AIML431 Assignment 4

1. Introduction

First of all, most of the work was done on the Colab, linked here https://colab.research.google.com/drive/1dQGyb91vSZ_T8bM4OBE_xAKfEYxnmS8F?usp=sharing. In addition, the python files in program.zip can also run normally. Please refer to readme.txt for details.

According to the requirements of the assignment, complete the requirements of each part in sequence. First load and preprocess the data. Format the image size to 32*32, perform augmentation, and set the Grayscale parameter to num_output_channels=params['nc']. In the second step, the Discriminator and Generator are constructed according to the given illustrations, and the code is referenced from the relevant documentation in the PyTorch tutorials. In the third step, the DCGAN are trained and generated, and the saving and loading of the model is implemented. Eventually five sets of data were trained separately, which are "u" (folder 2), "W" (folder 3), "O" (folder 0), "E" (folder 14), and "H" (folder 17). Each set of data generates 11 images (one per 10 epochs), totaling 55 images, which can be animated in the colab to show the changes. And the corresponding 5 generator models are saved. Finally, a conditional DCGAN was implemented to generate images by loading images of characters "A" and "B" and using 0/1 as the character labels.

At last, the trained DCGAN model is saved in program.zip, which can be loaded via text.py and generate images. In addition, conditional DCGAN models are not saved, but the results can be viewed directly on the Colab.

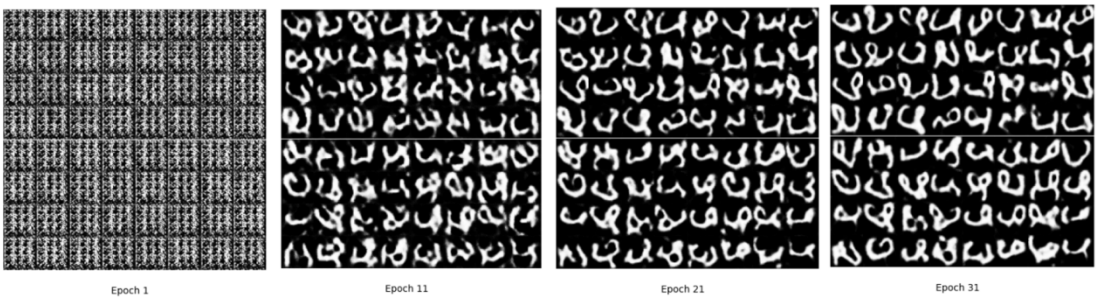
2. Implementation

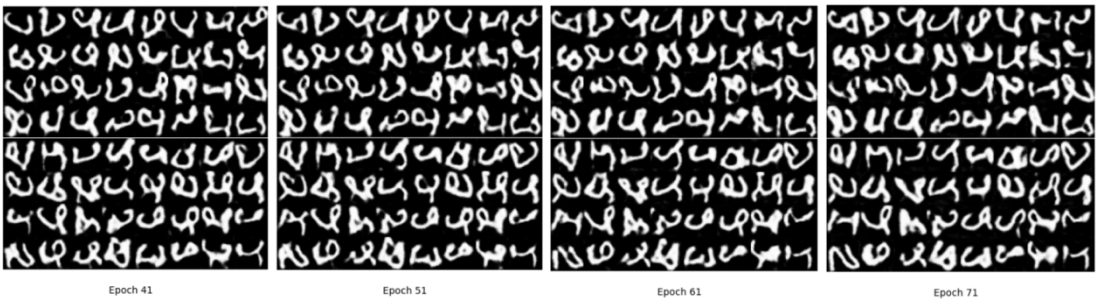
One problem occurs during the implementation of DCGAN is that when the input image is not set to Grayscale, an error occurs at runtime indicating that the input image tensor is different from the input dimensions of the model. Since the default image has 3 RGB channels instead of 1, I have set the Grayscale parameters to num_output_channels=params['nc'] during preprocessing to solve this problem.

Another problem that occurs when implementing conditional DCGAN is that at runtime, the number of input channels to the hidden_layer in the Discriminator and Generator is different from the number of input channels required by the model itself. The reason is because in conditional DCGAN, the number of input channels of hidden_layer is pieced together from the output channels of hidden_layer1 and hidden_layer2. If want to achieve a similar channel setup as the DCGAN model, need to divide the output channels of hidden_layer1 and hidden_layer2 by 2, respectively.

3. Experiment discussion

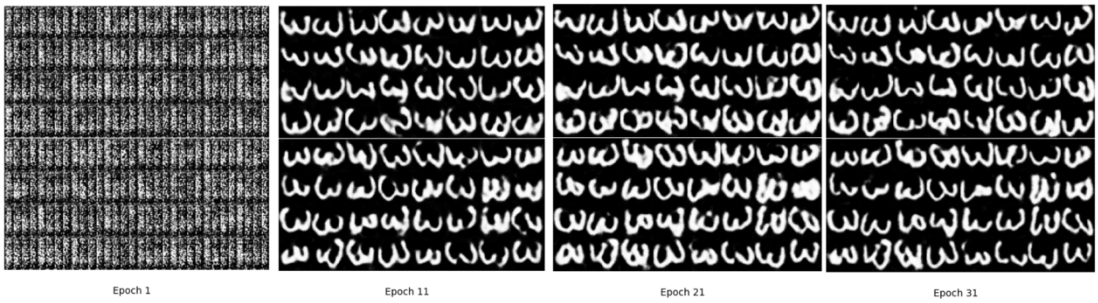
The final generation results of my DCGAN model:
"u" (folder 2):


Epoch 1
Epoch 11
Epoch 21
Epoch 31

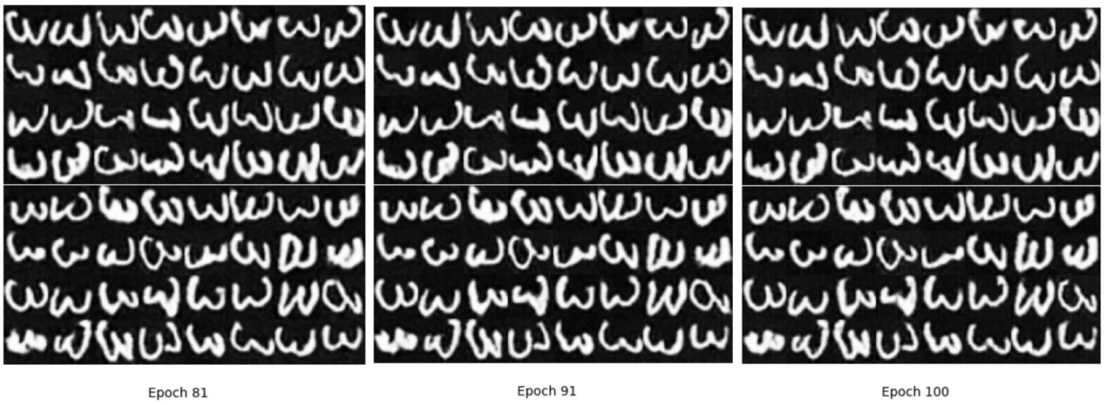

Epoch 41
Epoch 51
Epoch 61
Epoch 71


Epoch 81
Epoch 91
Epoch 100

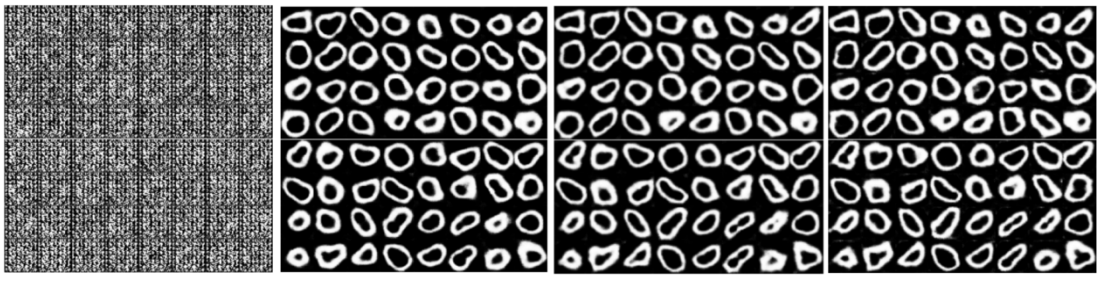
"W" (folder 3):

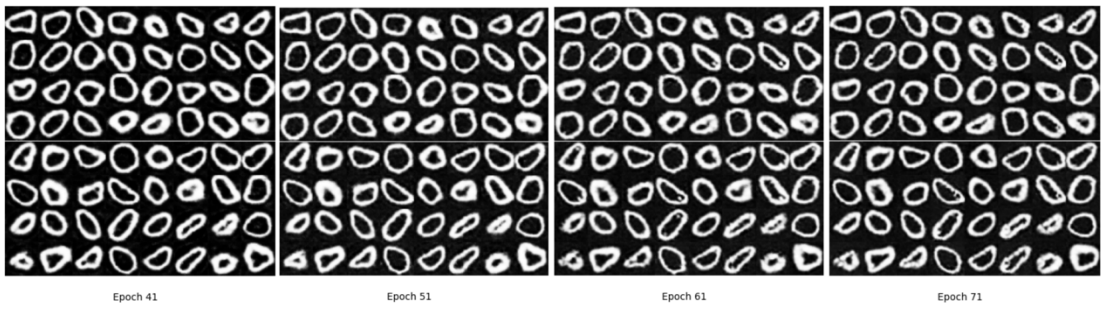

Epoch 1
Epoch 11
Epoch 21
Epoch 31


Epoch 41
Epoch 51
Epoch 61
Epoch 71

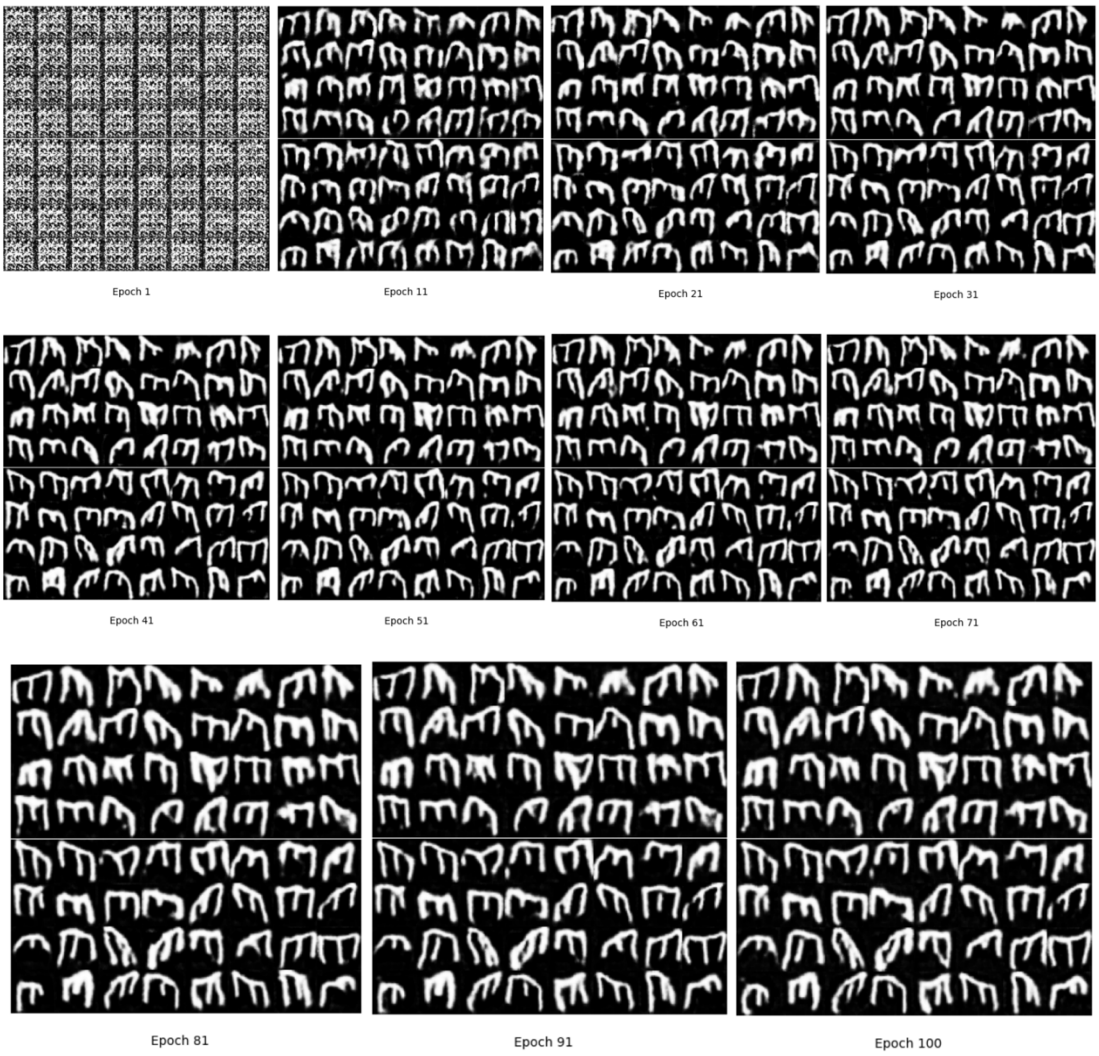

Epoch 81
Epoch 91
Epoch 100

"O" (folder 0):

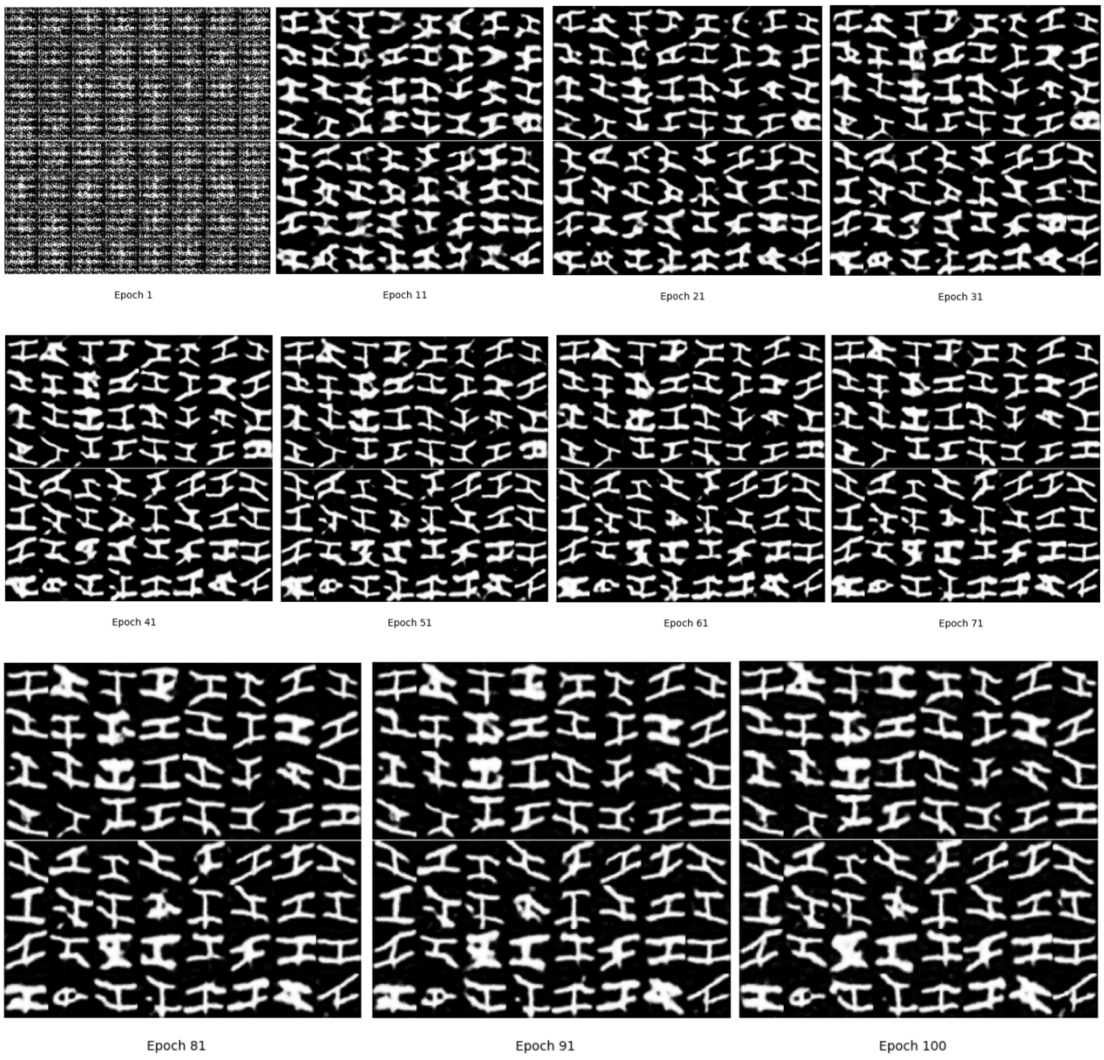

Epoch 1
Epoch 11
Epoch 21
Epoch 31


Epoch 41
Epoch 51
Epoch 61
Epoch 71

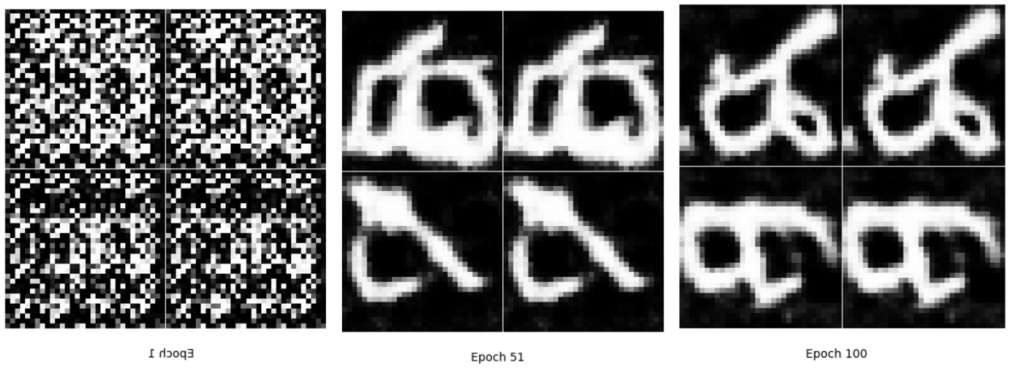
"E" (folder 14):



"H" (folder 17)



conditional DCGAN(A and B):

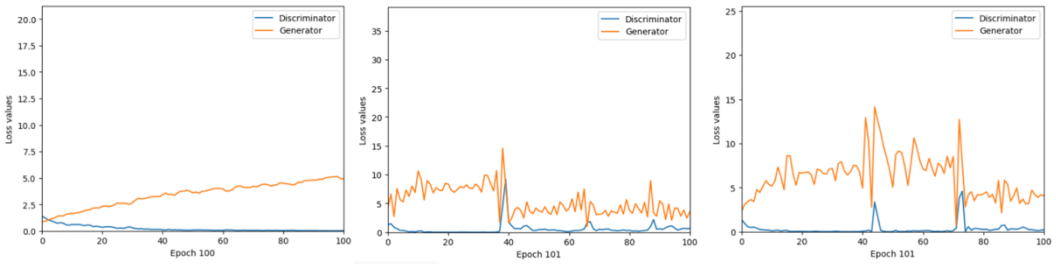


Some insights and analysis:

1. Judging from the above results, the efficiency of text generation in different character folders is different. Some characters will be learned by the generator more easily than others. For example, the characters "O", "W" and "E" have been well generated around epoch 11, while the characters "u" and "H" are still not very accurate after 100 epochs.

2. During the training process, the losses of the generator and discriminator change as shown below. As can be seen from the figure and the output log, the loss changes of generator and discriminator are opposite, that is: Loss_D increases and Loss_G decreases, and vice versa. This is because of the loss function, the generator and the discriminator is a game process, when the discriminator's loss decreases, it means that the discriminator can easily distinguish between true and false samples, at this time, the generator's loss should be increased in order to generate more realistic samples.

In addition, it can be seen that the loss function is smoother at lr = 0.0001, while the other two loss functions fluctuate more. This is because a small loss function makes the model training slower and thus smoother. On the other hand, too small learning rate may fall into local optimal solution.

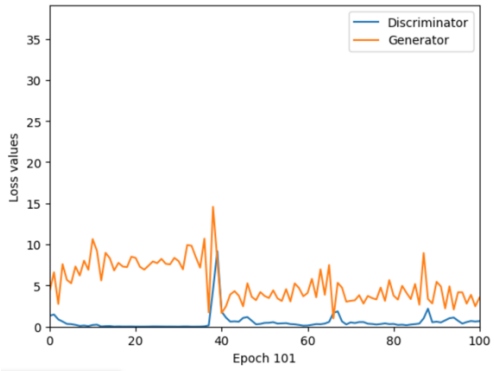


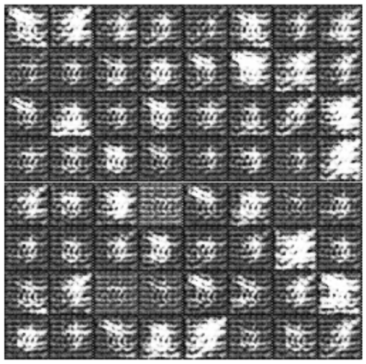
Lr = 0.0001 Lr = 0.001 Lr = 0.0005

3. The quality of the generated images is improved in the early epochs of training. For the "E", "O" and "W" characters, it can be found that after training for a period of time, the generated characters do not change significantly. It may be that due to the gradual convergence of the model, it is difficult for the discriminator to distinguish the authenticity of the characters generated by the generator, and the generated characters of the generator do not change much because the discriminator is difficult to improve.

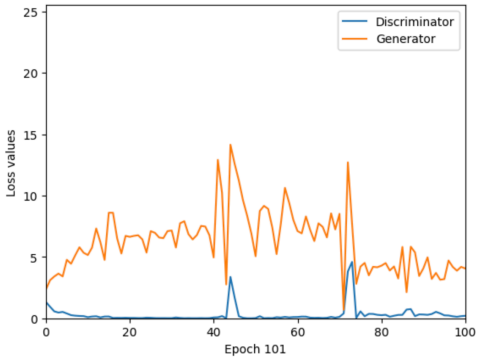
4. Data augmentation leads to better models. Although the data augmentation part was not experimented in this assignment, theoretically, data augmentation will increase the data diversity, which will improve the generalization ability of the model as well as reduce the overfitting problem.

5. Adjustments to learning rates. I tried two new lr configurations of 0.001 and 0.0005. When lr=0.001 the loss function is shown below and it can be seen that the loss function fluctuates more. Also at Epoch 31 the image suddenly becomes unrealistic. It may be due to the increase of the learning rate, which leads to each parameter update is too large, thus departing from the distribution that approximates the real image and generating more noise.




Epoch 31

When lr = 0.0005, there is no noise phenomenon except at the beginning. The generated image looks pretty much the same as when lr=0.0001.




Epoch 101