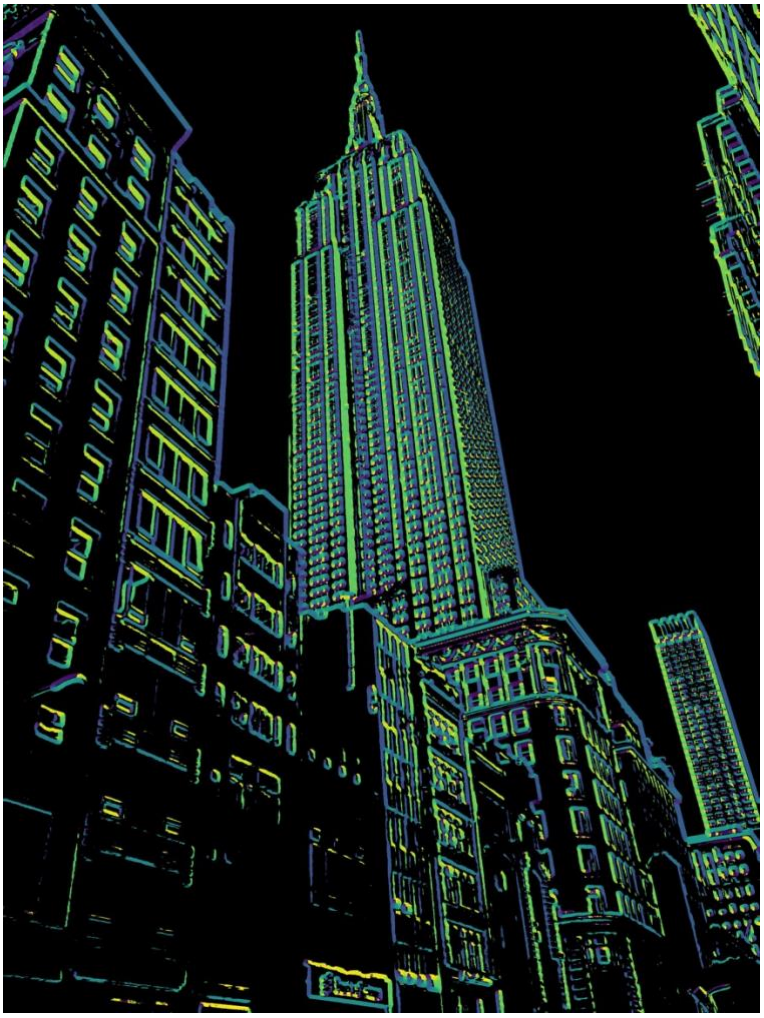
Shouvik Mani
Art and Machine Learning
Assignment 2: Algorithmic Generative Art
February 11, 2018

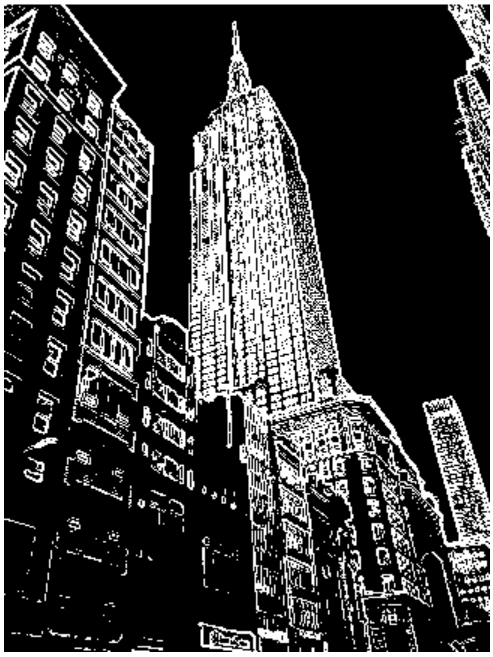
Edge Colorizer



Description
Edge Colorizer is a program I built to detect edges in an image and color them in a deliberate and coherent manner. It uses an image processing pipeline that consists of a series of convolutions and other matrix operations to blur an image, detect edges, and color the edges pixel-by-pixel. The final result is an image that resembles a skeleton of the original image, colored with bright, neon colors and contrasted against a black background.

All code and documentation for Edge Colorizer can be found in the Jupyter notebook edge_colorizer.ipynb. The original images are in the images/ directory and the results from Edge Colorizer are in results/.

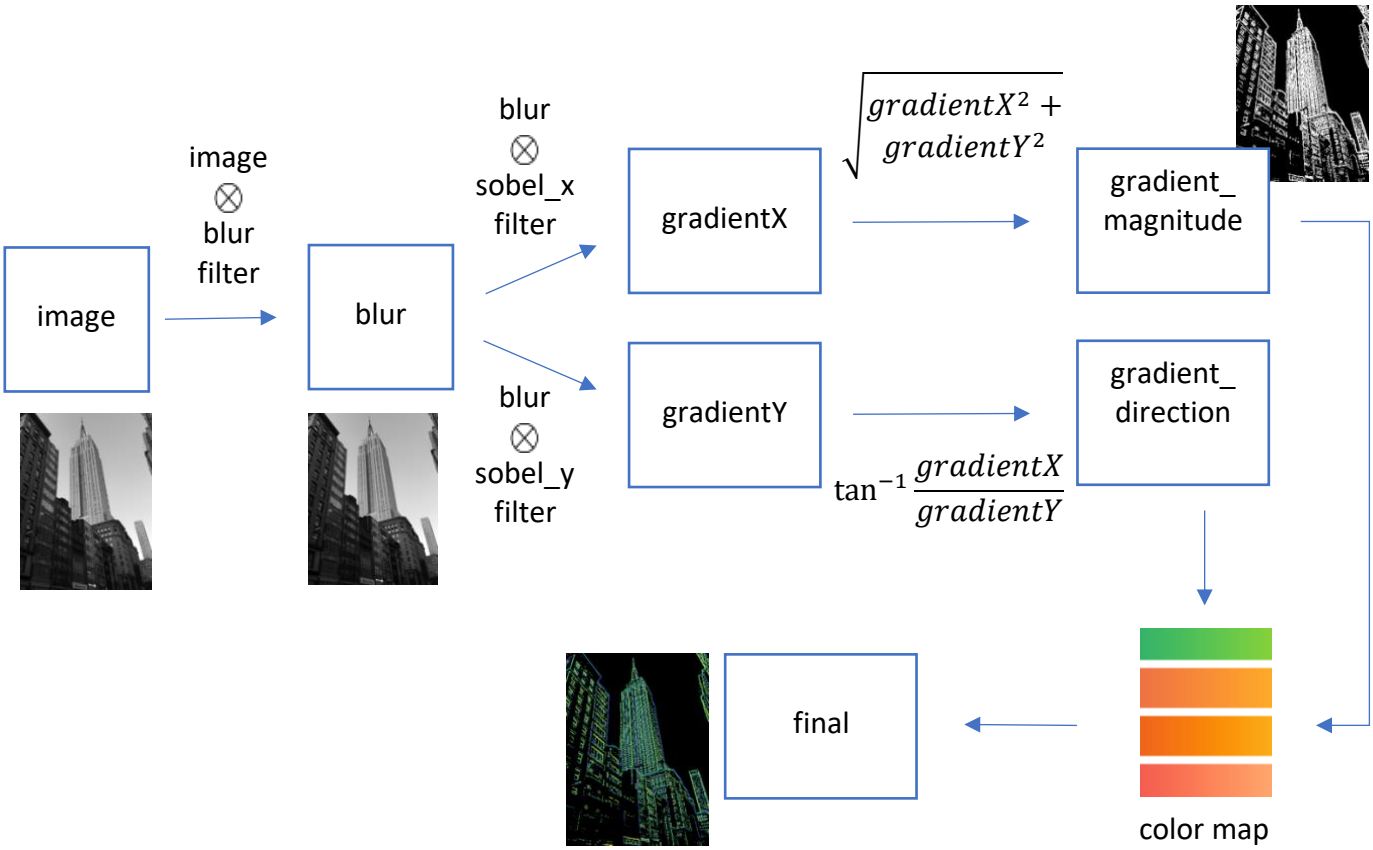
Concept
I got the idea for this project while doing an assignment in my Computer Vision class. In this assignment, I had to implement an algorithm to detect edges in an image. The algorithm was based on using convolutions with Sobel filters to create image gradients. It was a useful assignment because I learned how to process and transform images using linear algebra operations. However, I was left a bit unsatisfied at the end because the edges I detected were black-and-white. They all looked like this:



I was curious what it would be like if I could somehow color those edges. Although I liked the simple texture of the black-and-white edges, I imagined that adding some color could bring the edge images to life. So, I got to work in creating Edge Colorizer.

Technique
To create Edge Colorizer, I put together a sequence of linear algebra operations:

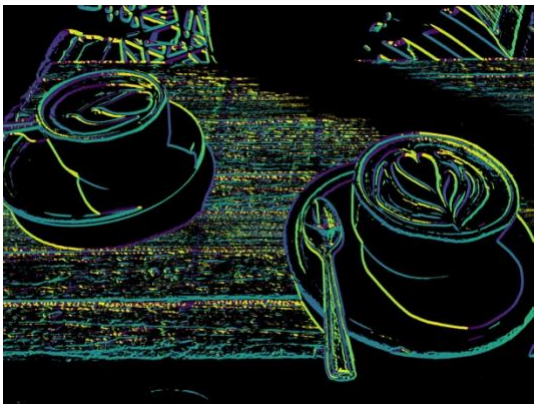
1. Blur the image: convolve the image with a simple blur filter. This step denoises the image and makes it easier to detect edges.
2. Detect edges: convolve the image with horizontal and vertical Sobel filters. The Sobel filters approximate the gradient of the image in the horizontal and vertical directions, which correspond to edges in the image. From these two convolutions, compute the magnitude and direction of the edges in the image.
3. Color the edges: take the thresholded edge image and convert it from grayscale to RGB using colors based on the direction of the edge pixels.

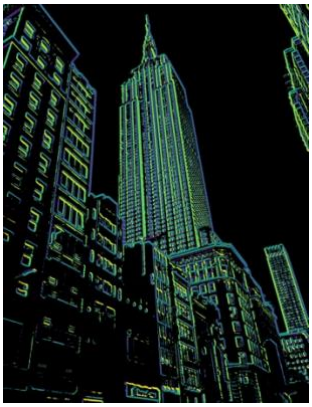


Convolution is the central operation in this pipeline. It is denoted by the ⊗ symbol in the diagram above.

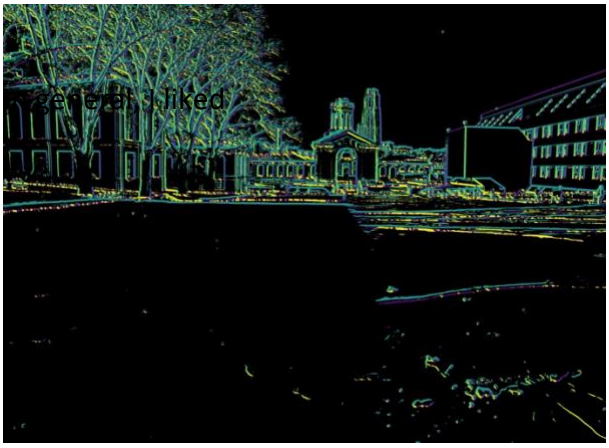
Process

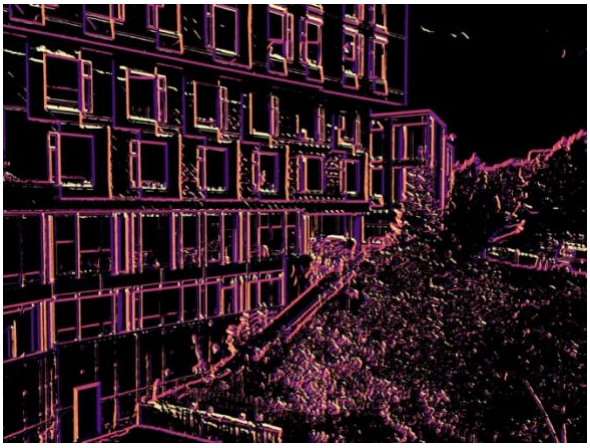
I used my program to color 10 of my own images. Here are some of my favorite results:





And here are some results I disliked:

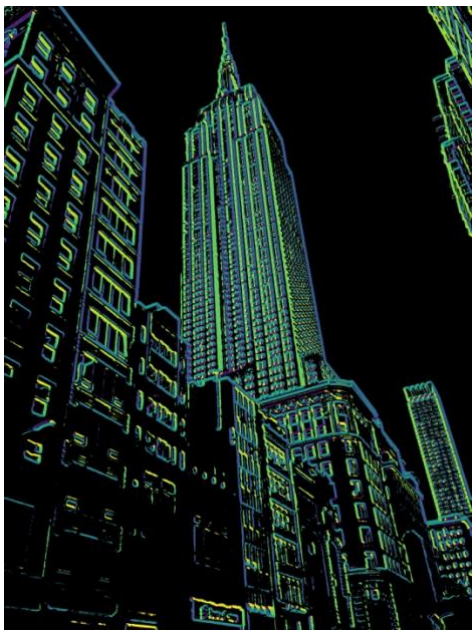




In general, I liked the images that were less noisy and had nice, clean edges. The teacups have beautiful curves that my algorithm is able to capture. The car and building also look stunning in their “edge color” representations. In contrast, the bottom two results (of Hamerschlag Hall and Gates) are full of noise coming from the trees in the original images (not shown).

I encountered many failures while implementing this edge coloring pipeline. For instance, I was detecting edges in the image without blurring it first. While this seemed to capture the edge magnitudes successfully, it resulted in extremely noisy edge directions. Eventually, I realized that I had to blur the image to denoise it before doing edge detection. Another challenge I struggled with was in figuring out how to color the edges. I thought of many strategies such as coloring the edges with a gradient, with colors from the original image, and with colors based on the edge directions of the image. Ultimately, I decided to use the edge directions to choose from a color map because this made the resulting image the most novel, yet most consistent.

Result



In the end, I chose the edge colorized version of the Empire State Building picture as my final result. The edges are colored green and blue in the vertical direction, and yellow in the horizontal direction. There are multiple buildings in the image, but the Empire State dominates. The black background adds a nice contrast. However, some of the edges in the bottom-left corner of the image are not captured well. Parts of the image, such as the base of the Empire State, feel three-dimensional and appear as if the colors are popping out.

Reflection

Part of the reason I chose this result is personal – I remember I snapped this photo from my friend's car as we were leaving New York City last semester. I had a great time in New York that weekend, and this photo is a memory of that. So, obviously, I was pleasantly surprised to see such a different and beautiful representation of that same image.

But aside from personal reasons, this is my favorite result because of its vibrant colors and coherent composition. I love how the colors are uniform across an edge and how the colors are similar for edges with similar angles. This adds consistency and coherence to the image, which was one of my goals when I started. Not surprisingly, I found that images with dominant lines had the best results from Edge Colorizer. The Empire State Building is a great example of that.

I am proud of my work in this project and impressed that I could apply mathematical techniques to create a work of art. This project taught me that a simple concept can give great results if it's implemented well. I also realized that it's important to understand the math behind algorithms so that I can tweak an algorithm as necessary to achieve the desired results. This project has given me confidence in my ability to combine my technical and creative skills to do art and machine learning.