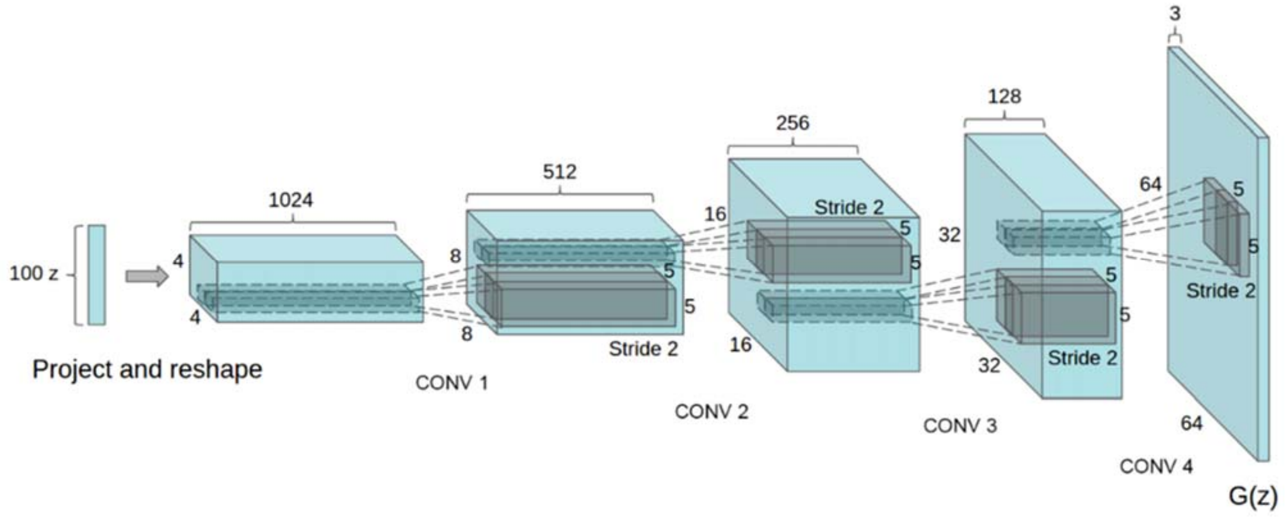
TheCEDL Homework 3

李青峰、陳金博、羅羿牧

1. Introduction:

Generate simulated images from DCGAN generator learned from real images distribution. The structure of our system is a discriminator which discriminates real world pictures from pictures generated from our DCGAN generator and a generator which always generates images to confuse the discriminator. (Fig. 1) demonstrates how our generator generates simulated graphs:



(Fig. 1) Deconvolution is applied to generate an 64 x 64 image (right most image) from image representation (left most vector)

Likewise, our discriminator uses the same structure (same stride, step size) to discriminate images (Fig.2). In our project, the resolution of our images are of size 256 x 256, and the hidden convolutional layers are scaled accordingly (4 times wider and higher).

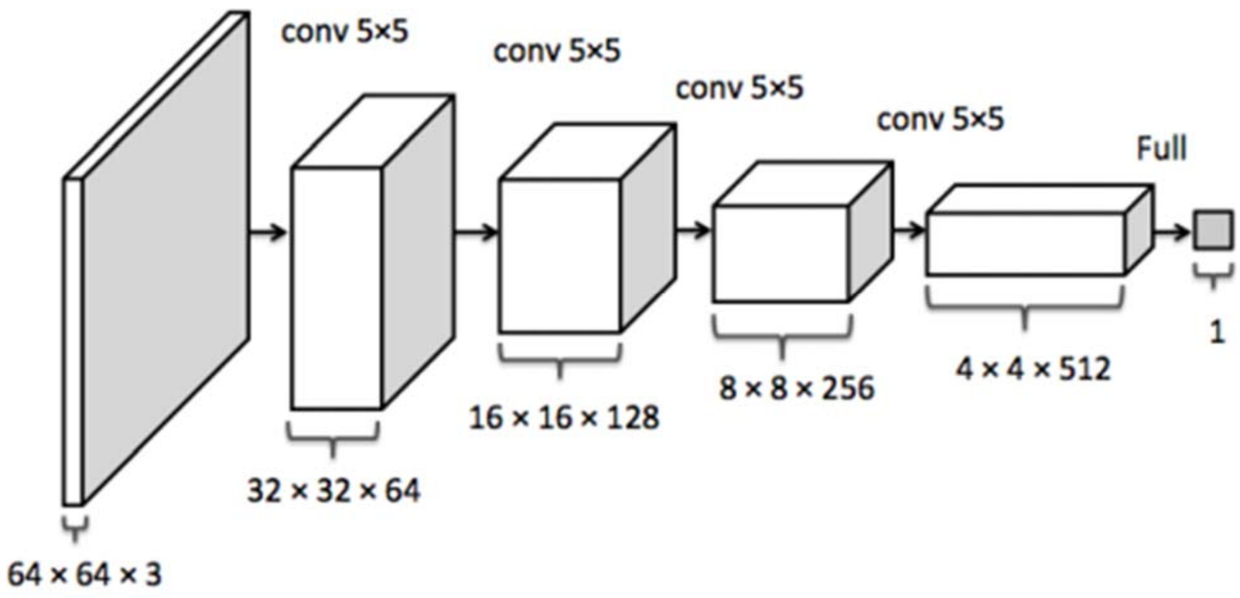


Fig. 2

2. Method applied:

1. Different filter size at the first conv. layer [16x16, 32x32, 64x64]

   We tried different filter size for our experiment to discuss the image quality of different image resolution and its training time

2. Randomize the input figure at the beginning of every epoch

   We try to avoid the same group of pictures appearing in the same mini-batch for each epoch. This will (hopefully) break dependencies among images in the same batch and get better generalization performance of the trained DNN.

3. Add additional layer (2 additional layers are added)

4. Add more training images

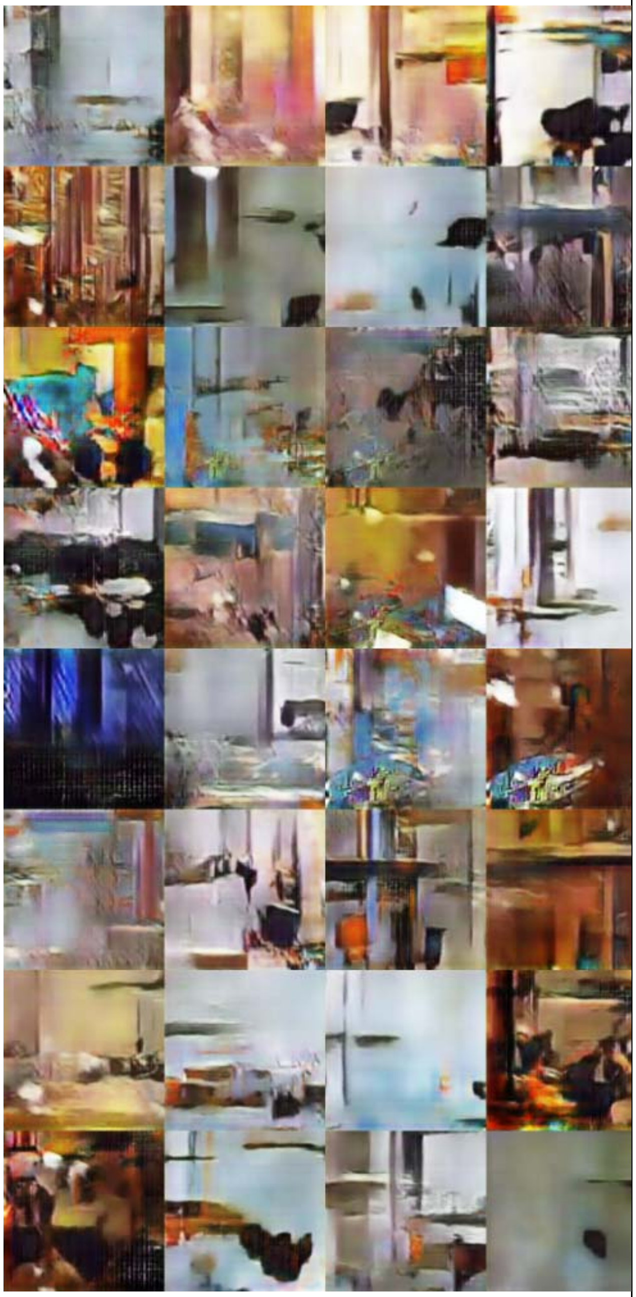
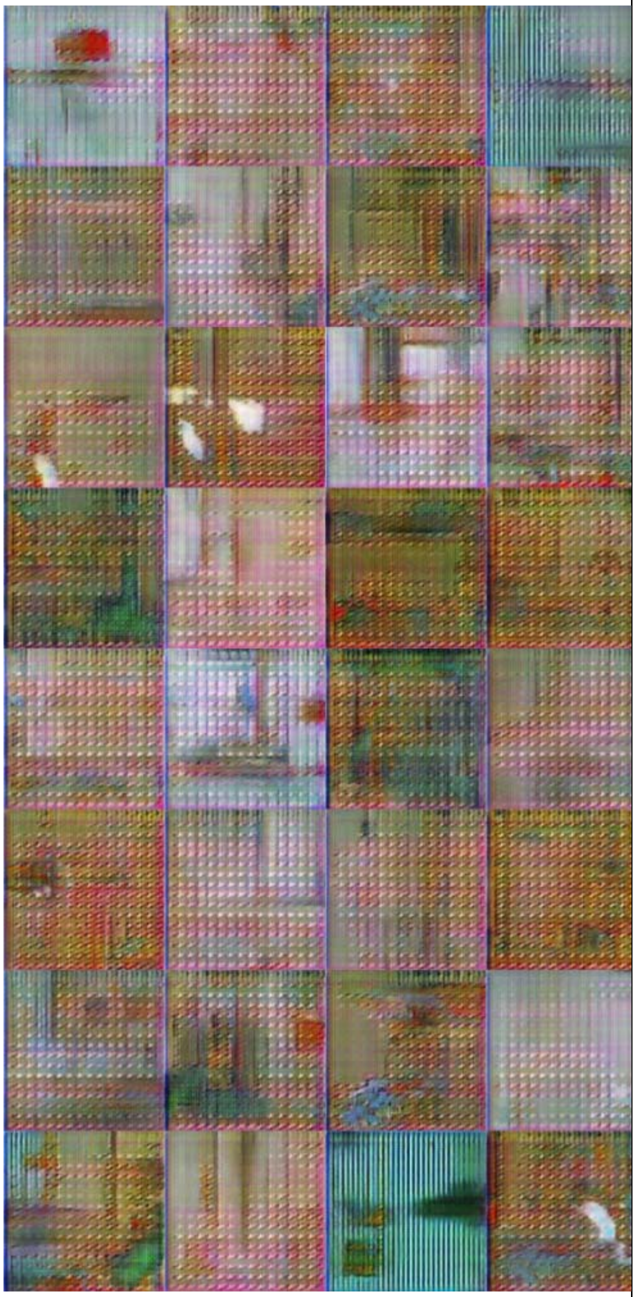
5. Change the update rate of generator and discriminator

   We found that the G network often lags behind D's progress after some time as d_loss goes below 1e-2. It seems that learning to generate an image is much more difficult to learn than discriminate real images from generated ones.

   We tuned the update ratio between discriminator and generator to be 1:4 and 1:8. That is, generator update its parameter 4 to 8 times for every step (batch) while the discriminator stays the same. This effectively gives the generator network a better chance to catch up with the discriminator's performance.

Experiments

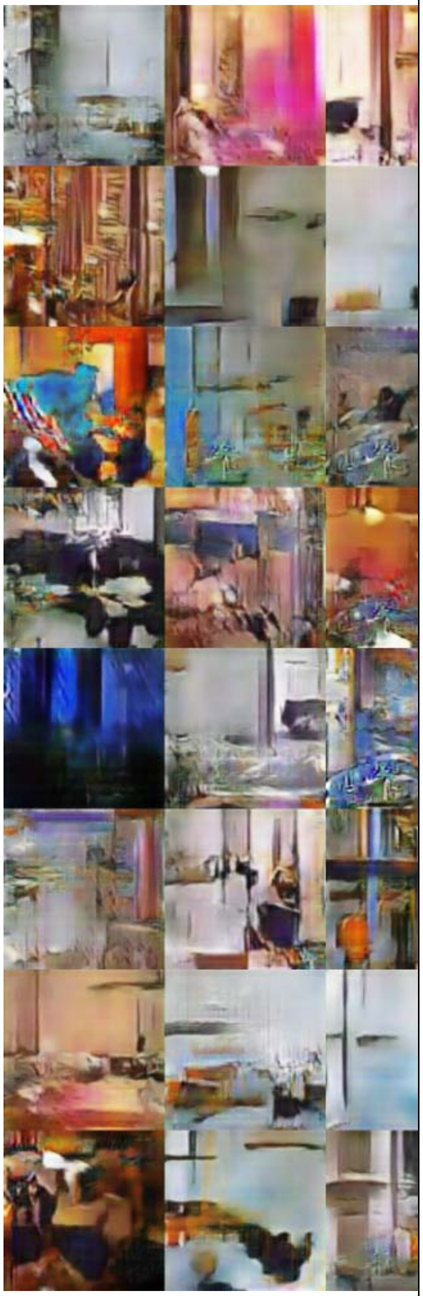
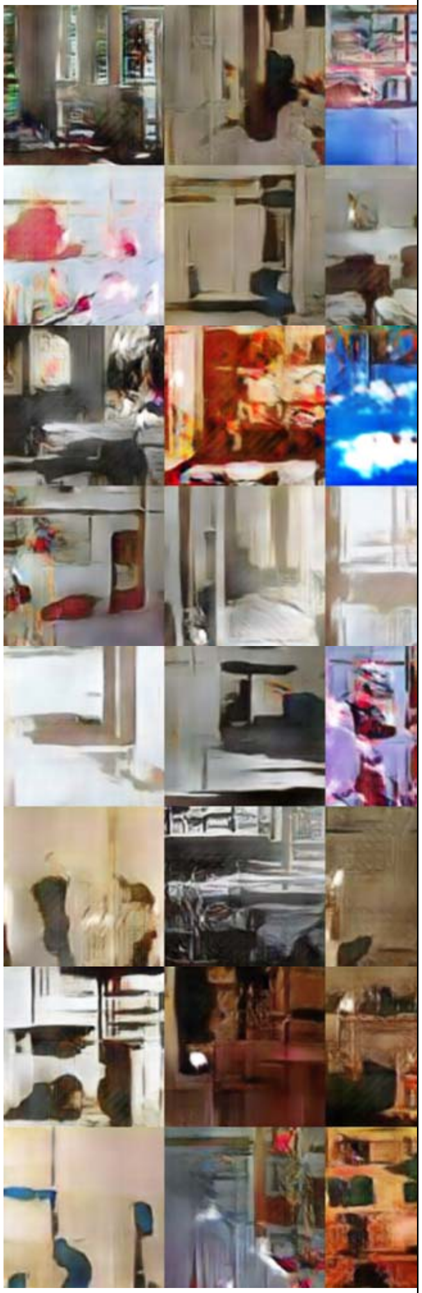
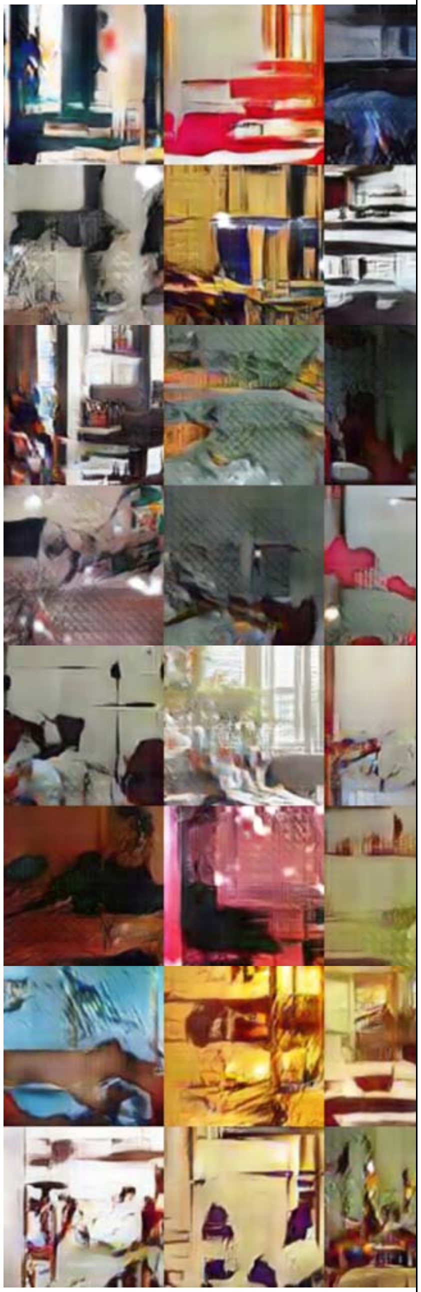
In this section, we'll demonstrate the result after we changed our training strategy. In each of the setting, we trained up to 90 epoches, however, some of them crashes in the middle of training (generator crashes and generated odd images) for example:

Epoch 59 training step: 420	Epoch 59 training step: 520
	

(Fig. 3) the example of generator break down

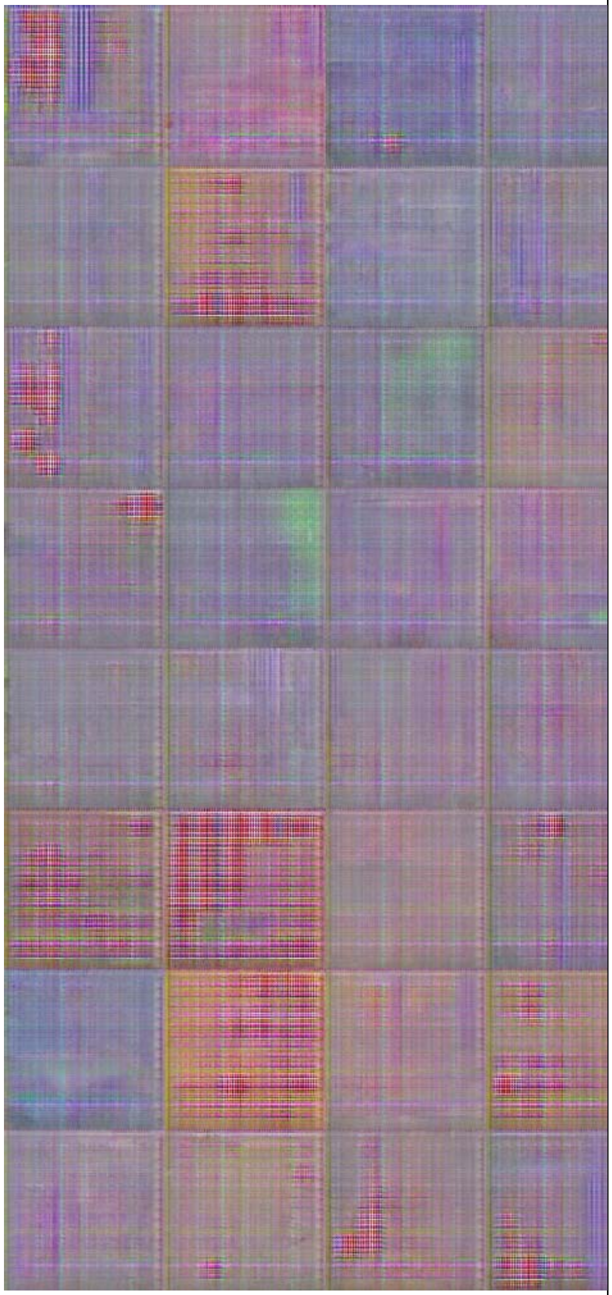
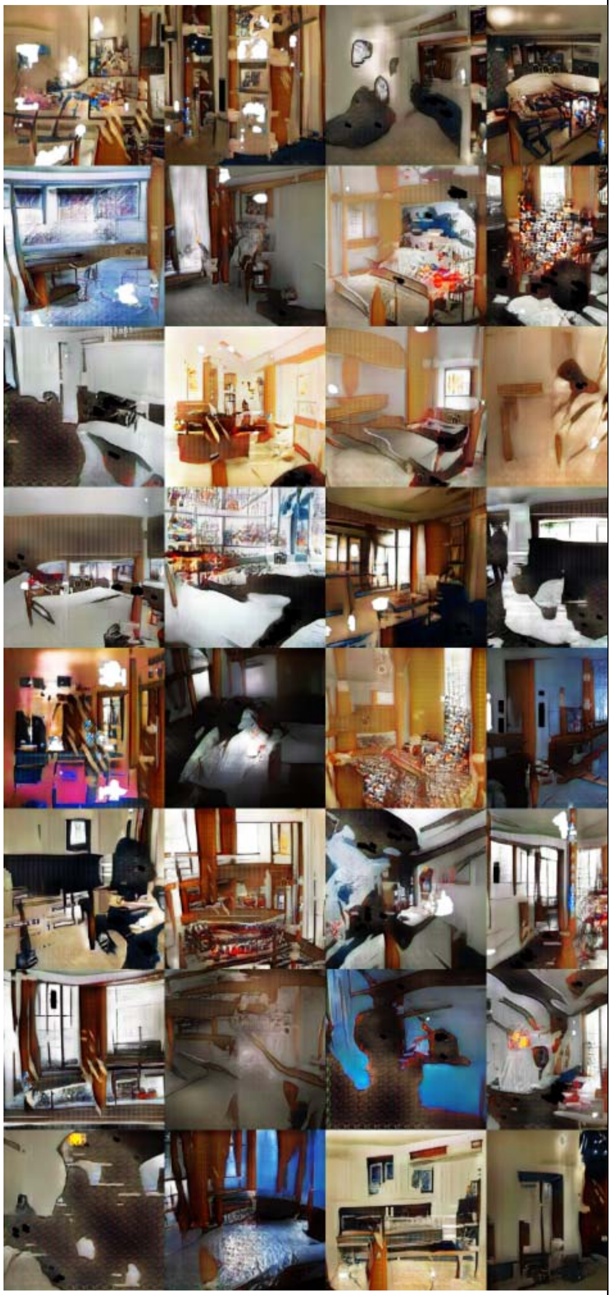
1. Different size of filter for the first conv. layer:

   It is clear that the rightmost one has the best image quality in terms of the large scale structure of the generated rooms. Our other experiments will stick with this configuration.

filter size = 16x16	filter size = 32x32	filter size = 64x64
Epoch = 30	Epoch = 30	Epoch = 30
		

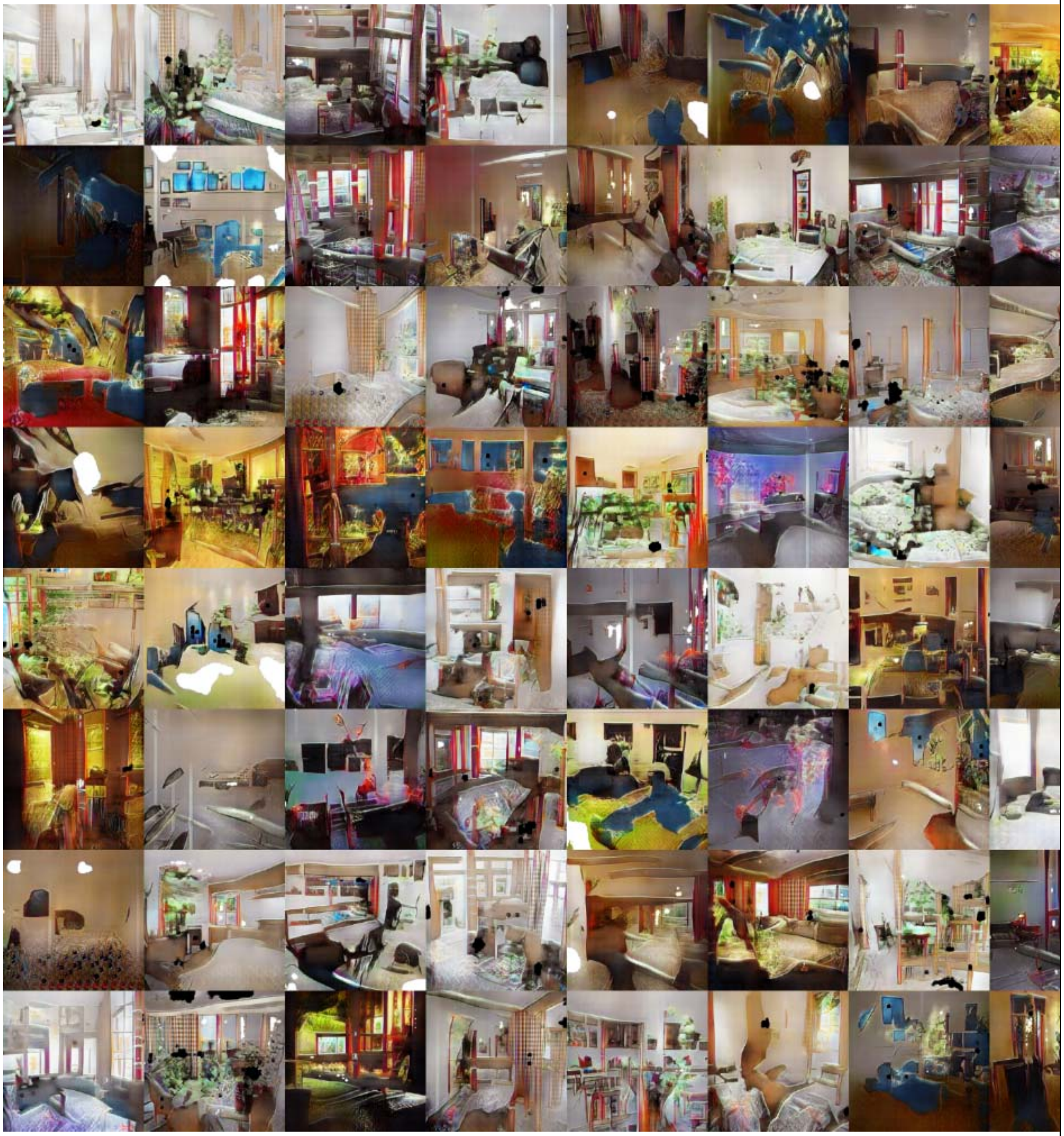
2. Effect of shuffling training data:

   In our 3 repeated experiments, the generators all collapse before reaching epoch 60 if no training data shuffling is performed.

Before randomize: Collapse on around 50 epoches	After randomize: Survives after 78 epoches
	

3. More training images:

After adding more "real" images and trained for the same steps (not epoch), the image quality did improve for the network of this size.

Epoch 14


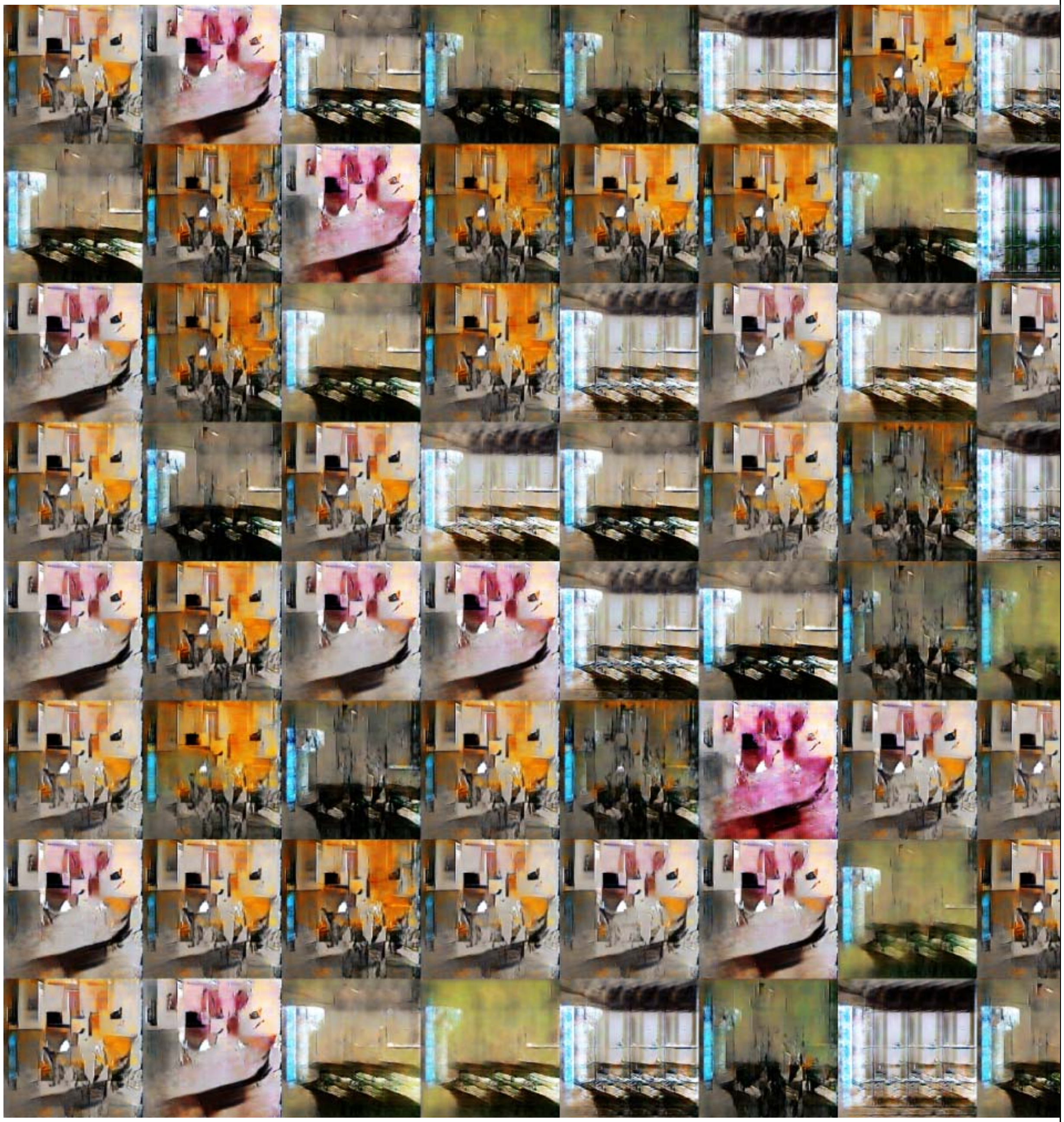
4. Decrease learning rate of both D and G networks after some time:

   This technique helps to obtain better looking images at the end of a training session by making only subtle adjustments.

5. Add 2 additional layers:

   For some unclear reasons, only a few images are generated after increasing the dimension of both D and G networks, regardless of the input vector of G.

   There is an article discussing about the exact same issue here (http://www.araya.org/archives/1183).

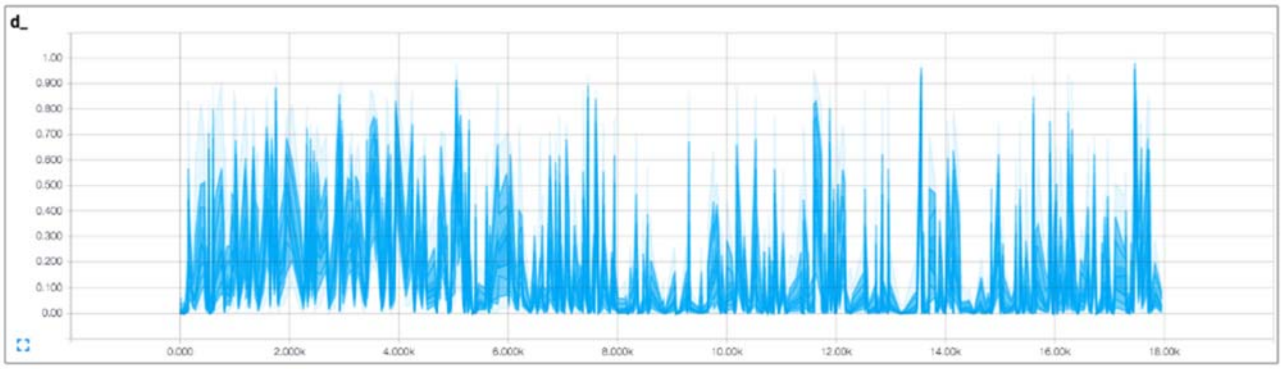
Epoch: 4


6. Update rate tuning:

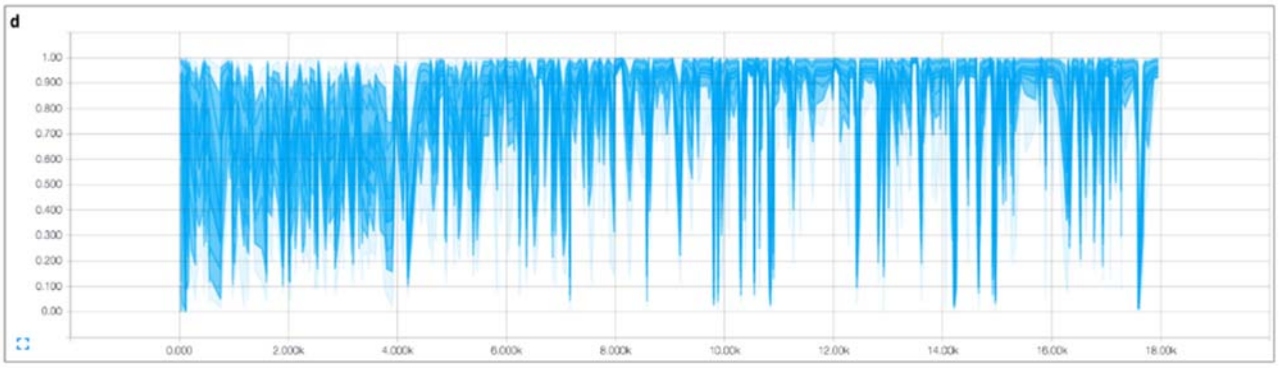
   We didn't provide the result images of tuned refresh rate, instead, we modify the refresh rate by monitoring d_loss. If we observe that there are imbalance between d_loss and g_loss, then we tune the refresh rate to balance the learning result.

To conclude, the best performance occurs after we add training images, shuffle the training sample for every epochs, and tuned the refresh frequency of generator and discriminator. The training process is shown below:

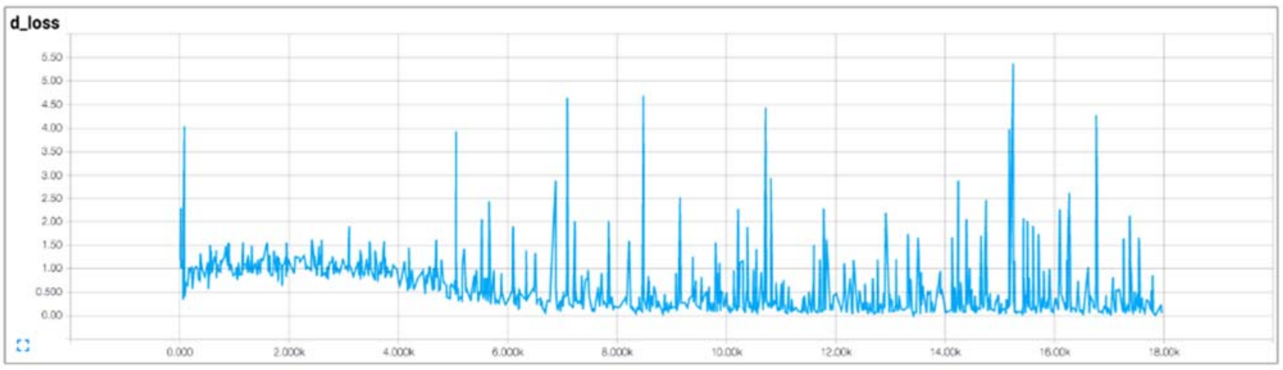
d_false:



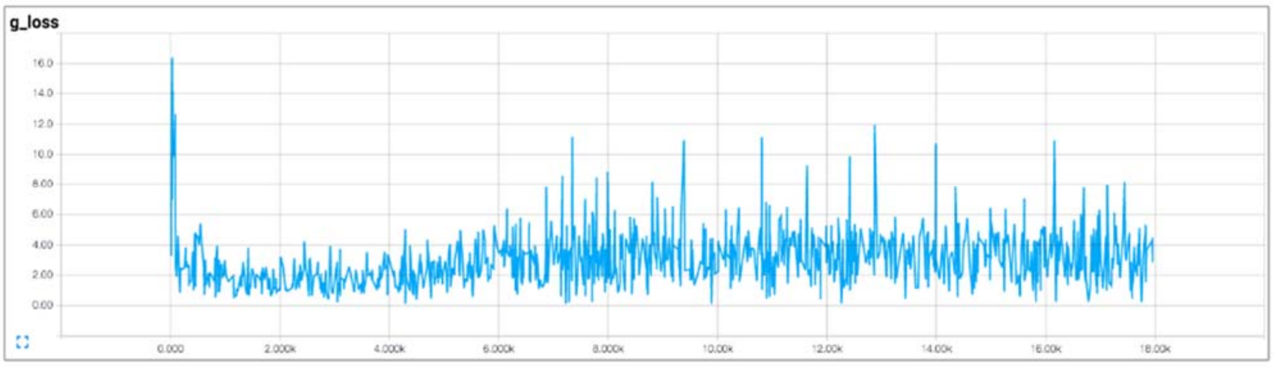
d_True:



d_loss total:



g_loss:



3. Discussion

First of all, although output size should be fixed. We can improve the training result by adding more training images (see experiment 3) in our training set.

While training, we found that the generator will crash after certain number of epochs (see experiment 2). To avoid that, We shuffle the training images in every epoch to generalize the distribution of P_g, the distribution of generator.

In addition, we found that the performance of discriminator will exceed the generator and cause the generator to break down (and generates images with strange patch and textures), so we slightly make the update frequency larger for the generator than discriminator.

Finally, we added two layers on our CNN model and found the performance of generator converges faster, but since we have only trained a few epochs and it seems most of the images looks similar, it is still too early to discuss about the performance of adding two layers.

4. 貢獻

李青峰：嘗試各種方法增加 model 的準確性

陳金博：統計結果, 做成報告

羅羿牧：文獻搜尋, 資料查找