Sketchy Business Final Writeup and Reflection

Apoorva Talwalkar (atalwalk), Sophie Shao (swshao), Fiona Fan (ffan7)
Github: https://github.com/sophie-shao/dlFinalProject

Introduction

Sketches are an important design tool for developing ideas, creating aesthetics, and communicating ideas. While sketching can be done by anyone, we wanted to train a computer to take care of the task for us. For this we implemented the method in the existing paper CLIPascene by researchers at Tel Aviv University and Reichman University. CLIPascene is a generative computer vision solution which converts an input image into a customizable sketch based on two parameters of abstraction--fidelity and simplicity. Fidelity determines how closely the output sketch matches the original input image and visual simplicity controls how sparse the sketch is.

We chose this paper because it provides a straightforward approach to transforming images into a new artistic style while giving users some adjustability. Its methods also form the foundation for more advanced projects, for instance NeuralSVG, which means we can build on it later to tackle more complex transformations. We think it's especially valuable for artists and designers because it helps them explore ideas, create visual drafts, and speed up the creative process. As artists ourselves, we found this paper to be especially interesting and applicable to us.

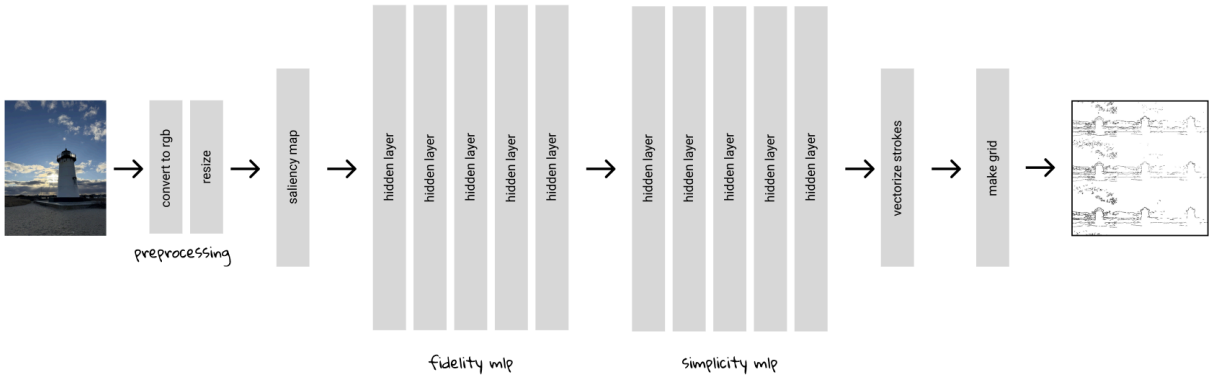
Paper Methodology

The paper's architecture trains 2 multi-layer perceptrons. First, the image is converted into a set of Bezier curves. The first MLP takes the curves' control points as input and outputs offsets to adjust them. Then, for a given n levels of fidelity and m levels of simplicity, a sketch abstraction matrix of size m * n is constructed. The sketches along the first row are generated (each decreasing in fidelity).

Then, for each of these sketches, an iterative simplification is performed to fill in each column of the matrix, for which they trained another MLP. This second MLP receives a random-valued vector and learns an n-dimensional vector representing the probability of the i-th stroke appearing in the final sketch and outputs a simplified sketch.

Our Implementation

In response to challenges we faced while trying to implement the paper's exact implementation, we decided to opt to usea saliency model with learned filtersin our implementation.



Preprocessing and Data

The pipeline begins by loading the input and rescaling it to 512 × 512 pixels while normalising pixel intensities to the range [0, 1]. To focus later processing on visually important regions, we run a Sobel filter that produces a saliency map. This saliency map is later used as a mask so that edges and textured areas drive the sketch more strongly than smooth regions.

Next, we feed the RGB image to OpenCV's pencilSketch filter. The filter's softness is governed by the parameter σ (sigma), which controls the blur radius applied inside the algorithm. We then multiply the monochrome pencil layer by a binary mask obtained from the saliency map: any pixel whose gradient magnitude exceeds a threshold τ (tau) is retained, while flatter regions are whitened. The result is a raster pencil drawing that is already simplified, where the image is visually quiet.

We perform morphological thinning (skeletonize) followed by contour tracing to transform this raster into clean vector strokes. Each traced polyline is down-sampled, jittered slightly to retain a hand-drawn feel, and finally redrawn onto a blank canvas with a random width chosen from a small palette. The outcome is a pure RGB image composed entirely of crisp Bézier-compatible strokes, making it ideal for further editing or laser-plotting.

Fidelity and Simplicity MLPs

Both σ and τ are produced, not by fixed formulas, but by two residual multi-layer perceptrons called FidelityMLP and SimplicityMLP. Each receives a single scalar slider value in [0, 1] and returns either σ in the physically meaningful interval [10, 110] pixels or τ in [0, 0.35]. Internally, every MLP contains five Dense → LayerNorm → GELU → Dropout blocks. A trainable scalar γ scales the block output before we optionally add it back to the running representation, so if the incoming and outgoing widths match, the network behaves like a residual network. Otherwise, it simply passes the new activation forward. After the final Dense unit, we squash with a sigmoid and

affine-transform to the target range, guaranteeing that the pencil filter never receives out-of-bounds parameters.

Because the intended mapping is conceptually simple—"low, medium, high" slider positions should yield low, medium, high numerical parameters—we pre-train each MLP on only three anchor pairs. We use mean-squared error between predicted and target values, Adam optimisation with a learning rate of 5×10^{-3}, and run for one thousand epochs, which is enough to drive the root-mean-square error below two σ-units or 0.01 τ-units.

In short, our MLPs are not learning from the world. Rather, they are learning our designer-specified scale. Because that scale is simple, deterministic, and one-dimensional, three anchor pairs plus a few steps of gradient descent give us everything we need.

Fidelity MLP

epoch	250/1000	500/1000	750/1000	1000/100
loss	26.3986	5.9129	2.8505	1.8717
Learned weights		30.140686	60.378193	90.44204

Simplicity MLP:

epoch	250/1000	500/1000	750/1000	1000/100
loss	0.0000	0.0001	0.0001	0.0000
Learned weights		0.04740788	0.20857121	0.30877233

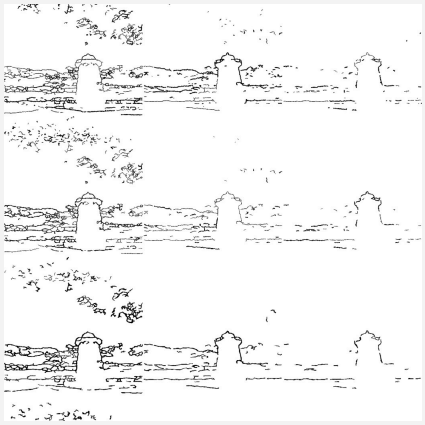
We note that our Simplicity MLP had extremely low loss across the epochs indicating some issue with the learning. With more time we would hope to improve on this, and, hopefully, generate even better results.

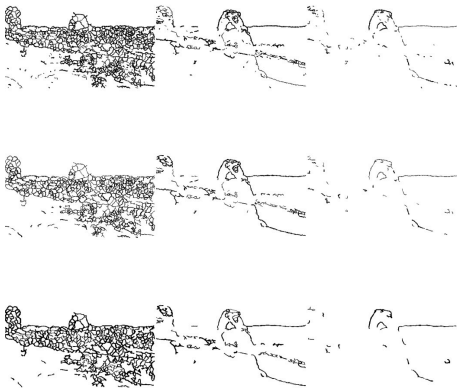
Outputs

Lastly, we vectorize strokes and create a grid to convert the pencil sketches into clean, abstract vector-style renderings and organize them into a comparative grid. This stage repeats the vectorization process across a matrix of input settings: combinations of fidelity and simplicity values derived from the two MLPs.

Results

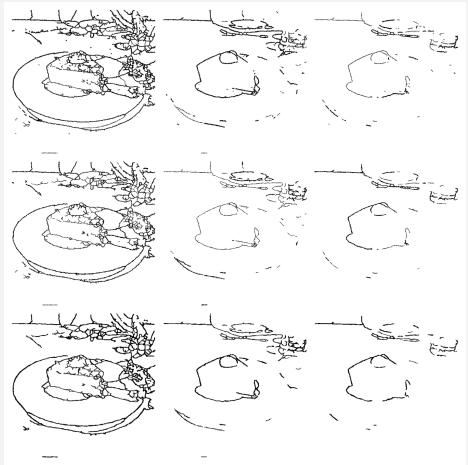
To measure accuracy, we ran our sketches through three LLMs–ChatGPT, Claude, and Perplexity–to see what objects the LLMs found in the sketches. We compared this to a list of ground-truth tags we had generated from the original input images to generate a quantitative accuracy value.

input	output	quantitative accuracy
		93.3% LLMs noted the lighthouse, sky, rocks, clouds all present in the input image, resulting in a high accuracy.
		73.3% Objects identified were circular plates, food, table, sauce cups. One LLM was able to identify the oysters, but the other two were not able to.
		53.3% LLMs noted the two foreground figures. But, they were confused by the background of the image. ChatGPT thought it was an amusement park. Claude was able to distinguish the tables in the background but thought there were some railings and flowers in front.



80%

Predictions across the LLMs included some kind of arch, rock formations, open sky, and other rocks. Some thought the right side of the sketch represented vegetation.



96.6%

All LLMs were able to identify the plate, cake, table, and utensils. One LLM predicted other plates of food in the background due to the noise.

Discussion of Results

Our model had mixed accuracy. We found that landscapes generally had a higher accuracy as we see with the lighthouse and rock images above. We think this is because the distinctive structural features in landscapes are preserved even when converted into sketches. Also, images like the cheesecake did well because the simpler background meant that there was less noise and distraction in the sketches. In contrast, images with foreground figures and busy backgrounds were harder for the LLMs to decipher. Facial features can become ambiguous when turned into line drawings. Additionally, the backgrounds in these images often are rendered in less detail making them harder to discern.

Challenges

We had the most trouble implementing a rasterizer for the MLPs that was compatible with tensorflow. We experimented with diffvg and TensorFlow Graphics' `tfg.rendering.rasterization_backend.rasterize`. However, these approaches failed because the TF rasterizer leaked semaphores and hung training jobs, while diffvg couldn't install cleanly due to CUDA/toolkit mismatches and PyTorch version conflicts. To get around this, we instead decided to implement a saliency map that we built the two MLPs on top of, to try and stay consistent with the architecture of the paper.

Another challenge we faced was defining accuracy. The authors of the original paper compared the results of their model to previous attempts to generate sketches from images, however they didn't explicitly define how they arrived at a numerical accuracy value. To measure accuracy for ourselves, for each image we tested, we predefined a set of "ground-truth" tags (eg. person, lighthouse, rocks, sky, grass, etc.). We fed the output sketches for each of these into three different LLMs (Chat-GPT, Perplexity, and Claude) with the prompt: "list the top 5 objects you see in this sketch". From here, the percentage of predicted objects that were included in the "ground-truth" tags and used this as our accuracy metric.

Reflection

Our project successfully achieved its base goal of producing sketch abstractions with controllable parameters. While we didn't implement the exact Bezier curve approach from the paper, our alternative method delivers comparable results with similar user control capabilities. We reached our target objectives but didn't fully realize some stretch goals such as implementing additional abstraction types. The model's performance differs from our initial expectations—our vectorization approach generates more organic, hand-drawn appearing results compared to the paper's cleaner Bezier curves, with the MLPs effectively mapping user parameters to appropriate internal settings.

Our approach evolved significantly from the initial plan, pivoting from Bezier curves to direct vectorization, simplifying the two-MLP architecture to focus on core parameters, and incorporating jitter and stroke variation for more natural results. Perhaps, had we abandoned the rasterizer faster instead of trying multiple approaches, we could have had more time to implement our target goal or explore the original Bezier curve approach more thoroughly.

Potential future improvements include implementing additional abstraction styles like hatching and shading, enhancing stroke generation with more advanced vectorization techniques, developing a user interface for interactive parameter adjustment, and optimizing performance for higher resolution images. Through this project, we gained valuable insights into the practical implementation challenges of computer vision papers, developed hands-on experience with image processing pipelines, and cultivated a better understanding of parameterizing artistic effects while balancing algorithmic control with artistic quality.