CS230

2-Stage Conditional GAN for Sketch Auto-Coloring

Yu-shun Cheng
Stanford Center for Professional Development
Stanford University
ysc270@stanford.edu

Abstract

I show in this project how GAN can automate the image coloring process in Manga style, based merely on line-art sketches and some color hints. In the end not only do I have a promising model but also an interactive tool that users can play with.

1 Introduction

The viability of automatic sketch coloring was brought to my attention when I was attending Comiket 93 in Tokyo and trying out the PaintsChainer service at Preferred Networks booth. Since then, other work has been done on the same problem with varying results.
Optimally the coloring algorithm should take only a grayscale image as input and output an RGB image, however, due to limited scene understanding and the importance of knowing user intent, we have to redefine the input as a grayscale image along with some sort of color hints.
In my implementation the color hints take the form of an RGB image, hence the model is essentially a mapping from 4 to 3 channels, while maintaining the same width and height.

2 Related work

The best publicly known commercial work was demonstrated by Preferred Networks, Inc. (2017), however since it does not disclose any of the algorithms it uses I only reference it later for comparison. Some very similar work was done around the same time by Frans (2017) and Liu et al. (2017), which then led to a CS229 student project by Fu et al. (2017). The common approach they all take coincidentally is to apply a Gaussian filter on the original image to get the hint image, then feed that together with the edge detected sketch into GAN to generate the colored image. One of the common traits of them is that they seem to generate images with lots of artifacts.
My work is highly inspired by Zhang and Li (2018), which uses a 2nd stage GAN to further refine the image proposed by the 1st stage. This technique seems to solve the aforementioned issue quite well. Following this approach, I also adopt a refinement model, though in a much simpler fashion, in terms of both network architecture and data preparation. This enables me to get a far smaller model, while still being able to show better results than the previous generations.

3 Dataset and Features

I use a subset(70GB) of the freely available Danbooru 2017 512px for training, which is a downscaled version of all the latest fan art uploaded by viewers. Some of the images are already in 8-bit grayscale format that lack color information, which I have run my script to remove.
During training, for each image, a randomly cropped 256 x 256 region is selected as a training example to reduce overfitting.To artificially create the sketch images, I use the naive edge detector

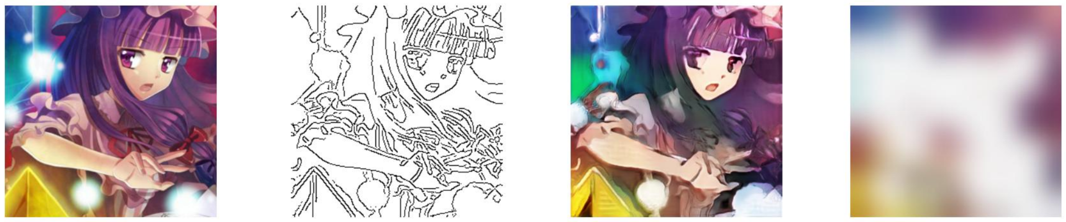

(a) Original (b) Sketch (c) Draft (d) Hint

Figure 1: Dataset for training the refinement GAN

provided by PaintsChainer as recommended by Zhang and Li (2018). For the hint images, 30 randomly selected 45 x 45 patches are whitened out and blurred with 115 x 115 Gaussian kernel, to reduce model reliance on full and high quality hints.
Instead of using the data synthesis method proposed by Zhang and Li (2018) for training the refinement stage GAN, which can come from a completely different distribution than the actual data, I use the half-way trained draft generator to create training data for the refinement model.
For simplicity and speed, we save out 256 x 256 image bundles comprising original 1a, sketch 1b, draft 1c, and hint 1dimages, to the disk before we train our refinement model.

4 Methods

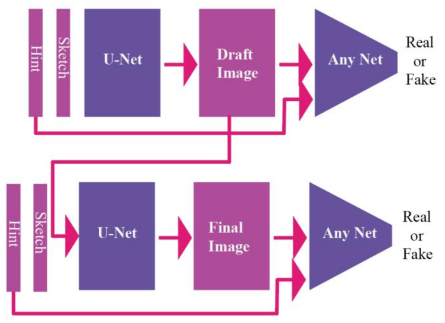
The CGAN loss is defined as:

$$\mathcal{L}(D) = -\mathbb{E}_{x,y}[\log D(y|x)] - \mathbb{E}_{x,z}[\log(1 - D(G(z|x)|x))]$$
$$\mathcal{L}(G) = -\lambda\mathbb{E}_{x,z}[\log(D(G(z|x)|x))] + \mathbb{E}_{x,y,z}[||y - G(z|x)||_1]$$

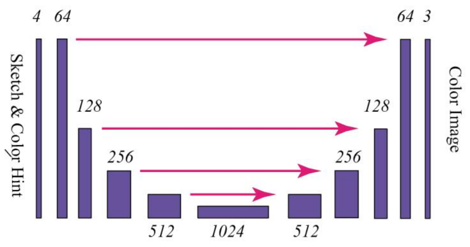
Where x is the color hint, z the sketch, and y our ground truth, which is the original image. I choose L1 loss for the generator because it gives me a less blurry image than L2. λ term here is necessary for tuning the ratio between adversarial loss and pixel value loss, which is chosen as 0.01 by default.
For the generator I use U-Net2b from Ronneberger et al. (2015), which has quickly become the standard for image segmentation, and is ideal for this type of work. I would argue it does not really matter what model is used for discriminator, as long as it is able to classify an image. Unless it has a hard time reducing the loss, no further complication is required. Therefore I manually built a very simple network made up of [conv2, batchnorm, relu] blocks with a stride of 2, and never had to look back.
Even though the original implementation from Zhang and Li (2018) injects the output from another pre-trained Inception network into the middle of the U-Net-like model, I didn't find it particularly useful, especially since a single pixel gets expanded to 16 x 16 for concatenation, which could explain why the extra information got grounded(ignored) by the network in one of my experiments.
The resulting architecture 2a is quite simple, with the same GAN applied twice for both drafting and refinement stages, which can be trained separately in parallel. The only difference is that instead of doing a 4-to-3 mapping in stage 1, stage 2 is a mapping of 7-to-3.

5 Experiments

Initially I did my best to map the architecture in Zhang and Li (2018) as closely as possible, by using the transposed convolution layer with a stride of 2, however this resulted in images with a very visible checkerboard pattern 3a. Changing it to upsample and convolution blocks introduces yet another issue, the color leak 3b. Fortunately after adopting the popular U-Net model these issues disappeared.
As pointed out by many sources, noise plays an important role in training GAN, especially in the beginning, due to the fact that the discriminator's work is always easier than the generator's. I found noise via label flipping works the best. Assuming we use a noise rate of 10%, for real images, there is a 10% chance for the label to be 0, while for fake images, there is also a 10% chance for the label to be 1. Handicapping the discriminator allows the generator to learn from its mistakes, before the discriminator gets too clever. It remains to be seen what the lowest noise level could be, but I used 5% and 10% with success. This phenomenon can be observed in 4a, where I removed the noise

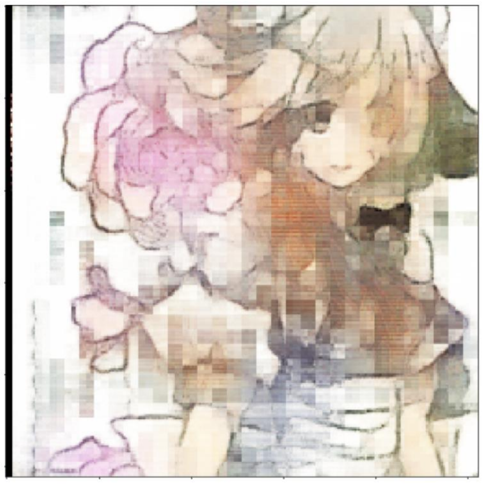


(a) An overview

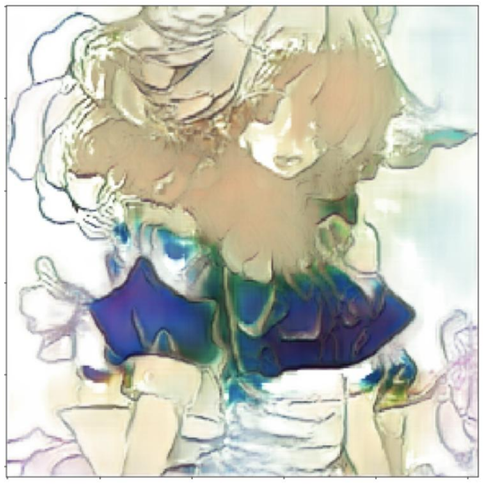


(b) Simplified U-Net

Figure 2: The architecture



(a) Checkerboard pattern



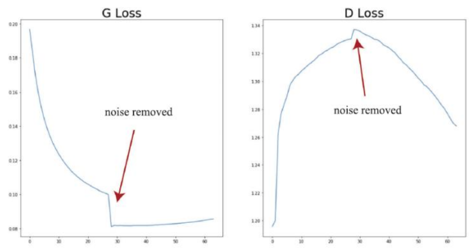
(b) Color leak

Figure 3: Initial failed experiments

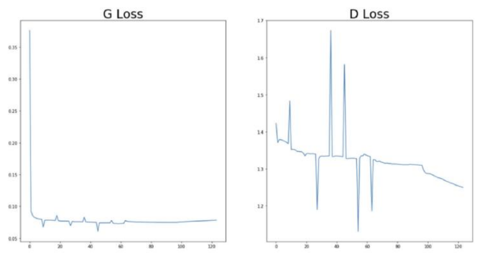
completely from training, even though the discriminator loss dropped really fast, the generator's quality worsened over time.

6 Results/Discussion

In the end, the model is able to color most images in a convincing way, if the color hint is well formed, even with the very naive line extraction algorithm to get the synthetic sketch. The green tint color drift seen in all 3 draft images are removed nicely in the final images, as well as the irregular lines



(a) Draft training curve w noise removal



(b) Refinement training curve

Figure 4: Training curves

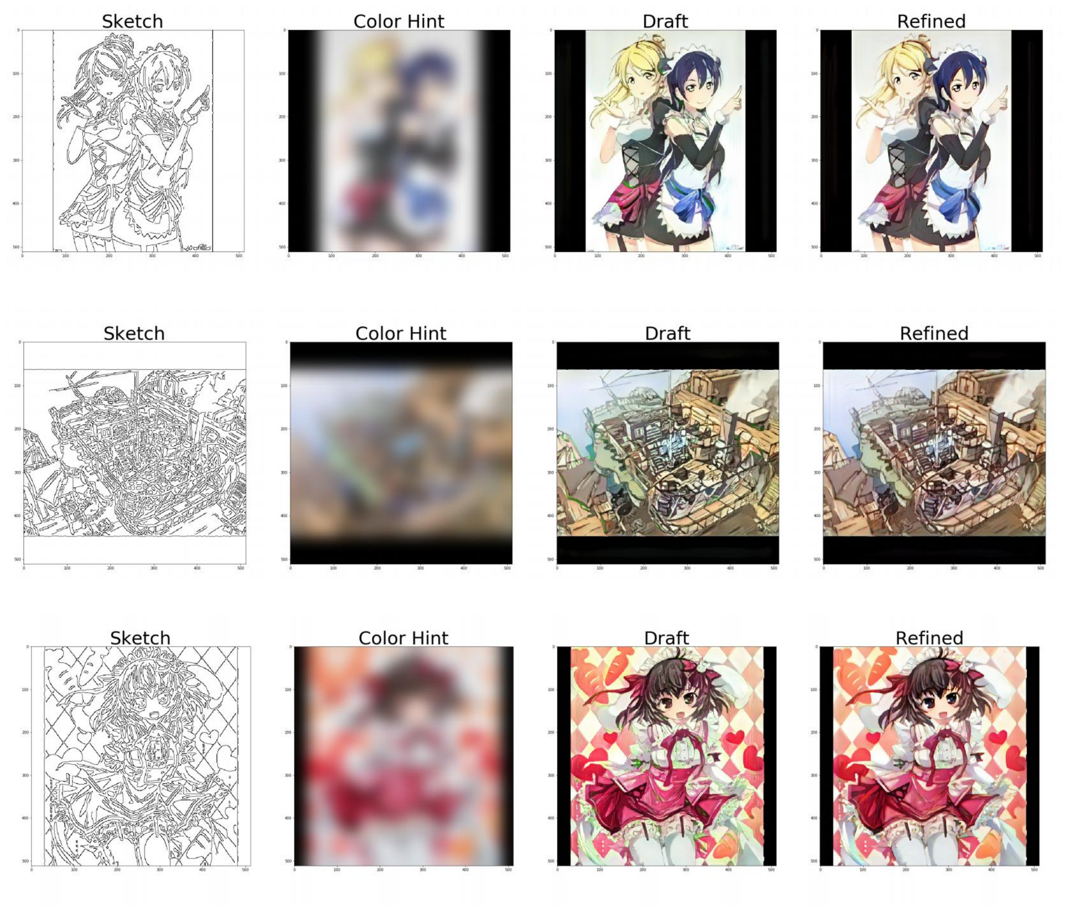
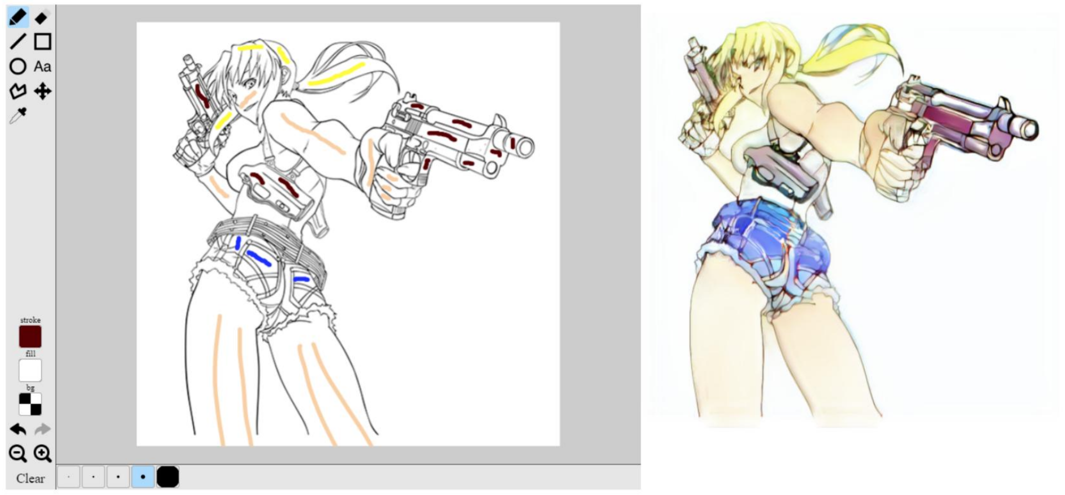


Figure 5: Example results

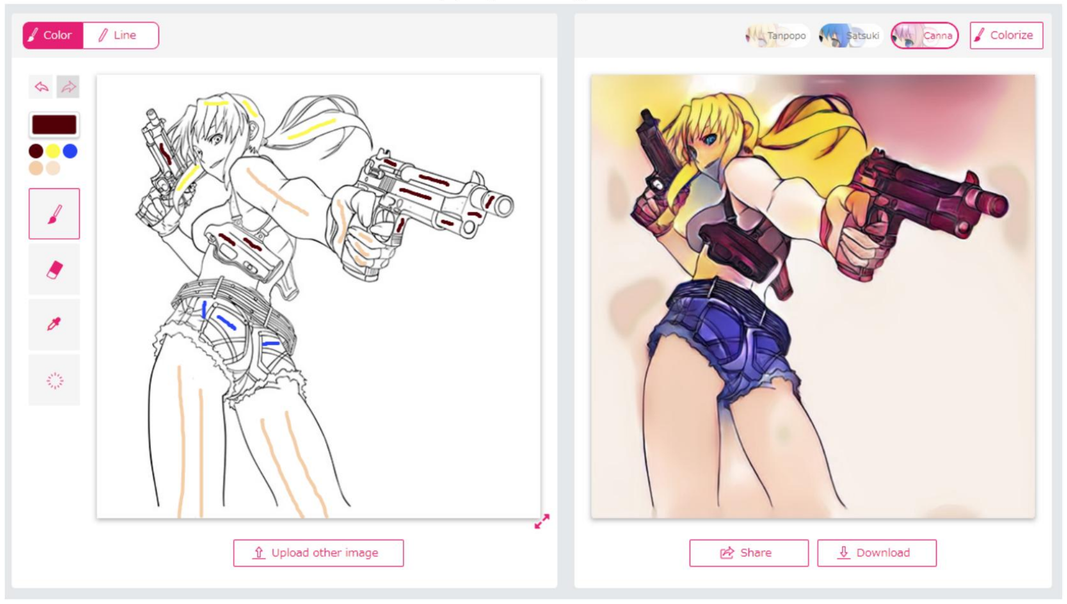
and the minor checkerboard pattern, even if the training data is generated from earlier epochs.
For a better demo experience, I also quickly pieced together an open source canvas tool made by Landey (2016) and my coloring engine, which enables me to compare my results 6a with PaintsChainer's 6b.
It's hard to say which one does a better job of coloring, since this is a highly subjective matter, but we can still observe that the one from PaintsChainer aggressively increases saturation, while suffering from color leakage, in comparison our model selects a less saturated coloring scheme, while being more careful about which blank regions should be colored.

7 Conclusion/Future Work

The 2-stage colorization idea coined by Zhang and Li (2018) seems like a brilliant one, even though implementation-wise there is a lot more work to do. Due to the time constraint I had(3 weeks), I didn't have the luxury to explore more variations. Nevertheless, there are a few things for sure that I can do to improve both the model and workflow.
First, the synthesized color hint computed from the original image plays a critical role in how users interact with the tool, in my approach and many others', randomly selected regions are erased, followed by a Gaussian blur. This kind of heuristic algorithm may not match how users actually paint colors on the sketch(e.g. thin lines get diluted), hence we need to either change how hints are generated or how the tool works.
Naive edge detection works okay if the line art is not too complicated, however as more and more details are added, some may get lost during conversion. I also didn't put back the positive regulation loss when training the final model, as it made it harder for my previous models to converge. Given



(a) My coloring tool



(b) PaintsChainer

Figure 6: Comparison

more time, however, it is a fairly effortless way to boost the overall saturation.
To sum up, I successfully showed the usefulness of the progressive 2-stage GAN architecture, by reusing the existing network, which I hope will be adopted for other applications as well. Then I also deployed the model to AWS and hacked together a tool to demonstrate how to use it for real work.

8 Source code

For interested readers, the complete source code can be found at https://github.com/talentjp/irodoru You are more than welcome to contribute to the code base.

References

Frans, K. (2017). Deepcolor. `https://github.com/kvfrans/deepcolor`.

Fu, K. I., Wang, Y., and Liu, B. (2017). Cs 229 final project automatic colorization for line arts.

Landey, S. (2016). Literally canvas. `http://literallycanvas.com`.

Liu, Y., Qin, Z., Luo, Z., and Wang, H. (2017). Auto-painter: Cartoon image generation from sketch by using conditional generative adversarial networks. *CoRR*, abs/1705.01908.

Preferred Networks, Inc. (2017). PaintsChainer. `https://paintschainer.preferred.tech`.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*.

Zhang, L. and Li, C. (2018). Two-stage sketch colorization.