Comparing Different Generators of cGAN in Chinese Calligraphy Synthesis

Hsu Shen Lee
1005764585
hsushen.lee@mail.utoronto.ca

Han-Shin Chen
1005403957
henson.chen@mail.utoronto.ca

Abstract

We examine one of the most intuitive methods for calligraphy synthesis: paired image to image translation with cGAN. A series of experiments were conducted, and we found out that (1) the Encoder-Decoder model as a generator outperforms U-net as a generator in this dataset since stroke placemnets differ greatly. (2) batchsize=1 did not perform the best in our dataset even though it was suggested in (5). (3) Adding a more traditional loss to the objective can boost performance, and L_1 performed better than MSE loss. (4) a 16×16 PatchGAN as a discriminator produced better results than the 1×1PixelGAN and the 70×70ImageGAN.

1 Introduction

Chinese calligraphy (書法, Shu fa) is a special visual art that dates back to the Shang Dynasty three thousand years ago. Writing Chinese calligraphy is difficult since the calligrapher also needs to consider the angle and pressure of the brush. It also highlights the importance of the structure of a character. Over the years, millions of different styles emerged, and some of them do not align with the standard font style. In our project, we are interested in finding out methods to synthesize a certain style of Chinese calligraphy using generative deep learning methods.

2 Related Works

Throughout the years, different approaches to calligraphy synthesis have been proposed and tested. (1) use shape decomposition and a hierarchical parametric approach to generate artistic calligraphy. Their modal extracts strokes from known characters and assembles them to form a new character. (2) uses Long Short-term Memory recurrent neural networks to generate complex sequences by predicting a specific data point one at a time, which is applied to generating cursive handwriting. (4) used a similar approach to pix2pix proposed by (5), which is image-to-image translation using Conditional Generative Adversarial Networks.
Among the papers regarding Chinese calligraphy generation, there is a variety of different representations of the training data. For instance, (1) chose to generate handwritten Chinese characters conditioned on their GBK encodings. (4), on the other hand, used an image-to-image pairing approach that's based on pix2pix. Similarly, (6) also use an image-to-image translation, but they used unpaired Chinese characters instead of paired ones. This is because they highlight the importance of learning the overall calligraphy style instead of imitating the individual characters.

3 Dataset

The calligraphy images are from an open sourced Chinese calligraphy database "Chen Zhong Jian Calligraphy Character Database" （陳忠建書法字庫） (7), which offers thousands of samples of Chinese calligraphy font in various styles. The style we picked is the unique style of *Zhi Yong*, which is known for its round, fat strokes and high contrast in thickness – thus making synthesising more difficult. We first collected 4799 entries using web scraping methods. Next we grayscaled, padded, and cropped the images to 140×140 pixels since they come in various sizes but their max width and length does not exceed 140 pixels. We then resize them to 64×64 pixels for easier training. After concatenating each image (ground-truth) with their corresponding character in NotoSansTC-Regular font (input), we divide them into training and test sets in a 8:2 ratio.

4 Experiment: Variants of cGAN

We extend our research from (5), using image to image with paired training. The main difference is that we made changes accordingly for the new dataset based on Chen Zhong Jian Calligraphy Character Database （陳忠建書法字庫） (7). We are particularly interested in the performance of two types of generators: the encoder-decoder model and the U-net model. As suggested by other papers (8) (9) and in Assignment 2, the U-net model performs better in preserving low-level information compared to the encoder-decoder model, where each layer only connect to the next layer. Since the stroke placement of *Zhi Yong's* style and the NotoSansTC-Regular font should be similar, we predict that using U-net as a generator will produce better results.

4.1 Objective

The objective of a conditional GAN is as follows:

$$\mathcal{L}_{cGAN}(G, D) = \mathbf{E}_{x,y}(logD(x, y)) + \mathbf{E}_{x,y}(1 - logD(x, G(x, z)))$$

where x is the input, y is the output, z is a Gaussian noise, and the generator G tries to minimize this objective while the discriminator D tries to maximize it. In $\mathbf{E}_{x,y}(logD(x, y))$ the discriminator tries to identify real images, while in $\mathbf{E}_{x,y}(1 - logD(x, G(x, z)))$ it tries to identify images generated by the generator as fake. This can also be expressed as: $min_G max_D \mathcal{L}_{cGAN}(G, D)$ (5). In the same paper, better performance was acquired by appending a more traditional loss to the objective. During the experiements it was discovered that L1 produced better results, which aligned with (5)'s findings. The L1 term can be expressed as: $\mathcal{L}_1(G) = \mathbf{E}\|y - G(x, y)\|_1$. Hence we rewrite our objective as follows:

$$min_G max_D \mathcal{L}_{cGAN}(G, D) + \mathcal{L}_1(G)$$

4.2 Generator (U-Net and Encoder-Decoder)

In this paper, we tested with two generators: an encoder-decoder model and an U-net. In the U-net architecture, data will bypass parts of the downsampling process and preserve some of the spatial information by adding layers between layer i and $n - i$, $\forall i \in [1..\lfloor n/2 \rfloor]$, where n is the number of layers. While in the encoder-decoder model, it follows a single path in which the data is first encoded (downsampled) into a smaller hidden layer, and then decoded (upsampled) back to a larger layer. Each block in our model is in the form convolution - batchnorm - relu. For more details see Appendix A.6.

4.3 Discriminator

A PatchGAN was used as a discriminator. It earned its name since each pixel of its output looks at a $N \times N$ patch of a image, and identifies it as real or fake. It is essentially a CNN which consists of convolution blocks, batchnorms, and activation functions. Due to its successfullness in (10) (5) in addition to its ease of development, we selected it as our discriminator. Each block is in the form convolution - batchnorm - leakyRelu. For more details see Appendix A.7.

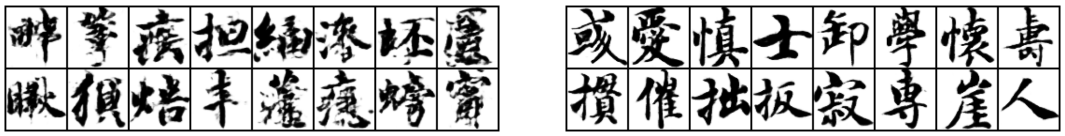


Figure 1: Left: Images produced by U-Net as a generator. Right: Images Produced by Encoder-Decoder algorithm as a generator. It is discovered that holding everything else constant, a Encoder-Decoder generator leads to better results.

4.4 Optimization

Most optimization process were adapted from (5). First, instead of training the generator to minimize $(1 - logD(x, G(x, z))$, we train it to maximize $logD(x, G(x, z))$. Next, we use Adam optimization algorithm along with minibatch, with a learning rate of 0.0002, $\beta_1 = 0.5$, and $\beta_2 = 0.999$.

5 Evaluation Metrics

The evaluation of cGANs have always been difficult. As stated in (11), distinct quantitative metrics have been established in response to specific properties. Different metrics result in different models, which leads to difficulty in model benchmarking. For example, per pixel MSE doesn't work since we are more concerned about the spital behavior of the model. To over come this, we rely on the human eye to check if the generated characters are readable by a native Mandarin speaker. In addition, we plot four graphs to assist the evaluation of the model – the ability for a discriminator to identify the ground truth image, the ability for it to identify the generated image, discriminator loss, and generator loss, where the x-axis is the number of iterations.

6 Analysis and Discussion

The following experiments were conducted on google colab and took about 300 compute power on a premium GPU environment.[1] [2]

6.1 U-Net Vs. Encoder-Decoder

In Figure 1 above, it was discovered that using an encoder-decoder model as the generator produced better results. This is surprising since the U-net should preserve more spatial information (the placement of strokes). After doing research, we found out that in fact, U-net is doing its job *too* well. The U-net is good for colorization problems since it preserves the geometrical information better. However in our case, we are transforming the NotoSansTC-Regular font to another style with different stroke placements. Although they have a similar structure, the geometric shape differs greatly. This is also the reason why characters with the component have clearer results when using U-net as a generator – it has a similar geometric shape in both styles. In the following sections, we will conduct experiments using U-net as a generator since the modifications didn't have any observable difference when using the Encoder-Decoder model.

6.2 Batchsize

Fixing number of data to be 1500 and using L1 loss while using a 70×70 PatchGAN, we experimented batchsizes of 1, 4, 16, 32 with 200 epochs. In Figure A.1, we see that increasing batchsize results in stable, but not necessarily better performance. When batchsize is 4, the discriminator is doing its job well since there are constantly high values of D_real and low values of D_fake. (D_real indicates the probability the discriminator classifies a real image as real, and D_fake is the probability that it classifies a fake image as real). However the reason behind this might be that the generator isn't

[1]The code can be found here: `https://github.com/drunkint/Pix2Pix-Calligraphy`

[2]The code was inspired by (12), used under the MIT license

doing well, similar to what happened in Figure A.2 when batchsize=1. We see that in this case the final objective loss oscillates dramatically. This is different from (5)'s findings as they proposed that batchsize=1 is producing the best results. We also observe a "U and upside-down U" curve in Figure A.1 where batchsize=16 and 32. This is because at first the generator is doing badly, so the discriminator catagorizes well (this is also the reason we divide the discriminator loss by two to prevent from the discriminator learning too quickly). But then the generator learns to generate, so both D_real and D_fake are near 0.5. On the other hand, in the long run, the discriminator is again overtaking the generator gradually.

6.3 L1 or L2

Fixing number of data to be 4799 and batchsize to be 16 while using a 16×16 PatchGAN, we experimented L1 and L2 loss for 200 epochs. As shown in Figure A.3, this behavior is surprising since it didn't happen when dataset size equals to 1500. When the generator used L1 loss, we see that the discriminator had a fairly challenging time classifying: sometimes D_real and D_fake crossed the 0.5 line. This shows that the generator is producing better results. On the other hand, when the generator used L2 loss, the discriminator classifies better: around 100 steps, it is seen that the discriminator has an accuracy of more than 0.9! (D_real went over 0.9 and D_fake went under 0.1). We conclude that L1 loss is better for image generation.

6.4 PatchGAN varients in the discriminator

Fixing number of data to be 4799 and batchsize to be 16 with L1 loss, we experimented 1×1 "PixelGAN", 16×16 "PatchGAN", and 70×70 "ImageGAN". Looking at PixelGAN at Figure A.4, there is a lot of intersection between D_fake and D_real, which means that the discriminator classified real and generated images poorly. PatchGAN produced an acceptable result: it classifies most of the images in an uncertain manner, and the D_real and D_fake likes remained parallel. This means that the discriminator and the generator are evolving at a similar speed. The ImageGAN produced interesting results: it classifes some of the images correctly in a certain manner and some of them totally wrong. This shows its inability to classify images.

6.5 Limitations

While the results look good, some limitations persists. For example, we seem to hit a plateau at around loss=3.4. The quality of the pictures generated at epoch 30 looked similar to those generated at epoch 200. In addition, this method is useful only when data are paired together, which most of the time is not the case.

7 Conclusion

The training of GANs have always been a non-trivial task for a variety of reasons, including non convergence and the unbalance between the generator and the discriminator, which were found in some experiments above. Through the above experiments, we discovered that the Encoder-Decoder generator outperforms the U-Net one. In addition, we found out that batchsize=1 isn't the best choice in this dataset, even though it was suggested by (5). We also observed a "U and upside-down U" curve for D_real and D_fake in Figure A.1. [3] for larger batchsizes. Another thing about generators is that L1 loss worked better than L2 loss. For the discriminator, it was discovered that a 16×16 PatchGAN works better than 1×1 PixelGAN and 70×70 ImageGAN.

[3]D_real indicates the probability the discriminator classifies a real image as real, and D_fake is the probability that it classifies a fake image as real.

References

[1] S. Xu, F. C. M. Lau, W. K. Cheung, and Y. Pan, “Automatic Generation of Artistic Chinese Calligraphy,” IEEE Intelligent Systems, vol. 20, no. 3, pp. 32–39, May 2005, doi: `https://doi.org/10.1109/mis.2005.41`.

[2] A. Graves, “Generating Sequences With Recurrent Neural Networks,” arXiv.org, 2013. `https://arxiv.org/abs/1308.0850` (accessed Apr. 18, 2023).

[3] W. Kong and B. Xu, “Handwritten Chinese Character Generation via Conditional Neural Generative Models,” www.semanticscholar.org, 2018. `https://www.semanticscholar.org/paper/Handwritten-Chinese-Character-Generation-via-Neural-Kong-Xu/17e9103812d375f06663e1802dad02d24c28c998` (accessed Apr. 19, 2023).

[4] Y. Tian, “kaonashi-tyc/zi2zi,” GitHub, Apr. 29, 2020. `https://github.com/kaonashi-tyc/zi2zi` (accessed Apr. 18, 2023).

[5] P. Isola, J.-Y. Zhu, T. Zhou, and Efros, Alexei A, “Image-to-Image Translation with Conditional Adversarial Networks,” arXiv.org, 2016. https://arxiv.org/abs/1611.07004

[6] B. Chang, Q. Zhang, S. Pan, and L. Meng, “Generating Handwritten Chinese Characters using CycleGAN,” arXiv.org, Jan. 25, 2018. `https://arxiv.org/abs/1801.08624` (accessed Apr. 19, 2023).

[7] 陳忠建, “陳忠建-製作- 書法教學資料庫,” 163.20.160.14, 2001. `http://163.20.160.14/~word/` (accessed Apr. 19, 2023).

[8] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, “Context Encoders: Feature Learning by Inpainting,” arXiv:1604.07379 [cs], Nov. 2016, Accessed: Apr. 19, 2023. [Online]. Available: https://arxiv.org/abs/1604.07379

[9] X. Wang and A. Gupta, “Generative Image Modeling using Style and Structure Adversarial Networks,” arXiv:1603.05631 [cs], Jul. 2016, Accessed: Apr. 19, 2023.[Online]. Available: https://arxiv.org/abs/1603.05631

[10] U. Demir and G. Unal, “Patch-Based Image Inpainting with Generative Adversarial Networks,” arXiv:1803.07422 [cs], Mar. 2018, Accessed: Dec. 14, 2022. [Online]. Available: https://arxiv.org/abs/1803.07422

[11] T. DeVries, A. Romero, L. Pineda, G. W. Taylor, and M. Drozdzal, “On the Evaluation of Conditional GANs,” arXiv:1907.08175 [cs, eess, stat], Dec. 2019, Accessed: Apr. 19, 2023. [Online]. Available: https://arxiv.org/abs/1907.08175

[12] A. Perrson, “Pix2Pix implementation from scratch,” www.youtube.com, Mar. 09, 2021. https://www.youtube.com/watch?v=SuddDSqGRzg

8 Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes]

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

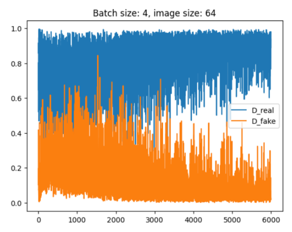
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

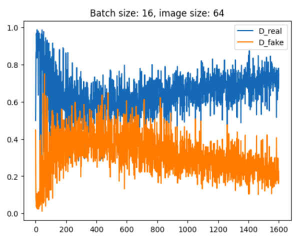
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

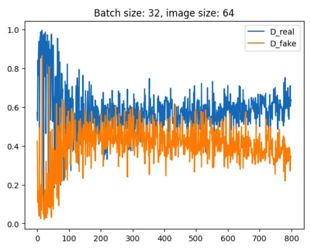
A Appendix

A.1 For different Batchsizes

Plots of D_real (blue) and D_fake (orange) for different batchsizes (4, 16, 32), which represents the probability of an image to be real on ground truth (blue) and generated images (orange). D_real indicates the probability the discriminator classifies a real image as real, and D_fake is the probability that it classifies a fake image as real.

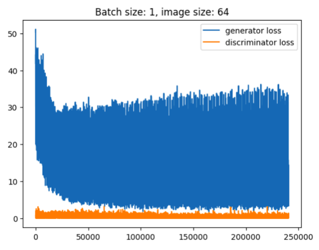


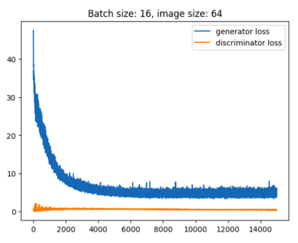


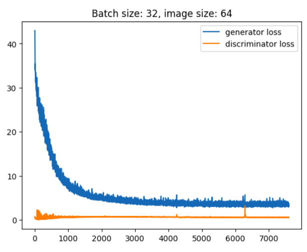


A.2

Plots of discriminator loss and final objective for different batchsizes (1, 16, 32).

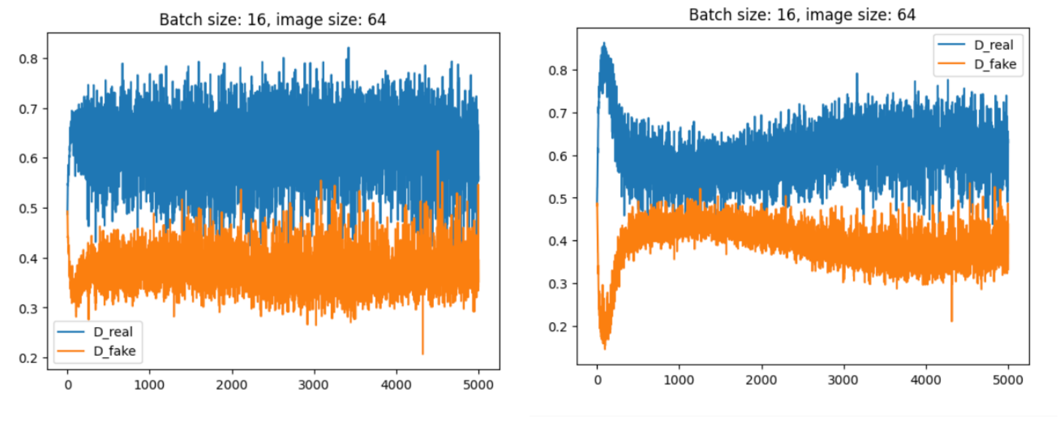






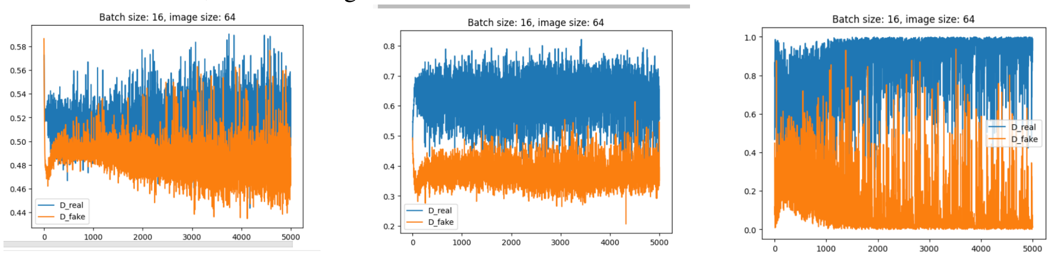
A.3

Left: L1. Right: L2. The L1 graph is surprising since this behavior didn't happen when datasize is 1500.



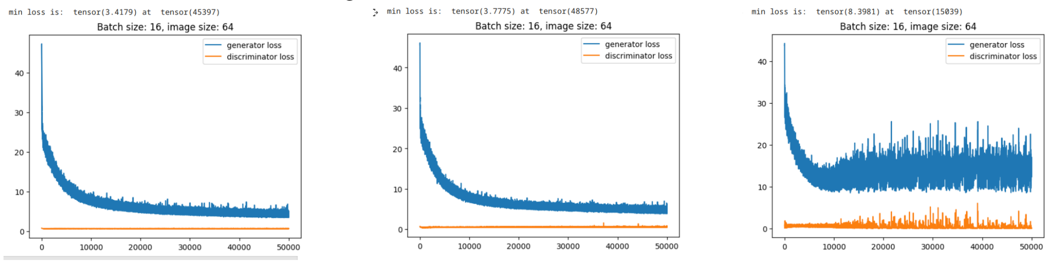
A.4

Plots of D_real (blue) and D_fake (orange) for different $N \times N$ PatchGAN.(1×1ImageGAN, 16×16PatchGAN, 70×70ImageGAN



A.5

Plots of discriminator loss and final objective for different $N \times N$ PatchGAN.(1×1ImageGAN, 16×16PatchGAN, 70×70ImageGAN



```
min loss is:  tensor(3.4179) at  tensor(45397)
min loss is:  tensor(3.7775) at  tensor(48577)
min loss is:  tensor(8.3981) at  tensor(15039)
```

A.6 Generator details

Abiding to the conventions of (5), we let C_k be a Convolution-batchNorm-Relu layer, and CD_k to be a Convolution-batchNorm-dropout-Relu layer. Our final model used a Encoder-Decoder model with the following structure:
encoder: C32, C64, C128, C256, C256, C256
decoder: C256, C256, C128, C64, C32
All layers in the encoder are leaky, all layers in the decoder are not, all of them with negative slope 2. We also don't apply batchNorm in the first layer of the encoder and decoder. Kernel size is 4, stride 2, padding 1. Padding mode is reflect in the encoder. For the U-net structure we just add connections

between layers i and |layers| - i.
Code can be accessed here: `https://github.com/drunkint/Pix2Pix-Calligraphy`

A.7 Discriminator details

Abiding to the conventions of (5), we let C_k be a Convolution-batchNorm-Relu layer, and CD_k to be a Convolution-batchNorm-dropout-Relu layer. Our final model used a PatchGAN with the following structure:
The 16×16 PatchGAN structure is: C32, C64
For PatchGAN used in experiments we used the same structure but with the following modifications:
1×1 **PixelGAN:** C32, C64 (kernel_size is set to one)
70×70 **ImageGAN:** C32, C64, C128, C256
We don't apply batchNorm in the first layer. LeakyRelu was applied to everything. At the end we append a convolutional layer which leads to only one channel in the output. Code can be accessed here: `https://github.com/drunkint/Pix2Pix-Calligraphy`