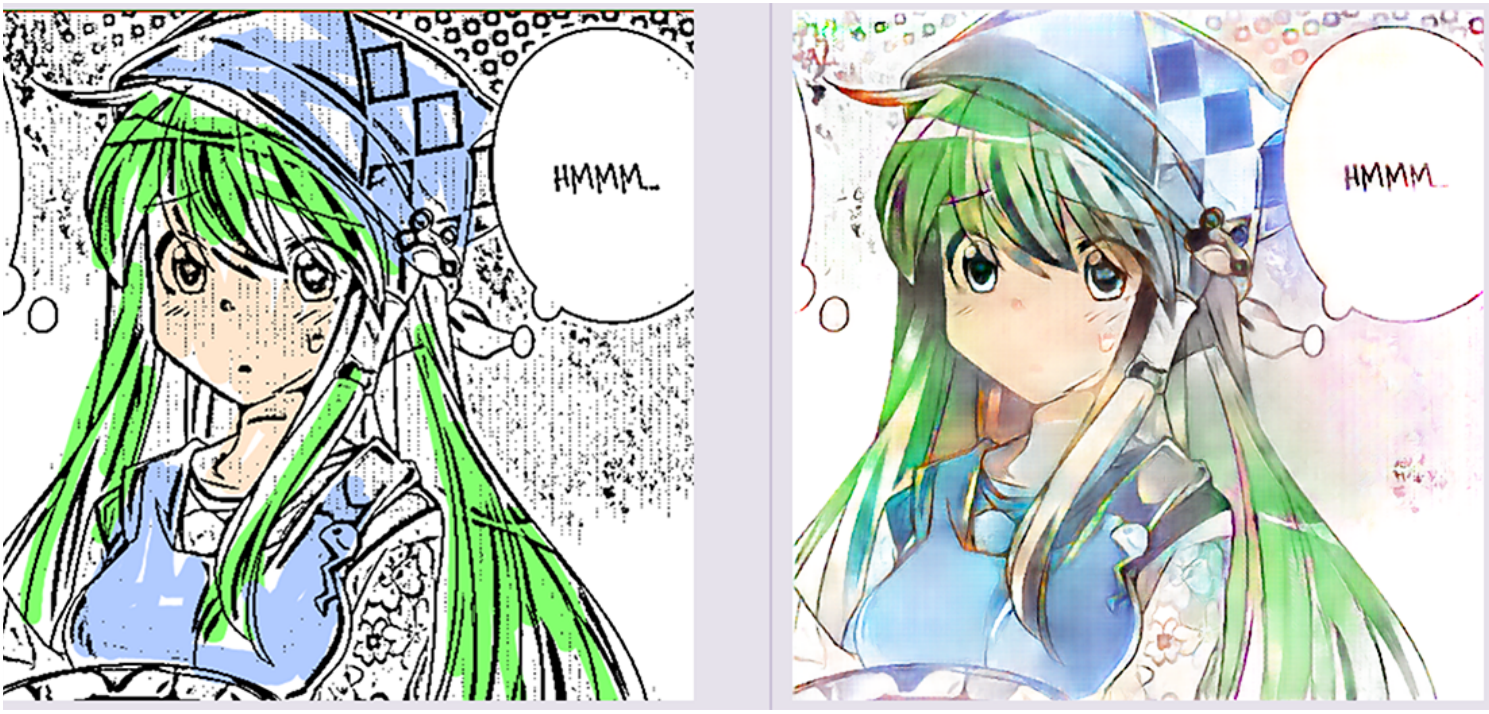
MENU

Deepcolor: automatic coloring and shading of manga-style lineart

01 MARCH 2017



In the past few months, I've been experimenting with digital art. The process involves creating a rough sketch, cleaning it up into lineart, and then finally coloring and shading to create a final product.

Out of these three steps, adding color is by far the most time consuming. So let's get the computer to do it for us! In this post I'll go over the methods behind deepcolor: a network that automatically finishes a given lineart image.

Note: this post will assume a basic understanding of convolutional neural networks. For more on how they work, check this post or this chapter of the deeplearning book.

> *As a teaser, try the deepcolor demo here!*

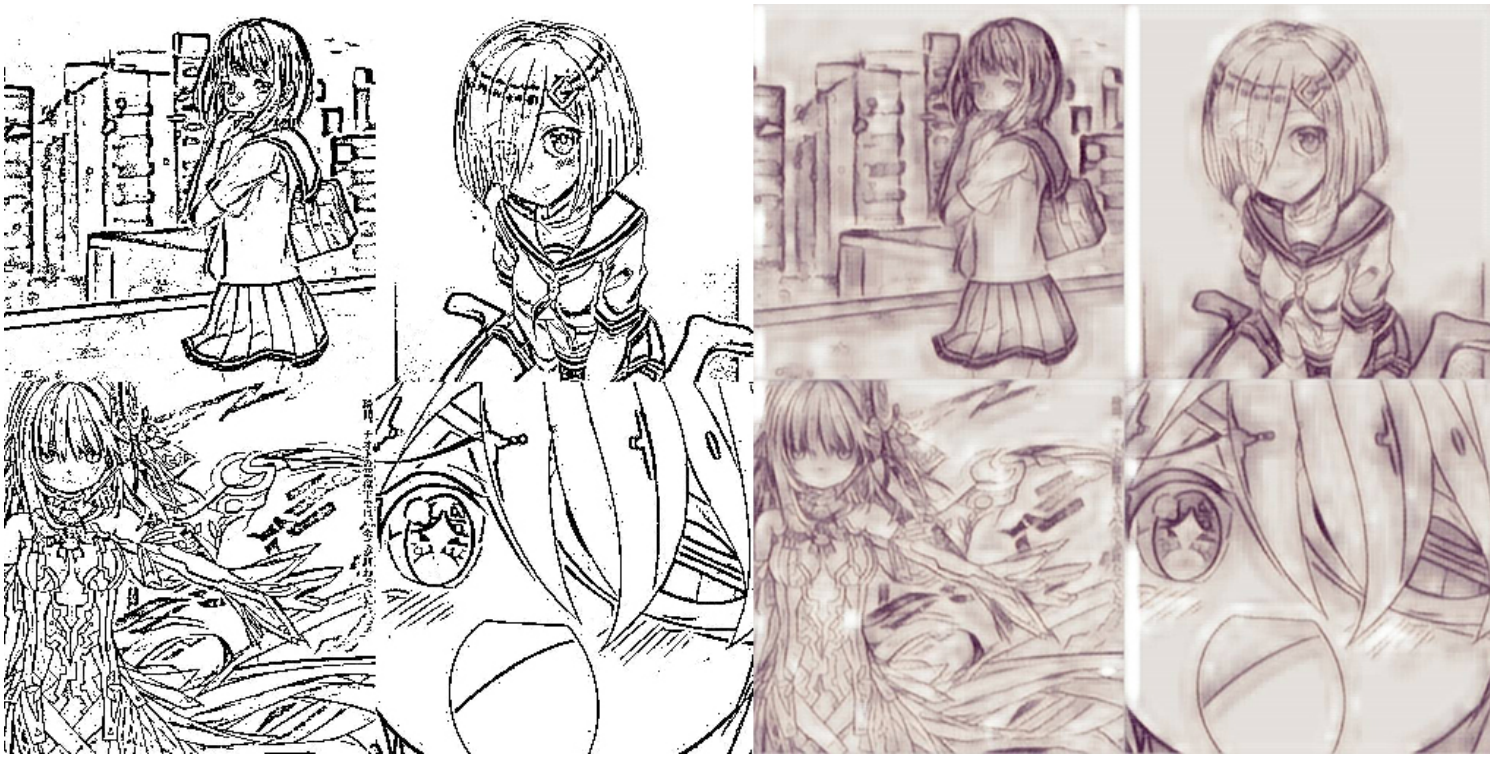
Starting points

First, let's take a look at what other people had already made to get a sense of where to start. Convolutional Sketch Inversion uses deep convolutional networks to directly map from input sketch images to a photorealistic face image. Colorful Image Colorization uses the same methods to colorize grayscale photographs.

This seems like the most straightforward approach. A simple convolutional network could potentially solve the entire problem, so let's start there.

I set up a model mapping from a 256x256x1 line-art image to its 256x256x3 colored image, using an L2 loss.

Unfortunately, recreating color from only lines is a hard task. The network completely failed in inserting color at all.



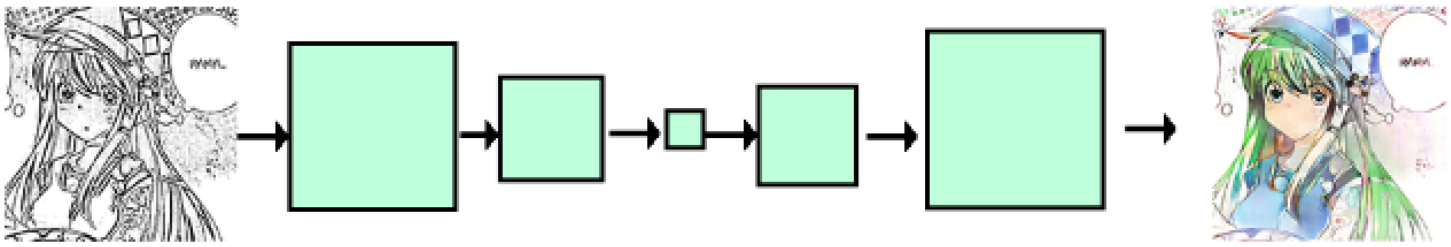
> *Not really what we're looking for, although there is a nice sketch-like effect.*

There's a few things we can do from here to make our outputs better.

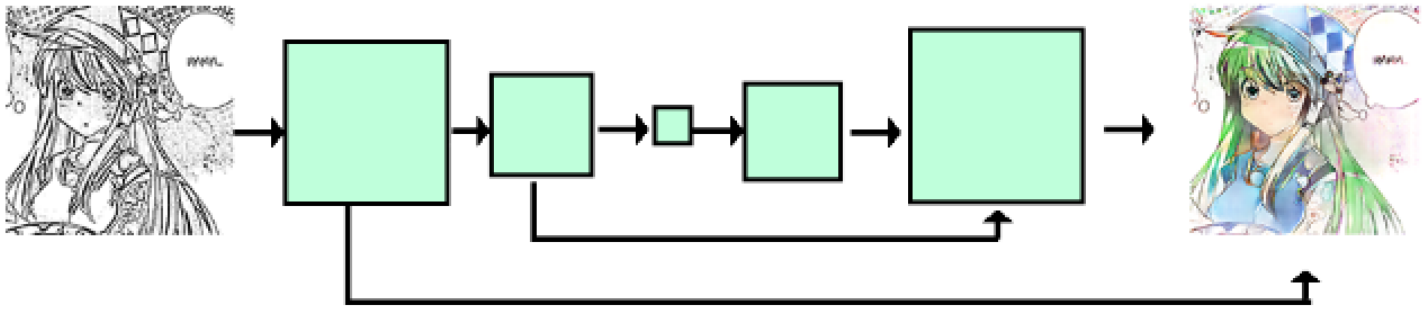
L2 loss is known to create blurry image generations. Instead, we can use an adversarial loss, allowing the network to learn its own loss function. Essentially, we'll transform our model into a generative adversarial network, except rather than creating images from random noise, we'll create them based on the line art.

Additionally, we'll use residual connections between each convolutional layer. This is a fancy way of saying that during generation, convolutional layers will operate not only on the previous layer, but also on same-dimension layers from before.

It's hard to describe in words, so a visual may help clear things up.



> *Network without residual connections. Each convolutional layer (green square) has a stride of two, so the dimensions shrink in half after each operation. Transpose convolutions (sometimes referred to as deconvolutions) double the dimensions.*



> *Network with residual connections.*

What residual connections do is allow the network to have a lengthy pathway for heavy processing (such as recognizing facial features), while still having references to the original input. In our case, the lineart itself is often present in the final drawings, so we want a way for this information to be passed through without going through every layer.

Check out this Torch post for more on residual connections.

I implemented the changes, and trained the network once more. How well did we do?

> *[original, lineart, recreation] using GAN + residual connections.*

It's definitely an improvement! We've got actual color this time. At least for the top example, the network learned the color of skin. Of course, it colored in the hair and shirt incorrectly, since the network has no idea what the original colors were.

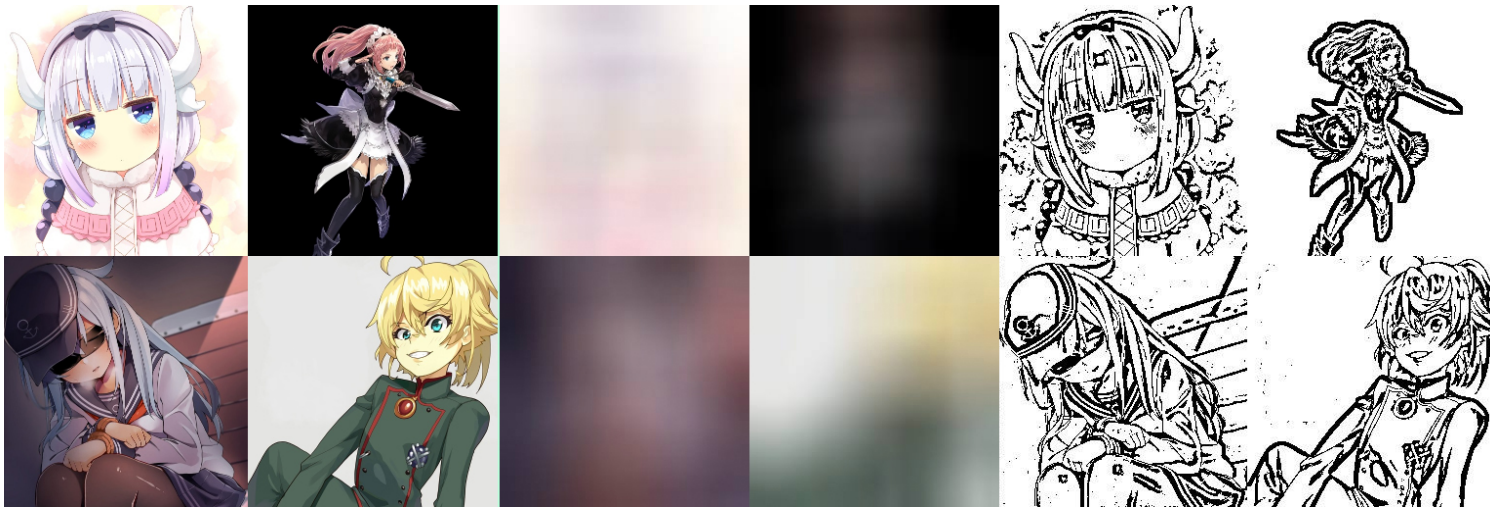
In the second example, we're left with a big mess. There are some hints of success (the hair and hat are black!), but mostly the colors are all over the place.

At this point, there's really not much to change in terms of the network. We have something that's *kind of* correct, and we can work with that.

What's causing the network to have such a mess of colors everywhere? It's the probably most confused when dealing with objects that could be any color, such as clothes.

To combat this, let's give the network a "color hint". In addition to the lineart, we'll give the network another image containing the colors of the original image.

Of course, if we give the network every single pixel color, it wouldn't have to do anything at all to recreate it. To combat this, we'll apply a large 100px radius blur, and pass that as our color hint.



original image, color hint, line art

The network now takes in both the blurred image, as well as the line art. Looking at the above image, we can see how helpful the color hints are. Without the color hint, the network would have no idea whether to make the background white, black, or some other color.

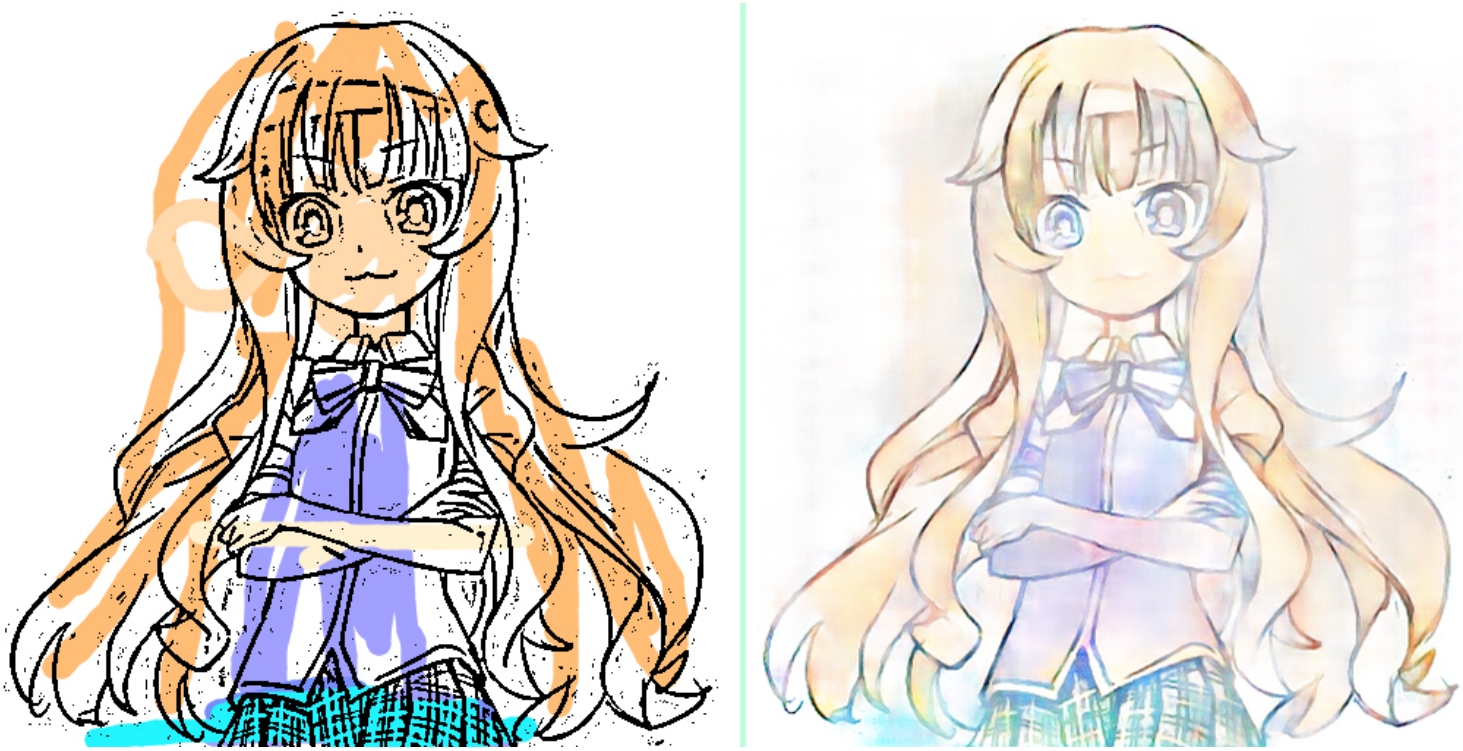
How well did it work?

generated images

It's a big success! The generated images look natural, and match the originals closely.

Of course, this comes at a price. We're now dependent on the color hints to generate images. It's important to take a step back, and realize that our dataset is not actual linearts and sketches, but a simulated set created by processing the original image.

I decided to attempt and use my own (intentionally messy) coloring as a color hint, rather than the original image, and see what happened.



left: lineart + my coloring. right: generated image.

The results were unimpressive. At the areas where I missed colors, the network failed to color in the hair. Sure we've got incredibly accurate recreations, but using a bad color hint will

still lead to a bad generation. If we need the original image in order to generate the original image, we really haven't solved anything.

To really drive the point home, here's what happens if we pass in no color at all.

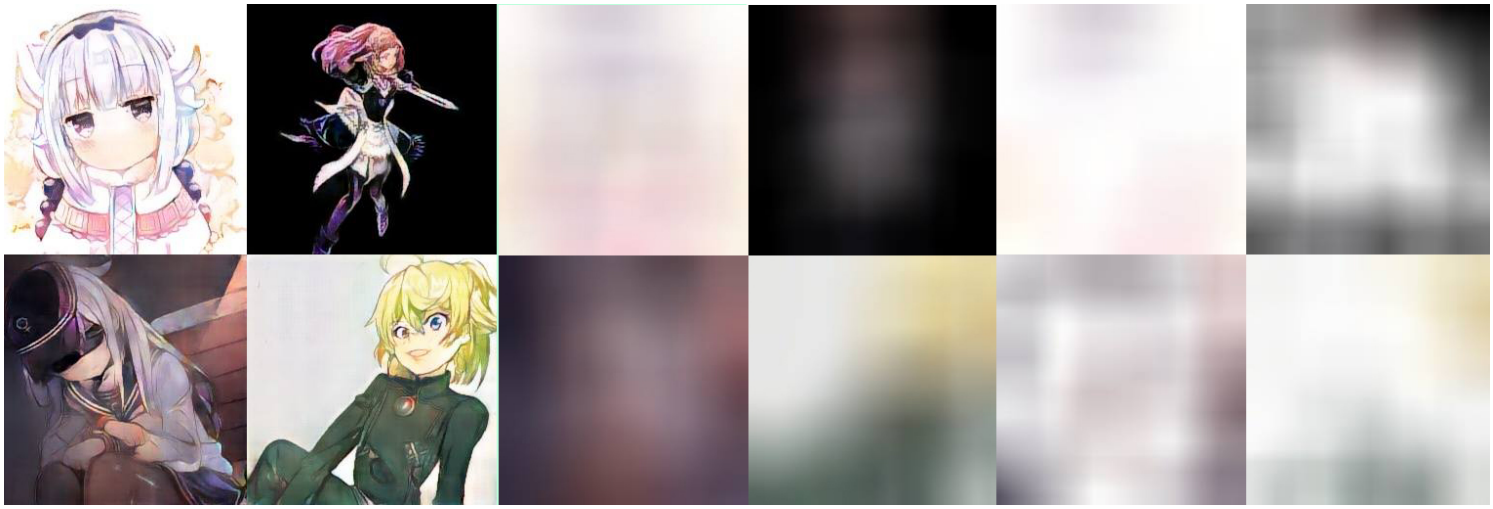
The network's not doing anything with the lineart. Almost all the details in the recreations came from the color hints.

It's at this point that we have to make a compromise. Clearly, the color hint is vital to generating good images, and performs well on simulated color hints. However, we can't realistically expect users to completely color in their lineart themselves and call that a "hint".

To remedy this, we need to reduce the network's dependence on the color hint. Most of the information in the final image should come from the line art, with the color hint as a secondary.

How should we go about doing this? Let's restrict the amount of information the color hint can contain. The blur was supposed to act as an information barrier, but apparently it isn't doing a good enough job.

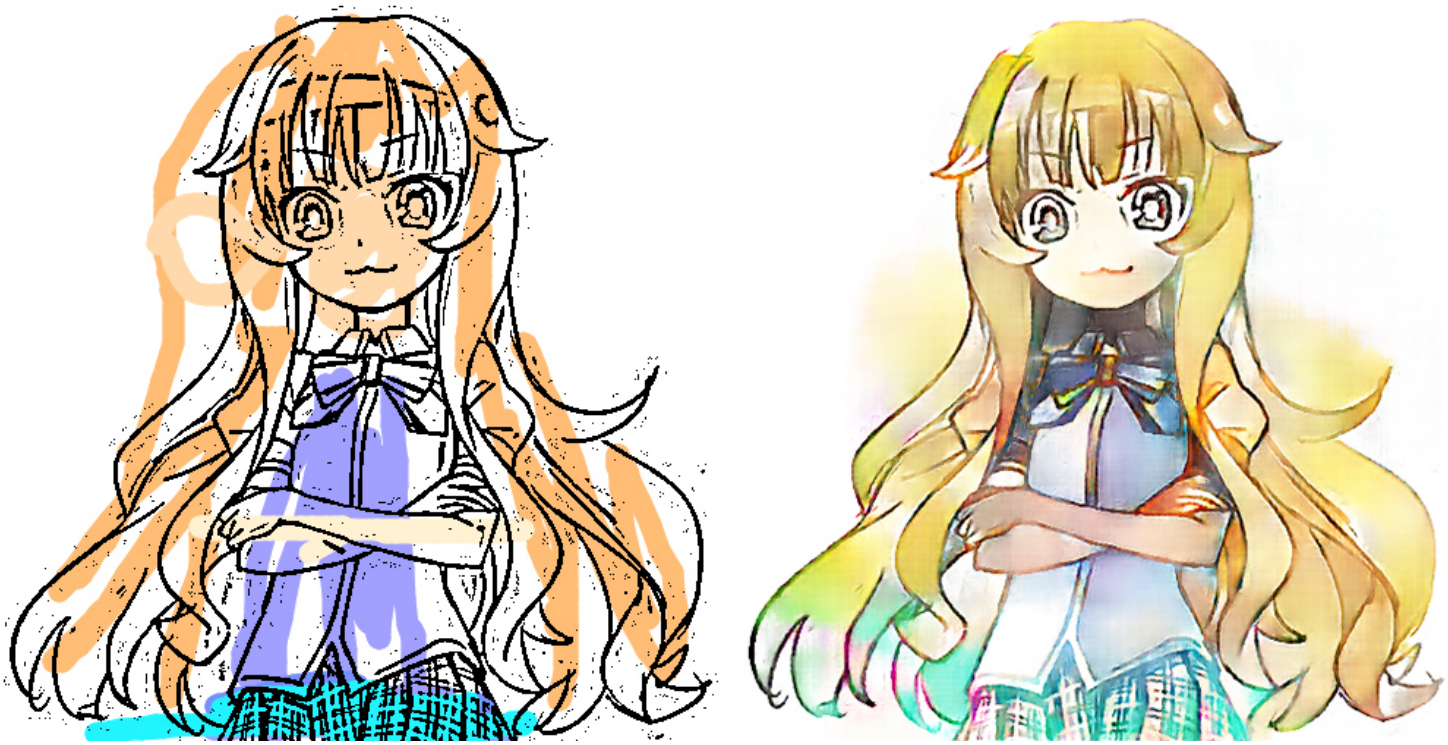
Instead, we'll use a somewhat crude but straightforward method to make the color hints less useful. Take the original image, and randomly white out a 10x10 area of the image. Do this 30 times, then we will apply the same blur as before.



[original, only blur, whiteout + blur]

Noticable in the image above, the new form of color hint is missing lots of information. Now, the network is forced to guess what colors should go there, based only on the little color hints it's receiving.

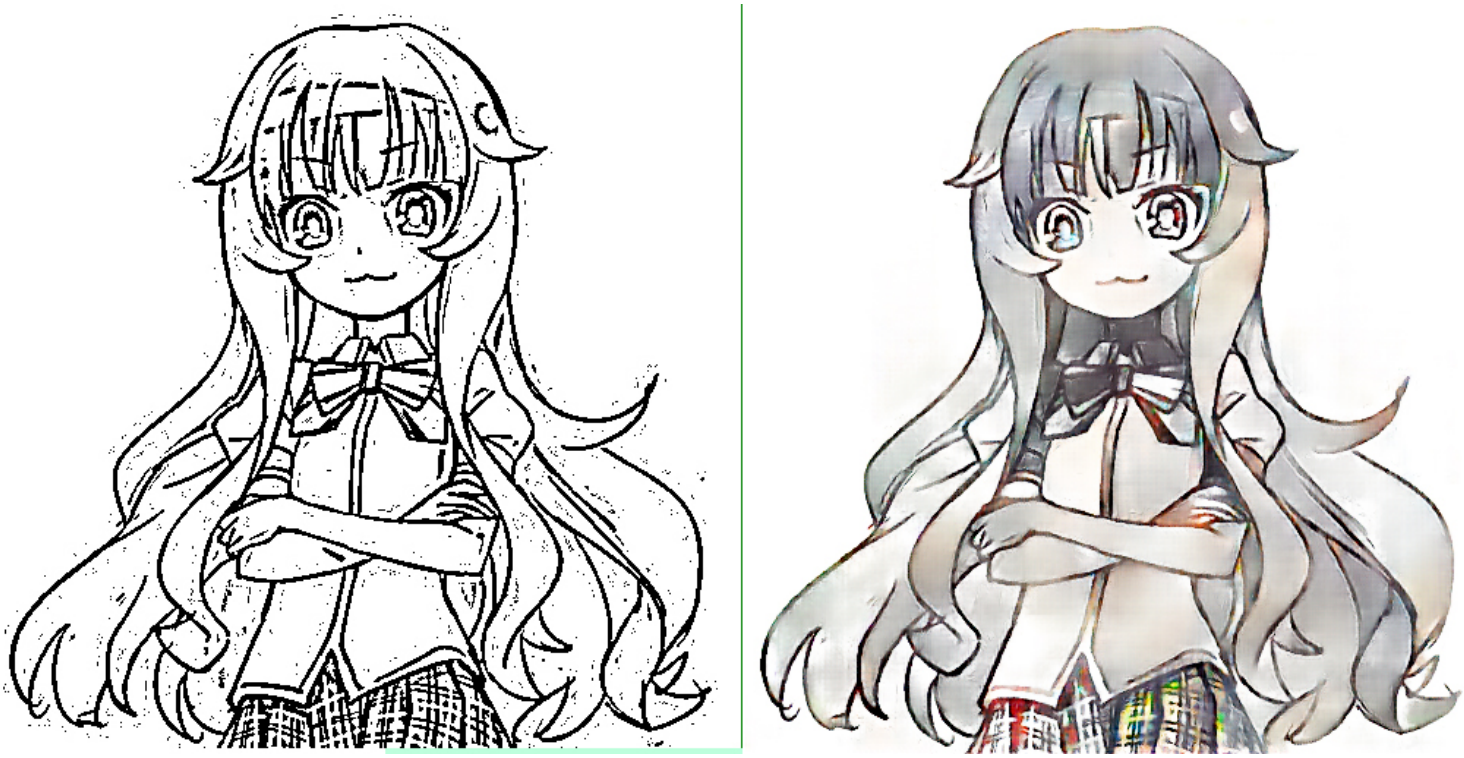
I trained the model once again, using the new color hints.



This, time we've performed much better! Even though I missed many spots in the coloring, the network still realizes what I meant to do, and fills colors in accordingly. This makes sense: since we trained the network with holes in the color hints, it had to learn to account for them.

Note: When training, we randomly removed parts of the color hint, to force the network to learn better. When we're actually using the network, we want as much detail as possible, so we don't remove anything, Instead, we scale down every color intensity by 0.3 (expected value).

Now, let's take a look at what happens if we pass in no color at all.



This time, we can see the network *has* done something with the line art. Since the color hints are unreliable, the network had to learn to guess shading based only on the line art.

And with that, the model is complete! The online demo uses this exact setup.

Interesting finds

1. Network learns shadows and highlights well!
2. Coloring can go out of bounds, or not fill in completely, and it will fill in nicely.
3. Small color details, such as eye color, are not preserved.
4. The color hint processing can be related to applying dropout on the input image.
5. The network always expects a color hint, so when passing in no colors, it thinks we want all aspects of the image (hair,

skin, clothes) to be white. Lines and shadows that show up regardless of color still appear.

6. The method of generating line-art (openCV edge detection) left many artifacts. The network did learn to filter these out, but in doing so, sometimes ignores small lines such as noses and mouths.

All code for this project is available on my Github. The training images were all taken from Safebooru.

This project was inspired by PaintsChainer. I've been working on this idea slowly for a few months, but seeing someone get close to what I was envisioning motivated me to finish it up!

7 Comments kevin 1 Login

Recommend 1 Share Sort by Best

Join the discussion…

Turid Klomstad • 2 months ago

has been working wonderful. today i saw the results were messed up - what happened?? hope it will be back up running as before... thanks

Reply • Share ›

kvfrans Mod → Turid Klomstad • 2 months ago

Hey, I just checked and the results are definitely off. I did a reset on the server and it seems to be back to normal -- not sure what the problem was.

1 Reply • Share ›

Taizan Yonetuji • 2 months ago

Hi, I'm Taizan.
Thanks for mention to PaintsChainer.
I also happy that my work could impress someone who want same thing!

Reply • Share ›

kvfrans Mod → Taizan Yonetuji • 2 months ago

Hi, big fan of your work! Glad you noticed my project, this has been a goal of mine for a long time

Reply • Share ›

Taizan Yonetuji → kvfrans • 2 months ago

Thanks.
Did you see AutoDraw?
It is also interesting but I want make more art supportable system! (I dont have elegant idea)
We have very big dataset of line drawing from PaintsChainer, so if you interest to such project, you can came as internship or part time engineer
We have Japanese Office in Tokyo and American office in San Mateo.

Reply • Share ›

Mike • 2 months ago

This is awesome. I would love to be able to do higher def images than 512x512. Some kind of desktop app/gui would be awesome also. I'm not good enough at python to really make it work. I'll definitely be following this. Amazing work!

Reply • Share ›

kevinzakka • 2 months ago

Great post Kevin!

Reply • Share ›

ALSO ON **KEVIN**

What is the natural gradient, and how does it work?

1 comment • 5 months ago•

Chatavut Viriyasuthee — I see that you manage to push yourself further on the topic to elucidate the idea to yourself

Simple reinforcement learning methods to learn CartPole

22 comments • a year ago•

kvfrans — Randomness is required to use the generated probabilities. prob[0][0] contains a number between 0 and 1,

Generative Adversial Networks Explained

2 comments • a year ago•

Atcold — Very nice writeup!Just an advice about "This generative network takes in some 100 parameters of noise":

Markov Processes in Reinforcement Learning

1 comment • a year ago•

Artsiom Chapialiou — Thanks for nice explanation. Looks like there is a typo: second P11 should be Pn1 isn't it? ...

Kevin Frans

Read more posts by this author.

Share this post

YOU MIGHT ENJOY

What is the natural gradient, and how does it work?

A few months ago I attempted to understand the natural gradient, and wrote a post to help organize what...