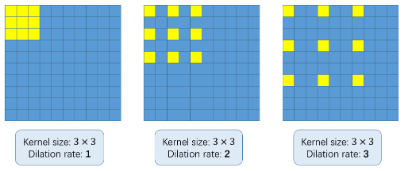
**GREENHOLICS INTEGRATED SOLUTION (PART-2)**

**CROWD COUNTING :**

**Understanding the Architecture and Training Method**

Our algorithm uses VGG-16 as the front end because of its strong transfer learning ability. The output size from VGG is ⅛th of the original input size. Cv2 also uses dilated convolutional layers in the back end.

But what in the world are dilated convolutions? It’s a fair question to ask. Consider the below image:



The basic concept of using dilated convolutions is to enlarge the kernel without increasing the parameters. So, if the dilation rate is 1, we take the kernel and convolve it on the entire image. Whereas, if we increase the dilation rate to 2, the kernel extends as shown in the above image (follow the labels below each image). It can be an alternative to pooling layers.

AS SOON AS our edge end algorithm will detect a gathering, the whole pictures from the video feed will be cropped saved, and sent to the authourity as log.

We will implement Algorithm on the ShanghaiTech dataset. This contains 1198 annotated images of a combined total of 330,165 people. **You can download the dataset from here**.

Please install [CUDA](https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html) and [PyTorch](https://github.com/pytorch/pytorch" \l "installation) before you proceed further. These are the backbone behind the code we’ll be using below.

Now, move the dataset into the repository you cloned above and unzip it.

In the dataset above change the folders Ground\_truth to Ground-truth in both part\_A\_final and part\_B\_final.

Now, open make\_dataset.ipynb in Greenholics-Ct Master folder, execute it.

**1.Train Dataset**

**If directly want to test it also a readymade trainded sytem abaialbe as best\_train.tar.gz**

**2.Test Dataset**

**AFTER RUNNING THE WHOLE SEQUENCE**

Check the MAE (Mean Absolute Error) on test images to evaluate our model:

|  |  |
| --- | --- |
|  | mae = 0 |
|  | for i in tqdm(range(len(img\_paths))): |
|  | img = transform(Image.open(img\_paths[i]).convert('RGB')).cuda() |
|  | gt\_file = h5py.File(img\_paths[i].replace('.jpg','.h5').replace('images','ground-truth'),'r') |
|  | groundtruth = np.asarray(gt\_file['density']) |
|  | output = model(img.unsqueeze(0)) |
|  | mae += abs(output.detach().cpu().sum().numpy()-np.sum(groundtruth)) |
|  | print (mae/len(img\_paths)) |

**We got an MAE value of 75.69 which is pretty good.** Now let’s check the predictions on a single image:

Now use the val.ipynb to execute;

|  |  |
| --- | --- |
|  | from matplotlib import cm as c |
|  | img = transform(Image.open('part\_A/test\_data/images/IMG\_100.jpg').convert('RGB')).cuda() |
|  |  |
|  | output = model(img.unsqueeze(0)) |
|  | print("Predicted Count : ",int(output.detach().cpu().sum().numpy())) |
|  | temp = np.asarray(output.detach().cpu().reshape(output.detach().cpu().shape[2],output.detach().cpu().shape[3])) |
|  | plt.imshow(temp,cmap = c.jet) |
|  | plt.show() |
|  | temp = h5py.File('part\_A/test\_data/ground-truth/IMG\_100.h5', 'r') |
|  | temp\_1 = np.asarray(temp['density']) |
|  | plt.imshow(temp\_1,cmap = c.jet) |
|  | print("Original Count : ",int(np.sum(temp\_1)) + 1) |
|  | plt.show() |
|  | print("Original Image") |
|  | plt.imshow(plt.imread('part\_A/test\_data/images/IMG\_100.jpg')) |
|  | plt.show()  CSRNet output |

Wow, the original count was 382 and our model estimated there were 384 people in the image. That is a very impressive performance!