CS230

Drawbot - Converting Bad Sketches into World Class Masterpieces

Jimmie Harris, Tayo Falase, Usman Khaliq
Design Impact
Stanford University
jdharris@stanford.edu,tfalase@stanford.edu,usmank@stanford.edu

Abstract

For our project we designed and developed a deep learning application that takes as its input a rough(or badly) made sketch and outputs a more uniform, aesthetically pleasing sketch.

1 Introduction

How might we design and develop a deep learning application that takes as its input a rough (or badly) made sketch and transforms it into a drop-dead masterpiece?

As designers, two of us have often struggled to bring our ideas to life in an aesthetically pleasing manner, while all three of us would dearly love to have a tool that makes it easy for us to churn out beautiful sketches. This problem also excites us because we want to focus on generating unique outputs from the rough sketches—the same badly drawn sketch of a car would generate two completely different, yet beautiful sketches of the vehicle. We are also interested in translating a medium like design, which is difficult to box within a set of parameters, into the deep learning realm.

The input to our algorithm is a bad sketch of a face. We then use a Multimodal UNsupervised Image-to-image Translation (NVlabs, MUNIT) trained network to output a much cleaner and aesthetically accurate image of a face.

2 Related work

The original CycleGan paper[1] has been used in a number of projects for translating a specific style of artwork from some artwork onto other artwork. For instance, it was used to translate photographs into the painting styles of Monet and Van Gogh. Although CycleGan works very well with superimposing a specific "style" on some images, in our case, the "style" of the good sketches did not translate well onto the "bad" sketches. A major reason for this was that since the good and bad sketches were substantially different from each other, CycleGan was unable to handle this more extreme transformation.

Cartoon GAN[2] is another project that used a generative adversarial network to transform photos of real-world scenarios into cartoon style images. It used a semantic content loss to deal with substantial style differences between photos and cartoons, and an edge-promoting adversarial loss for preserving clean edges. However, it wasn't specifically suited to our case since we did not want to generate output in a specific cartoon style.

The Sketch simplification[3] project trained a network that converted complex rough sketches into clean line drawings. The project jointly trained both supervised and unsupervised data by employing

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

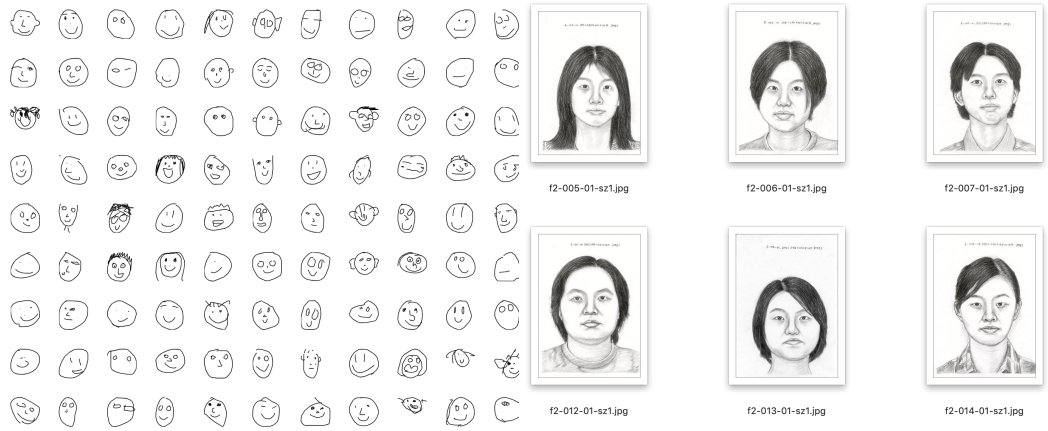

Figure 1: A subset of our source images for the MUNIT model from the Google Quick, Draw! dataset next to a subset of our target images from the CUHK Face Sketch database.

an auxiliary discriminator network. The model was trained with fully supervised rough sketch-line drawing pairs, unsupervised rough sketches, and unsupervised line drawings. It did not perform that well for our case, since the "bad" sketches needed to be translated into more features instead of just getting their outline cleaned up, according to our design goals.

Sketch-rnn[4] is a recurrent neural network(RNN) that constructs stroke based drawings of common objects. The model carried out conditional and unconditional sketch generation. It was slightly unsuitable for our case because if the user drew "bad" sketches, sketch-rnn ended up generating similar "bad" sketches, whereas we wanted to generate a good version of a sketch from its corresponding bad version.

The Multimodal Unsupervised Image-to-image Translation(MUNIT)[5] is the most promising project that was similar to what we wanted from Drawbot. MUNIT takes as input an image in the source domain and learns the conditional distribution of corresponding images in the target domain. Rather than a one-to-one mapping, MUNIT has the ability to generate a diverse range of images from a source image. This outcome was desirable for us as we wished to differentiate DrawBot from other algorithms that return images with little-to-no variation. MUNIT decomposes the image representation into a content code that is domain-invariant and a style code that captures domain-specific properties. In our case, MUNIT would potentially translate a "bad" sketch into a "good" sketch by recombining the content code of the bad sketch with a random style code sampled from the "good" sketch domain.

3 Dataset and Features

The dataset of “bad sketches” required some preprocessing. We use Google’s open-source Quick, Draw! Dataset[6], which has thousands of hand-drawn sketches stored as time-stamped vectors, capturing information such as hand strokes and information about when the pen was lifted. We needed the data in the format images rather than vectors, so we wrote a script in Python using a Quick, Draw! Python API[7]. The script saves each vectorized representation of an image as a 255 x 255 pixel PNG file. We had originally found a script on Github that converts the data from the .npc format to PNG images, but it only output 25 x 25 pixel images. After training the model on a small subset of data, we found that larger images had better potential to produce promising results.

The “good sketch” dataset for the target domain was more difficult to find. Early in our project we had thought about using images from The Noun Project, a popular site with thousands of vectorized images in a consistent style. However, there isn’t a way to download images from the site en masse without scraping the images, risking infringement on copyrights and terms-of-use. To start, we found a dataset on kaggle.com that had a subset of images scraped from the Noun Project. As mentioned before, this didn’t produce the results we desired, we suspect because the datasets were too dissimilar for the CycleGAN model.

In the end, we decided to focus on one category of sketches, faces, instead of training the model on a variety of drawings, which included over a hundred categories, from cars to horses to fruits and beyond. We hypothesized that using input and target datasets that mapped more neatly to each other would produce better results. We found a dataset of "good" portrait sketches from the Chinese University of Hong Kong to use as our target[8].

We then used 300 Quick, Draw! bad sketches as our training A group and 300 CUHK faces sketches as our training B group. The model then attempted to learn the relationship between the two unpaired groups.

4 Methods

As our project is an application project, we will outline our process here from our initial tests with CycleGAN to our current use of a MUNIT model. The implementations of these models are well documented, so we will summarize the key aspects that we sought out for DrawBot.

At the start of our project, we trained a CycleGAN model on 3 distinct datasets. For the first test, we used 300 randomized Quick, Draw! images of any type and 300 Noun Project svg files. Our second test involved using 300 faces from the Quick, Draw! dataset with 300 sketched faces collected from the Behance Artistic Media database (BAM!). Finally, we used 300 faces from the Quick, Draw! database along with 300 similarly sketched faces from the CUHK database.

As the project progressed, we realized that a key component of our vision for DrawBot was that the generated images needed to be varied, rather than enhanced regurgitations of the input. MUNIT achieves this by combining the content code of the input to a randomly sampled style code of the target. The model learns by employing an encoder E_i and decoder G_i for each domain. To translate an input to an output, the encoder/decoder pairs of each domain are swapped. The nonlinearity of the decoder allows the model to be multimodal. This is all under the assumption of a partially shared latent space (the content is shared across the domains while the styles are unique).

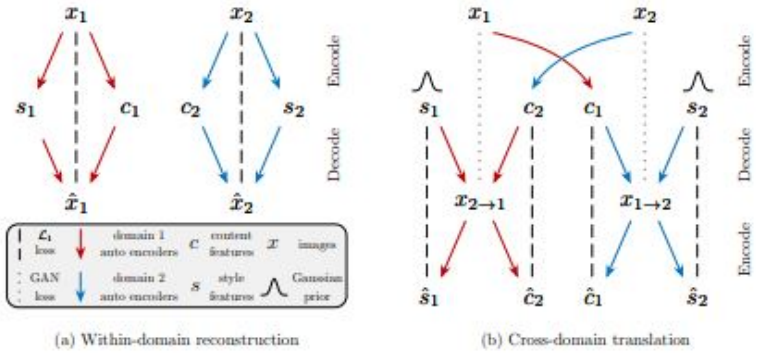

Figure 2: Model depicting how MUNIT extracts content and style information and then swaps this information to form new novel pictures. [Figure, Huang]

MUNIT combines a bidirectional reconstruction loss and an adversarial loss. The bidirectional reconstruction loss ensures that the encoders and decoders are inverses, while the adversarial loss evaluates the distribution of translated images against the distribution in the target domain.

MUNIT utilizes a learning algorithm that extracts both content and style information from images. After learning these different parameters, the algorithm extracts the content code from the input image and swaps in new stylistic information. To ensure accuracy both constructing and deconstructing images, it uses a bidirectional reconstruction loss. This loss is made up of:

Image reconstruction loss: Which ensures an image can be reconstructed after translating

$$\mathcal{L}_{recon}^{x_1} = \mathbb{E}_{x_1 \sim p(x_1)}[||G_1(E_1^c(x_1), E_1^s(x_1)) - x_1||_1]$$

2 latent reconstruction losses: For a given latent style and content from the distribution, it can be reconstructed after translating

$$\mathcal{L}_{recon}^{c_1} = \mathbb{E}_{c_1 \sim p(c_1), s_2 \sim q(s_2)}[||E_2^c(G_2(c_1, s_2)) - c_1||_1]$$

$$\mathcal{L}_{recon}^{s_2} = \mathbb{E}_{c_1 \sim p(c_1), s_2 \sim q(s_2)}[||E_2^s(G_2(c_1, s_2)) - s_2||_1]$$

The loss here is $\mathcal{L}_1$ because it is more likely to produce sharper output images. The style loss encourages a diversity of output styles while the content loss encourages content preservation during translation. In addition to the reconstruction loss, MUNIT employs an adversarial loss:

Adversarial loss: To make sure that the images generated appear similar to real images in the target domain

$$\mathcal{L}_{GAN}^{x_2} = \mathbb{E}_{c_1 \sim p(c_1), s_2 \sim q(s_2)}[log(1 - D_2(G_2(c_1, s_2)))] + \mathbb{E}_{x_2 \sim p(x_2)}[logD_2(x_2)]$$

Total Loss Function:

$$\overset{min}{E_1, E_2, G_1, G_2} \quad \overset{max}{D_1, D_2} \quad \mathcal{L}(E_1, E_2, G_1, G_2, D_1, D_2) =$$
$$\mathcal{L}_{GAN}^{x_1} + \mathcal{L}_{GAN}^{x_2} + \lambda_x(\mathcal{L}_{recon}^{x_1} + \mathcal{L}_{recon}^{x_2}) + \lambda_c(\mathcal{L}_{recon}^{c_1} + \mathcal{L}_{recon}^{c_2}) + \lambda_s(\mathcal{L}_{recon}^{s_1} + \mathcal{L}_{recon}^{s_2})$$

5 Experiments/Results/Discussion

When training the MUNIT algorithm we focused on varying 3 main hyperparameters – learning rate, number of downsampling layers, and number of iterations. After some experimentation we saw that increasing the default learning rate from 0.0001 to 0.0002 increased the speed at which the model "learned" some parameters without sacrificing quality. We also increased the number of downsampling layers in MUNIT in the hopes that more abstraction would allow the model to learn from the basic inputs and speed the learning process. However, when we increased the number of downsampling layers from 2 to 4, the model produced worse results and the parameter was left as is. We also varied the number of iterations that the model underwent as we were training. We noticed a steady increase in image quality up until 50,000 iteration but at that point it is hard to tell if the model was getting better.

From the pictures the model appears to overfit the data, manifesting as additional detail where none is provided (ex. the bad sketches don't have ears, but the good ones do). We believe that this is caused by the two domains being too different from one another. When the model attempts to learn the mapping, there is not enough raw input data to help it make accurate guesses about the desired outcome. This could be solved by training the model on sketches that are more complex (more features and lines, but still poorly drawn) which would allow MUNIT to learn more features to map.

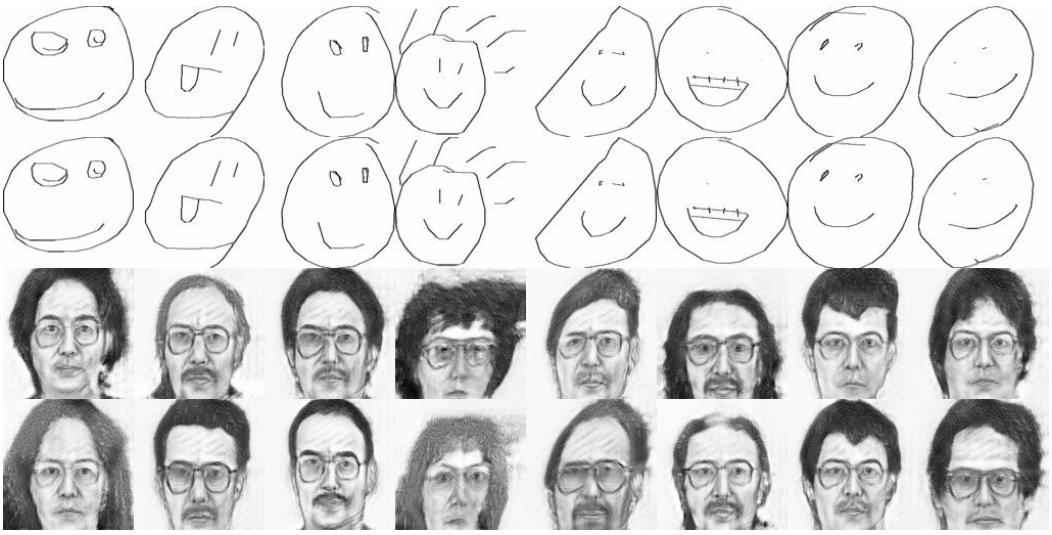

Figure 3: Results after 80,000 iterations of the MUNIT model with the best hyperparameters we found. Notice the general shape of the head is learned and filled well, however the fine features are not.

Being an image to image translation model, MUNIT does not have an easy evaluation metric. The authors used mechanical Turk to have people rate which images seemed the most accurate and also paired this with a LPIPS [9] distance test that measures diversity. While we did not have the time or resources to complete a similar Mechanical Turk survey (500 questions each answered by 5 different workers)[5] we were able to run an LPIPS test and saw from a randomized sample an LPIPS score of 0.388. This is a fairly high value, which says that the outputs we generate are fairly diverse, however it is not likely meaningful because our results would not have done well against human perception and the user has little control over the diversity.

6 Conclusion/Future Work

Figure 3 shows training results after 80,000 iterations of MUNIT. The model was trained on a set of 300 unpaired bad sketches and good sketches and tested on a set of 100 bad sketches from Quick, Draw!. This demonstrates the ability to create realistic sketches within the confines of the shapes given, but highlights trouble with specific features.

Our model seemed to have learned some features from the input images (face size and shape) fairly well. However, it is unable to adjust finer details such as nose shape and smiling/frowning. We believe that the main inhibitor was the quality of the input drawings. If we had been able to use drawings with more detailed features (many of them lack noses, ears, hair, etc) we believe that this model could have learned those features as well.

Once the model began outputting suitable images, we could then proceed to run a human perception study and another round of LPIPS distance testing on the data to see if the images are both believable and distinct.

7 Contributions

Tayo worked on collecting and preparing the data. The Google Quick Draw! dataset was in a form unsuitable for our algorithm, as it was captured as vectorized points and time stamps. However, we found an API that allowed us to generate 255 x 255 images from the dataset. Tayo wrote a Python script to facilitate the collection of images.

Before we landed on our current datasets, we experimented with other datasets that were key to informing our eventual direction. For example, we used images from the Behance Artistic Media database, which required downloaders to classify 200 images before access was granted. Jimmie and

Tayo classified the images necessary to gain access to the database and Tayo wrote a Python script to extract images from the SQL database provided.

Jimmie worked on identifying and testing different machine learning frameworks. We identified 5 distinct algorithms that might be suited for our purposes and ran multiple training tests on 4 of them.

Usman set up the AWS instance, researched models to find one suitable for our goals, researched methods of improving the model, and trained several models on the AWS servers.

References

[1] Zhu, J., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks. 2017 IEEE International Conference on Computer Vision (ICCV). doi:10.1109/iccv.2017.244

[2] Y. Chen, Y.-K. Lai, and Y.-J. Liu, "CartoonGAN: Generative Adversarial Networks for Photo Cartoonization," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 9465–9474.

[3] E. Simo-Serra, S. Iizuka, K. Sasaki, and H. Ishikawa, "Learning to simplify: fully convolutional networks for rough sketch cleanup," ACM Transactions on Graphics, vol. 35, no. 4, pp. 1–11, Jul. 2016.

[4] D. Ha and D. Eck, "A Neural Representation of Sketch Drawings," arXiv:1704.03477 [cs, stat], Apr. 2017.

[5] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal Unsupervised Image-to-Image Translation," arXiv:1804.04732 [cs, stat], Apr. 2018.

[6] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, and N. Fox-Gieg. The Quick, Draw! - A.I. Experiment. https://quickdraw.withgoogle.com/, 2016

[7] Quick, Draw! API: https://quickdraw.readthedocs.io/en/latest/api.html

[8] X. Wang and X. Tang, "Face Photo-Sketch Synthesis and Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), Vol. 31, 2009

[9] R. Zhang, P. Isola, A.Efros, E. Shechtman, and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric" IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 586-595

Appendix

[1] Our data collection scripts: https://github.com/tayo123/CS230-Final-Data-Collectors

[2] Additional code base including AWS setup: https://github.com/ottoman91/DrawBot