要连接的顶点，选择边权之和最小的树。

3. 图像的图表示

图像的图表示是指将图像表达成图论中的图。具体说来就是，把图像中的每一个像素点看成一个顶点vi ∈ V，像素点之间的关系对（可以自己定义其具体关系，一般来说是指相邻关系）构成图的一条边ei ∈ E，这样就构建好了一个图 G = (V,E)。图每条边的权值是基于像素点之间的关系，可以是像素点之间的灰度值差，也可以是像素点之间的距离。

4. 类内差异和类间差异

类内差异：Int(c)，可以近似理解为一个区域内部最大的亮度差异值，定义是MST中不相似度最大的一条边

类间差异：Diff(c1, c2)，即连接两个区域所有边中，不相似度最小的边的不相似度，也就是两个区域最相似的地方的不相似度。

所以两个区域是否可以合并就可以直观表示为：

$Diff(c1, c2) <= \min(Int(c1) + \tau(c1), Int(c2) + \tau(c2))$

5. 阈值函数τ

主要是为了更好的控制分割区域边界的定义。比较直观的理解，小分割区域的边界定义要强于大分割区域，否则可以将小分割区域继续合并形成大区域。本实验给出的阈值函数与区域的大小有关：

τ(c) = k / |c|

|C|是指分割部分顶点的个数（或者像素点个数），k是一个参数，可以根据不同的需求（主要根据图像的尺寸）进行调节，本实验中k值取200。

代码流程

总体流程：

1. 首先读入图片，进行高斯模糊，然后将其表达成图论中的图。即把图像中的每一个像素点看成一个顶点 vi ∈ V，每个像素与相邻 8 个像素构成图的一条边ei ∈ E，这样就构建好了一个图 G = (V,E)。图每条边的权值是像素与相邻像素的关系，表达了相邻像素之间的相似度。

2. 将每个像素点看成单一的区域，然后进行合并。
   (1) 对所有边根据权值从小到大排序，权值越小，两像素的相似度越大。
   (2) S[0]是一个原始分割，每个像素点都是一个分割区域。
   (3) 从小到大遍历所有边，如果这条边(vi,vj)的两个顶点属于不同的分割区域，并且权值不大于两个区域的内部差，那么合并这两个区域。更新合并区域的参数和内部差。

3. 最后对所有区域中，像素数都小于min_size的两个相邻区域，进行合并得到最后的分割。

4. 将分割后的图片与 data/gt 中对应的图片进行对比得到前景标注图，并计算IOU。

定义顶点类

每个顶点有三个性质:

parent：该顶点对应区域的**根顶点**；

priority：优先级，用于合并区域是确定根顶点；

size：每个顶点作为根顶点时，所在分割区域的顶点数量。

```
class Node:
    def __init__(self, parent, priority = 0, size = 1):
        self.parent = parent
        self.priority = priority
        self.size = size
```

定义森林类

self.num表示该图像当前分割区域的数量，self.nodes初始化森林类的所有顶点列表，self.labels定义每个顶点的索引：

```
def __init__(self, num_nodes):
    self.num = num_nodes
    self.nodes = [Node(i) for i in range(num_nodes)]
    self.labels = [i for i in range(num_nodes)]
```

size函数返回某个顶点所在分割区域的顶点数量：

```
def size(self, i):
    return self.nodes[i].size
```

利用并查集的思想获得该顶点所在区域的母顶点编号(索引):

```
def find(self, n):
    temp = n
    while temp != self.nodes[temp].parent:
        temp = self.nodes[temp].parent


    self.nodes[n].parent = temp
    return temp
```

根据顶点的优先级来合并两个顶点:

```
def merge(self, a, b):
    # 根据两个节点的优先级决定谁来当母顶点
    if self.nodes[a].priority > self.nodes[b].priority:
        self.nodes[b].parent = a
        self.nodes[a].size = self.nodes[a].size + self.nodes[b].size
    else:
        self.nodes[a].parent = b
        self.nodes[b].size = self.nodes[b].size + self.nodes[a].size

        if self.nodes[a].priority == self.nodes[b].priority:
            self.nodes[b].priority = self.nodes[b].priority + 1

    # 合并后分割的区域数量减1
    self.num = self.num - 1
```

计算两个顶点的差异性

```
def diff(img, x1, y1, x2, y2):
    res = np.sum((img[x1, y1] - img[x2, y2]) ** 2)
    return np.sqrt(res)
```

计算区域的类内差异

```
def threshold(size, const):
    return (const * 1.0 / size)
```

创建边

计算两个顶点的位置，调用diff函数计算边的权重：

```
def create_edge(img, width, x1, y1, x2, y2):
    vertex1 = x1 * width + y1
    vertex2 = x2 * width + y2
    #print(vertex1, vertex2)
    weight = diff(img, x1, y1, x2, y2)
    return (vertex1, vertex2, weight)
```

创建图

对每个顶点，创建←↑↖↗四条边，达到8-邻域的效果，来完成图的构建:

```
def build_graph(img):
    height = img.shape[0]
    width = img.shape[1]
    graph_edges = []

    for x in range(height):
        for y in range(width):
            if x > 0:
                graph_edges.append(create_edge(img, width, x, y, x-1, y))

            if y > 0:
                graph_edges.append(create_edge(img, width, x, y, x, y-1))

            if x > 0 and y > 0:
                graph_edges.append(create_edge(img, width, x, y, x-1, y-1))

            if x > 0 and y < height-1:
                graph_edges.append(create_edge(img, width, x, y, x-1, y+1))

    return graph_edges
```

合并小区域

对于初次分割后的图像，对于其中定点数均于min_size的两个相邻区域，进行合并：

```
def remove_small_components(forest, graph, min_size):
    for edge in graph:
        a = forest.find(edge[0])
        b = forest.find(edge[1])

        if a != b and (forest.size(a) < min_size or forest.size(b) < min_size):
            forest.merge(a, b)

    return  forest
```

分割图

(1) 首先初始化forest

(2) 对所有边，根据其权值从小到大排序

(3) 初始化区域内部差列表

(4) 从小到大遍历所有边，如果顶点在两个区域，且权值小于两个顶点所在区域的内部差(threshold[])，则合并这两个区域

(5) 调用remove_small_components函数合并过小的区域

```
def segment_graph(graph_edges, num_nodes, const, min_size):
    # 初始化
    forest = Forest(num_nodes)
    weight = lambda edge: edge[2]
    sorted_graph = sorted(graph_edges, key = weight)
    thresholds = [ threshold(1, const) for _ in range(num_nodes) ]

    # 合并
    for edge in sorted_graph:
        parent_a = forest.find(edge[0])
        parent_b = forest.find(edge[1])
        a_condition = weight(edge) <= thresholds[parent_a]
        b_condition = weight(edge) <= thresholds[parent_b]

        if parent_a != parent_b and a_condition and b_condition:
            forest.merge(parent_a, parent_b)
            a = forest.find(parent_a)
            thresholds[a] = weight(edge) + threshold(forest.nodes[a].size, const)

    return remove_small_components(forest, sorted_graph, min_size)
```

填充分割后的图

随机的对不同区域填充不同的颜色，得到分割后的彩色图：

```
def generate_image(forest, height, width):
    img_matrix = np.zeros((height, width), dtype=np.uint8)
    img = cv2.cvtColor(img_matrix, cv2.COLOR_GRAY2BGR)
    random_color = lambda: (int(random()*255), int(random()*255),
int(random()*255))
    colors = [random_color() for i in range(height * width)]

    for i in range(height):
        for j in range(width):
            comp = forest.find(i * width + j)
            img[i,j] = colors[comp]

    return img
```

得到前景图

遍历forest的每一个顶点，找到其对应的区域中的所有顶点，如果对应的gt图中该像素点灰度值是255，则mask

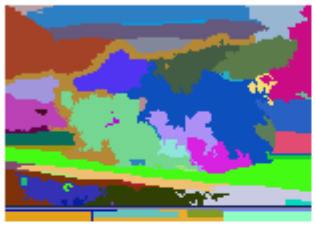
数量+1，最后如果mask的数量超过了该区域的顶点数量的一半，则这个区域所有顶点灰度值置为255，最后得到前景图并同时计算IOU。

```
def gt(forest, id):
    gt_img = cv2.imread('../data/gt/' + str(id) + '.png', cv2.IMREAD_GRAYSCALE)
    height = gt_img.shape[0]
    width = gt_img.shape[1]
    new_img = np.zeros(gt_img.shape, dtype = np.uint8)

    gt_label = {}
    for label in forest.labels:
        size = 0
        mask = 0

        for i in range(height):
            for j in range(width):
                root = forest.find(i * width + j)
                if root == label:
                    size += 1
                    if gt_img[i][j] == 255:
                        mask += 1

        # 区域 50%以上的像素在GT中标为255，则将区域认定为前景
        if mask * 2 > size:
            gt_label[label] = 1
            for i in range(height):
                for j in range(width):
                    root = forest.find(i * width + j)
                    if root == label:
                        new_img[i][j] = 255
        else:
            gt_label[label] = 0

    # 计算IOU
    intersection = 0
    union = 0
    for i in range(height):
        for j in range(width):
            if gt_img[i][j] == 255 or new_img[i][j] == 255:
                union += 1
            if gt_img[i][j] == 255 and new_img[i][j] == 255:
                intersection += 1
    res = intersection * 1.0 / union
    print("IOU为: ", res)

    return new_img
```

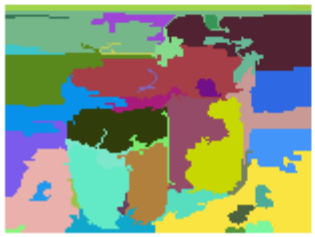
实验结果

将填充颜色后的分割图和对应的前景图打印到同一张图中，顺便输出IOU的值：

```
当前处理图片 66.png
IOU为： 0.7689185271093071
```




```
当前处理图片 166.png
IOU为： 0.808363784003248
```




全部10张图片的IOU结果：

图片编号	IOU
66.png	0.7598414047012177
166.png	0.8618239242005526
266.png	0.741643059490085
366.png	0.5558127114117857
466.png	0.5480163144234335
566.png	0.12694063926940638
666.png	0.5133228840125392
766.png	0.6120218579234973
866.png	0.8079911209766926
966.png	0.554296506137866

输出结果保存在1/result，可自行查看。

结果分析

可以看到566.png和966.png的结果比较差，可以对这两张图单独设置高斯模糊的参数以及分割时的K值，应该会得到更好的效果，时间有限，在这里并没有尝试。

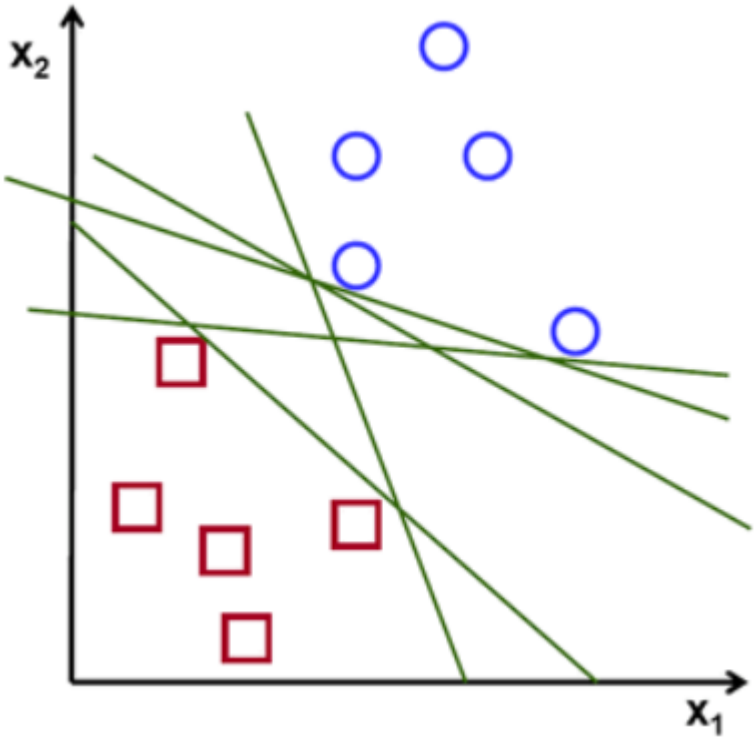
任务二

问题描述

对训练集中的每一张图提取归一化 RGB 颜色直方图特征（8 * 8 * 8 = 512维），同时结合问题 1，对训练集中的每张图进行分割（分割为 50~100 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，因此区域特征的维度为 2*512=1024 维，采用 PCA 算法对特征进行降维取前 50 维。训练集中的每张图被分割为 50~100 个区域，每个区域可以提取 50 维特征，且根据问题 1，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进行测试（将图像分割为 50~100 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。

SVM介绍

支持向量机(SVM)是一个类分类器，正式的定义是一个能够将不同类样本在样本空间分隔的超平面。 换句话说，给定一些标记(label)好的训练样本(监督式学习), SVM算法输出一个最优化的分隔超平面。
考虑如下问题，假设给定一些分属于两类的2维点，这些点可以通过直线分割， 我们要找到一条最优的分割线：



在上面的图中，你可以直觉的观察到有多种可能的直线可以将样本分开。那是不是某条直线比其他的更加合适呢？我们可以凭直觉来定义一条评价直线好坏的标准：距离样本太近的直线不是最优的，因为这样的直线对噪声敏感度高，泛化性较差。 因此我们的目标是找到一条直线，离所有点的距离最远。由此，SVM算法的实质是找出一个能够将某个值最大化的超平面，这个值就是超平面离所有训练样本的最小距离。这个最小距离用SVM术语来说叫做间隔(margin)。概括一下，最优分割超平面最大化训练数据的间隔。

求解过程

计算区域颜色直方图

输入参数为图，图对应的 forest 和区域的根顶点。遍历所有顶点，将位于根顶点对应区域的顶点标记为1，然后只对有标记的顶点计算颜色直方图。

```
def local(img, forest, c):
    height = img.shape[0]
    width = img.shape[1]
    mask = np.zeros((height, width), dtype="uint8")

    for i in range(height):
        for j in range(width):
            if forest.find(i * width + j) == c:
                mask[i][j] = 1

    # 只处理mask为1的点，即同一区域的点
    hist = cv2.calcHist([img], [0,1,2], mask, [8,8,8], [0,256,0,256,0,256])
    return hist.ravel()
```

用训练集得到分类模型

(1) 新建两个空列表用于存放区域的mask的和对应的特征值；
(2) 通过for循环遍历1-20号图作为训练集，对读进来的图进行高斯滤波理；
(3) 调用在任务一中封装好的graph.py中的函数进行基于图的图像分割；
(4) 计算全图颜色直方图；
(5) 对分割出来的每一个区域求区域颜色直方图，将区域颜色直方图和全图颜色直方图进行拼接，得到区域的特征向量；
(6) 对特征向量采用PCA降维降到50维；
(7) 调用sklearn库的SVM函数对区域的mask和对应特征值进行训练得到分类模型。

```
def train():
    features = []
    roots = []

    # 用前10张图作为训练集
    for i in range(1, 21):
        print("当前训练图片", str(i) + '.png')
        pic_path = '../data/imgs/' + str(i) + '.png'
        image = cv2.imread(pic_path, cv2.IMREAD_COLOR)

        # 高斯滤波
        source = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = cv2.GaussianBlur(source, (3, 3), sigmaX = 1.0, sigmaY = 1.0)

        height = image.shape[0]
        width = image.shape[1]

        graph_edges = graph.build_graph(image)
        forest = graph.segment_graph(graph_edges, height * width, 200, 20)
        gt_root = graph.gt(forest, i)

        # 计算全图颜色直方图
        hist = cv2.calcHist([image], [0,1,2], None, [8,8,8], [0,256,0,256,0,256])
        g_feature = hist.ravel().tolist()
```

```
        for c in forest.roots:
            l_feature = local(image, forest, c)
            # 拼接区域颜色直方图和全图颜色直方图
            feature = np.concatenate((g_feature, l_feature), axis=0)
            roots.append(gt_root[c])
            features.append(feature)

    X = np.array(features)
    Y = np.array(roots)

    # PCA降维至50维
    pca = PCA(n_components=50)
    new_X = pca.fit_transform(X)

    clf = svm.SVC(gamma = 0.001, C = 100.)
    print("训练开始")
    model = clf.fit(new_X, Y)
    print("训练结束")
    return model
```

计算预测准确率

输入参数为训练好的模型、X 和 Y。对 X 中的每一个分量，如果其预测值等于 Y 的对应分量，则count+1，最终用count和size的比值来衡量准确率。

```
def accuracy(model, X, Y):
    size, _ = X.shape
    result = model.predict(X)
    count = 0

    for i in range(size):
        if result[i] == Y[i]:
            count += 1

    right = count / size
    print("预测准确率为:", right)
```

主函数

对每一张测试图执行和训练图差不多的步骤，最终得到 features 和 roots。然后调用accuracy函数计算准确率。

```
if __name__ == "__main__":
    model = train()
    features = []
    roots = []
    for i in range(1, 1001):
        if i % 100 == 66:
            print("当前测试图片", str(i) + '.png')
            pic_path = '../data/imgs/' + str(i) + '.png'
            image = cv2.imread(pic_path, cv2.IMREAD_COLOR)
```

```
            # 高斯滤波
            source = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            image = cv2.GaussianBlur(source, (3, 3), sigmaX = 1.0, sigmaY = 1.0)

            height = image.shape[0]
            width = image.shape[1]

            graph_edges = graph.build_graph(image)
            forest = graph.segment_graph(graph_edges, height * width, 200, 20)
            gt_root = graph.gt(forest, i)

            # 计算全图颜色直方图
            hist = cv2.calcHist([image], [0,1,2], None, [8,8,8],
[0,256,0,256,0,256])
            g_feature = hist.ravel().tolist()

            for c in forest.roots:
                l_feature = local(image, forest, c)
                # 拼接区域颜色直方图和全图颜色直方图
                feature = np.concatenate((g_feature, l_feature), axis=0)
                roots.append(gt_root[c])
                features.append(feature)

    X = np.array(features)
    Y = np.array(roots)

    # PCA降维至50维
    pca = PCA(n_components = 50)
    new_X = pca.fit_transform(X)
    accuracy(model, new_X, Y)
```

实验结果

采用1-20号图片作为训练集，后两位是66的图片作为测试集。

```
当前训练图片 1.png
当前训练图片 2.png
当前训练图片 3.png
当前训练图片 4.png
当前训练图片 5.png
当前训练图片 6.png
当前训练图片 7.png
当前训练图片 8.png
当前训练图片 9.png
当前训练图片 10.png
当前训练图片 11.png
当前训练图片 12.png
当前训练图片 13.png
当前训练图片 14.png
当前训练图片 15.png
当前训练图片 16.png
当前训练图片 17.png
当前训练图片 18.png
当前训练图片 19.png
当前训练图片 20.png
训练开始
训练结束
当前测试图片 66.png
当前测试图片 166.png
当前测试图片 266.png
当前测试图片 366.png
当前测试图片 466.png
当前测试图片 566.png
当前测试图片 666.png
当前测试图片 766.png
当前测试图片 866.png
当前测试图片 966.png
预测准确率为: 0.8823529411764706
```

结果分析

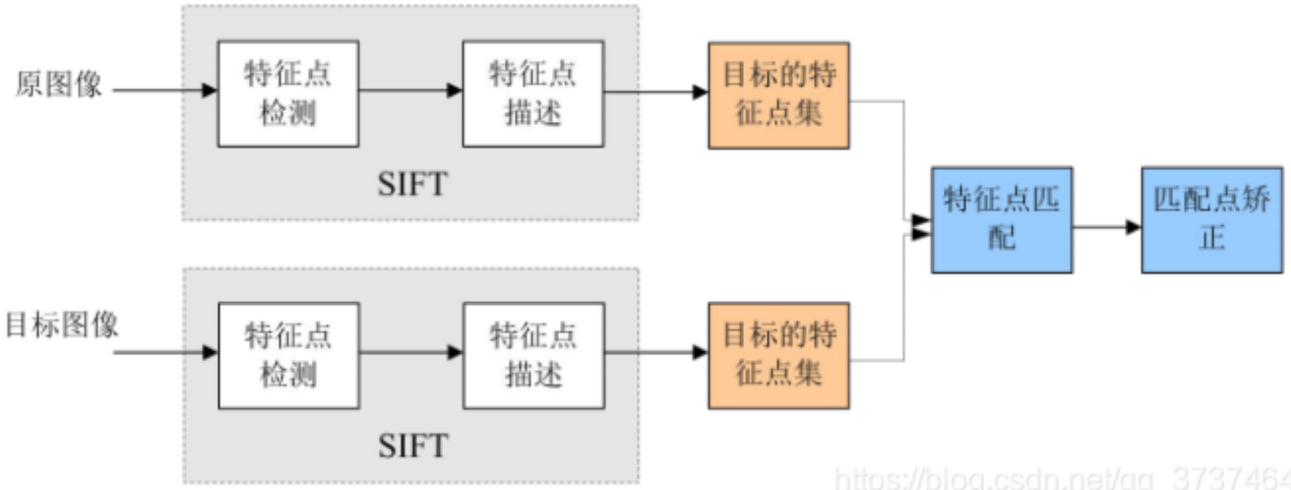
事实上，不论是用1张图，5张图还是10张图作为训练集，最终得到的预测准确率都一模一样。不禁让我感到迷惑，但是迫于时间有限，无法深入思考，留待日后解决。

任务三

问题描述

对每张测试图提取 Sift 特征点及 Sift 特征，采用 PCA 算法将特征降维至 10 维，以每个 sift 特征点为中心截取 16*16 的 patch（超出边界的 patch 可以舍去），计算归一化颜色直方图（4*4*4=64 维），将两个特征拼接（sift 特征和 颜色直方图特征），这样每个 sift 特征点表示为 74 维向量，采用 k-means 聚类算法将该图像所有 sift 特征点聚为 3 类，并依次将各聚簇中的 patch 组合形成一张展示图（例如总共有 N 个 sift 点，对应 N 个 patch，展示图中需要将同一聚簇的 patch 按顺序粘贴，不同聚簇用分割线分开）。要求每张测试图生成一张可视化聚类展示图。

SIFT介绍：
尺度不变特征转换即SIFT (Scale-invariant feature transform)是一种计算机视觉的算法。它用来侦测与描述影像中的局部性特征，它在空间尺度中寻找极值点，并提取出其位置、尺度、旋转不变量，此算法由 David Lowe在1999年所发表，2004年完善总结。其应用范围包含物体辨识、机器人地图感知与导航、影像缝合、3D模型建立、手势辨识、影像追踪和动作比对。



PCA介绍：
主成分分析（Principal Component Analysis，PCA）是一种多变量统计方法，它是最常用的降维方法之一，通过正交变换将一组可能存在相关性的变量数据转换为一组线性不相关的变量，转换后的变量被称为主成分。

K-means介绍：
kmeans算法又名k均值算法,K-means算法中的k表示的是聚类为k个簇，means代表取每一个聚类中数据值的均值作为该簇的中心，或者称为质心，即用每一个的类的质心对该簇进行描述。
其算法思想大致为：先从样本集中随机选取 k个样本作为簇中心，并计算所有样本与这 k个“簇中心”的距离，对于每一个样本，将其划分到与其距离最近的“簇中心”所在的簇中，对于新的簇计算各个簇的新的“簇中心”。

求解过程

此任务调用opencv中的库函数即可解决。
调用的库：

```
import cv2
import numpy as np
import pandas as pd
```

```
from matplotlib import pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from math import ceil
```

首先用for循环筛选出文件名最后两位是66（学号后两位）的图片，将其读入并转化为灰度图：

```
for i in range(1, 1001):
    if i % 100 == 66:
        pic_path = '../data/imgs/' + str(i) + '.png'
        img = cv2.imread(pic_path)
        img1 = img.copy()
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

计算特征点kp和特征向量des，并将特征点在图中画出来，其中flags参数是绘图功能的标识设置，cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS表明对每一个特征点绘制带大小和方向的特征点图形：

```
sift = cv2.xfeatures2d.SIFT_create()
kp, des = sift.detectAndCompute(gray, None)
cv2.drawKeypoints(gray, kp, img1,
flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

调用sklearn库的PCA算法相关函数将特征向量降维至 10 维得到新的特征向量newdes：

```
pca = PCA(n_components = 10)
newdes = pca.fit_transform(des)
```

以每个 sift 特征点为中心截取 16*16 的 patch，删除掉越界的patch及对应的特征值和特征向量。首先新建三个空的列表patchs、final_kp和final_des，用for循环遍历每一个特征点。特征点是一个类，其坐标在pt分量中，提取出特征点的坐标后，首先判断此特征点为中心的patch是否会越界，若不会，则取特征点上边7个单位和下边8个单位，左边7个单位和右边8个单位，再加上特征点本身，组成一个16*16的patch加入patchs列表中。然后将此符合要求的patch对应的特征点和特征向量加入final_kp和final_des中:

```
patchs = []
final_kp = []
final_des = []
for j in range(len(kp)):
    x = int(kp[j].pt[0])
    y = int(kp[j].pt[1])
    if x - 7 >= 0 and x + 8 < width and y - 7 >= 0
     and y + 8 < height:
        patchs.append(img[y - 7 : y + 9, x - 7 : x + 9])
        final_kp.append(kp[j])
```

```
            final_des.append(newdes[j])
            desMat = np.array(final_des)
```

计算归一化颜色直方图。首先新建一个空的列表colorHistlist，然后遍历patchs，对每个patch进行通道拆分，得到b、g、r，使用cv2.equalizeHist函数对三个通道分别进行直方图均衡化后在将通道合并，最后调用cv2.calcHist函数计算直方图:

```
b = cv2.equalizeHist(b)
g = cv2.equalizeHist(g)
r = cv2.equalizeHist(r)
testimg = cv2.merge([b, g, r])     # 通道合并
hist = cv2.calcHist([testimg], [0, 1 ,2], None, [4, 4, 4], [0, 256, 0, 256, 0,
256])
```

调用sklearn库的Kmeans函数将该图像所有 sift 特征点聚为 3 类，将每一个聚簇中的特征点按顺序加入到class0、class1、class2列表中：

```
clf = KMeans(n_clusters=3)
cls = clf.fit(siftMat)
labels = cls.labels_
class0 = []
class1 = []
class2 = []
for j in range(len(patchs)):
    if labels[j] == 0:
        class0.append(patchs[j])
    elif labels[j] == 1:
        class1.append(patchs[j])
    elif labels[j] == 2:
        class2.append(patchs[j])
```

最后将各聚簇中的 patch 组合形成一张展示图。我们选择每一行拼接10个 patch ,即列数column定义为10，然后用每一个聚簇的 patch 个数除以10再向上取整得到行数，选取最大的行数作为row。用plt.figure函数新建一个画布，size定义为row和column的一半，根据输入图片的不同，输出图片的长宽也是不同的。

```
row0 = ceil(len(class0) / 10)
row1 = ceil(len(class1) / 10)
row2 = ceil(len(class2) / 10)
row = max(row0, row1, row2)
plt.figure(num='image', figsize=(32/2,row/2))
```

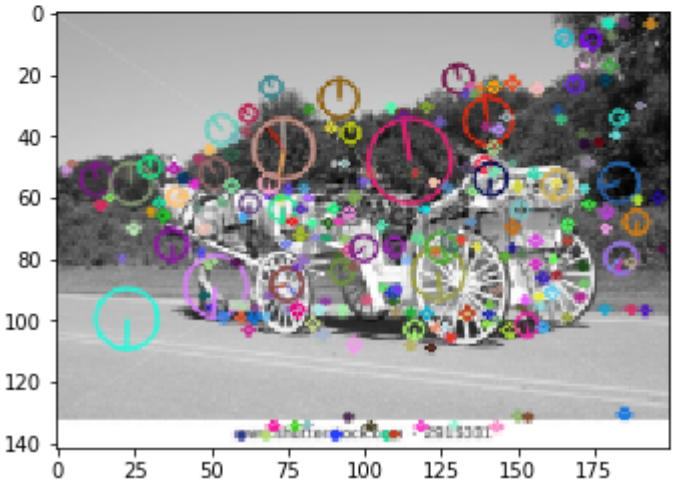
用plt.subplot函数设置row * 32个子图，最左边是第0类聚簇，然后空一列，中间是第1类聚簇，然后再空一列，最右边是第2类聚簇。分别将3类聚簇填充进画布中，保存结果图。

```
    k = 0
    for j in range(len(class0)):
        plt.subplot(row, 32, j + k + 1)
        plt.imshow(class0[j])
        plt.axis('off')
        plt.subplots_adjust(wspace=0,hspace=0)
        if (j + 1) % 10 == 0:
            k += 22

    k = 11
    for j in range(len(class1)):
        plt.subplot(row, 32, j + k + 1)
        plt.imshow(class1[j])
        plt.axis('off')
        plt.subplots_adjust(wspace=0,hspace=0)
        if (j + 1) % 10 == 0:
            k += 22

    k = 22
    for j in range(len(class2)):
        plt.subplot(row, 32, j + k + 1)
        plt.imshow(class2[j])
        plt.axis('off')
        plt.subplots_adjust(wspace=0,hspace=0)
        if (j + 1) % 10 == 0:
            k += 22

    plt.savefig('result/' + str(i) + '.png')
    plt.show()
```

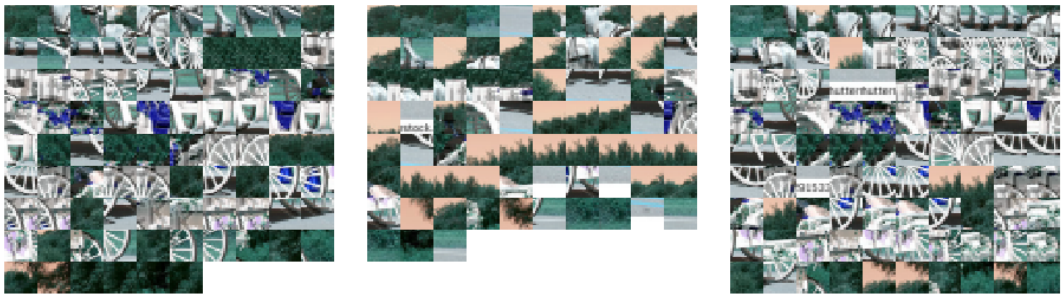
实验结果

将实验中的一些中间结果打印出来作为日志：

```
当前处理图片 66.png
原特征点的个数： 271
原特征向量的维数： (271, 128)
PCA降维后特征向量的维数： (271, 10)
patchs的数量： 249
删减后特征向量的维数： (249, 10)
patchs的归一化颜色直方图维数: (249, 64)
最终特征向量的维数： (249, 74)
第0簇的特征点个数 86
第1簇的特征点个数 73
第2簇的特征点个数 90
```



最终输出的结果图：



10张图的运行结果都放在了3/result文件夹中，可自行查看。

心得体会

这次作业不得不说难度真的很大，网络上并没有什么直接的资料可以参考，任务要求里的每一句话都要仔细思考并大量查阅资料才能知道怎么做。其中难度最大的是第一题，但是网上有源代码可以参考，慢慢做还是能做的。后两道题要相对简单一些，基本上都是在调用opencv的库函数来完成的。本门课程的期中作业是写一篇精读读书笔记，但是几乎没有涉及代码，真正的接触到图像处理的代码实现本次作业还是头一次。因此本次大作业虽然做的过程十分痛苦，但还是学到了很多实用的东西，对我的提高很大。