COMP4901J Final Report
Cell Counting by Adaptive Fully Convolutional Redundant Counting

Wang Xinyi
20412431
xwangcs@connect.ust.hk

Zhang Daofu
20493928
dzhangat@connect.ust.hk

Sun Dajun
20491572
dsunad@connect.ust.hk

Abstract

Cell counting is an important task in biological domain. However, it could be time consuming and error-prone to count cells manually. Some deep fully convolutional neural networks have been proposed recently to automatically count cells from the microscopic images. But the current methods can only be trained to count one type of cells. If we want to count another kind of cells, a proper amount of new data is needed to train the whole network with millions of parameters which is very time consuming. To enable fast domain transfer between different kind of cells, we propose to pre-train the network on a large dataset of simple synthetic microscopic cell images and then freeze the domain agnostic parameters and only train the domain specific parameter on the new domain. In details, our network structure is obtained by adding several residual adapters to the count-ception network, which is the current state of the art method for cell counting task. Experiment results show that our proposed method significantly outperforms the original training from scratch method on two small datasets of real microscopic cells images.

1. Introduction

Automatic objects counting in digital images have been drawing lots of research interests since counting crowded objects by human is very time consuming and error-prone. In the biology domain, counting cells of interest is an important task in many different domains. In cell growth studies, pictures of a same group of cells are taken every fixed period of time and researchers need to count the number of cells in each picture to keep track of the cell growth, usually to see the effectiveness of the treatment of a certain type of compounds. In hospitals, doctors use Full Blood Count(FBC) to determine the health condition of a patient. Microbiologists use cell counting to study the behaviour of infectious viruses and bacteria.

Previous attempts to solve the counting problem can be divided into two directions: the detection-based methods and the regression-based methods. The regression-based methods, which are the most popular methods in crowd counting and general counting problems, can be further divided by its ground truth labels: using a scalar, i.e. the count number, or using dot annotations, i.e. a one-pixel dot is obtained for each object to be counted. Most of the recent works use the dot annotations instead of a single count number because of the following reasons:

Dot annotations are no harder to obtain than the count number as it is intuitive for human to count by dotting. Additional spatial information is preserved which could be very useful for training. Also, in biochemistry labs, researchers sometimes need to track back some cells for details after counting.

Although dot annotation is surely a better choice of label than a single count number for learning, it still cannot be directly used as the ground truth label for learning since it is very sensitive to noises as each object is represented by a single-pixel dot. In 2010, Lempitsky and Zisserman [3] proposed to utilized the dot annotation by producing a Gaussian density map from it, which is widely used in many recent works of crowd counting. However, producing a Gaussian distribution at the position of every dot is very computational expensive and it is not clear why we should choose to use a Gaussian distribution to form the ground truth map.

A novel solution is proposed in 2017, by Joseph et.al. [1]: instead of finding the object density over the whole image, they count the number of cells exists in a square neighborhood area of the image. This square kernel is much more easy to compute and much more intuitive for a convolutional neural network since it also a square shaped receptive field. Also, since the kernels are overlapped, an cell is counted redundantly so the error can be reduced by average over these repeated counts. This method significantly outperforms all the previous methods on cell counting.

In reality, we often need to count different types of objects (e.g. different types of cells) separately rather than just giving a final count. However, current cell counting methods can only be trained to count one type of cells and the whole neural network needs to be re-trained if we want to count a new kind of cells.

Our goal is to build a easy-to-use system that can be quickly transferred to count cells of different types. To achieve this, we add some adaptive modules to separate the domain-agnostic parameters from the domain-specific parameters. This idea is used in some other works on domain-agnostic counting before such as [6] and [5].

The contributions of this work can be summarized as below: First, we are among the first to combine the redundant counting model proposed by Joseph et.al.[1] and the residual adapters proposed by Sylvestre et. al. in 2017[7]. Previous attempts to combine a counting network and the adaptive modules like [6] by Marsden et. al. in 2018, only slices the original image into nine non-overlapping square pieces instead of many overlapped ones. Lu et. al. [6] not only use adaptive modules but also utilize the relation network to perform general object counting in a more few-shot learning fashion. Second, we produce some very good results, if not the best, on small real cell datasets by only training a small fraction of parameters. In our experiments, we transferred the pre-trained network using our proposed dataset and the MBM dataset [2], which achieves a much better performance than training from scratch. This consistent with the idea that deep neural networks may share a lot of domain-agnostic parameters.

2. Related Work

In 2010, Lempitsky and Zisserman[3] raised the idea of counting with a density map, which becomes the core of many other counting approaches. To construct the density map of an image given its dot annotation, a Gaussian distribution is assumed to be formed at each annotation point. Summing up all the distributions at each annotation point and integrate over the image gives us the approximate number of count since integrate a single Gaussian distribution gives us 1, which is exactly the count number of one dotted object. However, their prediction comes from a linear regression model using SIFT features as input and deep neural networks are not involved.

Many approaches were introduced to predict a better density map. In 2016, Xie et.al.[8] proposed a deep neural network to solve this problem. Their model accepts 100×100 input image and convolves it to a 100×100 density map. Their method achieves better performance but is still limited by the nature of Gaussian density map, as using density map forces the model to predict location-specified values based on the distance between the object and the dot annotation point. This is usually harder than just predicting the existence of objects in the receptive field.

Our work is largely based on the current state of the art method on cell counting proposed by Joseph et.al. in 2017[1]. Instead of finding the object density over the whole image, their redundant counting model builds on the idea of counting the number of cells in a square-shaped neighborhood area of the image. Since these neighborhood areas can be overlapped, each cell can be counted more than once and therefore can average over the random error produced in each prediction. A 10-layered fully convolutional neural network is designed to have a receptive field of the same size of the square area. However, the count-ception model they proposed only works for one specific type of object and can't perform domain-transfer efficiently, which differentiates their work from ours.

The residual adapters proposed by Sylvestre et. al. in 2017[7] are applied in our model to deal with the domain shifting problem. The idea of adapter module is to decompose the model parameters into two parts: a domain-agnostic part and a domain-specific part. Since the amount of domain-specific parameters are often significantly fewer the other part, it is possible to use only a few new data to train a new model in a short time and still maintain a good performance. Unlike Sylvestre, we only add a few adapters at certain layers rather than putting one after each layer.

3. Data

In this section, we introduce the three datasets we use in details:

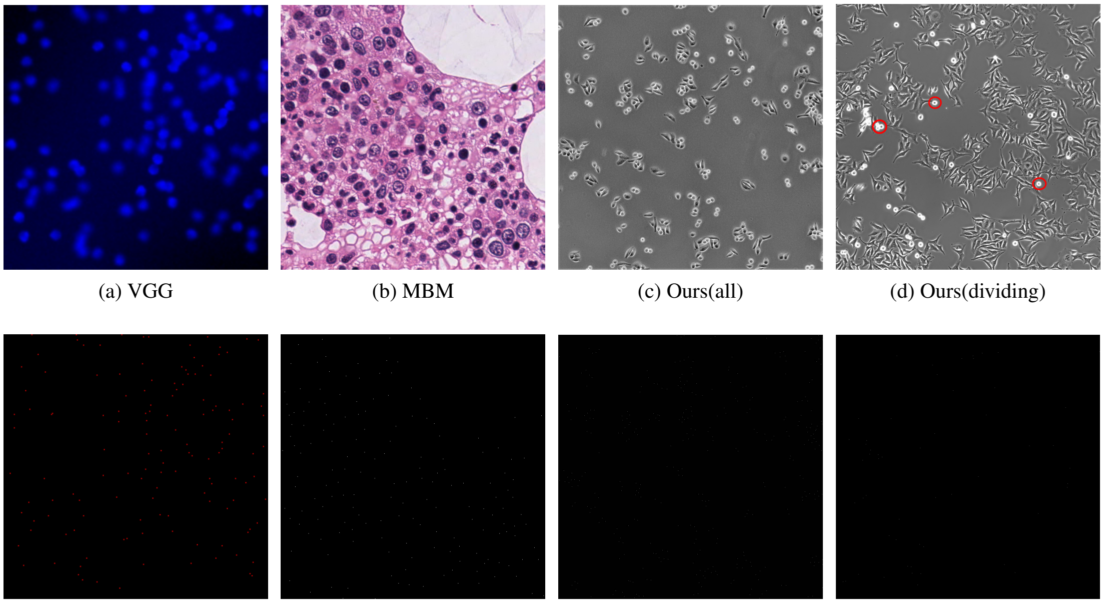
VGG Cells: The VGG Cells dataset is a standard benchmark dataset introduced by Lempitsky and Zisserman in 2010[3]. There are 200 synthetic images with 256×256 resolution containing simulated bacterial cells from fluorescence-light microscopy created by Lehmussola et.al. Dot annotations are also synthesized and guaranteed to be accurate. Additionally, overlapping are considered and pictures are taken at various focal distances to stimulate real life imaging with a microscope.

MBM Cells: The MBM Cells dataset is a real dataset modified from the BM dataset [1]. The BM dataset is introduced by Kainz et al. in 2015[2] and consists of eleven 1200×1200 images of human bone marrow. The cells are processed in a manner such that the nuclei of each cell is stained blue whereas the other cell components appear pink and red. There are two main modifications of MBM from BM. First each 1200×1200 image is cropped into four 600×600 images in order to get more data, this yields a total of 44 images. Second, the the ground truth annotations were updated to capture unlabelled nuclei.

Our Proposed Dataset: Our proposed dataset is collected from Professor Hong Xue's biochemistry lab in HKUST. This dataset contains 1925 images without any annotation, and each contains around 200 to 500 Hela cells of different types and status (i.e. single-nucleus cells and binuclear cells, normal cells and dividing cells), with a resolution of 1024×1024. Our goal is to count both the total number of all cells and the number of dividing cells, therefore,

Dataset	#train	#validation	#test	image size	average cell count
VGG	160	20	20	256 × 256	169.10 ± 56.90
MBM	35	5	4	256 × 256	94.41 ± 21.10
Ours (count all cells)	67	8	4	256 × 256	93.36 ± 18.62
Ours (count dividing cells)	67	8	4	256 × 256	19.63 ± 10.00

Table 1: Statistics of our datasets. The last column is the average cell count in the whole dataset and standard deviation.



(a) VGG (b) MBM (c) Ours(all) (d) Ours(dividing)

(e) VGG (f) MBM (g) Ours(all) (h) Ours(dividing)

Figure 1: (a)-(d): Raw images from VGG, MBM and ours dataset. Note that (c) is used to count all cells and (d) is used to count dividing cells. The dividing cells in (d) are those bright circular ones. The ones surrounded by red circles are some examples. (e)-(h): The corresponding dot annotations for each image.

our dataset is further divided into two categories. The first group is used to count all cells and therefore all living cells are labelled with dot annotation. In the second group we only labelled dividing cells to just count dividing cells. As making labels is very time-consuming and the original images is large enough, we only labelled four images for each category and apply some data augmentation techniques to get more train data.

In our experiment, we use the three datasets for different purposes. The VGG Cells dataset is used for pre-training while the MBM dataset and our proposed dataset are used to perform the domain transformation. The detailed statistic of these datasets are shown in Table 1 and some sample images and the corresponding dot annotations are shown in Figure 1.

Data Augmentation and Pre-Process The size of images in each dataset are adjusted to fit our model.

For the MBM dataset, each image is first down-sampled by 2 × 2 max-pooling into 300 × 300 and then cropped into 256 × 256. For our proposed dataset, the images are processed in a more complicate method. As we labelled four images for each group, we decide to use three for training and one for test. For the train part, we first down sampled images into 512×512 by 2×2 max-pooling. Then, slice the down-sampled 512 × 512 images into overlapped 256×256 pieces withe stride 64. This generates 25 images from each original image and 75 images in total. For the test part, similarly we first down-sampled it to 512 × 512 and then we simply divide it into four 256 × 256 pieces. The total number of images is therefore 79, as shown in Table 1.

The reason why we first down-sample our image to 512× 512 rather than slicing it directly is simple. Considering that images in VGG dataset contain 169.1 cells on average, to utilize the features learned in pre-training better, we'd better let our data have similar cell density as those used for pre-training. In order to obtain a considerable amount of data and maintain cell density at the same time, 512 × 512 resolution is the best choice.

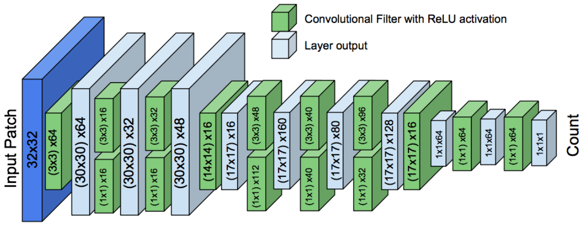


Figure 2: Fully convolutional network architecture from [1]. Note that the 3×3 convolution is also padded so it doesn't reduce image size.

4. Methodology

Our research problem can be formulated as: how to quickly transfer a neural network to count cells of different types using minimal amount of data. Our method can be summarized as follow: redundant counting using full convolutional network and residual adapters.

4.1. Redundant Counting

The ground truth label we used for the cell images is the redundant count map [1]. The core concept of this method is to split the original input image into small, overlapping sqaure patches and predict the number of cells in each patch separately. Suppose the patch size is $W \times W$ with a stride of S, then with suitable padding on the borders (i.e padding by W), each object is counted for $N = (\frac{W}{S})^2$ times (i.e. the number of patches including it). Therefore, to recover the true count we need to sum up all predictions and divided by N. The reason behind this is that we expect each prediction to produce errors. Thus, by doing this task redundantly we can reduce the random error. This is also the main improvement compared to count using density map.

The count-ception network [1] is a fully convolutional network (FCN) consists of ten convolutional layers using 1×1 and 3×3 filters with stride one (details shown in Figure 2, and batch normalization is performed after each convolutional layer. Each pixel in the output has a receptive field of size 32×32, which is chosen to be the patch size (i.e. $W = 32$) of the redundant count map. By choosing $S = 1$, each pixel in the redundant count map actually gives us the number of cells in its receptive field.

4.2. Adaptive Module

To enable our model to adapt to various domains (e.g. different cell types), a set of domain-specific modules are added.

The original idea of adapter module comes from the fact that, although domains sometimes look fairly different, they may still share a significant amount of low and mid-level visual patterns. Therefore, its a natural to try to decompose the model parameters into two partsa domain-agnostic part and a domain specific part. In our project, we adopt the residual adapter module proposed by Sylvestre et al.[7], shown in Figure 3. Inspired by ResNet, the residual adapter module utilizing the domain-translation ability of batch normalization layer to perform as well or even better than full fine tuning with only a small increasing in the total number of parameters.

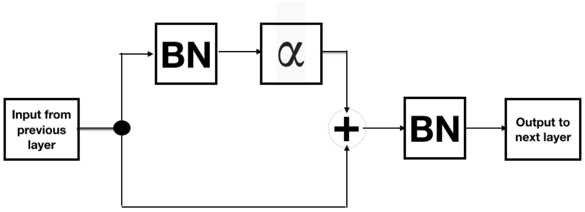


Figure 3: The figure shows a standard adapter module. Here α is selected to be a bank of 1×1 filters to limit the number of domain-specific parameters. Also note that the input and output tensor have the same shape

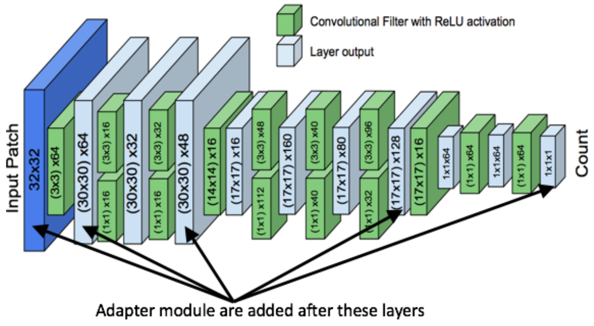


Figure 4: The architecture of our count model.

One advantage of using the adapter module is that this method enables a high degree of parameter sharing while maintaining or even improving the performance of domain-specific tasks. In the pre-train phase, we first freeze the adapter layers and just train our convolutional layers. When there is a need of changing the domain, we then freeze the task-agnostic parameters (pre-trained FCN) and just train on the task-specific parameters (the adapter layers).

4.3. Network Architecture

We use the fully convolutional network adapted from count-ception as our backbone. To perform domain transfer, unlike in [7] where an adapter is added after each layer, we only add five adapter modules at the first and last few layers of our network as shown in Figure 4 and increase only less than 2% of total parameters. Our experiment result demonstrated that it is enough to take care of the translation of both low level and high level features in this way. By experiments, adding too many adapters harms the result.

The count of cells in a image can be obtained in the following way:

- Input image zero padded by $W = 32$ on both sides.
- Feed the padded image to our model and output a redundant count map.
- Sum up the redundant count map and divide by the number of redundant counting. i.e. $N = (\frac{W}{S})^2 = 1024$.

We use zero padding to handle objects close to the border to ensure that each object is counted for the same number of times.

4.4. Loss Function

As pointed out by [8], L2 loss is too harsh to the counting task so we use Mean Absolute Error(MAE) as our loss function.

$$MAE = \frac{\sum_{i=1}^{n} |p_i - y_i|}{n}$$

where n is the number of images, p_i is the predicted count of the $i - th$ image and y_i is the actual count of the $i - th$ image. It is shown that MAE performs well for our task. No regularization is used in our model.

5. Experiment Results

Baselines We consider two cell counting baselines as described below.

- The first baseline is the standard deviation of ground truth cell counts in each dataset. We call this baseline *true std*.
- The second baseline is we train our model from scratch instead on each dataset. We call this baseline *from scratch*.

We choose the first baseline to show that our method actually learned to count in different images instead of just learned about the general statistics of the dataset.

In our proposed method *adapt*,[1] we only train the residual adapters and the batch normalization layers, which consist of only less than 2% of the total number of parameters . We choose the second baseline to show that by utilizing the pre-trained convolutional layers, we can achieve a much better results by only train a very small fraction of parameters.

Results We train all the models for 100 epochs using a batch size of 4 and a learning rate of 0.001, then we choose the best model by validation and report the test results. All experiments are repeated for 10 times by setting different random seeds. The mean and standard deviation of MAE and significance level are reported in Table 2.

On the VGG dataset, training our model from scratch gives a rather satisfying result since the cell images from this dataset is relatively sample. A test sample is shown in Figure 5(a)-(c). Because its simplicity, VGG is very suitable for pre-training our model to capture some of the most basic feature of cells. For example, in the most cases, cells are small round solid objects.

On the MBM dataset, by using the pre-trained weights on VGG dataset, our model significantly outperforms the training from scratch method by a very large margin of 81.3%. We also observed that when we train the model from scratch on the MBM dataset, the train loss is very volatile while when we adapt the model to the MBM dataset, the train loss decreased steadily. This may because that the background of cell images in MBM dataset is much more complex that the synthetic VGG dataset as shown in Figure 5, so it might be hard to separate the cells from the background when we train from scratch. But the pre-trained model has already learned to detect cells as small round solid objects, and all it need to do is to learning to detect cells darker than the background instead of cells lighter than the background .

On our dataset (counting all the cells), by using the pre-trained weights on VGG dataset, our model significantly outperforms the training from scratch method by 18.6%. The improvement of adapting the pre-trained model to our dataset is not as significant as adapting to the MBM dataset might because that the shape of the cells in our dataset varies much more that that in the MBM dataset. And a typical cell in our dataset is not solid actually: it is usually dark in the interior while bright on the boundary, as shown in Figure 5(h).

When it comes to only counting dividing cells on our proposed dataset, adapting pre-trained model does not perform better than training from scratch. This may because that the dividing cells is actually brighter than other kind of cells and pretty standing out. It could be much easier to learn to count dividing cells from scratch than adapt the pre-trained model to ignore the other cells.

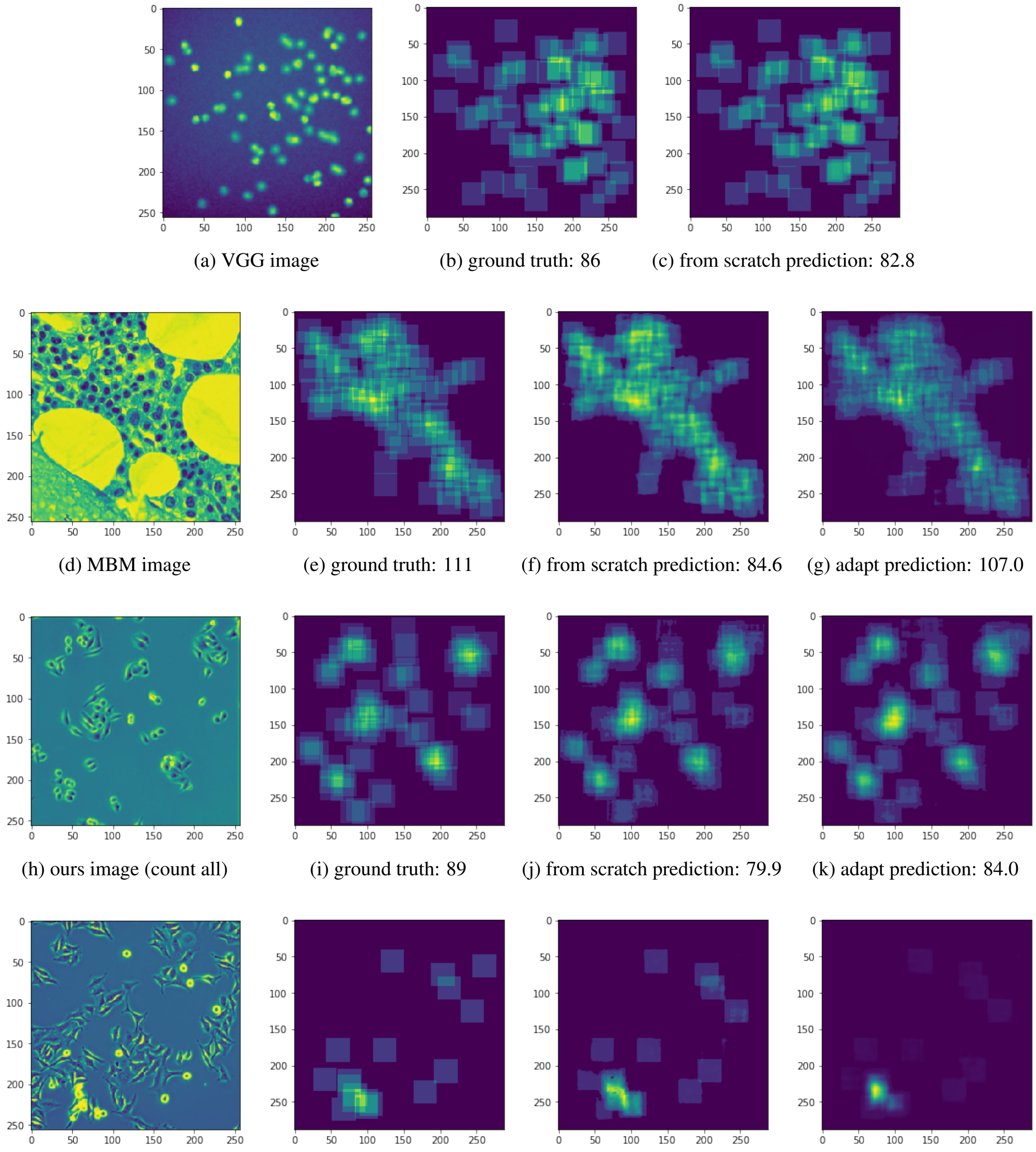
Test Samples Figure 5 shows a test sample from each dataset. (a)-(c) show a test sample from VGG dataset. The ground truth count is 86 cells and the prediction error of the model trained from scratch 3.2 cells.

(d)-(g) show a test sample from MBM dataset. The ground truth count is 111 cells and the prediction error of the model adapted from the pre-trained model (4.0 cells) is significantly better than the prediction error of the model

[1] Our implementation is based on the Keras implementation of count-ception at https://github.com/fizzoo/countception-recreation. Our code is available at https://github.com/WANGXinyiLinda/adaptive-count-ception

Dataset	true std	from scratch	adapt
VGG	56.90	3.25 ± 2.58	*Not Applicable*
MBM	21.10	32.59 ± 5.31	$\mathbf{6.08 \pm 5.98}$*
Ours (count all cells)	18.62	15.15 ± 3.70	$\mathbf{11.86 \pm 4.11}$*
Ours (count dividing cells)	10.00	1.87 ± 1.96	8.34 ± 12.21

Table 2: MAE of cell counts by training our model from scratch and adapting our model to different datasets. An asterisk means statistically significant result compared to train from scratch at 5% level under a one-tailed t-test ($p < 0.05$).



(a) VGG image (b) ground truth: 86 (c) from scratch prediction: 82.8

(d) MBM image (e) ground truth: 111 (f) from scratch prediction: 84.6 (g) adapt prediction: 107.0

(h) ours image (count all) (i) ground truth: 89 (j) from scratch prediction: 79.9 (k) adapt prediction: 84.0

(l) ours image (count dividing) (m) ground truth: 18 (n) from scratch prediction: 19.7 (o) adapt prediction: 43.4

Figure 5: (a)-(c): a test example from the VGG dataset. (d)-(g): a test example from the MBM dataset. (h)-(k): a test example from our proposed dataset (count all cells). (l)-(o): a test example from our proposed dataset (count dividing cells only).

trained from scratch (26.4 cells).

(h)-(k) show a test sample counting all the cells from our dataset. The ground truth count is 89 cells and the prediction error of the model adapted from the pre-trained model (5.0 cells) is significantly better than the prediction error of the model trained from scratch (9.1 cells).

(h)-(k) show a test sample counting only the dividing cells all the cells from our dataset. The ground truth count is 18 cells and the prediction error of the model adapted from the pre-trained model (25.4 cells) is worse than the prediction error of the model trained from scratch (1.7 cells).

Limitations and Future Works A problem we encountered is that the adaptive layers cannot handle objects of different scales. At the beginning, we tried to slice our original 1024×1024 images directly as input but this gives bad result. Therefore we first down-sampled them to 512×512 and then slice them into pieces. In this case the performance is much better and we noticed that the size of cells in our down-sampled image is similar to that in pre-training data. This problem is quite natural since most neural networks have a common defect that they are scale-sensitive. One possible solution is to build feature pyramids during pre-training according to the method proposed by Lin et. al.[4].

6. Conclusion

In this work we improved the count-ception network by separating the domain-specific parameters from the domain-agnostic parameters and therefore it can perform domain transfer with only a small fraction of trainable parameters. We demonstrate that our model using adapter outperforms the model trained from scratch and can perform well even for very complicated cell structures. However, the method has some limitations. When the new domain is of significant different scale or our task changed significantly, for example only to count some of the cells, the adapter doesn't perform as well. In this case, training from scratch is a better solution.

References

[1] J. P. Cohen, G. Boucher, C. A. Glastonbury, H. Z. Lo, and Y. Bengio. Count-ception: Counting by fully convolutional redundant counting. In *International Conference on Computer Vision Workshop on BioImage Computing*, 2017.

[2] P. Kainz, M. Urschler, S. Schulter, P. Wohlhart, and V. Lepetit. You should use regression to detect cells. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 276–283, 2015.

[3] V. Lempitsky and A. Zisserman. Learning to count objects in images. In *Advances in Neural Information Processing Systems*, 2010.

[4] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016.

[5] E. Lu, W. Xie, and A. Zisserman. Class-agnostic counting. *CoRR*, abs/1811.00472, 2018.

[6] M. Marsden, K. McGuinness, S. Little, C. E. Keogh, and N. E. O'Connor. People, penguins and petri dishes: Adapting object counting models to new visual domains and object types without forgetting. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8070–8079, 2018.

[7] S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *Advances in Neural Information Processing Systems*, 2017.

[8] W. Xie, J. A. Noble, and A. Zisserman. Microscopy cell counting and detection with fully convolutional regression networks. *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 2016.