Deep Learning Framework for Automatic Crowd Counting via Density Maps Generation

Mattia Miolato
mattia.miolato@studenti.unipd.it

Giulia Campesan
giulia.campesan.1@studenti.unipd.it

1. Introduction

Automatic Crowd Counting in Computer Vision aims at displaying the distribution of a crowd and to get an estimation of the number of people in it via numeric integration of the density map[3]. It has been gaining popularity thanks to its application to public safety and transportation systems[6]. Moreover, the pandemic situation has made crowd behaviour analysis more and more popular. Recent works have been applying this task to social distancing for Covid-19 pandemic containment [4].

Crowd counting is part of the wider field of object counting, that aims at estimating the number of target objects, such as vehicles, cells, people or items in a warehouse, in still images or video frames [6].

Early stages approaches to this task were either relying on detection or regression.

Detection-based approaches aim at finding the right-sized moving-window able to detect a single object. Clearly, this approach fails in congested scenes, in which the whole object is not visible. Then, it is not of particular interest for the application to people counting.

Instead, regression-based techniques try to map low-level extracted features to the number of objects in the image. Both methods share the limit that they can retrieve only the counting of items but do not contain any information on their spatial distribution. Then, up-to-date works focus on density-map generation methods, most of which are based on Convolutional Neural Networks (CNN)[2].

Several CNN-based architectures exploit the VGG-16 as their backbone. VGG-16 is a deep convolutional (conv) network for large-scale image recognition. With respect to other well-known architectures, such as AlexNet, it exploits very small kernels of size 3×3. The image is processed via stacks of conv filters, followed by max-pooling layers of size 2×2 and stride 2. Its 138 MM parameters have been trained on the ImageNet dataset[5].

In this work, we implement a deep CNN architecture for density map generation of congested crowd images. We use pre-trained layers from VGG-16 as the front-end encoder and stacks of trainable dilated conv layers as back-end decoder. Dilation is included in filters by setting a dilation rate value bigger than 1. It allows enlarging the receptive field while keeping the kernel size small[2].

We obtain satisfactory results, with an MAE and RMSE comparable with strong baselines reported in the literature.

The paper is structured as follows: in section 2 we recall published works that share our aims and we discuss similarities and differences with our method. In 3 we describe the well-known Shanghai Tech dataset, that we choose to train and evaluate our network. Going to section 4, we describe, besides our architecture, data pre-processing, data augmentation and ground-truth generation. We report and analyse the results of our experiments in 5 and draw conclusions in 6.

2. Related Work

Several works [3] exploit a certain number of pre-trained layers from VGG-16 as a backbone.

In particular, CSRNet [5] uses the same dilated convolutional approach that we choose. The overall architecture structure is shared between our model and CSRNet. The main difference is that, despite using the same number of pre-trained layers, we propose a much smaller net, reducing the extent of the trainable back-end. This choice has the intent of making density map crowd counting within reach of basic-user hardware.

Previous works [8] in crowd density control propose Multi-Column CNN, in the framework of multi-scale architecture technique. This means that thanks to different kernel sizes exploited in different network branches, they are flexible with respect to images congestion levels. Despite giving overall good results as a first approach, they suffer from bad-scaling of training time with network depth [2].

Lastly, we recall the detection and regression methods, having the limits displayed in 1

3. Dataset

The Shanghai Tech Dataset has been collected and published by Zhang et al in 2016 [8].

It is constituted of 1198 crowd images and related labels.

The label is a vector reporting the coordinates of each person in the image. It can be visualized with a 2D plot in which each person in the crowd is annotated with one point in correspondence of his head. The whole dataset reports a total of 330,165 people.

Most of the images are RGB, however, some B&W pictures obtained by older frames are present.

The dataset is divided into two separate sections, Part A and Part B, distinguished by crowd density.

Part A accounts for 482 images with different resolutions, split into 300 instances for training and 182 for testing. The shots, taken from the internet, report congested scenes with an average of 501.4 annotated heads [6].

Part B is sub-setted into train and test of respectively 400 and 316 pictures with the same 768×1024 resolution, for a total of 716 items. Images are captured in the streets of Shanghai and display a lower density compared with Part A frames, with an average of 123.6 people per image[6].

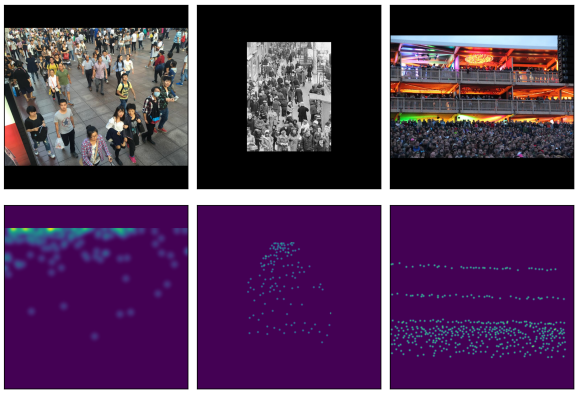

Figure 1. Some images taken from ShangaiTech dataset and their related ground-truth density map

4. Method

4.1. Ground-truth Generation

In order to get the ground-truth density map from people's locations, we convolve the image with normalized Gaussian kernels G_σ centred at annotated heads position $\mathbf{x}_i$. The obtained label is displaying the crowd distribution map

$$F(\mathbf{x}) = \sum_{i=1}^{N} \delta(\mathbf{x} - \mathbf{x_i}) G_\sigma \quad [2]$$

in which we are considering, as a first approximation, the hypothesis in which crowd average distance σ is supposed to be constant in the whole dataset. This hypothesis would clearly not hold if applied to the whole Shanghai dataset. Then, we choose to consider separately Part A, in which we set $\sigma_A = 3$ and Part B, where $\sigma_B = 15$.

It should be kept in mind that Gaussian kernel filtering generates really low pixel values: this could lead to gradient vanishing issues in network training. We face this problem by multiplying pixels intensity by a factor k[3], that we will discuss in 5. We report in fig. 2 the distribution of the pixels' value for density map in both part of the dataset.

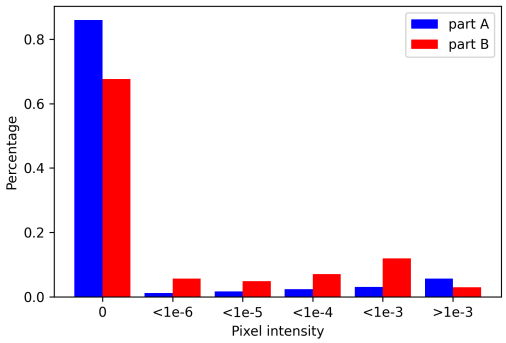

Figure 2. Pixels' value distribution

4.2. Image Pre-Processing

Image pre-processing turns out to be one of the key elements for performance boosting.

At first, we face the fact that images in Part A do not share a common resolution, in contrast to Part B, made of 768×1024 pixel pictures.

We proceed on each image in the whole dataset as follows:

- we apply zero-padding to reach 1024×1024 size,
- we resize to 256×256 resolution. This choice will be justified in 4.3.

Please, notice that the 'resize' operation is not preserving normalization: when going from $H \times W$ to $H' \times W'$ resolution, the cumulative intensity is scaled by $H'W'/(HW)$. In our case, this means that a factor 16 is needed to retrieve literature-comparable metrics values.

After ending up with images of the same size, we perform intensity rescaling as prescripted when using VGG-16 as the front-end. We make use of the built-in function vgg16.preprocess_input() in the Keras package with which 'the images are converted from RGB to BGR, then each colour channel is zero-centred with respect to the ImageNet dataset, without scaling'[1].

4.3. Data Augmentation

A common trick to increase training efficiency is to perform data augmentation by cropping. Our approach to this

technique is the following: we start, as in 4.2, by zero-padding to 1024×1024 resolution. At this point, we crop 256×256 patches, to get 16 images, that will be treated as independent training objects.

4.4. Network Architecture

At first, we point out that our network has been implemented using the Keras library in python.

As proposed by several works, we use VGG-16 as the backbone for our architecture. In particular, we keep its first 11 convolutional layers as front-end. This stack includes 3 max-pooling layers, reducing the image size of a factor 8.

Then, the back-end will include both dilated convolutional layers with 3×3 kernel size and 3 up-sampling ones, to expand back to the input size. The up-sampling phase is performed with bilinear interpolating 2×2 filters.

A dilated conv filter $w(i, j)$ of size $M \times N$ acts on the input pixels $z(h, w)$ and outputs $y(m, n)$ as

$$y(m,n) = \sum_{i=1}^{N} \sum_{j=1}^{M} z(m + d \times i, n + d \times j) \quad [2]$$

The dilation rate d controls how sparse the kernel is (see fig. 3) and allows to expand the receptive field without enlarging the filter size, which would imply an increase in the number of trainable parameters.

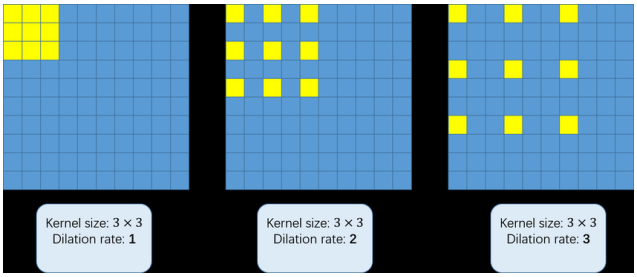

Figure 3. Dilated convolutional filters with $d = 1, 2, 3$.

We set the kernel stride to 1.

Moreover, we perform batch normalization after each convolutional layer.

Finally, we report in table 1 a schematic representation of our network structure.

5. Experiments

We recall that all the experiments and tests we performed are implemented in Python with Keras.

5.1. Loss and Metrics

The loss function used for training is the built-in 'Mean Squared Error' (MSE) Keras loss, that is returning the squared Euclidean distance between the ground-truth density map Z_i and the network predicted one Z_i^{GT}, averaged over a training batch of size N:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} \left\| Z_i - Z_i^{GT} \right\|_2^2$$

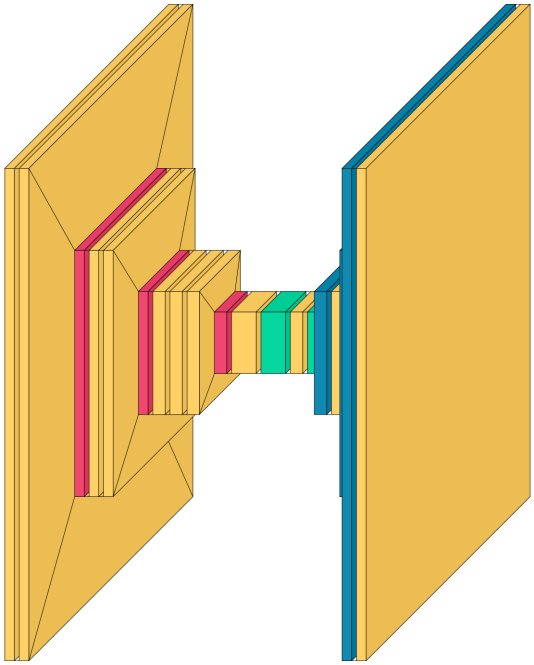

Figure 4. Visual representation of our network

fine-tuned front-end from VGG-16, 11 layers
Conv2D(64, 3×3, 1)
Conv2D(64, 3×3, 1)
MaxPool2D(2×2, 2)
Conv2D(128, 3×3, 1)
Conv2D(128, 3×3, 1)
MaxPool2D(2×2, 2)
Conv2D(256, 3×3, 1)
Conv2D(256, 3×3, 1)
Conv2D(256, 3×3, 1)
MaxPool2D(2×2, 2)
Conv2D(512, 3×3, 1)
trainable back-end, 8 layers
Conv2D(512, 3×3, 2)
Conv2D(256, 3×3, 2)
Upsampl2D(2×2, bilinear)
Conv2D(128, 3×3, 2)
Upsampl2D(2×2, bilinear)
Conv2D(64, 3×3, 2)
Upsampl2D(2×2, bilinear)
Conv2D(1, 1×1, 1)

Table 1. Network architecture. Notation is the following: Conv2D(number of filters, kernel size, dilation rate) for 2D conv layers, MaxPool2D(window size, stride) for 2D max pooling layers and Upsampl2D(up-sampling factor, interpolation) for 2D up-sampling layers

We split the training set to reserve its 30% for validation during learning, in order to have solid comparisons between different settings of hyper-parameters.

To finally evaluate the performance of our architecture, we use 'MAE' and 'RMSE' metrics[2]:

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

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(C_i - C_i^{GT}\right)^2},$$

where we are averaging over the test set of size N and C_i, C_i^{GT} are, respectively, the countings on the predicted and ground-truth density maps, with pixel intensity $z_{h,w}$ at site (h, w).

$$C_i = \sum_{h=1}^{H}\sum_{w=1}^{W} z_{h,w} \quad [2]$$

It has to be reminded that, when evaluating results with the augmented dataset, the network predictions are for single patches, corresponding to $\frac{1}{16}$ of the original image. To compare our performance with state-of-the-art results, we compute metrics on every patch composing a single dataset image and retrieve cumulative values by summation.

5.2. Training details

As a first point, we would like to stress the fact that, despite choosing the same exact network structure for processing Part A and Part B, the actual architectures are two: one trained on Part A and predicting over it and another identical one specialized on Part B. Our choice is led by the strong difference in the average distance σ in the two sub-datasets.

5.2.1 Baseline

As the first evaluation of our model performance, we implement and test a baseline architecture.

To keep the learning phase computationally feasible, back-propagation is applied only to the back-end weights, that at first are initialized following the default uniform Xavier initialization. The VGG-16 extracted front-end stack is already finely pre-trained and is kept fixed in all of our experiments.

Gradient descent is performed on batches of size 16 via Adam optimizer, with learning rate $lr = 10^{-6}$. As the activation function, we propose the Rectified Linear Unit (ReLU). The model is trained for 60 epochs.

To get an insight on the dilation effect on training, initially we exclude it, by setting $d = 1$.

At first, no regularization terms are included in our baseline.

We generate the ground truth maps as explained in the previous section and set the enhancing parameter k to 500 (see sec. 4.1). The data are pre-processed as explained in sec. 4.2 but in this phase we do not include any data augmentation.

Performance metrics for this baseline can be found in table 2 at the *No data augm* row.

After getting our baseline architecture, we try several network configurations, changing both our network hyper-parameters and image generations criteria.

5.2.2 Data augmentation

As just mentioned, for the first trial we train the model with the pre-processed dataset but without augmenting it. Network performance is definitely poor, as it could be inferred both from metrics values and from density maps appearance. MAE and RMSE details can be found in table 2

Therefore, for any subsequent runs, we train our network with the augmented dataset, prepared as exposed in 4.3

Type of dataset	Part A		Part B	
	MAE	RMSE	MAE	RMSE
No data augm	224.6	335.6	22.1	35.1
Data augm	137.1	202.5	18.3	25.2

Table 2. MAE and RMSE metrics before and after data augmentation

5.2.3 Enhancing factor k

We try different values for the enhancing factor k. What we notice is that values lower than 64 does not produce any good result. In particular, without scaling (so having set $k = 1$), the model is not able to produce any even interpretable output.

Then, we test also higher values as 200, 500, 1000 but we do not encounter an increase in performance. We attribute this effect to the fact that now irrelevant pixels confuse the model since they assume too high values. The selected optimal k is then 64.

Values for k	Part A		Part B	
	MAE	RMSE	MAE	RMSE
1	1226.1	1259.6	875.9	881.0
64	97.3	148.0	16.6	27.7
500	164.5	236.9	17.7	24.5

Table 3. MAE and RMSE metrics for different values of k

5.2.4 Kernel initialization

Without ad-hoc weights initialization, the model is not able to obtain significant results. We try common-use initializing

distribution and we opt for a Gaussian kernel $\mathcal{N}(0, 10^{-2})$ as we found in the literature on similar problems and nets[2].

5.2.5 Values for Gaussian kernels

The average distance parameter σ turns out to be critically affecting the network performance. For Part B, we set $\sigma = 15$, as reported in the literature[3]. Due to higher density, a lower value of σ is needed in Part A. After several trials, we set it to 3.

5.2.6 Kernel regularization

As refinement to prevent over-fitting, we test both L_1, L_2 and mixed $L_1 - L_2$ penalties. We find the best performance increase with $L_2(10^{-4})$ regularization.

5.2.7 Dilation rate

We have already mentioned the power of dilated conv filters in 4.4. After testing our baseline without dilation by setting $d = 1$, we test also $d = 2, d = 3$, whose effect can be visualized in 3. The best performance, while keeping the other parameters fixed (to the one exposed in 5.3), is achieved by d=2. MAE values for these configuration are reported in table 4.

Dilation rate	Part A		Part B	
	MAE	RMSE	MAE	RMSE
1	111.1	166.5	18.5	26.7
2	88.4	130.5	12.8	20.2
3	122.8	184.2	24.8	36.2

Table 4. MAE and RMSE metrics for dilation rate values $d = 1, 2, 3$

5.3. Best Model

After the phase of the experiments, we end up having the model parameters tuned to their optimal value. In particular, we now include dilation, with $d = 2$.

Gradient descent is again performed on batches of size 16 via Adam optimizer, with an adaptive learning rate, with starting value of 10^{-6}. The activation function is ReLU.

Moreover, kernel regularitazion has been implemented via L_2 regularizer with penalty 10^{-4}. Initial weights are drawn from a Gaussian distribution $N(0, 10^{-2})$.

The pre-processing and data augmentation tasks are the same described in 5.2.1, but now $k = 64$. The model has been trained for 100 epochs.

The metrics values for this model are the following:

- on Part A, MAE=88.4 and RMSE=130.5,
- on Part B, MAE=12.8 and RMSE=20.2.

In table 5, the comparison of our results with related literature can be found.

Finally, the predictive capability of our network can be visualized in fig. 5 and fig. 6. They display the forecasted density maps for some images from, respectively, Part A and Part B.

Method	Part A		Part B	
	MAE	RMSE	MAE	RMSE
Zhang et al. [7]	181.8	277.7	32.0	49.8
MCNN [8]	110.2	173.2	26.4	41.3
ours	88.4	130.5	12.8	20.2
CSRNet [6]	68.2	115.0	10.6	16.0
Rong et al. [3]	60.1	95.5	7.9	11.7

Table 5. Comparison with the literature

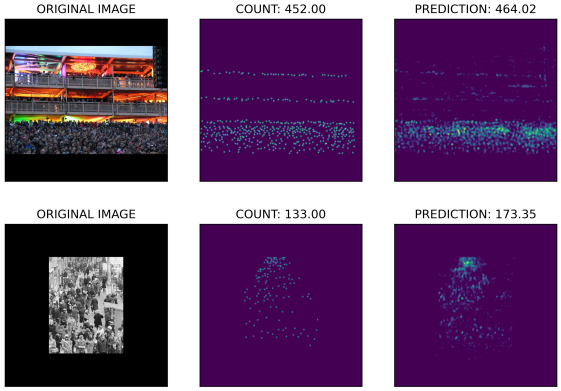

Figure 5. Some results on part A

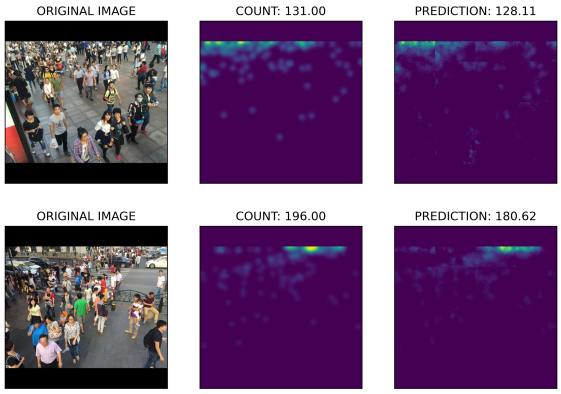

Figure 6. Some results on part B

6. Conclusion

In this report, we exhibited in broad terms our steps into the development of a deep convolutional neural network for automatic crowd counting via density map generation. We have been able to get good performance, close to the state-of-the-art best results while keeping our architecture feasible for training on a basic user hardware device. This provides a good starting point for discussion and further research, in particular for the generalization of our architecture to the counting of other objects categories.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Yuhong Li, Xiaofan Zhang, and Deming Chen. Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes, 2018.

[3] Liangzi Rong and Chunping Li. A strong baseline for crowd counting and unsupervised people localization. *CoRR*, abs/2011.03725, 2020.

[4] Manar Salamah Ali Salma Kammoun Jarraya, Maha Hamdan Alotibi. A deep-cnn crowd counting model for enforcing social distancing during covid19 pandemic: Application to saudi arabia's public places. *Computers, Materials & Continua*, 66(2):1315–1328, 2021.

[5] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[6] Qian Wang and Toby P. Breckon. Segmentation guided attention network for crowd counting via curriculum learning. *CoRR*, abs/1911.07990, 2019.

[7] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[8] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-image crowd counting via multi-column convolutional neural network. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 589–597, 2016.