Overview

This project is inspired by physical string art[1], where you reproduce images using a sequence of overlapping strings.

The usual way to generate a pattern like that is with an incremental approach: start at some arbitrary pin, then add a single line at a time looking for the optimal next pin that gets your vector representation closer to the bitmap version. This works surprisingly well. I made a web app called Raveler[2] that can convert any image into string art using that method.

That algorithm takes ~200 milliseconds in native C++, and I got it to around 600 milliseconds in the browser using a C++ to wasm/javascript compiler. I think it would be really cool to do this in real time with video or webcam input where 600ms per frame would be too slow. It seems like something a neural net would be good at, since it's basically just a weird compression algorithm.

The goal of this project is to train a neural network that will be able to generate similar string art representations of arbitrary images in real time. My plan is to convert a bunch of random images into line paths using the iterative approach I mentioned before, and train a neural net to generate similar patterns, using the original bitmap images as examples.

Challenges

This problem is not trivial, as the original algorithm is stochastic and quantized into discrete outputs (locations of pins around the circumference of the circle). Comparing the quality of one string path to another is challenging, and the fact that the problem is discretized means that there are no obvious derivatives that an iterative optimizer can follow to improve its results.[3]

I've identified two basic approaches to handle this problem:

1. Match the exact pattern (e.g. pin 1, 10, 62, 76, 89, 1, ...).
    a. This sort of works, but because the original algorithm is stochastic it doesn't do a very good job at generalizing. For instance, the above example would be exactly equivalent to traversing those same nodes in the opposite direction. The original algorithm is highly sensitive to minor variations in the input, so it may very well take either route while the model has no way to know about that symmetry.
    b. It isn't good at comparing the similarity of two points, since it doesn't know about the spatial wrap-around inherent in the problem -- pin 299 is adjacent to pin 0. For instance, pin 1 is closer to pin 280 than it is to pin 150.

[1] https://vimeo.com/175653201
[2] https://jperryhouts.github.io/raveler/
[3] I believe this problem is provably NP-hard, but I can't recall where I read that

2. Encode the target pattern as a probability density with respect to the edges traversed, rather than the nodes. e.g. the above example would be encoded such that the edge between pin 1->10 is traversed once, as is the edge between 10->62, 62->76 etc. This allows the model to match string patterns independent of their particular ordering.

3. Some sort of recurrent network, where the sequence is built up incrementally, similar to the original algorithm.

I'm going to rule out approach 3 right away, because the whole point is to do the whole process in one-shot. Building a path incrementally is the starting point here, not the end goal. Approach 2 is more likely to generalize than approach 1, but it has several important drawbacks:

A. It loses all information about the inherent symmetry and spatial layout of the problem. For instance, it can't know that an edge from A-B is more similar to an edge from A-B+1 than A-B+100.
B. It isn't true to the original point of the exercise-- using one continuous string path.
C. Rendering a few thousand separate lines is slower than rendering one long path
D. The resulting model is big because it has $O(N^2)$ output dimensions, where N is the number of pins around the circumference of the circle. It could probably be pruned down quite a bit, but I'll cross that bridge when I come to it.

Plan

My guess at this point is that I will need to come up with some differentiable loss function based on the probability distribution of edge locations. At this point it isn't clear exactly what route forward will be most promising.

I'm assuming similar work has been done in fields like reinforcement learning, graph theory, or protein folding in which the target metrics tend to be stochastic rather than deterministic. I plan to research similar systems to get ideas for solving this problem.

Ultimately I would like to deploy a usable demo of this model in a client-side web app. My goal is to develop a real-time version of what I did in the Raveler app. It will use an image stream from the user's webcam to generate a live updating string art representation of the same stream.