Interactive Cartoonization with Controllable Perceptual Factors

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

Figure 1. **Interactive cartoonization.** The proposed cartoonization method allows user interaction over texture and color and can generate diverse outputs that meet users' demands. Leftmost: source photography and an exemplar image of the target cartoon.

Abstract

Cartoonization is a task that renders natural photos into cartoon styles. Previous deep cartoonization methods only have focused on end-to-end translation, which may hinder editability. Instead, we propose a novel solution with editing features of texture and color based on the cartoon creation process. To do that, we design a model architecture to have separate decoders, texture and color, to decouple these attributes. In the texture decoder, we propose a texture controller, which enables a user to control stroke style and abstraction to generate diverse cartoon textures. We also introduce an HSV color augmentation to induce the networks to generate diverse and controllable color translation. To the best of our knowledge, our work is the first deep approach to control the cartoonization at inference while showing profound quality improvement over to baselines.

1. Introduction

Cartoons gain steep popularity in a recent, and the number of cartoon creators have also increased. The universal workflow of cartoon painters is as follows: character drawing, which is then composed into a background scene. Post-processing such as shading is added afterward. Professional tools [1, 3] provide helpful plugins to assist the artist. Despite this, cartoon creation still remains an arduous task even for the more skilled creators.

We follow the observation that cartoon-styled scene generation has received notable attention. Many artists convert real-world photographs into cartoon styles to utilize as a background scene, dubbed as *image cartoonization*. This allows creators to more focus on effective decisions in making cartoons, such as character generation. It is shown that deep learning-based cartoonization approach is able to produce cartoon-stylized output with a prominent quality, that is possible to be utilized in real service production [5, 6, 19].

However, the previous deep methods skip the intermediate procedures of cartoon-making processes, thus disabling the creators from controlling outputs. The artists follow a series of structured steps when creating a cartoon background from a photo (Figure 2). **1)** Color stylization, where the author changes the color both locally and globally. Sky synthesis is performed along with this procedure. **2)** Texture stylization, where additional sketch lines are drawn, and fine details are selectively removed to achieve the different

(a) Source photo (b) Sky synthesis (c) Color stylization (d) Texture stylization (e) Post-processing

Figure 2. **Example of the background making process.** We visualize how the artist creates the background scene by the step-by-step procedure. Note that some steps can be skipped or changed in the order depending on the artist. ©Kawaii Studio

levels of abstraction. **3)** Post-processing, which includes lighting and image filters. Unfortunately, due to the end-to-end inference nature of the previous deep cartoonization methods, the artist has no control over the generation process. The creators may only intervene with a source photo (Figure 2a) or the final output (Figure 2e), which harms the usability of the cartoonization methods in artists' workflow.

In this study, we present an effective approach to embedding interactivity in cartoonization. The proposed solution focuses on building a pipeline for more controllable texture and color. We define texture control as the manipulation of stroke thickness and abstractions. This concept can be utilized in many scenarios; the artist can abstract the details of the far-distance scene to depict the natural perspective or emphasize the details of the character. The creators can also choose to change the delicacy of the brushstroke to match the texture of the foreground objects when composing the scene. As for color control, we aim to build a control system in which the creator freely manipulates arbitrary regions with the desired color. This is designed to assist the artist in the color stylization procedure (Figure 2c).

To obtain user controllability in cartoonization, we separately build texture and color decoders to minimize interference across the features (Figure 3). We also found that the decomposed architecture provides a robust and superb quality of texture stylization. For texture control, we investigated the role of the receptive field and the target image resolution in the level of stroke thickness and abstraction. Based on these observations, we present a *texture controller*, which adjusts the receptive field of the network through a dynamic replacement of the intermediate features. For color control, we jointly train the color decoder in a supervised manner with the paired dataset that is built based on the proposed HSV augmentation. Throughout this training strategy, the color module gains the ability to produce diverse colors. With the combination of the decoupled texture and color modules, we achieve a two-dimension of control space that can create a variety of cartoonized results upon user communication. Such a design also provides robust and perceptually high-quality cartoonized outcomes.

To the best of our knowledge, our framework is the first approach that presents interactivity to deep learning-based cartoonization. Based on the proposed solution, we demonstrate application scenarios that permit user intentions to create cartoonized images with diverse settings. Extensive experiments demonstrate that the proposed solution outperforms the previous cartoonization methods in terms of perceptual quality, while also being able to generate multiple images based on the user's choices of texture and color.

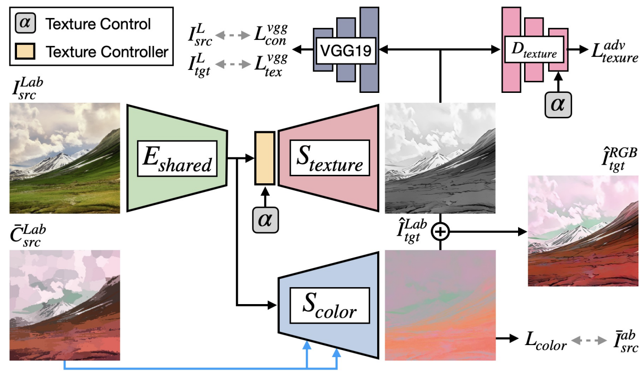


Figure 3. **Model overview.** Given a photo, CARTOONER estimates the stylized texture and color images, which are then composed for the final product. We design decomposed texture/color paths, a texture controller, and a multi-texture discriminator for interaction.

2. Related work

Non-photorealistic rendering. Non-photorealistic rendering (NPR) is a computer vision/graphics task that focuses on representing diverse expressive styles for digital art. Among a variety of subtasks of NPR, we briefly describe the methods that stylize natural domain image to the specific artistic style. Because of its usefulness in digital art creation, NPR has been expanded into various applicable scenarios such as line drawing [20], image abstraction [21], and cartoonization [18]. Style transfer methods [9, 13] are notable approaches in NPR. By jointly optimizing the content and style losses, they can generate decent-quality of stylization.

Cartoonization. Deep learning-based methods show profound improvement over conventional NPR algorithms on this task. CartoonGAN [6], a pioneering study on deep cartoonization, adopts adversarial training [10] along with

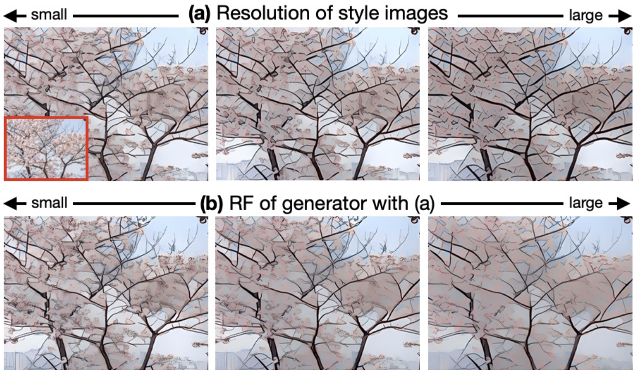


Figure 4. **What affects texture levels?** (**a**) As the resolution of target cartoon (style) images is increased, the stroke is thickened. (**b**) As both the resolution of images and the receptive field (RF) of the generator is increased, the abstraction becomes a high degree.

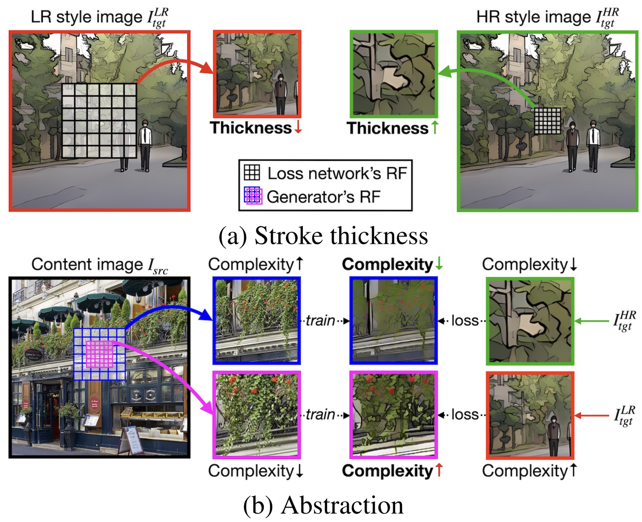


Figure 5. **Why affects texture levels?** (**a**) With a fixed receptive field (RF) of loss network, as the resolution of target domain images changes, the stroke thickness within an RF window varies. (**b**) With a fixed resolution of a content image, as the RF of the generator changes, the scene complexity within an RF window varies. When we provide style images with disparate complexity, the generator alters the scene complexity of the cartoonized results.

an edge-promoting loss to improve cartoon style. AnimeGAN [5] enhances CartoonGAN with advanced losses suitable for cartoon style such as Gram-based loss [9]. With a careful inspection of cartoon drawing process, Whitebox-GAN [19] decomposed cartoon images as surface, structure, and texture representations to tackle each factor with tailored losses. This approach achieves superior cartoonization quality compared to previous methods.

Despite the imposing cartoonization results, none of the deep cartoonization methods support interaction, failing to create diverse conditional outputs. Our method aims to enable user control while maintaining perceptually appealing cartoonization, moving closer to the actual service level.

3. Method

We describe the interactive cartoonization method dubbed as CARTOONER. It uses separate decoders for texture and color (Figure 3), contrary to the single decoder architecture of the previous methods. The decision was made by observing professional artists' workflow, where they separate color modification from texture editing. We further inspect that isolated modeling of texture and color produces reliable and high-quality cartoonized results.

The controllable features are defined as texture level vector $\boldsymbol{\alpha}$ and users' color modification $\boldsymbol{c}$. Given a photo I_{src}, the goal is to generate a cartoonized image $\hat{I}_{tgt}$ that follows the user intention $\boldsymbol{\alpha}$ and $\boldsymbol{c}$. To achieve this, CARTOONER encodes an image to the latent feature through E_{shared}, then delivers it to the separate decoders, $S_{texture}$ and S_{color}. Note that we use Lab color space instead of RGB, hence the texture module produces an L-channel texture map, while the color module generates an ab-channel color map. These outputs are finalized by converting back to RGB space.

3.1. Texture module

Analysis of texture level. In this module, the primary goal is to provide a fine control mechanism and we define texture control as altering *stroke thickness* and image *abstraction*. To do that, we first analyze which components influence stroke thickness and abstraction change. In our preliminary experiments, we observed that increasing the resolution of target cartoon images affects stroke thickness, and expanding the receptive field (RF) of the generator along with the increased resolution inflates the abstraction level (Figure 4).

For stroke thickness change, we argue that the *loss network with a fixed RF* (*e.g.,* VGG or discriminator) is involved as shown in Figure 5a. Given the fixed loss network, when we increase the resolution of cartoon images (green box), the strokes are enlarged within an RF window, thus inevitably, the generator learns to produce thick strokes at training. When we decrease the resolution (red box), the opposite behavior occurs. This impacts cartoonization more since cartoon images mostly have flat texture regions.

For abstraction change, we argue that *scene complexity* affects this as shown in Figure 5b. When the RF of the generator is expanded (blue box), the network can perceive a wider region of a content image, which results in high scene complexity. In contrast, when the resolution of a cartoon image grows, its scene complexity becomes lower since the loss network can only see relatively tiny regions (green box). With these, if we utilize high-resolution cartoon images I_{tgt}^{HR} to train the generator with a large RF (which are

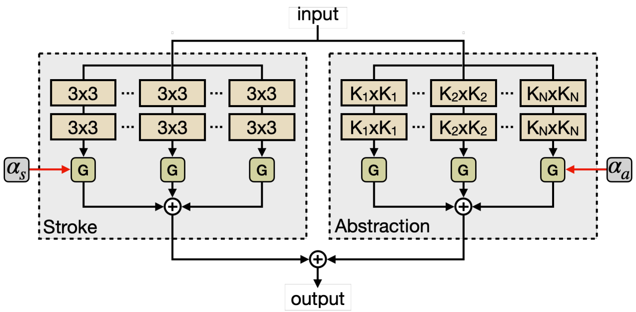


Figure 6. **Texture controller.** This module consists of the stroke and abstraction control units, which both are designed as multi-branch. Intermediate features from branches are fused through a gating unit, which is controlled by texture factors, $\{\alpha_s, \alpha_a\}$. In the abstraction unit, kernel sizes of $K_1, ..., K_N$ are increasing order.

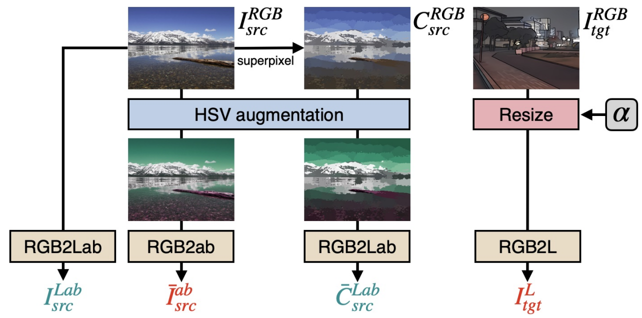


Figure 7. **Data preprocessing on training.** Given a source photo I_{src}^{RGB}, we prepare an input color map C_{src}^{RGB}. Then, the HSV augmentation is operated on both I_{src}^{RGB} and C_{src}^{RGB}. Target domain (cartoon) images I_{tgt}^{RGB} are resized by referring to the prefixed texture control levels $\boldsymbol{\alpha}$. All the processed images are then converted to Lab, L, or ab space. The green symbols indicate the input data and red ones represent data used in the loss calculation.

green and blue box, respectively), the generator is guided to reduce the complexity of the high-complexity scene. This arises from the loss calculation with the low-complexity scene extracted from the loss network. As a result, the generator with a large RF gains the ability to "abstract" the complex details. For the lower RF (pink window), the contrary behavior happens. However, the abstraction change is not as dramatic as the high one since in general, the scene complexity of cartoon is lower than the content images. We also analyzed the scenario where *only RF of the generator is expanded*, however, the results are not drastic as Figure 4b since the generator is not guided by the different-complexity cartoon scene. Note that Jing et al. [12] inspected the role of the resolution and RF in style transfer literature, however, our in-depth analysis reveals that their behavior patterns are disparate in cartoonization.

Texture controller. The above analysis requires multiple networks to handle diverse levels and it cannot produce consistent styles for each other. Also, it only supports discrete control levels, making it challenging to be used as a real-world solution. Hence, based on the analysis, we introduce a simple but effective texture control module, dubbed as *texture controller* (Figure 6). It consists of the stroke and abstraction control units and we design both units as to be a multi-branch architecture. In the stroke unit, each branch is composed of two consecutive 3×3 conv layers, and these are fused by the gating module. The abstraction unit is identical in structure to the stroke unit except it uses conv layers with large kernel size, $K_1 < K_2 < ... < K_N$.

The texture controller is influenced by texture level $\boldsymbol{\alpha} = \{\alpha_s, \alpha_a\}$, specifically, the stroke and abstraction units are guided by stroke thickness α_s and abstraction α_a levels, respectively. With the feature f from the encoder, the stroke unit generates a feature set $\boldsymbol{g}_s = \{g_s^1, ..., g_s^N\}$ through conv branches, and the abstraction unit produces $\boldsymbol{g}_a$ as the same way. Then, according to texture levels $\alpha_{\{s,a\}}$, which are a positive rational number, the two features of $\boldsymbol{g}_{\{s,a\}}$ with indices closest to a texture level are chosen. The chosen features are then interpolated based on the respective distance between $\alpha_{\{s,a\}}$ and indices. Finally, these are combined by an element-wise addition operation.

We design the stroke control unit to have all 3×3 conv layers since the texture level analysis showed that RF of the generator does not affect the stroke thickness. Instead, each branch is trained by different resolutions of target cartoon images. At inference, the feature interpolation via α_s enables continuous control over stroke thickness. For the abstraction unit, we also construct a single module based on the analysis. However, unlike the stroke unit, each branch includes conv layers with different kernel sizes (with increasing order) because changing both the RF of the generator and the resolution of target images alters the abstraction. The output features are interpolated through α_a, as identical to the stroke unit. As we design the decoupled structure of stroke and abstraction in parallel, each unit can concentrate on a different aspect and it provides the ability that can control the texture as a two-dimensional space.

In addition, to incorporate adversarial learning [10] into texture control, we utilize a multi-texture discriminator. It is based on the multi-task discriminator [8,16], which consists of multiple output branches. Each branch corresponds to a different texture level and learns to distinguish whether a given image is from a real cartoon domain or generated.

Loss function. We use adversarial loss $L_{texture}^{adv}$ to guide the model mimicking the texture of the target cartoon.

$$L_{texture}^{adv} = log D_{\boldsymbol{\alpha}}(I_{tgt}^{L,\boldsymbol{\alpha}}) + log(1 - D_{\boldsymbol{\alpha}}(G(I_{src}^{Lab}, \boldsymbol{\alpha}))) \quad (1)$$

where G is the generator and $D_{\boldsymbol{\alpha}}$ denotes the multi-texture discriminator with a given texture factors $\boldsymbol{\alpha}$. $I_{tgt}^{L,\boldsymbol{\alpha}}$ is a resized target cartoon image to fit a respective texture level $\boldsymbol{\alpha}$.

(a) Source photo (b) CARTOONER (c) CartoonGAN [6] (d) AnimeGANv2 [5] (e) WhiteboxGAN [19]

Figure 8. **Visual comparison.** Images along with source photos indicate exemplar images of the target cartoon. Best viewed in zoom.

To ensure a cartoonized output well preserves the semantic information of a source photo, we employ content loss.

$$L^{vgg}_{content} = ||VGG(I^{L}_{src}) - VGG(G(I^{Lab}_{src}, \boldsymbol{\alpha}))||_1. \quad (2)$$

We use a *conv4_4* layer of the pre-trained VGG19 [17]. In addition, we enforce the generator to learn high-level texture representation via Gram-based loss as:

$$L^{vgg}_{texture} = ||Gram(I^{L,\boldsymbol{\alpha}}_{tgt}) - Gram(G(I^{Lab}_{src}, \boldsymbol{\alpha}))||_1. \quad (3)$$

$Gram$ indicates the Gram calculation with VGG feature extraction (of *conv4_4*). We also use total variation loss [2] to impose spatial smoothness on the output.

$$L^{tv}_{texture} = ||\nabla x(G(I^{Lab}_{src}, \boldsymbol{\alpha})) + \nabla y(G(I^{Lab}_{src}, \boldsymbol{\alpha}))||_1 \quad (4)$$

With balancing parameters $\lambda^{1,\ldots,4}_{texture}$, the final loss of the texture decoder (and the shared encoder) is defined as:

$$L_{texture} = \lambda^1_{texture} * L^{adv}_{texture} + \lambda^2_{texture} * L^{vgg}_{content} + \lambda^3_{texture} * L^{vgg}_{texture} + \lambda^4_{texture} * L^{tv}_{texture}. \quad (5)$$

3.2. Color module

The goal is to transfer the color of a given source photo to a provided color intention while reflecting the color *nuance* of the target cartoon. CARTOONER takes an input photo I^{Lab}_{src} as well as an input color map $\bar{C}^{Lab}_{src}$ and generates an ab-channel image $\hat{I}^{ab}_{src}$, which is later concatenated with the texture map $\hat{I}^{L}_{tgt}$ generated from the texture decoder. To simulate control manipulation, we synthetically generate a color map, $\bar{C}^{Lab}_{src}$ (Figure 7). Given an input photo I^{RGB}_{src}, we create an initial color map C^{RGB}_{src} by applying a superpixel algorithm. Without this, fine details of an input image become too noisy and thus not adequate to be utilized as a color cue; a superpixel is used as a noise reduction procedure. Then, the HSV augmentation changes the color of I^{RGB}_{src} and C^{RGB}_{src}, creating color manipulated images $\bar{I}^{RGB}_{src}$ and $\bar{C}^{RGB}_{src}$. These are converted to Lab space. Note that we observed that color transfer to either input or output image, unlike ours, cannot achieve faithful visual quality.

HSV augmentation is a simple but effective method that can reflect diverse color control intentions from a user. It randomly alters all color channels of HSV; hue, saturation, and value (brightness). To prevent the color shifting from generating perceptually implausible outputs, we further apply the L caching trick [7] prior to the color augmentation, which caches the luminance (L) of image and reverts the luminance of the augmented image to a cached one. We cache L instead of V-channel since V indirectly interferes with L, which is important in regard to diverse cartoonization.

Loss function. We use a simple mean squared error-based reconstruction loss as shown below.

$$L_{color} = ||\bar{I}^{ab}_{src} - G(I^{Lab}_{src}, \bar{C}^{Lab}_{src})||_2 \quad (6)$$

In our experiment, additional adversarial loss or regulariza-

Table 1. **Quantitative comparison.** We compare with previous deep cartoonization methods using FID [11] and FID_{CLIP} [15] metrics. Lower score is better.

Method	Hayao		Shinkai		FreeDraw		Barkhan	
	FID	FID_{CLIP}	FID	FID_{CLIP}	FID	FID_{CLIP}	FID	FID_{CLIP}
CartoonGAN	172.63	59.09	167.66	56.67	162.69	49.09	108.80	28.04
AnimeGANv2	156.21	57.81	146.28	54.62	132.67	45.27	82.67	23.38
WhiteboxGAN	153.50	63.38	150.69	58.53	120.12	41.10	93.25	27.67
CARTOONER	**142.77**	**56.76**	**132.57**	**51.78**	**103.88**	**28.50**	**74.68**	**16.44**

Table 2. **User study.** Higher quality preference score is better.

Method	Preference
CartoonGAN	8.9%
AnimeGANv2	17.5%
WhiteboxGAN	16.1%
CARTOONER	**57.5%**

tion shows marginal improvement in color quality. We suspect that the decomposed color modeling, as well as the color cue ($\bar{C}^{Lab}_{src}$) provision, ease the training difficulty.

Reflecting target cartoon's color. We designed to preserve the color information of an input image to increase the controllability, however, one might want to generate an image that has a similar color distribution to the target cartoon. To handle this scenario, we additionally fine-tune the color decoder with an assist of adversarial loss as in below.

$$L^{tgt}_{color} = \lambda^1_{color} * L_{color} + \lambda^2_{color} * L^{adv}_{color} \quad \text{where,}$$
$$L^{adv}_{color} = logD(I^{ab}_{tgt}) + log(1 - D(G(I^{Lab}_{src},\ C^{Lab}_{src}))) \quad (7)$$

Note that we use a color cue that is generated from an original image (C^{Lab}_{src}), instead of $\bar{C}^{Lab}_{src}$. With the aforementioned color decoder parts, a user can choose which "color mode" to use interchangeably depending on the situation.

3.3. Model training

Unlike previous deep cartoonization methods, we do not perform network warm-up [6]. We train the entire framework with a loss of $L = L_{texture} + L_{color}$, except for the abstraction control unit. Then, the abstraction unit is trained (via $L_{texture}$) while other components are all frozen. To provide various resolution images to the generator, we resize I^{RGB}_{tgt} according to texture level $\boldsymbol{\alpha}$ (Figure 7). We set kernel sizes of the abstraction unit, $\{K_1, K_2, ..., K_N\}$, as $\{3, 7, 11, 15, 19\}$ each. When training CARTOONER, we randomly choose $\alpha_{\{s,a\}} \in \{1, ..., 5\}$, which respectively resize I^{RGB}_{tgt} to be $\{256^2, 320^2, 416^2, 544^2, 800^2\}$ resolutions, but $\alpha_{\{s,a\}}$ can be expanded to arbitrary numbers at inference. More detailed setups are described in Suppl.

4. Experiment

Baselines. We compare CARTOONER with the state-of-the-art deep learning-based cartoonization methods, CartoonGAN [6], AnimeGANv2 [5], and WhiteboxGAN [19]. Since they have trained their cartoonization network on different datasets and setups from each other, we retrained using our cartoon datasets using official codes.

Datasets. We built datasets focused on landscape, to better target the domain of cartoonizing background scenes. We used *monet2photo* [22] as the photo domain. We collected cartoon datasets from Japanese animations and Webtoons. Specifically, we acquired artworks by Miyazaki Hayao and Shinkai Makoto, and comics of titles "FreeDraw" and "Barkhan" from the NAVER Webtoon platform. Detailed dataset generation protocols are described in Suppl.

Metrics. We evaluated the cartoonization with Frećhet Inception Distance (FID) [11] and FID_{CLIP} [15]. We additionally conducted a user study to measure perceptual quality. We asked 26 users to select the best results for how well the outputs follow both the cartoon styles and source photos.

4.1. Comparison with state-of-the-art method

When comparing CARTOONER with others, we generate images to reflect the target cartoon since FIDs can be influenced by color information. In addition, we set the texture levels (α_s, α_a) as zero, which is identical stylization setting to others. Table 1 shows the quantitative comparison. CARTOONER achieves exceeding performance on both FID and FID_{CLIP} with significant margins for all the cases. We also present the visual comparison in Figure 8. Separation of the texture and color decoders helps prevent image artifacts, for instance, CARTOONER produces fewer color bleeding (Figure 8, 2nd row). The visual quality is also profoundly enhanced and CARTOONER can capture adequate stroke and color nuance of the target cartoon. A user study shows the superiority of CARTOONER as well (Table 2).

4.2. Interactivity

As shown in Figure 9, CARTOONER creates diverse results according to user interaction. When the artist manipulates the colors to their tastes (with any color adjustment UI), CARTOONER automatically reflects the intention. They can also edit textural details by simply controlling the stroke or abstraction factors to match the output in various cartoon situations. These editings can be performed locally or globally through a simple mask-based region control UI (shown in Suppl.). Our cartoonization workflow is more compact than the traditional editing tools, while still maintaining an adequate level of user intervention. Although CARTOONER may not achieve the degree of meticulous editing workflows (which requires the effort of skilled artists), it can

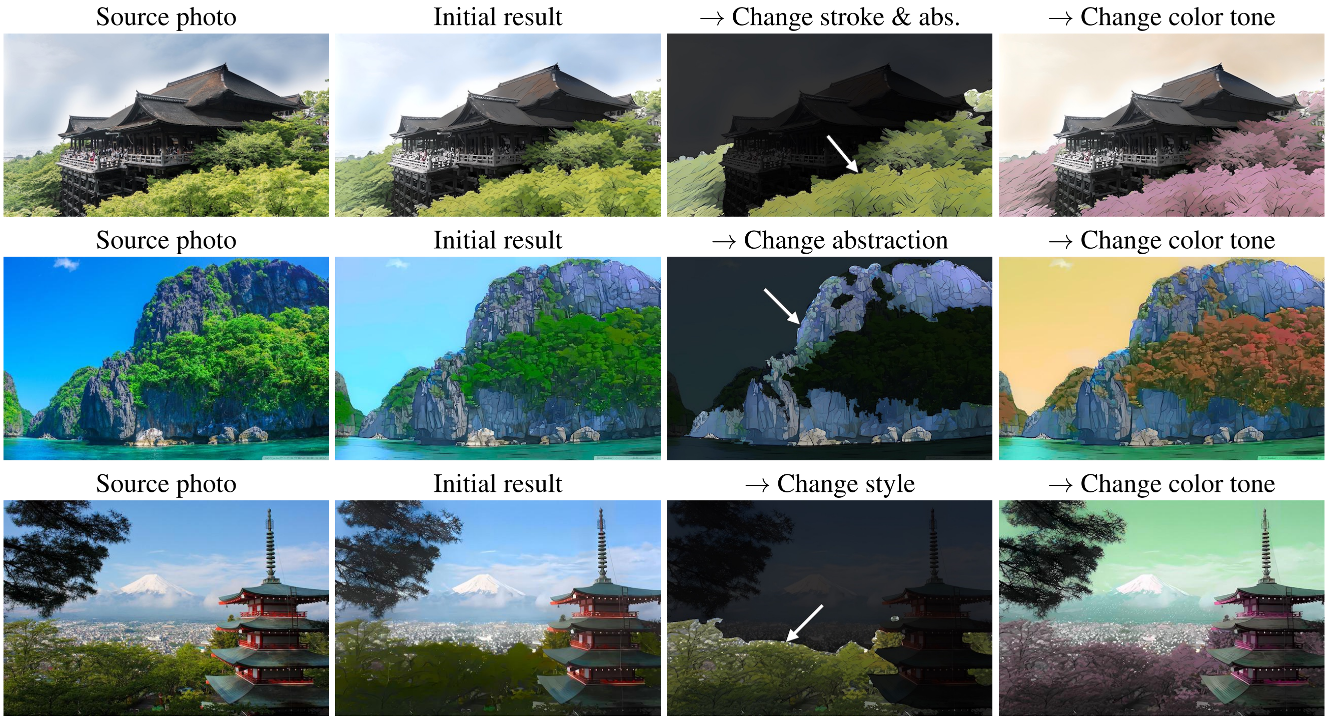


Figure 9. **Results of interactive cartoonization.** In 3rd column, we highlight the altered region for better visualization.

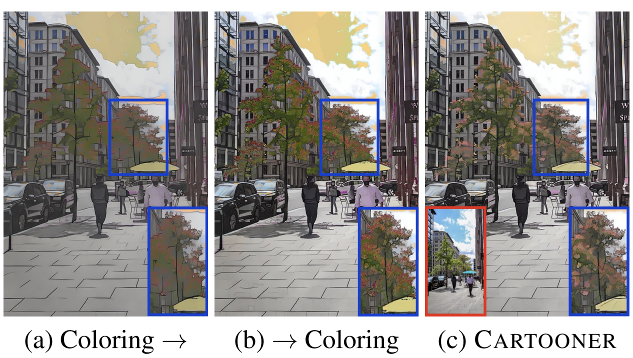


Figure 10. **Comparison to naive coloring approach.** We alter the color of sky to yellow and tree to orange maple. (**a**) A pipeline of re-colorization (to an input image) → cartoonization. (**b**) A pipeline of cartoonization → re-colorization. (**c**) Our method.

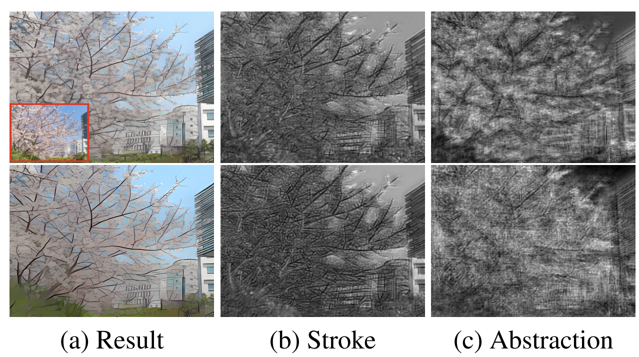


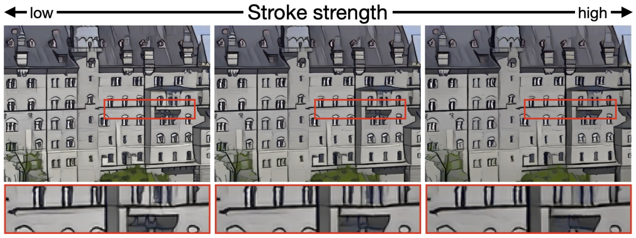
Figure 11. **Output feature of texture controller.** (**Top**) Lower stroke thickness and abstraction levels. (**Bottom**) Higher texture levels. The stroke unit focuses on fine details regards to edge region while the abstraction unit concentrates on the overall region.

provide a broader range of user experiences with suitable quality. We would like to emphasize that none of the deep cartoonization methods can provide controllability nor produce diverse results of a given source photography.

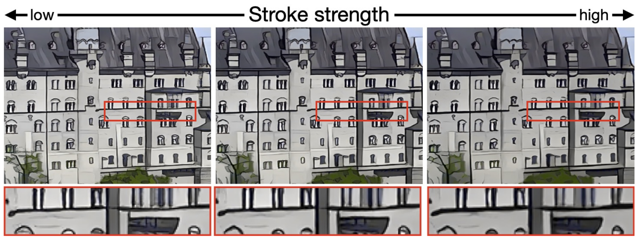
4.3. Model analysis

Color module. In Figure 10, we present the result where the color change is performed before or after cartoonization, unlike ours that jointly models the color and texture. The pre-execution of color change (Figure 10b) cannot adequately handle the delicate color alters and produces uneven texture level since the model has not observed the recolorized input image at training, which becomes out-of-distribution. The pipeline of color change after cartoonization (Figure 10c) cannot generate cartoon-style colors at all.

What does texture controller learn? We visualize the output feature map of the texture controller in Figure 11. The stroke control unit produces features that more concentrate on the high-frequency edge regions, which empirically demonstrates why this can control the stroke thickness. On the other hand, the abstraction control unit focuses on a wide range of regions including flat texture and some



(a) # levels = 2



(b) # levels = 5 (ours)

Figure 12. **Study on the stroke levels.** We compare the model trained with 2-level and 5-level stroke thickness. The 2-level case cannot adequately reflect the subtle change in stroke thickness.

mid-frequency details. Consequently, this unit can deliver helpful clues about the abstraction change to the decoder.

Stroke control unit. We decrease the number of stroke levels at training and examine how the models react at inference. Figure 12 shows that the model trained with 2-levels cannot capture high stroke thickness; we observed that it produces saturated thickness only. In contrast, the model with increased stroke levels adequately expresses a wide dynamic range of stroke thickness. In Suppl., we demonstrate a similar analysis regards on the abstraction control unit.

5. Application

Reference image-based color control. In Section 4.2, we demonstrated a simple interactive cartoonization workflow with CARTOONER. However, unskilled users might struggle to choose appropriate color tones if they have little experience in coloring. To increase usability for inexpert users, we present reference image-based color control (Figure 13). Instead of direct color manipulation, a user prepares a color guidance image and chooses which regions to be referred via region masking UI. After this, CARTOONER transfers the color information of the selected area to the cartoonized outputs. To implement this, we first extract a color palette from a reference image and then manipulate the color map, C_{src} using palette-based color transfer algorithm [4].

Semi-automatic cartoon making. As discussed, making background scenes is repetitive and time-consuming. CARTOONER can help to reduce the burden of background creation with interactive texture-color editing so the artists can focus more on other creative tasks. Figure 14 shows an example where one can use CARTOONER to effectively create a cartoon cut, consisting of a background scene blended with character(s) and/or speech balloons, which would have been a strenuous task for previous pipelines.

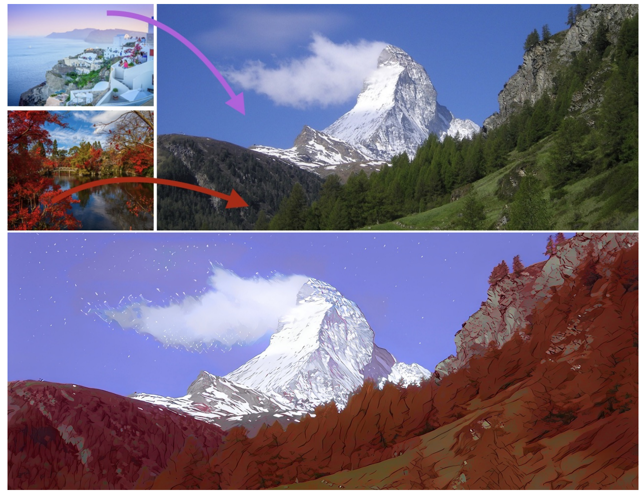


Figure 13. **Reference-based color control.** (**Top**) Color-reference images and a source photo. (**Bottom**) Cartoonized output.

(a) Source photo

(b) Result

Figure 14. **CARTOONER in cartoon making.** The artists can employ CARTOONER to their workflow to improve productivity.

6. Conclusion

We proposed an interactive cartoonization model, CARTOONER. The proposed method accepts user-guided texture control in the form of abstraction and stroke strength levels, which are passed to a *texture controller* to dynamically control the overall texture of the generated image. The user can also manipulate the color scheme through a color module, which is reinforced by the HSV augmentation. Experimental results demonstrate CARTOONER's superiority in both quality and usability as applications for cartoon creators.

Although we provided effective control space, there exist more controlling factors, especially for texture control (Figure 2d). In the future, it is worth exploring the other aspects of texture editing such as brush stroke's style [14].

References

[1] Adobe. Adobe after effects, 2022. 1

[2] Hussein A Aly and Eric Dubois. Image up-sampling using total-variation regularization with a new observation model. *IEEE Transactions on Image Processing*, 14(10):1647–1659, 2005. 5

[3] CELSYS. Clip studio paints, 2022. 1

[4] Huiwen Chang, Ohad Fried, Yiming Liu, Stephen DiVerdi, and Adam Finkelstein. Palette-based photo recoloring. *ACM Trans. Graph.*, 34(4):139–1, 2015. 8

[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, 3, 5, 6

[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, 5, 6

[7] Junho Cho, Sangdoo Yun, Kyoung Mu Lee, and Jin Young Choi. Palettenet: Image recolorization with given color palette. In *CVPR Workshops*, pages 62–70, 2017. 5

[8] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8188–8197, 2020. 4

[9] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016. 2, 3

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014. 2, 4

[11] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 6

[12] 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

[13] 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. 2

[14] Dmytro Kotovenko, Matthias Wright, Arthur Heimbrecht, and Bjorn Ommer. Rethinking style transfer: From pixels to parameterized brushstrokes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12196–12205, 2021. 8

[15] Tuomas Kynkäänniemi, Tero Karras, Miika Aittala, Timo Aila, and Jaakko Lehtinen. The role of imagenet classes in fr\'echet inception distance. *arXiv preprint arXiv:2203.06026*, 2022. 6

[16] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? In *International conference on machine learning*, pages 3481–3490. PMLR, 2018. 4

[17] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 5

[18] Jue Wang, Yingqing Xu, Heung-Yeung Shum, and Michael F Cohen. Video tooning. In *ACM SIGGRAPH 2004 Papers*, pages 574–583. 2004. 2

[19] Xinrui Wang and Jinze Yu. Learning to cartoonize using white-box cartoon representations. In *CVPR*, pages 8090–8099, 2020. 1, 3, 5, 6

[20] Holger Winnemöller, Jan Eric Kyprianidis, and Sven C Olsen. Xdog: an extended difference-of-gaussians compendium including advanced image stylization. *Computers & Graphics*, 36(6):740–753, 2012. 2

[21] Li Xu, Cewu Lu, Yi Xu, and Jiaya Jia. Image smoothing via l 0 gradient minimization. In *Proceedings of the 2011 SIGGRAPH Asia conference*, pages 1–12, 2011. 2

[22] 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. 6