Crowd Counting by Adaptively Fusing Predictions from an Image Pyramid

Di Kang
dkang5-c@my.cityu.edu.hk
Antoni Chan
abchan@cityu.edu.hk

Department of Computer Science
City University of Hong Kong
Hong Kong SAR

Abstract

Because of the powerful learning capability of deep neural networks, counting performance via density map estimation has improved significantly during the past several years. However, it is still very challenging due to severe occlusion, large scale variations, and perspective distortion. Scale variations (from image to image) coupled with perspective distortion (within one image) result in huge scale changes of the object size. Earlier methods based on convolutional neural networks (CNN) typically did not handle this scale variation explicitly, until Hydra-CNN and MCNN. MCNN uses three columns, each with different filter sizes, to extract features at different scales. In this paper, in contrast to using filters of different sizes, we utilize an image pyramid to deal with scale variations. It is more effective and efficient to resize the input fed into the network, as compared to using larger filter sizes. Secondly, we adaptively fuse the predictions from different scales (using adaptively changing per-pixel weights), which makes our method adapt to scale changes within an image. The adaptive fusing is achieved by generating an across-scale attention map, which softly selects a suitable scale for each pixel, followed by a 1x1 convolution. Extensive experiments on three popular datasets show very compelling results.

1 Introduction

Automatic analysis of crowded scenes from images has important applications in crowd management, traffic control, urban planning, and surveillance, especially with the rapidly increasing population in major cities. Crowd counting methods can also be applied to other fields, e.g., cell counting [13], vehicle counting [10, 18], animal migration surveillance [1].

The number and the spatial arrangement of the crowds are two types of useful information for understanding crowded scenes. Methods that can simultaneously count and predict the spatial arrangement of the individuals are preferred, since situations where many people are crowded into a small area are very different from those where the same number of people are evenly spread out. Explicitly detecting every person in the image naturally solves the counting problem and estimates the crowd's spatial information as well. But detection performance decreases quickly as occlusions between objects become severe and as object size becomes smaller. In order to bypass the hard detection problem, regression-based methods [2, 7, 22] were developed for the crowd counting task, but did not preserve the spatial information, limiting their usefulness for further crowd analysis, such as detection and tracking.

Object density maps, originally proposed in [13], preserve both the count and spatial arrangement of the crowd, and have been shown effective at object counting [9, 13, 18, 23, 25, 30, 31], people detection [10, 16] and people tracking [10, 20, 21] in crowded scenes. In an object density map, the integral over any sub-region is the number of objects within that corresponding region in the image. Density-based methods are better at handling cases where objects are severely occluded, by bypassing the hard detection of every object, while also maintaining some spatial information about the crowd.

Crowd counting is very challenging due to illumination change, severe occlusion, various background, perspective distortion and scale variation, which requires robust and powerful features to overcome these difficulties. The current state-of-the-art methods for object counting use deep learning to estimate crowd density maps [9, 10, 18, 23, 25, 28, 30, 31]. In contrast to image classification, where the main objects are roughly centered, cover a large portion of the image, and exhibit no severe scale variations, in crowd counting, the people appear at very different scales in the same image due to the orientation of the camera. The detection and image segmentation tasks often require some special designs intending to better handle scale variation, such as image pyramids [3, 6, 18], multi-scale feature fusion [14, 19, 23, 31] or special network layers, such as spatial transformer layer [8] and deformable convolution layer [4]. For the counting task, several methods for handling large scale variations have been proposed. Patch size normalization [30] resizes each image patch so that the objects inside become roughly a fixed scale, but requires the image perspective map. With Hydra-CNN [18], the image patch along with its smaller center crops are all resized to the same fixed size, and fed into different columns of the Hydra-CNN. In [23, 25, 31], multi-column CNN (MCNN) uses different filter sizes for each column to extract multi-scale features. ACNN [9] handles scale variation by dynamically generating convolution filter weights according to side information, such as perspective value, camera angle and height.

In this paper, we consider an alternative approach to handling multiple scales based on image pyramids. We first design a backbone fully convolutional network (FCN) with a reasonable receptive field size and depth, serving as a strong baseline. Secondly, to handle scale variations within the same image, we construct an image pyramid of the input image and run each image through the FCN to obtain predicted density maps at different scales. Finally, the density map predictions at different scales are fused adaptively at every pixel location, which allows switching predictions between scales within the same image. The adaptive fusion is obtained by using a sub-network to predict an across-scale attention map [3], followed by a 1×1 convolution. Our method is tested on three popular crowd counting datasets and shows very compelling performance, while having faster than real-time performance.

2 Related works

Most of the recent counting methods adopt object density maps and deep neural networks, due to the counting accuracy and preservation of spatial information in the density map, and the powerful learning capability of deep neural networks. [10, 18, 28, 30] use Alexnet-like [12] networks including convolution layers and fully-connected layers. CNN-pixel [10] only predicts the density value of the patch center at one time, resulting in a full-resolution density map, but at the cost of slow prediction. [18, 28, 30] predict a density patch reshaped from the final fully-connected layer, resulting in much faster prediction. However, in order to avoid huge fully-connected layers, only density maps with reduced resolution are produced, and artifacts exist at the patch boundaries. Starting from [15], fully convolutional network (FCN) becomes very popular for dense prediction tasks such as semantic segmentation [15],

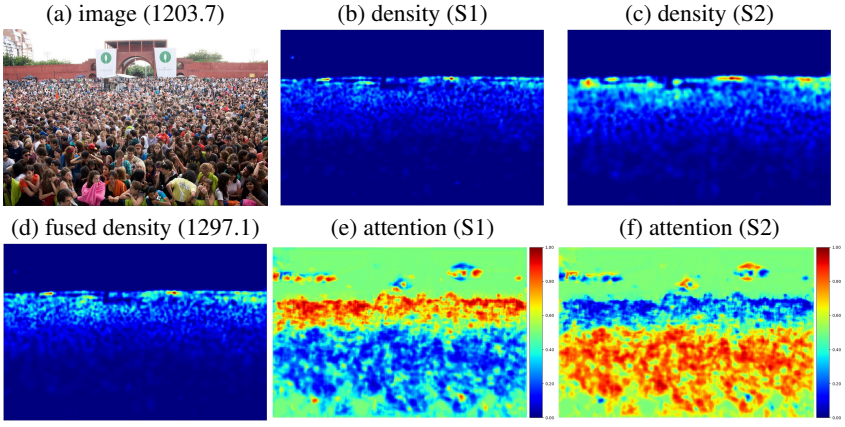

Figure 1: Example of our image pyramid CNN (FCN-7c-2s) showing the predicted attention maps (e, f), density predictions (1-channel feature map) of each scale (b, c) and the final fused density map (d). Numbers inside the parenthesis are the GT density and predicted density. Best viewed in color.

edge detection [29], saliency detection [5] and crowd counting [31] because FCN is able to reuse shared computations, resulting in much faster prediction speed compared to sliding window-based prediction.

Scale variation (from image to image) coupled with perspective distortion (within one image) is one major difficulty faced by counting tasks, especially for cross-scene scenarios. To overcome this issue, [30] resizes all the objects to a canonical scale, by cropping image patches whose sizes vary according to perspective value, and then resizing them to a fixed size as the input into their network. However, there is no way to handle intra-patch scale variation for this patch normalization method, and the perspective map needs to be known. Hydra-CNN [18] takes the image patch and several of its smaller center crops, and then upsamples them to the same size before inputing into each of the CNN columns – the effect is to show several zoomed in crops of the given image patch. Since the input image contents are different, feature maps extracted from different inputs cannot be spatially aligned. Hence, Hydra-CNN uses a fully-connected layer to fuse information across scales. In contrast to Hydra-CNN, our proposed method is based on an image pyramid, where each scale contains the whole image at lower resolutions. Hence, the feature maps predicted for each scale can be aligned after upsampling, and a simple fusion using 1x1 convolution can be used.

[9] uses side information, such as perspective, camera angle and height, as input to adaptively generate convolution weights to adapt feature extraction to the specific scene/perspective. However, side information is not always available. MCNN [31] uses 3 columns of convolution layers with the same depth but with different filter sizes. Different columns should respond to objects at different scales, although no explicit supervision is used. MCNN is adopted by several later works [23, 25]. Switch-CNN [23] relies on a classification CNN to classify the image patches to one of the columns to make sure every column is adapted to a particular scale. CP-CNN [25] includes MCNN as a part of their network and further concatenates its feature maps with local and global context features from classification networks.

Despite the success of MCNN, we notice three major issues of MCNN. Firstly, in order to get a larger receptive field (with fixed depth), 9×9 filters are used, which is not as effective as smaller filters [24, 26, 27]. We also experimentally show that a network with similar receptive field size but using larger filters (e.g. 7×7 vs 5×5) gives worse performance (see Table 7). Secondly, since images/image patches containing objects of all scales are used to

train the network, it is highly likely that every column still responds to all scales, which turns MCNN into an ensemble of several weak regressors (but extracting multi-scale features). To alleviate this problem, [23] introduces a classification network to assign image patches to one of the three columns, so that only one column is activated every time and trained for a particular scale only. However, the classification accuracy is limited and the true label of an image patch is unclear. Another issue is that the classification CNN makes a hard decision, resulting in Switch-CNN extracting features from only one scale, thus ignoring other scales that might provide useful context information. Thirdly, MCNN uses 1×1 convolution layer to fuse features from different columns, which is essentially a weighted sum of all the input maps. However, the weights are the same for all spatial locations, which means the scale with the largest weight will always dominate the prediction, regardless of the image input. However, to handle scale variations within an image requires each pixel to have its own set of weights to select the most appropriate scale. This intra-image variation is partially handled by Switch-CNN [23] by dividing the whole image into several smaller image patches, which then are processed by a certain column according to the classification result.

3 Methodology

To overcome the issues mentioned above, our solution is to adaptively fuse features from an image pyramid. Instead of increasing the filter size to see more image content, we send the downsampled images into an FCN with a fixed filter size. To some extent, this is equivalent to using larger filter size that can cover larger receptive field size, but with less computation and less issues associated with large filters. Unlike [18], our image pyramid contains the same image content but at different resolutions. Hence, feature maps from different columns can be spatially aligned by upsampling, which is required for fusion by convolution layers.

Due to scene perspective, the scales of the objects changes based on the location in the image. Hence, different fusing weights are required for every single location of the feature map. To this end, we estimate an attention map for each scale, which is used to adaptively fuse the predictions. A similar method is also used in [3] for semantic segmentation. Experimentally, we find applying attention map on density predictions from different scales is better than applying on feature maps (see Table 7). Our proposed network is shown in Fig. 2.

3.1 Backbone FCN

Given an input image, an image pyramid is generated by downsampling the image to other scales. Each scale is then input into a backbone FCN to predict a density map for that scale. The backbone FCN uses shared parameters for all scales.

Our backbone FCN contains two max-pooling layers, which divides the whole FCN into three stages and results in predicted density maps downsampled by 4. The number of convolution layers depends on the general object size of the dataset (7 for ShanghaiTech and 5 for UCSD). Only two pooling layers are used because the resolution of the density map is very important, and it is not easy to compensate for the lost spatial information in overly downsampled density maps [10]. With only two pooling layers (limited stride size), we use 5×5 filters in order to get large enough receptive field size. For example, the FCN-7c used for ShanghaiTech and WorldExpo dataset has a receptive field size of 76. Table 1 lists the layer configurations. The final prediction layer uses ReLU activation, while the other convolution layers use leaky ReLU [17] activation (slope of 0.1 for the negative axis).

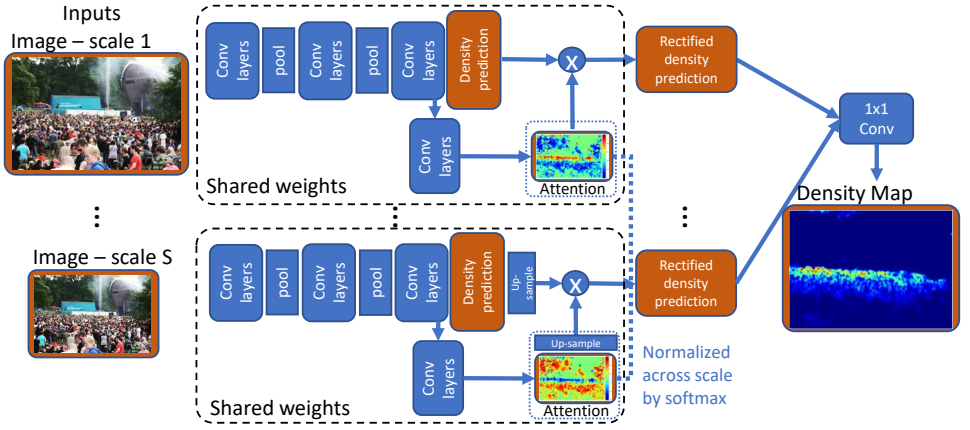

Figure 2: Our image pyramid CNN for crowd counting. Each scale of the image pyramid is sent into an FCN to predict a density map for that scale. At each scale, the attention sub-net takes as input the last feature map and predicts an attention map. An across-scale softmax is applied to all the attention maps, which is then multiplied to the density maps at each scale. Finally, a 1x1 convolution fuses the density maps from all scales. The FCN and attention sub-net weights are shared across all scales.

FCN-7c		FCN-5c	
Layer	Filter	Layer	Filter
1	16x1x5x5	1	16x1x5x5
2	16x16x5x5	-	-
max-pooling	2x2	max-pooling	2x2
3	32x16x5x5	2	32x16x5x5
4	32x32x5x5	-	-
max-pooling	2x2	max-pooling	2x2
5	64x32x5x5	3	64x32x3x3
6	32x64x5x5	4	32x64x3x3
7	1x32x5x5	5	1x32x3x3

Table 1: Our FCN-7c and FCN-5c networks. The four numbers in the Filter columns represent output channels, input channels and filter size (H, W). Bias term is not shown.

Layer	Filter
1	8x32x3x3
2	1x8x1x1
upsampling	-
across-scale softmax	-
3 (fusion)	1xSx1x1

Table 2: Configuration of our attention subnet, and subsequent layer to fuse predictions from all scales into one. S is the number of scales used.

3.2 Attention map network and fusion

Similar to [4], a network containing only two convolution layers is used to generate attention weights from its precedent feature maps, summarizing the information into a 1-channel attention map (see Table 2 for details). The feature maps from the last convolution layer (e.g. conv-6 output for FCN-7c) are used to generate the attention map because they are more semantically meaningful than the low-level features, which respond to image cues.

To perform fusion, the attention maps and the density maps are bilinearly upsampled to the original resolution. Next, an across-scale softmax normalization is applied on the attention maps so that, at every location, the sum of the attention weights is 1 across scales. At each scale, the corresponding normalized attention map is element-wise multiplied with the density map prediction for that scale, resulting in a rectified density map where off-scale regions are down-weighted. Finally, a 1×1 convolution layer is applied to fuse the rectified density maps from all scales into a final prediction. Note that the density predictions (1-channel feature map) from each scale may not strictly correspond to the ground-truth density map, since the fusion step uses a 1×1 convolution with bias term and its weights are not normalized.

Method	Part A		Part B	
	MAE	RMSE	MAE	RMSE
CNN-patch [30]	181.8	277.7	32.0	49.8
MCNN [31]	110.2	173.2	26.4	41.3
FCN-7c (ours)	82.3	124.7	12.4	20.5
FCN-7c-2s (ours)	81.3	132.6	10.9	19.1
FCN-7c-3s (ours)	**80.6**	126.7	**10.2**	18.3
Switch-CNN [23]	90.4	135.0	21.6	33.4
CP-CNN [25]	**73.6**	106.4	20.1	30.1

Table 3: Test errors on the ShanghaiTech dataset. "2s" and "3s" indicate using 2 or 3 different scales in the image pyramid. Switch-CNN and CP-CNN both use pre-trained VGG [24] network.

Method	S1	S2	S3	S4	S5	Avg.
CNN-patch[30]	9.8	14.1	14.3	22.2	3.7	12.9
MCNN [31]	3.4	20.6	12.9	13.0	8.1	11.6
FCN-7c (ours)	2.4	15.6	12.9	28.5	5.8	13.0
FCN-7c-2s (ours)	2.0	13.4	12.1	26.5	3.3	11.5
FCN-7c-3s (ours)	2.5	16.5	12.2	20.5	2.9	10.9
Switch CNN [23]	4.2	14.9	14.2	18.7	4.3	11.2
CP-CNN [25]	2.9	14.7	10.5	10.4	5.8	8.9

Table 4: MAE on WorldExpo dataset. Only results using the same ground truth densities as the original paper [30] are included. Other methods are not comparable since they use different ground-truth counts due to different ground-truth densities.

3.3 Training details

The whole network is trained end-to-end. The training inputs are 128×128 grayscale randomly cropped image patches (for the original scale), and outputs are 32×32 density patches (every pixel value is the sum of a 4×4 region on the original density patch). The loss function is the per-pixel MSE between the predicted and ground-truth density patches. Since the density value is a small fractional number (e.g. 0.001), we multiply it by 100 during training. SGD with momentum (0.9) is used for training. Occasionally, when SGD fails to converge, we use Adam [11] optimizer for the first few epochs and then switch to SGD with momentum. Learning rate (initially 0.001) decay and weight decay are used. During prediction, the whole image is input into the network to predict a density map downsampled by a factor of 4.

4 Experiment

We test our method on three crowd counting datasets: ShanghaiTech, WorldExpo, and UCSD.

4.1 Evaluation metric

Following the convention of previous works [2, 30], we compare different methods using mean absolute error (MAE), and mean squared error (MSE) or root MSE (RMSE),

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{c}_i - c_i|,\ \text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{c}_i - c_i)^2,\ \text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{c}_i - c_i)^2}, \quad (1)$$

where c_i is the ground truth count, which is either an integer number (for UCSD) or fractional number (for other datasets), and $\hat{c}_i$ is the predicted total density which is a fractional number.

4.2 ShanghaiTech dataset

The ShanghaiTech dataset contains two parts, A and B, tested separately. Part A contains 482 images randomly crawled online, with different resolutions, among which 300 images are used for training and the rest are for testing. Part B contains 716 images taken from busy streets of the metropolitan areas in Shanghai, with a fixed resolution of 1024×768. 400 images are used for training and 316 are for testing. Following [31], we use geometry-adaptive kernels (with k=5) to generate the ground-truth density maps for Part A, and isotropic Gaussian with fixed standard deviation (std=15) to generate the ground-truth density maps for Part B. We test our baseline FCN, denoted as FCN-7c, as well as 2 versions using image pyramids, FCN-7c-2s uses 2 scales $\{1.0, 0.7\}$ and FCN-7c-3s uses 3 scales $\{1.0, 0.7, 0.5\}$.[1]

Results are listed in Table 3. Our FCN-7c is a strong baseline, and adaptively fusing predictions from an image pyramid (FCN-7c-2s and FCN-7c-3s) further improves the perfor-

[1] Scale 0.7 indicates the image width and height are only 70% of the original size.

mance. On Part B, our method achieves much better result than other methods. On Part A, our method achieves second best performance, with only CP-CNN performing better. Note that CP-CNN uses a pre-trained VGG network and performs per-pixel prediction with a sliding window, which is very slow for high-resolution images, running at 0.11 fps (9.2 seconds per image). In contrast, FCN-7c and FCN-7c-3s run at 439 fps and 158 fps (see Table.7).

Fig. 1 shows an example results and attention maps from our FCN-7c-2s. Since scale 1 (S1) takes as input the original image, its features are suitable (higher weight on attention map) for smaller objects. Scale 2 (S2) takes as input the lower resolution image, and responds to the originally larger objects that are now resized down. More demo images and predictions can be found in the supplemental.

4.3 WorldExpo dataset

The WorldExpo'10 dataset [30] is a large scale crowd counting dataset captured from the Shanghai 2010 WorldExpo. It contains 3,980 annotated images from 108 different scenes. Following [30], we use a human-like density kernel, which changes with perspective, to generate the ground-truth density map. Models are trained on the 103 training scenes and tested on 5 novel test scenes. The network setting is identical to those used for ShanghaiTech.

Results are listed in Table 4. Adaptive fusion of scales (FCN-7c-2s and FCN-7c-3s) improves the performance over the single scale FCN-7c, and achieves slightly better performance than MCNN and Switch-CNN. Switch-CNN uses pre-trained VGG weights to fine-tune their classification CNN. CP-CNN, taking global and local context information into consideration, performs the best. CP-CNN also uses pre-trained VGG weights when extracting global context features. Demo images and predictions can be found in the supplemental.

4.4 UCSD dataset

The UCSD dataset is a low resolution (238×158) surveillance video of 2000 frames, with perspective change and heavy occlusion between objects. We test the performance using the traditional setting [2], where all frames from 601-1400 are used for training and the remaining 1200 frames for testing. The ground-truth density maps use the Gaussian kernel with $\sigma = 4$. Since the largest person in UCSD is only about 30 pixels tall, our FCN backbone network uses five convolution layers, denoted as FCN-5c (see Table 1).

Results are listed in Table 5. Again, our adaptive fusion improves the performance over FCN-5c, even on this single scene dataset without severe scale variation. Switch-CNN performs worse than either our method or MCNN on UCSD. One possible reason is that the object size is too small compared to ImageNet, and the limited training samples cannot fine-tune their classification CNN well. See supplemental for our predicted density maps.

4.5 Summary

Overall, MCNN [31], Switch-CNN [23], CP-CNN [25] and our proposed method using three scales (FCN-x-3s) perform the best on the tested datasets. We summarize their performance rank and runtime speed in Table 6. Overall, our method has better average rank than MCNN [31] and Switch-CNN [23], but is worse than CP-CNN. Switch-CNN [23] and CP-CNN [25] both use pre-trained VGG model and do not run in a fully convolutional fashion, resulting in slower prediction. CP-CNN is especially slow because it uses sliding-window per-pixel prediction to get the local context information. In contrast, MCNN [31] and our FCN-x-3s are fully convolutional, resulting in faster than real-time prediction. They also use much smaller models and do not need extra data to pre-train the models (c.f., VGG).

Method	MAE	MSE
CNN-patch [30]	1.60	3.31
MCNN [31]	**1.07**	1.82*
CNN-boost [28]	1.10	-
CNN-pixel [10]	1.12	2.06
FCN-5c (ours)	1.28	2.65
FCN-5c-2s (ours)	1.22	2.33
FCN-5c-3s (ours)	1.16	2.29
Switch-CNN [23]	1.62	4.41*
Hydra 3s [18] ("max")	2.17	-
ACNN [9] ("max")	0.96	-

Table 5: Test errors on the UCSD dataset when using the whole training set. "max" means the methods are trained on the downsampled training set, where only one out of 5 frames is used for training. * [31] and [23] report RMSE of 1.35 and 2.10.

	Rank					
Method	Shanghai Tech A	Shanghai Tech B	WorldExpo	UCSD	Average	Runtime (fps)
MCNN [31]	4	4	4	1	3.25	307
Switch-CNN [23]	3	3	3	3	3.00	12.9
CP-CNN [25]	1	2	1	-	1.33	0.11
Ours (FCN-x-3s)	2	1	2	2	1.75	158

Table 6: Comparison of the four most competitive methods. All runtimes are using PyTorch 0.3.0 on a GeForce GTX 1080 on 1024×768 images. Methods without published code were built (without training) and speed tested in the prediction stage.

			Part A		Part B		
Method	RF	# parameters	MAE	RMSE	MAE	RMSE	runtime (fps)
FCN-7c (ours)	76	148,593	82.3	124.7	12.4	20.5	439
FCN-7c-40	40	162,337	106.0	153.4	19.8	33.5	433
FCN-7c-40 (scale 0.7)	40	162,337	97.2	155.9	17.1	23.9	-
FCN-5c-78	78	178,369	101.8	137.9	15.5	26.8	494
FCN-14c-76	76	178,369	83.4	142.4	12.7	19.0	329
FCN-7c-3s (ours)	-	150,918	80.6	126.7	10.2	18.3	158
FCN-7c-3s (fixed)	-	148,597	85.8	135.8	10.3	16.7	-
FCN-7c-3s (w/o softmax)	-	150,918	87.5	130.0	11.0	16.6	-
FCN-7c-3s (sum)	-	150,914	83.8	120.1	10.4	16.8	-
FCN-7c-3s (low)	-	149,182	85.6	133.8	11.0	19.2	-
FCN-7c-3s (feat)	-	150,210	82.0	126.6	11.1	19.0	-
FCN-7c-2s (ours)	-	150,917	81.3	132.6	10.9	19.1	234
FCN-7c-2s (fixed)	-	148,596	86.6	130.8	*	*	-
FCN-7c-2s (w/o softmax)	-	150,917	84.3	132.2	11.0	18.5	-
FCN-7c-2s (sum)	-	150,914	82.8	123.4	11.3	19.4	-
FCN-7c-2s (low)	-	149,181	82.8	129.4	12.2	19.0	-
FCN-7c-2s (feat)	-	150,178	90.0	137.5	12.4	22.0	-

Table 7: Comparison of different variants on ShanghaiTech dataset. RF is the receptive field size. * indicates that the network failed to converge in limited trials.

4.6 Ablation study and discussion

In this section, we discuss the design of our model with ablation studies. All the ablation study results tested on ShanghaiTech are summarized in Table 7, and Table 8 for UCSD.

4.6.1 Backbone FCN design

We find that receptive field (RF) plays a very crucial role in the density estimation task. The RF needs to be large enough so that enough portion of the object is visible to recognize it. On the other hand, if the RF is too large, then too much irrelevant context information will distract the network. Hence, the most suitable RF size should be related to the average object size of the dataset. We conduct several experiments to show the importance of the RF.

Receptive field size: On ShanghaiTech dataset, we test another FCN with 7 convolution layers whose filter sizes are 3×3 for all the convolution layers, resulting in RF size only equal to 40 (denoted as FCN-7c-40). In order to make a fair comparison, we increase the channel numbers to balance the total parameters. Specifically, the 7 convolution layers now use $\{32, 32, 64, 64, 96, 48, 1\}$ filter channels respectively now. Although the density values are

mostly assigned to head and shoulder regions, but considering the image size (e.g. 1024×768 for ShanghaiTech B), an RF of only 40 is too small, and only achieves 106.0 MAE on part A and 19.8 MAE on part B. However, if this FCN-7c-40 model is trained and tested on scale 0.7, its performance increases to 97.2 MAE on part A and 17.1 MAE on part B.

On UCSD, we test two variants: 1) FCN-5c with filter size of $\{5,5,5,5,5\}$, resulting in RF size of 64 (denoted as FCN-5c-64); 2) FCN-7c used for ShanghaiTech/WorldExpo whose RF size is 76. These two models have more capacity (trainable parameters) than the proposed FCN-5c but achieve worse MAE (1.40 for FCN-5c-64 and 1.54 for FCN-7c), since an RF size of 64 or 76 is too large on UCSD where the largest object is only about 30 pixels tall.

Filter size: Smaller filter size is preferred since it saves parameters and computation [24, 26, 27]. Here we experimentally show that the 7×7 filter used in MCNN is not the best choice. We test a FCN with 5 convolution layers, whose filter sizes are $\{7,7,7,5,5\}$ respectively, resulting in RF size equal to 78 (denoted as FCN-5c-78) on ShanghaiTech. A substantial performance drop is observed (101.8 MAE on part A and 15.5 MAE on part B).

Since 3×3 filters is most commonly used, we replace the 5×5 filters in our FCN-7c with 2 convolution layers using 3×3 filters. In order to make a fair comparison, we increases the filter number to balance the total trainable parameters. The resulting FCN-14c-76 uses $\{24,24,24,24,32,32,32,32,64,64,32,32,32,1\}$ filters respectively, in total 143,225 parameters (FCN-7c uses 148,593 parameters). It achieves 83.4 MAE on part A and 12.7 on part B (see Table 7). Since no performance improvement is observed, we would prefer to use our FCN-7c, considering that deeper network are normally more difficult to train and FCN-14c-76 runs slower than FCN-7c during prediction stage (329 fps vs 439 fps on part B).

4.6.2 Fusion design

Here we consider different variants of fusing two scales. We test simple 1×1 convolution without attention-map adaptivity to fuse predictions from different scales (denoted as "FCN-7c-2s (fixed)"), similar to MCNN [31]. On ShanghaiTech, it only achieves 86.6 MAE on part A and failed to converge on part B, showing the necessity of adaptivity during fusion. We also test 2 other variations of our network: 1) without across-scale softmax normalization (denoted as "FCN-7c-2s (w/o softmax)"); 2) using simple summation to fuse the density predictions instead of 1×1 convolution layer (denoted as "FCN-7c-2s (sum)"). These variants also give worse MAE than our FCN-7c-2s.

To generate the attention map, we consider using feature maps from other convolution layers, such as the last layer of stage 1. Since its resolution is higher, we insert two average pooling layer after both the convolution layers in the attention sub-network, denoted as "FCN-7c-2s (low)". It gives worse performance than FCN-7c. One possible reason is that lower layer features, although containing more spatial information, are semantically weak, e.g. more similar to edge features. However, the attention map requires high-level abstraction to distinguish the foreground objects at the desired scale from the background.

Other than fusing the density predictions from different scales, similar to MCNN [31], we can also concatenate the feature maps and then predict a final density map. In this method denoted as "FCN-7c-2s (feat)", the generated attention map from the last convolution layer of stage 3 is applied on its output, similar to spatial transformer [8]. However, it does not perform as well as our density fusion.

5 Conclusion

By taking into consideration the design of backbone FCN and the fusion of predictions from an image pyramid, our proposed adaptive fusion image pyramid counting method achieves

Method	RF	# parameters	MAE	MSE
FCN-5c (ours)	40	50,497	1.28	2.65
FCN-5c-64	64	116,545	1.40	3.16
FCN-7c	76	148,593	1.54	3.80
FCN-5c-2s (ours)	-	52,821	1.22	2.33
FCN-5c-3s (ours)	-	52,822	1.16	2.29

Table 8: Comparison of different variants on UCSD dataset.

better average ranking on 4 datasets than MCNN and Switch-CNN. Although CP-CNN has better ranking than our proposed method, it is much slower since it uses both a very deep network (VGG) and sliding window per-pixel prediction. Taking into consideration of model size and running time, our method is the most favorable, especially for cases requiring real-time prediction.

Acknowledgement: This work was supported by grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project No. [T32-101/15-R] and CityU 11212518), and by a Strategic Research Grant from City University of Hong Kong (Project No. 7004887). We grateful for the support of NVIDIA Corporation with the donation of the Tesla K40 GPU used for this research.

References

[1] Carlos Arteta, Victor Lempitsky, and Andrew Zisserman. Counting in The Wild. In *ECCV*, 2016.

[2] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *CVPR*, 2008.

[3] Liang-Chieh Chen, Yi Yang, Jiang Wang, Wei Xu, and Alan L. Yuille. Attention to Scale: Scale-aware Semantic Image Segmentation. In *CVPR*, 2016.

[4] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *ICCV*, 2017.

[5] Qibin Hou, Ming-Ming Cheng, Xiao-Wei Hu, Ali Borji, Zhuowen Tu, and Philip Torr. Deeply supervised salient object detection with short connections. In *CVPR*, 2017.

[6] Peiyun Hu and Deva Ramanan. Finding Tiny Faces. In *CVPR*, 2017.

[7] Haroon Idrees, Imran Saleemi, Cody Seibert, and Mubarak Shah. Multi-source multi-scale counting in extremely dense crowd images. In *CVPR*, 2013.

[8] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial Transformer Networks. In *NIPS*, 2015.

[9] Di Kang, Debarun Dhar, and Antoni B Chan. Incorporating Side Information by Adaptive Convolution. In *NIPS*, 2017.

[10] Di Kang, Zheng Ma, and Antoni B. Chan. Beyond Counting: Comparisons of Density Maps for Crowd Analysis Tasks - Counting, Detection, and Tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[11] Diederik P. Kingma and Jimmy Lei Ba. Adam: a Method for Stochastic Optimization. In *ICLR*, 2015.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[13] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. In *NIPS*, 2010.

[14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. In *CVPR*, 2017.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[16] Zheng Ma, Lei Yu, and Antoni B Chan. Small Instance Detection by Integer Programming on Object Density Maps. In *CVPR*, 2015.

[17] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML*, 2013.

[18] Daniel Onoro-Rubio and Roberto J López-Sastre. Towards perspective-free object counting with deep learning. In *ECCV*, 2016.

[19] Pedro O. Pinheiro, Tsung-yi Yi Lin, Ronan Collobert, Piotr Doll, and Piotr Doll??r. Learning to Refine Object Segments. In *ECCV*, 2016.

[20] Weihong Ren, Di Kang, Yandong Tang, and Antoni Chan. Fusing crowd density maps and visual object trackers for people tracking in crowd scenes. In *CVPR*, 2017.

[21] Mikel Rodriguez, Ivan Laptev, Josef Sivic, and Jean-Yves Yves Audibert. Density-aware person detection and tracking in crowds. In *ICCV*, 2011.

[22] David Ryan, Simon Denman, Clinton Fookes, and Sridha Sridharan. Crowd counting using multiple local features. In *Digital Image Computing: Techniques and Applications*, 2009.

[23] Deepak Babu Sam, Shiv Surya, and R. Venkatesh Babu. Switching Convolutional Neural Network for Crowd Counting. In *CVPR*, pages 5744–5752, 2017.

[24] Karen; Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *ICLR*, 2015.

[25] Vishwanath A. Sindagi and Vishal M. Patel. Generating High-Quality Crowd Density Maps using Contextual Pyramid CNNs. In *ICCV*, pages 1861–1870, 2017.

[26] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *CVPR*, 2015.

[27] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, 2016.

[28] Elad Walach and Lior Wolf. Learning to count with CNN boosting. In *ECCV*, 2016.

[29] Saining Xie and Zhuowen Tu. Holistically-Nested Edge Detection. In *ICCV*, 2015.

[30] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene Crowd Counting via Deep Convolutional Neural Networks. In *CVPR*, 2015.

[31] Yingying Zhang, Desen Zhou, Siqin Chen, Shenghua Gao, and Yi Ma. Single-Image Crowd Counting via Multi-Column Convolutional Neural Network. In *CVPR*, 2016.