Interactive Cartoonization with Controllable Perceptual Factors (Supplementary Material)

Namhyuk Ahn[1] Patrick Kwon[1] Jihye Back[1] Kibeom Hong[2] Seungkwon Kim[1]
[1] NAVER WEBTOON AI [2] Yonsei University

A. Method details

In this section, we explain detailed settings and experimental analyzes conducted in the main text.

Texture control analysis. Here, we describe detailed setups of the texture control analysis (Section 3.1, main text). In Figure 4a (main text), each stroke thickness case was generated by the separately trained CARTOONER. We used three models and these are trained with the cartoon image resolutions of $\{256^2, 416^2, 800^2\}$. We also set the texture controller to only have a single branch. With these setups, the models trained with $\{256^2, 416^2, 800^2\}$ resolutions generated thin, moderately thick, and thick strokes respectively.

To conduct an abstraction change experiment, as shown in Figure 4b (main text), we trained multiple CARTOONER similar to the stroke change scenario except for the receptive field (RF) of the generator. We differentiated the RF of the network by changing the kernel size of conv layers in the texture controller by $\{3, 11, 19\}$ each, which corresponds to the low, moderate, and high abstraction scenes.

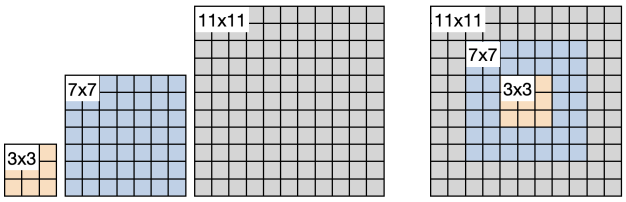
Network architecture. Table 1 presents the network architecture of CARTOONER. We applied LeakyReLU for all conv layers and did not use any normalization layer. In our experiments, we observed that the existence of a normalization layer (batchnorm [9] and instance norm [17]) drops the cartoonization quality. The cardinality of conv layers in the ResNeXt blocks were set to 32. We used the bilinear interpolation method for *Upsample* layer. In *col2* and *col3* layers in the color decoder, the additional 3-channel of each first conv layer is for the color cue injection.

Abstraction control unit. We designed this unit to be a shared multi-branch system to increase the quality robustness and reduce the model size. Specifically, the multi-branch module was composed of conv layers with varying kernel size, $K_1 < ... < K_N$, where K denotes kernel size and N is the number of branches. Instead of using N conv kernels, we only initialized a single conv layer with K_N kernel size. All other conv kernels were set to be a subset of K_N kernel as illustrated in Figure 1. By doing so, the abstraction control unit can produce robust outcomes for the different abstraction levels. We will demonstrate how crucial this design is in Section C. Such a design also successfully reduces the model parameters; without a shared scheme, the model parameters of CARTOONER becomes 26.5M, while the shared kernel version (ours) is 5.9M.

Table 1. **Network architecture.**

Layer	Output size	Building block
Shared Encoder (E_{shared})		
conv1	256×256	Conv(3, 32, 7, 1, 3)
conv2	128×128	Conv(32, 32, 3, 2, 1) Conv(32, 64, 3, 1, 1)
conv3	64×64	Conv(64, 64, 3, 2, 1) Conv(64, 128, 3, 1, 1)
conv4	64×64	3 × ResNeXt(128, 128)
Texture Decoder ($D_{texture}$)		
tex0	64×64	*texture controller*
tex1	128×128	Conv(128, 128, 3, 1, 1) Conv(128, 64, 3, 1, 1) Upsample(2)
tex2	256×256	Conv(64, 64, 3, 1, 1) Conv(64, 32, 3, 1, 1) Upsample(2)
tex3	256×256	Conv(32, 32, 3, 1, 1) Conv(32, 1, 7, 1, 3)
Color Decoder (D_{color})		
col0	64×64	ResNeXt(128, 128)
col1	128×128	Conv(128, 128, 3, 1, 1) Conv(128, 64, 3, 1, 1) Upsample(2)
col2	256×256	Conv(64+3, 64, 3, 1, 1) Conv(64, 32, 3, 1, 1) Upsample(2)
col3	256×256	Conv(32+3, 32, 3, 1, 1) Conv(32, 2, 7, 1, 3)

Model training. We trained CARTOONER using Adam [11] for 100K steps with a batch size of 32 and learning rate of 2×10^{-4}. Unlike previous deep methods [5, 6, 18], we



(a) Unshared conv (b) Shared conv

Figure 1. **Schematic overview of the shared conv kernel.** (**a**) Unshared (vanilla) conv kernel scheme. (**b**) Our shared conv kernel scheme, which was applied to constitute a multi-branch module in the abstraction control unit. To do that, we employed the conv with the largest kernel size only. All the other conv kernels were set to be a subset of the largest one.

did not perform network pre-training [6]. For all results shown in this paper, we used the same hyper-parameters: $\lambda^1_{texture} = 1.0$, $\lambda^2_{texture} = 0.0025$, $\lambda^3_{texture} = 0.0045$, $\lambda^4_{texture} = 0.0015$. We utilized the official Caffe-version of VGG19 [16], and when creating a Gram matrix, we divided it by the product of # of the channels, width, and height.

When generating an initial color map, C^{RGB}_{src} from an input photo I^{RGB}_{src}, we adopted *Zhu et al.* [21] for both training and inference. We selected this algorithm since it is GPU-friendly, nevertheless, any off-the-shelf super-pixel algorithm can be adopted. We tested other algorithms, such as SLIC [1], and observed no performance drop even when we use a different approach at train and inference.

B. Experimental settings

Dataset. In our study, we mainly focused on outdoor scenes and landscapes, to better target the domain of cartoonizing background scenes. We collected 8,227 real-world outdoor images from the *monet2photo* dataset [20] for the source photo domain. Then, this was split into 6,227 and 2,000 images for the train and test set. For the target cartoon domain, we collected cartoon datasets from Japanese animations and Webtoons. Specifically, we acquired animation images from 'The Garden of Words' and 'Your Name' by Shinkai Makato, and 'Spirited Away' by Miyazaki Hayao. For the Webtoon dataset, we collected comics of titles 'FreeDraw'[1] and 'Barkhan'[2] from the NAVER Webtoon platform.

We resized source domain images as 256×256 resolution. For cartoon datasets, we cropped images to be a resolution of 512×512 and applied ×2 super-resolution [19] beforehand when the raw image is <1024×1024 resolution. In total, we gathered images of 5,480 Hayao, 5,647 Shinkai, 5,308 FreeDraw, and 6,186 Barkhan datasets. In our experiment, we treated the above four style datasets as independent since each artwork has a unique style.

[1] https://comic.naver.com/webtoon/list?titleId=597447

[2] https://comic.naver.com/webtoon/list?titleId=650305

(a) Photo (b) w/ Pre-train (c) w/o Pre-train

Figure 2. **Is pre-training necessary?** Previous deep cartoonization methods [5, 6, 18] rely on pre-training strategy. However, CARTOONER does not require tedious pre-training at all; both cases show a very similar performance.

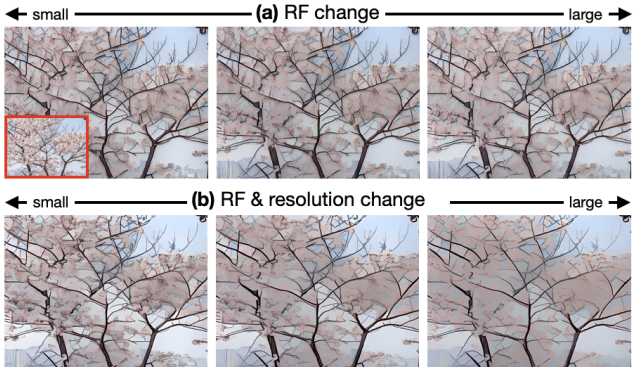
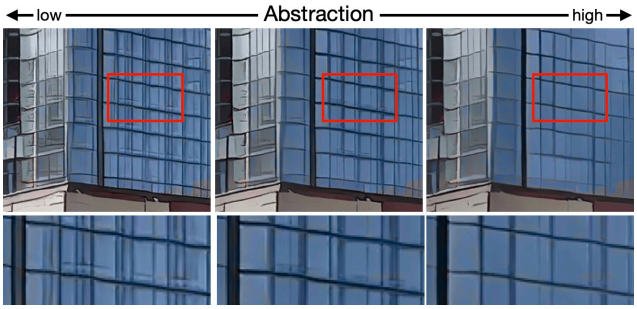


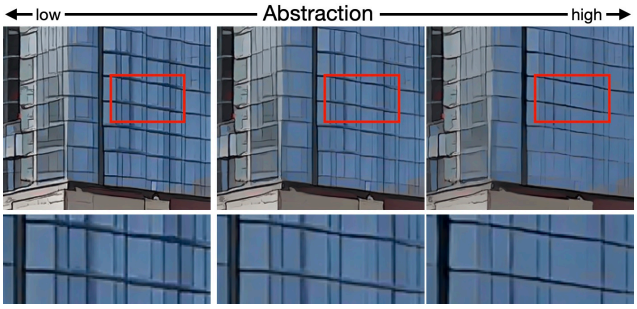
Figure 3. **What affects the abstraction level?** (**a**) When we only increase receptive field (RF) of the generator, the abstraction is not well changed. (**b**) When we increase both RF of the generator and the resolution of target cartoon images, the abstraction is adequately changed (this result (b) is also in the main text).

User study. We asked 26 participants to pick the best results for how well the outputs follow both the cartoon styles and source photos. Each of them was asked to vote on 16 questions, thus we collected 416 samples in total. In every question, we presented source photography, exemplar cartoon image, and the results of the previous and our methods. For better visibility, we also showed cropped patches for all the results. We computed the *quality preference* score by the ratio of the voted (as the best) cases.

Interactive UI. Figure 6 shows an interactive UI of CARTOONER. In the left panel, selection tools (*e.g.,* selection, quick selection, and eraser) offer mask-based region selection so the user can easily manipulate local region. In the right panel, style change tools offer texture and color con-



(a) Unshared conv kernel



(b) Shared conv kernel

Figure 4. **Study on the shared kernel in the abstraction control unit.** (**a**) When the unit does not share the conv kernels, the abstraction levels are not smoothly changed since each conv branch may learn different representations. (**b**) When the unit shares the conv kernels, the abstraction levels are gradually transited thanks to the shared representations.

trol over the selected region. Here, the creators can change the **1)** target cartoon style, **2)** texture (*i.e.,* stroke thickness and abstraction), and **3)** color of the cartoonized outcomes. For the color control, we provide both a color picker and an HSV control slider UIs since we observed that the latter is straightforward to utilize for unskilled users. Throughout this, the creators easily render given natural photos into the cartoon styles as well as manipulate the results with their own desired texture and color.

Naive coloring approach. In Figure 10, we presented a comparison between `Cartooner` and two alternative approaches that accept color control. The first one involves executing the re-colorization before the cartoonization (Figure 10a), while the second one involves the re-colorization after the cartoonization (Figure 10b). For both, we employed a palette-based re-colorization [4] and `Cartooner` that is trained without a color decoder.

Reference image-based color control. It can be achieved with a simple pipeline. With a given reference image, we extract the color palette through the K-means clustering. In our demonstration (Figure 13, main text), we used eight palettes (*i.e.,* clusters). Note that when the user selects the specific region (via selection tools) to transfer the color, we generate a color palette from that region instead. Then, the user chooses the region to be changed in a source photo and the framework calculates the average color of the region, denoted as c. In the meantime, the user also picks the color c' from the palette. Using these colors, c and c', the framework performs a palette-based color transfer algorithm [4] to the initial color map C_{src} and generates $\bar{C}_{src}$. However, we observed that a simple color transfer in RGB space (Eq. 1) also produces robust results. With the manipulated color map, CARTOONER now generates appropriate cartoonized outputs that fulfill the users' color intention.

$$\bar{C}^{RGB}_{src} = C^{RGB}_{src} + (c' - c) \tag{1}$$

C. Model analysis

Model training. We analyzed the pre-training that has been a prevalent strategy on deep cartoonization [5, 6, 18]. Previous studies reported that the warm-up process, which optimizes the network through the content loss only in advance, guarantees better convergence and cartoonization quality. However, we found that the network pre-training does not require to CARTOONER (Figure 2). We claim that a separate design of texture and color enables robust training since the texture and color decoders can solely concentrate on synthesizing texture and color alone, respectively.

What affects the abstraction? In Figure 3, we present results of the abstraction analysis where we only change the receptive field (RF) of the generator (Figure 3a), or change both RF of the generator and the image resolution of cartoon domain dataset (Figure 3b). Note that the latter result was shown in the main text. When we alter the RF of the generator alone, the abstraction seems not much affected since the low-complexity cartoon scene does not guide the generator, hence the generator would not have any incentive to increase the abstraction. On the other hand, as we discussed in Figure 5 (main text), increasing both RF and the resolution effectively affects the abstraction due to the scene complexity guidance from the cartoon images.

Abstraction control unit. We designed this unit to have a shared conv kernel scheme in a multi-branch system. When the conv kernels are not shared (Figure 4a), it is not guaranteed consistent and smooth abstraction transitions. For example, the detailed textural gradations near window edges intensify even when we increase the abstraction (2nd column). This is because each branch learns separate representations without communication, thus, they are not tuned to each other to generate smooth abstraction change. In contrast, the shared conv kernel approach (Figure 4b) adequately produces continuous abstraction modification. We claim that robustness can be achieved since the RF of the current abstraction is gradually evolved based on the previous RF (Figure 1). By doing so, all the branches share the viewpoint near the center point (of RF), and the kernels with

(a) kernel sizes: {3, 11, 19, 27, 35}

(b) kernel size: {3, 7, 11, 15, 19}

Figure 5. **Study on the kernel sizes in the abstraction control unit.** (**a**) When we extremely increase kernel sizes, the model tends to generate blurred output since the conv kernel refers to too many flat regions. (**b**) We found that this kernel size setting enables the best abstraction control. Decreasing kernel size also produces good abstraction, but is saturated on the highest level.

larger RF look wider region while maintaining the perspective of the previous ones. As a consequence, they produce consistent and gradual transitions on the abstraction.

We demonstrate the results of the different kernel size cases. In Figure 5a, we excessively expand the kernel sizes and the network generates blurred outputs since the model is guided from too many flat regions. In our experiment, we found that kernel size in Figure 5b shows the best abstraction change in terms of perceptual quality. However, the other settings (such as {3, 5, 9, 11, 13}) also make plausible outcomes as long as kernel sizes are in increasing order.

D. Discussion

Comparison to Jing et al. [10]. Unlike style transfer, which utilizes general artistic paintings, expressing abstraction is crucial in cartoonization since cartoon scenes have many flat regions. Thus, we decomposed stroke into *stroke thickness* and *abstraction*. For stroke thickness, our observations are similar to the stroke size analysis in Jing et al. [10]. However, we argue that the distinctiveness of the cartoon, characterized by its numerous flat regions, renders this distinctive more conspicuous than that of style transfer. For abstraction, we argue that our analysis based on *scene complexity* is more relevant to the cartoon domain, given the prevalence of flat regions and abstraction in this genre.

Comparison to WhiteBoxGAN [18]. We present a comparison between our proposed method and WhiteBoxGAN [18]. The shared objective of both methods is to decouple features in order to improve the training and synthesize quality. WhiteBoxGAN achieves this by decomposing representations in its loss design such as utilizing structure, texture, and surface losses. In contrast, `Cartooner` separates representations in the model design such as texture and color decoders. Upon comparing the "decomposition" in loss and model design, we posit that the latter approach provides a more explicit separation of features, as each decoder can concentrate on its specific task (texture or colormap generation). In contrast, when multiple losses are incorporated in a single decoder (as WhiteBoxGAN), the decoder may become confused due to the diverse signals emanating from multiple synthesis tasks. As a result, our framework offers the significant advantage of being more efficient during the training process, leading to superior quality with fewer artifacts, as demonstrated in Figure 8. Moreover, our model design philosophy is universally applicable, as the texture and color decoders are designed with identical structures that can be integrated into any network architecture.

Comparison to Stable diffusion [14]. We tackled image-to-image (I2I) based cartoonization. It requires maintaining the source photos' structure while altering their style and mood. In this respect, we compared the proposed method with *unpaired I2I approach* [5, 6, 18] since these faithfully meet the above requirements. Nevertheless, it is worth noting the application of recent diffusion models [8] in the cartoonization task considering its unprecedented progress in the image synthesis task [7, 13, 14]. To investigate this, we fine-tuned a pre-trained Stable diffusion (SD) [14] via Dreambooth strategy [15] and then incorporated the most popular I2I method, SDEdit [12], to make SD an I2I framework. As shown in Figure 7, SD with SDEdit struggles to produce satisfactory outcomes compared to `Cartooner`. When the denoising strength (s) is set to a higher value ($= 0.6$), it fails to preserve the source photo's content information, which hinders usability in the background-making process. When s is set to a lower value ($= 0.4$), it produces severe artifacts, which results in inferior visual quality than ours. We suspect this is due to SDEdit's inability to effectively eliminate high-frequency information when the denoising strength is low. In the future, the SD-based content-preserving I2I approach would be a valuable research topic.

Comparison to cartoon filters. The majority of cartoon filters in commercial software [2, 3] rely on traditional algorithms so they support very limited cartoon styles. In this respect, "deep cartoonization" studies have not included comparisons with filters since filters cannot express various

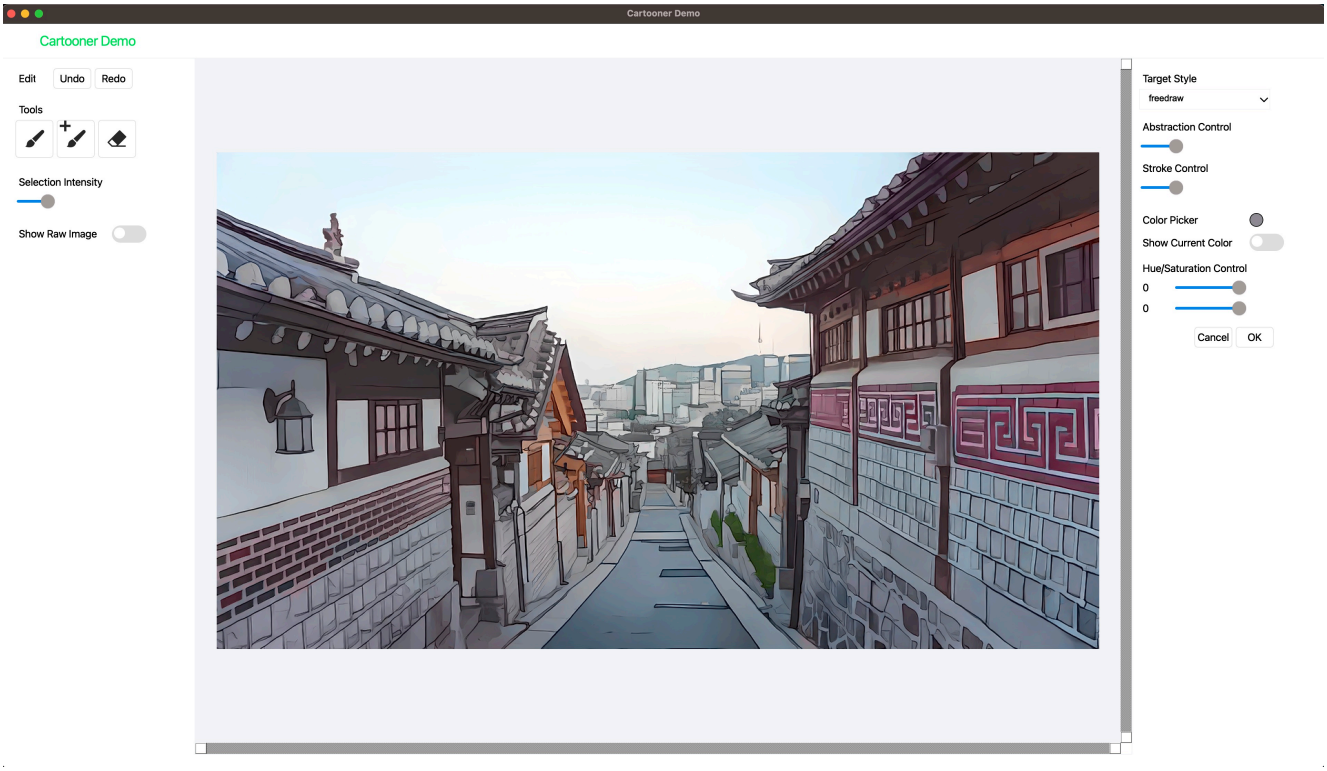


Figure 6. **Interactive UI of CARTOONER.**

styles that deep methods can produce. In addition, through interviews with many artists, we found that commercial filters require significant retouching to fit the artists' desired cartoon style and inevitably consume extensive effort. In contrast, deep cartoonization [5, 6, 18], including ours, can stylize images into a diverse range of styles with superior quality, thus reducing the need for thorough retouching.

Limitation. Although `Cartooner` successfully demonstrates an interactive to the cartoonization task, other properties could be incorporated to turn into a more artist-friendly solution. In our study, we have focused on color and texture, as these are (in our early interviews) the most needed aspects by artists. Nevertheless, the lack of other features might limit its usability in many scenarios. For example, stroke shape (or style) control, sky synthesis (Figure 2), vectorization, or layer decomposition which are commonly used by artists, would greatly enhance the workflow.

E. Additional results

Figure 8 displays interactive scenarios and Figure 9, 10, 11, 12 show qualitative comparison.

(a) Source Photo

(b) `Cartooner` (ours)

(c) SD [14] + SDEdit [12] ($s = 0.6$)

(d) SD [14] + SDEdit [12] ($s = 0.4$)

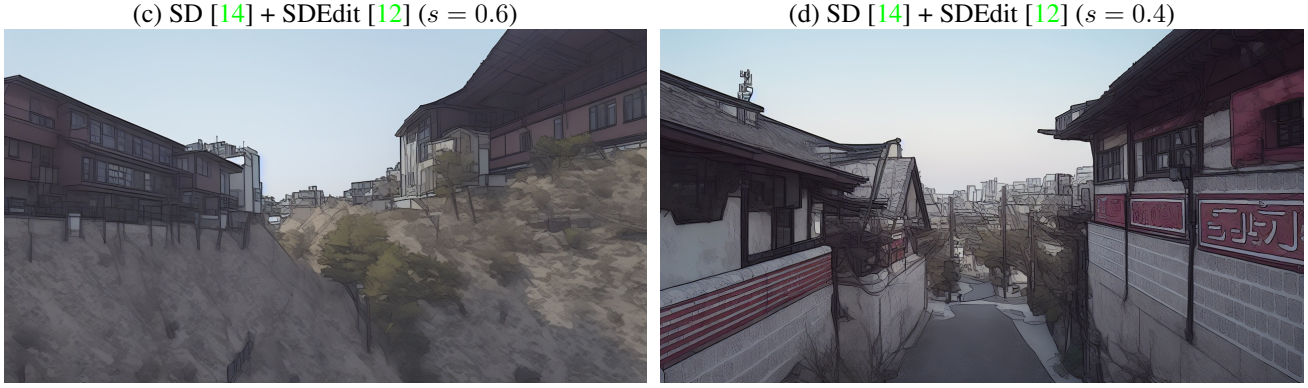


Figure 7. **Comparison to Stable diffusion (SD) [14] based image-to-image translation method [12].** Best viewed in zoom.

Figure 9. **Visual comparison.** Images along with source photos indicate exemplar images of the target cartoon. Best viewed in zoom.

Figure 10. **Visual comparison.** Images along with source photos indicate exemplar images of the target cartoon. Best viewed in zoom.

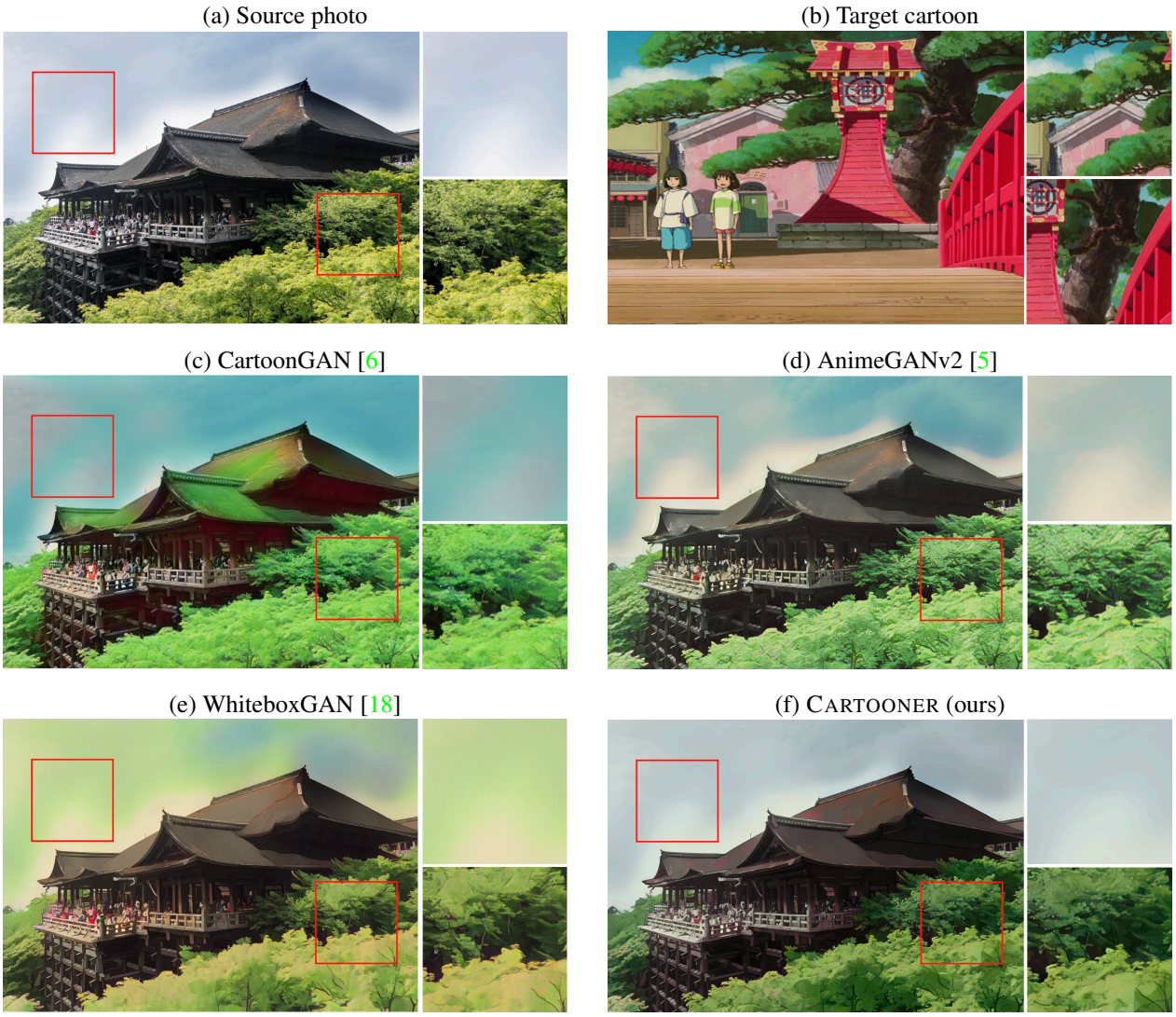


Figure 11. **Visual comparison.** Images along with source photos indicate exemplar images of the target cartoon. Best viewed in zoom.

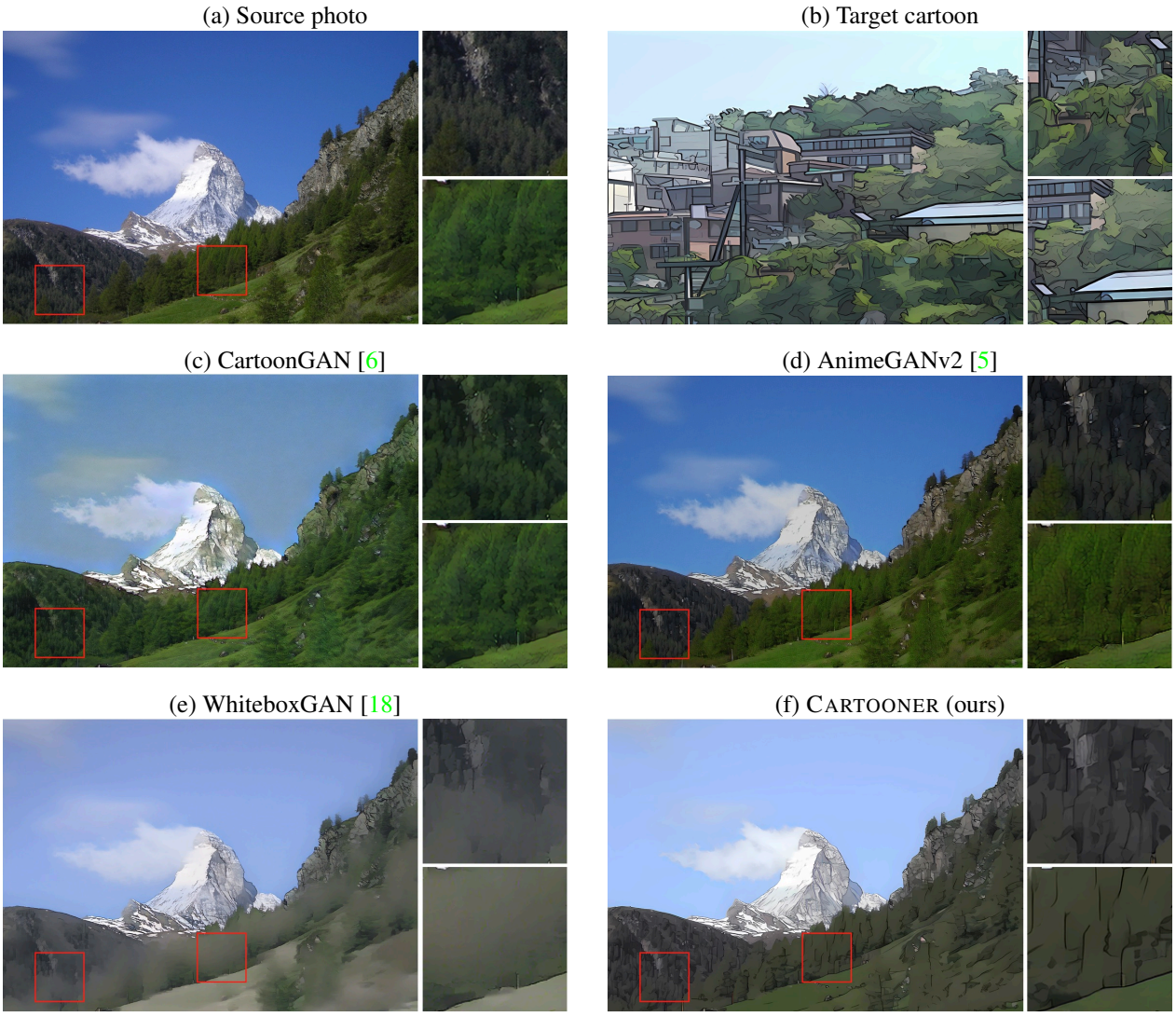


Figure 12. **Visual comparison.** Images along with source photos indicate exemplar images of the target cartoon. Best viewed in zoom.

References

[1] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012. 2

[2] Adobe. Adobe photoshop, 2022. 4

[3] CELSYS. Clip studio paints, 2022. 4

[4] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. Palette-based photo recoloring. *ACM Trans. Graph.*, 34(4):139–1, 2015. 3

[5] Jie Chen, Gang Liu, and Xin Chen. Animegan: A novel lightweight gan for photo animation. In *International Symposium on Intelligence Computation and Applications*, pages 242–256. Springer, 2019. 1, 2, 3, 4, 5, 8, 9, 10, 11

[6] Yang Chen, Yu-Kun Lai, and Yong-Jin Liu. Cartoongan: Generative adversarial networks for photo cartoonization. In *CVPR*, pages 9465–9474, 2018. 1, 2, 3, 4, 5, 8, 9, 10, 11

[7] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in Neural Information Processing Systems*, 34:8780–8794, 2021. 4

[8] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 4

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015. 1

[10] Yongcheng Jing, Yang Liu, Yezhou Yang, Zunlei Feng, Yizhou Yu, Dacheng Tao, and Mingli Song. Stroke controllable fast style transfer with adaptive receptive fields. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 238–254, 2018. 4

[11] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 1

[12] Chenlin Meng, Yutong He, Yang Song, Jiaming Song, Jiajun Wu, Jun-Yan Zhu, and Stefano Ermon. Sdedit: Guided image synthesis and editing with stochastic differential equations. In *International Conference on Learning Representations*, 2021. 4, 6

[13] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021. 4

[14] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 4, 6

[15] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. *arXiv preprint arXiv:2208.12242*, 2022. 4

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 2

[17] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 1

[18] Xinrui Wang and Jinze Yu. Learning to cartoonize using white-box cartoon representations. In *CVPR*, pages 8090–8099, 2020. 1, 2, 3, 4, 5, 8, 9, 10, 11

[19] Xintao Wang, Ke Yu, Shixiang Wu, Jinjin Gu, Yihao Liu, Chao Dong, Yu Qiao, and Chen Change Loy. Esrgan: Enhanced super-resolution generative adversarial networks. In *Proceedings of the European conference on computer vision (ECCV) workshops*, pages 0–0, 2018. 2

[20] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017. 2

[21] Lei Zhu, Qi She, Bin Zhang, Yanye Lu, Zhilin Lu, Duo Li, and Jie Hu. Learning the superpixel in a non-iterative and lifelong manner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1225–1234, 2021. 2