

README

Link to repository: <https://github.com/signelaijing/coding3>

Link to video demonstration: https://youtu.be/C9Fa9R_fd6s

CONTEXT OF PROJECT

This project is inspired by the nüshu characters of the Jiangyoung dialect, Yang Zhuang (often called tuhua, but that term refers to an inferior language) in the province of Hunan, China. The word, translating literally to ‘women’s script’, was developed by and for women in the Shang or Song dynasty, a time when women were not allowed inside schools and institutions. (Lofthouse, 2020)

It was used by Han, Maio and Yao people and passed down through generations. Therefore, the variations of the scripts are many as each area and each woman would develop their own version that would pass down through generations. (Lofthouse, 2020) Yang Huanyi, the last fluent speaker of the language died in 2004 and therefore the documentation around the script is very limited.

I am drawn to the fluidity and autonomy that lies in this. The aim of this project is to showcase this. I will do this by training DCGAN using this guide: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html (other resources are referenced in the notebook) and then creating an animation based on the generated images, emphasizes how oral practices and dead languages creates gaps between the people who created it and those today.

GETTING THE DATA

This was the first issue: That there was no dataset available, and the images that I found available to scrape was of very poor quality. I did manage to find unicodes for the script, but I never managed to display it in a notebook even though I tried downloading the .tff files, installing them, converting unicodes and loading in one at a time. The .csv file didn’t even show it either.

```
1 # Unicode code points of Nushu characters
2 nushu_codes = [
3     0x1B18B, 0x1B18C, 0x1B18D, 0x1B18E, 0x1B18F,
4     0x1B190, 0x1B191, 0x1B192, 0x1B193, 0x1B194
5 ]
6
7
8 # Display Nushu characters
9 for code in nushu_codes:
10     character = chr(code)
11     print(f"Code point: {hex(code)} Character: {character}")
12
```

Code point: 0x1b18b Character: ☐
Code point: 0x1b18c Character: ☐
Code point: 0x1b18d Character: ☐
Code point: 0x1b18e Character: ☐
Code point: 0x1b18f Character: ☐
Code point: 0x1b190 Character: ☐
Code point: 0x1b191 Character: ☐
Code point: 0x1b192 Character: ☐
Code point: 0x1b193 Character: ☐
Code point: 0x1b194 Character: ☐

```
1 df = pd.read_csv('nushu.csv')
2
3 display(df)
```

	女书字符	《字帖》序	对应汉字	江永方言代表发音
0	☐	0	—	i5
1	☐	1	两日入二	na33
2	☐	2	像错七	tsha5
3	☐	3	人	ie21
4	☐	4	八	poe5
...
391	☐	389	博忽撮复嘴腹并嘴	fu5
392	☐	390	谷穀轔歌	ku5
393	☐	391	凤	fang33
394	☐	392	转渐登	tɕye21
395	☐	393	豊	fɛ21

396 rows × 4 columns

I ended up finding an image containing the characters with unicodes in a grid. I asked ChatGPT to generate code that could split the image into rows x columns of the grid, which provided me with 397 images each depicting one character.

Due to this not being enough data to train on, I used the dataset-tools repository that works through the command line to datawrangle. Repo: <https://github.com/dvschultz/dataset-tools>

See following pictures:

```
C:\Users\Bruger>git clone https://github.com/dvschultz/dataset-tools.git
Cloning into 'dataset-tools'...
remote: Enumerating objects: 467, done.
remote: Counting objects: 100% (144/144), done.
remote: Compressing objects: 100% (31/31), done.
remote: Total 467 (delta 124), reused 117 (delta 113), pack-reused 323
Receiving objects: 97% (453/467)
Receiving objects: 100% (467/467), 688.62 MiB | 6.38 MiB/s, done.
Resolving deltas: 100% (269/269), done.

C:\Users\Bruger>cd dataset-tools

C:\Users\Bruger\dataset-tools>pip install -r requirements.txt
Requirement already satisfied: numpy>=1.7.0 in c:\anaconda_3\lib\site-packages (from -r requirements.txt (line 1)) (1.23.5)
Requirement already satisfied: imutils in c:\anaconda_3\lib\site-packages (from -r requirements.txt (line 2)) (0.5.4)
Requirement already satisfied: opencv-python>=4.1.0.25 in c:\anaconda_3\lib\site-packages (from -r requirements.txt (line 3)) (4.7.0.68)
Requirement already satisfied: scipy in c:\anaconda_3\lib\site-packages (from -r requirements.txt (line 4)) (1.10.0)
Collecting mac-tag (from -r requirements.txt (line 5))
  Downloading mac-tag-2020.12.3.tar.gz (1.9 kB)
  Preparing metadata (setup.py) ... done
Collecting psd-tools3 (from -r requirements.txt (line 6))
  Downloading psd-tools3-1.9.2.tar.gz (73 kB)
    73.1/73.1 kB | 3.4 MB/s eta 0:00:00
```

```
PS C:\Users\Bruger> cd dataset-tools
PS C:\Users\Bruger\dataset-tools> python dataset-tools.py -i "C:\Users\Bruger\Documents\UAL\coding3\final_project\data\nushu" -o "C:\Users\Bruger\Documents\UAL\coding3\final_project\data\nushu_final" --rotate --max_size 128
usage: dataset-tools.py [-h] [--verbose] [--force_max]
                        [-i INPUT_FOLDER] [-o OUTPUT_FOLDER]
                        [-p PROCESS_TYPE] [--blur_type BLUR_TYPE]
                        [--blur_amount BLUR_AMOUNT] [--cmin CANNY_MIN]
                        [--cmax CANNY_MAX] [--max_size MAX_SIZE]
                        [--height HEIGHT] [--width WIDTH]
                        [--shift_y SHIFT_Y] [--v_align V_ALIGN]
                        [--h_align H_ALIGN] [--shift_x SHIFT_X]
                        [--scale SCALE] [--skip_tags SKIP_TAGS]
                        [--direction DIRECTION]
                        [--border_type BORDER_TYPE]
                        [--border_color BORDER_COLOR] [--mirror]
                        [--rotate] [--f FILE_EXTENSION]
                        [--keep_name | --numbered]
dataset-tools.py: error: unrecognized arguments: --max_size128
PS C:\Users\Bruger\dataset-tools> python dataset-tools.py -i "C:\Users\Bruger\Documents\UAL\coding3\final_project\data\nushu" -o "C:\Users\Bruger\Documents\UAL\coding3\final_project\data\nushu_final" --rotate --max_size 128
Processing folder: C:\Users\Bruger\Documents\UAL\coding3\final_project\data\nushu
PS C:\Users\Bruger\dataset-tools>
```

```
PS C:\Users\Bruger\dataset-tools> python dataset-tools.py --input_folder "C:\Users\Bruger\Documents\UAL\coding3\final_project\nushu_images" --output_folder "C:\Users\Bruger\Documents\UAL\coding3\final_project\nushu_images_final" --process_type crop_to_square --border_type solid --border_color 255,255,255
```

This produced a dataset of 1588 images.

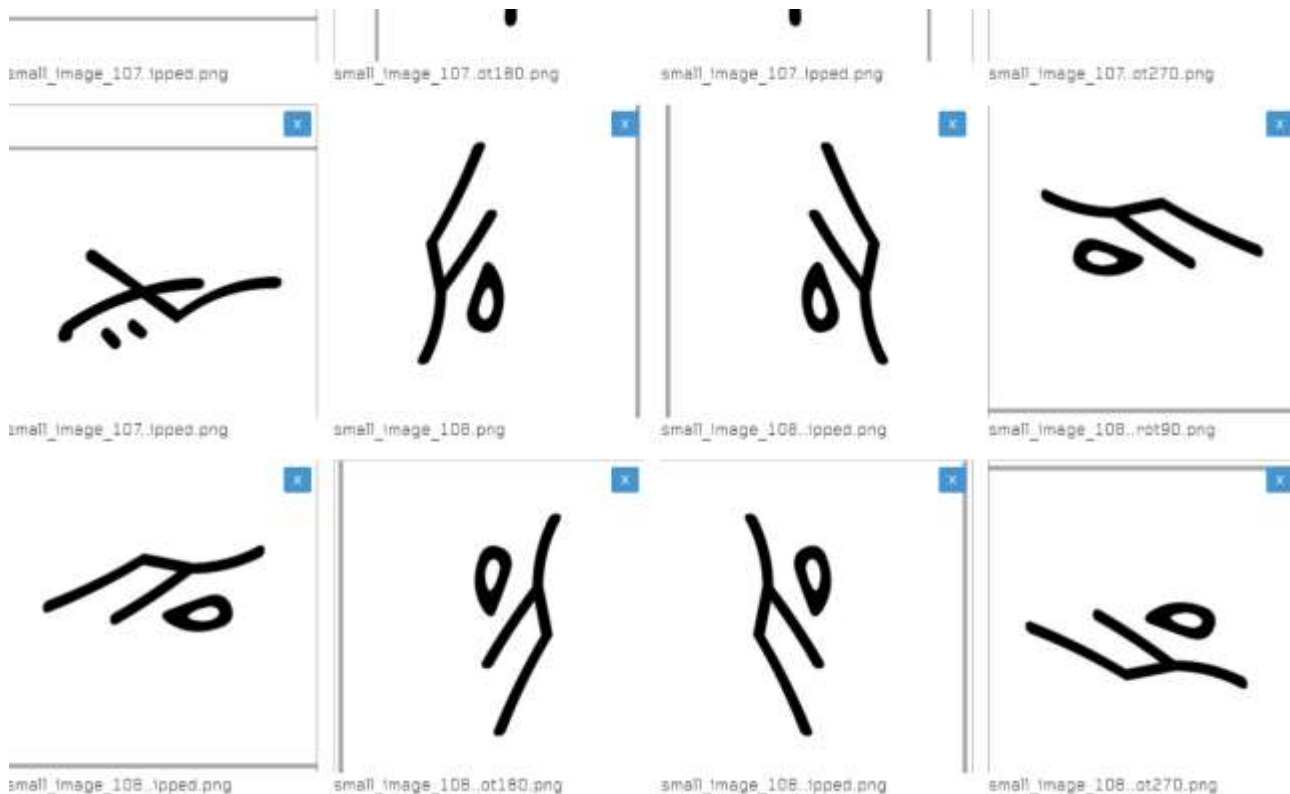
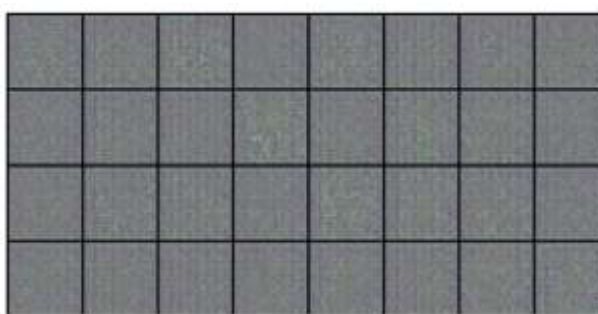
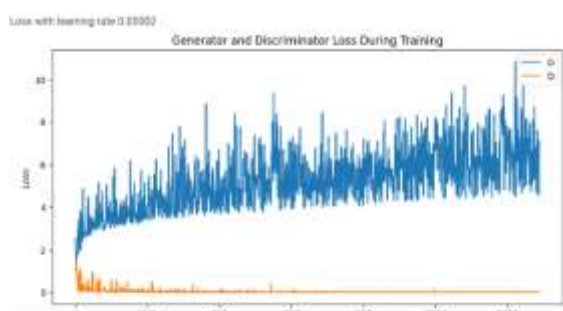


Figure 1 data after wrangling

Then I trained on the DCGAN - see .ipynb notebook for details.

RESULTS

First I got super bad results. The loss output started at around 30 and ended at for the generator and 18 for the discriminator. The output of images was also not good, and the loss for generator and discriminator was horrible.



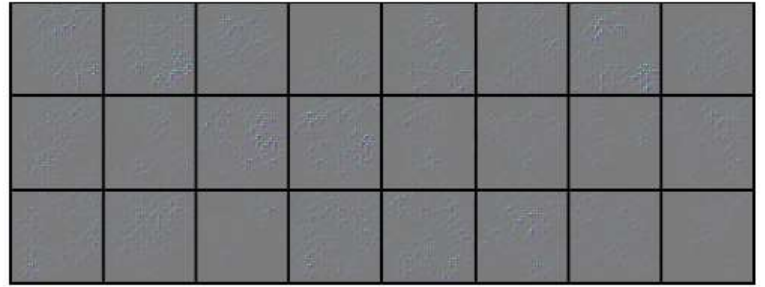
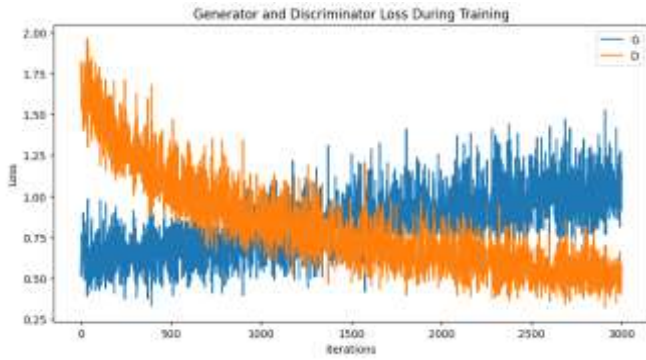
Also the confidence for the discriminator was a 100 percent from the get go for both fake and real images.

[2/60][0/25]	Loss_D: 0.0000	Loss_G: 43.3022 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[3/60][0/25]	Loss_D: 0.0000	Loss_G: 50.5572 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[4/60][0/25]	Loss_D: 0.0000	Loss_G: 44.4393 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[5/60][0/25]	Loss_D: 0.0000	Loss_G: 49.5014 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[6/60][0/25]	Loss_D: 0.0000	Loss_G: 50.1255 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[7/60][0/25]	Loss_D: 0.0000	Loss_G: 46.3725 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[8/60][0/25]	Loss_D: 0.0000	Loss_G: 43.8996 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[9/60][0/25]	Loss_D: 0.0000	Loss_G: 50.2470 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[10/60][0/25]	Loss_D: 0.0000	Loss_G: 45.2102 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[11/60][0/25]	Loss_D: 0.0000	Loss_G: 50.3231 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[12/60][0/25]	Loss_D: 0.0000	Loss_G: 47.9592 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[13/60][0/25]	Loss_D: 0.0000	Loss_G: 48.0354 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[14/60][0/25]	Loss_D: 0.0000	Loss_G: 41.9147 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[15/60][0/25]	Loss_D: 0.0000	Loss_G: 40.2114 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[16/60][0/25]	Loss_D: 0.0000	Loss_G: 46.6363 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[17/60][0/25]	Loss_D: 0.0000	Loss_G: 49.6690 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[18/60][0/25]	Loss_D: 0.0000	Loss_G: 45.7427 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[19/60][0/25]	Loss_D: 0.0000	Loss_G: 48.2730 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[20/60][0/25]	Loss_D: 0.0000	Loss_G: 47.6358 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[21/60][0/25]	Loss_D: 0.0000	Loss_G: 43.8123 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[22/60][0/25]	Loss_D: 0.0000	Loss_G: 47.2833 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[23/60][0/25]	Loss_D: 0.0000	Loss_G: 46.6085 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[24/60][0/25]	Loss_D: 0.0000	Loss_G: 44.6134 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[25/60][0/25]	Loss_D: 0.0000	Loss_G: 49.1727 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[26/60][0/25]	Loss_D: 0.0000	Loss_G: 49.4009 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[27/60][0/25]	Loss_D: 0.0000	Loss_G: 49.0298 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[28/60][0/25]	Loss_D: 0.0000	Loss_G: 39.5415 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[29/60][0/25]	Loss_D: 0.0000	Loss_G: 44.8093 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[30/60][0/25]	Loss_D: 0.0000	Loss_G: 38.3037 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[31/60][0/25]	Loss_D: 0.0000	Loss_G: 43.8271 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[32/60][0/25]	Loss_D: 0.0000	Loss_G: 45.4669 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[33/60][0/25]	Loss_D: 0.0000	Loss_G: 42.3678 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[34/60][0/25]	Loss_D: 0.0000	Loss_G: 46.9083 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[35/60][0/25]	Loss_D: 0.0000	Loss_G: 50.6045 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[36/60][0/25]	Loss_D: 0.0000	Loss_G: 49.7361 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[37/60][0/25]	Loss_D: 0.0000	Loss_G: 45.5256 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[38/60][0/25]	Loss_D: 0.0000	Loss_G: 49.2132 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[39/60][0/25]	Loss_D: 0.0000	Loss_G: 48.8727 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[40/60][0/25]	Loss_D: 0.0000	Loss_G: 40.8095 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[41/60][0/25]	Loss_D: 0.0000	Loss_G: 48.2330 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[42/60][0/25]	Loss_D: 0.0000	Loss_G: 49.1699 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[43/60][0/25]	Loss_D: 0.0000	Loss_G: 49.9594 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[44/60][0/25]	Loss_D: 0.0000	Loss_G: 43.4091 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[45/60][0/25]	Loss_D: 0.0000	Loss_G: 49.9827 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[46/60][0/25]	Loss_D: 0.0000	Loss_G: 49.6020 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[47/60][0/25]	Loss_D: 0.0000	Loss_G: 50.2588 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[48/60][0/25]	Loss_D: 0.0000	Loss_G: 42.8317 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[49/60][0/25]	Loss_D: 0.0000	Loss_G: 44.3081 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[50/60][0/25]	Loss_D: 0.0000	Loss_G: 47.6218 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[51/60][0/25]	Loss_D: 0.0000	Loss_G: 49.8584 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[52/60][0/25]	Loss_D: 0.0000	Loss_G: 47.6777 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[53/60][0/25]	Loss_D: 0.0000	Loss_G: 49.2221 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[54/60][0/25]	Loss_D: 0.0000	Loss_G: 43.7335 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[55/60][0/25]	Loss_D: 0.0000	Loss_G: 49.5041 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[56/60][0/25]	Loss_D: 0.0000	Loss_G: 49.3455 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[57/60][0/25]	Loss_D: 0.0000	Loss_G: 43.1020 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[58/60][0/25]	Loss_D: 0.0000	Loss_G: 43.0064 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000
[59/60][0/25]	Loss_D: 0.0000	Loss_G: 45.8759 D(x): 1.0000	D(G(z)): 0.0000 / 0.0000

This is the results from the final training.

Therefore, I played around with the hyperparameters like the learning rate, the beta1 parameter for the optimizer, the size of the feature maps for both the generator and the discriminator going between 8 and 64 in order to capture more details. I also trained for between 20-60 epochs. I thought of implementing an early stop, but from this website, I learned that it might have very insignificant impact on the training. <https://towardsdatascience.com/10-lessons-i-learned-training-generative-adversarial-networks-gans-for-a-year-c9071159628>

I got the same results each time, with only minor improvements in the aesthetics. However, the loss and confidence did improve a lot.



[1/00][0/50]	Loss_D: 1.5237	Loss_G: 0.6768	D(x): 0.4445	D(G(z)): 0.5098 / 0.5088
[2/00][0/50]	Loss_D: 1.0197	Loss_G: 0.5301	D(x): 0.4831	D(G(z)): 0.5902 / 0.5886
[3/00][0/50]	Loss_D: 1.4053	Loss_G: 0.5019	D(x): 0.5187	D(G(z)): 0.5547 / 0.5533
[4/00][0/50]	Loss_D: 1.5010	Loss_G: 0.5190	D(x): 0.5525	D(G(z)): 0.5905 / 0.5951
[5/00][0/50]	Loss_D: 1.2744	Loss_G: 0.6522	D(x): 0.5850	D(G(z)): 0.5220 / 0.5200
[6/00][0/50]	Loss_D: 1.1634	Loss_G: 0.7127	D(x): 0.6144	D(G(z)): 0.4915 / 0.4903
[7/00][0/50]	Loss_D: 1.0806	Loss_G: 0.7575	D(x): 0.6406	D(G(z)): 0.4703 / 0.4688
[8/00][0/50]	Loss_D: 1.5052	Loss_G: 0.4879	D(x): 0.6039	D(G(z)): 0.6154 / 0.6130
[9/00][0/50]	Loss_D: 1.2202	Loss_G: 0.5672	D(x): 0.6847	D(G(z)): 0.5689 / 0.5671
[10/00][0/50]	Loss_D: 1.1904	Loss_G: 0.5029	D(x): 0.7033	D(G(z)): 0.5711 / 0.5696
[11/00][0/50]	Loss_D: 1.0750	Loss_G: 0.6449	D(x): 0.7202	D(G(z)): 0.5261 / 0.5247
[12/00][0/50]	Loss_D: 0.9472	Loss_G: 0.7527	D(x): 0.7353	D(G(z)): 0.4726 / 0.4711
[13/00][0/50]	Loss_D: 0.9503	Loss_G: 0.7282	D(x): 0.7404	D(G(z)): 0.4840 / 0.4826
[14/00][0/50]	Loss_D: 1.1262	Loss_G: 0.5568	D(x): 0.7624	D(G(z)): 0.5747 / 0.5730
[15/00][0/50]	Loss_D: 0.9251	Loss_G: 0.7209	D(x): 0.7741	D(G(z)): 0.4878 / 0.4863
[16/00][0/50]	Loss_D: 0.7985	Loss_G: 0.8600	D(x): 0.7850	D(G(z)): 0.4210 / 0.4198
[17/00][0/50]	Loss_D: 1.0052	Loss_G: 0.6210	D(x): 0.7945	D(G(z)): 0.5393 / 0.5374
[18/00][0/50]	Loss_D: 0.9345	Loss_G: 0.6747	D(x): 0.8032	D(G(z)): 0.5100 / 0.5093
[19/00][0/50]	Loss_D: 0.8756	Loss_G: 0.7239	D(x): 0.8113	D(G(z)): 0.4864 / 0.4848
[20/00][0/50]	Loss_D: 1.0316	Loss_G: 0.5748	D(x): 0.8185	D(G(z)): 0.5645 / 0.5628
[21/00][0/50]	Loss_D: 0.8002	Loss_G: 0.7212	D(x): 0.8254	D(G(z)): 0.4874 / 0.4862
[22/00][0/50]	Loss_D: 0.6809	Loss_G: 0.9400	D(x): 0.8318	D(G(z)): 0.3915 / 0.3903
[23/00][0/50]	Loss_D: 0.8671	Loss_G: 0.7001	D(x): 0.8376	D(G(z)): 0.4904 / 0.4895
[24/00][0/50]	Loss_D: 0.7833	Loss_G: 0.7836	D(x): 0.8432	D(G(z)): 0.4581 / 0.4568
[25/00][0/50]	Loss_D: 0.8623	Loss_G: 0.6021	D(x): 0.8483	D(G(z)): 0.5023 / 0.5005
[26/00][0/50]	Loss_D: 0.8195	Loss_G: 0.7394	D(x): 0.8533	D(G(z)): 0.4708 / 0.4774
[27/00][0/50]	Loss_D: 0.7632	Loss_G: 0.7881	D(x): 0.8576	D(G(z)): 0.4564 / 0.4547
[28/00][0/50]	Loss_D: 1.0388	Loss_G: 0.5322	D(x): 0.8618	D(G(z)): 0.5894 / 0.5873
[29/00][0/50]	Loss_D: 0.6887	Loss_G: 0.8714	D(x): 0.8658	D(G(z)): 0.4199 / 0.4184
[30/00][0/50]	Loss_D: 0.7070	Loss_G: 0.7059	D(x): 0.8698	D(G(z)): 0.4605 / 0.4609
[31/00][0/50]	Loss_D: 0.8011	Loss_G: 0.6640	D(x): 0.8733	D(G(z)): 0.5100 / 0.5143
[32/00][0/50]	Loss_D: 0.7401	Loss_G: 0.7815	D(x): 0.8767	D(G(z)): 0.4591 / 0.4577
[33/00][0/50]	Loss_D: 0.7033	Loss_G: 0.8301	D(x): 0.8800	D(G(z)): 0.4376 / 0.4360
[34/00][0/50]	Loss_D: 0.7200	Loss_G: 0.7951	D(x): 0.8830	D(G(z)): 0.4532 / 0.4515
[35/00][0/50]	Loss_D: 0.6765	Loss_G: 0.8568	D(x): 0.8859	D(G(z)): 0.4261 / 0.4245
[36/00][0/50]	Loss_D: 0.6563	Loss_G: 0.8007	D(x): 0.8886	D(G(z)): 0.4162 / 0.4145
[37/00][0/50]	Loss_D: 0.7497	Loss_G: 0.7592	D(x): 0.8912	D(G(z)): 0.4698 / 0.4689
[38/00][0/50]	Loss_D: 0.7700	Loss_G: 0.7242	D(x): 0.8936	D(G(z)): 0.4865 / 0.4847
[39/00][0/50]	Loss_D: 0.5206	Loss_G: 1.0762	D(x): 0.8959	D(G(z)): 0.3421 / 0.3400
[40/00][0/50]	Loss_D: 0.6182	Loss_G: 0.6794	D(x): 0.8981	D(G(z)): 0.5007 / 0.5060
[41/00][0/50]	Loss_D: 0.6044	Loss_G: 0.9363	D(x): 0.9003	D(G(z)): 0.3931 / 0.3921
[42/00][0/50]	Loss_D: 0.6066	Loss_G: 0.9307	D(x): 0.9025	D(G(z)): 0.3959 / 0.3943
[43/00][0/50]	Loss_D: 0.5817	Loss_G: 0.9657	D(x): 0.9046	D(G(z)): 0.3821 / 0.3807
[44/00][0/50]	Loss_D: 0.6386	Loss_G: 0.8770	D(x): 0.9065	D(G(z)): 0.4275 / 0.4160
[45/00][0/50]	Loss_D: 0.4904	Loss_G: 1.1244	D(x): 0.9094	D(G(z)): 0.3258 / 0.3248
[46/00][0/50]	Loss_D: 0.6385	Loss_G: 0.8714	D(x): 0.9103	D(G(z)): 0.4199 / 0.4184
[47/00][0/50]	Loss_D: 0.4876	Loss_G: 1.1217	D(x): 0.9121	D(G(z)): 0.3267 / 0.3257
[48/00][0/50]	Loss_D: 0.5220	Loss_G: 1.0510	D(x): 0.9137	D(G(z)): 0.3507 / 0.3493
[49/00][0/50]	Loss_D: 0.5621	Loss_G: 0.9795	D(x): 0.9153	D(G(z)): 0.3773 / 0.3755
[50/00][0/50]	Loss_D: 0.6101	Loss_G: 0.9014	D(x): 0.9168	D(G(z)): 0.4074 / 0.4060
[51/00][0/50]	Loss_D: 0.5329	Loss_G: 0.9005	D(x): 0.9183	D(G(z)): 0.3736 / 0.3721
[52/00][0/50]	Loss_D: 0.4126	Loss_G: 1.2740	D(x): 0.9198	D(G(z)): 0.2885 / 0.2795
[53/00][0/50]	Loss_D: 0.4888	Loss_G: 1.1034	D(x): 0.9211	D(G(z)): 0.3328 / 0.3317
[54/00][0/50]	Loss_D: 0.5534	Loss_G: 0.9798	D(x): 0.9224	D(G(z)): 0.3766 / 0.3754
[55/00][0/50]	Loss_D: 0.6354	Loss_G: 0.8558	D(x): 0.9236	D(G(z)): 0.4264 / 0.4250
[56/00][0/50]	Loss_D: 0.4657	Loss_G: 1.1390	D(x): 0.9247	D(G(z)): 0.3212 / 0.3201
[57/00][0/50]	Loss_D: 0.5395	Loss_G: 0.9074	D(x): 0.9259	D(G(z)): 0.3703 / 0.3688
[58/00][0/50]	Loss_D: 0.3999	Loss_G: 1.2873	D(x): 0.9271	D(G(z)): 0.2709 / 0.2700
[59/00][0/50]	Loss_D: 0.5887	Loss_G: 0.9143	D(x): 0.9282	D(G(z)): 0.4020 / 0.4008

Outcome for the loss for both generator and discriminator goes down while still being quite stable, as well as the confidence for the discriminator goes up. However, the generated pictures are still not really looking like the training managed to capture the main features of the project.

FINAL REFLECTIONS

The outcome of this project is mildly very disappointing. However, I do know that my intentions and methods could have been sharper. There was a clear obstacle in the language barrier - my mandarin is horrendous as well as the Jiangyoung dialect being very hard to understand, forcing me to derive to aesthetic from my initial intention of making a translation/language-based project.

The images are quite simple and are all black and white resulting in very limited diversity in the generated tensors. Also, I have a feeling that that might make the training a bit weird, as the discriminator quite quickly becomes very good.

If I were to try to carry out this project again, I would probably train on a completely different model.

Additional note added on 16th June:

I JUST stumbled upon this project interpolating between traditional Chinese characters and then trying to incorporate Hangul - the characters used for Korean script. This would have been the perfect model to use for this project, but unfortunately, I discovered it too late to do it for this deadline. Project can be found here: <https://kaonashi-tyc.github.io/2017/04/06/zi2zi.html>

REFERENCES

Lofthouse, A. (2020). Nüshu: China's secret female-only language. BBC Travel. Retrieved 8th June from <https://www.bbc.com/travel/article/20200930-nshu-chinas-secret-female-only-language>