МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Полоцкий государственный университет»

Факультет информационных технологий

Кафедра технологий программирования

**ОТЧЕТ** **ПО ПРАКТИЧЕСКИМ РАБОТАМ**

Дисциплина: **«**Цифровая обработка изображений**»**

ВЫПОЛНИЛ студент группы 15-ИТ-1

Веселов С.В.

ПРОВЕРИЛ Богуш Р.П.

Полоцк 2019 г.

Цель: изучить и реализовать один из методов с использованием OpenCV или dlib по варианту.

Вариант 1. Суперпиксельная сегментация цветного изображения.

**Теоретическая часть**

Сегментация — это процесс разделения цифрового изображения на несколько сегментов (множеств пикселей). Очень эффективным с точки зрения вычислительных ресурсов является использование для сегментации методов кластерного анализа. Суть кластеризации состоит в том, что все исходные объекты (в данном случае пиксели ) разбиваются на несколько не пересекающихся групп таким образом, чтобы объекты, попавшие в одну группу, имели сходные характеристики, в то время как у объектов из разных групп эти характеристики должны значительно отличаться. Полученные группы называются кластерами. Исходными значениями в простейшем способе для кластеризации являются координаты пикселя (x, y).

#### Метод к-средних.

#### Центроид — точка которая является центром кластера. k-средних (k-means) — наиболее популярный метод кластеризации. Алгоритму широко отдается предпочтение из-за его простоты реализации, большой скорости (а это очень важно при работе с видео). Действие алгоритма таково, что он стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров. В простонародье говоря, это итеративный алгоритм, который делит данное множество пикселей на k кластеров точки, которых являются максимально приближенными к их центрам, а сама кластеризация происходит за счет смещения этих же центров.

**Результат работы**

**Листинг 1** – Реализация программы

import math

import numpy as np

from PIL import Image

import random

from collections import defaultdict

import operator

im = Image.open('./input2.jpg')

arr = np.asarray(im)

arr\_out = np.asarray(im.convert('L'))

rows,columns = np.shape(arr\_out)

print('background pixel level',arr\_out[0][0])

#print '\nrows',rows,'columns',columns

ground\_out = np.zeros((rows,columns))

''' Converting ground truth image into binary image for evaluation'''

for i in range(rows):

  for j in range(columns):

    if arr\_out[i][j] > 120:

      ground\_out[i][j] = 0

    else:

      ground\_out[i][j] = 1

shape = np.shape(arr)

rows = shape[0]

columns = shape[1]

'''obtaining 6 random centroid points'''

r\_points = [ random.randint(0, 255) for i in range(6) ]

g\_points = [ random.randint(0, 255) for i in range(6) ]

b\_points = [ random.randint(0, 255) for i in range(6) ]

grey\_l = defaultdict(list)

''' Grey scale levels corresponding to 6 clusters'''

grey1 = 40

grey2 = 80

grey3 = 120

grey4 = 160

grey5 = 200

grey6 = 240

grey\_l[0] = 40

grey\_l[1] = 80

grey\_l[2] = 120

grey\_l[3] = 160

grey\_l[4] = 200

grey\_l[5] = 240

g = defaultdict(list)

g2 = []

g3 = []

g4 = []

g5 = []

g6 = []

end = np.zeros((rows,columns))

zavg = [0,0,0]

''' computing average centroids after every iteration'''

def find\_centroids(g):

  red\_cent\_list = []

  blue\_cent\_list = []

  green\_cent\_list = []

  #print '0 shape',np.shape(g)

  for i in range(0,6):

    array = np.matrix(g[i])

    avg = np.mean(array,0)

    #print '\naverage values',avg

    pavg = np.ravel(avg)

    #print '2 shape', np.shape(pavg)

    #print 'pavg', pavg

    if not len(pavg):

      red\_cent\_list.append(zavg[0])

      blue\_cent\_list.append(zavg[1])

      green\_cent\_list.append(zavg[2])

    else:

      red\_cent\_list.append(pavg[0])

      blue\_cent\_list.append(pavg[1])

      green\_cent\_list.append(pavg[2])

  return[red\_cent\_list,blue\_cent\_list,green\_cent\_list]

''' Computing 10 iterations to obtain converged centroids of six clusters'''

for it in range(0,10):

  print('\niteration',it)

  g= defaultdict(list)

  for r in range(rows):

    for c in range(columns):

      img = arr[r][c]

      #print '\nimg', img

      red = img[0]

      green = img[1]

      blue = img[2]

      #print '\n red',red,'blue',blue,'green',green

      distance\_list = []

      for k in range(0,6):

        ''' computing absolute distance from each of the cluster and assigning it to a particular cluster based on distance'''

        #print '\n red ref point',r\_points[k],'blue ref point ',b\_points[k],'green ref point',g\_points[k]

        distance = math.sqrt(((int(r\_points[k])- red)\*\*2)+((int(g\_points[k]) - green)\*\*2)+((int(b\_points[k])-blue)\*\*2))

        #print '\ndistance',distance

        distance\_list.append(distance)

      #print '\ndistance list',distance\_list

      index, value = min(enumerate(distance\_list), key=operator.itemgetter(1))

      end[r][c] = grey\_l[index]

      #print '\nindex',index

      g[index].append([red,blue,green])

  centroids= find\_centroids(g)

  #print 'centroids',centroids

  r\_points = []

  b\_points = []

  g\_points = []

  r\_points = centroids[0]

  b\_points = centroids[1]

  g\_points = centroids[2]

  #print '\nr points',r\_points

  #print '\nb points',b\_points

  #print '\ng points',g\_points

''' From observation we know that ground truth image pixel[0,0] is part of background. This is used for evaluation'''

result = np.zeros((rows,columns))

ref\_val = end[0][0]

#print '\nref\_val',ref\_val

for i in range(rows):

  for j in range(columns):

    if end[i][j] == ref\_val:

      result[i][j] = 1

    else:

      result[i][j] = 0

'''displaying the clusters in different gray scale levels'''

plt.figure()

plt.imshow(result, cmap="Greys\_r")

plt.colorbar()

plt.show()

Дя начала мы переводим изображение в бинарный вид.

Затем берем из фотографии шесть случайных точек, что и будет являться нашими центроидами.

Далее проходим через 10 итераций, чтобы получить сходящиеся центроиды для шести кластеров.

Результаты работы программы представлены на рисунках 1 и 2.



Рисунок 1 – изначальное изображение

Рисунок 2 – изображение после сегментации