**Practical Assignment**

**Objective: - Human Counting**

A camera can be used to monitor and count the number of people present in the room, building, street, etc.

First, you need to detect people and then we count their occurrence. It is useful to control the crowd.

**Dataset Link: -**

Use anyone of your choice.

**Task: -** Create a Web Application using FASTAPI. Use the end user should be able to upload an image or video and get results with the count of persons.

**Deployment: -** Any Free Platform(Try to look out for free options.)

**Assignment Submission: -** Only submit the hosted app link. OR GitHub Link

|  |
| --- |
| # importing libraries |
|  | import h5py |
|  | import scipy.io as io |
|  | import PIL.Image as Image |
|  | import numpy as np |
|  | import os |
|  | import glob |
|  | from matplotlib import pyplot as plt |
|  | from scipy.ndimage.filters import gaussian\_filter |
|  | import scipy |
|  | import json |
|  | from matplotlib import cm as CM |
|  | from image import \* |
|  | from model import CSRNet |
|  | import torch |
|  | from tqdm import tqdm |
|  | %matplotlib inline |

|  |
| --- |
| # function to create density maps for images |
|  | def gaussian\_filter\_density(gt): |
|  | print (gt.shape) |
|  | density = np.zeros(gt.shape, dtype=np.float32) |
|  | gt\_count = np.count\_nonzero(gt) |
|  | if gt\_count == 0: |
|  | return density |
|  |  |
|  | pts = np.array(list(zip(np.nonzero(gt)[1], np.nonzero(gt)[0]))) |
|  | leafsize = 2048 |
|  | # build kdtree |
|  | tree = scipy.spatial.KDTree(pts.copy(), leafsize=leafsize) |
|  | # query kdtree |
|  | distances, locations = tree.query(pts, k=4) |
|  |  |
|  | print ('generate density...') |
|  | for i, pt in enumerate(pts): |
|  | pt2d = np.zeros(gt.shape, dtype=np.float32) |
|  | pt2d[pt[1],pt[0]] = 1. |
|  | if gt\_count > 1: |
|  | sigma = (distances[i][1]+distances[i][2]+distances[i][3])\*0.1 |
|  | else: |
|  | sigma = np.average(np.array(gt.shape))/2./2. #case: 1 point |
|  | density += scipy.ndimage.filters.gaussian\_filter(pt2d, sigma, mode='constant') |
|  | print ('done.') |
|  | return density |

|  |
| --- |
| part\_A\_train = os.path.join(root,'part\_A/train\_data','images') |
|  | part\_A\_test = os.path.join(root,'part\_A/test\_data','images') |
|  | part\_B\_train = os.path.join(root,'part\_B/train\_data','images') |
|  | part\_B\_test = os.path.join(root,'part\_B/test\_data','images') |
|  | path\_sets = [part\_A\_train,part\_A\_test] |

|  |
| --- |
| img\_paths = [] |
|  | for path in path\_sets: |
|  | for img\_path in glob.glob(os.path.join(path, '\*.jpg')): |
|  | img\_paths.append(img\_path) |

|  |
| --- |
| for img\_path in img\_paths: |
|  | print (img\_path) |
|  | mat = io.loadmat(img\_path.replace('.jpg','.mat').replace('images','ground-truth').replace('IMG\_','GT\_IMG\_')) |
|  | img= plt.imread(img\_path) |
|  | k = np.zeros((img.shape[0],img.shape[1])) |
|  | gt = mat["image\_info"][0,0][0,0][0] |
|  | for i in range(0,len(gt)): |
|  | if int(gt[i][1])<img.shape[0] and int(gt[i][0])<img.shape[1]: |
|  | k[int(gt[i][1]),int(gt[i][0])]=1 |
|  | k = gaussian\_filter\_density(k) |
|  | with h5py.File(img\_path.replace('.jpg','.h5').replace('images','ground-truth'), 'w') as hf: |
|  | hf['density'] = k |

|  |
| --- |
| path\_sets = [part\_B\_train,part\_B\_test] |
|  | img\_paths = [] |
|  | for path in path\_sets: |
|  | for img\_path in glob.glob(os.path.join(path, '\*.jpg')): |
|  | img\_paths.append(img\_path) |
|  |  |
|  | # creating density map for part\_b images |
|  |  |
|  | for img\_path in img\_paths: |
|  | print (img\_path) |
|  | mat = io.loadmat(img\_path.replace('.jpg','.mat').replace('images','ground-truth').replace('IMG\_','GT\_IMG\_')) |
|  | img= plt.imread(img\_path) |
|  | k = np.zeros((img.shape[0],img.shape[1])) |
|  | gt = mat["image\_info"][0,0][0,0][0] |
|  | for i in range(0,len(gt)): |
|  | if int(gt[i][1])<img.shape[0] and int(gt[i][0])<img.shape[1]: |
|  | k[int(gt[i][1]),int(gt[i][0])]=1 |
|  | k = gaussian\_filter\_density(k) |
|  | with h5py.File(img\_path.replace('.jpg','.h5').replace('images','ground-truth'), 'w') as hf: |
|  | hf['density'] = |

|  |
| --- |
| #importing libraries |
|  | import h5py |
|  | import scipy.io as io |
|  | import PIL.Image as Image |
|  | import numpy as np |
|  | import os |
|  | import glob |
|  | from matplotlib import pyplot as plt |
|  | from scipy.ndimage.filters import gaussian\_filter |
|  | import scipy |
|  | import json |
|  | import torchvision.transforms.functional as F |
|  | from matplotlib import cm as CM |
|  | from image import \* |
|  | from model import CSRNet |
|  | import torch |
|  | %matplotlib inline |