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

Computer Vision Final project

17341111 Xuehai Liu

2020 年 7 月 26 日

目录

1 问题描述

- (1) 实现基于 Graph-based image segmentation 方法，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。

- (2) 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。

- (3) 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.

2 求解过程与结果展示

2.1 Graph-based image segmentation

本小节实现基于图的图片分割算法。该算法在 2004 年的论文 Efficient graph-based image segmentation 中被首次提出并得到广泛的引用。

该算法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（Image Segmentation）的主要目的也就是将图像（image）分割成若干个特定的、具有独特性质的区域（region），然后从中提取出感兴趣的目标（object）。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（predicate），其分割算法就是利用这个判断标准（predicate）使用贪心选择（greedy decision）来产生分割（segmentation）。该算法在时间效率上，基本上与图像（Image）的图（Graph）表示的边（edge）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（region）的细节，同时能够忽略高变化（high-variability）区域（region）的细节。

2.1.1 Keypoints of Method

本项目主要有两个关键点：1. 图像（image）的图（graph）表示；2. 最小生成树（Minimun Spanning Tree）。

图像（image）的图表示是指将图像（image）表达成图论中的图（graph）。具体说来就是，把图像中的每一个像素点看成一个顶点 $vi \in V$（node 或 vertex），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 $ei \in E$，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。

将图像表达成图之后，接下来就是要如何分割这个图，或者说，将每个节点（像素点）看成单一的区域，然后进行合并. 论文中使用最小生成树方法合并像素点，然后构成一个个区域。其原理是将图（Graph）简化，相似的区域在一个分支（Branch）上面（有一条最边连接），大大减少了图的边数。

2.1.2 Algorithm

Algorithm 1 *Segmentation algorithm.*

The input is a graph $G = (V, E)$, with n vertices and m edges. The output is a segmentation of V into components $S = (C_1, \ldots, C_r)$.

0. Sort E into $\pi = (o_1, \ldots, o_m)$, by non-decreasing edge weight.
1. Start with a segmentation S^0, where each vertex v_i is in its own component.
2. Repeat step 3 for $q = 1, \ldots, m$.
3. Construct S^q given S^{q-1} as follows. Let v_i and v_j denote the vertices connected by the q-th edge in the ordering, i.e., $o_q = (v_i, v_j)$. If v_i and v_j are in disjoint components of S^{q-1} and $w(o_q)$ is small compared to the internal difference of both those components, then merge the two components otherwise do nothing. More formally, let C_i^{q-1} be the component of S^{q-1} containing v_i and C_j^{q-1} the component containing v_j. If $C_i^{q-1} \neq C_j^{q-1}$ and $w(o_q) \leq MInt(C_i^{q-1}, C_j^{q-1})$ then S^q is obtained from S^{q-1} by merging C_i^{q-1} and C_j^{q-1}. Otherwise $S^q = S^{q-1}$.
4. Return $S = S^m$.

图 1: Algorithm Definition

2.1.3 Class Definition

接下来，进入具体的项目实现部分。首先，定义类 Node 和 Forest，分别表示图像的一个像素，以及分割的图像块的集合。

定义顶点 (node) 类，把每个像素定义为节点 (顶点)。顶点有三个性质：

(1) parent，该顶点对应分割区域的母顶点，可以认为的分割区域的编号或者索引。后面初始化时，把图像每个像素当成一个分割区域，所以每个像素的母顶点就是他们本身。

(2) rank，母顶点的优先级（每个顶点初始化为 0），用来两个区域合并时，确定唯一的母顶点。

(3) size（每个顶点初始化为 1），表示每个顶点作为母顶点时，所在分割区域的顶点数量。当它被其他区域合并，不再是母顶点时，它的 size 不再改变。

(4) number，表示这个顶点对应的像素编号，在初始化后它不再改变。

```
class Node: #one node refers to one pixel in graph
    def __init__(self,parent, rank = 0, size = 1):
        self.parent = parent
        self.rank = rank
        self.size = size
        self.number = parent
    def __repr__(self):
        return '(parent=%s, rank=%s, size=%s, number = %s)'%(self.parent,self.rank,self.
                                                    size,self.number)
```

(1) self.nodes 初始化 forest 类的所有顶点列表，初始化时把图像每个像素作为顶点，当成一个分割区域，每个像素的母顶点就是他们本身。forest.num_sets 表示该图像当前的分割区域数量。(2) size_of()，获取某个顶点的 size，一般用来获得某个母顶点的 size，即为母顶点所在分割区域的顶点数量。(3) find()，获得该顶点所在区域的母顶点编号 (索引) (4) merge(self, a,b)，合并顶点 a 所在区域和顶点 b 所在区域，找到 a,b 的母顶点，根据其优先级 rank 确定新的母顶点，更新合并后新区域的顶点数量 size，新母顶点的优先级 rank，分割区域的数量 num_sets。(5)dict_parent2node()，获取一个字典，得到 parent - 区域内所有 node 编号的映射。

```
class Forest: #forest defines sets of all nodes and operation among nodes.
    def __init__(self,num_nodes):
            self.nodes = [Node(i) for i in range(num_nodes)]
            self.num_sets = num_nodes

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

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

    def merge(self, a , b):
            if self.nodes[a].rank >self.nodes[b].rank:
                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[a].size + self.nodes[b].size
                if(self.nodes[a].rank == self.nodes[b].rank):
                    self.nodes[b].rank = self.nodes[b].rank + 1
            self.num_sets = self.num_sets - 1
```

```
    def printnodes(self):
            for node in self.nodes:
                print (node)
    def dict_parent2node(self):
        dict = {}
        for node in self.nodes:
            dict[self.find(node.number)]= []
        for node in self.nodes:
            dict[self.find(node.number)].append(node.number)
        return dict
```

2.1.4 Graph Segmentation Functions

接下来根据上述类，来对图像进行数据表示并分割。下面定义了一些函数，他们的功能已在注释中指出。实验部分，我将指出合适的参数，从而能够将区域划分为 50-100 个区域。同时，通过指定 merge_small_components 中 min_size = 20，能够令像素数过少的区域被合并。

```
#diff function defines the difference between 2 nodes
def diff(img, x1, y1, x2, y2):
    _out = np.sum((img[x1, y1] - img[x2, y2]) ** 2)
    return np.sqrt(_out)

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

# an edge defines the id of two nodes and the weight of edge.
def create_edge(img,width, x,y, x1,y1,diff):
    vertex_id = lambda x,y: x*width + y
    w = diff(img,x,y,x1,y1) #weight of edge
    return (vertex_id(x,y),vertex_id(x1,y1),w)

#build graph with edges and nodes
def build_graph(img,width,height, diff, neighbour8 = False):
    graph = []
    for x in range(height):
        for y in range(width):
            if (x>=1):
                graph.append(create_edge(img,width,x,y,x-1,y,diff))
            if (y>=1):
                graph.append(create_edge(img,width,x,y,x,y-1,diff))
            if neighbour8:
                if (x>0 and y>0):
                    graph.append(create_edge(img,width,x,y,x-1,y-1,diff))
                if (x>0 and y< width -1):
                    graph.append(create_edge(img,width,x,y,x-1,y+1,diff))
```

```
    return graph

#merge 2 components if they are too small
def merge_small_components(forest, graph,min_size):
    for edge in graph:
        a = forest.find(edge[0])
        b = forest.find(edge[1])
        # merge 2 nodes if area_size < min_size
        if a!=b and (forest.size_of(a)<min_size or forest.size_of(b)<min_size):
            forest.merge(a,b)
    return forest
```

然后，进入核心算法部分. 算法的详细说明请见 2.1.2 节。

```
#segment graph and return forest
def segment_graph(graph,num_nodes,const, min_size,threshold_func):
    weight = lambda edge: edge[2]
    forest = Forest(num_nodes)
    sorted_graph = sorted(graph,key = weight)
    threshold = [threshold_func(1,const)] *num_nodes

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

        if parent_a != parent_b and a_condition and b_condition:
            forest.merge(parent_a,parent_b)
            a = forest.find(parent_a)
            #print(a)
            threshold[a] = weight(edge) + threshold_func(forest.nodes[a].size,const)
    return merge_small_components(forest,sorted_graph,min_size)
```

2.1.5 Test on graphs

接下来，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）。我们需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。

graphbased_segmentation 函数接收一个图片名称以及若干参数，生成该图像的黑白前背景图，并计算该生成的图片与 GT 的 IOU 分数。

```
def graphbased_segmentation(imagename, sigma = 0.5 ,K = 200, min_size = 20, neighbour =
                                                8):
    neighbour = 8
    gt_folder = "./data/gt/"
    img_folder = './data/imgs/'
```

```
    img_path = os.path.join(img_folder, imagename)
    img_file = Image.open(img_path)
    gt_path = os.path.join(gt_folder, imagename)
    gt_file = Image.open(gt_path)

    rgb_im = img_file.convert('RGB')
    rgb_gt = gt_file.convert('RGB')
    #print(forest.dict_parent2node())

    forest = get_forest(img_file,sigma,K,min_size,neighbour)

    size = img_file.size
    width = size[0]
    height = size[1]

    #generate graph with black or white color
    area_dict = forest.dict_parent2node()
    counter = {}
    colors = {}
    for key in area_dict.keys():
        counter[key] = 0
        colors[key] = (255,255,255)
    for parent in area_dict.keys():
        area = area_dict[parent]
        for number in area:
            i = number % width
            j = (number - i) / width
            gr, gg, gb = rgb_gt.getpixel((i, j))
            if(gr<=128 and gg<=128 and gb<=128):
                counter[parent] += 1
    for parent in area_dict.keys():
        if(counter[parent] >= len(area_dict[parent])/2):
            colors[parent] = (0,0,0)
        else:
            colors[parent] = (255,255,255)

    image = generate_image(forest, size[0],size[1],colors)

    IOU = calc_IOU(image,rgb_gt,width,height)
    return IOU
```

其中，IOU 计算方式如下：

```
def calc_IOU(image,gt,width,height):
    #calculate IOU
    gt_fore_pixel_num = 0
    fore_pixel_num = 0
    correct_pixel_num = 0
```

```
for i in range(width):
    for j in range(height):
        gr,gg,gb = gt.getpixel((i,j))
        r,g,b = image.getpixel((i,j))
        if(gr<= 128 and gg<=128 and gb<=128):
            gt_fore_pixel_num += 1
            if(r==0 and g ==0 and b == 0):
                correct_pixel_num += 1
        if(r == 0 and g ==0 and b ==0 ):
            fore_pixel_num += 1
IOU = correct_pixel_num / (gt_fore_pixel_num +fore_pixel_num - correct_pixel_num) *
                                                    1.00
print("IOU:",IOU)
return IOU
```

在测试集上运行，可以看到结果如下：

```
Image info: PNG (200, 129) RGB
number of components: 77
IOU: 0.8058364109705153
Image info: PNG (200, 138) RGB
number of components: 72
IOU: 0.9096641967523253
Image info: PNG (200, 134) RGB
number of components: 70
IOU: 0.928763440860215
Image info: PNG (200, 141) RGB
number of components: 89
IOU: 0.9187355726197767
Image info: PNG (200, 75) RGB
number of components: 36
IOU: 0.8418040067107471
Image info: PNG (150, 200) RGB
number of components: 83
IOU: 0.9786860112926747
Image info: PNG (175, 116) RGB
number of components: 60
IOU: 0.8845252051582649
Image info: PNG (200, 133) RGB
number of components: 84
IOU: 0.9518502698714713
```

图 2: 参考运行结果

取两位小数，在测试集上的 IOU 结果为：[0.81, 0.91, 0.93, 0.92, 0.84, 0.98, 0.88, 0.95, 0.9, 0.96]
一个参考的生成的前背景分割图如下。全部的前背景分割图像与原测试图请见邮件附件。

图 3: 11.png

2.2 RGB color histogram features and SVM classification

第二题的要点是，对于一张图片，

- 提取全局的归一化 RGB 颜色直方图特征
- 分割图像并提取各个区域的归一化 RGB 颜色直方图特征
- 对于每一个划分的区域，拼接两个直方图特征降维得到 50 维的特征，根据 GT 得到这个区域对应的 label，从而训练机器学习模型（本文采用 SVM）。
- 根据测试集 GT，验证机器学习算法的预测准确率。

2.2.1 Global RGB Histogram

下面，我调用 Python 的 opencv 库的 calchist 函数计算了全局的 RGB 直方图特征，代码如下：

```
def gain_dataset(imagename,sigma = 0.5 ,K = 200, min_size = 20):
    img_path = os.path.join(img_folder, imagename)
    original_img = cv2.imread(img_path)
    img = cv2.resize(original_img, None, fx=1, fy=1, interpolation=cv2.INTER_CUBIC)
    b, g, r = cv2.split(img)

    # get the global hist
    min_max_scaler = preprocessing.MinMaxScaler()
    histb = cv2.calcHist([b], [0], None, [8], [0.0, 255.0])
    normal_b = min_max_scaler.fit_transform(histb)

    histg = cv2.calcHist([g], [0], None, [8], [0.0, 255.0])
```

```
    normal_g = min_max_scaler.fit_transform(histg)

    histr = cv2.calcHist([r], [0], None, [8], [0.0, 255.0])
    normal_r = min_max_scaler.fit_transform(histr)

    global_feature = get_feature(normal_r, normal_g, normal_b)
```

2.2.2 Area RGB Histogram

此外，定义获取局部区域直方图特征的函数 calculate_area_feature 定义如下：

```
def calculate_area_feature(image,area_dict,width,Bins = 8):
    area_feature = {}
    areahist = {}
    #calculate area hists and get RGB features
    for parent in area_dict.keys():
        area = area_dict[parent]
        areahist[parent] = [[ [0.] for i in range(Bins)] for j in range(3)]
        area_feature [parent] = []
        for pixel in area:
            i = pixel % width
            j = (pixel - i) / width
            r,g,b = image.getpixel((i,j))
            areahist[parent][0][r // (256 // Bins)][0] += 1
            areahist[parent][1][g // (256 // Bins)][0] += 1
            areahist[parent][2][b // (256 // Bins)][0] += 1
        #print(areahist[parent])
        normal_r = min_max_scaler.fit_transform(np.array(areahist[parent][0]))
        normal_g = min_max_scaler.fit_transform(np.array(areahist[parent][1]))
        normal_b = min_max_scaler.fit_transform(np.array(areahist[parent][2]))
        areahist[parent][0] = normal_r
        areahist[parent][1] = normal_g
        areahist[parent][2] = normal_b

        area_feature[parent] = get_feature(normal_r,normal_g,normal_b)
        #arr = np.array(area_feature[parent])
    return area_feature
```

2.2.3 PCA and Label Obtained by GT

接下来，合并全局和区域的直方图特征并进行降维，并根据 GT 得到该区域的 label(0 表示背景，1 表示前景)。GT 中单个像素的判别方法为 r，g，b 通道的像素值均小于等于 128，则记为背景，否则为前景。

```
    #combind global and area feature
    combind_feature = {}
```

```
    for parent in area_feature:
        combind_feature[parent] = []
        combind_feature[parent].append(area_feature[parent])
        combind_feature[parent].append(global_feature)
        combind_feature[parent] = np.array(combind_feature[parent]).reshape(1,1024)

    gt_path = os.path.join(gt_folder, imagename)
    gt_file = Image.open(gt_path)

    #get labels of areas
    counter = {}
    labels = {}
    for key in area_dict.keys():
        counter[key] = 0
        labels[key] = 0
    for parent in area_dict.keys():
        area = area_dict[parent]
        for number in area:
            i = number % width
            j = (number - i) / width
            gr, gg, gb = gt_file.getpixel((i, j))
            if(gr<=128 and gg<=128 and gb<=128):
                counter[parent] += 1
    for parent in area_dict.keys():
        if(counter[parent] >= len(area_dict[parent])/2):
            labels[parent] = 0
        else:
            labels[parent] = 1
```

2.2.4 Dataset

最后，定义 dataset() 函数，它返回处理好的训练集和测试集，方便后续 SVM 算法使用。数据集使用 dataframe 的形式，每一行对应一个划分的区域，它拥有 50 维特征和一个 label。

```
def dataset():
    testlist = [str(11 + (i * 100)) + ".png" for i in range(1, 10)]
    trainlist = [str(i) + ".png" for i in range(2,50) if i % 100 != 11 ]

    pca = PCA(n_components=50)

    #train dataset
    train_dataset,train_labels = gain_dataset("1.png")

    for train_pic in trainlist:
        dataset,pic_labels = gain_dataset(train_pic)
        train_dataset = train_dataset.append(dataset)
        train_labels += pic_labels
```

```
    train_dataset = pca.fit_transform(train_dataset)
    train_dataset = pd.DataFrame(train_dataset)
    train_dataset['labels'] = train_labels

    #test dataset
    test_dataset,test_labels = gain_dataset("11.png")

    for test_pic in testlist:
        dataset,pic_labels = gain_dataset(test_pic)
        test_dataset = test_dataset.append(dataset)
        test_labels += pic_labels
    test_dataset = pca.fit_transform(test_dataset)
    test_dataset = pd.DataFrame(test_dataset)
    test_dataset['labels'] = test_labels

    #print(train_dataset)
    #print(test_dataset)
    return train_dataset,test_dataset
```

	0	1	2	...	48	49	labels
0	-9.578698	-2.253356	-7.248905	...	4.134334e-16	-1.858902e-15	0
1	-6.813383	-2.234100	-8.685558	...	-2.854104e-16	-8.569201e-16	0
2	-1.206453	-2.025969	-6.862166	...	-3.024591e-16	8.349493e-17	0
3	-9.447582	-0.531828	-9.027741	...	1.063045e-16	4.793505e-16	0
4	-19.038377	-2.559384	-2.832376	...	-3.446011e-16	-2.787657e-16	0
5	-4.355513	8.847323	-5.434652	...	1.738709e-15	-5.812065e-16	0
6	-5.234190	8.208356	-5.751008	...	-8.700578e-16	3.589882e-16	0
7	-0.221853	8.757722	-4.338498	...	1.625529e-15	-5.457372e-16	0

图 4: Dataframe

2.2.5 SVM

准备好数据集，接下来就可以使用 SVM 模型对上述数据进行训练和分类了。构造 SVM 模型并训练的代码如下。其中，我展示了训练集精度，测试集精度以及 F1 score 三个衡量分类准确率的指标。

```
def main():
    #dataset 的形式是特征 + labels
    df_train, df_test = dataset()
    #labels = df.columns.values.tolist()

    #获取训练和测试数据
    x_train = np.array(df_train.ix[:,0:df_train.shape[1]-2].values.tolist() )
    y_train = np.array(df_train.ix[:,'labels'].values.tolist() )
    x_test = np.array(df_test.ix[:, 0:df_test.shape[1]-2].values.tolist())
```

```
y_test = np.array(df_test.ix[:, 'labels'].values.tolist())
#训练SVM 模型
clf = svm.SVC(C=0.8, kernel='linear', gamma=15, decision_function_shape='ovr')
clf.fit(x_train, y_train.ravel())

#验证模型，取得分数
print ("训练集精度：",clf.score(x_train, y_train))  # 精度
y_pred = clf.predict(x_train)
print ("测试集精度：",clf.score(x_test, y_test))
y_pred = clf.predict(x_test)
print("F1 score:",f1_score(y_test, y_pred, average='macro'))
```

以下展示了使用 50 张图片作为训练集的结果，可以看到，训练集精度为 0.86，测试集精度为 0.75。此外，输出了预测的各个区域对应的标注，可以看到，大部分的预测结果为背景，少部分为前景，这可能是训练集中背景的比例较大所导致的。

```
训练集精度： 0.8612424037812288
测试集精度： 0.7489539748953975
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

图 5: SVM Result

2.3 Sift features and K-means

最后一个实验，我们需要实现以下问题：对于每一张测试图，

- 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，舍去超出边界的，并计算归一化颜色直方图（4*4*4=64 维）
- 拼接上述两个特征，那么对于每一个 sift 特征点就得到了一个 74 维的特征，最后使用 k-means 算法将他们聚类成 3 类，并将各个 patch 黏贴到同一张图片上进行展示。

2.3.1 Sift Feature points and Feature

首先，调用 cv2.xfeatures2d.SIFT_create() 以及 sift.detectAndCompute() 方法获取 sift 特征和 sift 特征点。

```
def knn_sift(pic_num):
    pic_name = str(pic_num) +'.png'
    img = cv2.imread('./data/imgs/'+pic_name)
    img1 = img.copy()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    sift = cv2.xfeatures2d.SIFT_create()
    kp = sift.detect(gray, None)

    keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)

    cv2.drawKeypoints(gray, keypoints_1, img)
    cv2.drawKeypoints(gray, keypoints_1, img1, flags=cv2.
                                                DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

    plt.subplot(121), plt.imshow(img),
    plt.title('Dstination'), plt.axis('off')
    plt.subplot(122), plt.imshow(img1),
    plt.title('Dstination'), plt.axis('off')
    plt.show()
```

获取的 sift 特征点参考如下：

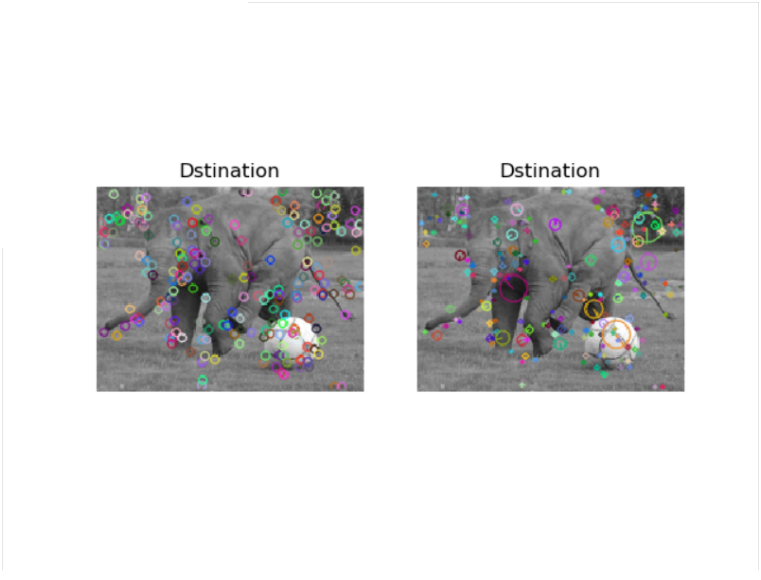

图 6: sift

2.3.2 Area RGB Histogram

接下来，获取 sift 特征点所在 patch 区域的 RGB 归一化颜色直方图。这里由于区域的归一化颜色直方图我在第二题中已经实现了一个比较完善的函数，此处只需要获得每一个 sift 特征点对应的 patch 内的点的编号即可。(编号 = y*width + x) 调用该函数，获得区域特征 area_feature，代码如下：

```
size = image.size
width = size[0]
height = size[1]
#get patch
area_dict = {}

point_num = 0
for keypoint in keypoints_1:
    x,y = keypoint.pt
    x = int(x+0.5)
    y = int(y+0.5)
    keypoint_num = x*width + y
    patch_nums = []
    for j in range(0,17):
        for i in range(0,17):
            a = (x-8+i)
            b = (y-8+j)
            if( a>=0 and a<=width-1 and b>=0 and b<=height-1):
                patch_nums.append(a + b*width)
    area_dict[point_num] = patch_nums
    point_num += 1


#gain area feature
area_feature = calculate_area_feature(image,area_dict,width,4)
```

2.3.3 K-means

然后，拼接两个特征，每个 sift 特征点得到一个 74 维的特征，使用 k-means 将一幅图中的 sift 特征点聚类为 3 类，并将结果使用 dataframe 的形式存储，将每个特征点的对应类作为一列添加到 dataframe 中。代码如下：

```
combind_feature = [[] for i in range(len(keypoints_1))]
sift_feature = sift_feature.tolist()
i = 0
for keypoint in area_feature.keys():
    combind_feature[i] += area_feature[keypoint]
    combind_feature[i] += sift_feature[i]
    i += 1
combind_feature = np.array(combind_feature)
```

```
#k-means
data = pd.DataFrame(combind_feature)
model = KMeans(n_clusters=3, init='k-means++')
model.fit(data)

r1 = pd.Series(model.labels_).value_counts()  # 统计各个类别的数目
r2 = pd.DataFrame(model.cluster_centers_)  # 找出聚类中心
r = pd.concat([r2, r1], axis=1)  # 横向连接（0是纵向），得到聚类中心对应的类别下的数
                                                目
r.columns = list(data.columns) + ['class num']  # 重命名表头
#print(r)

r = pd.concat([data, pd.Series(model.labels_, index=data.index)], axis=1)  # 详细输
                                                出每个样本对应的类别
r.columns = list(data.columns) + ['category']  # 重命名表头
```

最后，将每一个 patch 使用一个图片存储，使用红绿蓝三个颜色分别表示每一个类别，准备将它们粘贴到结果图“resulti.png”上作为最终结果。

```
if(not os.path.exists("patch_pic"+str(pic_num))):
    os.mkdir("patch_pic"+str(pic_num))
for center_num in area_dict.keys():
    category = r.at[center_num,'category']
    area = area_dict[center_num]
    colors = {}
    for pixel in area:
        color = [0,0,0]
        color[category] = 255
        color = tuple(color)
        colors[pixel] = color
    img = generate_image(width,height,colors)
    img.save('./patch_pic' + str(pic_num)+'/patch'+str(center_num)+".png")

img = generate_source_image(width,height)
img.save('./result'+str(pic_num)+'.png')
```

参考 patch 结果如下：

图 7: patch

2.3.4 Paste patches

最后一步，将每个测试图对应的 patch 按顺序粘贴到 result 图上，大功告成。这里我自定义了 blend_picture() 函数，它定义了图像粘贴的混合模式，让 result 中已经存在颜色的 patch 不再被其他的覆盖，不存在颜色的 patch 的颜色改为粘贴上来的 patch 的颜色。代码如下

```
def blend_image(width, height,img1,img2):
    img = Image.new('RGB', (width, height))
    im = img.load()
    for y in range(height):
        for x in range(width):
            r,g,b = img1.getpixel((x,y))
            r2,g2,b2 = img2.getpixel((x,y))
            if(r>0 or g>0 or b>0):
                im[x,y] = (r,g,b)
            else:
                im[x,y] = (r+r2,g+g2,b+b2)
    return img


def handle_img(pic_num):
    img_folder = './patch_pic' + str(pic_num)
    imgs = os.listdir(img_folder)
    imgNum = len(imgs)
    print(imgNum)

    for i in range(imgNum):
        img1 = Image.open(img_folder + '/' + imgs[i])
        #img = img1.resize((102, 102))  # 将图片调整到合适大小

        oriImg = Image.open("./result"+str(pic_num)+".png")  # 打开图片
        size = oriImg.size  # 获取图片大小尺寸
        width = size[0]
        height = size[1]
        # oriImg.paste(img, (image[0]-102, image[1]-102))
```

```
        oriImg = blend_image(width,height,oriImg,img1)

        oriImg1 = oriImg.convert('RGB')
        oriImg1.save("./result"+str(pic_num)+".png")
```

本题有大量的图片结果需要展示，此处给出 111.png 的结果作为参考，其余的图片请见附件。下列图片中，红，绿，蓝三个颜色分别代表三个聚簇，黑色为背景色。由于我的学号尾号为 11，我的附件中将展示 11.png-911.png 的全部结果。结合原图（原图与聚类结果的对比图已放在邮件附件）可以看到，k-means 能够较好地将 sift 特征点中具有相似特征的点聚类出来，但是它仍然可能在少数区域受到背景噪音的影响。

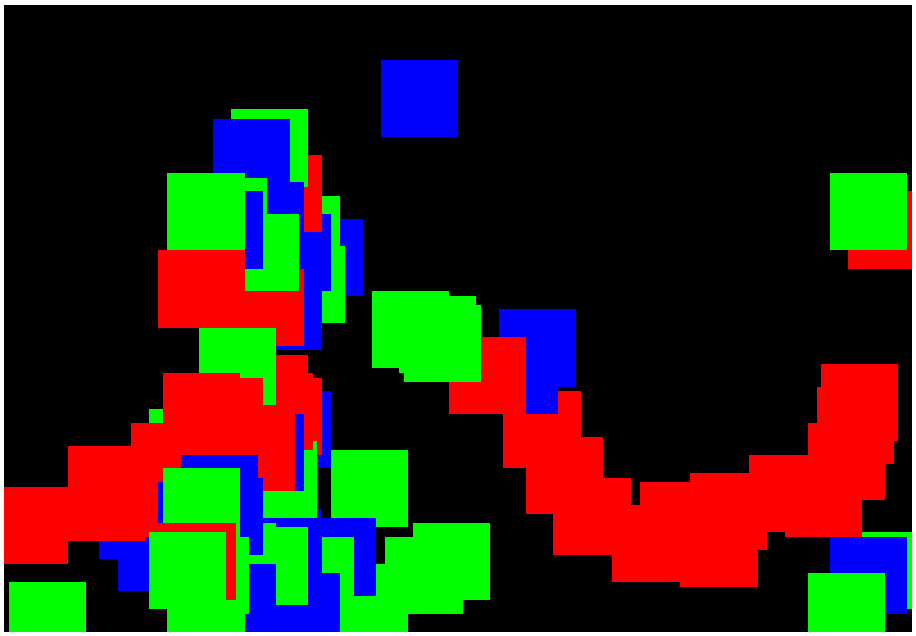

图 8: 111.png