# Explore Image Deblurring via Encoded Blur Kernel Space — Supplementary material —

Phong Tran[1] Anh Tuan Tran[1,2] Quynh Phung[1] Minh Hoai[1,3]
[1]VinAI Research, Hanoi, Vietnam, [2]VinUniversity, Hanoi, Vietnam,
[3]Stony Brook University, Stony Brook, NY 11790, USA
{v.phongtt15,v.anhtt152,v.quynhpt29,v.hoainm}@vinai.io

## Abstract

*In this supplement material, we further provide some material including architecture details, hyper-parameter tuning, and qualitative results to further analyze our method.*

## 1. Training environment

We implement the proposed method using Pytorch 1.4.0. The experiments are conducted using a single Nvidia RTX 2080 Ti GPU. We train $\mathcal{F}$ and $\mathcal{G}$ for $6 \times 10^6$ iterations on either REDS [7] or GOPRO [6] datasets.

## 2. Architecture choices

Here we illustrate in detail the architecture of each component in our proposed method. Details of the overall network are given in Fig. 2

### 2.1. Preprocessing and Postprocessing blocks

We follow a common practice by using a pre-processing block that downsamples the input image twice and convert it to a feature map of size $64 \times H/4 \times W/4$ at the beginning of both $\mathcal{F}$ and $\mathcal{G}$. We denote it as PreprocessBlock or **PrB** in short. At the end of $\mathcal{F}$, we apply a post-processing block that converts the output feature map of size $64 \times H/4 \times W/4$ back to the image domain. This block is called PostprocessBlock, and denoted as **PoB**. Their architectures are illustrated in Table 1 and Table 2 respectively.

| Layer | Output shape |
|---|---|
| $\text{Conv}(3, 64, 3, 1, 1)$ | $64 \times H \times W$ |
| $\text{Conv}(64, 64, 3, 2, 1)$ | $64 \times H/2 \times W/2$ |
| $\text{Conv}(64, 64, 3, 2, 1)$ | $64 \times H/4 \times W/4$ |
| $\text{ResBlock}(64) \times 10$ | $64 \times H/4 \times W/4$ |

Table 1. Structure of PreprocessBlock.

| Layer | Output shape |
|---|---|
| $\text{ResBlock}(64) \times 20$ | $64 \times H/4 \times W/4$ |
| $\text{Conv}(64, 256, 3, 1, 1)$ | $64 \times H/4 \times W/4$ |
| $\text{PixelShuffle}(2)$ | $64 \times H/2 \times W/2$ |
| $\text{LeakyReLU}(0.1)$ | $64 \times H/2 \times W/2$ |
| $\text{Conv}(64, 256, 3, 1, 1)$ | $256 \times H/2 \times W/2$ |
| $\text{PixelShuffle}(2)$ | $64 \times H \times W$ |
| $\text{LeakyReLU}(0.1)$ | $64 \times H \times W$ |
| $\text{Conv}(64, 64, 3, 1, 1)$ | $64 \times H \times W$ |
| $\text{Conv}(64, 3, 3, 1, 1)$ | $3 \times H \times W$ |

Table 2. Structure of PostprocessBlock.

| Layer | Output shape |
|---|---|
| PreprocessBlock | $64 \times H/4 \times W/4$ |
| $\text{Conv}(128, 64, 7, 1, 1)$ | $64 \times H/4 \times W/4$ |
| $\text{LeakyReLU}(0.1)$ | $64 \times H/4 \times W/4$ |
| $\text{Conv}(64, 128, 3, 2, 1)$ | $128 \times H/8 \times W/8$ |
| $\text{LeakyReLU}(0.1)$ | $128 \times H/8 \times W/8$ |
| $\text{Conv}(128, 256, 3, 2, 1)$ | $256 \times H/16 \times W/16$ |
| $\text{LeakyReLU}(0.1)$ | $256 \times H/16 \times W/16$ |
| $\text{Conv}(256, 512, 3, 2, 1)$ | $512 \times H/32 \times W/32$ |
| $\text{LeakyReLU}(0.1)$ | $512 \times H/32 \times W/32$ |
| $\text{Conv}(512, 512, 3, 2, 1)$ | $512 \times H/64 \times W/64$ |
| $\text{LeakyReLU}(0.1)$ | $512 \times H/64 \times W/64$ |
| $\text{Conv}(512, 512, 3, 2, 1)$ | $512 \times H/128 \times W/128$ |
| $\text{LeakyReLU}(0.1)$ | $512 \times H/128 \times W/128$ |
| $\text{ResBlock}(512) \times 4$ | $512 \times H/128 \times W/128$ |

Table 3. Structure of $\mathcal{G}$

### 2.2. Architecture of $\mathcal{G}$

We use $\mathcal{G}$ to extract the blur kernel $k$ from a given sharp-blur pair of images $x, y$. We implement $\mathcal{G}$ using the mentioned PreprocessBlock and a follow-up residual neural network [1]. Input of $\mathcal{G}$ is the concatenation of $x$ and $y$. Its output is a blur kernel of size $512 \times H/128 \times W/128$. Details of its architecture are given in Table 3.

| Encoder | |
|---|---|
| Layer | Output shape |
| PreprocessBlock | $64 \times H/4 \times W/4$ |
| Conv$(64, 64, 3, 2, 1)$ | $64 \times H/8 \times W/8$ |
| LeakyReLU$(0.1)$ | $64 \times H/8 \times W/8$ |
| Conv$(64, 128, 3, 2, 1)$ | $128 \times H/16 \times W/16$ |
| LeakyReLU$(0.1)$ | $128 \times H/16 \times W/16$ |
| Conv$(128, 256, 3, 2, 1)$ | $256 \times H/32 \times W/32$ |
| LeakyReLU$(0.1)$ | $256 \times H/32 \times W/32$ |
| Conv$(256, 512, 3, 2, 1)$ | $512 \times H/64 \times W/64$ |
| LeakyReLU$(0.1)$ | $512 \times H/64 \times W/64$ |
| Conv$(512, 512, 3, 2, 1)$ | $512 \times H/128 \times W/128$ |
| LeakyReLU$(0.1)$ | $512 \times H/128 \times W/128$ |
| Decoder | |
| Layer | Output shape |
| TransConv$(1024, 512, 3, 2, 1)$ | $512 \times H/64 \times W/64$ |
| LeakyReLU$(0.1)$ | $512 \times H/64 \times W/64$ |
| TransConv$(1024, 256, 3, 2, 1)$ | $256 \times H/32 \times W/32$ |
| LeakyReLU$(0.1)$ | $256 \times H/32 \times W/32$ |
| TransConv$(512, 128, 3, 2, 1)$ | $128 \times H/16 \times W/16$ |
| LeakyReLU$(0.1)$ | $128 \times H/16 \times W/16$ |
| TransConv$(256, 64, 3, 2, 1)$ | $64 \times H/8 \times W/8$ |
| LeakyReLU$(0.1)$ | $64 \times H/8 \times W/8$ |
| TransConv$(128, 64, 3, 2, 1)$ | $64 \times H/4 \times W/4$ |
| LeakyReLU$(0.1)$ | $64 \times H/4 \times W/4$ |
| PostprocessBlock | $64 \times H \times W$ |

Table 4. Structure of the encoder and decoder of $\mathcal{F}$

### 2.3. Architecture of $\mathcal{F}$

$\mathcal{F}$ takes two inputs, the sharp image $x$ and the blur kernel from $\mathcal{G}(x, y)$. As mentioned, $\mathcal{F}$ uses a PreprocessBlock at the beginning and a PostprocessBlock at the end. Between these blocks, we use an encoder-decoder with skip connection [9]. The encoder downsamples the pre-processed feature map five times and flattens to an embedding vector. This vector is then concatenated with $k$ and fed into a decoder that reconstructs the output feature map. Details of its architecture are illustrated in Table 4.

### 2.4. Architectures of Deep Image Prior

We adopt the architecture of $\mathcal{G}$ for the network of DIP of the blur kernel. The input $z_k$ is a normal-distributed random tensor with the size equal to the size of the input of $\mathcal{G}$.

For DIP for image, we adopt a U-net [9] as suggested in [12]. The input $z_x$ is a normal-distributed random tensor with size $1 \times 64 \times 64$.

## 3. Hyper-parameters tuning

We trained the networks with an Adam optimizer [2] with $\beta_1$ and $\beta_2$ are 0.9 and 0.99 respectively. The initial learning rate was $10^{-4}$ with cosine annealing scheduler [5] was applied. We set the weight of kernel regularization $||k||_2$ to $6 \times 10^{-4}$ for all image debluring experiments. The weight of Hyper-Laplacian prior [3] was set to $2 \times 10^{-2}$.

## 4. Cross-dataset experiment

Here we provide quantitative comparisons on GOPRO and HIDE dataset [10] in Table 5. We train the model using GOPRO dataset and test on HIDE dataset and vice versa. To make the testing sets, we randomly sample 500 images from each GOPRO and HIDE testing set. Qualitative results are given in Fig. 1.

| | DeblurGANv2 [15] | SRN-Deblur [36] | ours |
|---|---|---|---|
| GOPRO | 24.35 | 25.21 | **26.17** |
| HIDE | 24.65 | 25.25 | **25.97** |

Table 5. PSNR scores of deblurring methods on the HIDE and GOPRO datasets.

Figure 1. Deblurring results (left: SRN, right: ours) on HIDE dataset

## 5. Inference time

We trained the kernel extractor using an Nvidia V100 with 5GB memory. It took $600K$ iterations to converge (about 4 days). The average inference time for a $256 \times 256$ image using an Nvidia V100 is 209.53s.

## 6. More qualitative results

Here we provide more qualitative results of our methods including: Blur transferring (Fig. 3 and Fig. 4) and image deblurring on face domain (Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10, Fig. 11, and Fig. 12).

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceed-*

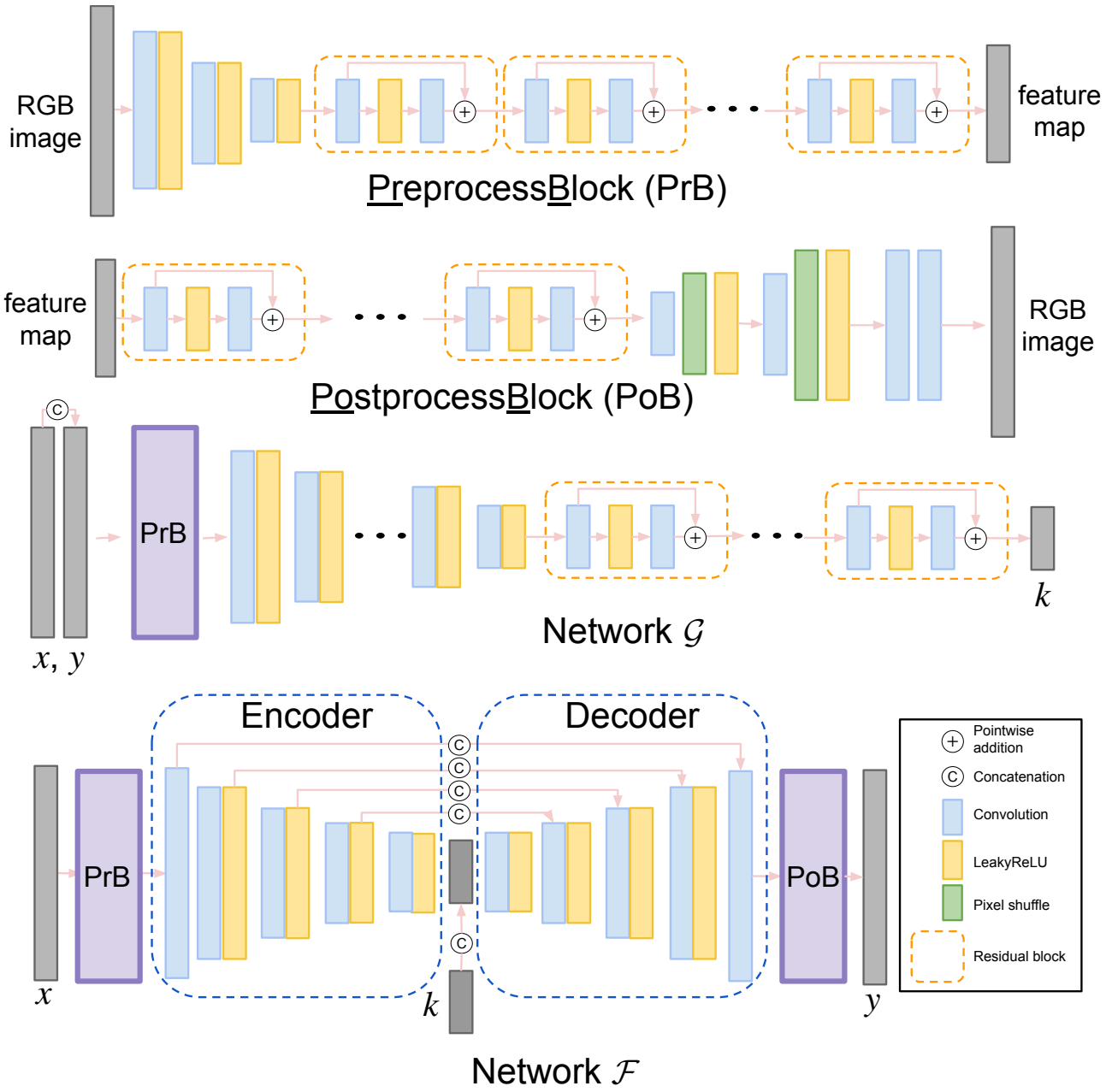


Figure 2. Detailed architecture of the proposed method

*ings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] Dilip Krishnan and Rob Fergus. Fast image deconvolution using hyper-laplacian priors. In *Advances in Neural Information Processing Systems*, 2009.

[4] Orest Kupyn, Tetiana Martyniuk, Junru Wu, and Zhangyang Wang. Deblurgan-v2: Deblurring (orders-of-magnitude) faster and better. In *Proceedings of the International Conference on Computer Vision*, 2019.

[5] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[6] Seungjun Nah, Tae Hyun Kim, and Kyoung Mu Lee. Deep multi-scale convolutional neural network for dynamic scene deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[7] Seungjun Nah, Sungyong Baik, Seokil Hong, Gyeongsik Moon, Sanghyun Son, Radu Timofte, and Kyoung Mu Lee. Ntire 2019 challenge on video deblurring and super-


Figure 3. Transferring blur kernel from the source pair $x, y$ to the target sharp $\hat{x}$ to generate the target blurry image $\hat{y}$

resolution: Dataset and study. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019.

[8] Dongwei Ren, Kai Zhang, Qilong Wang, Qinghua Hu, and Wangmeng Zuo. Neural blind deconvolution using deep priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020.

[9] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 2015.

[10] Ziyi Shen, Wenguan Wang, Jianbing Shen, Haibin Ling,

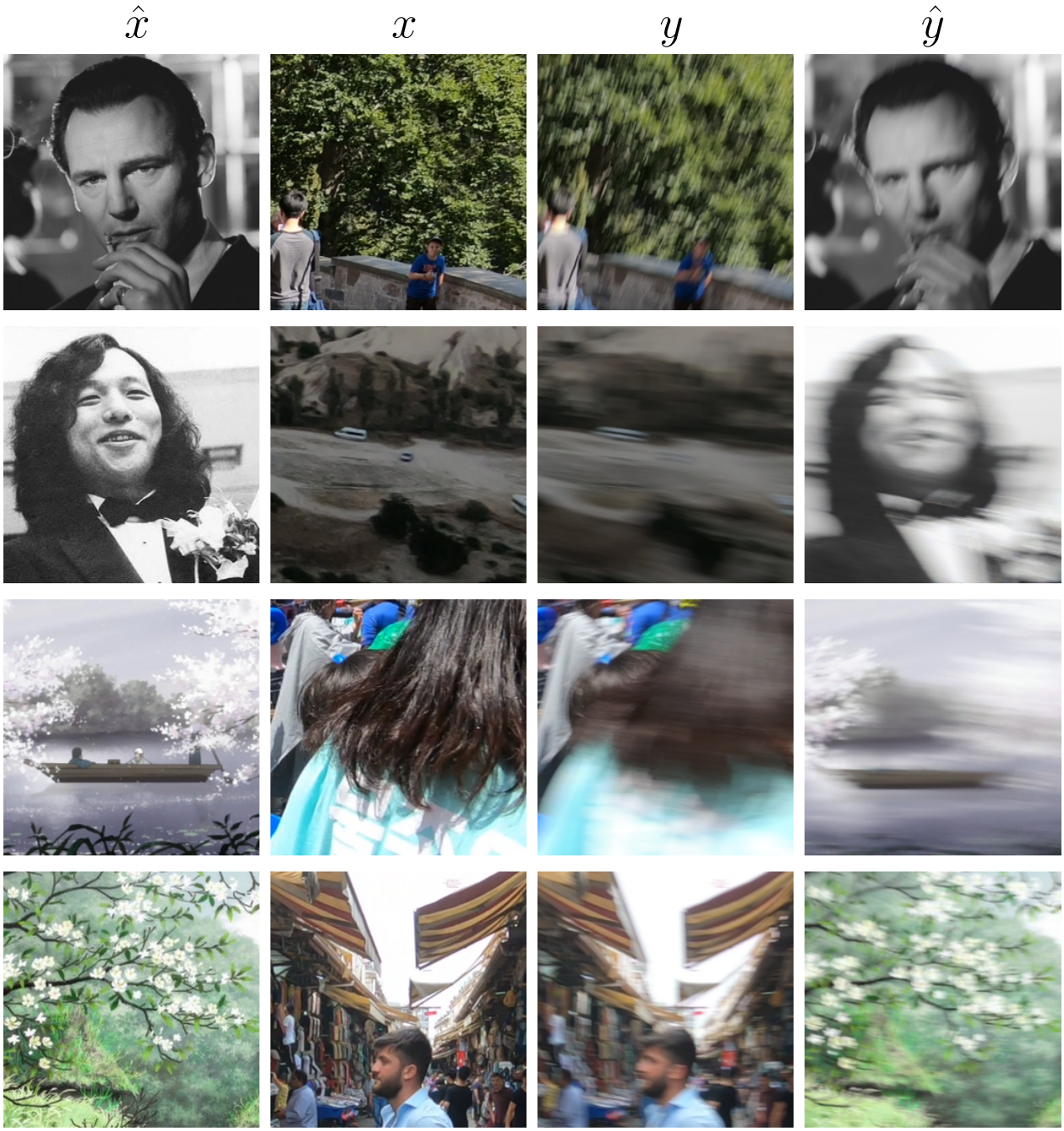


Figure 4. transferring blur kernel from the source pair $x, y$ to the target sharp $\hat{x}$ to generate the target blurry image $\hat{y}$

Tingfa Xu, and Ling Shao. Human-aware motion deblurring. In *IEEE International Conference on Computer Vision*, 2019.

[11] Xin Tao, Hongyun Gao, Xiaoyong Shen, Jue Wang, and Jiaya Jia. Scale-recurrent network for deep image deblurring. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[12] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Deep image prior. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

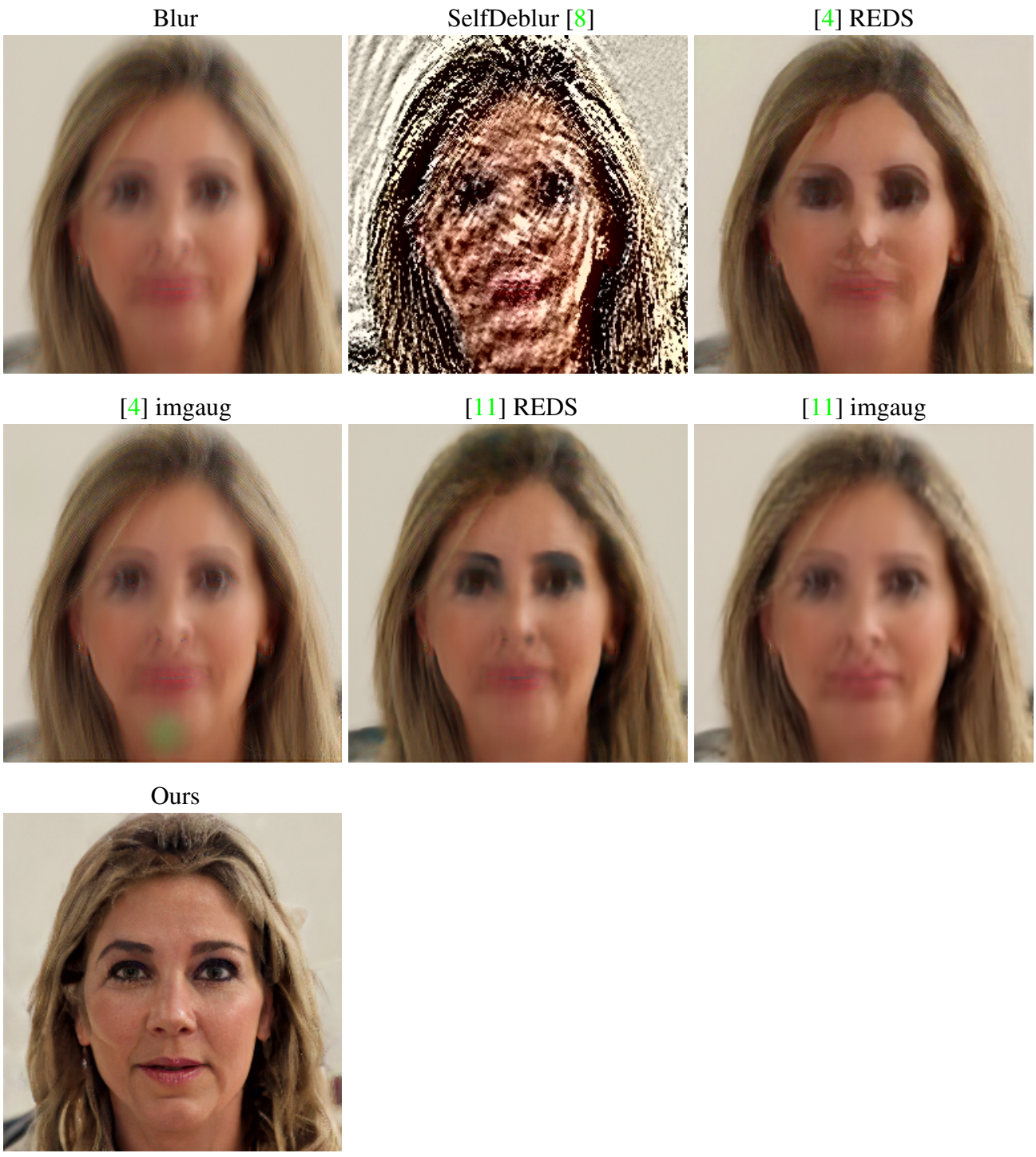


Figure 5. Results of deblurring methods trained on REDS and tested on GOPRO

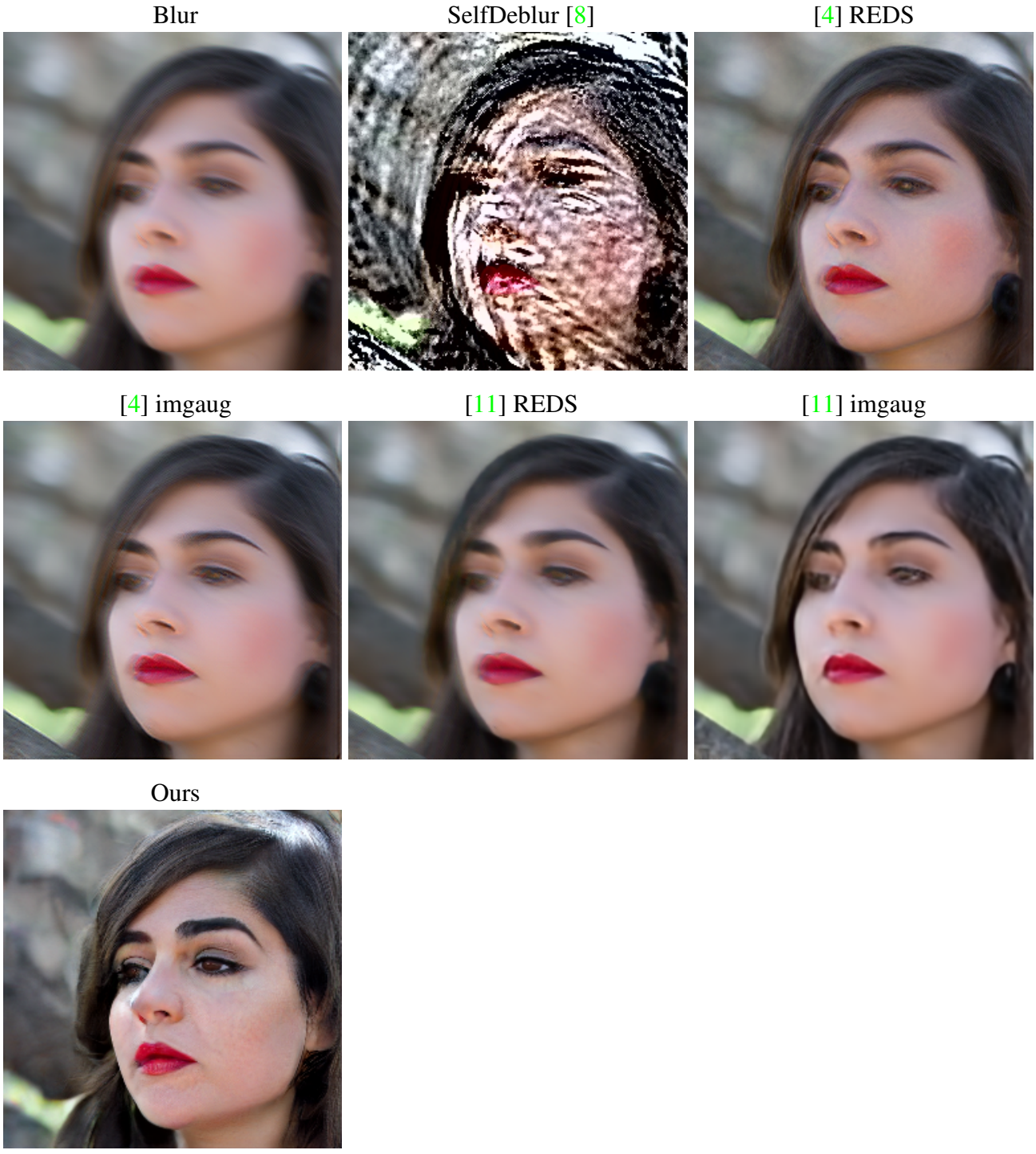


Figure 6. Results of deblurring methods trained on REDS and tested on GOPRO

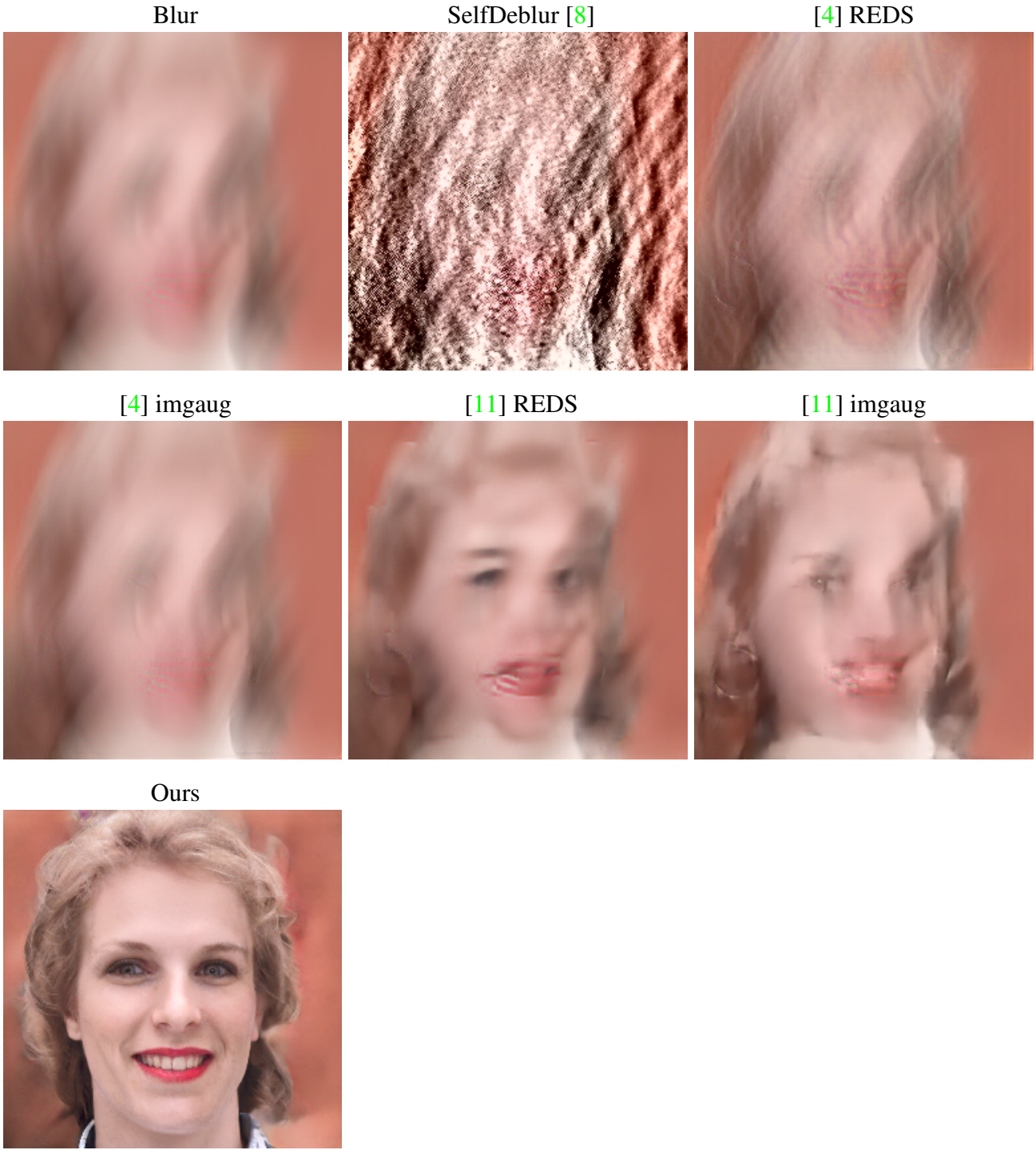


Figure 7. Results of deblurring methods trained on REDS and tested on GOPRO

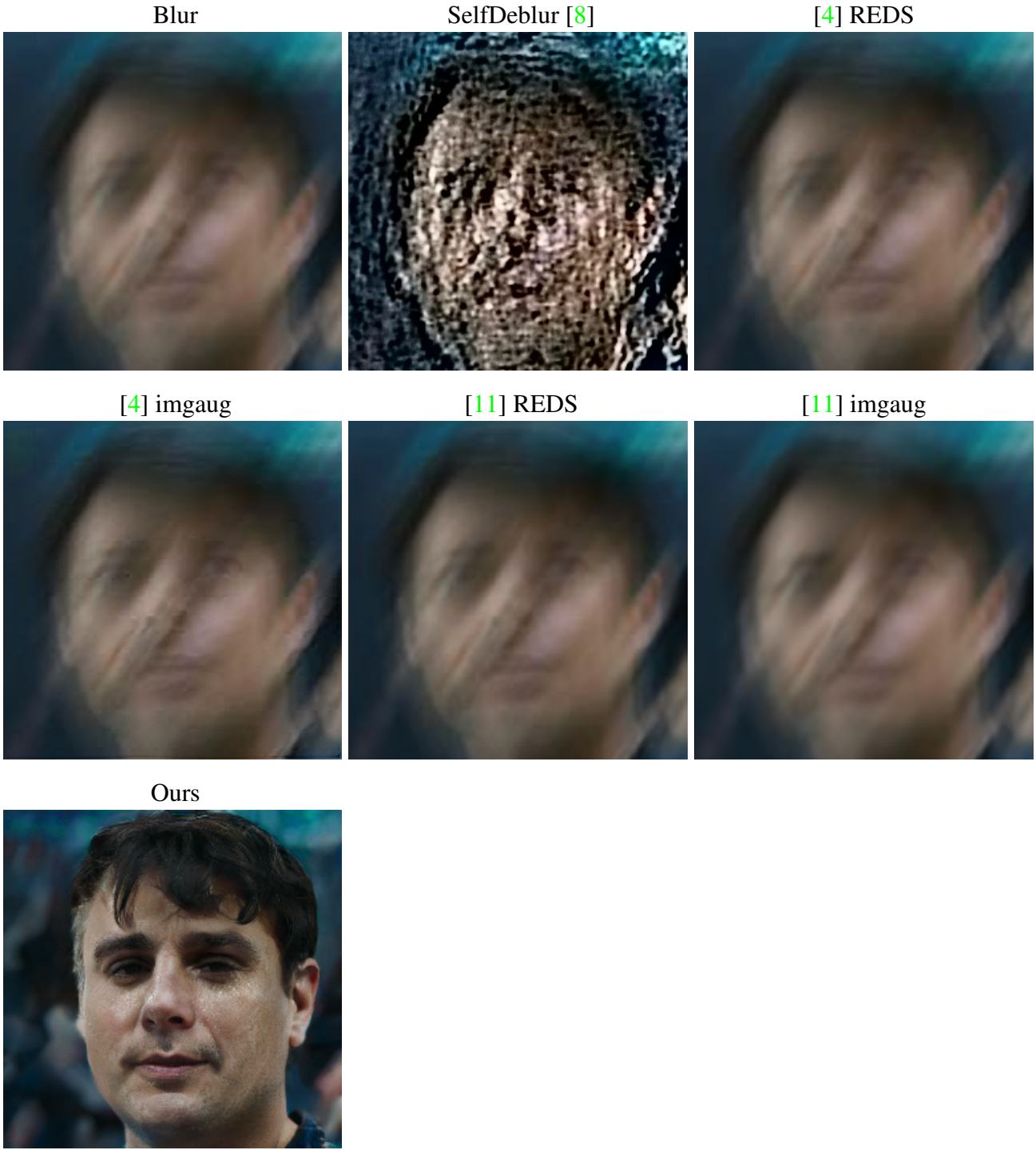


Figure 8. Results of deblurring methods trained on REDS and tested on an in-the-wild example

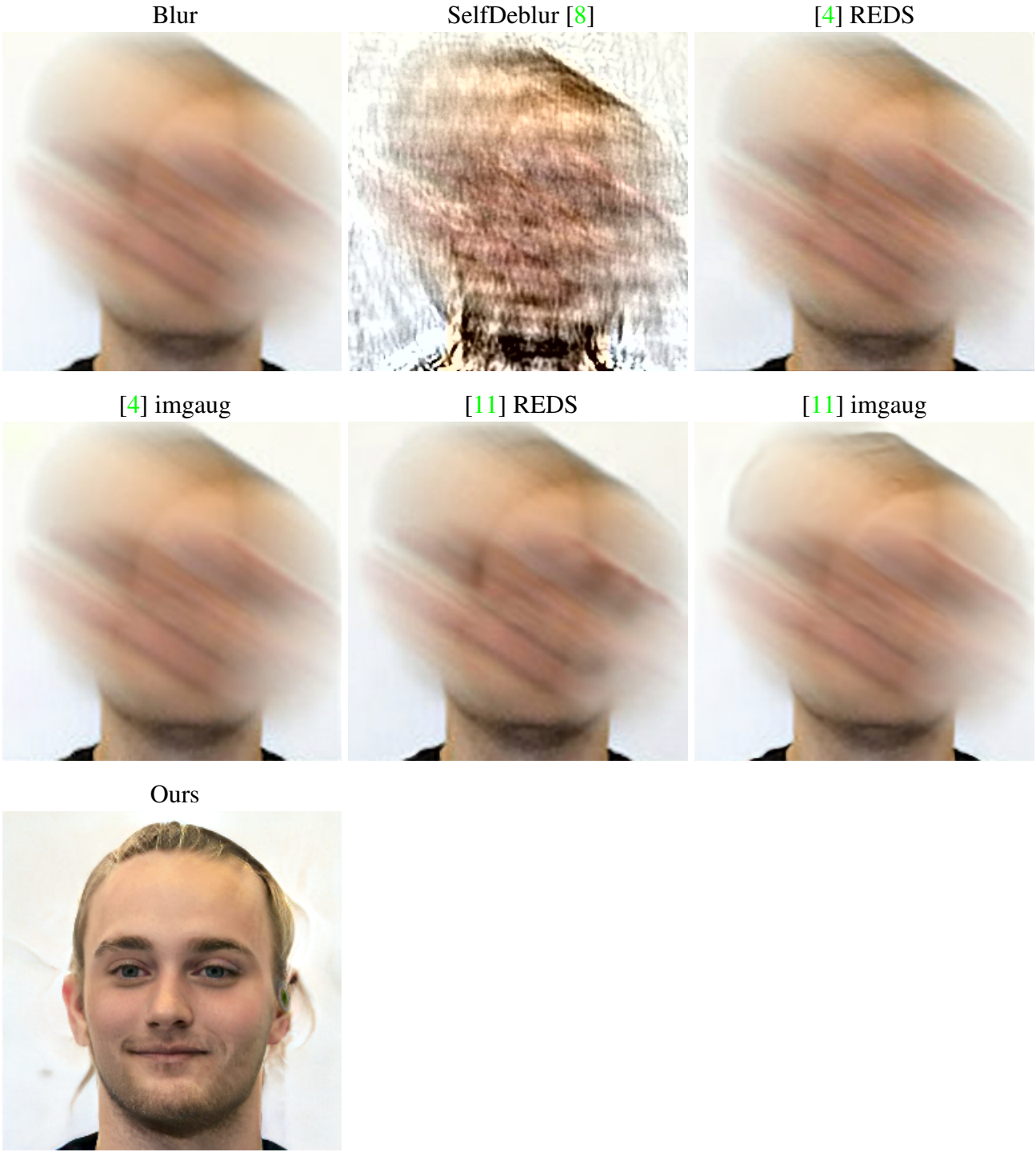


Figure 9. Results of deblurring methods trained on REDS and tested on an in-the-wild example


Figure 10. Results of our method trained on REDS and tested on GOPRO


Figure 11. Results of our method trained on REDS and tested on GOPRO

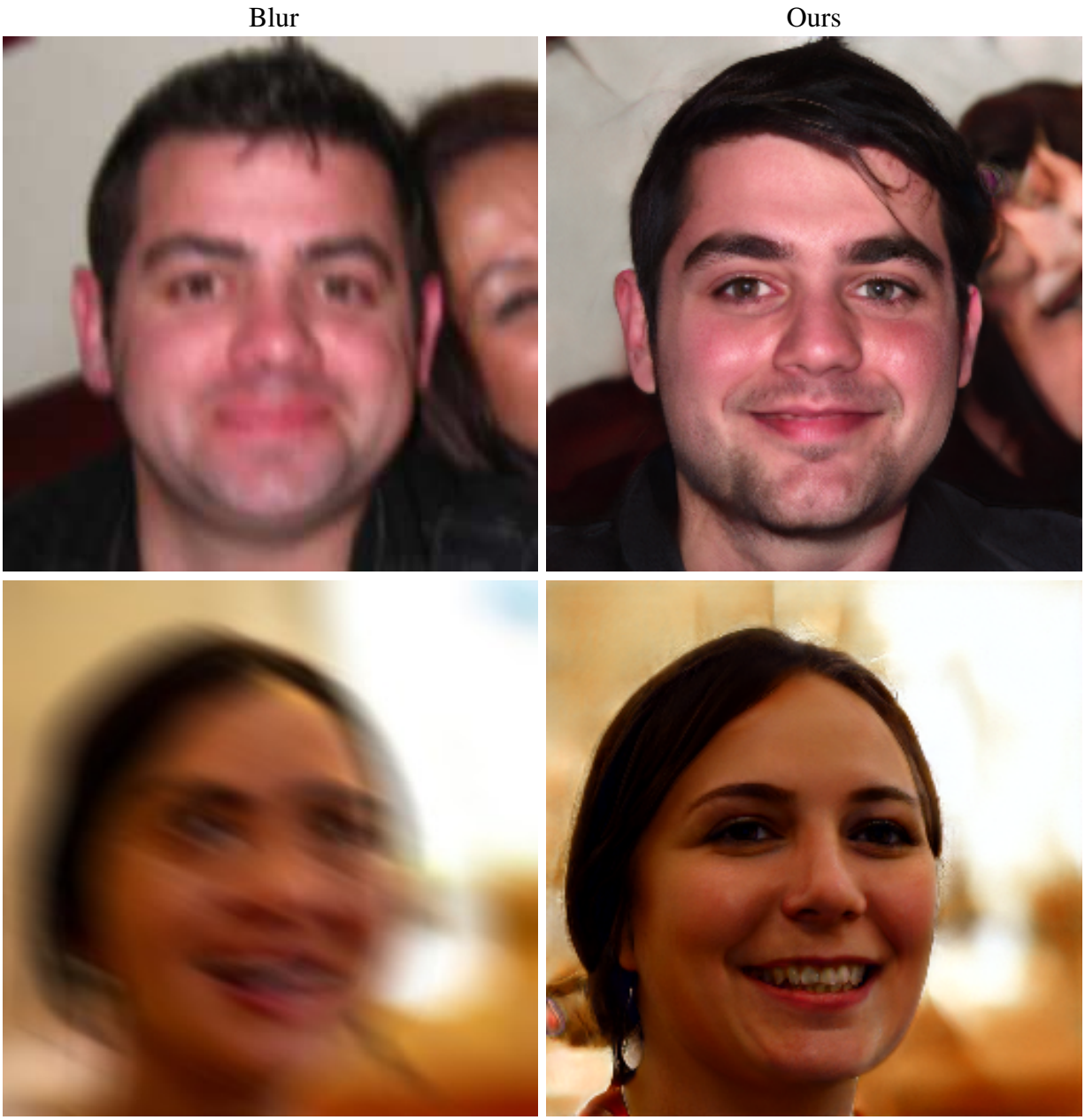


Figure 12. Results of our method trained on REDS and tested on GOPRO