图像处理第四次作业

刘坤鑫*

2020 年 1 月 7 日

摘要

本文主要实现了 Shape Context 算法，并对结果进行了分析。

1 Shape Context

1.1 原理

根据维基百科[1]及原始论文[2]，形状上下文算法可描述如下：

1. Finding a list of points on shape edges. 将图像边缘提取出来，并对边缘进行均匀采样。

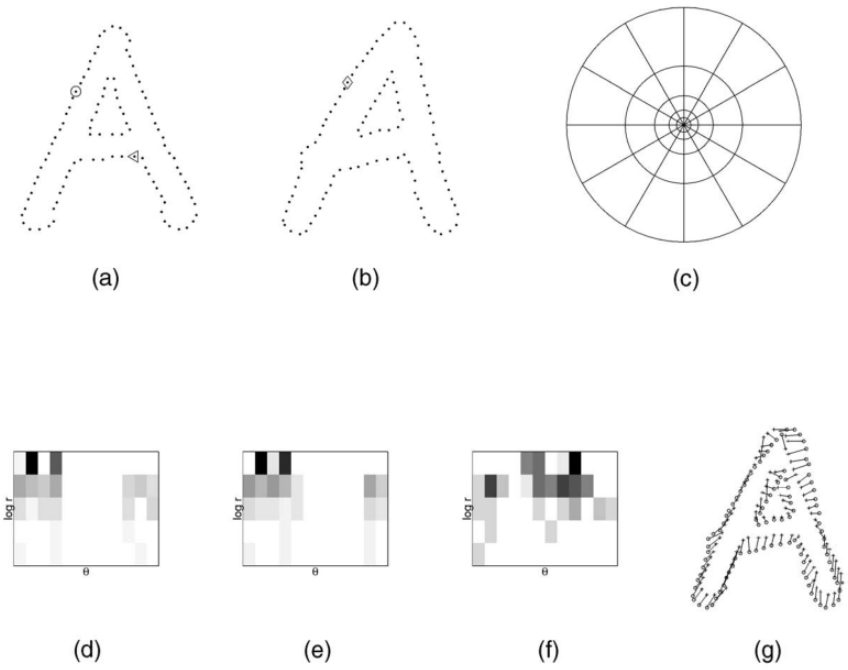


图 1: k-bin 统计图

*3017218061 软件工程一班

[1]https://en.wikipedia.org/wiki/Shape_context

[2]https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=853834

2. Computing the shape context. 如图1所示，计算形状上下文，即论文中的 k-bin 图。计算方法如下：对采样的每个点，统计出其他所有点到其的方位角及对数距离的值。

3. Computing the cost matrix. 计算费用矩阵。费用矩阵中，行表示图 A 的所有采样点，列表示图 B 的所有采样点，采样点的费用计算根据统计图得到。计算公式如下：

$$C_S = \frac{1}{2}\sum_{k=1}^{K}\frac{[g(k)-h(k)]^2}{g(k)+h(k)} \tag{1}$$

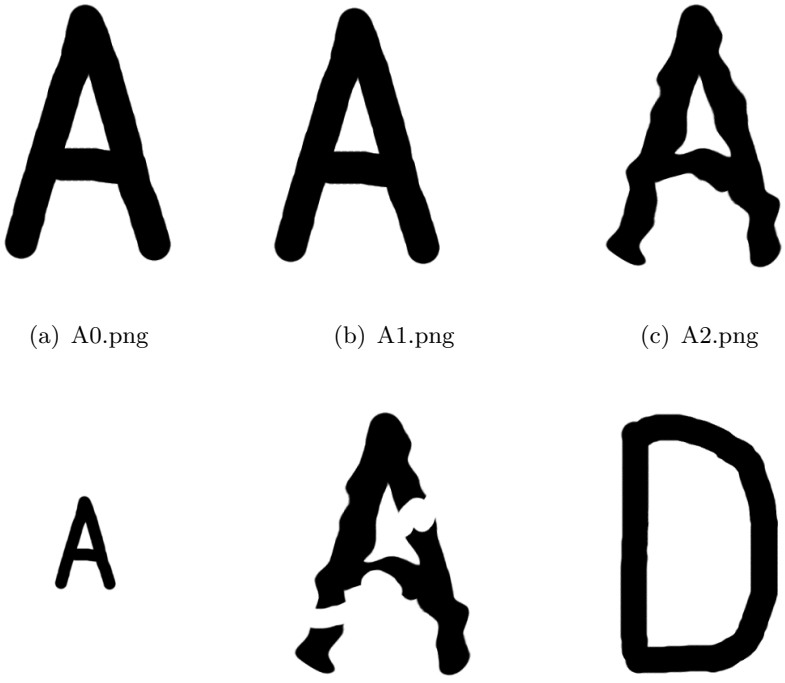
4. Finding the matching that minimizes total cost. 给两个图的采样点进行一一对应匹配，使得总费用最小。问题转化为经典的带权二分图匹配，可用改进的匈牙利算法 KM 算法求出，时间复杂度为 $O(n^3)$

2 详细设计

见附录 (内含注释)。

3 结果及结论

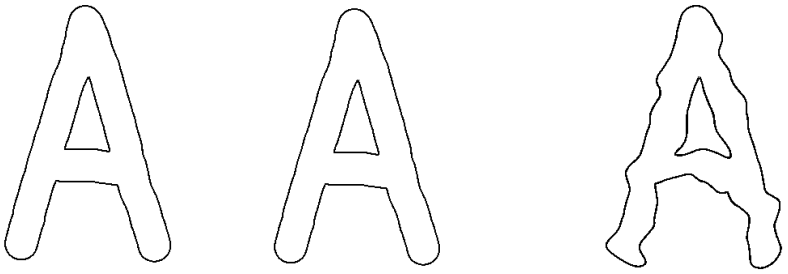
如图所示2，第一张为原图字母 A，后面为处理后的图像及对照字母 D。
如图所示3，为提取边缘后的图像。
如图所示4，均匀采样 200 个点后的图像。



(a) A0.png (b) A1.png (c) A2.png

(d) A3.png (e) A4.png (f) A5.png

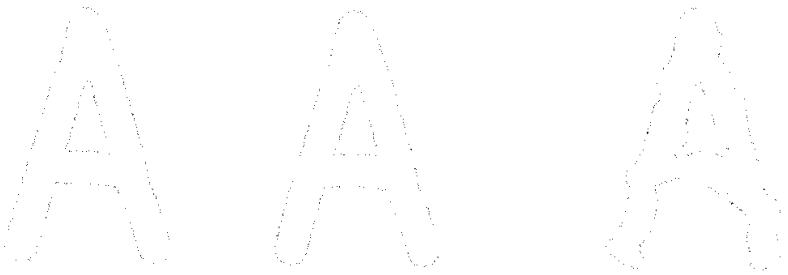
图 2: 原始图像



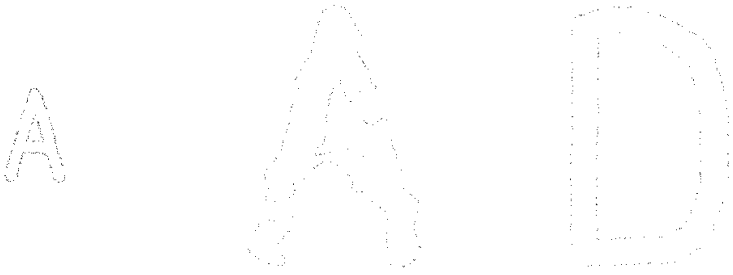
(a) A_points0.png (b) A_points1.png (c) A_points2.png

(d) A_points3.png (e) A_points4.png (f) A_points5.png

图 3: 提取边缘后的图像



(a) A_points_sample0.png (b) A_points_sample1.png (c) A_points_sample2.png



(d) A_points_sample3.png (e) A_points_sample4.png (f) A_points_sample5.png

图 4: 均匀采样 200 个点后的图像

如表3所示为不同采样点数的计算结果，如表3所示为原图与其他图像的结果。可以看出，原图经过平移、缩放、略微变形等处理后，计算结果几乎无变化。如果形状变化过大，结果就会变大。如果采样点数越多，精确程度越高，当然运行时间越慢。

表 1: 不同采样点数的计算结果

采样点数	结果
50	2053
100	8020
200	32142

表 2: 原图与其他图像的结果

图像编号 (原图为 0)	结果
1	8020
2	8091
3	8104
4	7990
5	8326

A 完整源码

ShapeContext.py

```
from PIL import Image
import os
import cv2
from skimage import io, filters
import matplotlib.pyplot as plt
import numpy as np
np.random.seed(19260817)


def get_images():
    '''
    读取所有图像
    '''
    images = []
    for i in range(6):
        path = os.path.join('pic', 'A{}.png'.format(i))
        image = io.imread(path, as_gray=True)
        images.append(image)
    return images


def get_edges(image):
    '''
    提取边缘，采用sobel算子
    '''
    edges = filters.sobel(image)
    return edges


def get_points(edges):
    '''
    先对图像进行二值化，再抽离出所有点。
    '''
    mean = np.mean(edges[edges > 0])
    edges = edges > mean
    res = []
    row, col = edges.shape
    for i in range(row):
        for j in range(col):
            if edges[i][j]:
                res.append((i, j))
    return np.array(res)


def sample(points, size=100):
    '''
    对抽离出来的点进行随机采样，数目为size个。
    '''
    idx = np.random.choice(np.arange(len(points)), size=size)
    return points[idx]


def save_points(points, shape, name):
    '''
    将点转化为图像并保存。
```

```
    '''
    for i, r in enumerate(points):
        image = np.zeros(shape=shape, dtype=np.uint8)
        for p in r:
            image[p[0], p[1]] = 255
        image = 255-image
        path = os.path.join('pic', '{}{}.png'.format(name, i))
        Image.fromarray(image).save(path)


def k_bin(A, k=12, bins=3):
    '''
    计算k-bin统计图，默认方位12等分，最大距离1000(log1000=3)
    '''
    n = len(A)
    k_bin = np.zeros(shape=(n, k, bins), dtype=np.int32)
    for i in range(n):
        for j in range(n):
            theta = np.arctan2(A[j][1]-A[i][1], A[j][0]- A[i][0])
            dis = np.hypot(A[j][1]-A[i][1], A[j][0]- A[i][0])
            x = int(np.floor((theta/np.pi+1)*6-1e-6))
            y = int(np.floor(np.log10(dis+1)))
            k_bin[i][x][y] += 1
    return k_bin


def get_cost(A, B, k=12, bins=3):
    '''
    计算花费矩阵
    '''
    k_bin_A = k_bin(A, k, bins)
    k_bin_B = k_bin(B, k, bins)
    nA = len(A)
    nB = len(B)
    cost = np.zeros(shape=(nA, nB))
    for i in range(nA):
        for j in range(nB):
            m, n = k_bin_A[i], k_bin_B[j]
            t = m+n > 0
            m, n = m[t], n[t]
            cost[i][j] = np.sum((m-n)**2/(m+n+1e-6))/2
    assert(np.sum(np.isnan(cost)) == 0)
    return cost


def KM(g):
    '''
    KM算法，计算带权二分图的最小费用。
    Input:
        np.array([
            [3, 4, 6, 4, 9],
            [6, 4, 5, 3, 8],
            [7, 5, 3, 4, 2],
            [6, 3, 2, 2, 5],
            [8, 4, 5, 4, 7],
        ], dtype=np.float64)
    Output:
        29
    '''
    nx, ny = g.shape
    linker = np.full(ny, fill_value=-1, dtype=np.int32)
```

```
    lx = np.zeros(nx, dtype=np.float64)
    ly = np.zeros(ny, dtype=np.float64)
    slack = np.zeros(ny, dtype=np.float64)
    vis_x = np.zeros(nx, dtype=np.bool)
    vis_y = np.zeros(ny, dtype=np.bool)

    def dfs(x):
        nonlocal g, ny, linker, lx, ly, slack, vis_x, vis_y
        vis_x[x] = True
        for y in range(ny):
            if vis_y[y]:
                continue
            t = lx[x]+ly[y]-g[x][y]
            if (np.abs(t) < 1e-6):
                vis_y[y] = True
                if linker[y] == -1 or dfs(linker[y]):
                    linker[y] = x
                    return True
            elif slack[y] > t:
                slack[y] = t
        return False

    lx = np.max(g, axis=1)
    for x in range(nx):
        slack[:] = np.inf
        while True:
            vis_x[:], vis_y[:] = False, False
            if dfs(x):
                break
            d = np.min(slack[vis_y == False])
            lx[vis_x] -= d
            ly[vis_y] += d
            slack[vis_y == False] -= d
    t = linker != -1
    res = np.sum(g[linker[t], t])
    return res


def shape_context(A, B):
    '''
    计算形状上下文的值，这里只计算到带权二分图的费用，
    实际论文中后面还有几个步骤没有实现。
    '''
    cost = get_cost(A, B)
    return KM(cost)


'''
采样点数 结果
50: 2053
100: 8020
200: 32142

100个采样点下，不同结果
1: 8020
2: 8091
3: 8104
4: 7990
5: 8326
'''
```

```
def main():
    images = get_images()
    shape = images[0].shape
    edges = [get_edges(i) for i in images]
    points = [get_points(i) for i in edges]
    points_sample = [sample(i) for i in points]
    save_points(points, shape, 'A_points')
    save_points(points_sample, shape, 'A_points_sample')
    print(shape_context(points_sample[0], points_sample[5]))


if __name__ == '__main__':
    main()
```