2D_Gaussian_filter_test

February 28, 2019

0.1 ASSIGNMENT 1

0.2 CS5187 VISION AND IMAGE

You are required to complete both written and programming tasks in this assignment.

0.2.1 PART-1: PROOF

A. Show that a 2D Gaussian filter is separable into two 1D Gaussian filters. (5%)

0.3 ASSIGNMENT 1

0.4 CS5187 VISION AND IMAGE

You are required to complete both written and programming tasks in this assignment. ### PART-1: PROOF A. Show that a 2D Gaussian filter is separable into two 1D Gaussian filters. (5%) B. Derive the 1st derivative of 2D Gaussian filter. (5%) C. Derive the 2nd derivative of 2D Gaussian filter. (5%) D. Derive the Laplacian of Gaussian (LoG) filter.

```
In [2]: import cv2
        import numpy as np
        import matplotlib.pylab as plt
        import scipy.stats as st
        import math
        from scipy import ndimage
        from skimage import io
        import os
        import heapq
        import scipy
        from scipy.misc import imread
        import cPickle as pickle
        import matplotlib.pyplot as plt
```

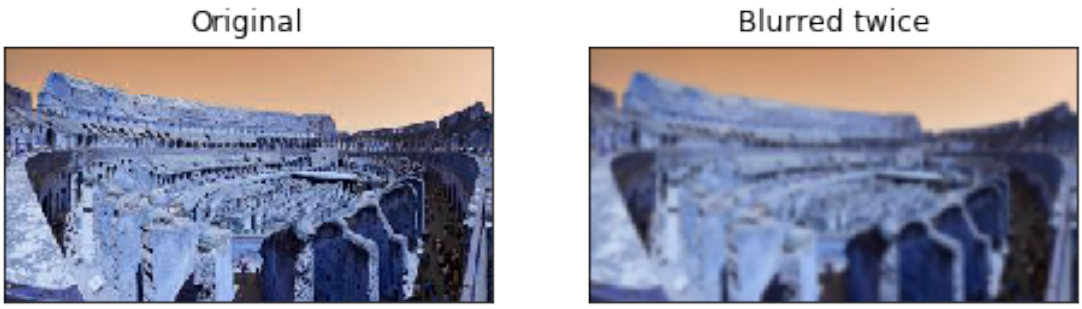
```
In [6]: imgPath = 'Assignment1_data/data/0000.jpg'
```

```
In [254]: def showImg(row, col, title, kernel, index):
              plt.subplot(row, col, index)
              #plt.subplots_adjust(left=0.0,bottom=0.0,top=1,right=1)
              #plt.subplots_adjust(wspace =0, hspace =0)#
```

```
        #plt.subplots_adjust(top = 1, bottom = 0, right = 1, left = 0, hspace = 0, wspac
        #plt.margins(0,0)
        plt.imshow(kernel, interpolation='none')
        plt.title(title)
        plt.xticks([])
        plt.yticks([])
```

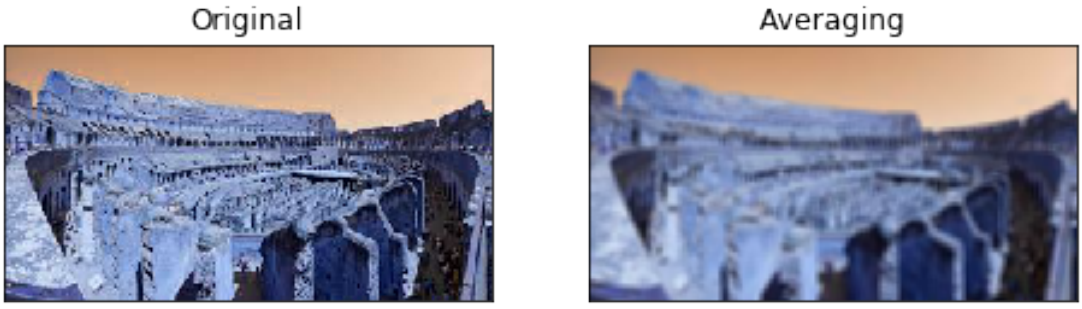
1D Gaussian filters:

```
In [29]: img = cv2.imread(imgPath)
         blur = cv2.GaussianBlur(img, (5, 5), 0)
         blur1 = cv2.GaussianBlur(blur, (5, 5), 0)
         plt.figure(figsize=(8, 8))
         showImg(1, 2, 'Original', img, 1)
         showImg(1, 2, 'Blurred twice', blur1, 2)
         plt.show()
```



2D Gaussian filters:

```
In [30]: img = cv2.imread(imgPath)
         kernel = np.ones((5, 5), np.float32)/25
         dst = cv2.filter2D(img, -1, kernel)
         plt.figure(figsize=(8, 8))
         showImg(1, 2, 'Original', img, 1)
         showImg(1, 2, 'Averaging', dst, 2)
         plt.show()
```



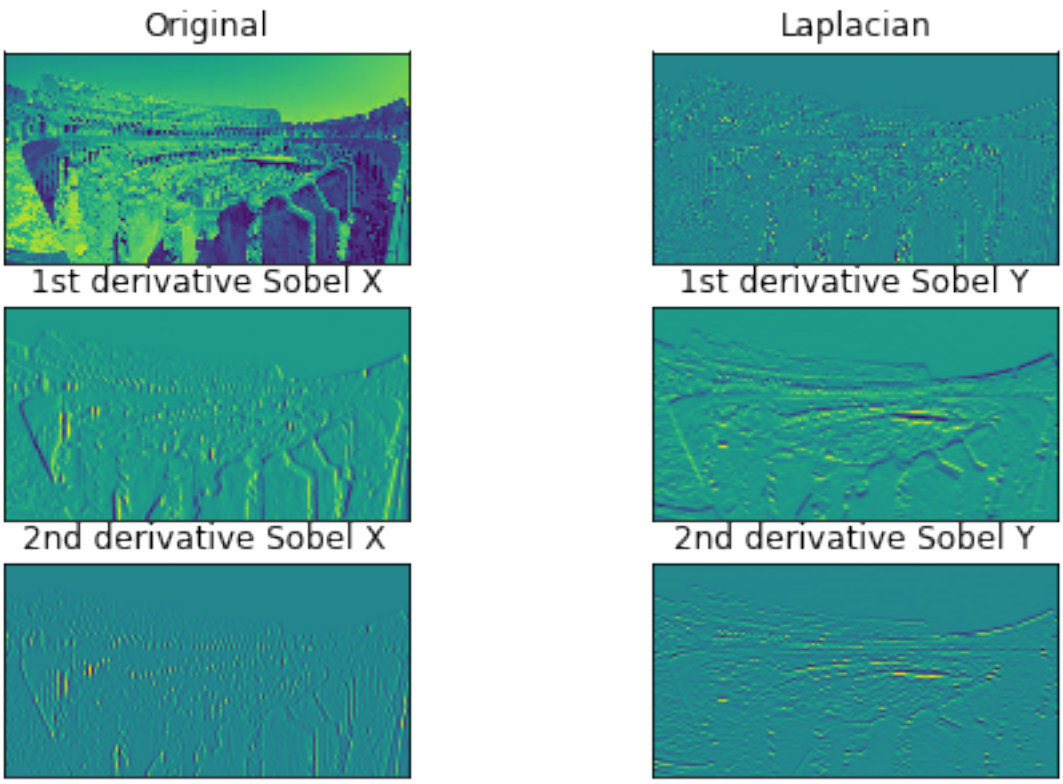
As you can see, the results are almost the same. These two methods perform consistently.

B. Derive the 1st derivative of 2D Gaussian filter. (5%)

C. Derive the 2nd derivative of 2D Gaussian filter. (5%)

D. Derive the Laplacian of Gaussian (LoG) filter. (5%)

```
In [31]: img = cv2.imread(imgPath, 0)

         laplacian = cv2.Laplacian(img, cv2.CV_64F)
         sobelx_1 = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)
         sobely_1 = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)
         sobelx_2 = cv2.Sobel(img, cv2.CV_64F, 2, 0, ksize=5)
         sobely_2 = cv2.Sobel(img, cv2.CV_64F, 0, 2, ksize=5)
         plt.figure(figsize=(8, 5))
         showImg(3, 2, 'Original', img, 1)
         showImg(3, 2, 'Laplacian', laplacian, 2)
         showImg(3, 2, '1st derivative Sobel X', sobelx_1, 3)
         showImg(3, 2, '1st derivative Sobel Y', sobely_1, 4)
         showImg(3, 2, '2nd derivative Sobel X', sobelx_2, 5)
         showImg(3, 2, '2nd derivative Sobel Y', sobely_2, 6)

         plt.show()
```



0.4.1 PART-2: CONVOLUTION

Following below to generate a filter bank of 48 image filters, each with size 32 Œ 32 pixels: - 4 Gaussian filters with = {1, 2, 2, 22}. - 8 LOG (Laplacian of Gaussian) filters with = {2, 2, 22, 4, 32, 6, 62, 12}. - 18 x-directional first derivation of Gaussian filters with - = and . = 3 in three different scales = {1, 2, 2} and six rotation orientations = {0, 1 , 1 , 1 , 41 , 51}. 23432 - 18 x-directional second derivation of Gaussian filters with - = and . = 3 in three different scales = {1, 2, 2} and six rotation orientations = {0,1 ,1 ,1 ,41 ,51}. 23432

A. Display the 48 image filters in the report. (5%)

```
In [4]: # generate Gaussian Kernel
        def gGaussianKernel(sig):
            getGakernel = cv2.getGaussianKernel(ksize=32, sigma=sig)
            kernel = getGakernel * getGakernel.T
            return kernel
```

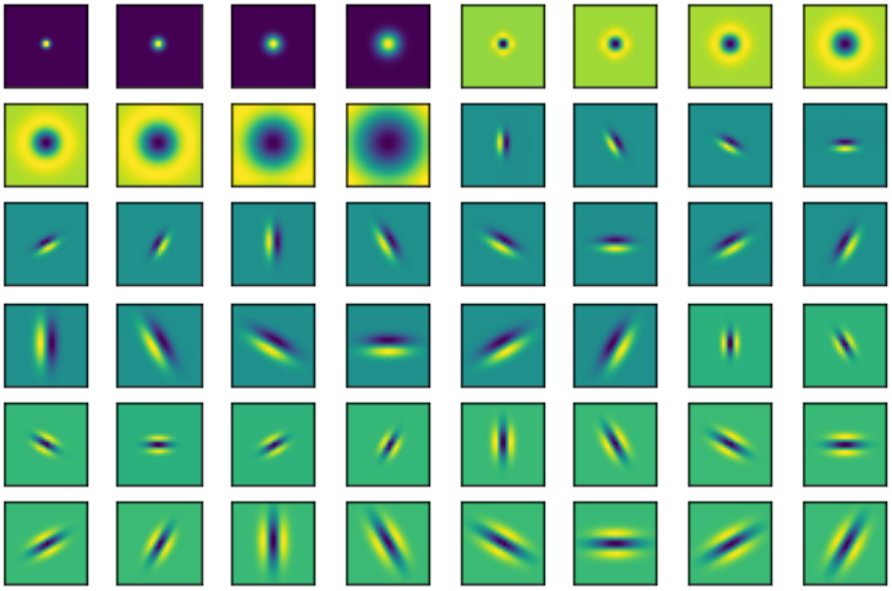
dst = cv2.Sobel(src, ddepth, dx, dy[, dst[, ksize[, scale[, delta[, borderType]]]]])

```
In [5]: # generate LOG Kernel
        def gLOGKernel(sig):
            kernel = gGaussianKernel(sig)
```

```
            d1x = np.gradient(kernel, axis=0)
            d2x = np.gradient(d1x, axis=0)
            d1y = np.gradient(kernel, axis=1)
            d2y = np.gradient(d1y, axis=1)
            dst = d2x + d2y
            return dst

In [6]: # generate rotated Kernel
        def gRotation(kernel, angle):
            M = cv2.getRotationMatrix2D((16, 16), angle, 1)
            dst = cv2.warpAffine(kernel, M, (32, 32))
            return dst

In [7]: # generate derivation Kernel
        def gDeriKernel(xsig, ysig, angle, derivation):
            getxGakernel = cv2.getGaussianKernel(ksize=32, sigma=xsig)
            getyGakernel = cv2.getGaussianKernel(ksize=32, sigma=ysig)
            kernel = getyGakernel * getxGakernel.T
            sobel = np.gradient(kernel, axis=1)
            if derivation == 2:
                sobel = np.gradient(sobel, axis=1)
            dst = gRotation(sobel, angle)
            return dst

In [253]: kernel_data = []
          G_sigs = [1, math.sqrt(2), 2, 2 * math.sqrt(2)]
          i = 0
          for sig in G_sigs:
              i = i + 1
              result = gGaussianKernel(sig)
              kernel_data.append(result)
              showImg1(6, 8, '', result, i)

          LOG_sigs = [np.sqrt(2), 2, np.sqrt(8), 4, np.sqrt(18), 6, np.sqrt(72), 12]
          for sig in LOG_sigs:
              i = i + 1
              result = gLOGKernel(sig)
              kernel_data.append(result)
              showImg1(6, 8, '', result, i)

          x = [1, math.sqrt(2), 2]
          y = [3, 3 * math.sqrt(2), 6]
          pi = 180
          angles = [0, pi/6, pi/3, pi/2, 2*pi/3, 5*pi/6]

          for (xsig, ysig) in zip(x, y):
              for angle in angles:
                  i = i + 1
```

```
                result = gDeriKernel(xsig, ysig, angle, 1)
                kernel_data.append(result)
                showImg1(6, 8, '', result, i)

        for (xsig, ysig) in zip(x, y):
            for angle in angles:
                i = i + 1
                result = gDeriKernel(xsig, ysig, angle, 2)
                kernel_data.append(result)
                showImg1(6, 8, '', result, i)
        print len(kernel_data)
        plt.savefig("Assignment1_data/result/48filter.png",bbox_inches = 'tight')
        plt.show()
```

```
48
```



B. Display the 48 image responses of the images "leapord.jpg" and "panda.jpg" after performing convolution with the filter bank. (5%)

```
In [2]: #image convolute with 48 filters, and display the result by figures
        def convolution(img, row, col):
            grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            i = 0
            results = []
            for fil in kernel_data:
                i = i + 1
```

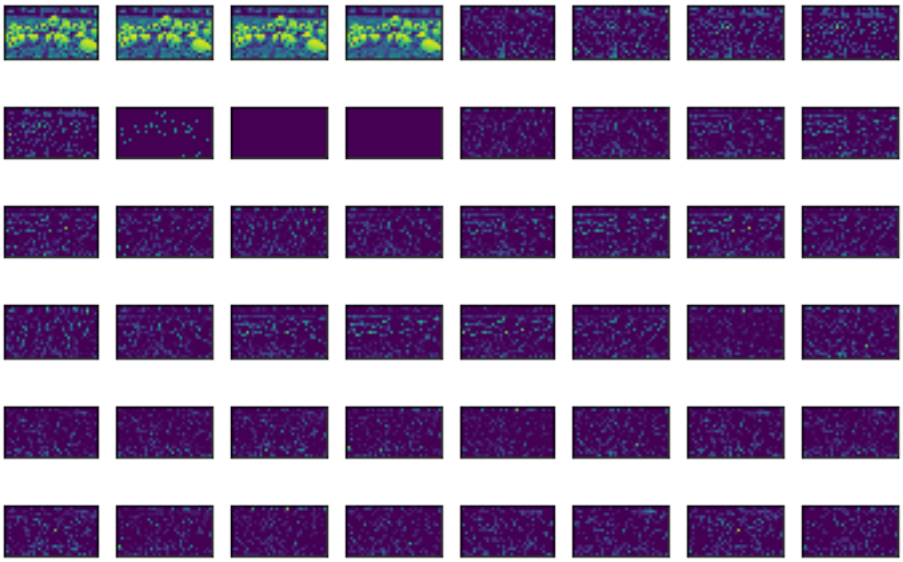
```
            res = cv2.filter2D(grayImg, -1, fil)
            results.append(res)
            showImg(row, col, '', res, i)
        return results
```

In [256]:
```
def convolution1(img, row, col):
    i = 0
    results = []
    for fil in kernel_data:
        i = i + 1
        res = cv2.filter2D(img, -1, fil/1024)
        results.append(res)
        showImg(row, col, '', res, i)
    return results
```

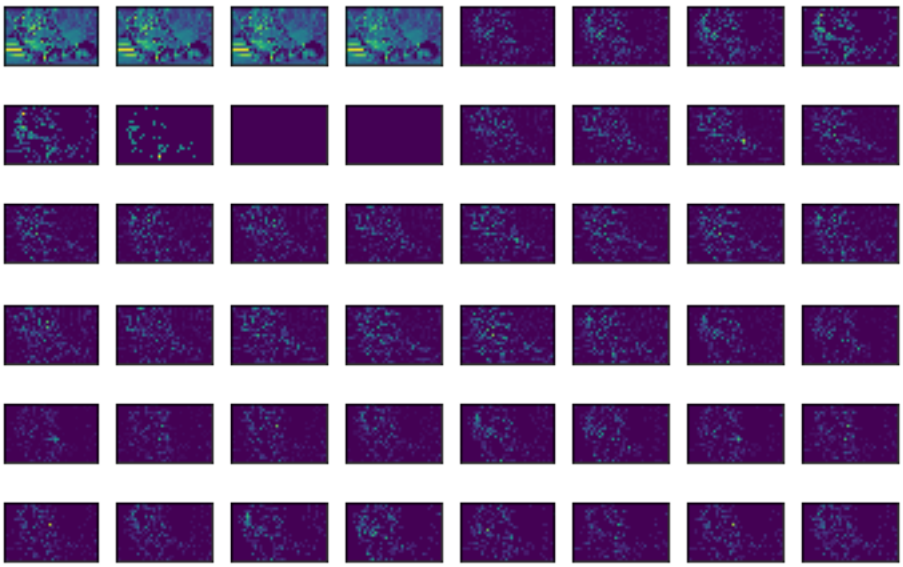
In [3]:
```
#image convolute with 48 filters but not to show them
def convolution_no_print(img):
    grayImg = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    results = []
    for fil in kernel_data:
        res = cv2.filter2D(grayImg, -1, fil)
        results.append(res)
    return results
```

In [4]:
```
#Calculate the mean and var
def meanAndstddv(kernels):
    verix = []
    i = 0
    for res in kernels:
        mean = np.mean(res)
        stddv = np.var(res)
        verix.insert(i, mean)
        verix.insert(i + 48, stddv)
        i = i + 1
    return verix
```

In [259]:
```
img = cv2.imread("Assignment1_data/panda.jpg")
pandaRes = convolution(img, 6, 8)
plt.savefig("Assignment1_data/result/pandafilter.png",pad_inches = 0,bbox_inches = 't
plt.show()
```



In [260]:
```
img = plt.imread("Assignment1_data/leopard.jpg")
leopardRes = convolution(img, 6, 8)
plt.savefig("Assignment1_data/result/leopardfilter.png",pad_inches = 0,bbox_inches =
plt.show()
```



0.4.2 C. Compute the mean and variance of each image response to form a vector of length 96 elements. Write down the filter that gives the largest value of mean and the filter that gives the largest value of variance for "leapord.jpg" and "panda.jpg" in the report.

(5%)

```
In [261]: pandaVer = meanAndstddv(pandaRes)
          leopardVer = meanAndstddv(leopardRes)
```

The filters that gives the largest value of mean and the filter that gives the largest value of variance for these two images are both the first filter which is gaussian filter with sigma = 1.

```
In [262]: pandaVerIndex = pandaVer.index(max(pandaVer[:48]))
          print pandaVerIndex+1
          pandaVerIndex = pandaVer.index(max(pandaVer[47:95]))
          print pandaVerIndex-47
          leopardVerIndex = leopardVer.index(max(leopardVer[:48]))
          print leopardVerIndex+1
          leopardVerIndex = leopardVer.index(max(leopardVer[47:95]))
          print leopardVerIndex-47
```

```
2
1
1
1
```

0.5 PART-3: IMAGE RANKING

0.5.1 You are given a collection of 2,000 images and 5 query images (in the canvas). You need to extract visual features from these images by performing convolution with the 48 filters Your task is: For each query, retrieve the five most similar images from the collection of 2,000 images. Show the five most similar images of each query in the report.

```
In [137]: dataImages = []
          dataBeforeImg = []
          queryBeforeImg = []
          path_list = os.listdir("Assignment1_data/data")
          path_list.sort()
          i = 0
          for filename in path_list:
              img = cv2.imread('Assignment1_data/data/%s' %filename)
              dataBeforeImg.append(cv2.imread('Assignment1_data/data/%s' %filename))
              dataImages.append(cv2.resize(img, (200, 200)))
              i = i + 1
          queryImages = []
          path_list = os.listdir("Assignment1_data/query")
          path_list.sort()
          for filename in path_list:
```

```
            img = cv2.imread('Assignment1_data/query/%s' %filename)
            queryBeforeImg.append(cv2.imread('Assignment1_data/query/%s' %filename))
            queryImages.append(cv2.resize(img, (200, 200)))
```

```
In [139]: path_list
```

```
Out[139]: ['0000.jpg', '0001.jpg', '0002.jpg', '0003.jpg', '0004.jpg']
```

```
In [14]: dataImagesKernels = []
         for kernel in dataImages:
             dataImagesKernels.append(convolution_no_print(kernel))
```

```
In [15]: queryImagesKernels = []
         for kernel in queryImages:
             queryImagesKernels.append(convolution_no_print(kernel))
```

```
In [16]: from scipy.spatial import distance as dist
```

```
In [17]: euclidean_res = []
         for queryImagesKernel in queryImagesKernels:
             results = []
             for dataImagesKernel in dataImagesKernels:
                 res = np.sqrt(np.sum([(a - b) ** 2 for a, b in zip(dataImagesKernel, queryImag
                 results.append(res)
             euclidean_res.append(results)
         print len(euclidean_res)
```

```
In [18]: len(euclidean_res[0])
```

```
Out[18]: 2000
```

```
In [52]: lists = []
         for i in range(0, len(euclidean_res)):
             querylist = euclidean_res[i]
             lists.append(map(querylist.index, heapq.nsmallest(5, querylist)))
         print lists
```

```
[[319, 1758, 1165, 655, 270], [285, 1994, 535, 519, 8], [535, 285, 1579, 267, 1719], [342, 159
```

```
In [103]: for index in range(0, len(lists)):
              i = 1
              showImg(1, 6, '', queryBeforeImg[index], i)
              for l in lists[index]:
                  i = i + 1
                  showImg(1, 6, '', dataBeforeImg[l], i)
              plt.savefig("Assignment1_data/result/filterEuclideanQ%d.png"%index,pad_inches = 
          plt.savefig("Assignment1_data/result/filterEuclidean.png")
          plt.show()
```

0.6 PART-4: METHOD COMPARISON

0.6.1 A. Implement any two feature extraction methods that you know (e.g., color histogram, LBP, SIFT, deep learning) to extract features for 2,000 images in Part-3. Show the five most similar images of each query for each method. (10%)

Deep Learning:

```
In [19]: import sys
         import matplotlib.pylab as plt
         sys.path.append("..")  # Adds higher directory to python modules path.
         from img_to_vec import Img2Vec
         from PIL import Image
         from sklearn.metrics.pairwise import cosine_similarity


         input_path = 'Assignment1_data/query'
         input1_path = 'Assignment1_data/data'

         img2vec = Img2Vec()

In [20]: # For each test image, we store the filename and vector as key, value in a dictionary
         pics = {}
         for file in os.listdir(input_path):
             img = Image.open('Assignment1_data/query/%s'%file)
             vec = img2vec.get_vec(img)
             pics[file] = vec

         datapics = {}
         for file in os.listdir(input1_path):
             img = Image.open('Assignment1_data/data/%s'%file)
             vec = img2vec.get_vec(img)
             datapics[file] = vec

In [104]: import collections
```

```
In [127]: datapics = collections.OrderedDict(sorted(datapics.items()))
          pics = collections.OrderedDict(sorted(pics.items()))

In [128]: datapics.keys()
          pics.keys()

Out[128]: ['0000.jpg', '0001.jpg', '0002.jpg', '0003.jpg', '0004.jpg']

In [129]: dp_result = []
          j = 1
          #f = open("/Users/ponta/PycharmProjects/vision_1/Assignment1_data/result/result.txt"
          for query in list(pics.keys()):
              results = []
              #f.write('Q%d:' %j)
              i = 1
              showImg(1, 6,'', cv2.imread('Assignment1_data/query/%s'%query), i)
              sims = {}
              for data in list(datapics.keys()):
                  res = cosine_similarity(pics[query].reshape((1, -1)), datapics[data].reshape
                  sims[data] = res
                  results.append(res)
              dp_result.append(results)
              d_view = [(v, k) for k, v in sims.items()]
              #dp_result = [(v, k) for k, v in sims.items()]
              d_view.sort(reverse=True)

              #for v, k in d_view:
                  #f.write(k[:4])
                 # f.write(' ')
              #f.write('\n')
              for v, k in d_view[:5]:
                  i = i + 1
                  showImg(1, 6,'', cv2.imread('Assignment1_data/data/%s'%k), i)
                  #print(v, k)
                  plt.savefig("Assignment1_data/result/deeplearning1Q%d.png"%j,pad_inches = 0,
              #i = i + 1
              j = j + 1
          plt.savefig("Assignment1_data/result/deeplearning.png")
          plt.show()
```

Color histogram:

```
In [160]: import argparse
          import glob
          from scipy.spatial.distance import euclidean

In [177]: histdes = []
          for data in dataBeforeImg:
              histdes.append(describe(data))

In [179]: queryhistdes = []
          for data in queryBeforeImg:
              queryhistdes.append(describe(data))

In [207]: def feature_extraction(dataset,addstr):
              features = {}
              #descriptor = RGBHistogram(bins=[8, 8, 8])

              #for filename in glob.glob(os.path.join(dataset, '*.jpg|png$')):
              for filename in dataset:
                  # e.g. places/eiffel_tower.jpg => eiffel_tower
                  img_name = os.path.basename(filename).split('.')[0]
                  #print img_name
                  dataset = addstr + filename
                  image = cv2.imread(dataset)
                  feature = describe(image)
                  # key - image name, value - feature vector
                  features[img_name] = feature
              return features

In [208]: path_list = os.listdir("Assignment1_data/data")
          path_list.sort()
          addstr = 'Assignment1_data/data/'
          datafeature = feature_extraction(path_list,addstr)

In [209]: path_list = os.listdir("Assignment1_data/query")
          path_list.sort()
          addstr = 'Assignment1_data/query/'
          queryfeature = feature_extraction(path_list,addstr)

In [230]: def search():
                  results = {}
                  for queryname, querydes in queryfeature.items():
                      result = {}
                      for dataname, datades in datafeature.items():
                          dist = euclidean(querydes, datades)
                          result[dataname] = dist
                      result = sorted([(d, n) for n, d in result.items()])
                      results[queryname] = result
                  return results
```

```
In [231]: hisresults = search()
```

```
In [238]: for queryname, diswithdata in hisresults.items():
              #print query
              i = 1
              showImg(1, 6, '', cv2.imread('Assignment1_data/query/%s.jpg'%queryname), i)
              for dis, name in diswithdata[:5]:
                  i = i + 1
                  print name
                  showImg(1, 6, '', cv2.imread('Assignment1_data/data/%s.jpg'%name), i)
                  plt.savefig("Assignment1_data/result/hisQ%s.png"%queryname, pad_inches=0,
                              bbox_inches='tight')
```

```
0967
0759
0127
1589
1516
0029
0364
0038
0702
1420
1308
1439
1230
0632
1393
1337
0923
0868
1883
0967
0721
0456
0601
0908
1264
```

0.6.2 C. Propose a method to fuse (or combine) the results in Part-3 and Part-4. Show the five most similar images of each query.

```
In [148]: def combiner():
              data = {}
              n = 0
              for (res1,res2) in zip(euclidean_res, dp_result):
```

```
                    sortRes1 = sorted(res1)
                    sortRes2 = sorted(res2, reverse=True)
                    for index in range(0,len(res2)):
                        sum = 0.01 * sortRes1.index(res1[index]) + sortRes2.index(res2[index])
                        data[index] = sum
                    data_view = [(v, k) for k, v in data.items()]
                    data_view.sort()
                    m = 1
                    showImg(1, 6, '', queryBeforeImg[n], m)
                    for v, k in data_view[:5]:
                        m = m + 1
                        print m
                        showImg(1, 6, '', dataBeforeImg[k], m)
                    plt.savefig("Assignment1_data/result/CombineQ%d.png"%n, pad_inches=0,
                                bbox_inches='tight')
                    n = n + 1
```

```
In [149]: len(dp_result[0])
```

```
Out[149]: 2000
```

```
In [150]: combiner()
```

```
2
3
4
5
6
2
3
4
5
6
2
3
4
5
6
2
3
4
5
6
2
3
4
5
```

6

In [ ]: