One-Shot Scene-Specific Crowd Counting

Mohammad Asiful Hossain
hossaima@cs.umanitoba.ca
University of Manitoba
Winnipeg, Manitoba
Canada

Mahesh Kumar K
kumarkm@cs.umanitoba.ca
University of Manitoba
Winnipeg, Manitoba
Canada

Mehrdad Hosseinzadeh
mehrdad@cs.umanitoba.ca
University of Manitoba
Winnipeg, Manitoba
Canada

Omit Chanda
omitum@cs.umanitoba.ca
University of Manitoba
Winnipeg, Manitoba
Canada

Yang Wang
ywang@cs.umanitoba.ca
University of Manitoba
Winnipeg, Manitoba
Canada

Abstract

We consider the problem of crowd counting in static images. Given an image, the goal is to estimate a density map of this image, where each value in the density map indicates the density level of the corresponding location in the image. In particular, we consider a novel problem setting which we call the one-shot scene-specific crowd counting. During training, we assume that we have labeled images collected from different scenes. Each scene corresponds to a camera at a fixed location and angle. Given a target scene, we assume that we have one single labeled image collected from that scene. Our goal is to adapt the crowd counting model to this specific scene based on this single example. We argue that this setting is more realistic in terms of deploying crowd counting algorithms in real-world applications. We propose a novel one-shot learning approach for learning how to adapt to a target scene using one labeled example. Our experiment results demonstrate that our proposed approach outperforms other alternative methods.

1 Introduction

We consider the problem of crowd counting in static images via predicted density map. Given an image of a crowded scene, the goal is to predict a density map which has the same spatial dimension of the input. Each pixel of the density map indicates the crowd density at the corresponding location in the image. Given the estimated density map, the crowd count can be obtained by summing over entries of the density map. Crowd counting has many real-world applications, such as surveillance, traffic monitoring, urban planning [10]. Crowd counting is a very challenging problem due to factors such as varying crowd density, occlusion, perspective distortion, etc.

Previous work on crowd counting usually treats it as a standard supervised learning problem. The de facto solution in the literature is to collect a set of training images with ground-truth annotations (i.e. density maps). Then we can train a model using convolutional neural networks (CNNs) to map an input image to a density map. In the past few years, various CNN-based network architectures [10, 14, 19, 20] have been proposed for crowd counting.

We argue that there is a disconnect between the standard supervised learning setting and the real world scenarios in which these systems are deployed and used. First of all, the supervised learning setting assumes that the learning algorithm will produce one single crowd counting model in the end. This model will be deployed in all future use cases. This assumption puts too much demand on the learning algorithm, since the learned model has to be able to handle all scenarios that it might encounter during testing. However, this is not how crowd counting systems are typically used in the real world. In reality, the test images are usually captured by the same camera mounted at a fixed location and angle. In other words, all the test images are from the same scene. So we only need the learned model to perform well in this particular scene. Secondly, the supervised learning setting assumes that once a model is learned, it cannot be modified to fit a particular scene. Again, this is not true in many real-world scenarios. Often after a camera is installed in a new environment, there is a calibration process. During the calibration, we can afford to collect a small number of images from the camera and even manually annotate them. Intuitively, we should be able to take advantage of these small number of images (and their labels) collected during the calibration process and adapt the model to work better in the target scene. Of course, the number of labeled images collected during the calibration process is often very small. So we cannot use the brute-force approach of training a model from scratch based on labeled examples in the target scene.

In this paper, we propose a novel problem called the *one-shot scene-specific crowd counting*. During training, we assume that we have a set of labeled training data from several scenes. During testing, we would like to deploy the crowd counting algorithm in a specific target scene. Note that we do not have any images from the target scene during training. We also assume that we have one single labeled example from the target scene. Our goal is then to produce a crowd counting model specifically tuned to this target scene. We focus on one-shot learning in this paper, but the proposed method can be easily adapted to few-shot learning where we have more than one labeled example in the target scene. We believe our problem setting is closer to how crowd counting solutions are deployed in the real world. Consider the hypothetical scenario of a company providing crowd counting solutions to different customers. The company may have access to a large collection of images from previous customers. We can treat images from one customer as belonging to one "scene". The company can annotate these images with ground-truth labels and train some models offline. Now suppose the company needs to deploy the solution to a new customer (i.e. a new "scene"). It is reasonable to expect that we can have one or few labeled images from this new customer. If there is a reliable solution to the one-shot scene-specific crowd counting problem, the company will be able to quickly adapt the models trained offline to this new customer using one or few labeled examples from this customer.

One popular approach for one-shot learning problem in computer vision is to use fine-tuning. We can first learn a crowd counting model using all the training data under the standard supervised setting. Given a labeled image in the target scene, we keep the model parameters in early layers of the network (called the *encoder* in this paper) fixed and fine-tune the last few layers in the model (called the *decoder* in this paper) based on this labeled image. This fine-tuning has been used for one-shot object segmentation in videos [2]. We

call this approach "simple fine-tuning". The limitation of this simple fine-tuning approach is that the learning of the model parameters is disconnected from the testing. In other words, the encoder parameters are learned to achieve good performance in the standard supervised setting, they are not explicitly learned for the purpose of facilitating fine-tuning to the target scene. In this paper, we propose a novel one-shot learning approach using fine-tuning. Different from the simple fine-tuning, our learning algorithm is specifically designed for the goal of one-shot adaptation to a new scene.

The contributions of this work are manifold. First, we introduce a new problem called the one-shot scene-specific crowd counting. This problem setting is closer to how crowd counting algorithms are used in the real world. A reliable solution to this problem will help the wide adoption of crowd counting algorithms in real-world applications. Second, we propose a novel one-shot learning algorithm for this new problem. The novelty of our approach is that it is specifically designed to facilitate fine-tuning in target scenes. Finally, since this is a new problem and there are no existing datasets specifically created for it, we have amassed two benchmark datasets for this problem by repurposing existing crowd counting datasets. On these benchmark datasets, our proposed approach significantly outperforms several baseline methods.

2 Related Work

Crowd counting in images is an area of great interest due to its potential applications in many domains. Early work (e.g. [15]) on crowd counting uses detection-based approaches that locate persons in an image. There is also some work on regression-based approaches that directly map an image to the count. In recent years, convolutional neural networks (CNNs) have shown great success in crowd counting. Most of these approaches [9, 10, 12, 13, 14, 17, 20] work by using CNN to map an input map to a density map, where each pixel in the density map indicates the crowd density of the corresponding location in the image. The final crowd count can be obtained by summing over the density map. Our work is also related to one-shot learning in computer vision. The goal of one-shot learning is to learn a new concept from one or few examples. Some early work [6] performs one-shot learning of object categories by learning appearance and geometric knowledge of objects from some object categories and transferring this knowledge to other new object categories. Vinyals et al. [16] develop a matching network for one-shot learning. Santoro et al. [11] develop a memory-based neural network for one-shot learning. Some recent work in one-shot learning uses the idea of meta-learning. Bertinetto et al. [1] propose a method for learning to predict the weights of a neural network by another neural network. Wang et al. [18] develop a learning to learn approach for learning object categories from small data. Caelles et al [2] perform one-shot object segmentation in videos by fine-tuning a pre-trained model on the first frame in a video.

3 Approach

Figure 1 shows an overall of our approach. Our backbone counting network consists of an encoder and a decoder (Sec. 3.1). During testing, we are given a target scene. The parameters of the decoder are fine-tuned on one labeled image from the target scene (Sec. 3.2). During training, we learn the encoder and the decoder jointly in a way that allows the model to

effectively adapt to a new target scene (Sec. 3.3).

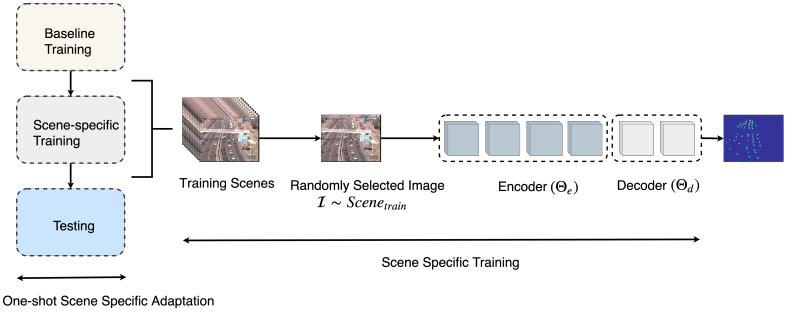

Figure 1: Illustration of one-shot scene-specific adaptation using CSRNet [8] as the backbone crowd counting network. Given a target scene during testing, the weights Θ_d are fine-tune on this target scene while Θ_e are fixed. During training, we first perform a baseline training using images from all training scenes. The weights of the baseline model are used to initialize Θ_e and Θ_d in the scene-specific training. In scene-specific training, the encoder Θ_e and decoder Θ_d are learned a way that allows effective adaptation in a target scene.

3.1 Model Architecture

In this paper, we use the Dilated Convolutional Neural Networks (CSRNet) architecture [8] as our backbone network, although our proposed approach is agnostic to the specific choice of network architectures. CSRNet [8] uses a convolutional neural network (CNN) for feature extraction and a dilated CNN to produce the density map.

With a little abuse of terminology, we refer to first part of the network architecture in Fig. 1 as the *encoder network* and the remaining part of the network architecture as the *decoder network*. The encoder network is used to extract features from an image, while the decoder network is used to produce the output from the extracted features. Note that the split of encoder/decoder in the network is flexible. In our experiments, we have considered either the last layer or the last two layers of CSRNet as the decoder. We consider the remaining layers as the encoder. We use Θ_e and Θ_d to denote the model parameters corresponding to the encoder and the decoder networks, respectively. The model parameters Θ of the entire CSRNet are the combination of Θ_e and Θ_d, i.e. $\Theta = \{\Theta_e, \Theta_d\}$. We use $F(x; \{\Theta_e, \Theta_d\})$ to denote the output density map of CSRNet for an input image x. In the standard supervised setting, the model parameters Θ are obtained by optimizing the loss function defined on a collection of training images.

3.2 One-Shot Scene-Specific Adaptation

During testing, we are given a collection of M images $\{x_i\}_{i=1}^{M}$ from the *same* scene. For example, these images might be captured by the same camera fixed at a particular location. We also have the ground-truth annotation of one "test" image for this scene. Here we abuse the terminology a bit – this image is technically not a test image since it has the ground-truth annotation. Without loss of generality, we assume that x_1 has the ground-truth annotation (i.e. density map) y_1.

During the inference, we would like to adapt the model parameters Θ to this specific scene. A popular approach for this adaption is by fine-tuning the model parameters on (x_1, y_1). This strategy has been successfully used in one-shot video object segmentation (OSVOS) [2]. In OSVOS, we are given a video for which the segmentation mask of the object of interest is provided on the first frame. OSVOS then fine-tunes the learned segmentation network on the first frame to obtain a segmentation network specifically tuned to this video. Finally, the fine-tuned segmentation network is used to segment the object in the remaining frames. In our work, we use a similar idea to fine-tune a crowd counting network on one single labeled example (x_1, y_1). As a result, the network will be adapted to this specific scene. However, since we have only one labeled example for this scene, fine-tuning the entire model parameters Θ will tend to overfit to this single example.

In our work, we choose to fix the parameters of the encoder network Θ_e and only fine-tune the parameters of the decoder network Θ_d. As mentioned previously, the split of encoder/decoder is somewhat arbitrary and depends on the application scenario. As a rule of thumb, if we only have very few labeled examples for a specific scene, we should choose a lightweight decoder with few layers. If we have more labeled examples, we can choose the decoder to have more layers.

Given an image x and its ground-truth density map y (note that y is usually represented as a matrix), we use the L_2 distance between the predicted density map $F(x; \{\Theta_e, \Theta_d\})$ and ground-truth density map y as the loss function. So the fine-tuning process is equivalent to solving the following optimization problem:

$$\Theta_d^* = \arg\min_{\Theta_d} L_d(\Theta_d), \quad \text{where } L_d(\Theta_d) = ||\text{vec}(F(x_1; \{\Theta_e, \Theta_d\})) - \text{vec}(y_1)||_2 \qquad (1)$$

where $\text{vec}(\cdot)$ converts an input matrix to a vector. Note that the loss $L_d(\Theta_d)$ in Eq. 1 is a function of Θ_d since we are only fine-tuning the decoder network. The parameters Θ_e in the encoder network are fixed during the inference and can be treated as constants. Also note that $L_d(\Theta_d)$ is defined on (x_1, y_1) which is the single labeled example for the specific scene. Fine-tuning according to Eq. 1 involves computing the gradient of $L_d(\Theta_d)$ with respect to Θ_d using back-propagation and updating Θ_d using gradient descent.

After fine-tuning on the first image of the target scene, we obtain updated decoder parameters Θ_d^*. For any other test image x_i (where $i = 2, ..., M$) from the target scene, we predict its density map as $F(x_i; \{\Theta_e, \Theta_d^*\})$. Note that the index i starts from 2 (instead of 1) since we assume that x_1 comes with its ground-truth annotation y_1, so we do not need to make the prediction for x_1.

In our formulation, the encoder parameters Θ_e are generic to all scenes. In contrast, the decoder parameters Θ_d are specific to a particular scene. By fine-tuning Θ_d on a labeled example from the target scene, we effectively tune our model to adapt to the target scene.

3.3 Model Learning

During training, we have a collection of labeled training images from T scenes. For example, each scene might correspond to a camera fixed at a particular location. For the t-th scene $(t = 1, 2, ..., T)$, we use $\{(x_i^{(t)}, y_i^{(t)})\}_{i=1}^N$ to denote the training images and the corresponding ground-truth density maps from that scene. To simplify the notation, we have assumed that each scene has the same number of N training images. But our proposed algorithm can be easily generalized to the case where different scenes have different numbers of training images.

Our goal of the training phase is to learn the parameters Θ_e of the encoder network. Note that technically speaking, we do not need to learn the decoder parameters Θ_d during the training phase. If we have the encoder parameters Θ_e, the decoder parameters Θ_d will be obtained via fine-tuning (see Eq. 1) on a single labeled example from the target scene. Ideally we would like the learned Θ_e to have the following property. Given a target scene t, suppose we use the learned Θ_d^* in Eq. 1 by fine-tuning on $(x_1^{(t)}, y_1^{(t)})$, we would like CSRNet with model parameters $\{\Theta_e, \Theta_d^*\}$ to perform well on the remaining training images in this scene. In other words, $F(x_i^{(t)}; \{\Theta_e, \Theta_d^*\})$ should be close to $y_i^{(t)}$ (where $i = 2, 3, ..., N$). Note that we do not necessarily need to enforce $F(x; \{\Theta_e, \Theta_d^*\})$ to perform well if x is an image from a different scene. This is a reasonable learning objective since it mimics what happens during testing. If the learned encoder parameters Θ_e have the above property, when we fine-tune the decoder parameters Θ_d on a single labeled example in a target scene, the fine-tuned parameters will likely work well on other images from this target scene.

For a fixed Θ_e, we use $\Theta_d^{(t)}$ to denote the decoder parameters for the t-th scene obtained by fine-tuning on $(x_1^{(t)}, y_1^{(t)})$ according to Eq. 2:

$$\Theta_d^{(t)} = \arg\min_{\Theta_d} L_d^{(t)}(\Theta_d), \quad \text{where } L_d^{(t)}(\Theta_d) = ||\text{vec}(F(x_1^{(t)}; \{\Theta_e, \Theta_d\})) - \text{vec}(y_1^{(t)})||_2 \tag{2}$$

For simplicity, let us ignore the randomness (e.g. due to random initialization and local minimum) in the specific optimization algorithm used for solving Eq. 2 and consider that the solution of $\Theta_d^{(t)}$ in Eq. 2 is unique for a fixed Θ_e. Then technically speaking, $\Theta_d^{(t)}$ is a function of Θ_e. If we vary the value of Θ_e, Eq. 2 will give a different solution of $\Theta_d^{(t)}$. To make this relationship more explicit, we write the decoder parameters as $\Theta_d^{(t)}(\Theta_e)$ to emphasize that the decoder parameters can be treated as a function of the encoder parameters.

Now we introduce the loss function for the learning. We first focus on the loss function on the t-th scene. Note that we would like $\{\Theta_e, \Theta_d^{(t)}(\Theta_e)\}$ to perform well on the remaining training images $\{(x_i^{(t)}, y_i^{(t)}\}_{i=2}^N$ in this scene, so a natural choice of the loss is as follows:

$$L_e^{(t)}(\Theta_e) = \sum_{i=2}^{N} ||\text{vec}(F(x_i^{(t)}; \{\Theta_e, \Theta_d^{(t)}(\Theta_e)\})) - \text{vec}(y_i^{(t)})||_2 \tag{3}$$

Note that the loss $L_e^{(t)}(\Theta_e)$ is a function of only the encoder parameters Θ_e, since the decoder parameters $\Theta_d^{(t)}(\Theta_e)$ become deterministic once we have Θ_e.

The final loss is simply the summation of $L_e^{(t)}(\Theta_e)$ over all scenes, i.e. $L_{final}(\Theta_e) = \sum_{t=1}^{T} L_e^{(t)}(\Theta_e)$. Note that $L_{final}(\Theta_e)$ is a function of the encoder parameters Θ_e. In theory, we can just take the gradient of $L_{final}(\Theta_e)$ with respect to Θ_e and use gradient descent. However, this is difficult in practice since there is another optimization problem (see Eq. 2) nested in the definition of the loss $L_{final}(\Theta_e)$. So it is non-trivial to directly calculate the gradient of $L_{final}(\Theta_e)$ with respect to Θ_e.

Our key insight is that if we fix the decoder parameters $\Theta_d^{(t)}$ (where $t = 1, 2, ..., T$) for all scenes, $L_{final}(\Theta_e)$ becomes a simple function that can be easily differentiated. Based on this observation, we propose to optimize $L_{final}(\Theta_e)$ using a coordinate descent style approach that alternates between the following two steps:

- Fix Θ_e, solve for $\Theta_d^{(t)}$ (where $t = 1, 2, ..., T$) using Eq. 2 for each scene;

- Fix $\Theta_d^{(t)}$ (where $t = 1,2,...,T$), solve for Θ_e by minimizing $L_{final}(\Theta_e)$. Note that in this case, Θ_e are simply part of the parameters in CSRNet and can be optimized using SGD with back-propagation.

These two steps can be repeated until convergence. In practice, we alternate between these two steps for a fixed number of iterations.

Note that our approach is different from the standard fine-tuning strategy used in other one-shot learning methods (e.g. [2]). The fine-tuning method in [2] first learns a full model (both Θ_e and Θ_d) by minimizing the empirical loss on the training data, i.e.

$$\Theta_e^*, \Theta_d^* = \arg\min_{\Theta_e,\Theta_d} \sum_{t=1}^{T}\sum_{i=1}^{N} ||\text{vec}(F(x_i^{(t)};\{\Theta_e,\Theta_d\})) - \text{vec}(y_i^{(t)})||_2 \quad (4)$$

Given a target scene, the encoder parameters Θ_e^* will be fixed. Only the decoder parameters Θ_d will be fine-tuned using Eq. 1. The problem of this simple fine-tuning method is that there is a disconnect between learning and testing. During learning, the loss function in Eq. 4 does not explicitly enforce Θ_e to be transferable to a new scene. In contrast, our proposed approach learns Θ_e in a specific way, so that when we use Θ_e to get the fine-tuned decoder parameters $\Theta_d^{(t)}$ in a target scene, the final model will perform well.

Randomization of Training Images: So far, we have assumed that the first example $(x_1^{(t)}, y_1^{(t)})$ in the t-th scene is used for fine-tuning. For testing, this is reasonable since this example corresponds to the only labeled example available in the target scene. But for training, this creates asymmetry among training data since the first training example in the t-th scene plays a different role from other training examples in this scene. In order to break this asymmetry, we randomly pick a training image from the t-th scene as $(x_1^{(t)}, y_1^{(t)})$ by shuffling the images in each scene at the beginning of every training epoch. This will make sure that all training examples in the t-th scene have the chance of being used for fine-tuning.

4 Experiments

4.1 Baselines

Since this paper is the first work on one-shot scene-specific crowd counting, there is no previous work that we can directly compare with. Nevertheless, we define several of our own baseline methods as follows.

No fine-tuning: This baseline is the CSRNet model [8] trained in a standard supervised setting. The model parameters are trained from all images in the training set according to Eq. 4. Once the training is done, the model is fixed and is applied to all test images. Note that the original CSRNet [8] uses the perspective maps and ground-truth ROI regions to improve the final performance. For simplicity, we do not use these extra components.

Simple fine-tuning: This baseline first learns $\{\Theta_e, \Theta_d\}$ in the standard supervised setting using all the training images (i.e. the "no fine-tuning" baseline). For a given scene during testing, it fixes Θ_e and fine-tunes Θ_d using Eq. 1. Note that this baseline is equivalent to the method for one-shot object segmentation in [2].

As mentioned earlier, the split of encoder/decoder networks in our formulation is flexible. We have experimented with using either the last layer or the last two layers of CSRNet [8] as the decoder. We refer to them as "simple fine-tuning (1 layer)" and "simple fine-tuning (2

Method	WorldExpo'10		Trancos	
	MAE	MSE	MAE	MSE
no fine-tuning	11.58	17.72	5.66	7.65
simple fine-tuning (1 layer)	7.82	10.87	5.51	6.11
ours (1 layer)	**7.7**	**10.62**	**5.45**	**6.04**
simple fine-tuning (2 layers)	8.39	11.41	4.98	5.55
ours (2 layers)	**7.53**	**10.42**	**4.46**	**5.18**

Table 1: Comparison of the performance (MAE and MSE) of our approach and the baselines on the WorldExpo'10 dataset and Trancos dataset. For "ours" and "simple fine-tuning", we consider either using the last layer or the last two layers of CSRNet as the decoder.

Method	WorldExpo'10	
	MAE	MSE
no fine-tuning	19.91	37.71
simple fine-tuning (2 layer)	10.62	13.98
ours (2 layer)	**8.23**	**12.08**

Table 2: Comparison of the performance (MAE and MSE) of our approach and the baselines on the WorldExpo'10 dataset using the standard split.

layers)" for the "simple fine-tuning" baseline, respectively. Similarly, we have experimented with using either one layer or two layers as the decoder in our proposed models. We refer to them as "ours (1 layer)" and "ours (2 layers)", respectively.

4.2 Datasets and Setup

Datasets: Most existing crowd counting datasets are not created for the problem studied in this paper. In particular, the training and testing images are often collected from the same scene in those datasets. As a result, we cannot use these datasets since our problem setting assumes that images in a dataset come from multiple scenes. To measure the effectiveness of scene specific training, we choose two datasets with more than 70 scenes, namely WorldExpo'10 [19] for counting people and Trancos [6] for counting vehicles. In our experiments, we use these two datasets to create our own benchmark datasets. For cross-dataset testing, we consider two more crowd counting datasets, Mall dataset [4] and UCSD dataset [3]. The WorldExpo'10 dataset has a total of 1132 annotated frames collected from 103 different scenes with a fixed resolution of 576×720. We choose 50 scenes for training and the remaining 53 scenes for testing. For each scene in testing, we choose one image randomly and corresponding ground-truth label as (x_1, y_1) and perform fine-tuning on this image. The test

Method	(a) W→U		(b) W→M		(c) W→T		(d) T→W	
	MAE	MSE	MAE	MSE	MAE	MSE	MAE	MSE
no fine-tuning	23.84	24.9	18.76	19.29	31.19	35.67	50.14	61.77
simple fine-tuning (L-1)	4.45	5.29	5.17	6.34	18.09	19	41.33	46.36
ours (L-1)	**2.32**	**2.39**	**1.91**	**2.48**	**7.12**	**8.02**	**19.23**	**23.23**
simple fine-tuning (L-2)	4.42	5.26	4.29	5.12	7.14	8.31	21.03	26.12
ours (L-2)	**2.31**	**3.12**	**1.85**	**2.40**	**5.72**	**6.61**	**8.91**	**12.19**

Table 3: Performance in the cross-dataset testing with the same (a,b) and different (c,d) object. We use "W", "U", "M", "T" to denote WorldExpo'10, UCSD, Mall, Trancos, respectively. In "W→U" and "W→M", we use WorldExpo'10 for training, then test on UCSD and Mall. In "W→T", we train on WorldExpo'10 (people) and test on Trancos (vehicles). In "W→T", we train on Trancos (vehicles) and test on WorldExpo'10 (people).

accuracy is calculated on the remaining images in this scene. The Trancos dataset consists of 1244 images captured from different scenes. We choose 50 scenes for training and the remaining 20 scenes for testing. Again, for each scene in testing, we choose one image and its ground-truth label for fine-tuning. Mall dataset contains 2000 images from a single scene with a size of 640 $\times$ 480. Each image contains the headcount of the people. According to standard split for the Mall dataset first 800 frames are used for training and the remaining 1200 images are used for testing. UCSD dataset contains 2000 labeled images from a single scene with the image size of 238 $\times$ 158. The training set contains images from 600 to 1400, which contains 800 frames. The remaining 1200 frames are held-out for testing. To generate the ground-truth density maps, we follow the procedure described in [20].

Implementation Details: The parameters $\{\Theta_e, \Theta_d\}$ learned using the "no fine-tuning" baseline are used to initialize our model. We use the Adam optimizer [7] with a learning rate of 1e-5 and a momentum of 0.9. We run 200 epochs when optimizing Eq. 2 and Eq. 3. The number of iterations for the coordinate descent style alternating algorithm is fixed to be 1000.

4.3 Experimental Results

Main Results: Table 1 shows the results of our method and the comparison with the baselines on the WorldExpo'10 dataset and Trancos dataset. We show results of using either the last layer or the last two layers of CSRNet as the decoder. According to the results, we can see that without fine-tuning during testing, the performance is poor. Simple fine-tuning can improve the performance. Our proposed method (using either one layer or two layers as the decoder) significantly outperforms other baselines.

As mentioned earlier, we have decided to use a different training/testing split in our experiments, since this paper studies a new problem and the standard split is not suitable for this problem setting. Nevertheless, we have also tried our method with 2 layers by following the standard split used in the literature as close as possible. Table 2 shows the results of using the standard split on WorldExpo'10. In this split, there are 5 test scenes. We can see that our method outperforms the baselines in this standard split setting as well. We like to emphasize that even in this standard split, the results are not directly comparable to previous works. Our problem setting assumes that one image in a testing scene is "labeled" and this image has to be excluded when calculating the performance numbers. In other words, there are slightly fewer number of test images used in calculating MAE/MSE in Table 2. Also note that CSRNet [8] uses extra components (perspective maps, ROI) to improve the final performance. Due to these differences, we do not include previously reported numbers in Table 2 since they are not directly comparable and can be misleading. Also note that the MAE/MSE numbers in Table 2 are in general higher than those in Table 1. This is because in the standard split, the 5 test scenes are totally different the training scenes, e.g. station vs street scenes. For the training/testing split in Table 1, a lot of test scenes are similar to training scenes. This domain shift explains the performance difference in Table 1 and Table 2.

Unfortunately we could not perform the experiment on Trancos using the standard split, since images from the same scene can be appear in both training and test sets according to the standard split on Trancos. This violates the assumptions of our problem setting which assumes that training and testing should have non-overlapping scenes.

Cross-Dataset Testing: In real-world applications, the images we encounter during testing are likely to be completely different from the training. To demonstrate the generalizability

of our model in the case of significant domain shift between training and testing, we perform an experiment using cross-dataset testing. First, we train our model on the WorldExpo'10 dataset and then evaluate the model on both the Mall dataset [4] and the UCSD dataset [5] using the standard test data splits. We take one test image in the target dataset as for fine-tuning. Table 3(a,b) shows the results of this cross-dataset testing. In Table 3(c,d) we show the cross object testing between two datasets with different objects, namely WorldExpo'10 (people) and Trancos (vehicle).

5 Conclusion

We have introduced a new problem called the one-shot scene-specific crowd counting. This new problem setting is motivated by the disconnect between current research in crowd counting algorithms and how these algorithms are deployed in real-world application. We introduce a novel algorithm for this problem. Our experimental results show that the proposed approach outperforms other baseline methods.

Acknowledgement

This work was funded by NSERC. We thank NVIDIA for donating some of the GPUs used in this work.

References

[1] Luca Bertinetto, João F Henriques, Jack Valmadre, Philip Torr, and Andrea Vedaldi. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*, 2016.

[2] Sergi Caelles, Kevis-Kokitsi Maninis, Jordi Pont-Tuset, Laura Leal-Taixé, Daniel Cremers, and Luc Van Gool. One-shot video object segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[3] Antoni B Chan, Zhang-Sheng John Liang, and Nuno Vasconcelos. Privacy preserving crowd monitoring: Counting people without people models or tracking. In *IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2008.

[4] Ke Chen, Chen Change Loy, Shaogang Gong, and Tony Xiang. Feature mining for localised crowd counting. In *British Machine Vision Conference*, 2012.

[5] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.

[6] Ricardo Guerrero-Gómez-Olmedo, Beatriz Torre-Jiménez, Roberto López-Sastre, Saturnino Maldonado-Bascón, and Daniel Onoro-Rubio. Extremely overlapping vehicle counting. In *Iberian Conference on Pattern Recognition and Image Analysis*, 2015.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[8] Yuhong Li, Xiaofan Zhang, and Deming Chen. Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[9] Jiang Liu, Chenqiang Gao, Deyu Meng, and Alexander G Hauptmann. Decidenet: Counting varying density crowds through attention guided detection and density estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[10] Deepak Babu Sam, Shiv Surya, and R Venkatesh Babu. Switching convolutional neural network for crowd counting. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[11] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. In *Advances in Neural Information Processing Systems*, 2016.

[12] Chong Shang, Haizhou Ai, and Bo Bai. End-to-end crowd counting via joint learning local and global count. In *IEEE International Conference on Image Processing*, 2016.

[13] Vishwanath A Sindagi and Vishal M Patel. Cnn-based cascaded multi-task learning of high-level prior and density estimation for crowd counting. In *Advanced Video and Signal Based Surveillance*, 2017.

[14] Vishwanath A Sindagi and Vishal M Patel. Generating high-quality crowd density maps using contextual pyramid cnns. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[15] Ibrahim Saygin Topkaya, Hakan Erdogan, and Fatih Porikli. Counting people by clustering person detector outputs. In *Advanced Video and Signal Based Surveillance*, 2014.

[16] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems*, 2016.

[17] Elad Walach and Lior Wolf. Learning to count with CNN boosting. In *European Conference on Computer Vision*, 2016.

[18] Yu-Xiong Wang and Martial Hebert. Learning to learn: Model regression networks for easy small sample learning. In *European Conference on Computer Vision*, 2016.

[19] Cong Zhang, Hongsheng Li, Xiaogang Wang, and Xiaokang Yang. Cross-scene crowd counting via deep convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

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