COP 6610(Machine Learning) Fall 2019
Project Report

Yefan Tao
UFID: 24055676
ustctao@ufl.edu

December 4, 2019

This report consists of the follwing four parts:

- Introduction and background
- Results of GAN(Generative Adversial Network) on MNIST and face data
- Results of VAE(Variational Auto Encoder) on MNIST and face data
- other advanced topic(ie, generate cat face)

1 Introduction and Background

The goal of this project is to apply GAN and VAE on MNIST and given input face data, to generate "fake" handwritten digit and faces.
The original idea of GAN was proposed by Goodfellow et al in 2014 [1], and grow rapidly into lots of different variations, including DCGAN(Deep Convolutional GAN) [6], LSGAN(Least Square GAN) [5], Softmax GAN [4] and so on. The key idea of GAN is that, we will have a Generator(**G**) trying to mimic the input data, and a Discriminator(**D**) trying to distinguish "real" data from the "fake" ones. After several epochs of training on **G** and **D**, if converged, **G** will be the same distribution as the input and **D** will go to 1/2, meaning it's unable to distinguish real input from the generation.

Variational Auto Encoder(VAE) was proposed by Kingma and Welling in 2013 [3]. VAE consists of an encoder, a decoder, and a loss function.Encoder and decoder are two separate neural network, the encoder converts an input image into a latent space, and decoder gets the latent space and reconstruct an image output. The difference between a VAE and a regular AE is that, while AE is trained over the representation in the latent space, the VAE is trained over the parameters of distribution on representation. In short, VAE allows some noise or error, which makes it better to avoid the problem of overfitting.

2 Results of GAN(Generative Adversial Network) on MNIST and face data

The realization of GAN is not hard, there are lots of examples avaible on the Internet. I am taking the one from the GitHub(`https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/gan/gan.py`) to run on MNIST, and use another DCGAN(`https://github.com/eriklindernoren/PyTorch-GAN/blob/master/implementations/dcgan/dcgan.py`) to run on face data.

2.1 Data

2.1.1 MNIST data

The MNIST data in loaded by using Dataloader in PyTorch directly. Each element in "DataLoader" has an input image of 25 handwritten digits, and in each epoch, there are 938 batches. Apparently MNIST has 10 classes representing digits from 0 to 9, and the training/validation split is done automatically. The dataset will be downloaded automatically at run time.

2.1.2 Face data

Face data is provided by TA, coming from the VGGFace2 dataset. These data set consists of faces from 500 different people(grouped in different subfolder), and a total number of 170k face images. For the face images, my thought is to use the subfolder's name to label each person, so we have 500 different

classes. And for each loop of the running, an input image with 65 faces will be trained with a DCGAN.

2.2 Model description

2.2.1 GAN for MNIST

The GAN model used for MNIST is from the original Goodfellow directly. The generator consists of 5 linear layers, and between each leayers there's a LeakyReLU activation function. The discriminator consists of 3 linear layers, and also uses LeakyReLU as activation functions.

2.2.2 DCGAN for face data

The DCGAN model is a little more complicated. The generator consists of 3 layers of 2D convolutional layers(nn.Conv2d), activated by LeakyReLU function. The discriminator consists of 4 layers of 2D convolutional layers, activated also by LeakyReLU function.

2.3 Results

Here I attach only parts of the results, the full set of results are available on my GitHub repository(`https://github.com/yefanTao/CAP6610_GAN`). For MNIST results, output image is stored for every 400 times of training, from 0 to 187200. For face data, output image is stored for every 50 times of training, from 0 to 6596.
Figure 1 shows the output results for MNIST at step=0,800,2000,10000,50000 and 100000. As we can see, the training result is fairly good, the output images look the same as handwritten digits.

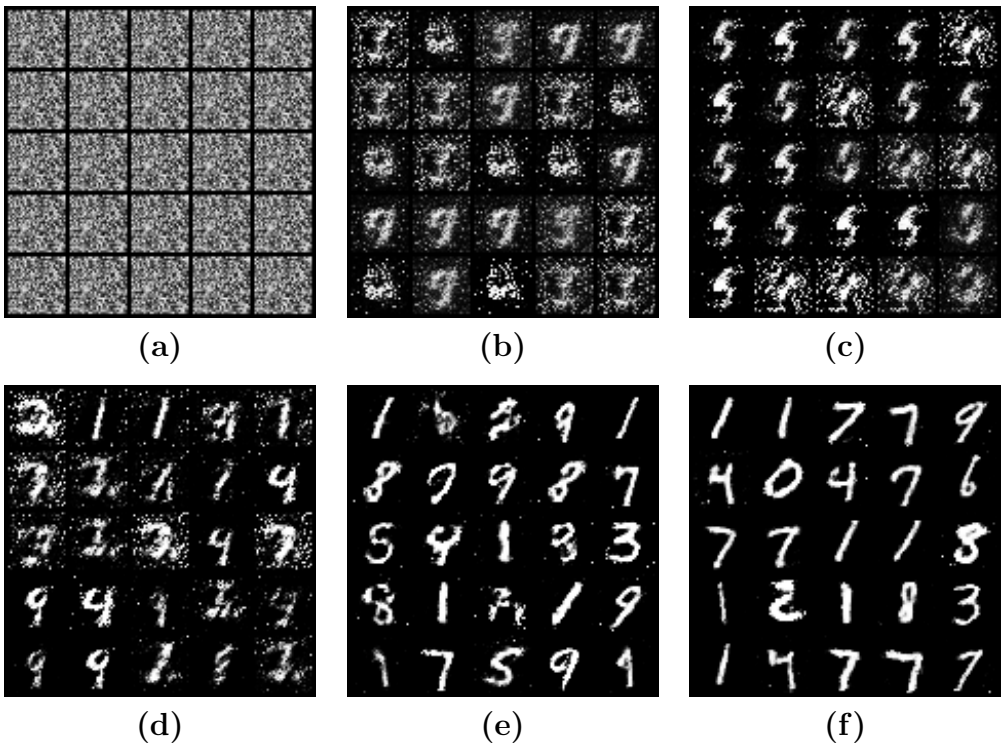
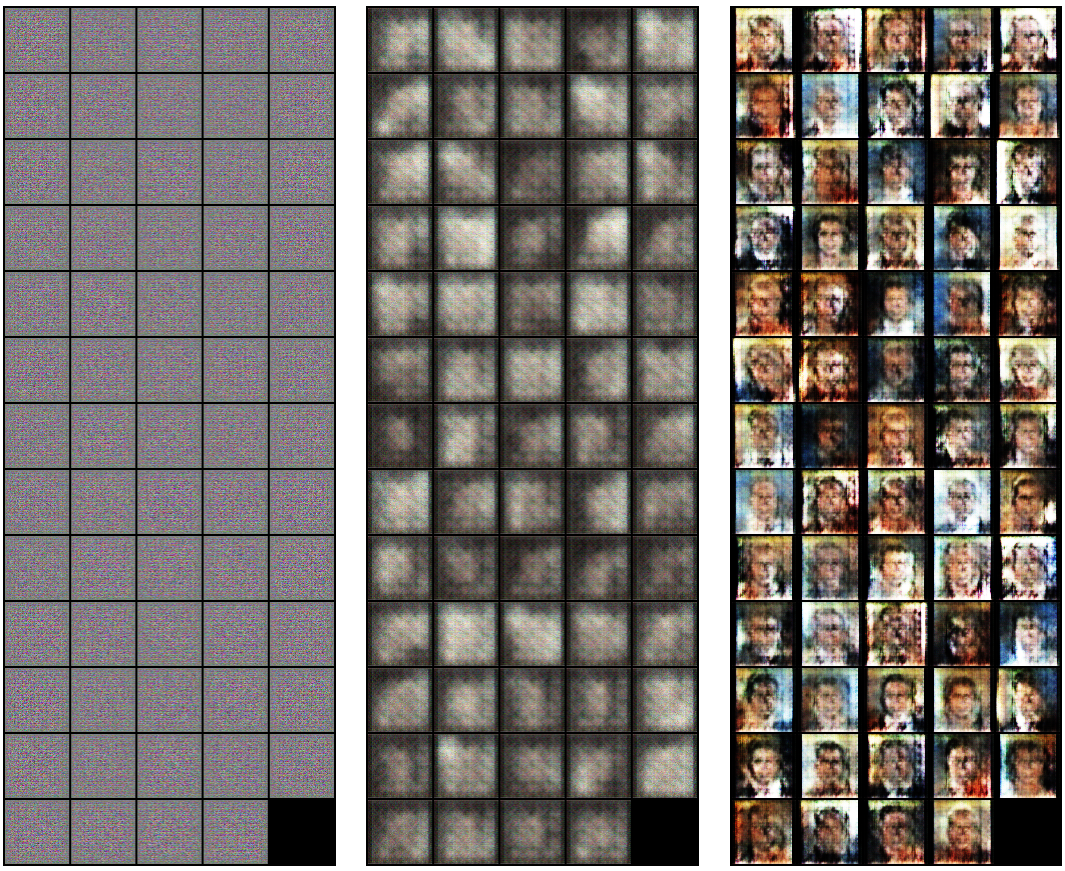


Figure 1: GAN Training result on MNIST at step **(a)** 0 **(b)** 800 **(c)** 2000 **(d)** 10000 **(e)** 50000 **(f)** 100000

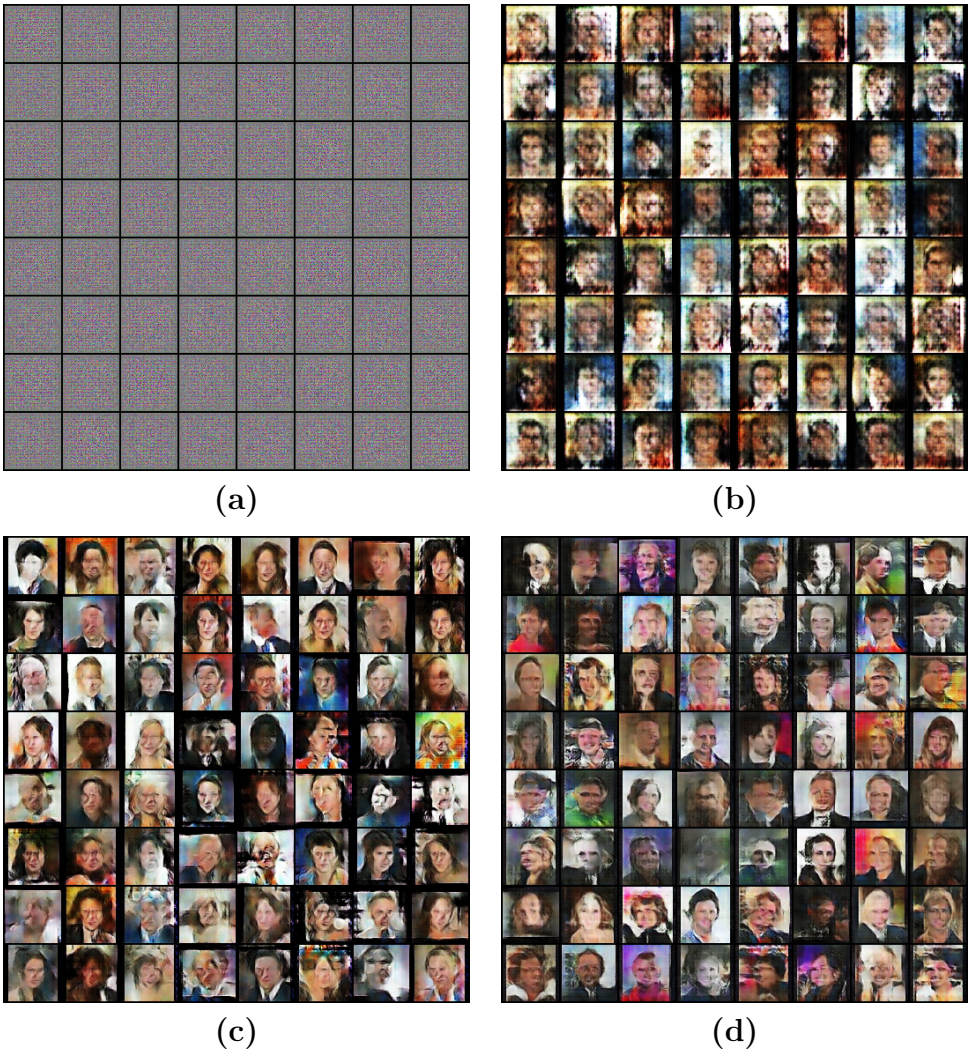
Figure 2 shows the output results for face data at step 0, 100, 1000. Similarly, we can see that the output images are getting more and more clear, and we can see the profiles of the faces in the last one. However, even after training for about 10 hours on my laptop, this output is still blurry and not clearly showing the details of the faces.



(a) **(b)** **(c)**

Figure 2: DCGAN Training result on MNIST at step **(a)** 0 **(b)** 100 **(c)** 200 **(d)** 1000

Then, I increase the number of epoches, and try to see if we can get better on the output image quality, results are shown in Figure 3. So obviously, after more epoches of training, the images get better and clearer.



(a) **(b)** **(c)** **(d)**

Figure 3: DCGAN Training result on face at step **(a)** 0 **(b)** 1000 **(c)** 5020 **(d)** 10218

3 Results of VAE(Variational Auto Encoder) on MNIST and face data

Similar to GAN, I use the official example tutorial in PyTorch, it's available on GitHub(https://github.com/pytorch/examples/blob/master/vae/main.py).

For the face data, linear network model won't work, I took a DFC-VAE [2] project on GitHub(https://github.com/matthew-liu/beta-vae/blob/master/models.py) for reference and modify the script to satisfy my needs.

3.1 Data

The input data on VAE is the same as GAN, which is shown in the previouis section.

3.2 Model description

3.2.1 VAE for MNIST

This model is from the official PyTorch tutorial. The encoder is two 4-liner layers, with first layer activated by ReLU. What's worth mentioning is that, the two output from VAE(mu and logvar) share the first linear layer, then get separted since the second layer, and get reparameterzied to latent variable z in the end. The decoder is one linear layer with a ReLU.

3.2.2 DFC-VAE for face data

Similar to GAN, to deal with face data, we need to switch from linear model to 2D model. The encoder in this case consists of 4 2D convolutional layers, with first layer activated by Sigmoid. After 4 layers of transformation, two separate linear layers prdouce mu and logvar from latent variable z. The decoder is just the reverse, a linear layer firstly convert z to desired size, then a reverse 4 2D convolutional layers are applied.

3.3 Results

My results is also attached on my GitHub. Results are shown as below: Figure 4 shows the output results for MNIST at step=0,10,20,30,40 and 50. The output images seems to converge very fast, and after the first epoches, the quality of output images tend to be stable.

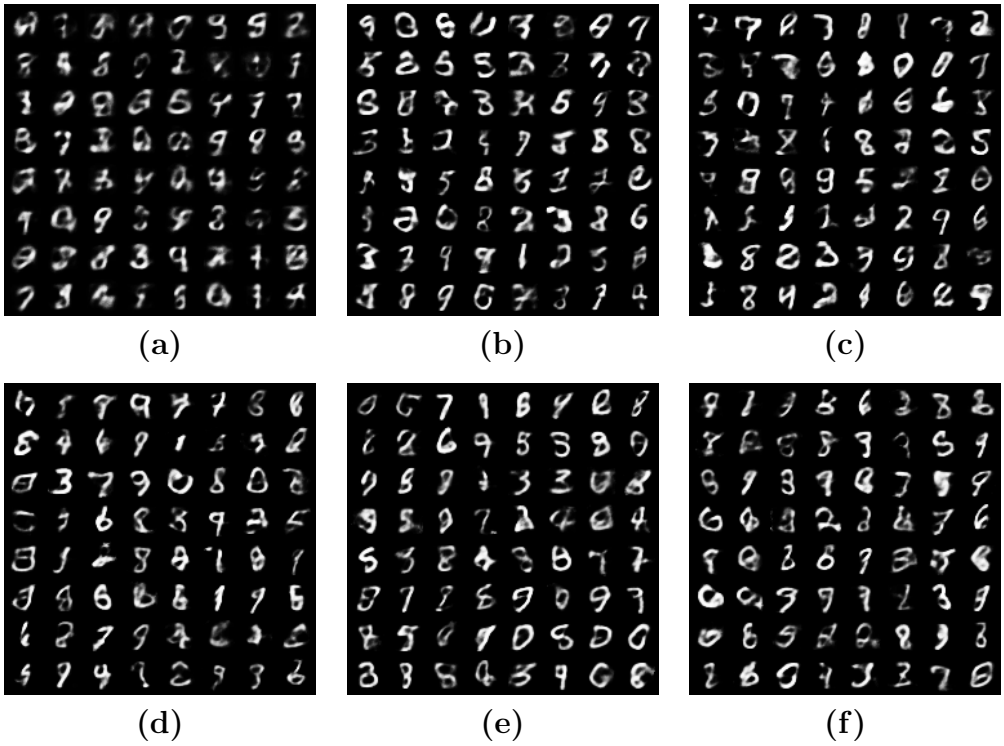


Figure 4: GAN Training result on MNIST after epoch **(a)** 0 **(b)** 10 **(c)** 20 **(d)** 30 **(e)** 40 **(f)** 50

Figure 5 shows the output results for face data at epoch 0, 10, 100, 500. After about 100 epoches, the image quality stops improving. It looks to me that, the blurriness of VAE output and GAN output are two differnt kinds, the GAN output is more like a blocking effect, while VAE output is an overall resolution problem.

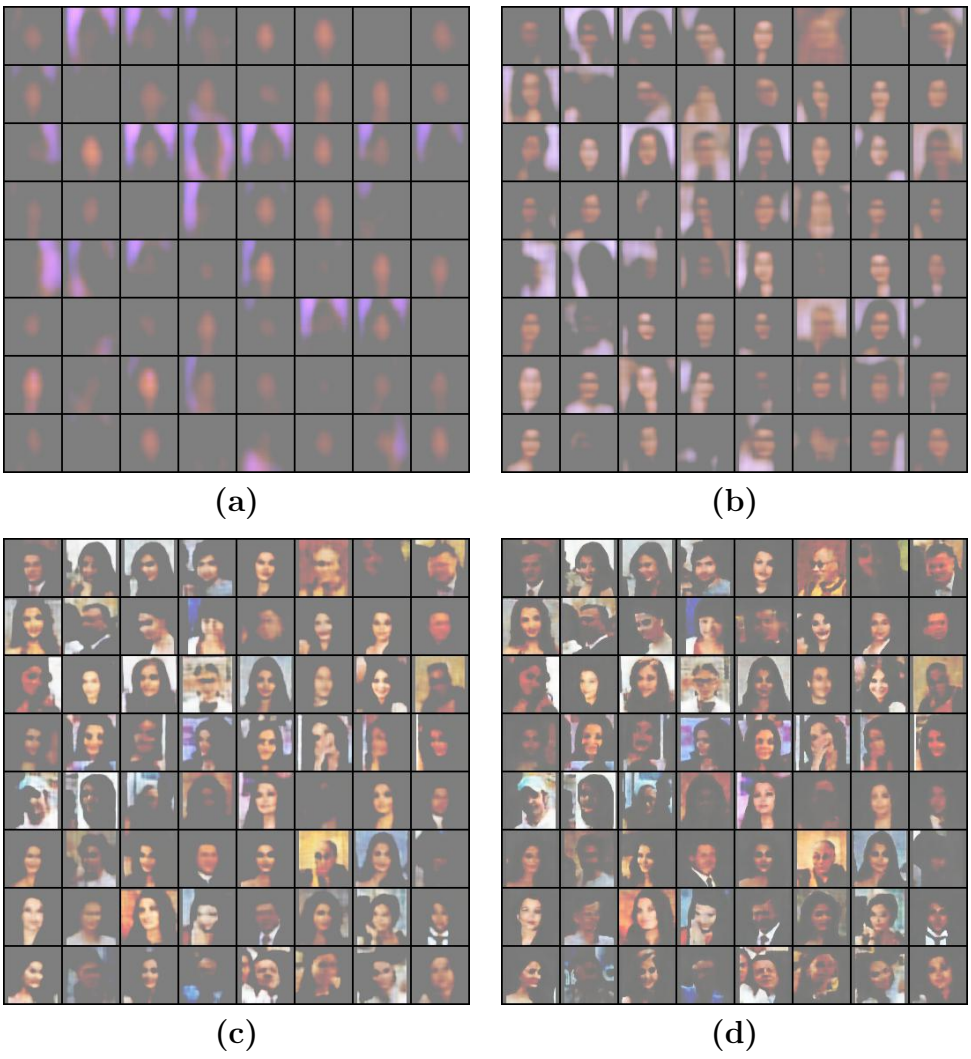


Figure 5: DFC-VAE Training result on face at epoch **(a)** 0 **(b)** 10 **(c)** 1000 **(d)** 500

4 Use GAN and VAE to generate cat face

With the same GAN and VAE method mentioned in section 2 and section 3, I can apply them on cat images. The training data set comes from the Internet (https://web.archive.org/web/20150703060412/http://137.189.35.203/WebUI/CatDatabase/catData.html). I wrote a script to format the cat images into

64*64*3. The result using GAN is shown in figure 6

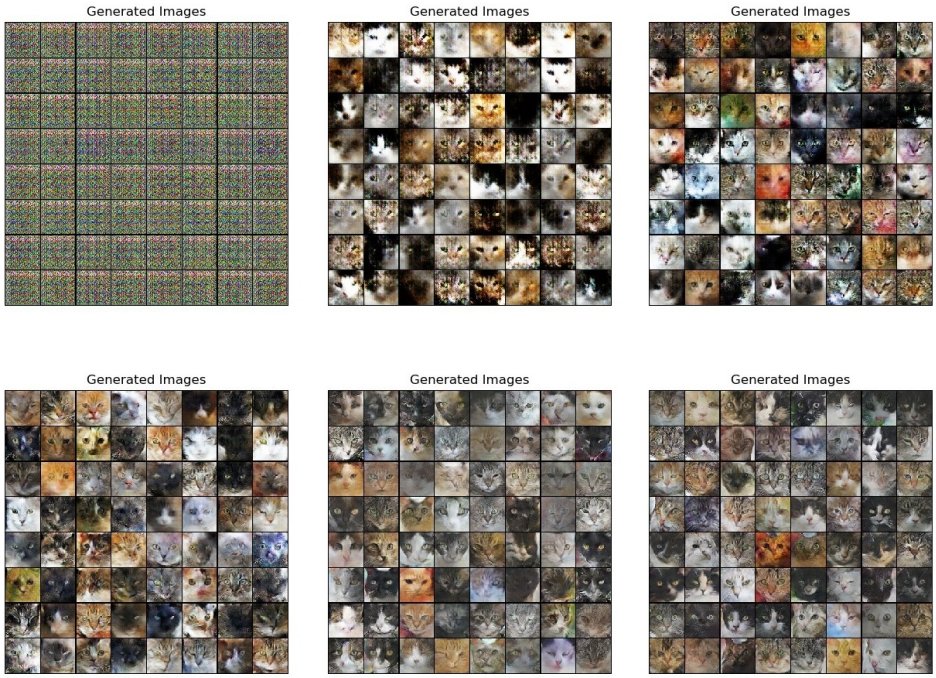


Figure 6: GAN Training result on cat faces after epoch 0, 10, 50, 100, 200, 400

Then DFC-VAE is also used to test on cat dataset, and result is shown in Figure 7.

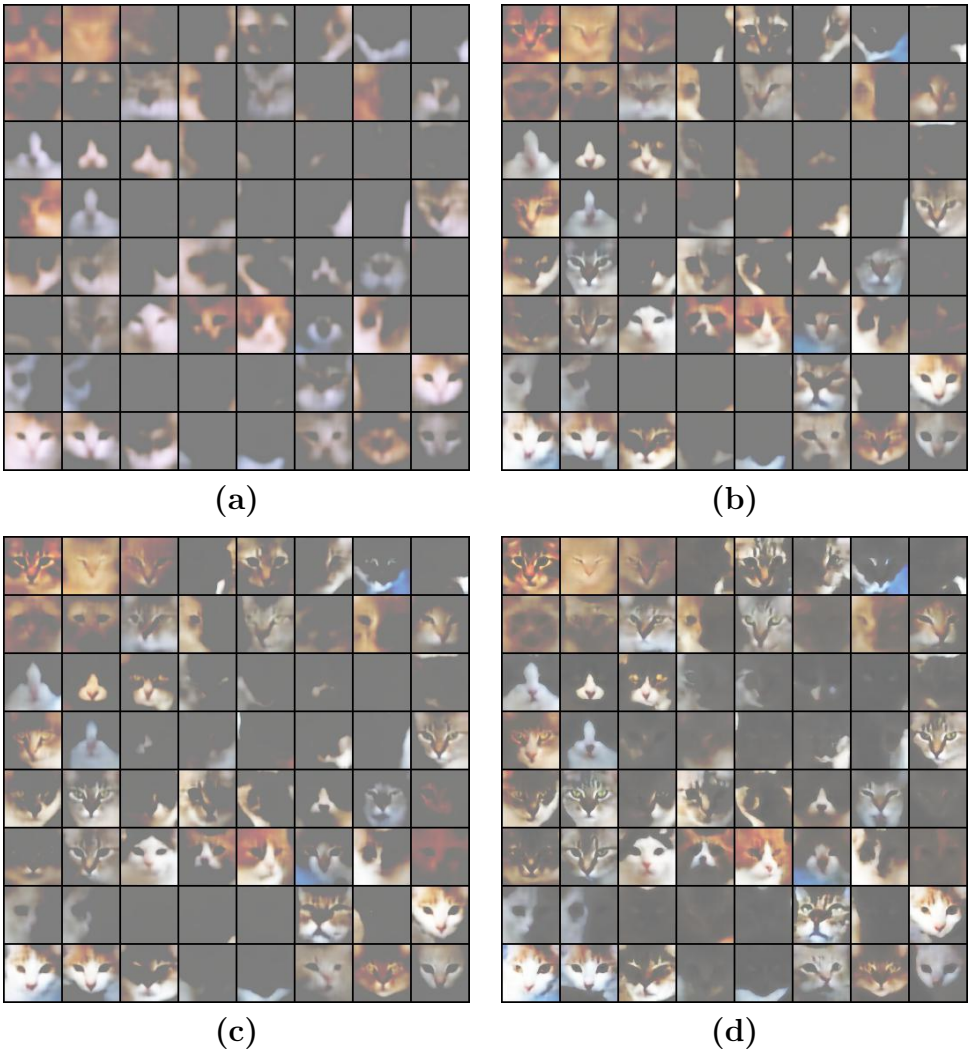


Figure 7: DFC-VAE Training result on cats at epoch **(a)** 0 **(b)** 5 **(c)** 1 **(d)** 50

5 Conclusion

To summarize, in this project, different types of ML models were used to generate fake images of hard-writing characters, human faces and cat faces. We can see that, even with the same model(human faces and cats), the results on cat faces are better than human faces. There can be two possible reasons,

one is the human faces have much more features than cats, so it will be harder to train on human faces. The other one might be, we as human has high sensitive recognition to human faces, so with two images with same loss value, we will think the cat one is having better quality than the humna face.

Another important thing is that, the results can be very much dependent on the model. For example, I also tried to run this project(https://github.com/bhpfelix/Variational-Autoencoder-PyTorch), which is a 6 layers(fairly deep) of 2D convolutional networks. However, when I ran it on my dataset, the results from the author is much better than mine(mine is in in Figure 8).

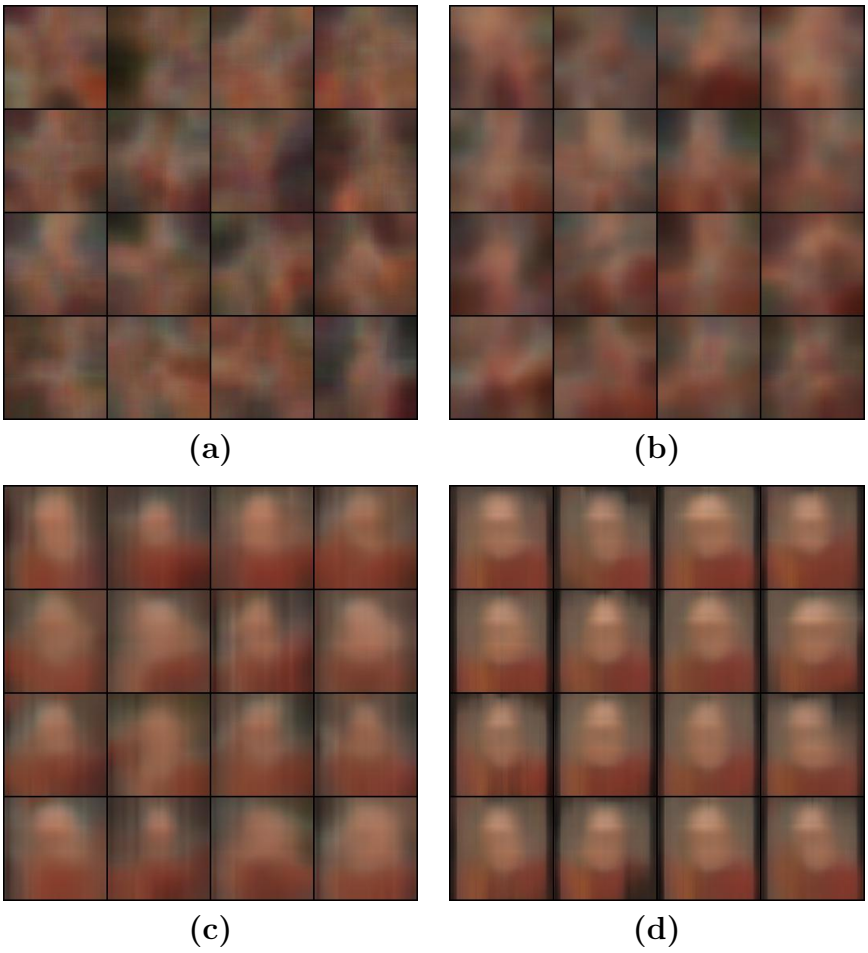


Figure 8: Another VAE Training result on faces at epoch **(a)** 5 **(b)** 10 **(c)** 50 **(d)** 100

I think, this is caused by the size of the inputs, the original input is 8*8 images per batch, but my desktop doesn't have enough memory to run it, so I shrink it down to 4*4 images per batch. This might cause the network to be too deep to find the correct configuration.

In all, tuning all these models can be very subtle works, and there's no too many effective theory guiding people to tune these parameters. Everything seems like depending on the researchers' experience. The choice of model, and how to train a ML model are definitely important topics need to be solved in the future.

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[2] Xianxu Hou, Linlin Shen, Ke Sun, and Guoping Qiu. Deep feature consistent variational autoencoder, 2016.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[4] Min Lin. Softmax gan, 2017.

[5] Xudong Mao, Qing Li, Haoran Xie, Raymond Y. K. Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks, 2016.

[6] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.