Deep Convolutional Generative Adversarial Networks on handwritten Chinese Characters

Haoran Ma (hm1651), Aotian Yang (ay954)

***Abstract*—Convolutional Neural Networks (CNNs) in supervised learning field has attracted attention and adoption. However, application of CNNs in unsupervised learning has been less covered. In 2014, Ian J. Goodfellow and his team invented Generative adversarial network (GAN) which incorporates game theory thinking in developing neural networks. Alec Radford and Luke Metz introduced a class of CNNs named Deep Convolutional Generative Adversarial Networks (DCGANs), which are strong in unsupervised learning. Handwritten Chinese characters Learning is a significant part in representation learning. We applied our DCGAN pair to teach Chinese characters writing, including a single character, a set of characters with the same part and general Chinese character writing.**

I. Introduction

A DCGAN is a type of Convolutional Neural Network which is capable of learning an abstract representation of a collection of images. This method has a ”generator” which fabricates fake images and a ”discriminator” which tries to discern if the generator’s images are authentic. After training, the generator can be used to convincingly generate samples from the originals.

Before DCGAN, GANs provide an attractive alternative to maximum likelihood techniques. One can additionally argue that their learning process and the lack of a heuristic cost function (such as pixel-wise independent mean-square error) are attractive to representation learning [1]. However, GANs are very unstable to train, sometimes it takes long time to converge. In 2016 Alec Radford and Luke Metz propose DCGAN which is more stable than GAN. After these algorithms are invented, it has been used in many practical applications.

Chinese characters is one of the most complex characters in our time. There are two kinds of them. One is a single character which has its meaning by itself, such as 女 (female). The other one is complex characters which consist of parts, such as 妈 (mother). We teach computer both kinds of them and further general character writing.

II. Related Work

1. A book from sky 天书
   In December 2015, digital artist Gene Kogan used DCGAN to train a dataset and created artworks based on the training result [2]. He did pretty well on some most frequently used Chinese characters such as 是 (is) and 在 (in). He has also explored the neighborhood of an individual character, and linguistic algebra (word vectors) on some of the most used characters. But he didn’t publish his dataset, code and network architecture.
2. AnimeGAN
   An open source software contributor Jie Lei generated animation characters’ faces using DCGAN in 2017 [3]. In his cases, more layers yield better images, in the sense that the generator is more powerful than discriminator. Also, he stated, adding noise to discriminator’s inputs and labels helps training.
3. Generating images from standard dataset
   Hyeonwoo Kang has applied several GAN and DCGAN models on MNIST and CelebA dataset. He achieves good results by training approximately 20 epochs. He also outlines the histogram of epochs with generator loss and discriminator loss [4].

Professor: Rob Fergus, Tutor: Rohit Muthyala

III. Method

DCGAN network architecture:

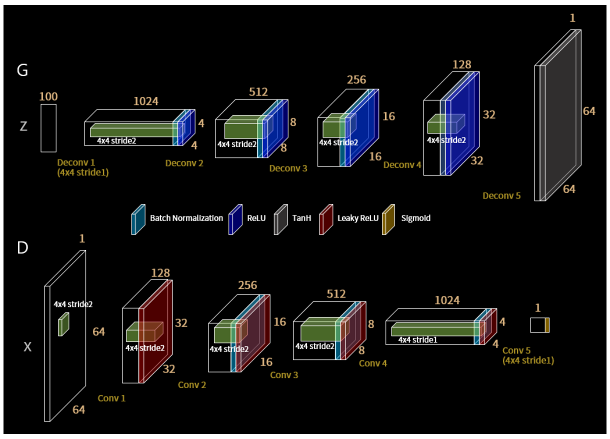


Fig. 1. DCGAN network: G and D

We referred the model of the picture from Hyeonwoo Kang’s work, and we used techniques from Dev Nag’s article [5].

Both G (generator) and D (detector) have four convolutional hidden layers. The layers of G is almost a reverse of D, except for G’s input is a noise vector while D’s output is a scaler(0 for fake, 1 for genuine). The input size of D and output size of G is 64 by 64, with 1 channel for grey scale picture or 3 channels for a color picture.

The hidden layer dimensions are determined by a parameter d. In the picture above, d is 128. This dimension defines the number of channels of the last hidden layer(the one before output) of G and the first hidden layer of D. In G, each preceding hidden layer has twice hidden layer channels as many as that of the following layer. In D, each following layer has twice channels as many as that of the preceding one.

From one layer to another, if the channel number doubles, the size halves, and vice versa. For example, in G, the layer with 128 channels and size of 32 by 32 follows the layer with 256 channels but which is only 16 by 16 in size.

IV. Experiments

A. Data

We used open source data from Harbin Institute of Technology Opening Recognition Corpus for Chinese Characters (HIT-OR3C). They have four categories: Offline Characters (807 Mb), Offline Documents (120 Mb), Online Characters (146 Mb), Online Documents (21 Mb). We choose 'Offline characters' as our database which contains handwritten Chinese characters in multiple binary files. These handwritten characters are collected from 122 individuals. Each character image is of 128*128 pixels. We made small changes on loading this data, transferring it to pytorch dataset. Since we wanted to train Chinese characters writing, we put single character (i.e 大) of all sampling writers into one file. Also, we explored putting all the characters into a dataloader and see if generator is able to learn Chinese characters in a high level. Learning with all sample characters may store the connection among these characters.

Samples from this dataset:

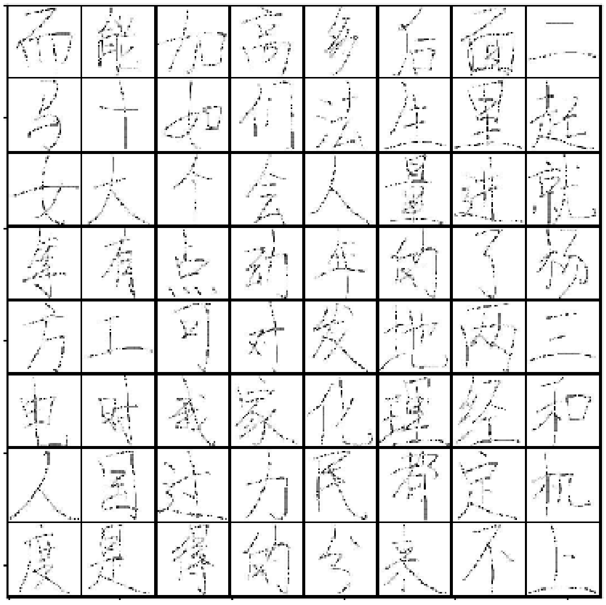


Fig. 2. Samples handwritten characters from dataset

B. Experiment 1 - learning with 140 characters

1) Motivation and setting: An interesting topic we wanted to explore is: can DCGAN generate new Chinese characters that we have never seen before?

Motivated by this question, we chose 140 most commonly used Chinese characters for training. Each character has 122 handwritings, hence there are 17080 training data in total.

2) Learning parameters: We chose $d = 128$ for both G and D. The batch size was 128 and learning rate was 0.0002. We set number of epochs as 200. We had the satisfying results when epoch was 100, so we stopped there.

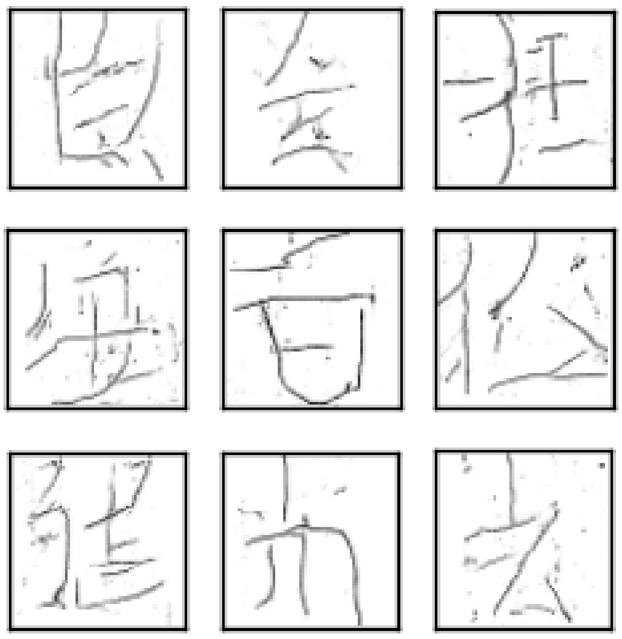


Fig. 3. Examples of DCGAN generated characters

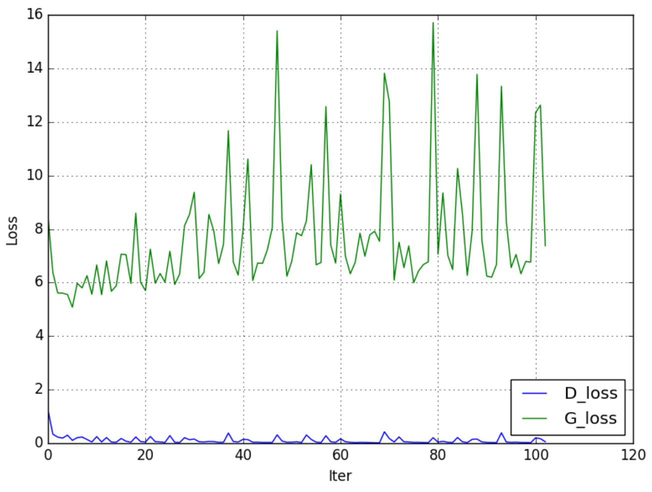


Fig. 4. G and D loss during training of experiment 1

3) Results: Figure.3 shows some examples of generated characters by DCGAN. Some of them successfully resembles real ones: the upper middle character looks like 全 (means ”total”), but has 三 instead of 王 under 人. The central left character looks like 纽 (usually represents for the ”New” in ”New York”), and the central middle one looks like 百 (hundred).

Figure.4 shows the D and G loss during each epoch. D keeps low loss but G seems unstable. During the low loss epochs, G is able to generate good fake characters, while in the peak loss epochs, the generated pictures are vague. So we stopped when output of an epoch is satisfying.

C. Experiment 2 - learning with characters with a specific part

1) Motivation and setting: A considerable part of Chinese characters are called ”Phono-semantic compound characters”, or ”radical-phonetic characters”. (Wikipedia says that they form the majority of Chinese characters by far over 90%.). They have a phonetic component, also called ”rebus”, and a semantic component, also called a determinative.

A determinative is one of a limited number of determinative characters which supplied an element of meaning. A rebus is another character with approximately the correct pronunciation, but has nothing to do with the meaning.

For example, character 女 means “woman” or ”female”. 女 is also a determinative. Characters with 女 determinative often has a meaning related with female or family (because these characters were created when a matriarchal society). For example, 妈 (mother)= 女 + 马 (horse), 妹 (younger sister)= 女 + 未 (not yet), 婚 (marriage)= 女 + 昏 (faint), 姓 (family name)= 女 + 生 (living). You can see, the second part usually has nothing to do with the meaning, actually, they only signify the pronunciation, and are the rebuses therefore.

In the second experiment, we wondered, if given all characters with the same component (e.g. a determinative), whether DCGAN can learn to create another character with the same component.

For this aim, we selected 90 characters with the determinative 女. We used these 10980 images in total for training.

2) Learning parameters: We chose $d = 128$ for both G and d. The batch size was 128 and learning rate was 0.0002. We set number of epochs as 200 but we stopped the training at epoch 160 as we saw the D became overfitting.

3) Results: The results were not good. Most generated characters do not show a clear determinative 女. We think this is because the handwriting is not a good source for learning this. Different person wrote this determinative in different way, and the same person may writes it differently when the other component differs (to make a handwriting good-looking, some people like to adjust the components so they ”fit” each other better).

But not all results were bad, we still found some interesting results.

Figure. 5 picks 3 examples which seems to have determinative 女. The first one looks like traditional Chinese character 媽 (mother), the third one looks like traditional character $\boxed{F}$ (you). Note that because our handwriting dataset contains only simplified characters, 媽 and $\boxed{F}$ are not in our training dataset.

Figure. 6 shows a strange phenomenon: at epoch about 45, the loss of D suddenly becomes very high and loss of G becomes very high.

At first, we could not figure out why. After some exploration in the next experiment, we believe that this is because of the over-fitting of D. This finding led us to decrease d of D in experiment 3.

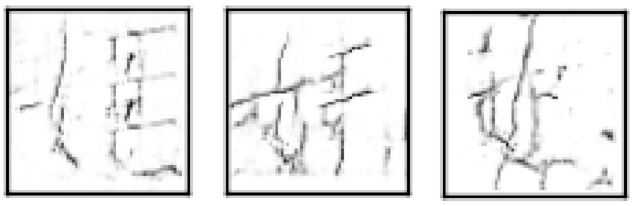


Fig. 5. Some DCGAN generated characters with determinative 女: 媽, 妹, $\boxed{F}$

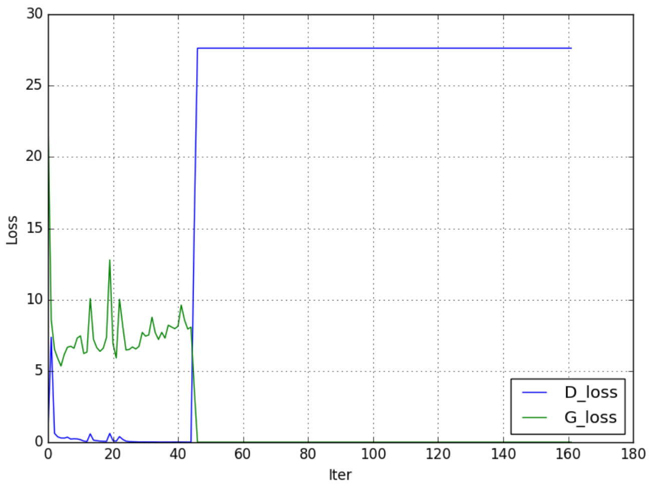


Fig. 6. D and G loss during training with characters with determinative 女

D. Experiment 3 - learning with a specific character

1) Motivation and setting: Creating a new character is hard to judge how good the performance is, as we have never seen the new character before. The third experiment is, therefore, rewriting an existing character.

For this experiment, in each sub-experiment, we only chose one character(each of which has 122 image data). We tried several characters, most of them showed not bad results. In the results section, we will show you some examples among them.

2) Learning parameters: We chose $d = 128$ for both G and $d = 64$ for D. The batch size was 128 and learning rate was 0.0002. Because for one character there are only 122 training data, We used a large number of epochs 20000. But we stopped the training whenever we saw the results were satisfying and the G net was not improving any more.

3) Results: Table 1 shows the learning results of 3 sub-experiment in experiment 3: a medium difficulty level character 我 (I), an easy one 马 (horse, or family name Ma), and a hard one 杨 (poplar, or family name Yang).

The results are actually very good. The generated characters do contains a large number of ones that resemble real hand written characters.

Figure.7 shows the loss of G and D. D's loss still keeps low. And G becomes stable after fluctuation for hundreds of epochs.

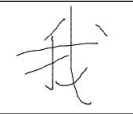
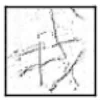
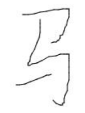
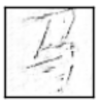
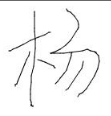
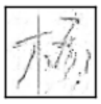
Computer fonts	Handwritten	DCGAN generated
我		
马		
杨		

TABLE I
COMPARISON BETWEEN COMPUTER FONTS, HANDWRITTEN AND DCGAN GENERATED CHARACTERS

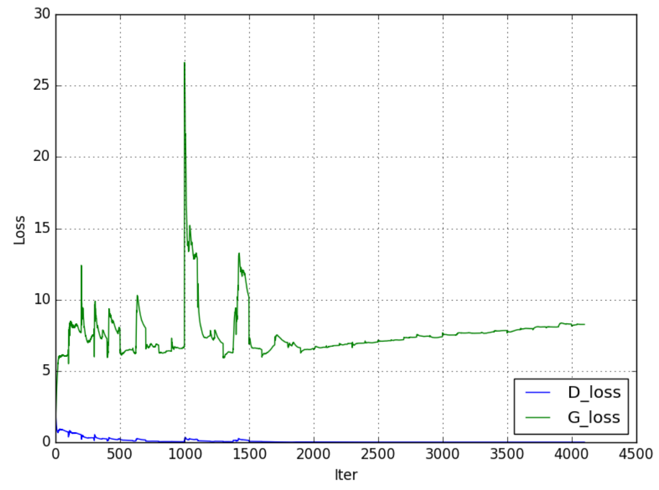


Fig. 7. G and D loss during training of character "我" of experiment 3

E. obstacles encountered and how to solve

We were faced with some obstacles during the experiments, and we found our way to overcome them.

One problem is that if the net work d is too small, G only outputs noise. The first several tries of us used very small d because we thought the data was simple and we wanted faster training. But it turned out that a larger d was necessary for our experiments, and we finally enlarged d and did our experiments on NYU HPC GPU clusters for faster training.

Another problem we have met was that the G may become over-fitting and the G cannot fool D no matter how hard it tries, as described above in experiment 2. A solution to this is enlarging G's or decreasing D's d: you can use a larger d for G than that for D to make the G more powerful than D.

The third lessen we learned is: detaching G during training D with D's output is important for a better G results. At first we forgot the detaching, and the D's output was always vague and ugly.

V. Discussion

In our paper, we present our training results on three levels of learning Chinese characters: single specific character, a set of characters with the same component and general characters. Effect of our training decreases when input gets more complicated. This may due to the lack of handwritten characters from a larger group of people. Besides, in our network, we heavily used convolution layer. This may be too general for Chinese characters because most Chinese characters consist of independent components. In future we can try R-CNN, which can train characters in segments. Last, our inputs are mostly simplified Chinese characters. We can also apply our model to traditional Chinese characters which have much more complicated components and structures.

VI. Description of our code

Open the code.zip you should see three folders. In src folder, you can find five .py files.

readimage.py is for loading HIT-OR3C image files and generate .jpg files which we can use for the training.

main.py is what you run for the training. It uses 3 helper modules, which are in data.py, model.py, and preprocess.py. This code refers to the work of Kang[4].

run.sh provides how we run the two scripts on NYU HPC.

Before running, you should download HIT-OR3C dataset and put the n_images files in the folder character_images and label and label.txt files in folder character_images_labels. Also, in chars.txt in folder character_images_labels, you should type in the characters you want to use for training. It can be a single character, or multiple separated by space. If you don't want to use space, you can type characters continously, but remember to edit srcread_image.py: in the final part under "if ___name___ == "___main___":", change "selected_char = getCharInd(Space=True)" to "selected_char = getCharInd(Space=False)".

References

[1] A. Radford, L. Metz, and S. Chintala. Unsupervised Representation Learning With Deep Convolutional Generative Adversarial Networks. arXiv preprint arXiv:1511.06434, 2015.

[2] Gene Kogan. A Book from the Sky 天书: Exploring the Latent Space of Chinese Handwriting. http://genekogan.com/works/a-book-from-the-sky/.

[3] Jie Lei. AnimeGAN https://github.com/jayleicn/animeGAN.

[4] Hyeonwoo Kang. MNIST-CelebA-GAN-DCGAN https://github.com/znxlwm/pytorch-MNIST-CelebA-GAN-DCGAN.

[5] Dev Nag. Generative Adversarial Networks (GANs) in 50 lines of code (PyTorch) https://github.com/devnag/pytorch-generative-adversarial-networks.