SPADE-based Line Art Colorization

Applying the existing models to our own problem (Option 1)

https://github.com/Ugness/Line-Art-Colorization-SPADE

Team 3

20160399 Jaehoon Yoo
KAIST
wogns98@kaist.ac.kr

20160479 Seungjoo Lee
KAIST
juicelee@kaist.ac.kr

20160534 Taeckyung Lee
KAIST
terry00123@kaist.ac.kr

20160409 Dongjun Youn
KAIST
f52985@kaist.ac.kr

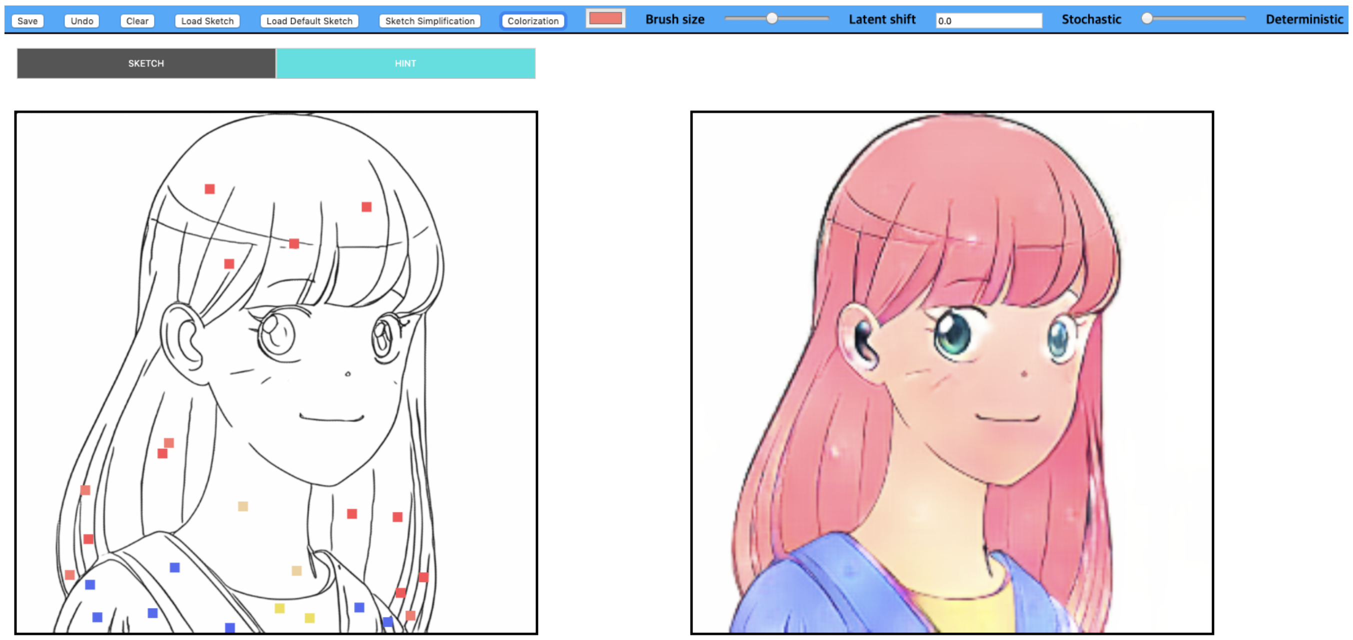

Figure 1. UI of demo application. Left canvas shows line art image and hint patch from the usr, and right canvas is the colorization result.

1 Introduction

SPADE[7] is a semantic image synthesis model that produces photo-realistic image with a given semantic input. It proposed new spatially-adaptive normalization layers since normalization layers tend to wash away semantic information, thus resulting in suboptimal synthesized images. We thought this model can be applicable to colorization since the line art and color hint can work like a semantic map of SPADE, and the model will synthesize the image looks like the illustration. It enables colorizing the line art with user-wanted color. As the property of semantic map is changed, we modified the original SPADE model to better perform on our colorization task.

To this end, we will deliver a SPADE-based model that colorizes a given line art image using a hint image. Instead of the semantic map the SPADE originally used, we use the line art image and hint image that consists of color patches as a semantic input. We also implement a web demo application that allows the users to use our resulting model interactively. Our model and demo can be found on https://github.com/Ugness/Line-Art-Colorization-SPADE

2 Methods

2.1 Dataset Preparing

The dataset we need for training is set of pairs of line image, and color images. In order to create the dataset, we used the following method: *1) Collect colored images, 2) Generate sketch images, and 3) Generate line image.*

Collect color images In order to collect color images, we used Safebooru[4]. Safebooru is an anime and manga picture search engine. The biggest feature of Safebooru is that the images are tag-based. There are various tags that describe the image, and each image is associated with specific tags. In order to create an appropriate dataset for the purpose of line-art coloring, for experiments, we selected images that have tags "white-background", "solo" and "upper-body". There were total 8644 images downloaded. Among them, 100 images were randomly selected and used as a test set. Those images were not included in training set, so that we could recognize the network just memorize the line art and color as it self. For demo, we used images that have tags "white-background", "solo". There were total 98944 images.

Create sketch images After downloading images, we used Sketchkeras[5] to generate rough sketch images from color images. Sketchkeras is a network that generates black and white sketch-style drawing from the given color image. By using it, we could generate corresponding sketch images. By using different denoising filters, we could generate 3 different styles for one colored images.

Create line images Since sketch images generated from Sketchkeras is pencil-styled drawing, it had thin lines and not suitable for training. Therefore, we used sketchSimplification models[8, 9] in order to make lines more digital-styled and ticker. This process was executed for all three different styles generated by sketchKeras. In order to augment the diversity of data, we decided to use all three of those images for dataset. The line art generated this by process would be the final input for our model.

2.2 Training Scheme

Our model is based on SPADE[7], which synthesizes the semantic image using spatially-adaptive normalization. SPADE takes label map, style image, and instance map as an input. To modify SPADE for our needs, we replace the input of SPADE: line/hint image as a label map, and color image as a style image.

Create hint images To generate a hint for each input color image, we used the user-guided hint simulation algorithm[11]. This algorithm extracts color pixels where their position follows Gaussian distribution. The number of hints per image is decided by repeated coin toss simulation. By the result, we obtain a hint image and corresponding mask (indicating the pixel is a valid hint pixel). Hint image (RGB; $3 \times H \times W$) and mask (L; $1 \times H \times W$) are concatenated to generate $4 \times H \times W$ input, where H and W are the same sizes with color and line art images.

As a result, total three types of images are passed to the model for training; *color images*, *line images*, and *hint images*.

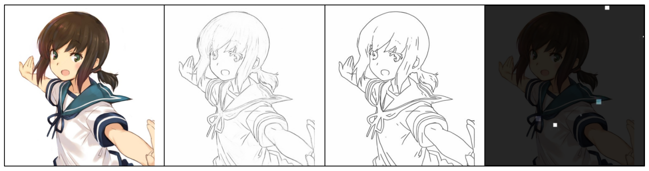

Figure 2. Sample from our dataset. Color image, sketch image, line image (result of sketch simplification on sketch image), and hint image (overlayed on the color image). Among images, sketch image is not used for an input of our model.

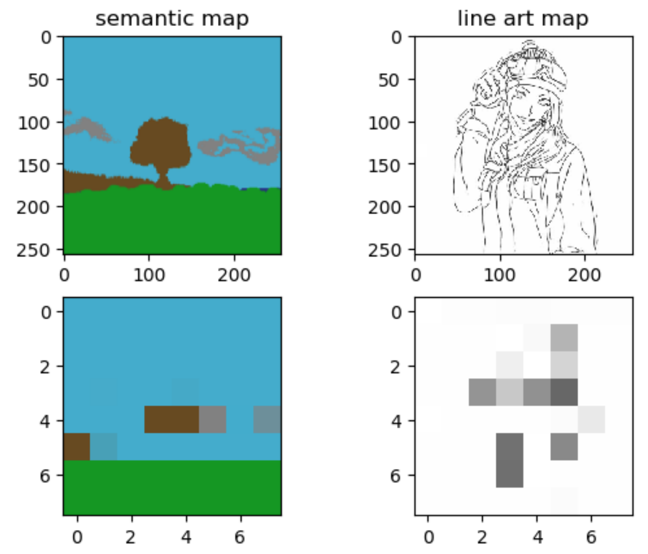

Figure 3. Comparison between a resized semantic label map and a line map. Although the resized semantic label map gives some evidence of contents, the resized line map cannot give any useful information.

2.3 Model Development

The original project with SPADE[7] Module was about semantic image synthesis. Since the semantic map is dense enough, it was good to feed the label maps by down-sampling. However, unlike semantic labels in SPADE, our hint/line art label is too sparse to process them directly. Fig 3 shows the information density between a semantic label map and a line art map when they are down-sampled. Most of the information is lost when we down-sample the hint/line art maps, thus we need to make a reasonable down-sampling way to feed more informative hint maps to the network.

We add LadderNet-like structure(netL) to SPADE to give a hint and line art with a small loss of information. Fig 4 shows our novel network's structure, and Fig 5 shows the results of original SPADE and our netL-SPADE. By providing a hint/line art map with an additional feature encoder, SPADE Module could get "semantic map-like" features.

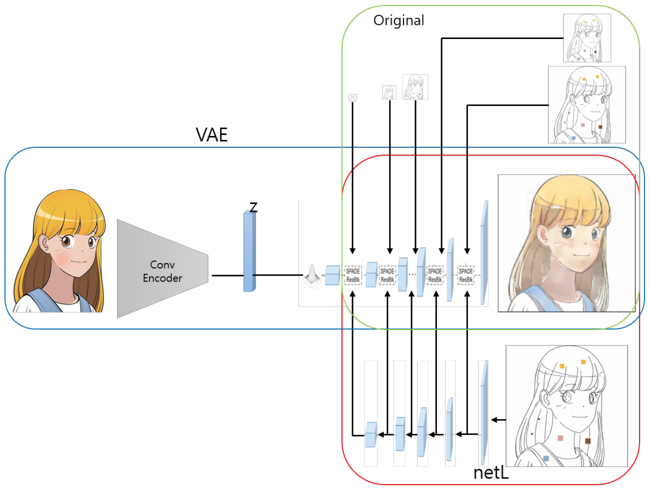

Figure 4. Comparison between original SPADE and Our netL model. We use netL module to feed hint and line art while the original SPADE feed them with down-sampling.

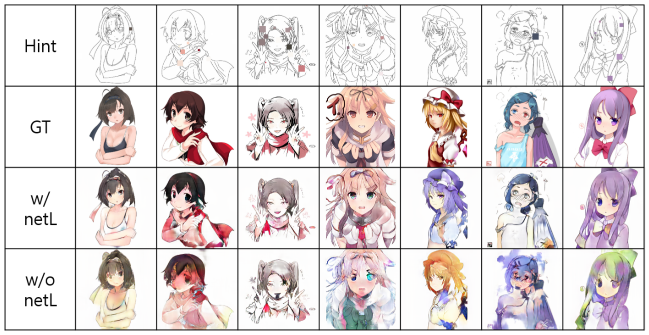

Figure 5. Comparison between result of original SPADE and Our netL model. The model with netL preserves more high level features like eyes or face components than the original model.

2.4 Implementation Details

To achieve high performance on the line art colorization, following techniques are applied in addition to original SPADE module: *1) reflection padding, 2) using Danbooru pre-trained ResNet, and 3) HSV augmentation and increased hint patch size*

Reflection padding Instead of using a zero-padding option, we used reflection padding in all convolutional layers to preserve sparse information. Unlike the semantic label or real picture images, line art has a lot of sparsity. Almost 90% of the image is white, and only a little proportion of black lines has all the information. And also, zero value in normalized color space means gray color. It makes the image to have a gray letterbox, not a white. This gray letterbox can interrupt the model to learn correctly. So we applied reflection padding instead of zero-padding, fit better in our environment.

Using Danbooru pre-trained ResNet We used Danbooru pre-trained ResNet[1] instead of ImageNet pre-trained VGG19 network for perceptual loss[2]. The color distribution between real-images and illustrations are too different. Since the original VGG Network is from ImageNet Dataset, it is hard to get meaningful features from illustrations. By exchange the network, we could get a more accurate feature comparison between fake and real color illustrations.

HSV augmentation and increased hint patch size We increased the size of hint patches and used HSV data augmentation to avoid model to overfit. Without color augmentation, a model can memorize the color image corresponding to the line art. But the colorization problem is a multi-modal problem. So we added HSV data augmentation to model to solve the multi-modal problem.

We used Adam optimizers with a learning rate of 0.0001 for the generator and 0.0004 for the discriminator[3]. Both optimizers used beta1 of 0, beta2 of 0.9. We used a cosine annealing and warmup learning rate scheduler[6]. We trained our network with 256*256 images for 50 epochs with a batch size of 32. It took about 12 hours with 4 RTX TITAN GPUs when we trained our model on the upper-body dataset containing about 8600 images.

2.5 Demo

To show our results in a interactive way, we implemented a demo web page with *HTML/CSS* frontend and *Flask* backend. As shown in Fig 1, the web page consists of color canvas with brushes, sketch loading, sketch simplification, colorization, and latent shift control.

Frontend User can load existing sketches from local file and default sketch, or draw their own sketch on the canvas using brushes. Then, apply sketch simplification to convert the sketch to a line image. With the line image, make hint pixels above line image layer to generate hint image. After generating line image and hint image, press colorization button to get the colorized image on the right canvas. Latent shift can be applied to generate different colored image as explained above.

Backend For the fast response, the backend server loads *sketchKeras*[5] model and our colorization model on a GPU when it boots. When the user send base64-encoded image for the simplification and colorization by *HTTP Post*, the server invokes appropriate model and sends the result image with *HTTP Reply*.

3 Conclusion And Discussion

We had some analysis on the model's understanding of line arts and hints.

First, we plot all possible images from different latent vectors. As shown in Fig 6, each dimension of latent space represents different color domains. For example, the right-uppermost image is from a latent vector which has a value of -100 in the first dimension, and the others are all zero.

Second, we want to find how much the quality of the sketch/line effects on the model. From Fig 7(a), we could conclude that some of the color hints were remaining as perturbations. Each row in Fig 7(a) means line art input and corresponding colorized results. (1) gets line art from the original image directly. (2) just erased the horizontal background lines from (1). (3) gets line from simplifying the line of (2). Since the line art of (2) had undesired color hints, the model could recognize that there was a color background. From these symptoms, we find out that there are undesired color hints in synthetic line arts.

Third, our model was too vulnerable with the thickness of line art. Fig 7(b) shows how the result changes with different line thicknesses. The left line art is from inference *sketchKeras*[5] with a size of 768*768, The right one is with a size of 512*512. Both line arts are fed to the colorization model with 256*256 using the nearest interpolation. Although both line art looks similar to the naked eye, the painted results are too different. The result from the size of 512 seems like it failed to recognize face parts.

From the above analysis, we suggest some methods which will improve the model's performance. First, feed the line art as a binary map. We trained our model with a line art that has 256 possible values. By using the line art as a gray image, the model was able to learn undesired color hints from the line art's perturbations. Second, prepare more diverse line arts. As we used the lines from a single sketch simplification model, our model works only on lines that are from the simplification model. There is another sketch simplification algorithm, Xdog[10]. And also, you can prepare line art with different line thickness by applying the images to the simplification model with different sizes. We hope the above methods will help you to get a more expert AI painter.

4 Contributions

- 20160399 Jaehoon Yoo
  - Download and parse the Safebooru dataset[4].
  - Imported Sketchkeras[5] and get sketch images from color images.
  - Colorization-SPADE Lnet and other method experiments.
  - Deploy model on our demo application.
- 20160479 Seungjoo Lee
  - Imported and changed SPADE[7] module to fit our own dataloader and input.
  - Implemented backend part of the demo application, and some of the frontend part.
- 20160534 Taeckyung Lee
  - Implemented Safebooru dataloader with hint generation based on colorization-pytorch[11].
  - Implemented frontend part of the demo application.
- 20160409 Dongjun Youn
  - Imported and changed SketchSimplification[8][9] to generate line-style drawing.
  - Prepared train set and test set of images.

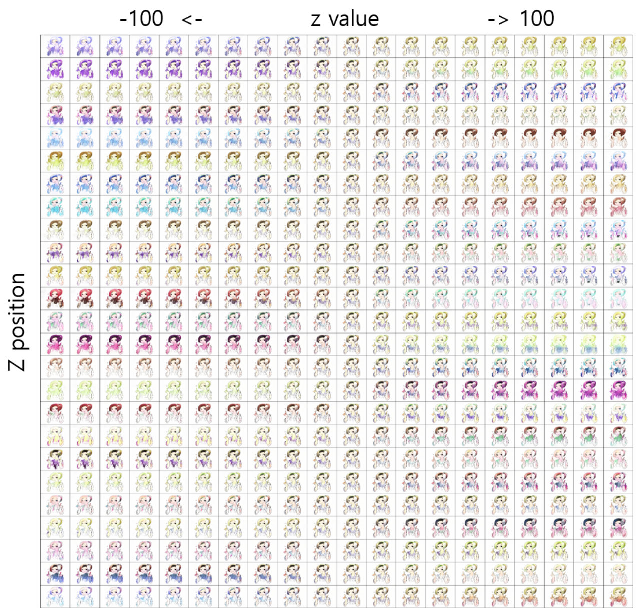

Figure 6. Painted Images from different latent vectors. No hint for inference.

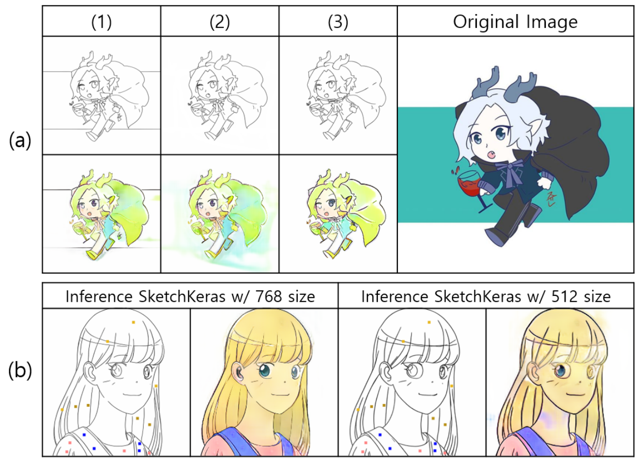
(1)
(2)
(3)
Original Image
(a)
Inference SketchKeras w/ 768 size
Inference SketchKeras w/ 512 size
(b)

Figure 7. *(a)* shows the effect of undesired color hint in the perturbations, *(b)* shows results from different line art thickness.

References

[1] Matthew Baas. 2019. Danbooru2018 pretrained resnet models for PyTorch. https://rf5.github.io. https://rf5.github.io/2019/07/08/danbuuro-

pretrained.html Accessed: 2019-11-27.

[2] L. A. Gatys, A. S. Ecker, and M. Bethge. 2016. Image Style Transfer Using Convolutional Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2414–2423. https://doi.org/10.1109/CVPR.2016.265

[3] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014).

[4] Alex Lamson. 2019. https://www.kaggle.com/alamson/safebooru Accessed: 2019-11-27.

[5] lllyasviel. 2017. https://github.com/lllyasviel/sketchKeras Accessed: 2019-11-27.

[6] Ilya Loshchilov and Frank Hutter. 2016. SGDR: Stochastic Gradient Descent with Restarts. *CoRR* abs/1608.03983 (2016). arXiv:1608.03983 http://arxiv.org/abs/1608.03983

[7] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. 2019. Semantic Image Synthesis with Spatially-Adaptive Normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

[8] Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. 2018. Mastering Sketching: Adversarial Augmentation for Structured Prediction. *ACM Transactions on Graphics (TOG)* 37, 1 (2018).

[9] Edgar Simo-Serra, Satoshi Iizuka, Kazuma Sasaki, and Hiroshi Ishikawa. 2016. Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup. *ACM Transactions on Graphics (SIGGRAPH)* 35, 4 (2016).

[10] Holger Winnemöller. 2011. XDoG: Advanced Image Stylization with eXtended Difference-of-Gaussians. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering (NPAR '11)*. ACM, New York, NY, USA, 147–156. https://doi.org/10.1145/2024676.2024700

[11] Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, and Alexei A. Efros. 2017. Real-Time User-Guided Image Colorization with Learned Deep Priors. *CoRR* abs/1705.02999 (2017). arXiv:1705.02999 http://arxiv.org/abs/1705.02999