Crowd Counting - From Real to Synthetic Datasets

Computer Vision and Multimedia Analysis

Massimo Clementi, Elisa Nicolussi Paolaz

14 February 2020

UNIVERSITY
OF TRENTO

Contents

1 Introduction **3**

2 Unity simulations **3**
2.1 Crowd generation 3
2.2 Output images and ground-truth 4
2.3 Image processing 5
2.4 Grayscale conversion 6

3 MATLAB script **7**

4 Deep Neural Network **8**
4.1 The neural network 8
4.2 Training phase 8
4.3 Testing phase 8
4.4 Training on Synthetic dataset 8
4.5 Cross training on both ShanghaiA and Synthetic datasets 9

5 Conclusions and further improvements **11**

1 Introduction

One of the most problematic aspect in neural network applications is the retrieval of annotated samples for the training phase. For crowd counting estimation, to annotate all the head-points may be challenging especially when it is necessary to deal with high density crowds. This leads inevitably to low availability of public datasets for crowd counting neural network training.

The goal of this project is therefore to improve the training of neural networks for crowd counting using synthetic datasets. The generation of both images and ground-truth data is automatically carried out by Unity simulations, where many parameters can be tuned to obtain crowds with different properties (i.e. density, orientation) and different camera properties (focal length, sensor size).

In this report, first the generation of the synthetic images will be discussed in detail, then samples will be fed to the novel Deep Structured Scale Integration neural network to obtain tangible numerical values and evaluate the effectiveness of the method, with respect to the traditional one.

2 Unity simulations

2.1 Crowd generation

The crowd is generated in Unity starting from a set of mesh models. The algorithm defines a grid of row and column positions according to the amount of people it needs to generate and for each grid position it instantiates a person, which is randomly chosen from the available set. In order to obtain a more realistic positioning, random values are evaluated and applied as both row and column small displacement. Moreover an arbitrary rotation within a specific range is applied to each person.

```
int posRand = 15;
int rowRandCumul = 25;
int columnRand = 40;
float yRotMin = 3*Mathf.PI/4;
float yRotMax = 5*Mathf.PI/4;

// Random components evaluation
columnRandIndex = (float)(UnityEngine.Random.Range(0,columnRand)-columnRand/2)/100;
rowRandCumulIndex += (float)(UnityEngine.Random.Range(0,rowRandCumul))/100;

// Compute position
randTemp = (float)(UnityEngine.Random.Range(0,posRand)-posRand/2)/100;
xPosition = (x + columnRandIndex + randTemp - xIstances/2) * xSpacing;
randTemp = (float)(UnityEngine.Random.Range(0,posRand)-posRand/2)/100;
zPosition = (z + rowRandCumulIndex + randTemp - zIstances/2) * zSpacing;

// Compute rotation
yRotation = UnityEngine.Random.Range(yRotMin,yRotMax);
```

In particular the following table shows all the parameters with a brief description.

Variable name	Variable description
`posRand`	contribution of the additive component of (x,z) model position
`rowRandCumul`	maximum range of the random row (z) cumulative displacement
`columnRand`	maximum range of the random column (x) displacement
`yRotMin, yRotMax`	range of the random angle displacement (in radians)

The implementation of the crowd generator algorithm is carried out exploiting the features of Unity's high-performance Data-Oriented Technology Stack (DOTS) and in particular employing the Entity Component System (ECS).

In ECS, standard GameObjects are being replaced by Entities in order to achieve a much higher memory efficiency and thus be able to handle a larger number of instantiated people in the crowd. Unity's Entity Component System helps eliminate inefficient object referencing, substituting GameObjects and their own collection of components with entities that only contain required data. This leads to an improvement in the management of data storage allowing high-performance operations.

Finally, in order to generate more realistic scenarios, we have created 4 different scenes with different backgrounds and crowd positioning.

a) Scene 1

b) Slanted

c) Scene 2

d) Scene 3

2.2 Output images and ground-truth

After the crowd generation process is completed, the output images are taken by a specified set of different cameras, each of which is characterized by some tunable parameters regarding its positioning with respect to the crowd and its properties such as the focal length and the sensor size.

In our implementation, in order to produce a dataset of images as similar as possible to images taken from security cameras, we have chosen to set the focal length to 3.6mm choosing either an 8mm or a super 8mm sensor type, thus obtaining a 50-60 degree field of view. This particular choice is suitable for residential surveillance applications.

In the simulation environment we have the following cameras and sensor types:

Camera	**Scene 1**	**Slanted**	**Scene 2**	**Scene 3**
Camera 1	Super 8mm	8mm	Super 8mm	8mm
Camera 2	8mm	8mm	8mm	8mm
Camera 3	8mm	8mm	8mm	8mm
Camera 4	8mm	8mm	8mm	8mm
Camera 5	8mm	8mm		
Camera 6	8mm	8mm		
Camera 7	8mm	Super 8mm		
Camera 8	8mm	Super 8mm		

Moreover it is possible to save the output images with different resolutions, adjusting a factor which directly scales the `cam.pixelWidth` and `cam.pixelHeight` parameters of the camera as follows:

```
int captureWidth = (int)(cam.pixelWidth*scaling_factor);
int captureHeight = (int)(cam.pixelHeight*scaling_factor);
```

By doing this we aim to enhance the multi-scale properties of the neural network used in the project, which will be explained later in detail.

Each output image is then associated with a corresponding ground-truth `.txt` file containing each person's head position converted into camera coordinates and saved as a row-column pixel position.

The head position is initially computed as follows in world coordinates and inserted in the `headPositions` array:

```
headx = xPosition;
headz = zPosition;
heady = refY[index] * 3/2;
```

where `refY[index]` is the height of the reference point of each mesh model, where the transform component is centered.

Each head position is then converted into camera coordinates, which are normalized between 0 and 1, checked if visible by the camera and finally saved as follows:

```
for(int i=0; i<size; i++){
    Vector3 pos = cam.WorldToViewportPoint(CrowdGenerator.headPositions[i]);
    if( (pos.x >= 0 && pos.x <= 1) && (pos.y >= 0 && pos.y <= 1) && (pos.z > 0) ){
        Save(fn,
            ((int)(pos.x*captureWidth)).ToString()+" "+
            ((int)(pos.y*captureHeight)).ToString()+"\n");
    }
}
```

2.3 Image processing

With the purpose of diversifying the samples in the synthetic dataset, both illumination and image features have been modified.

Firstly, in Unity environment, we have changed the illumination source switching from a directional light, which provided uniform illumination over the entire scene, to a point light with higher intensity to light up either people in the foreground or in the background and have thus different illuminations across the scene.

Secondly we have applied some image processing effects such as underexposing, overexposing, blurring (Gaussian, motion, radial), changing channels level or color saturation.

a) Point light

b) Motion blur

c) Overexposing effect

d) Color desaturation

2.4 Grayscale conversion

In order to deal with grayscale images in ShanghaiA dataset, a MATLAB conversion script has been included in the workfolder. After setting a convenient probability value `P`, it goes through all the image samples and transform them into grayscale with respect to that probability value, as follows:

```
for i = 1:nimages
    if(rand*100 <= P)
        Irgb = imread(strcat(pathimg, '/', images(i).name));
        Igray = rgb2gray(Irgb);
        imwrite(Igray,strcat(pathimg, '/', images(i).name));
    end
end
```

In conclusion the following table shows the final number of images generated for each type of scene, processing step and resolution.

	Number of images	Composition
Scene 1	48	6 takes × 8 cameras
Slanted	24	3 takes × 8 cameras
Scene 2	24	6 takes × 4 cameras
Scene 3	24	6 takes × 4 cameras
Scene 1 (scaled)	16	2 takes × 8 cameras
Slanted (scaled)	16	2 takes × 8 cameras
Scene 2 (scaled)	8	2 takes × 4 cameras
Scene 3 (scaled)	12	3 takes × 4 cameras
Illumination changes	20	–
Image Processed	40	–
Total	**232**	

3 MATLAB script

The custom synthetic dataset is composed of a set of *img* - *gt* pairs generated automatically by the Unity scripts. The `.jpg` files are the computer generated images while the `.txt` files contain all the head point coordinates. The MATLAB script `generate_gt`, which can be found in both `train_data` and `test_data` folders, loads for each `.jpg` image the corresponding `.txt` and creates the `.mat` ground-truth file that meets the input requirements of the neural network. The final `.mat` file embeds a `structure array` variable, which contains in its two fields the array of head positions ("`location`", nx2 matrix) and the total number of heads ("`number`", int).

The following code shows the method to read the head position from the `.txt` file, create the struct variable and finally save it.

```
gts = dir([pathgt filesep '*.txt']); images = dir([pathimg filesep '*.jpg']);

for i = 1:length(gts)
    fileID = fopen(strcat(pathgt, '/', gts(i).name),'r');
    formatSpec = '%f %f';
    sizeA = [2 Inf];
    A = fscanf(fileID,formatSpec,sizeA); A = A';
    fclose(fileID);

    filename = strcat(pathimg,'/',images(i).name);
    img = imread(filename);
    [r, c, ch] = size(img);

    for row = 1:size(A)
        A(row,2) = r - A(row,2);
    end

    image_info = cell(1);
    field1 = 'location';    value1 = A;
    field2 = 'number';      value2 = size(A);
    image_info{1,1} = struct(field1, value1, field2, value2);

    name = gts(i).name; name = strsplit(name,'.'); name = name{1,1};
    filename = strcat(pathgtmat,'/GT_',name,'.mat');
    save(filename,'image_info');
end
```

The following images display some samples with the head position overlayed.




4 Deep Neural Network

4.1 The neural network

The neural network used in this project is the official implementation of the *Deep Structured Scale Integration Network* for a multi-scale state-of-the-art crowd counting framework, developed by *Lingbo Liu et al.* in 2019.

The network is composed of three parallel subnetworks that process images at different scale, using convolutional layers and feature enhancement modules. Only the most relevant features are preserved and contribute to the development of the final density map.

The size and the number of people in the crowd may vary greatly and the use of density maps has been proved to be useful in reducing the problem. Moreover the integration of these maps give as result the approximated number of total heads in the image.

4.2 Training phase

The ground truth files (`.mat`) are loaded to generate the density maps via the geometry-adaptive Gaussian kernels method. The first ten convolutional layer weights are then loaded from the VGG-16 pretrained network and the remaining ones are initialized with random values from a Gaussian distribution.

The loss function for the training phase is the *Dilated Multiscale Structural Similarity* (DMS-SSIM) loss and at each training iteration, 16 image patches (size 224x224) are cropped from the input image and fed into the network.

The training settings can be customised in the `train.sh` script. To launch the train just run the script in the following way:

```
nohup ./train.sh &
```

Note that the `nohup` command is used to detach the bash process from the `tty` and make the process immune to sleep signals and tty logouts.

The computed training weights are saved for each epoch in the `saved_models/<dir_name>`; the directory name is generated by the script from the model used, the training dataset, date and time.

4.3 Testing phase

In the testing phase, the full image is fed to the network and the final density map evaluated. Various metrics are computed (*MAE*, *MSE*) based on the residual between the estimated and the ground truth values.

To test the network, both the model and the target dataset needs to be specified in the `test.py` script. Run the following terminal command to load the `.h5` weights and test the network performances:

```
./test.sh ./saved_models/<dir_name>/<epoch>.h5
```

4.4 Training on Synthetic dataset

To evaluate the similarities of the two datasets, the neural network has been trained on synthetic and tested on ShanghaiA. To carry out a proper comparison, we also reproduced the values of the ShanghaiA train of the official paper using several settings and choosing the most appropriate one.
The results are presented in the following table:

Training dataset	Testing on ShanghaiA	
	MAE	MSE
ShanghaiA	62.79	104.57
Synthetic	**112.91**	**182.30**

The synthetic dataset shows a significantly good similarity to the ShanghaiA; note that the training has been carried out only on synthetic images, no real images!

4.5 Cross training on both ShanghaiA and Synthetic datasets

Moreover a further analysis on the dataset integration has been evaluated, merging the two datasets and training the neural network on the resulting one. The new training dataset is composed of 300 ShanghaiA images and 232 Synthetic images. In order to obtain comparable results the train settings are once more those of the original ShanghaiA train.
Both reference and cross training results are presented in the following table:

Training dataset	Testing on ShanghaiA	
	MAE	MSE
ShanghaiA	62.79	104.57
Cross-training	**64.62**	**110.64**

It can be pointed out that the differences between the *MAE* and *MSE* values are considerably low, therefore we decided to perform a further analysis image per image on the test results of both ShanghaiA training and cross-training, with the help of a Python script.

The goal is therefore to identify both the images where the cross-training technique has allowed us to obtain better results compared to the ShanghaiA training and the ones where the network performs worse.

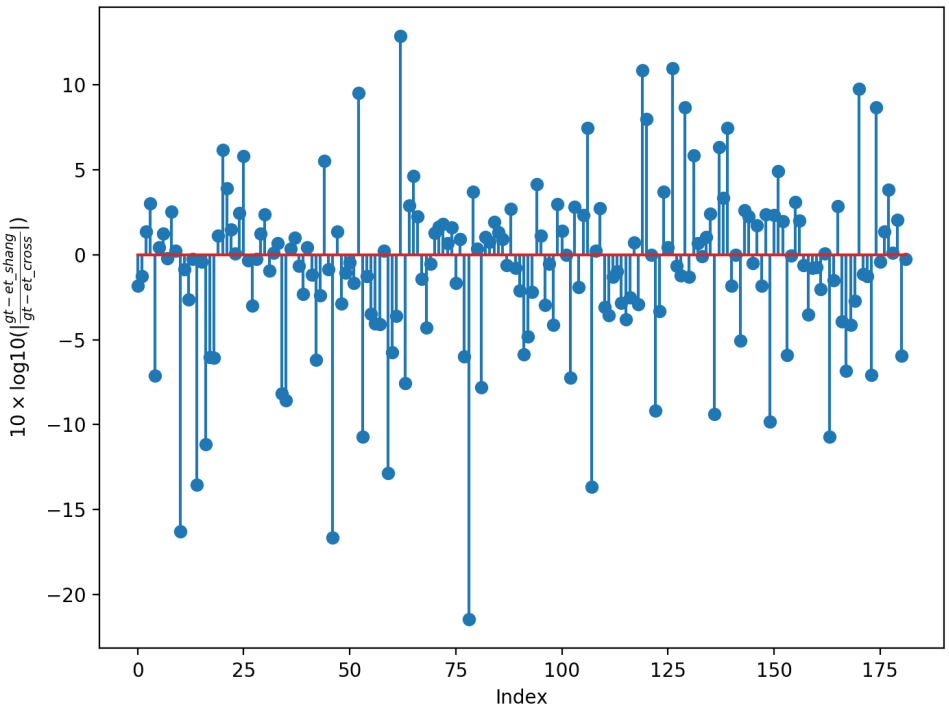
The metric that has been defined for the purpose is

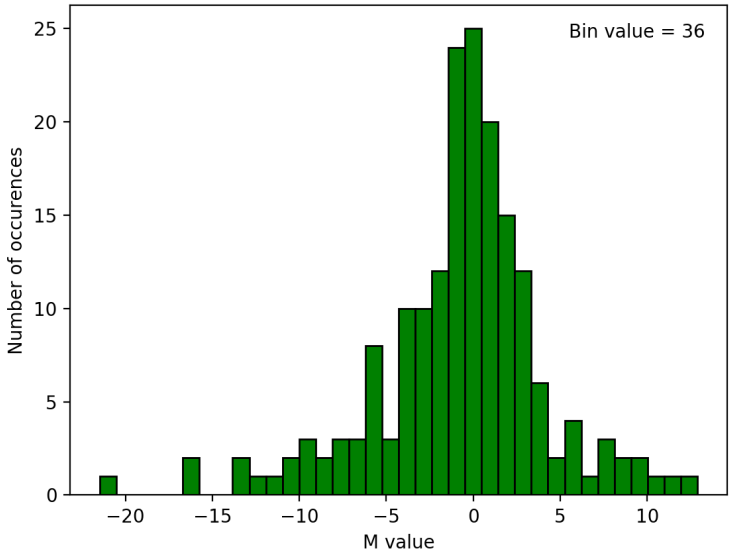
$$M = 10 \times \log 10 \left(\left| \frac{gt - et\_shang}{gt - et\_cross} \right| \right)$$

where gt is the ground-truth value of the sample, et_shang is the estimated value from the ShanghaiA training and et_cross is the estimated value from the cross-training.

It can be noticed that, when the $gt - et\_cross$ residual is smaller than $gt - et\_shang$, the value of the metric is significantly bigger than zero and therefore each positive peak represents an image where the cross-training estimation outperforms the ShanghaiA train.

On the other hand when the $gt - et\_shang$ residual is much smaller than $gt - et\_cross$, the fraction is close to zero and therefore the metric value approaches $-\infty$. The negative peaks represent thus images where the cross-training has poor performances.





Though at first glance the total number of positive and negative peaks in the first plot, which are respectively 83 and 99, might compensate each other and the values of the *MAE* and *MSE* metrics might not show a significant gain, the results obtained with the cross-training technique actually reveal a significant improvement which can be observed by looking at the characteristics of the images related to the positive and the negative peaks. Better results are obtained mainly on scenes similar to the ones modelled through the Unity generated samples.

In particular the cross-training brings significant benefits to images which are characterized by non-uniform illumination, high contrast, high exposure and images with bright light spots. In addition it has good performances on images with both buildings and trees in the background, on blurred pictures and on color desaturated or black and white photos.

Here are some of the best performing images:

a) IMG_30 - High exposure and contrast

b) IMG_42 - Trees in the background

c) IMG_146 - Side buildings and motion blur

d) IMG_60 - Bright light spots

On the other hand the cross-training performs worse than the original ShanghaiA training on images with predominant non-modelled structural elements such as buildings and balconies, in unusual light and perspective conditions and in scenes with uncommon objects such as chairs and tables among the crowd.

Here are some of the images with poor performances:

a) IMG_45 - Predominant building

b) IMG_108 - Unusual lights and angle

c) IMG_111 - Large balconies

d) IMG_10 - Uncommon objects

5 Conclusions and further improvements

The ability to generate annotated images of crowds through a graphic software and the possibility to use them as training samples for a crowd counting neural network obtaining meaningful results surely opens up many opportunities in the field. First of all it allows us to solve the crucial problem of the head-points annotation, producing automatically an accurate ground-truth for each generated image. Moreover the crowd generating process is completely adjustable to any type of scenario, granting thus the possibility to create synthetic datasets that best meet the requirements of different applications. This means that if the neural network needs to work with images characterized by a particular scenario, illumination or camera acquisition, it would be possible to produce specific samples in order to achieve a better training and thus better results.

The technique has proven to bring benefits on scenarios that have been previously modelled, enhancing effectively the network's accuracy on the final estimations, however it requires ad-hoc modelling and fine tuning. It is reasonable to think that with the generation of a sufficient amount of additional scenes, with a larger set of crowd properties (i.e. clothing, faces, accessories) and camera features it should be possible to have results in the cross-training that are even better than the reference ones.

Some of the aspects that might be improved:

- more accurate modelling of people, scenes and image processing, mostly application-based
- randomize camera positions and orientations, add more types of cameras, lenses and/or sensors
- experiment with different values of crowd parameters (i.e. number of people, arrangement, spacing)