HW2_part1_get_des

February 19, 2019

0.1 HW2 Solution Part 1

step1: get keypoints for each images
step2: get patches for each keypoints
step3: get descriptions for each keypoints (by forward pass patches to the network)
step4: caculate the similarity matrices
step5: get topK similar images for each query
step6: draw recall vs precision curves

```
In [1]: import os
        os.environ["CUDA_VISIBLE_DEVICES"] = "1"

In [2]: # load packages
        from __future__ import division, print_function
        import glob
        import os
        import cv2
        import PIL
        import random
        import numpy as np
        # import pandas as pd
        import matplotlib.pyplot as plt
        import matplotlib.image as mpimg
        import torch
        import torch.nn.init
        import torch.nn as nn
        import torch.optim as optim
        import torch.backends.cudnn as cudnn
        import torch.nn.functional as F
        import torchvision.datasets as dset
        import torchvision.transforms as transforms
        from tqdm import tqdm
        from torch.autograd import Variable
        from copy import deepcopy, copy
        from Utils import cv2_scale36, cv2_scale, np_reshape, np_reshape64
        from scipy.optimize import linear_sum_assignment
```

```
In [3]: # parameters setting
        query_path = "image_retrieval/query/"
        image_path = "image_retrieval/images/"
        query_num = 35
        image_num = 140
        kps_num = 30
        patch_size = 32
        patches = torch.zeros(query_num+image_num, kps_num,1, 32, 32)
```

```
In [4]: # Initiate SURF detector
        surf = cv2.xfeatures2d.SURF_create(30)
```

```
In [5]: def getPatches(kps, img, size=32, num=500):
            res = torch.zeros(num, 1, size, size)
            if type(img) is np.ndarray:
                img = torch.from_numpy(img)
            h, w = img.shape      # note: for image, the x direction is the verticle, y-directio
            for i in range(num):
                cx, cy = kps[i]
                cx, cy = int(cx), int(cy)
                dd = int(size/2)
                xmin, xmax = max(0, cx - dd), min(w, cx + dd ) - 1
                ymin, ymax = max(0, cy - dd), min(h, cy + dd ) - 1

                xmin_res, xmax_res = dd - min(dd,cx), dd + min(dd, w - cx)-1
                ymin_res, ymax_res = dd - min(dd,cy), dd + min(dd, h - cy)-1

                res[i, 0, ymin_res: ymax_res, xmin_res: xmax_res] = img[ymin: ymax, xmin: xmax]
            return res
```

```
In [6]: # tensor for query/image patches
        for idx in range(query_num+image_num):
            if idx < query_num:
                img_dir = os.path.join(query_path,"q{}.JPG".format(idx+1))
            else:
                img_dir = os.path.join(image_path,"{}.JPG".format(idx+1-query_num))
            image = cv2.imread(img_dir)
            img= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
            ## find the keypoints and descriptors with SURF
            kps, des = surf.detectAndCompute(img, None)
            ## find the keypoints and descriptors with SIFT
            keypoints_img = [kps[a].pt for a in range(kps_num)]
            patches[idx] = getPatches(keypoints_img, img, size=patch_size, num=kps_num)
```

```
In [7]: # import network, , load pretrained weights, and turn on testing mode
        from descriptor_CNN3 import DesNet
        model = DesNet()
        model.cuda()
        model.eval()
        trained_weights = torch.load("checkpoint.pth")
        model.load_state_dict(trained_weights["state_dict"])
```

```
In [8]: print(patches.shape)
```

```
torch.Size([175, 30, 1, 32, 32])
```

```
In [9]: # get descriptions
        with torch.no_grad(): # only 3612MiB, without this line, it will be 9132MiB
            des1 = model(patches[:query_num].view(-1, 1, 32,32).cuda()).view(-1, 30, 128)
            des2 = model(patches[query_num:].view(-1, 1, 32,32).cuda()).view(-1, 30, 128)
```

```
In [10]: print(des1.shape, des2.shape)
         # save des
         des_dir = "des.pt"
         torch.save([des1, des2], des_dir)
```

```
torch.Size([35, 30, 128]) torch.Size([140, 30, 128])
```