Multi-Curve Translator for High-Resolution Photorealistic Image Translation—Appendix

A Implementation Details

A.1 I2I Translation

We use LSGAN [17] and PatchGAN [7] to train CycleGAN [25], UNIT [15], and their MCT variants. The objective function of LSGAN is:

$$\begin{aligned}\min_D \mathcal{L}_{LSGAN}(D) = &\frac{1}{2}\mathbb{E}_{\boldsymbol{y}\sim p(\boldsymbol{y})}\left[(D(\boldsymbol{y})-b)^2\right] \\ &+\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[(D(G(\boldsymbol{x}))-a)^2\right] \\ \min_G \mathcal{L}_{LSGAN}(G) = &\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})}\left[(D(G(\boldsymbol{x}))-c)^2\right],\end{aligned} \tag{1}$$

where $a = c = 1$ and $b = 0$. Both CycleGAN and UNIT contain two discriminators ($D_{\mathcal{X}}$ and $D_{\mathcal{Y}}$) and two generators ($G_{\mathcal{X}\to\mathcal{Y}}$ and $G_{\mathcal{Y}\to\mathcal{X}}$). For UNIT, a generator can be further divided into an encoder E and a generator G, then $G_{\mathcal{X}\to\mathcal{Y}} = G_{\mathcal{Y}}(E_{\mathcal{X}}(x))$ and $G_{\mathcal{Y}\to\mathcal{X}} = G_{\mathcal{X}}(E_{\mathcal{Y}}(y))$. Note that the process of sampling latent codes is omitted here. For simplicity, we denote the adversarial loss when training the generator G as $\mathcal{L}_{gan}$.

CycleGAN and UNIT both contain the cycle-consistency loss [25,9], which is formulated as:

$$\begin{aligned}\mathcal{L}_{cyc} = &\mathbb{E}_{x\sim p(x)}\left[\|G_{\mathcal{Y}\to\mathcal{X}}(G_{\mathcal{X}\to\mathcal{Y}}(x)) - x\|_1\right] \\ &+\mathbb{E}_{y\sim p(y)}\left[\|G_{\mathcal{X}\to\mathcal{Y}}(G_{\mathcal{Y}\to\mathcal{X}}(y)) - y\|_1\right].\end{aligned} \tag{2}$$

We also employ the identity loss [22]. For CycleGAN, it is formulated as:

$$\begin{aligned}\mathcal{L}_{idt} = &\mathbb{E}_{y\sim p(y)}\left[\|G_{\mathcal{X}\to\mathcal{Y}}(y) - y\|_1\right] \\ &+\mathbb{E}_{x\sim p(x)}\left[\|G_{\mathcal{Y}\to\mathcal{X}}(x) - x\|_1\right]\end{aligned} \tag{3}$$

UNIT's identity loss is the reconstruction loss that is formulated as:

$$\begin{aligned}\mathcal{L}_{idt} = &\mathbb{E}_{y\sim p(y)}\left[\|G_{\mathcal{Y}}(E_{\mathcal{Y}}(y)) - y\|_1\right] \\ &+\mathbb{E}_{x\sim p(x)}\left[\|G_{\mathcal{X}}(E_{\mathcal{X}}(x)) - x\|_1\right]\end{aligned} \tag{4}$$

Besides, UNIT also adds a KL divergence loss to penalize deviation of the distribution of the latent code $z_{\mathcal{X}} = E_{\mathcal{X}}(x)$ and $z_{\mathcal{Y}} = E_{\mathcal{Y}}(y)$ from the prior distribution, denoted as:

$$\begin{aligned}\mathcal{L}_{KL} = &\mathrm{KL}\left(q_{\mathcal{X}}\left(z_{\mathcal{X}}|x\right)\|p_\eta(z)\right) \\ &+\mathrm{KL}\left(q_{\mathcal{Y}}\left(z_{\mathcal{Y}}|y\right)\|p_\eta(z)\right)\end{aligned} \tag{5}$$

where the prior distribution $p_\eta(z)$ is a zero-mean Gaussian $p_\eta(z) = \mathcal{N}(z|0, I)$. Note the KL terms also penalize the latent codes deviating from the prior distribution in the cycle-reconstruction stream. When training the MCT variants, we further add $\mathcal{L}_{\text{reg}}$ stated in the main paper:

$$\mathcal{L}_{reg} = \left\| G_b^{y \to x}(G_m^{x \to y}(x)) - x \right\|_1 . \tag{6}$$

The CycleGAN's full objective is:

$$\mathcal{L} = \mathcal{L}_{gan} + \lambda_1 \mathcal{L}_{idt} + \lambda_2 \mathcal{L}_{cyc}. \tag{7}$$

where $\lambda_1 = 5$ and $\lambda_2 = 10$. Then the MCT-CycleGAN's full objective is:

$$\mathcal{L} = \mathcal{L}_{gan} + \lambda_1 \mathcal{L}_{idt} + \lambda_2 \mathcal{L}_{cyc} + \lambda_3 \mathcal{L}_{reg}. \tag{8}$$

where $\lambda_1 = 5$, $\lambda_2 = 10$, and $\lambda_3 = 1$.

The UNIT's full objective is:

$$\mathcal{L} = \lambda_0 \mathcal{L}_{gan} + \lambda_1 \mathcal{L}_{idt} + \lambda_2 \mathcal{L}_{cyc} + \lambda_3 \mathcal{L}_{KL}. \tag{9}$$

where $\lambda_0 = 10$, $\lambda_1 = \lambda_2 = 100$, and $\lambda_3 = 0.1$. And MCT-UNIT's full objective is:

$$\mathcal{L} = \lambda_0 \mathcal{L}_{gan} + \lambda_1 \mathcal{L}_{idt} + \lambda_2 \mathcal{L}_{cyc} + \lambda_3 \mathcal{L}_{KL} + \lambda_4 \mathcal{L}_{reg}. \tag{10}$$

where $\lambda_0 = 10$, $\lambda_1 = \lambda_2 = 100$, $\lambda_3 = 0.1$ and $\lambda_4 = 10$.

We use the Adam optimizer [10] to train the models with a mini-batch size of 1. The base models were trained from scratch with a learning rate of 2×10^{-4}. We keep the same learning rate for the first 100 epochs and linearly decay the rate to zero over the next 100 epochs. The MCT variants load the base models' weights but also use an initial learning rate of 2×10^{-4}. We finetune them with half of the epochs, *i.e.*, keep the same learning rate for the first 50 epochs and linearly decay the rate to zero over the next 50 epochs.

A.2 Style Transfer

AdaIN [5] encodes the style image and the content image using a pre-trained VGG-19 [20] up to `relu4_1`. Let VGG-19 be E, the style image be s, and the content image be c. Then the transformed feature maps are:

$$t = \text{AdaIN}(E(c), E(s)). \tag{11}$$

The goal of AdaIN is to train a decoder G to reconstruct t into a stylized image. AdaIN employs content loss as:

$$\mathcal{L}_c = \|E(G(t)) - t\|_2 \tag{12}$$

Let the mean of the feature maps be μ and the variance be σ, AdaIN's style loss is:

$$\begin{aligned} \mathcal{L}_s = &\sum_{i=1}^{L} \left\| \mu\left(\phi_i(G(t))\right) - \mu\left(\phi_i(s)\right) \right\|_2 \\ &+ \sum_{i=1}^{L} \left\| \sigma\left(\phi_i(G(t))\right) - \sigma\left(\phi_i(s)\right) \right\|_2 \end{aligned} \tag{13}$$

where each ϕ_i denotes a layer in VGG-19 used to compute the style loss (`relu1_1`, `relu2_1`, `relu3_1`, `relu4_1`). Besides, we add a gradient loss $\mathcal{L}_g$ to prompt the preservation of the geometric structure as:

$$\mathcal{L}_g = \|\nabla_h G(c) - \nabla_h c\|_2^2 + \|\nabla_v G(c) - \nabla_v c\|_2^2, \tag{14}$$

where ∇_h (∇_v) denotes the gradient operator along the horizontal (vertical) direction.

We employs a weighted combination of the content loss $\mathcal{L}_c$, the style loss $\mathcal{L}_s$, and the gradient loss $\mathcal{L}_g$ to train AdaIN and MCT-AdaIN:

$$\mathcal{L} = \mathcal{L}_c + \lambda_1 \mathcal{L}_s + \lambda_2 \mathcal{L}_g, \tag{15}$$

where $\lambda_1 = 1$ and $\lambda_2 = 100$.

The key to WCT^2 [23] is the network architecture and the procedure of performing WCT [13], while its training scheme is simple. WCT^2 also uses the pre-trained VGG-19 as the encoder and aims to train a decoder to reconstruct the encoder's feature map into the input image. WCT^2 uses the $\mathcal{L}_2$ reconstruction loss and the additional feature Gram matching loss with the encoder to train the decoder. For simplicity, we only keep the reconstruction loss because the Gram matching loss is not necessary. The $\mathcal{L}_2$ reconstruction loss is formulated as:

$$\mathcal{L} = \|G(E(c)) - c\|_2. \tag{16}$$

Then its MCT variant's loss function is:

$$\mathcal{L} = \|G_m(E(c)) - c\|_2 + \|G_b(E(c)) - c\|_2. \tag{17}$$

The challenge of extending WCT^2 to its MCT variant is that it is extremely easy to fall into a collapse solution, even if we constrain the base output. Specifically, the curves represented by the parameter maps obtained by summing the two outputs of the MCT variant are always identity mapping, even if we perform WCT on the intermediate feature maps. For this reason, we need to further prevent the slicing operation from leaking low-frequency information from the input to the output. First, we perform the grayscale operation on the HR image. Then the HR image contains only one channel, so the corresponding number of curves is changed from 9 to 3 ($C = 3M$). When performing stylization, we further match the HR image's brightness with the style image's brightness to prevent the brightness of the HR image from being retained in the output image, which can be expressed as $\tilde{c} = c - \mu(c) + \mu(s)$. Note that we do not grayscale and brightness-align the LR images.

We use the Adam optimizer to train the models with a mini-batch size of 8. The base models were trained from scratch with a learning rate of 1×10^{-4} for 1×10^5 iterations. Their MCT variants load the base models' weights, and are trained with a learning rate of 1×10^{-4} for 5×10^4 iterations.

A.3 Image Dehazing

We use only the $\mathcal{L}_1$ loss function to train GCANet [1], MSBDN [4], and their MCT variants:

$$\mathcal{L} = \|G(x) - y\|_1. \tag{18}$$

All models are trained from scratch using the Adam optimizer with the cosine annealing strategy [16] but not warm restarts. For all models, we set the initial learning rate to 1×10^{-4}. For GCANet and MCT-GCANet, we set mini-batch size to 56 and train them for 500 epochs. For MSBDN and MCT-MSBDN, we set mini-batch size to 28 and train them for 300 epochs.

A.4 Photo Retouching

Since color mapping and receptive field are critical for photo retouching, we set $H_d = W_d = 32$, $M = 32$ for MCT-DPED and set $M = 16$ and $p = 8$ for MCT-DPE.

For paired training, we also use the $\mathcal{L}_1$ loss function to train DPED [6] and DPE [3]. All models are trained from scratch using the Adam optimizer with the cosine annealing strategy but not warm restarts. And they are trained from scratch with a learning rate of 1×10^{-4} for 100 epochs. Since the memory consumptions of these models are different, we train them using different mini-batch sizes. For DPED, we set the mini-batch size to 16. For DPED-MCT and DPE, we set the mini-batch size to 64. For DPE-MCT, we set the mini-batch size to 32.

For unpaired training, we employ the earlier described CycleGAN's training scheme. For DPED, all training hyperparameters are not modified. For DPE, we removed the identity loss because we found that it would cause the DPE to fall into a poor solution.

B Limitations & Future Works

MCT is a flexible framework that minimizes the effort of modifying the base model into an MCT variant. For a new I2I translation task, we only need to modify the output layer of the existing model to extend it to its MCT variant. Besides, we can not only load the pre-training weights of the base model but also use the training strategy of the base model directly without modifying any hyperparameters. Unfortunately, there are limitations to the tasks for which MCT is applicable.

The first is the tasks to which MCT can be applied. We assume that if the I2I translation process only slightly changes the shape and texture of the objects in the image (*i.e.*, high-frequency information), then a mapping responsible for local regions in the image domain can be learned to approximate this translation process. For tasks that do not satisfy our assumptions, such as `dog2cat`, MCT is helpless. The reason is that MCT is realized by interpolation to apply the

LR parameter maps to transform HR images, so the transformation of high-frequency information is spatially smooth. However, for some I2I translation tasks with large shape changes, the transformation of high-frequency information is highly unsmoothed spatially. We have tried to introduce spatial support similar to KPN [18], but no visible improvement has been achieved. It may mean that improving the translator's capability to translate high-frequency information without significantly increasing the number of parameters and the computational cost is a non-trivial task. And it is the focus of our future research.

In addition to this, MCT is not always plug-and-play. For CycleGAN and UNIT, we need to constrain the base output to prevent the MCT variants from falling into collapse solutions. For WCT^2, we need to grayscale and brightness-align the HR images to reduce further the low-frequency information flowing from the input to the output through the slicing operation. It is necessary to develop more generalized training strategies to reduce further the difficulty of extending the base model to their MCT variants.

Finally, although the MCT variants are significantly faster compared to their base models, they may still be computationally heavy for edge devices due to the high computational cost of some base models (*e.g.* CycleGAN). For this, we may need to introduce the model compression approach or design lightweight network architectures to lower the computational cost of the backbone network. In addition, the runtime of the lookup table should not be ignored since the memory bandwidth of the edge device can limit the speed of slicing operations. We plan to find a better dimensionality reduction operation than image downsampling and revise the slicing operation to reduce computational and memory access costs.

C More Experimental Results

We included the table of FID in the main paper, and we expanded it to FID / KID $\times$ 100 in Table 1. This experiment provides a rough indication of each method's translation capability for HR images, and the results are for reference only since the FID and KID are not suitable for evaluating HR images.

We expand the user study to style transfer in Table 2. Although the results of AdaIN preserve almost no high-frequency information, most users still feel that the quality of MCT-AdaIN is inferior. AdaIN is an artistic style transfer method, and its fixed VGG encoder without skip connection severely loses details. This property conflicts with the property of MCT to retain high-frequency information, making MCT-AdaIN's results unattractive.

We also explore not reintroducing high-frequency information, but using super-resolution to predict high-frequency information. We conducted experiments on image retouching (see Table 3). This method does not perform well, especially in terms of SSIM, and we found that little lost high-frequency information can be restored in the output.

We then quantitatively compare the translation capabilities of the state-of-the-art lightweight I2I translation network and the MCT variants using super-

vised learning-based tasks. Table 4 shows the quantitative comparison between LPTN [14] and MCT-DPE on the photo retouching. It can be seen that LPTN is significantly inferior to MCT-DPE in both speed and performance. It is worth mentioning that LPTN only achieves 23.09 dB on the SOTS dataset, 2.62 dB lower than MCT-GCANet, and 5.61 dB lower than MCT-MSBDN for image dehazing.

Considering that MCT is a curve-based method, we further compare MCT with other curve-based methods. Previous curve-based methods were employed only on image retouching. Since GleNet [8] was tested on downsampled images and provided an empty repository, we re-implemented GleNet's GEN (LEN is not real-time). We train CURL [19] in the unpaired setting because no previous works reported the result. We did not train StarEnhancer [21] in the unpaired setting because it is a multi-style method that is non-trivial to extend. Table 5 shows the comparison. The global transformation introduces inductive biases practical for image retouching, making the previous curve-based methods prevent overfitting and run fast. In contrast, DPE as a base model does not effectively aggregate global information.

Table 6 illustates more runtime comparison results. Figures 2-11 show more qualitative comparison results. Readers can generate more test results using the provided code and pre-trained models.

We finally visualize the effectiveness of the two training strategies. Fig. 1 shows a special case when training MCT-CycleGAN on `day2dusk`. If we train MCT-CycleGAN without constraining the base output, it may fall into a poor solution. In contrast, imposing constraints on the base output makes the backbone network responsible for low-frequency information and medium-frequency information, leaving only the high-frequency information lost during downsampling to be taken care of by the slicing operation. When we do not use the pixel unaligned training strategy, the output image of MCT may lose high-frequency information. Unlike increasing the weight of cycle-consistency loss, a pixel unaligned training strategy causes the slicing operation to focus more on high-frequency information. Note that although the output of MCT is blurred at this point, it still contains more high-frequency information than the upsampled base output due to the curve slicing operation.

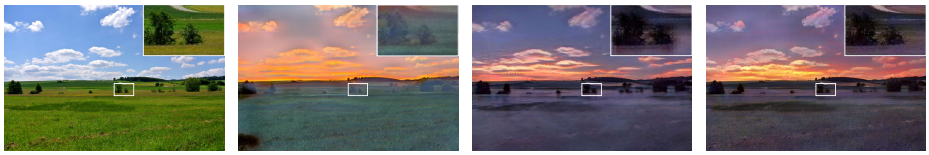

Fig. 1. Ablation study on `day2dusk`. The second image is the result without constraining the base output during training. The third image is the result without the pixel unalignment training strategy. The last image is the result using our proposed full training scheme.

Table 1. Quantitative comparison (FID / KID × 100) of the photorealistic I2I translation. Lower is better.

	`day2dusk`	`dusk2day`	`summer2autumn`	`autumn2summer`
CycleGAN	89.00 / 1.14	94.17 / 1.69	**101.98** / 1.45	100.34 / 1.65
UNIT	92.14 / 1.38	96.66 / **1.58**	105.15 / 1.54	95.18 / 1.50
MCT-CycleGAN	**81.67** / **0.67**	**92.14** / 1.75	103.45 / 1.56	94.72 / 1.55
MCT-UNIT	84.22 / 1.01	93.14 / 1.72	103.43 / **1.44**	**91.35** / **1.48**

Table 2. User study results. The percentage indicates the preferred model outputs out of 95 responses. Note that `d2d` means `day2dusk`, `s2a` means `summer2autumn`, and `M` means Mask.

Methods	CycleGAN		UNIT		AdaIN		WCT^2	
Task	`d2d`	`s2a`	`d2d`	`s2a`	w/ `M`	w/o `M`	w/ `M`	w/o `M`
Base	32.6%	47.4%	29.5%	42.1%	**84.2%**	**89.5%**	**54.7%**	42.1%
MCT	**67.4%**	**52.6%**	**70.5%**	**57.9%**	15.8%	10.5%	45.3%	**57.9%**

Table 3. Quantitative comparison of photo retouching in the unpaired setting. The FPS is tested using a single A100. TU means Translation-Upsampling (use EDSR-LIIF [2] for scale-arbitrary).

Methods	480p	1080p	original
	PSNR / SSIM / FPS	PSNR / SSIM / FPS	PSNR / SSIM / FPS
DPE	21.07 / 0.861 / **284.7**	21.02 / 0.859 / 43.7	20.92 / 0.854 / 10.8
TU-DPE	19.47 / 0.730 / 13.7	18.83 / 0.673 / 2.2	18.53 / 0.654 / 0.6
MCT-DPE	**23.40** / **0.903** / 271.6	**23.31** / **0.903** / **269.9**	**23.09** / **0.905** / **153.8**

Table 4. Quantitative comparison on the FiveK dataset in the unpaired setting. The FPS is tested on A100 with batch size = 1.

Methods	480p			1080p			original		
	PSNR	SSIM	FPS	PSNR	SSIM	FPS	PSNR	SSIM	FPS
LPTN ($L = 3$)	22.12	0.878	248.9	22.09	0.883	188.4	22.02	0.879	37.9
MCT-DPE	**23.40**	**0.903**	**271.6**	**23.31**	**0.903**	**269.9**	**23.09**	**0.905**	**153.8**

Table 5. Quantitative comparison of photo retouching. FPS is measured on 4K images using a single A100. Note that some results are replicated from [12,21].

Methods	Paired		Unpaired		FPS
	PSNR	SSIM	PSNR	SSIM	
FlexiCurve [12]	23.97	0.910	22.12	0.860	83.3
CURL [19]	24.20	0.880	21.62	0.873	3.4
GEN [8]	24.91	0.937	22.73	0.902	**364.3**
StarEnhancer [21]	**25.29**	**0.943**	-	-	242.1
MCT-DPE	25.10	0.941	**23.09**	**0.905**	153.8

Table 6. Runtime comparison of the base models and their MCT variants.

Method	Hardware	Resolution								
		256×256	360×360	512×512	1280×720	1920×1080	2560×1440	3840×2160	6000×4000	7680×4320
CycleGAN	A100-40G	238.4	146.1	80.7	23.8	10.8	6.1	2.7	OOM	OOM
	RTX 3090	138.6	72.7	37.1	11.7	5.2	2.9	1.3	OOM	OOM
	RTX 3080	130.7	65.6	32.8	7.1	4.3	2.4	1.1	OOM	OOM
	RTX 3070	77.3	41.2	21.4	6.0	2.9	1.7	OOM	OOM	OOM
	RTX 3060	52.6	29.4	15.0	4.4	2.0	1.1	0.5	OOM	OOM
	RTX 2080Ti	111.4	52.1	27.6	7.4	3.3	1.9	0.9	OOM	OOM
	GTX 1080Ti	58.1	29.6	15.7	4.1	1.9	1.0	0.5	OOM	OOM
	GTX 1070Ti	17.5	16.7	8.4	2.3	1.0	0.6	OOM	OOM	OOM
WCT^2	A100-40G	14.2	13.3	11.1	4.7	2.1	1.2	OOM	OOM	OOM
	RTX 3090	13.7	13.1	8.7	3.4	1.7	1.0	OOM	OOM	OOM
	RTX 3080	12.9	10.2	7.2	2.9	OOM	OOM	OOM	OOM	OOM
	RTX 3070	11.3	9.2	5.9	OOM	OOM	OOM	OOM	OOM	OOM
	RTX 3060	10.6	7.2	4.4	1.4	OOM	OOM	OOM	OOM	OOM
	RTX 2080Ti	12.4	10.7	7.0	2.5	OOM	OOM	OOM	OOM	OOM
	GTX 1080Ti	8.9	6.8	4.3	1.2	OOM	OOM	OOM	OOM	OOM
	GTX 1070Ti	7.6	4.7	2.8	0.7	OOM	OOM	OOM	OOM	OOM
GCANet	A100-40G	235.1	199.0	95.6	28.2	12.2	6.9	3.1	1.0	OOM
	RTX 3090	229.4	138.7	75.7	21.9	9.8	5.5	2.4	OOM	OOM
	RTX 3080	227.2	121.4	64.7	18.5	8.3	4.6	OOM	OOM	OOM
	RTX 3070	168.6	81.1	43.3	12.2	5.5	3.1	OOM	OOM	OOM
	RTX 3060	117.4	58.2	30.8	8.7	3.9	2.2	OOM	OOM	OOM
	RTX 2080Ti	205.6	78.5	52.0	14.0	6.1	3.5	OOM	OOM	OOM
	GTX 1080Ti	90.5	41.3	20.5	5.3	2.3	1.3	OOM	OOM	OOM
	GTX 1070Ti	60.6	28.6	14.2	3.8	1.7	0.9	OOM	OOM	OOM
DPE	A100-40G	410.6	403.5	317.9	94.8	43.7	24.4	10.8	3.6	2.6
	RTX 3090	400.3	390.6	279.1	84.2	39.1	21.9	9.9	OOM	OOM
	RTX 3080	396.4	357.3	252.2	74.5	34.5	19.1	OOM	OOM	OOM
	RTX 3070	382.0	315.4	175.8	51.2	23.5	13.1	OOM	OOM	OOM
	RTX 3060	372.1	231.3	127.6	37.2	17.0	9.5	4.2	OOM	OOM
	RTX 2080Ti	389.3	345.7	195.8	60.0	26.9	15.3	6.7	OOM	OOM
	GTX 1080Ti	378.8	238.8	137.5	37.5	16.7	9.3	4.1	OOM	OOM
	GTX 1070Ti	273.3	153.7	84.2	22.2	10.0	5.5	OOM	OOM	OOM
MCT-CycleGAN	A100-40G	173.5	172.3	171.5	169.6	168.9	153.1	116.0	64.2	51.1
	RTX 3090	116.1	115.9	114.6	111.1	103.5	95.1	77.7	47.6	39.3
	RTX 3080	108.9	108.1	106.4	100.7	93.1	84.7	68.1	40.9	33.2
	RTX 3070	64.7	64.6	63.8	61.0	57.3	52.6	42.8	26.1	OOM
	RTX 3060	44.5	44.3	43.8	42.1	39.5	36.3	29.9	18.6	15.2
	RTX 2080Ti	93.1	91.9	89.9	83.3	74.7	65.7	49.8	27.9	22.2
	GTX 1080Ti	50.0	49.6	48.8	45.9	41.5	36.9	28.2	15.9	12.6
	GTX 1070Ti	29.9	29.6	29.2	27.8	25.7	23.5	18.6	11.0	OOM
MCT-WCT^2	A100-40G	14.2	14.1	14.1	14.0	13.9	13.6	13.5	12.6	12.1
	RTX 3090	13.4	13.3	13.2	12.9	12.9	12.7	12.5	12.3	11.6
	RTX 3080	13.2	12.6	12.5	12.5	12.3	12.3	11.6	10.5	9.9
	RTX 3070	11.9	11.9	11.8	11.5	11.5	11.0	10.9	9.4	OOM
	RTX 3060	10.4	10.4	10.3	10.2	10.1	9.9	9.4	8.0	7.3
	RTX 2080Ti	14.4	14.4	14.4	14.2	14.1	13.7	12.9	10.6	9.8
	GTX 1080Ti	9.1	9.1	9.0	8.6	8.8	8.6	8.2	7.1	6.5
	GTX 1070Ti	7.5	7.4	7.4	7.4	7.2	7.1	6.6	5.4	OOM
MCT-GCANet	A100-40G	228.1	226.1	223.9	221.1	218.9	197.4	131.1	61.3	47.4
	RTX 3090	225.1	222.1	221.9	213.0	195.6	165.8	114.1	56.1	43.3
	RTX 3080	216.2	215.4	214.2	194.9	166.8	139.5	95.9	46.3	35.7
	RTX 3070	157.1	155.4	151.9	134.5	115.5	97.1	66.2	32.0	OOM
	RTX 3060	110.7	108.9	106.4	95.8	82.6	69.3	48.1	23.6	18.2
	RTX 2080Ti	194.0	190.2	186.5	158.7	131.1	106.0	70.0	32.7	24.8
	GTX 1080Ti	86.4	84.9	83.1	73.8	63.1	52.5	35.7	17.5	13.4
	GTX 1070Ti	57.2	56.2	55.2	49.7	43.2	36.7	25.8	12.6	OOM
MCT-DPE	A100-40G	307.1	301.3	300.0	297.3	280.9	252.1	162.4	72.9	55.2
	RTX 3090	289.8	288.9	288.7	288.3	263.5	224.4	142.4	63.7	48.0
	RTX 3080	274.3	273.4	272.9	265.4	251.1	198.6	123.3	53.8	40.5
	RTX 3070	251.5	251.4	247.7	226.6	178.3	138.4	83.3	35.6	OOM
	RTX 3060	208.5	204.9	196.0	165.4	130.0	99.5	60.4	25.9	19.3
	RTX 2080Ti	274.1	270.2	267.7	245.5	185.6	137.7	80.8	32.9	24.8
	GTX 1080Ti	216.8	211.5	198.9	152.6	110.2	80.3	45.7	18.5	14.0
	GTX 1070Ti	145.5	141.4	133.6	105.6	78.5	58.3	33.9	13.8	OOM

Fig. 2. Qualitative comparison of `day2dusk`.

Fig. 3. Qualitative comparison of `dusk2day`.

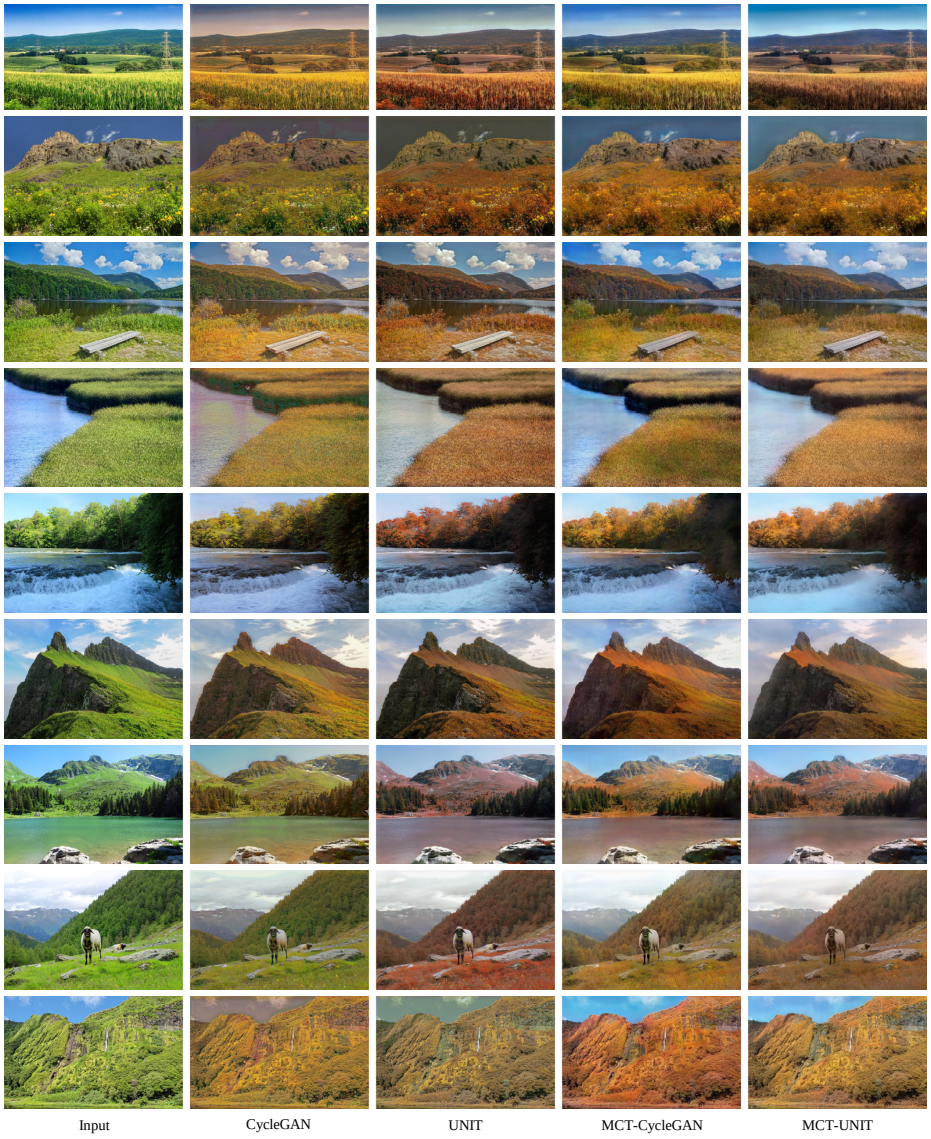

Fig. 4. Qualitative comparison of `summer2autumn`.

Fig. 5. Qualitative comparison of `autumn2summer`.

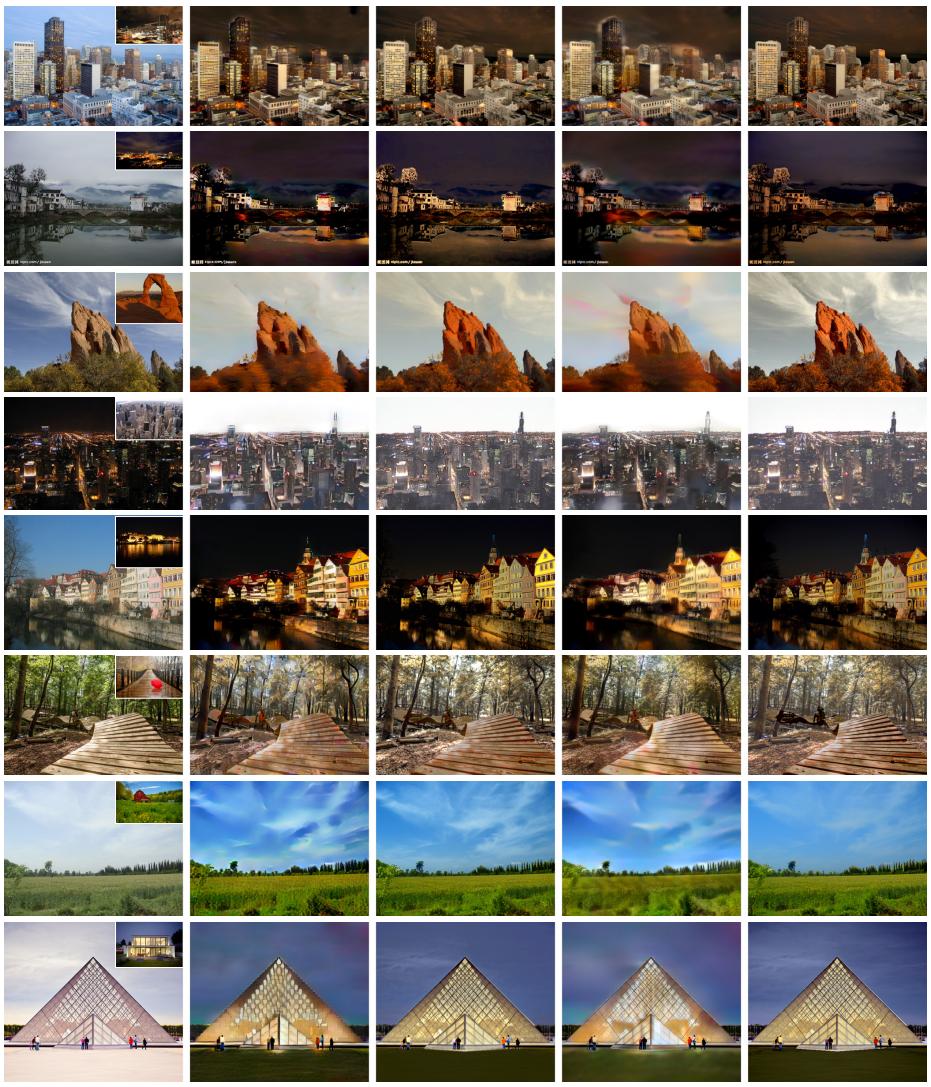

Fig. 6. Qualitative comparison of style transfer with paired segmentation label maps.

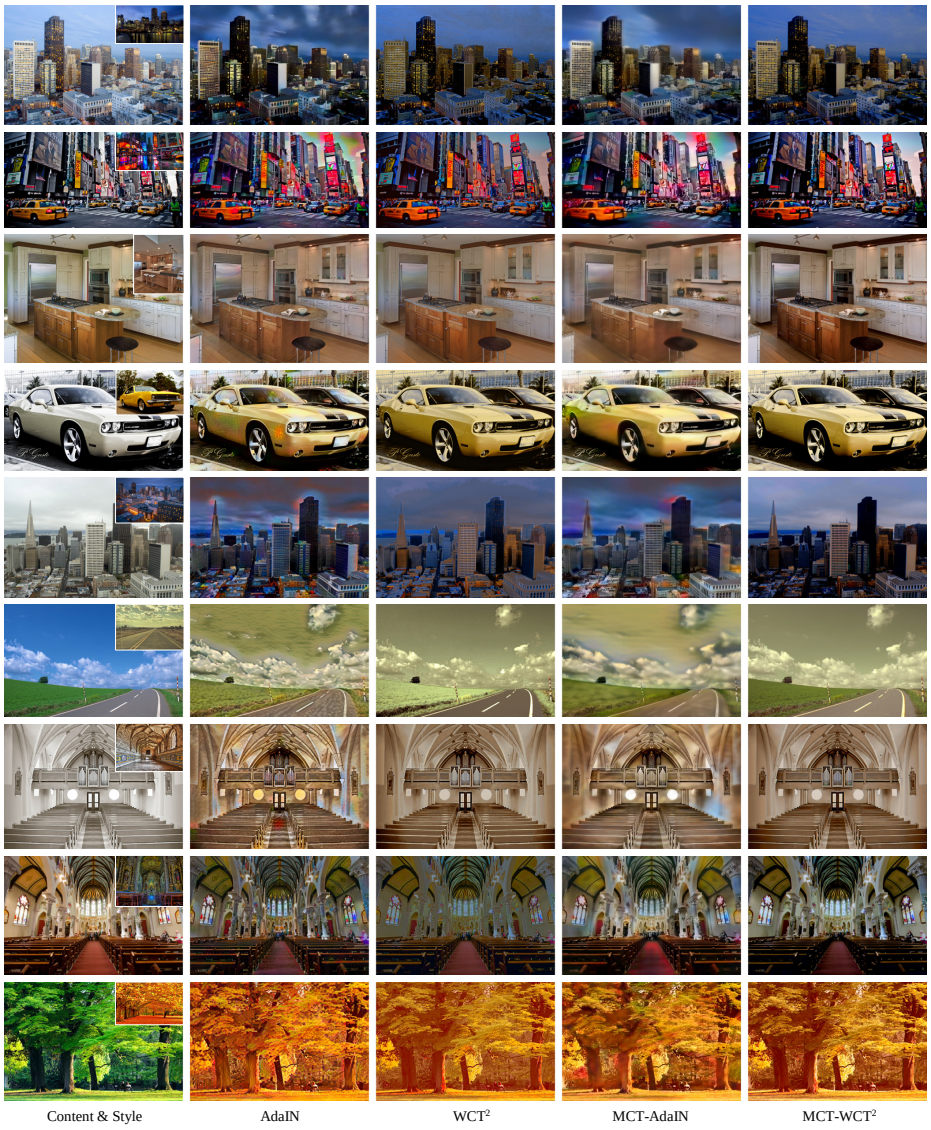

Fig. 7. Qualitative comparison of style transfer without segmentation label maps.

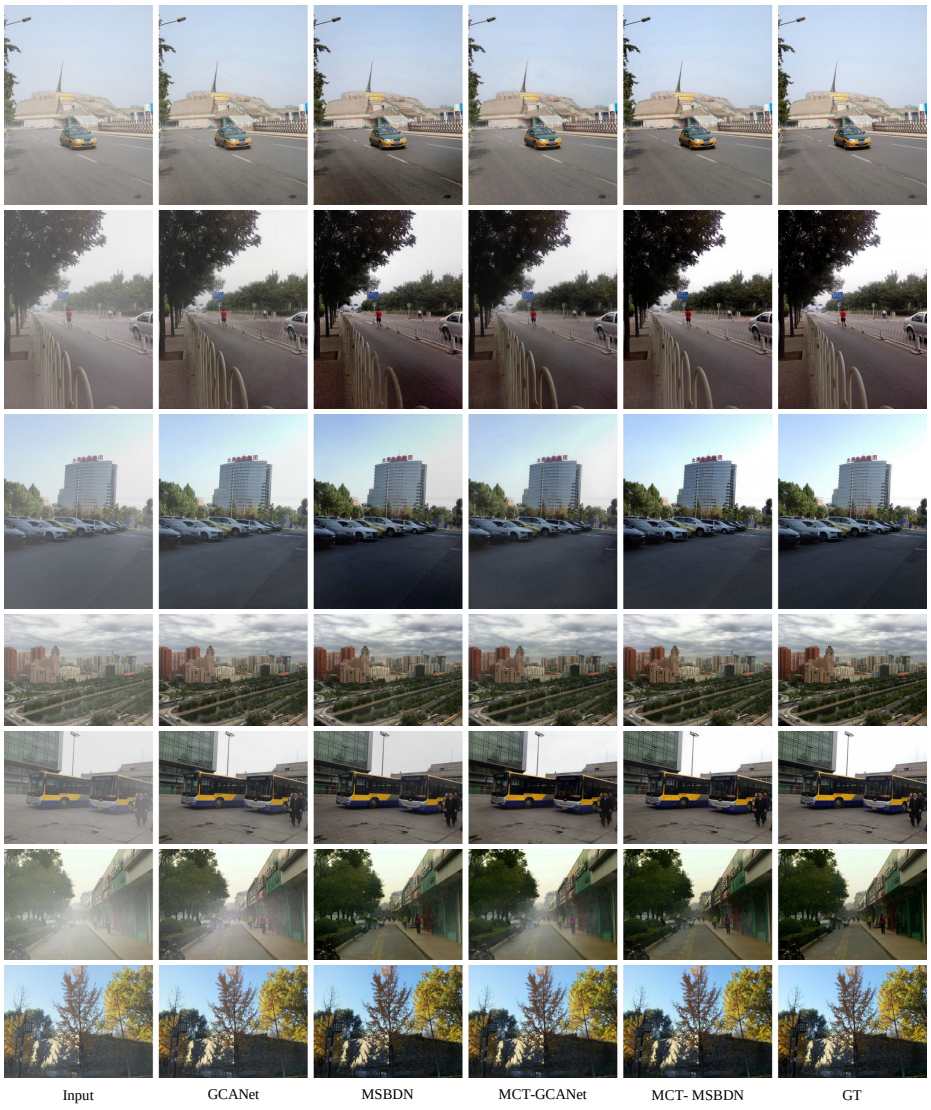

Fig. 8. Qualitative comparison of image dehazing on SOTS dataset [11].

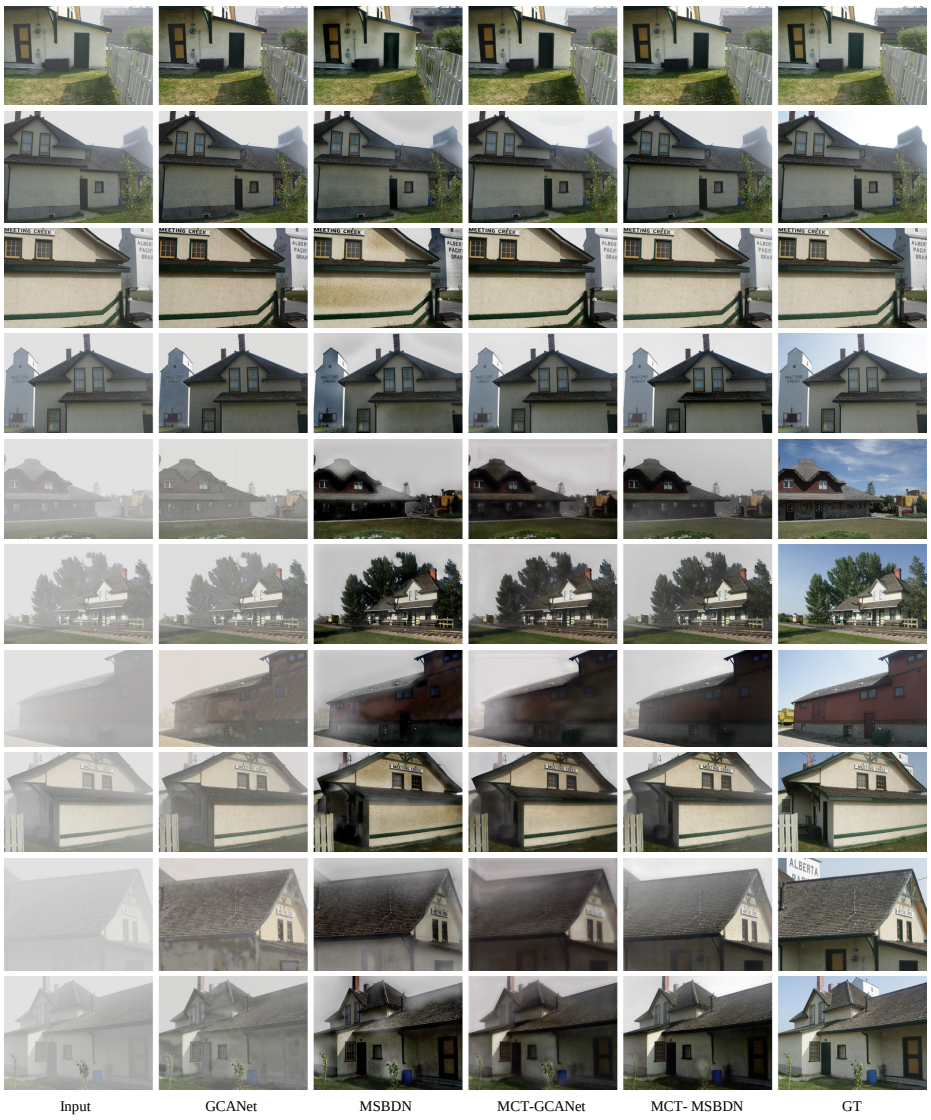

Fig. 9. Qualitative comparison of image dehazing on HazeRD dataset [24].

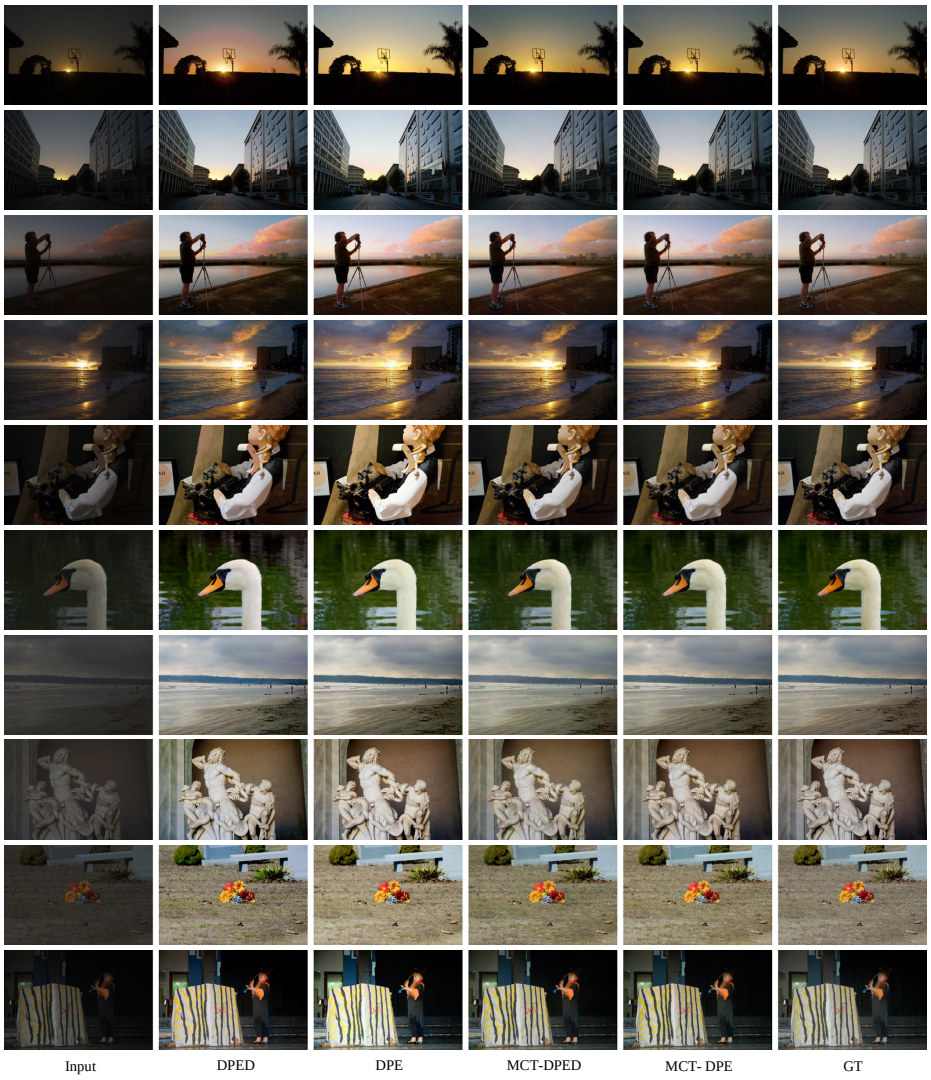

Fig. 10. Qualitative comparison of photo retouching with paired training.

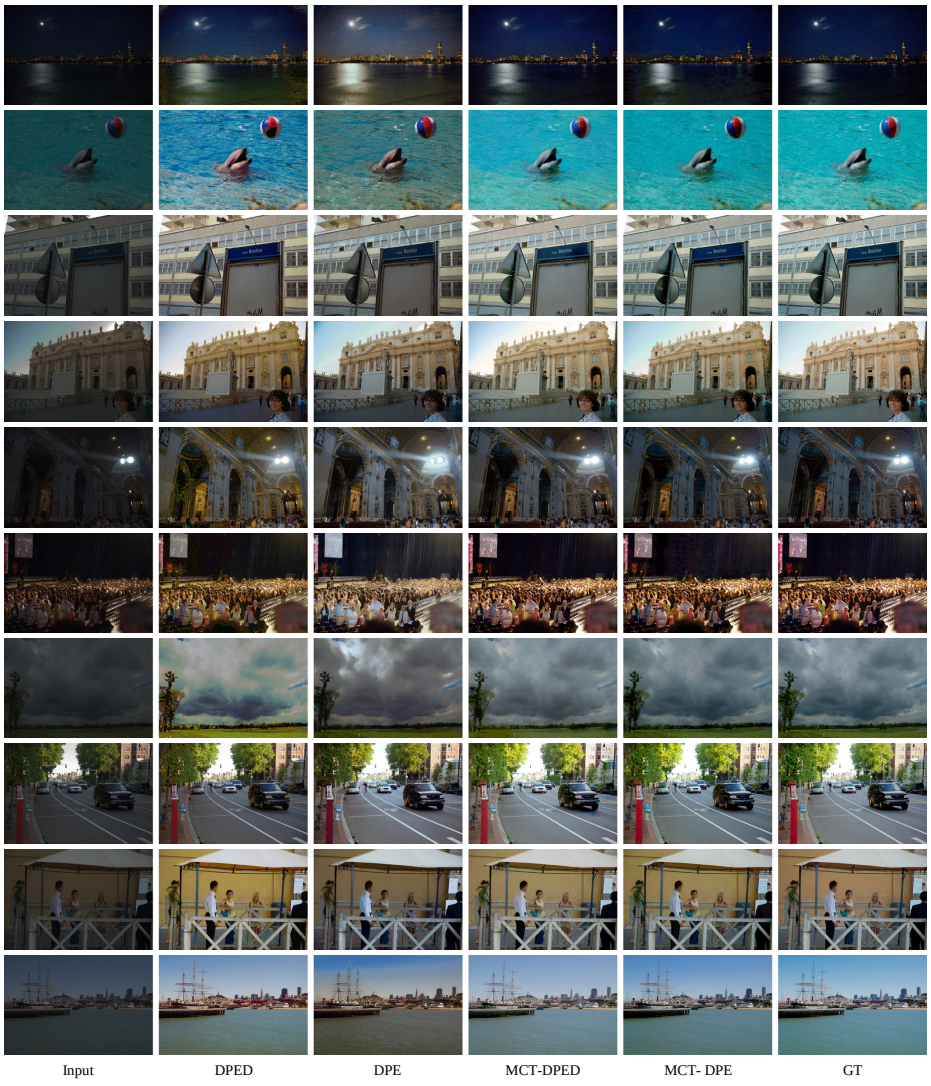

Fig. 11. Qualitative comparison of photo retouching with unpaired training.

References

1. Chen, D., He, M., Fan, Q., Liao, J., Zhang, L., Hou, D., Yuan, L., Hua, G.: Gated context aggregation network for image dehazing and deraining. In: WACV (2019) 4
2. Chen, Y., et al.: Chen, yinbo and liu, sifei and wang, xiaolong. CVPR (2021) 7
3. Chen, Y.S., Wang, Y.C., Kao, M.H., Chuang, Y.Y.: Deep photo enhancer: Unpaired learning for image enhancement from photographs with gans. In: CVPR (2018) 4
4. Dong, H., Pan, J., Xiang, L., Hu, Z., Zhang, X., Wang, F., Yang, M.H.: Multi-scale boosted dehazing network with dense feature fusion. In: CVPR (2020) 4
5. Huang, X., Belongie, S.: Arbitrary style transfer in real-time with adaptive instance normalization. In: ICCV (2017) 2
6. Ignatov, A., Kobyshev, N., Timofte, R., Vanhoey, K., Van Gool, L.: Dslr-quality photos on mobile devices with deep convolutional networks. In: ICCV (2017) 4
7. Isola, P., Zhu, J.Y., Zhou, T., Efros, A.A.: Image-to-image translation with conditional adversarial networks. In: CVPR (2017) 1
8. Kim, H.U., Koh, Y.J., Kim, C.S.: Global and local enhancement networks for paired and unpaired image enhancement. In: ECCV (2020) 6, 7
9. Kim, T., Cha, M., Kim, H., Lee, J.K., Kim, J.: Learning to discover cross-domain relations with generative adversarial networks. In: ICML (2017) 1
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. ICLR (2015) 2
11. Li, B., Ren, W., Fu, D., Tao, D., Feng, D., Zeng, W., Wang, Z.: Benchmarking single-image dehazing and beyond. IEEE TIP (2018) 15
12. Li, C., Guo, C., Ai, Q., Zhou, S., Loy, C.C.: Flexible piecewise curves estimation for photo enhancement. arXiv preprint arXiv:2010.13412 (2020) 7
13. Li, Y., Fang, C., Yang, J., Wang, Z., Lu, X., Yang, M.H.: Universal style transfer via feature transforms. In: NeurIPS (2017) 3
14. Liang, J., Zeng, H., Zhang, L.: High-resolution photorealistic image translation in real-time: A laplacian pyramid translation network. In: CVPR (2021) 6
15. Liu, M.Y., Breuel, T., Kautz, J.: Unsupervised image-to-image translation networks. In: NeurIPS (2017) 1
16. Loshchilov, I., Hutter, F.: Sgdr: Stochastic gradient descent with warm restarts. In: ICLR (2017) 4
17. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Paul Smolley, S.: Least squares generative adversarial networks. In: ICCV (2017) 1
18. Mildenhall, B., Barron, J.T., Chen, J., Sharlet, D., Ng, R., Carroll, R.: Burst denoising with kernel prediction networks. In: CVPR (2018) 5
19. Moran, S., McDonagh, S., Slabaugh, G.: Curl: Neural curve layers for global image enhancement. In: ICPR (2021) 6, 7
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015) 2
21. Song, Y., Qian, H., Du, X.: Starenhancer: Learning real-time and style-aware image enhancement. In: ICCV (2021) 6, 7
22. Taigman, Y., Polyak, A., Wolf, L.: Unsupervised cross-domain image generation. In: ICLR (2016) 1
23. Yoo, J., Uh, Y., Chun, S., Kang, B., Ha, J.W.: Photorealistic style transfer via wavelet transforms. In: ICCV (2019) 3
24. Zhang, Y., Ding, L., Sharma, G.: Hazerd: an outdoor scene dataset and benchmark for single image dehazing. In: ICIP (2017) 16
25. Zhu, J.Y., Park, T., Isola, P., Efros, A.A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: ICCV (2017) 1