图像处理大作业——超像素分割

刘书裴　刘坤鑫　郭炳成

2020 年 1 月 5 日

摘要

通过将具有相似纹理、颜色、亮度等特征的相邻像素构成的有一定视觉意义的不规则像素块，从而用少量的超像素代替大量的像素来表达图片特征，很大程度上降低了图像后处理的复杂度，这就是超像素分割。本文使用 python 实现了超像素分割中的 SLIC 算法，并在此基础上进行多种优化，最终得到了不错的效果。

1 小组成员信息

表 1: 小组成员信息

组员	学号
刘书裴	3017218062
刘坤鑫	3017218061
郭炳成	3017218056

2 SLIC

2.1 算法实现

2.1.1 RGB 转 LAB

公式:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{1}$$

$$L^* = 116 f\left(Y/Y_n\right) - 16 \tag{2}$$

$$a^* = 500\left[f\left(X/X_n\right) - f\left(Y/Y_n\right)\right] \tag{3}$$

$$b^* = 200\left[f\left(Y/Y_n\right) - f\left(Z/Z_n\right)\right] \tag{4}$$

我们可以使用 skimage.color.rgb2lab() 来实现这个操作。

2.1.2 初始化聚类中心

- 首先，我们设置超像素或者说聚类的数量 K。
- 然后按照等大小将图片分为 K 个超像素块。

$$\text{super PixelLength} = \sqrt{S/K} \tag{5}$$

- 如图1所示，初始化聚类中心，用一个名为 Cluster 的数组保存它们。

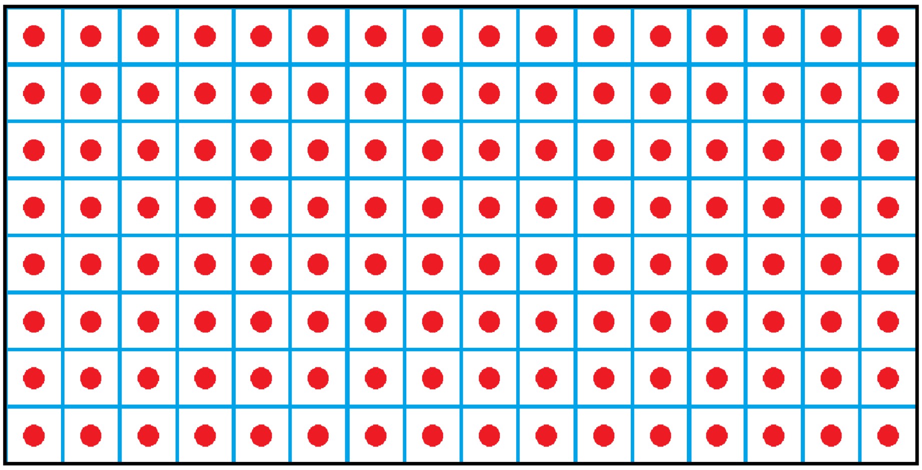

图 1: 初始化聚类中心

- 我们需要名为 Distance 和 Label 的两个数组，分别用来保存当前点和最近的聚类中心的距离和最近的聚类中心的标签。

2.1.3 迭代聚类

对于每次迭代，我们选取点周围 2Sx2S 的区域：

- 计算 Distance 数组。(每个点与它们最近的聚类中心的距离)

$$d_c = \sqrt{(l_i - l_j)^2 + (a_i - a_j)^2 + (b_i - b_j)^2} \tag{6}$$

$$d_s = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{7}$$

$$d = \sqrt{\left(\frac{d_c}{m}\right)^2 + \left(\frac{d_s}{s}\right)^2} \tag{8}$$

  - m(Nc) 是 Lab 空间中的最大颜色距离，设置为 10。
  - s(Ns) 是 Lab 空间中的最大空间距离，设置为 superPixelLength。

- 选择 Label 数组。(每个点最近的聚类中心的标签)
- 更新聚类中心。(使用每个聚类的几何中心)

2.2 注意

2.2.1 为什么要将 rgb 转为 lab？

- 它不仅是一种与设备无关的颜色模型，而且是一种基于生理特性的颜色模型。LAB 颜色模型由三个元素组成，一个是亮度（L），A 和 B 是两个颜色通道。A 包括从深绿色（低亮度值）到灰色（中等亮度值）到亮粉色（高亮度值）的颜色；B 是从亮蓝色（低亮度值）到灰色（中等亮度值）到黄色（高亮度值）。因此，这种颜色在混合后会产生鲜艳的颜色，LAB 模式定义了最多的颜色。
- LAB 模式的分割效果优于 RGB 模式下的分割效果。

2.2.2 为什么要选择 2Sx2S 的区域？

- 缩小了超像素的搜索区域。
- 使 SLIC 的复杂度无关与超像素的数量。

2.3 算法优化

2.3.1 选择梯度

它可以避免在边缘定位超像素，并且减少用噪声代替超像素的机会。
在每个点的 3x3 的区域中：

- 分别计算附近 8 个点的梯度。

$$dx(i,j) = [I(i+1,j) - I(i-1,j)]/2 \tag{9}$$

$$dy(i,j) = [I(i,j+1) - I(i,j-1)]/2 \tag{10}$$

$$g(i,j) = \sum_{k=0}^{2}(dx(i,j) + dy(i,j)) \tag{11}$$

- 选择梯度最小的点为新的聚类中心。

2.3.2 合并小块

由于聚类过程的特点，并不能保证每个类在 XY 空间都是连续的。
首先，使用 BFS 找出每个连接的块。
时间复杂度为 O(mn)，其中 n 和 m 分别为图像的长度和宽度。
然后，当块的大小小于预设阈值时，使用并行搜索集合并连接小块。

- 计算所有连通图。
- 初始化阈值。
- 将小块合并到附近的块中。

2.3.3 绘制边界

显然，在四个方向上具有不同标签的点是边界点。
由于数组的遍历方法，我们只需要查看右方或下方的点。
此外，如果左边或上面有标记，我们可以跳过这个点。

2.4 结果

分割后的超像素如图2所示。
分割后的图片如图3所示。

图 2: 分割后的超像素

图 3: 分割后的图片

A 完整源码

SLIC.py

```
from tool.imageTool import *
import math


class Cluster(object):
    ID = 0
    l, a, b, x, y = 0, 0, 0, 0, 0

    def __init__(self, l, a, b, x, y, ID=0):
        self.l = l
        self.a = a
        self.b = b
        self.x = x
        self.y = y
        self.ID = ID

    def update(self, l, a, b, x, y):
        self.l = l
        self.a = a
        self.b = b
        self.x = x
        self.y = y

    def __str__(self):
        return "{},{}:{} {} {} ".format(self.x, self.y, self.l, self.a, self.b)


class SLIC():
    def __init__(self, image, k, iterNumber=3):
        """
        - Initialize the image and its properties.
        - Set the length of super pixels.
        - Create label and distance with height and width.
            - label: Used to record which super pixel the current point belongs to.
            - distance: Used to record the shortest distance between the current point and the center of the super
                pixel.
        - Define a array saving the cluster information.
        """
        self.image = image
        self.height, self.width = self.image.shape[:2]
        self.superPixelLength = math.sqrt(self.height * self.width / k)
        self.label = np.zeros((self.height, self.width), dtype="int")
        self.distance = np.zeros((self.height, self.width), dtype="int")
        self.distance[:, :] = 1e9
        self.clusters = []
        self.clusterNumber = 0
        self.iterNumber = iterNumber
        self.connectNumber = 0
        self.boundary = color.rgb2lab(np.zeros(self.image.shape))

    def run(self):
        self.__clusterInit()
        self.__clusterMove()
        for i in range(self.iterNumber):
            self.__labelChoose()
```

```
            self.___clusterUpdate()
        self.___enforceConnect()
        self.___imageSplit()

    def ___clusterInit(self):
        """
        - Split the image according to superPixelLength.
        - Add all super pixels to clusters array.
        """
        print(" Initializing ... ")
        for i in range(int(self.height / self.superPixelLength)):
            for j in range(int(self.width / self.superPixelLength)):
                self.clusterNumber += 1
                x = int(self.superPixelLength * (i + 0.5))
                y = int(self.superPixelLength * (j + 0.5))
                l, a, b = image[x][y]
                self.clusters.append(Cluster(l, a, b, x, y, self.clusterNumber))

    def ___clusterMove(self):
        """
        - Define gradientCalculate function.
        - Change clusters' center according to smallest gredient.
        """

        def gradientCalculate(self, x, y):
            gradient = 0
            for i in range(3):
                gradient += math.fabs(self.image[x][y + 1][i] - self.image[x][y - 1][i]) + math.fabs(
                    self.image[x + 1][y][i] - self.image[x - 1][y][i])

            return gradient / 2

        print("Moving...")
        for index in range(self.clusterNumber):
            cluster = self.clusters[index]
            x, y = cluster.x, cluster.y
            minGradient = gradientCalculate(self, x, y)
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    if dx == dy == 0:
                        continue
                    _x = cluster.x + dx
                    _y = cluster.y + dy
                    currentGradient = gradientCalculate(self, _x, _y)
                    if currentGradient < minGradient:
                        minGradient = currentGradient
                        x, y = _x, _y
            currentPixel = self.image[x][y]
            self.clusters[index].update(currentPixel[0], currentPixel[1], currentPixel[2], x, y)

    def ___labelChoose(self):
        """
        - Define the distCalculate function.
        - Choose the nearest cluster's center for every point.
        - Update every cluster's center after choosing.
        """

        def distCalculate(point, clusterCenter, Nc=10, Ns=self.superPixelLength):
            Dc2 = (point.l - clusterCenter.l) ** 2 + (point.a - clusterCenter.a) ** 2 + (point.b - clusterCenter.b)
                ** 2
            Ds2 = (point.x - clusterCenter.x) ** 2 + (point.y - clusterCenter.y) ** 2
```

```
            D = Dc2 * (Ns ** 2) + Ds2 * (Nc ** 2)
            return D

        print("Clustering...")
        h, w = self.height - 1, self.width - 1
        for index in range(self.clusterNumber):
            clusterCenter = self.clusters[index]
            for i in range(int(clusterCenter.x - self.superPixelLength), int(clusterCenter.x +
                 self.superPixelLength)):
                if i < 0 or i > h: continue
                for j in range(int(clusterCenter.y - self.superPixelLength),
                               int(clusterCenter.y + self.superPixelLength)):
                    if j < 0 or j > w: continue
                    currentPixel = self.image[i][j]
                    D = distCalculate(Cluster(currentPixel[0], currentPixel[1], currentPixel[2], i, j), clusterCenter)
                    if D < self.distance[i][j]:
                        self.label[i][j], self.distance[i][j] = clusterCenter.ID, D

    def __clusterUpdate(self):
        """
        - Calculate the average of every cluster.
        - Use average to replace center.
        """
        print("Updating...")
        X, Y, num = [0] * self.clusterNumber, [0] * self.clusterNumber, [0] * self.clusterNumber
        for i in range(self.height):
            for j in range(self.width):
                k = self.label[i][j] - 1
                X[k] += i
                Y[k] += j
                num[k] += 1

        for index in range(self.clusterNumber):
            x, y = X[index] // num[index], Y[index] // num[index]
            currentPixel = self.image[x][y]
            self.clusters[index].update(currentPixel[0], currentPixel[1], currentPixel[2], x, y)

    def __enforceConnect(self):
        """
        - connect(): Get the connected graphs.
        - find(): Component identifier for x.
        - union(): Add connection between i and j.
        """
        def connect():
            label = self.label
            visit = np.zeros((self.height, self.width), dtype=np.int32)
            dx, dy = [-1, 1, 0, 0], [0, 0, 1, -1]

            for i in range(self.height):
                for j in range(self.width):
                    if not visit[i][j]:
                        self.connectNumber += 1
                        Q = [(i, j)]
                        visit[i][j] = self.connectNumber
                        while len(Q):
                            x, y = Q.pop(0)
                            for k in range(4):
                                xx, yy = x + dx[k], y + dy[k]
                                if 0 <= xx < self.width and 0 <= yy < self.height and \
                                        not visit[xx][yy] and label[xx][yy] == label[i][j]:
                                    visit[xx][yy] = self.connectNumber
```

```python
                                    Q.append((xx, yy))
            return visit

        def find(x):
            if p[x] == x:
                return x
            else:
                p[x] = find(p[x])
                return p[x]

        def union(i, j):
            x = find(i)
            y = find(j)
            if x != y:
                p[x] = y
                blockCount[x] += blockCount[y]

        blockLabel = connect()
        threshold = self.superPixelLength ** 2 / 4
        blockColor = np.zeros(self.connectNumber + 1, dtype=np.int32)
        blockCount = np.zeros(self.connectNumber + 1, dtype=np.int32)
        p = np.arange(self.connectNumber + 1)

        for i in range(self.height):
            for j in range(self.width):
                blockCount[blockLabel[i][j]] += 1
                if not blockColor[blockLabel[i][j]]:
                    blockColor[blockLabel[i][j]] = self.label[i][j]

        for i in range(self.height - 1):
            for j in range(self.width - 1):
                x = find(blockLabel[i][j])
                if blockCount[x] < threshold:
                    y = find(blockLabel[i][j + 1])
                    z = find(blockLabel[i + 1][j])
                    if x != y:
                        union(x, y)
                    elif x != z:
                        union(x, z)

        for i in range(self.height):
            for j in range(self.width):
                f = find(blockLabel[i][j])
                blockLabel[i][j] = f
                self.label[i][j] = blockColor[f]

    def __imageSplit(self):
        """
        - Draw the boundary of super pixels.
            - Depending on the surrounding super pixels.
            - Because pixels are traversed by rows and columns, we only need to look at the right and bottom.
            - Determine whether there is a divided boundary around to refine the boundary.
        - Draw all super pixel blocks.
        """
        print("Splitting ... ")
        for i in range(self.height):
            up = i - 1 if i - 1 > -1 else i
            down = i + 1 if i + 1 < self.height else i
            for j in range(self.width):
                left = j - 1 if j - 1 > -1 else j
                right = j + 1 if j + 1 < self.width else j
```

```
                k = self.label[i][j]
                if (k != self.label[i][right] and self.image[i][left][0] != 100) or (
                        k != self.label[down][j] and self.image[up][j][0] != 100):
                    self.image[i][j] = np.asarray([100, 0, 0])
                    continue
                cluster = self.clusters[k - 1]
                self.image[i][j] = np.asarray([cluster.l, cluster.a, cluster.b])

    def imageSave(self, filename):
        newImage = (color.lab2rgb(self.image) * 255).astype(np.uint8)
        io.imsave(filename, newImage)

    def boundarySave(self, filename):
        newImage = (color.lab2rgb(self.boundary) * 255).astype(np.uint8)
        io.imsave(filename, newImage)


if __name__ == "__main__":
    image = imageLoad()
    slic = SLIC(image, k=900, iterNumber=5)
    slic.run()
    slic.imageSave("../result/lena_SLIC_pixel.png")
    # slic.boundarySave("../result/cloth_SLIC_boundary.png")
```

test.py

```
from skimage import io, morphology, filters
from matplotlib import pyplot as plt

image = io.imread("../result/lena_SLIC_boundary.png", as_gray=True)
image = morphology.dilation(image)
io.imshow(image)
plt.show()

image = morphology.erosion(image)
io.imshow(image)
plt.show()
```