CS230

Deep Sketch Drawer
CS230 Final Project Report

Yipeng He
yipenghe@stanford.edu

Di Bai
dibai@stanford.edu

Wantong Jiang
wantongj@stanford.edu

Abstract

This is the final project report for CS230 Deep Learning Fall 2018-2019. In this project, we aim to generate human-style sketches from real pictures. We use a public dataset called Sketchy Dataset [1] and implement two different architectures: *Neural Style Transfer* and *Conditional GAN*. We show that sketch images are hard to generate with pure convolutional neural networks. On the other hand, neural style transfer can be used to generate realistic sketches and GAN models are promising for this task.

1 Introduction

In our project, we aim to generate human-style sketches from real pictures. The input is an arbitrary colored picture with one main object and the output is a human-style black and white sketch of the object. We tried three models: simple convolutional layers with mean squared error loss, neural style transfer and finally, conditional GAN (both with holistically-nested edge detection). We show that the simplest CNN does not generate satisfying results while the last two models could produce promising sketch pictures.

We have seen a lot of work relevant to sketches. However, most of these work aim to generate colored pictures from sketches. For instance, in [2], Sangkloy et al. aimed to produce images from sketches with some color specified. Most of these work used datasets with images and corresponding sketches drawn by human beings, which is time-consuming and cost inefficient. As a result, our project, intending to generate human-style sketches from real pictures, could help enlarge and enrich relevant datasets.

2 Related work

We found some related papers focusing on human-style sketches. In this section, we briefly introduce them.

Sketch Recognition In [3], Eitz *et al.* used SVM with hand-crafted features to classify human-style sketches. In [4], Yu *et al.* proposed a deep CNN model for free-hand sketch recognition that can even beat human performance.

Sketch Generation In [5], David H. *et al.* constructed *Quickdraw* dataset, and proposed sketch-rnn model to construct stroke-based drawings of common objects. Unlike our outputs, they generated a sequence of strokes simulating the drawing process, so RNN is appropriate in this case. In [6], Yajing C. *et al.* further improved the sketch-rnn model so that multiple categories could be fed into the model together. They used CNN instead of BRNN and modified the objective function. We

CS230: Deep Learning, Fall 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

inspired from them to use CNN to solve our problem. The above projects differentiate from us that they both used sketches from *Quickdraw* instead of real pictures for training. In [7], Muhammad *et al.* used reinforcement learning to train a sketch abstraction model. They got inspired from human sketching instead of image translation. Their photo-to-sketch approach is proved to be superior than the state-of-the-art alternatives.

Our project distinguishes from existing works that we use images with arbitrary background instead of white background in [7] or stroke sequence in [5] and [6]. We do not consider human behavior in this project (no strokes) but only image transfer.

3 Dataset and Features

3.1 Dataset

The dataset we use is Sketchy Database [1]. The dataset consists of approximately 75,471 sketches and 12,500 objects from 125 categories. The team generated the dataset by inviting volunteers to sketch photos that were shown to them for a limited time. It can be downloaded from `http://sketchy.eye.gatech.edu/`. In figure 1, we show an example. In the figure, the first one is the real image, while the following 5 pictures are sketches of the image drawn by volunteers. All the real pictures as well as the sketches have size $256 * 256 * 3$.

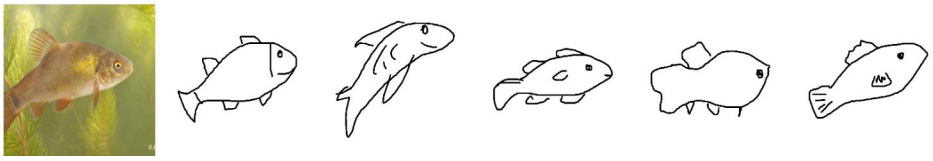

Figure 1: A sample of the Sketchy Database

To divide the data, there are 125 categories and each category contains 100 images, so we choose 80 images from each category for training, 10 images for validation and 10 images for test.

As shown in figure 1, the variance of the sketches is very large even for the same image. As a result, we need some data preprocessing to filter out sketches that are damaged or inaccurate in order not to affect training.

3.2 Data Preprocessing

Poor Sketch Removal Firstly, along with the dataset, they provided some annotation files that specified poor-quality sketches. We wrote a script to remove all those sketches to avoid them affecting training. After removal, there are four to five sketches for one image on average.

Sketch Selection After removing all the malicious sketches, we found that sketches of the same image still had very large variance. So we chose to use one sketch for one image. We used the edge picture generated by holistically-nested edge detection described in the next section and selected the most similar sketch based on cosine spacial distance between the sketch and the edge.

Data Augmentation Though there are totally 12,500 images, there are only 100 images in each category which are not enough for experiments with less number of categories. As a result, we rotated and flipped the images and sketches to obtain more data.

Data Modification As mentioned above, all the sketches are of size $256 * 256 * 3$. In order to fit for the GAN model, we re-sized them into $28 * 28 * 1$.

4 Methods

4.1 Edge Detection

We generate intermediate edge images using HED network, a convolutional neural network for end-to-end edge detection. It inserts side-output layers after convolutional layers, then imposes deep supervision on each side-output layer, and then adds a weighted-fusion layer to learn to combine outputs. The loss is defined as follows:

$$\mathcal{L}(\vec{W}, \vec{w}, \vec{h}) = \mathcal{L}_{side}(\vec{W}, \vec{w}) + \mathcal{L}_{fuse}(\vec{W}, \vec{w}, \vec{h})$$

where $\vec{W}$ represents standard network layer parameters, $\vec{w}$ represents corresponding weights of side-output layers' classifiers, $\vec{h}$ represents fusion weight. $\mathcal{L}_{side}$ denotes the image level loss for side-outputs, while $\mathcal{L}_{fuse}$ denotes the error between fused predictions and ground truth label maps.

4.2 Neural Style Transfer

We synthesize sketchy images by mixing edge images and style of dataset sketches, based on convolutional neural network VGG-Network. Inspired by neural style transfer algorithm, we use edge images for content reconstruction and dataset sketches for style representation. Feature spaces of 16 convolutional layers and 5 max-pooling layers are used. For content part, the loss is defined as follows:

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F^l_{i,j} - P^l_{i,j})^2$$

where $\overrightarrow{p}$ represents the original edge image, $\overrightarrow{x}$ represents generated image, P^l, F^l represents their feature representation (activation) in layer l respectively.
For style part, the loss is defined as follows:

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} \frac{1}{4N_l^2} \sum_{i,j} (G^l_{i,j} - A^l_{i,j})^2$$

where $\overrightarrow{a}$ represents the style image, $\overrightarrow{x}$ represents generated image, G^l, A^l represents their style representations in layer l respectively, N^l represents number of filters in layer l times the size of corresponding feature map.
The total loss is the sum of content loss, style loss and total variation regularization. We perform gradient descent on a white noise image to generate new imaged that match the 'content' of edge images and 'style' of dataset sketches.

4.3 Conditional GAN

Another approach that we have explored is the usage of Conditional Generative Adversarial Networks (CGAN). CGAN has been used widely in translation tasks and have proved to be effective in various tasks [12]. A CGAN network is formed by two parts: a generator and a discriminator. A generator aims to generate an image that is like a real image to discriminator, while the discriminator aims to distinguish between a real image and a fake image generated by the generator. For CGANs, discriminator and generator are provided an image as input, which allows the discriminator and generator to use this extra information to perform their tasks. We adapt a normal GAN model used on quickdraw dataset [14] and convert it into a CGAN where the generator has an edge map as input. In our model, discriminator is not given an input.

For the discriminator, the model uses convolutional neural networks that had filter numbers with 64, 128, 256, 512, respectively, and then it is flattened and passed to a dense net with sigmoid activation. The loss function used is binary cross entropy.

For the generator, the model first flattens the input image and passes it through a dense net. The output is reshaped to shape $7 * 7 * 64$. The model then performs transpose convolution to the image with filters 32, 16, 8 with dropout. Finally, the model uses a convolutional layer with sigmoid activation to output the fake sketch image.

5 Experiments/Results/Discussion

Before we turned to neural style transfer and GAN, we first tried using pure convolutional neural networks to generate sketches. However, the results were not ideal. We then started to frame the problem as finding a "bad" edge detector problem. When human draw sketches, they are mostly sketching out the edges of the images.

In neural style transfer method, how a generated image looks like a human-style sketch cannot be directly reflected by the loss (but it still conveys information of similarity), so we tuned hyperparameters (e.g., learning rate, β_1 and β_2 in Adam Optimizer) also by looking and judging the outputs by ourselves. We tried different learning rates 0.5, 1.0, 10.0, 30.0. Among them 10.0 produces the best results. It can been seen that after 1000 iterations, when learning rate is smaller than or equal to 1.0, the outputs are pretty scribbled and noised, denoting that it hasn't learned enough to generate sketches, in other words it's underfitted. When learning rate is 10.0, pretty good sketches are generated, as shown below, with total loss 6.2257e+06 on average. When learning rate is 30.0, the results turned out to have many more weird lines that shouldn't appear in a human-style sketch.

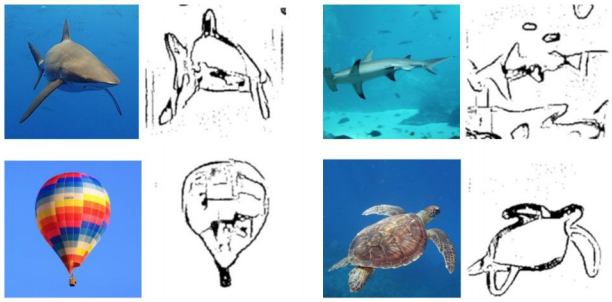

Figure 2: outputs of HED + Neural Style Transfer

In CGAN experiments, we have also tried using different learning rates for the discriminator model loss. We tried 0.8, 0.08, 0.008 and 0.0008. Out of the four learning rates, 0.0008 produces the most reasonable images, which is shown below. With higher learning rates, the accuracy of the discriminator quickly drops to around 0.5, which means the discriminator is randomly guessing. The higher learning rates therefore limits the performance of the generator, as the generator can easily "fool" the discriminator.

An important preprocessing step we did for training our GAN was to convert all the sketches from black stroke white background to white stroke black background. Originally the GAN failed to learn anything and produced images like below.

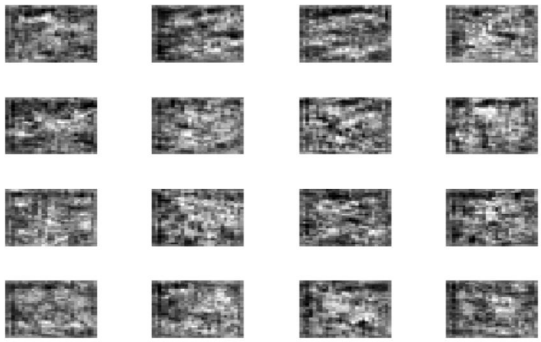

Figure 3: outputs using white background sketches

We can see that the fake images generated in the first two rows have similar orientations to the conditional images. This shows a great progress in achieving what we aimed for - a sketch that preserves the pose and orientation of an image.

Figure 4: outputs of Conditional GAN with edge input

Admittedly, the results are far from satisfying and still have large room for improvements. After experimenting with different convolutional neural network architectures, neural style transfer and GAN structures, we have a better understanding of the task. First of all, Different human draw sketches in different styles. For the same photo, different people can sketch in different detail levels. Simply feeding color image as input and sketch image as target to a network is similar to asking it to learn a general representation of different sketch styles. Therefore it may be hard to use pure convolutional neural networks to generate sketches with color images. Having identified such problem, we then tried approaches using edge detection. Edge detection algorithms reduces the noise in an image and provides the most significant information of sketches: edges. With edges as input, generative adversarial networks are able to capture the position and pose of an object. Even though our discriminator and generator are relatively simple, this might indicate that black backgrounds with with white strokes might be easier for networks to learn information from.

6 Conclusion/Future Work

Going directly from color image to a sketch that has a similar pose is difficult with our initial attempts with convolutional neural networks. It is much easier to use edge inputs to generate sketches than with color images as edges provide cleaner and conciser information to the network. Thus, we believe framing it as a "bad" detector problem might be a good way of thinking to tackle this task. If we had more time, we would first structure our project better.

First of all, we would spend more time in using GAN to generate sketches. We will develop and experiment with more complex and advanced generator and discriminator networks. Currently the discriminator model flattens image immediately, which may limit the information it can get from an image. We would also try using color image as input to generator and discriminator. Additionally, we would also explore cycle GAN, an advanced GAN network that has been used in art creation tasks[11].

Second, we will try using Recurrent Neural Networks to develop generators. In the sketchy databases, there are time stamps information for the strokes and there has been work using RNN networks to generate sketches with time information.

Last, the cross domain embedding mentioned in [13] might also be a great way to explore the mapping from image to sketch with better computational power.

7 Contributions

The code can be found at `https://github.com/Baidi96/deep_sketch_drawer`

Contributions of the team:

- **Yipeng He**: CNN model, Conditional GAN model
- **Di Bai**: HED, Neural Style Transfer model
- **Wantong Jiang**: Data preprocessing, Paper writing

References

[1] Patsorn, S., Nathan, B., Cusuh, H. & James, H. (2016) The Sketchy Database: Learning to Retrieve Badly Drawn Bunnies *ACM Transactions on Graphics (proceedings of SIGGRAPH)*.

[2] Sangkloy, P., Lu, J., Fang, C., Yu, F. & Hays, J. (2017, July) Scribbler: Controlling deep image synthesis with sketch and color. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (Vol. 2).

[3] Eitz, M., Hays, J., & Alexa, M. (2012). How do humans sketch objects?. *ACM Trans. Graph.*, 31(4), 44-1.

[4] Yu, Q., Yang, Y., Song, Y. Z., Xiang, T., & Hospedales, T. (2015) Sketch-a-net that beats humans. *arXiv preprint arXiv:1501.07873.*

[5] David H. & Douglas E. (2017) A Neural Representation of Sketch Drawings. *arXiv preprint arXiv:1704.03477v4.*

[6] Yajing C., Shikui T., Yuqi Y. & Lei X. (2017) Sketch-pix2seq: a Model to Generate Sketches of Multiple Categories. *arXiv preprint arXiv:1709.04121v1.*

[7] Muhammad, U. R., Yang, Y., Song, Y. Z., Xiang, T. & Hospedales, T. M. (2018) Learning Deep Sketch Abstraction. *arXiv preprint arXiv:1804.04804.*

[8] Gatys, L. A., Ecker, A. S. & Bethge, M. (2015). A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576.*

[9] Xie, S., & Tu, Z. (2015). Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 1395-1403).

[10] Anish, A. (2015). Neural Style. `https://github.com/anishathalye/neural-style`

[11] Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*.

[12] Isola, P., Zhu, J. Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *arXiv preprint*.

[13] Sangkloy, P., Burnell, N., Ham, C., & Hays, J. (2016). The sketchy database: learning to retrieve badly drawn bunnies. *ACM Transactions on Graphics (TOG)*, 35(4), 119.

[14] Auger, C. (2018). quickdraw-gan. `https://github.com/coreyauger/quickdraw-gan`