ALGORITHM COMPUTER VISION DEEP LEARNING IMAGE IMAGE ANALYSIS PROJECT PYTHON TECHNIQUE UNSTRUCTURED DATA UNSUPERVISED

It's a Record-Breaking Crowd! A Must-Read Tutorial to Build your First Crowd Counting Model using Deep Learning

PULKIT SHARMA, FEBRUARY 18, 2019

Introduction

Artificial Intelligence and Machine Learning is going to be our biggest helper in coming decade!

Today morning, I was reading an article which reported that an AI system won against 20 lawyers and the lawyers were actually happy that AI can take care of repetitive part of their roles and help them work on complex topics. These lawyers were happy that AI will enable them to have more fulfilling roles.

Today, I will be sharing a similar example – How to count number of people in crowd using **Deep Learning and Computer Vision**? But, before we do that – let us develop a sense of how easy the life is for a Crowd Counting Scientist.

Act like a Crowd Counting Scientist

Let's start!

Can you help me count / estimate number of people in this picture attending this event?

Ok – how about this one?

Source: ShanghaiTech Dataset

You get the hang of it. By end of this tutorial, we will create an algorithm for Crowd Counting with an amazing accuracy (compared to humans like you and me). Will you use such an assistant?

P.S. *This article assumes that you have a basic knowledge of how convolutional neural networks (CNNs) work. You can refer to the below post to learn about this topic before you proceed further:*

- *A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch*

Table of Contents

This article is highly inspired by the paper – CSRNet : Dilated Convolutional Neural Networks for Understanding the Highly Congested Scenes.

What is Crowd Counting?

Crowd Counting is a technique to count or estimate the number of people in an image. Take a moment to analyze the below image:

Source: ShanghaiTech Dataset

Can you give me an approximate number of how many people are in the frame? Yes, including the ones present way in the background. The most direct method is to manually count each person but does that make practical sense? It's nearly impossible when the crowd is this big!

Crowd scientists (yes, that's a real job title!) count the number of people in certain parts of an image and then extrapolate to come up with an estimate. More commonly, we have had to rely on crude metrics to estimate this number for decades.

Surely there must be a better, more exact approach?

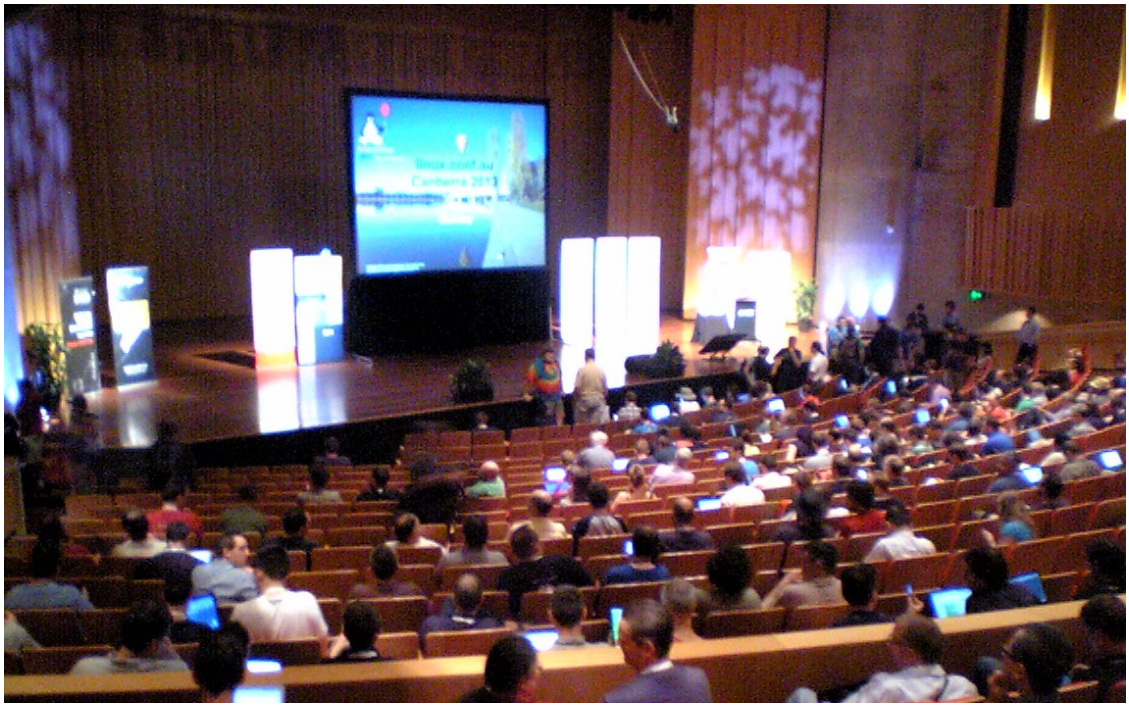
Yes, there is!

While we don't yet have algorithms that can give us the EXACT number, most computer vision techniques can produce impressively precise estimates. Let's first understand why crowd counting is important before diving into the algorithm behind it.

Why is Crowd Counting useful?

Let's understand the usefulness of crowd counting using an example. Picture this – your company just finished hosting a huge data science conference. Plenty of different sessions took place during the event.

You are asked to analyze and estimate the number of people who attended each session. This will help your team understand what kind of sessions attracted the biggest crowds (and which ones failed in that regard). This will shape next year's conference, so it's an important task!



There were hundreds of people at the event – counting them manually will take days! That's where your data scientist skills kick in. You managed to get photos of the crowd from each session and build a computer vision model to do the rest!

There are plenty of other scenarios where crowd counting algorithms are changing the way industries work:

- Counting the number of people attending a sporting event
- Estimating how many people attended an inauguration or a march (political rallies, perhaps)
- Monitoring of high-traffic areas
- Helping with staffing allocation and resource allotment

Can you come up with some other use cases? Let me know in the comments section below! We can connect and try to figure out how we can use crowd counting techniques in your scenario.

Understanding the Different Computer Vision Techniques for Crowd Counting

Broadly speaking, there are currently four methods we can use for counting the number of people in a crowd:

1. Detection-based methods

Here, we use a moving window-like detector to identify people in an image and count how many there are. The methods used for detection require well trained classifiers that can extract low-level features. **Although these methods work well for detecting faces, they do not perform well on crowded images as most of the target objects are not clearly visible.**

2. Regression-based methods

We were unable to extract low level features using the above approach. Regression-based methods come up trumps here. **We first crop patches from the image and then, for each patch, extract the low level features.**

3. Density estimation-based methods

We first create a density map for the objects. Then, the algorithm learn a linear mapping between the extracted features and their object density maps. We can also use random forest regression to learn non-linear mapping.

4. CNN-based methods

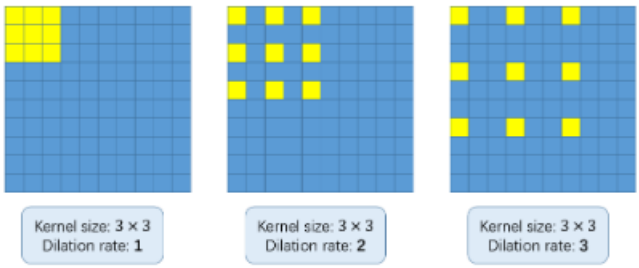
Ah, good old reliable convolutional neural networks (CNNs). Instead of looking at the patches of an image, we build an end-to-end regression method using CNNs. This takes the entire image as input and directly generates the crowd count. CNNs work really well with regression or classification tasks, and they have also proved their worth in generating density maps.

CSRNet, a technique we will implement in this article, deploys a deeper CNN for capturing high-level features and generating high-quality density maps without expanding the network complexity. Let's understand what CSRNet is before jumping to the coding section.

Understanding the Architecture and Training Method of CSRNet

CSRNet 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. CSRNet 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.

Underlying Mathematics (Recommended, but optional)

I'm going to take a moment to explain how the mathematics work. *Note that this isn't mandatory to implement the algorithm in Python, but I highly recommend learning the underlying idea.* This will come in handy when you need to tweak or modify your model.

Suppose we have an input x(m,n), a filter w(i,j), and the dilation rate r. The output y(m,n) will be:

$$y(m,n) = \sum_{i=1}^{M} \sum_{j=1}^{N} x(m + r \times i, n + r \times j) w(i,j)$$

We can generalize this equation using a (k*k) kernel with a dilation rate r. The kernel enlarges to:

([k + (k-1)*(r-1)] * [k + (k-1)*(r-1)])

So the ground truth has been generated for each image. **Each person's head in a given image is blurred using a Gaussian kernel.** All the images are cropped into 9 patches, and the size of each patch is ¼th of the original size of the image. With me so far?

The first 4 patches are divided into 4 quarters and the other 5 patches are randomly cropped. Finally, the mirror of each patch is taken to double the training set.

That, in a nutshell, are the architecture details behind CSRNet. Next, we'll look at its training details, including the evaluation metric used.

Stochastic Gradient Descent is used to train the CSRNet as an end-to-end structure. During training, the fixed learning rate is set to *1e-6*. The loss function is taken to be the Euclidean distance in order to measure the difference between the ground truth and estimated density map. This is represented as:

$$L(\Theta) = \frac{1}{2N} \sum_{i=1}^{N} \| \ Z(X_i; \Theta) - Z_i^{GT} \|_2^2$$

where N is the size of the training batch. **The evaluation metric used in CSRNet is MAE and MSE**, i.e., Mean Absolute Error and Mean Square Error. These are given by:

$$MAE = \frac{1}{N} \sum_{i=1}^{N} | \ C_i - C_i^{GT} \ |$$

$$MSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} | \ C_i - C_i^{GT} \ |^2}$$

Here, Ci is the estimated count:

$$C_i = \sum_{l=1}^{L} \sum_{w=1}^{W} z_{l,w}$$

L and W are the width of the predicted density map.

Our model will first predict the density map for a given image. The pixel value will be 0 if no person is present. A certain pre-defined value will be assigned if that pixel corresponds to a person. So, calculating the total pixel values corresponding to a person will give us the count of people in that image. Awesome, right?

And now, ladies and gentlemen, it's time to finally build our own crowd counting model!

Building your own Crowd Counting model

Ready with your notebook powered up?

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

Use the below code block to clone the CSRNet-pytorch repository. This holds the entire code for creating the dataset, training the model and validating the results:

```
git clone https://github.com/leeyeehoo/CSRNet-pytorch.git
```

Please install CUDA and PyTorch 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. We'll then need to create the ground truth values. The *make_dataset.ipynb* file is our savior. We just need to make minor changes in that notebook:

```
# 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
```

import_libraries.py hosted with ♥ by **GitHub** **view raw**

```
# 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
```

density_map.py hosted with ♥ by **GitHub** **view raw**

```
#setting the root to the Shanghai dataset you have downloaded
# change the root path as per your location of dataset
root = '/home/pulkit/CSRNet-pytorch/'
```

Now, let's generate the ground truth values for images in part_A and part_B:

```
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]
```

ground_truth.py hosted with ♥ by **GitHub** **view raw**

```
img_paths = []
for path in path_sets:
    for img_path in glob.glob(os.path.join(path, '*.jpg')):
        img_paths.append(img_path)
```

image_location.py hosted with ♥ by **GitHub** **view raw**

```
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)
```

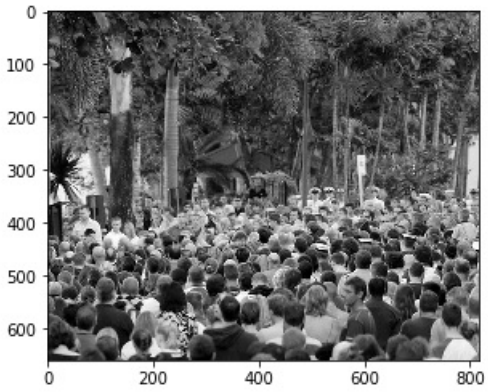
```
    with h5py.File(img_path.replace('.jpg','.h5').replace('images','ground-truth'), 'w') as hf:
            hf['density'] = k
```

create_density_map.py hosted with ♥ by **GitHub** **view raw**

Generating the density map for each image is a time taking step. So go brew a cup of coffee while the code runs.

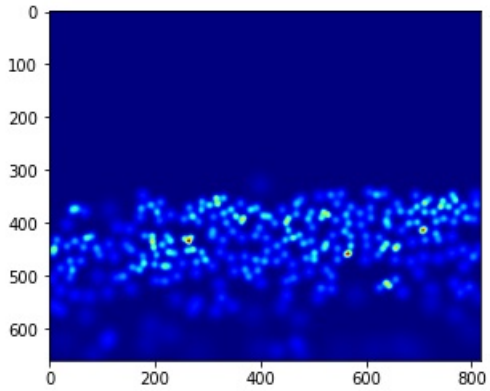
So far, we have generated the ground truth values for images in part_A. We will do the same for the part_B images. But before that, let's see a sample image and plot its ground truth heatmap:

```
plt.imshow(Image.open(img_paths[0]))
```



Things are getting interesting!

```
gt_file = h5py.File(img_paths[0].replace('.jpg','.h5').replace('images','ground-truth'),'r')
groundtruth = np.asarray(gt_file['density'])
plt.imshow(groundtruth,cmap=CM.jet)
```



Let's count how many people are present in this image:

```
np.sum(groundtruth)
```

270.32568

Similarly, we will generate values for part_B:

```
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'] = k
```

part_b_images.py hosted with ♥ by **GitHub** **view raw**

Now, we have the images as well as their corresponding ground truth values. Time to train our model!

We will use the .json files available in the cloned directory. We just have to change the location of the images in the json files. To do this, open the .json file and replace the current location with the location where your images are located.

Note that all this code is written in Python 2. Make the following changes if you're using any other Python version:

1. In model.py, change xrange in line 18 to range
2. Change line 19 in model.py with: list(self.frontend.state_dict().items())[i][1].data[:] = list(mod.state_dict().items())[i][1].data[:]
3. In image.py, replace ground_truth with ground-truth

Made the changes? Now, open a new terminal window and type the following commands:

```
cd CSRNet-pytorch
python train.py part_A_train.json part_A_val.json 0 0
```

Again, sit back because this will take some time. You can reduce the number of epochs in the *train.py* file to accelerate the process. **A cool alternate option is to download the pre-trained weights from here if you don't feel like waiting.**

Finally, let's check our model's performance on unseen data. We will use the *val.ipynb* file to validate the results. Remember to change the path to the pretrained weights and images.

```
#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
```

import_library.py hosted with ♥ by **GitHub** **view raw**

```
from torchvision import datasets, transforms
transform=transforms.Compose([
                        transforms.ToTensor(),transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                    std=[0.229, 0.224, 0.225]),
                    ])
```

transform.py hosted with ♥ by **GitHub** **view raw**

```
#defining the location of dataset
root = '/home/pulkit/CSRNet/ShanghaiTech/CSRNet-pytorch/'
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_test]
```

location.py hosted with ♥ by **GitHub** **view raw**

```
#defining the image path
img_paths = []
for path in path_sets:
    for img_path in glob.glob(os.path.join(path, '*.jpg')):
        img_paths.append(img_path)
```

```
model = CSRNet()
```

```
#defining the model
model = model.cuda()
```

```
#loading the trained weights
checkpoint = torch.load('part_A/0model_best.pth.tar')
model.load_state_dict(checkpoint['state_dict'])
```

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))
```

mae.py hosted with ♥ by **GitHub** **view raw**

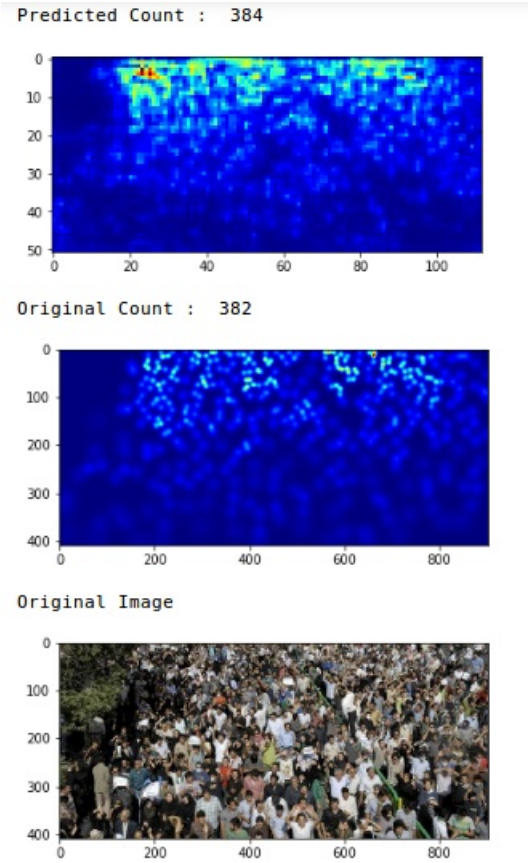
We got an MAE value of 75.69 which is pretty good. Now let's check the predictions on a single image:

```
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()
```

prediction.py hosted with ♥ by GitHub view raw



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

Congratulations on building your own crowd counting model!

End Notes

I encourage you to try out this approach on different images and share your results in the comments section below. Crowd counting has so many diverse applications and is already seeing adoption by organizations and government bodies.

It is a useful skill to add to your portfolio. Quite a number of industries will be looking for data scientists who can work with crowd counting algorithms. Learn it, experiment with it, and give yourself the gift of deep learning!

Did you find this article useful? Feel free to leave your suggestions and feedback for me below, and I'll be happy to connect with

you.

You should also check out the below resources to learn and explore the wonderful world of computer vision:

- Certified Course: Computer Vision using Deep Learning
- A Step-by-Step Introduction to the Basic Computer Vision Algorithms
- Understanding and Building your First Object Detection Model from Scratch
- Learn Object Detection using the Popular YOLO Framework

You can also read this article on Analytics Vidhya's Android APP GET IT ON Google Play

Share this:

TAGS : CNN, COMPUTER VISION, CONVOLUTIONAL NEURAL NETWORK, CROWD COUNTING, CSRNET, PYTHON

NEXT ARTICLE

Don’t Miss this Comprehensive 7 Step Process to Ace Data Science Interviews!

PREVIOUS ARTICLE

An Awesome Tutorial to Learn Outlier Detection in Python using PyOD Library

Pulkit Sharma

My research interests lies in the field of Machine Learning and Deep Learning. Possess an enthusiasm for learning new skills and technologies.

This article is quite old and you might not get a prompt response from the author. We request you to post this comment on Analytics Vidhya's **Discussion portal** to get your queries resolved

75 COMMENTS

SAURABH SINGH Reply

February 18, 2019 at 12:46 pm

Nicely explained the complex concept of Crowd counting . And its very helpful.

PULKIT SHARMA

February 18, 2019 at 2:29 pm

Reply

Thank you Saurabh!

PREETI

February 18, 2019 at 7:10 pm

Reply

Very nicely explained. Good one Pulkit

SANDESH

February 18, 2019 at 11:42 pm

Reply

Great learning through your article .

VIVEK

February 20, 2019 at 4:00 pm

Reply

The zip file with pre-trained weights is corrupt , https://drive.google.com/file/d/1KY11yLorynba14Sg7whFOfVeh2ja02wm/view?usp=sharing

Could you provide the correct file please?

“A cool alternate option is to download the pre-trained weights from here if you don’t feel like waiting.”

PULKIT SHARMA

February 20, 2019 at 6:19 pm

Reply

Hi vivek,

The link is working fine at my end. What is the error that you are getting while downloading the weights?

VIVEK

February 20, 2019 at 7:47 pm

Reply

The file downloaded from the link is not correct.
I cant un-tar it, also facing issue while loading it at Kaggle kernel.

VIVEK

February 21, 2019 at 5:21 am

Reply

Hello Pultik,

I am able to use pre-trained weights now.
I am using google colab kernel (with GPU) to process data.
Kaggle on other hand tries to process your uploaded .tar file and failed to do so, which results in; coucln’t upload file at all!

Here I have slightly modified code to work with Python 3, https://github.com/vivek-bombatkar/CSRNet-pytorch.git

Thanks a lot for sharing wonderful information!

VIVEK

February 21, 2019 at 5:28 am

Reply

*Pulkit
Apology for typo.

CÉSAR MARTÍNEZ

April 12, 2019 at 1:55 am

Reply

Hi, how could you untar the 0model….tar file?

Thanks

MOHAN

February 23, 2019 at 9:41 am

Reply

Hi,

I'm getting an issue, when trying to replicate the concept. I kept the screenshots here about the issue as well as the part of the code where it is occurring.

https://www.dropbox.com/sh/vkyqzogbhvzdj8s/AABC337K_nUxcA1y78OpCswja?dl=0

PULKIT SHARMA

February 25, 2019 at 11:42 am

Reply

Hi Mohan,

Make sure that you are providing the correct location for the images. You have to give the entire location of the directory where your images are.

MOHAN

February 25, 2019 at 3:04 pm

Reply

The path and everything is correct only.

check this pic out, i added some logs

https://www.dropbox.com/sh/3evijlz3yval2ix/AABGdarqzm70_ObBCAudESWQa?dl=0

ASHISH CHOUDHARY

March 18, 2019 at 2:58 pm

Reply

Hi I was facing the similar issue and the solution that worked for me was that i was using python3 for the code but when i switched to python2 everything was working fine, So you could give it a try?

YOGESH

February 23, 2019 at 10:45 pm

Reply

I thought I won't be able to deal with computer vision. But after reading this article i am amazed.

PULKIT SHARMA

February 25, 2019 at 1:07 pm

Reply

Thank you Yogesh!

NIMISH

March 3, 2019 at 1:59 pm

Reply

Hi Sir! I tried this method using CPU version of torch, and I modified the code accordingly to make it work. But after one picture has een evaluated, the RAM usage suddenly booms and freezes my system entirely. Is this a memory leak on my part, or does the model require a lot of RAM? (I have 8GB RAM)

PULKIT SHARMA

March 5, 2019 at 1:25 pm

Reply

Hi Nimish,
This might be an issue related to RAM. As it is a heavy dataset, you will require more RAM. Instead of training your own model, you can download the trained weights given in the article to make predictions for new images.

ADITYA AYYAGARI

March 19, 2019 at 12:56 pm

Reply

How do we give the path in Windows?
UNABLE TO GENERATE GROUND TRUTHS!
Does this code run only on Linux? What changes must be made to replicate this code on Windows considering that the system has Cuda, Anaconda with Python3.7 and PyTorch ?

PULKIT SHARMA

March 25, 2019 at 1:27 pm

Reply

Hi Aditya,
These codes will work on windows too. You just have to give the entire path of the images and their corresponding density maps.

ANJALINA

March 22, 2019 at 3:14 pm

Reply

Any option to bypass this cuda since my machine has nvidea GPU 410M only which is not supporting the cuda9.X

PULKIT SHARMA

March 25, 2019 at 1:23 pm

Reply

Hi Anjalina,
You can use cloud services to run these codes. Some of the options could be:
Google Colab
Paperspace

ANJALINA

April 3, 2019 at 2:02 pm

Reply

Hi Pulkit,

Sorry for disturbing you . I am very new to this field and it would be great if you could give some pointer to find out solutions to the issue that I am facing.

I was trying to predict the crowed count of a local image(xyz.jpg). I don't have .mat files of my local image.

I could see the dataset that you have shared contains images and the corresponding .mat files.

I just created a predict.py as you mentioned in your code.
Without training (python train.py part_A_train.json part_A_val.json 0 0) and all, I just added the following code to predict.py(hope this is sufficient for all other steps that you mentioned)

checkpoint = torch.load('part_A/0model_best.pth.tar')
model.load_state_dict(checkpoint['state_dict'])

while running the predict.py it was complaining about the "xyv.h5". Says this file not found in:
temp = h5py.File('part_A/test_data/ground-truth/xyv.h5', 'r')

Could you please assist me to get an understanding about the .mat, .h5 and its dependency in original count prediction.

ESTHER MEAD

May 2, 2019 at 4:36 am

Reply

This is the same problem that I am having.

PULKIT SHARMA

May 2, 2019 at 10:34 am

Reply

Hi,
For predicting the count of a new image, you can use the first 7 lines of code from the article where I have showed the prediction for a single image, i.e.:

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()

The later part of the code was to compare the result when you know the actual count of the person in the image which in your case seems is not known.

So, use the above code and just change the path of the image. This will give you the count for new image.

YURIY

March 23, 2019 at 10:44 pm

Reply

Hey, thanks for the article
I'm running CUDA on my laptop with 16gb ram and 2gb video memory but alway caught the "CUDA out of memory. Tried to allocate 86.00 MiB (GPU 0; 2.00 GiB total capacity; 927.71 MiB already allocated; 51.00 MiB free; 355.54 MiB cached)"
what is the requirements for training? or how can I fix this?

PULKIT SHARMA

May 17, 2019 at 2:42 pm

Reply

Hi Yuriy,
I trained the model using very high compute. I had a ram of 128 GB. Generally, it is not easily available. So, you can just download the trained weights which I have shared in the article and use them to get predictions for new images.

RUCHIKA
August 25, 2019 at 12:29 pm

Reply

link to trained weights is not working

PULKIT SHARMA
August 26, 2019 at 1:55 pm

Reply

Hi Ruchika,
The link is working at my end. Can you share what is the error that you are getting?

RUCHIKA
August 27, 2019 at 10:29 pm

Reply

Hello Pulkit
the error when trying on windows 10 is
the archive is either in unknown format of is damaged
similar error in linux also
an error occured while loading the archive

MR. T
March 29, 2019 at 11:16 am

Reply

Hi Pulkit,

I love the tutorial, but the theory about making ground truth and density map were a bit hard for me. Could you give some sources to find an explanation of how to generate ground truth without counting the number of people in images manually?

Rgs,
Mr. T

PULKIT SHARMA
May 17, 2019 at 2:48 pm

Reply

Hi,
You can look at the following resource :
https://github.com/gjy3035/Awesome-Crowd-Counting

BEENU
April 2, 2019 at 2:53 pm

Reply

please correct me if I am wrong. I am very new to this field.This might be a simple question for you.
From this article what I could understand is evenif I supply any image from my local collection, the prediction.py will provide the crowed count.

With this assumption when I tried, for example I have added IMG5000.jpg in the images and tried to run prediction.py, its complaining about the h5 and mat files since its not there in the dataset.

mat = io.loadmat(img_path.replace('.jpg','.mat').replace('images','ground-truth').replace('IMG_','GT_IMG_')) is complaning about this.

PULKIT SHARMA
May 17, 2019 at 2:46 pm

Reply

Hi,
For predicting the count of a new image, you can use the first 7 lines of code from the article where I have showed the prediction for a single image, i.e.:

```
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()
```

You just have to change the image name with its location in the second line of the code.

ESTHER

April 17, 2019 at 3:49 am

Reply

How do you get the .mat files for each image? Thanks.

PULKIT SHARMA

May 17, 2019 at 2:50 pm

Reply

Hi esther,
I took the ShanghaiTech dataset. They have provided the .mat files for those images.

ESTHER MEAD

April 21, 2019 at 11:32 am

Reply

Thank you for this. How can I use it to count the number of people in unlabeled images?

PULKIT SHARMA

May 17, 2019 at 2:51 pm

Reply

Hi Esther,
For predicting the count of a new image, you can use the first 7 lines of code from the article where I have showed the prediction for a single image, i.e.:

```
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()
```

You just have to change the image name with its location in the second line of the code.

ESTHER MEAD

April 23, 2019 at 1:54 am

Reply

Pulkit, thank you so much for this. I've gotten everything to work using google colab with GPU acceleration. Now, how can I use this to test new images? Thanks!

PULKIT SHARMA

April 23, 2019 at 11:58 am

Reply

Hi Esther,
As mentioned in the post as well, you can use the following code to get predictions for the new images:

```
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()
```

Here, you just have to give the path to your image.

ESTHER MEAD

May 3, 2019 at 9:36 pm

Reply

I got it to work now. I had to change the filenames of my new images to begin with "IMG_"
Thanks so much!!!

ARUN

April 25, 2019 at 11:46 pm

Reply

Your article is very helpful. Thanks a lot for explaining this in a better way. But when I tested the "part_B_final/test_data_backup/images/IMG_1.jpg" with the above code(reply to Esther), I could see the count that we are getting is 54 , actual the count is only less than 30. Similarly getting some weird counts if I try images with very less count , say 1, 2 or three. May I know whether this issue is with the the way that I am executing this code or this code is not considering all those cases

PULKIT SHARMA

April 26, 2019 at 10:10 am

Reply

Hi Arun,
This model is trained on images that have large number of people. So, if you test this model on images have more density of people, the model will perform well but on the other hand, if you are using this model for images having only 1,2 or 3 (as you have mentioned) person, the performance of the model might not be very accurate.

AISWARYA

April 30, 2019 at 1:30 am

Reply

Hi Pulkit,

Awesome doc. Thanks you for posting this. This doc definitely would be helpful to all those who are interested to explore in this area.

I am facing an issue while running this code. It would be great if you could suggest a solution for the same.

I have 14GB RAM and the following is the nvidia-smi console output.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 418.56 Driver Version: 418.56 CUDA Version: 10.1 |
```

```
|-------------------------------+----------------------+----------------------|
| GPU Name Persistence-M| Bus-Id Disp.A | Volatile Uncorr. ECC |
| Fan Temp Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
|===============================+======================+======================|
| 0 GeForce MX110 Off | 00000000:01:00.0 Off | N/A |
| N/A 49C P0 N/A / N/A | 443MiB / 2004MiB | 3% Default |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes: GPU Memory |
| GPU PID Type Process name Usage |
|=============================================================================|
| 0 1480 G /usr/lib/xorg/Xorg 172MiB |
| 0 1651 G /usr/bin/gnome-shell 118MiB |
| 0 5000 G ...quest-channel-token=8194962126415318391 146MiB |
| 0 6268 G /snap/pycharm-community/123/jre64/bin/java 2MiB |
```

I am getting frequent Runtimeerror, even-if I try to predict the crowed count of 150 kb file. The error I am getting is as follows:

RuntimeError: CUDA out of memory. Tried to allocate 174.38 MiB (GPU 0; 1.96 GiB total capacity; 1.31 GiB already allocated; 65.50 MiB free; 1.12 MiB cached).

The following the code base that I am running:

```
import PIL.Image as Image
import numpy as np
from matplotlib import pyplot as plt
from image import *
from model import CSRNet
import torch
import gc; gc.collect()
torch.cuda.empty_cache()
from torchvision import datasets, transforms

transform=transforms.Compose([
transforms.ToTensor(),transforms.Normalize(mean=[0.485, 0.456, 0.406],
std=[0.229, 0.224, 0.225]),
])

model = CSRNet()
model = model.cuda()

#loading the trained weights
checkpoint = torch.load('/home/abc/Downloads/Shanghai/part_A_final/0model_best.pth.tar')
model.load_state_dict(checkpoint['state_dict'])

from matplotlib import cm as c
img = transform(Image.open("/home/abc/Downloads/Shanghai/xyz.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()
```

PULKIT SHARMA

May 1, 2019 at 5:39 pm

Reply

Hi Aiswarya,

Try to use a GPU to run these codes. As the codes are heavy, they might not run on a cpu. You can try to run these codes on google colab as well which provides free GPU access.

BHARTH SINGH

April 30, 2019 at 1:39 am

Reply

Hi Pulkit,

The '0model_best.pth.tar' contain the pretrained weight of the 1198 images(ie the train and test images of part_A and part_B) right?

PULKIT SHARMA

May 1, 2019 at 5:33 pm

Reply

Hi Bharth,
These pretrained weights only contains the learning from the train images and not the test images.

MISHAL

May 1, 2019 at 7:54 am

Reply

Hi Pulkit,
Excellent explanation, keep on the good work!
I am running the code on Anaconda python 3 on windows and it did not like the following line

if gt_count > 1:

in the "# function to create density maps for images" Line 20
it complained about the semicolon. is there a problem in the syntax ?
Also I had to install openCV which was not mentioned in the explanation but needed by the code.

PULKIT SHARMA

May 1, 2019 at 5:31 pm

Reply

Hi Mishal,
If you are using if statement, you have to give : after the condition.
And thank you for pointing out about openCV. I will add that in the article.

BEENU

May 2, 2019 at 5:17 pm

Reply

Thanks a lot for this doc. For the crowed prediction, DO you think it has some dependency with the image size and pixels. I could see the trian images are with a dimensions of 1024*768 . Do you think if the image size and dimension is bigger then the accuracy will also be more fine tuned.

PULKIT SHARMA

May 2, 2019 at 5:52 pm

Reply

Hi Beenu,
It depends. I can not confirm that increasing the shape of image will always give better performance. The performance might improve and you might get a better model. But, as you increase the size of the image, computation cost also increases. You will need more memory to store those images and stronger computation to build your model on larger images. So, you must also try to reduce the computation cost instead of just focusing on improving the model performance.

NIHED

May 16, 2019 at 8:16 pm

Reply

Hi! great article!
I have stumped into a little problem and I do not know how to fix it.
This is the error I am receiving:

IndexError Traceback (most recent call last)
in ()
1 #now see a sample from ShanghaiA
—-> 2 plt.imshow(Image.open(img_paths[0]))

IndexError: list index out of range

The location of my root is this:
#set the root to the Shanghai dataset you download
root = '/Desktop/CSRNet-pytorch-master/ShanghaiTech'

#now generate the ShanghaiA's ground truth
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]

I do not understand why I am receiving the error
Could someone help me?

PULKIT SHARMA

May 17, 2019 at 2:54 pm

Reply

Hi Nihed,
It seems like you have not given the correct path and hence img_paths is empty. Try to give the exact path of the images and then run this code.

NIHED

May 20, 2019 at 4:21 am

Reply

Hi Pulkit!

I do not have an GPU processor that works for this.. How do I use Google Colab?
OR could you maybe send me the right pre weights because your I am unable to open your tar. file..
I am working on a MacBook Pro

Kind regards,
Nihed

NIHED

May 20, 2019 at 4:29 am

Reply

I mean that I do not understand how I should use this in Google Colab:
cd CSRNet-pytorch
python train.py part_A_train.json part_A_val.json 0 0

Because I receive this:
[Errno 2] No such file or directory: 'CSRNet-pytorch'
/content/drive/My Drive/CSRNet-pytorch

[Errno 2] No such file or directory: 'train.py part_A_train.json part_A_val.json 0 0'
/content/drive/My Drive/CSRNet-pytorch

while the file is there in my google drive……

AAYUSH JAIN

May 22, 2019 at 4:06 pm

Reply

Hi Pulkit ,
Can you share the drive link for pre trained weights again ?
it seems as if it is damaged because i tried open it in both linux and windows but failed to do so .
It will be really helpful !

PULKIT SHARMA

July 15, 2019 at 6:09 pm

Reply

Hi Aayush,
The link is working fine at my end. Here is the link for your reference:
https://drive.google.com/file/d/1KY11yLorynba14Sg7whFOfVeh2ja02wm/view

---

RAHUL SHARMA

May 26, 2019 at 11:43 am

Reply

I am new in this field, i know only basics of machine learning. What are the pre-requisite to know this article completely. If you have source, then please send me their links.
Thank you .

PULKIT SHARMA

July 15, 2019 at 6:17 pm

Reply

Hi Rahul,
You can refer this course: https://courses.analyticsvidhya.com/courses/computer-vision-using-deep-learning-version2

JARVAH

May 27, 2019 at 1:06 pm

Reply

Hi! This is a great article! Can this framework be used to do people detection?

PULKIT SHARMA

July 15, 2019 at 6:15 pm

Reply

Hi Jarvah,
This framework is just to count the number of people in an image and will not be effective for detection tasks. For Object detection or people detection tasks, you can try frameworks like Faster RCNN or YOLO, etc.

VIGNESH

June 14, 2019 at 11:33 am

Reply

As, I am very new to machine learning. Kindly, clear me this error. I cant find .h5 file inside gt_file='part_A/test_data/ground-truth/IMG_100.h5',r in Part-B Cool Method .Thank you

---

PULKIT SHARMA

Reply

June 18, 2019 at 7:16 pm

Hi Vignesh,
For test images, you have to predict the count of people in the images and hence .h5 file is not available for test images. .h5 file contains the predictions.

MINORU AIKAWA

June 26, 2019 at 12:43 pm

Reply

This is really great work! very impressed!
I'd like to know whether I can use these scripts for commercial use.

PULKIT SHARMA

June 26, 2019 at 2:52 pm

Reply

Hi Minoru,
Contact me at pulkits@analyticsvidhya.com so that we can discuss where you will be using these scripts, what would be the application and other things. We can discuss these things and then decide whether it can be used or not.

SOMAYAH

July 22, 2019 at 12:42 pm

Reply

Thank for the great tutorial.
How I can visualize the counted heads? either by boundary boxes or points on the considered head?
Thank you

PULKIT SHARMA

July 22, 2019 at 2:44 pm

Reply

Hi Somayah,
It return the density plot. You can visualize that to see where are the people in the image.

SURAJ DAKUA

August 6, 2019 at 3:47 pm

Reply

Hello Pulkit Sharma,
Can we merge Part A and Part B images instead of splitting in two parts?
Thankyou in advance.

PULKIT SHARMA

August 6, 2019 at 4:37 pm

Reply

Hi Suraj,
Part A and Part B contains two different type of problems. In Part A, we have images having a high density of people whereas Part B images have a lower density. You can try to merge these Part A and Part B images and then check the results.

SURAJ DAKUA

August 7, 2019 at 1:03 pm

Reply

Hello Pulkit,
When importing libraries it gives error like this.
– no module named 'model'
– no module named 'image'

– no module named 'image'
I'm using Anaconda3 and Python 3.7.

RUCHIKA
August 23, 2019 at 10:00 am

Reply

how long does it take to train the network

PULKIT SHARMA
August 26, 2019 at 2:24 pm

Reply

Hi Ruchika,
It depends on the configuration of the system that you are using. I trained it on a system having 16 GB RAM, and 20 cores and it took around 5-6 hours to train the model.

RUCHIKA
August 27, 2019 at 10:04 pm

Reply

the link below of drive you have shared is not working
https://drive.google.com/file/d/1KY11yLorynba14Sg7whFOfVeh2ja02wm/view
please share a working link again
Thanks

DIVYAKANT TAHLYAN
September 18, 2019 at 11:46 pm

Reply

Was the pre-trained model that you provided trained on part A dataset or part B dataset. If it was for part A, can you please provide the one trained on part B as well.

PULKIT SHARMA
September 19, 2019 at 12:26 pm

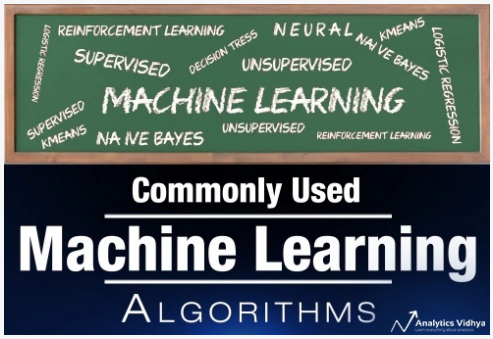
Reply

Hi Divyakant,
The pre-trained weights are for Part A images. I have not trained the model for Part B images. You can easily train the model for images in Part B as well following the steps mentioned in the article. If you get stuck somewhere, or need any assistance, I will be happy to help.

RECOMMENDED READS

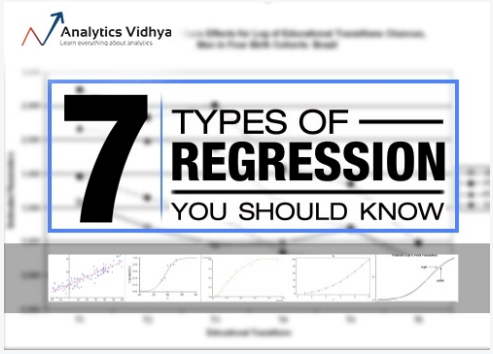
A Complete Python Tutorial to Learn Data...

January 14, 2016



Commonly used Machine Learning Algorithm...

September 9, 2017



7 Regression Techniques you should know!...

August 14, 2015

RECOMMENDED RESOURCES

Free Course for you

Comprehensive Learning Path for Deep Lea...

Practice & Learn

Identify the apparels...

Try for free

Applied Machine Learning - Beginner to P...

Analytics Vidhya

Learn everything about analytics

DATA SCIENTISTS

COMPANIES

JOIN OUR COMMUNITY :

Don't have an account? Sign up here.

7513