End-to-End Bokeh Effect Synthesis Based on PyNET

Zhe Xiong u7150030 Chen Yang u7201888 Zhoushu He u5926302

Abstract

Bokeh is the out-of-focus area in an image. It emphasizes the main object by aesthetically blurring distracting objects in the background. The bokeh effect can be naturally captured by a DSLR equipped with a fast lens. However, the camera of the smartphone cannot achieve the natural bokeh effect due to the small aperture. In this paper, we address the problem of rendering the natural bokeh effect with a single all-in-focus image by reproducing the work of Ignatov et al.[5]. In experiments, both quantitative and qualitative results show that our work is close to the original work, except to output image size and sharpness of the foreground object. In an extended experiment, We further study the role of each level in the original model.

1. Introduction

The bokeh or bokeh effect usually means the out-of-focus area in an image. Visually, the out-of-focus area shows a blur and soft edges on the image, in contrast, the in-focus area shows a sharp and distinct edge. In professional photography, the bokeh effect can emphasize the main object and reduce the distractions in the background. This effect is widely used in portrait, still life, and commercial photography. Among amateurs, the smartphone is the major device for photography. Compared to transitional DSLR, the smartphone is more accessible, compact, and less expensive. However, the compact design of smartphones limits the functions of photography, and the bokeh effect is one of them. Technically, the bokeh effect requires the shallow depth-of-field, which is mainly manipulated by the state of the aperture. When the aperture of the lens is wide open, the shallow depth-of-field is activated in the camera. Usually, the apertures ranging from $f/1.2$ to $f/2.8$ generate an aesthetic bokeh in the out-of-focus area in the image. However, such a wide aperture cannot fit into the compact body of smartphones. Therefore, the bokeh effect is optically unachievable on devices, such as smartphones, with small apertures.

To address this problem, we reproduce and modify the work originally proposed [5]. Our contribution can be summarized as,

- The original repository[1] written with Tensorflow is fixed and updated. We substitute deprecated functions in the original repository, fix bugs on model saving, loading, and image shape inconsistency, and write valid environment lists for the following researchers.
- The original repository is rewritten in PyTorch based on a PyNET repository[2] which addresses a different problem of converting RAW to RGB image.
- A MegaDepth repository[3] is modified to generate the required depth map as the original paper prescribed, due to the method for generating depth map being missing in the original repository.
- The training scheme is refactored to meet the limited time/storage sources.

2. Related Work

Recently, the topic of bokeh effect synthesis draws attention to both academic and industry communities. Shen *et al.*[12] firstly approach this problem by a proposed automatic segmentation algorithm. They firstly use a modified FCN to segment the portrait from the background, then apply a filter to the background for Stylization. Since the focus of this paper is image matting, they only uniformly blur the background to achieve the bokeh effect. Later, a similar framework can also be found [16, 13]. However, they share a common drawback. They all assume the main object in the image is a human portrait. Therefore, their method is hard to generalize to the other objects.

The depth map is often involved in the task of bokeh effect synthesis. In [3], the depth map is used to render the bokeh effect. However, the generation process requires the user to adjust the camera angle and take multiple shots. Wadhwa *et al.*[13] proposed a method which only needs one shoot. The first retrieves the depth map by a dual-pixel sensor, then the depth map is combined with human segmentation for high-quality bokeh rendering. However, only the high-end mobile phone is equipped with dual-pixel sensors. Dutta [1]

[1] https://github.com/aiff22/PyNET-Bokeh
[2] https://github.com/aiff22/PyNET-PyTorch
[3] https://github.com/yunlongdong/MegaDepthEval

proposed an end-to-end network to address this problem. In the proposed method, a depth-estimation network is used to generate a set of weights that combine sharp input images with different blurred input images to generate a final image with a visually plausible bokeh effect. Purohit *et al.*[10] leverage MegaDepth[9] to generate depth maps from the single input images, and the method in [4] to generate a salient segmentation map. Then, two maps are used as prior knowledge to train a dynamic filter network[7] to render the bokeh effect.

Considering successes of Generative Adversarial Networks(GANs) on the task of image deblurring and style transferring, Qian *et al.*[11] adopted a GANs-based network to synthesis the bokeh effect from single image input. Visually, this method even renders the "spot effect" in the out-of-focus areas. This method also shows a state-of-art rendering speed on the existing smartphone chipset. In AIM 2020 Rendering Realistic Bokeh Challenge[15], this method is ranked first on both CPU and GPU tracks.

3. Proposed Method

This paper attempts to reproduce the work in [5]. Therefore, most parts of the method refer to the method proposed in the original paper. There are some small modifications on loss function and training schedule due to the limited computational resources and time.

3.1. Dataset

The Everything is Better with Bokeh!(EBB!) the dataset is used to address the problem. This dataset includes 4694 high-resolution image pairs, which are taken with Canon 7D DSLR equipped with a standard 50mm lens. In each pair, an all-in-focus image is taken with a narrow aperture of $f16$, and an image with a bokeh effect is taken with a wide-open aperture of $f1.8$. The images are captured in wild during the daytime with varied object, scene, and weather conditions. The preprocess methods of SIFT and RANSAC are used to align image pairs, then the images are cropped to a height equal to 1024. The MegaDepth[9] is used to generate the depth map from each all-in-focus image, then the depth map is concatenated to input images as a guide for better bokeh rendering. In Figure 1, the images in the first row are the all-in-focus images, the second row represents the bokeh images, and the last row represents the depth maps generated by the method of MegaDepth.

3.2. Architecture of PyNET

As illustrated in Figure 2, the PyNET has a multi-scale CNN architecture, and it has 47.5 million parameters. In this inverted pyramid structure, there are seven levels. The level with a higher number is responsible for more holistically rendering, and the level with a lower number is responsible for more detailed rendering. At the left side of the network, there are successive down-sampling operations. In the contrast, there are successive up-sampling operations on the right side. Except for up/downsampling, the features in the lower scale level are fed into lower number levels by concatenation. The higher level has a more complicated structure than the lower level. At a higher level, the input features are processed parallel in a convolutional layer with different kernel sizes from 3×3 to 9×9, then the output of those layers is concatenated. This structure adds diversity to networks at each level during learning. Leaky ReLU is used as the activation function for the convolutional layers, except for the output layer at each level. The activation function for output layers is a weighted tanh. From level 2 to level 7, the instance normal layers add to each convolutional layer.

The input image is resized and cropped to size 512×512, After four down-sampling operations, the size of the input feature for level 7 is 32×32, and the output size of level 7 is also 32×32. As the level increase, the size of output for each level is increased as the result of up-sampling. At level 3, the size of the output is 512×512. Level 1 and level 2 are two transposed convolutional layers, and they are designed for up-sampling the output to the size before transformations.

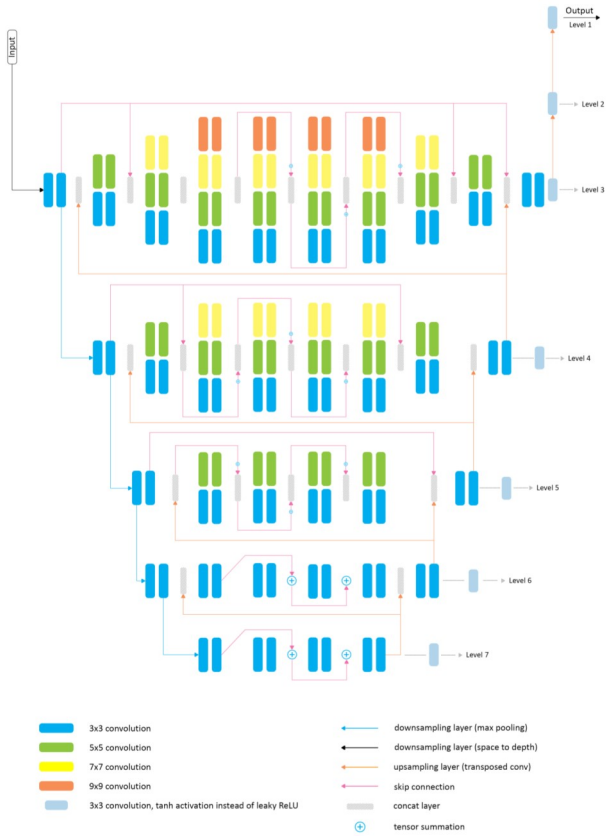

Figure 2: Architecture of PyNET[5]

Figure 1: Sample images in EBB! dataset and corresponding depth maps

3.3. Training Details

Loss Function The network is trained layer-wisely, and training is started from level 7. From level 2 to level 7, the network is trained with $L1$ loss. For level 1, the network train with the loss as,

$$L_{\text{output}} = L1 + (1 - L_{\text{SSIM}}) + 0.01 \cdot L_{\text{VGG}} \quad (1)$$

where, L_{SSIM} is the SSIM loss[14], which measures the structural similarity between the input image and the rendered image. L_{VGG} is the perceptual VGG-based loss[8], and this loss can be defined as,

$$L_{VGG/i.j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} \left(\phi_{i,j}\left(I^{HR}\right)_{x,y} - \phi_{i,j}\left(G_{\theta_G}\left(I^{LR}\right)_{x,y}\right)^2 \right. \quad (2)$$

where, I^{HR} is the ground truth image, and the I^{LR} is the input image. The $\phi_{i,j}(\cdot)$ maps input image to a feature map obtained after $j\text{-}th$ convolutional layer and before $i\text{-}th$ max-pooling layer in a pretrained VGG-19 network. The $\frac{1}{W_{i,j}H_{i,j}}$ is the normalization term. In this paper, the feature map directly after $16\text{-}th$ convolutional layer(after activation) is used for calculating the perceptual VGG-based loss.

Note that, the 64GB RAM of the device on serve is not enough for training level 1. Therefore, the level is abandoned due to the limited RAMs resources, and the loss function is adjusted accordingly. In our case, the loss function for level 2 is L_{output} in Equation (1), and the loss functions for training the rest levels are $L1$ loss.

Device The model is train on a rented serve[4] with single *NVIDIA Tesla V100-FHHL-16GB* GPU, *12× Xeon Platinum 8260* CPU, and 64GB RAM.

Training Scheme The training scheme in the original paper prescribes 5,000 to 20,000 iterations for training lower scale level, and 100,000 iterations for training the output level. The suggested batch size varied from 50 to 5, as the scale increased. Also, the ADAM optimizer with $5e^{-5}$ learning rate is used in the original paper.

Due to the size of the input image and the number of parameters in PyNET, the estimated time for training 1,000 iterations for lower scale level is about 20 to 30 minutes. As the scale of the level increases, the training time increases dramatically. It takes more than one hour to train 1,000 iterations for the output-level(level 2). Considering the time constraint of the project, the original training scheme is adjusted accordingly.

The training scheme in Table 1 is adjusted based on the original training scheme. The training for level 1 is dropped because of the computational limitation. In order to replace level 1 in the original model, level 2 is trained with the L_{output} loss instead of $L1$ loss. The drawback of this replacement is that the size of the final output image will be downscaled to 512. The learning rate for training levels 5 and

[4] https://www.matpool.com/

Figure 3: Qualitive evaluation results

Level	Batch Size	Learning Rate	Iterations	Loss
7	50	$5e^{-5}$	5,000	$L1$
6	50	$5e^{-5}$	5,000	$L1$
5	40	$1e^{-4}$	5,000	$L1$
4	14	$1e^{-4}$	5,000	$L1$
3	9	$5e^{-5}$	10,000	$L1$
2	5	$5e^{-5}$	40,000	L_{output}

Table 1: Training scheme

4 are increased to partly compensate for the decrease in the number of iterations. Considering the training stability[2] at the beginning of the training, The learning rate at levels 7 and 6 are unchanged. With this training scheme, It takes less than 80 hours to train the entire model.

4. Experiments

In this section, the reproduced results[5] are evaluated on the test set with 200 image pairs. Besides, an analysis is conducted on the behaviors of each level in the PyNET.

4.1. Evaluation Results

In evaluation, our reproduced model is compared to a pre-trained model provided by authors. Both models are evaluated on a predefined test set with 200 original-bokeh image pairs. The Peak Signal-to-Noise Ratio(PSRN), SSIM, and multi-scale SSIM(MS-SSIM)[14] are used as metrics in quantitative evaluation. The evaluation results are shown as,

[5]Only raw experiment results are shared and explained to the project team. Different people may choose different results to present and have different interpretations on their results

Method	PSRN	SSIM	MS-SSIM
original method	19.93	0.8098	0.713
our method	19.96	0.7433	0.7105

Table 2: quantitative evaluation results

In Table 2, the metrics are given by our method and the original method are close. The differences in quantitative results root in different training schemes, network architecture, and dataset partitioning.

The metrics in the evaluation can only be taken as a basic reference since those metrics are originally used in image quality assessment. The quantitative metric for measuring the bokeh effect is still absent. Xiong *et al.*[6] achieve the highest PSRN and SSIM scores in Aim 2019 Challenge on bokeh effect synthesis, however their rendered images have significant artifacts on the edge of focused objects. Therefore, qualitative measurements are required for further evaluation.

The Figure 3 compares the rendering quality between our method and the original method. The image in the first row is the input images, which are shaped in both object and background. The second row shows the rendering images generated with a pre-trained model provided by authors. The third row shows the rendering images generated with our proposed method. Overall, the results of our model represent a visually plausible bokeh effect. The bokeh area of our method is almost visually identical to the bokeh area of the original method. However, there are two drawbacks to the results of our method. First, the size of our results is half of the original results, since the level 1 layers in PyNET are dropped in our method. Second, the foreground objects from our results are less sharper than the foreground objects from the original results. This drawback may root in insufficient training iterations.

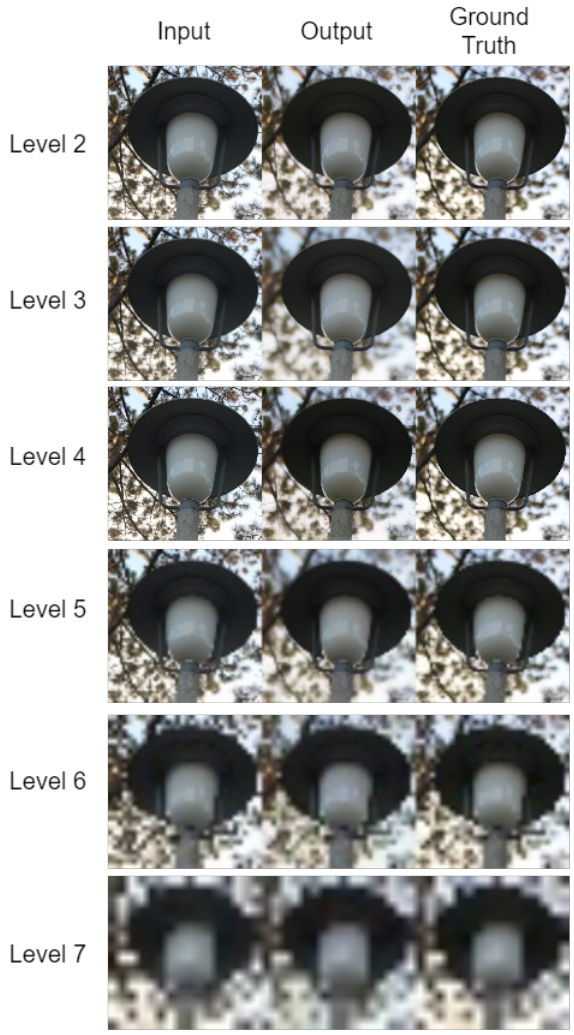

Figure 4: Output images from each level at the end of the training

4.2. Network Analysis

To better understand the rendering process of the PyNET, an analysis of the outputs of each layer is conducted. In Figure 4, the output images for each level at the end of training are presented in the second column. The input images for each layer and their corresponding bokeh pair are presented in the first and third columns. Note that the image size at each level is different. Level 2 outputs the largest images with size 512×512, and level 7 outputs the smallest images with size 16×16. For illustration, all images are set to an identical size.

The output images from levels 6 and 7 are low in resolution, therefore it is hard to recognize visually what rendering job is done by those two levels. But from the images in level 5, we can observe that the output image is globally blurry. Especially, light poles are sharp in both input and ground truth images, but the pole is blurry in the output image. We can inference that levels 5 to 7 are responsible for rendering a global bokeh effect on the input images. From level 4, the foreground object in the output image starts to become sharper. In level 3, even more, defined details are shown on the foreground object. The foreground objects in level 3 output and level 2 output are almost identical, however, there is a significant difference in background. The background in level 3 output looks like a Gaussian blur. In the background of level 2 output, however, the fine details in background twigs and leaves are enhanced. The bokeh effect in the level 2 output image is even more visually plausible.

From these experiments, we can conclude that the level 5 to 7 of the PyNET are mainly responsible for rendering a global bokeh effect on an input image, the levels 3 and 4 are responsible for defining the foreground object and handling the edges between in-focus and out-of-focus areas, the level 2 takes one step further to refine the background bokeh, and makes it more close to the real bokeh.

5. Conclusion

In this paper, we attempt to reproduce the work in [5], which addresses the problem of bokeh effect synthesis from a single image. Due to the time and computational limitations, the original model and training details are modified in our work. Both quantitative and qualitative results in experiments show that the performance of our reproduced work is very close to the original work. Extended experiments are conducted to study the behaviors of PyNET at each level. The results show that the lower levels are responsible for rendering the global bokeh effect, and higher levels focus on foreground object sharping and background refining.

References

[1] Saikat Dutta. Depth-aware blending of smoothed images for bokeh effect generation. *Journal of Visual Communication and Image Representation*, 77:103089, 2021.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[3] Carlos Hernández. Lens blur in the new google camera app. *Retrieved from Google reasearch blog: http://googleresearch. blogspot. mx/2014/04/lens-blur-in-new-google-cameraapp. html*, 2014.

[4] Qibin Hou, Ming-Ming Cheng, Xiaowei Hu, Ali Borji, Zhuowen Tu, and Philip HS Torr. Deeply supervised salient object detection with short connections. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3203–3212, 2017.

[5] Andrey Ignatov, Jagruti Patel, and Radu Timofte. Rendering natural camera bokeh effect with deep learning.

In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 418–419, 2020.

[6] Andrey Ignatov, Jagruti Patel, Radu Timofte, Bolun Zheng, Xin Ye, Li Huang, Xiang Tian, Saikat Dutta, Kuldeep Purohit, Praveen Kandula, et al. Aim 2019 challenge on bokeh effect synthesis: Methods and results. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3591–3598. IEEE, 2019.

[7] Xu Jia, Bert De Brabandere, Tinne Tuytelaars, and Luc V Gool. Dynamic filter networks. *Advances in neural information processing systems*, 29:667–675, 2016.

[8] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer, 2016.

[9] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018.

[10] Kuldeep Purohit, Maitreya Suin, Praveen Kandula, and Rajagopalan Ambasamudram. Depth-guided dense dynamic filtering network for bokeh effect rendering. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pages 3417–3426. IEEE, 2019.

[11] Ming Qian, Congyu Qiao, Jiamin Lin, Zhenyu Guo, Chenghua Li, Cong Leng, and Jian Cheng. Bggan: Bokeh-glass generative adversarial network for rendering realistic bokeh. In *European Conference on Computer Vision*, pages 229–244. Springer, 2020.

[12] Xiaoyong Shen, Aaron Hertzmann, Jiaya Jia, Sylvain Paris, Brian Price, Eli Shechtman, and Ian Sachs. Automatic portrait segmentation for image stylization. In *Computer Graphics Forum*, volume 35, pages 93–102. Wiley Online Library, 2016.

[13] Neal Wadhwa, Rahul Garg, David E Jacobs, Bryan E Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. *ACM Transactions on Graphics (ToG)*, 37(4):1–13, 2018.

[14] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[15] Kai Zhang, Martin Danelljan, Yawei Li, Radu Timofte, Jie Liu, Jie Tang, Gangshan Wu, Yu Zhu, Xiangyu He, Wenjie Xu, et al. Aim 2020 challenge on efficient super-resolution: Methods and results. In *European Conference on Computer Vision*, pages 5–40. Springer, 2020.

[16] Bingke Zhu, Yingying Chen, Jinqiao Wang, Si Liu, Bo Zhang, and Ming Tang. Fast deep matting for portrait animation on mobile phone. In *Proceedings of the 25th ACM international conference on Multimedia*, pages 297–305, 2017.

My Reflection

To reproduce the work from a paper is challenging. Many critical implementation details are missing in the original paper. Sometimes the authors' intention can only be inferred from the source code. Unfortunately, the provided codes is written in TensorFlow, which are unfamiliar to us. After hours and hours of reading, discussing, debugging the source code, the implementation of the original work becomes more and more clear.

Our working process can be concluded into four stages. At the first stage, we are read the original paper solely. At the second stage, we attempt to run through the codes provided by the authors. When we work with authors' codes, we find that the authors only provide a brief suggestion on environment setup, and some libraries are deprecated. In order to run the original implementation, I and Chen spend about a week on method substitution and debugging in the original implementation. In the third stage, I am responsible to train the model on the server. There are many problems during training. Because limitation of GPU on the local, some bugs are not triggered during local debugging. Besides that, the I/O problem, the storage problem, and then the resume training problem occasionally happen during the training, and I have to fix them. At the last stage, I propose to rewrite the original implementation in PyTorch. At this stage, all team members are participating. Regrettably, due to the time constraint, we have only tested the validity of our implementations in PyTorch and trained the last level of layers with few epochs.

Peer Review

As for contribution to this project, I confirm that I participated in debugging and training the official implementation, writing the entire `dataset.py`, part of `main.py` in our PyTorch implementation. Also, the MegaDepth implementation for generating a depth map is not provided by the authors. I solely find, test, debugging and modify a MegaDepth repository[6] to fit into our project. I consider my work takes about 40% to 45% of the total project.

Chen Yang participated in debugging the official implementation and solving a couple of major bugs in the original implementation, writing the entire `train.py` and `loadvgg.py` complete the rest of `main.py` in our PyTorch implementation. I consider his work takes about 40% to 45% of the total project.

Zhoushu He is absent from a couple of early meetings on Teams, therefore he does not participate in debugging and training the official implementation. Later, He modified `msssim.py`. I consider his work takes about 10% to 15% of the total project.

[6]https://github.com/yunlongdong/MegaDepthEval

Note that the above contribution is only based on participation in the implementation and experiment. As for report and presentation, we all agree on sharing raw experiment results and discussions, then writing an individual report. As for the presentation, We put in an equal amount of effort.