CS492(E) Machine Learning for Computer Vision - CourseWork2

Nguyen Tuan Kiet – 20200734, Vu Trong Kim – 20200883

1. DCGAN on MNIST

We use the DCGAN architecture described in the 41st slide in the lecture 'Generative Networks', which contains 1 linear projection then 4 *transpose convolution* layers in the **generator (G)**, and 4 *convolution* layers then 1 linear projection in the **discriminator (D)**. Besides, as suggest in the lecture, $-E_{z\sim p(z)}[\log(D(G(z)))]$ is used instead of $E_{z\sim p(z)}[\log(1-D(G(z)))]$ to mitigate the vanishing gradient problem when the discriminator is over--confident (especially in early stages, when the dis--criminator is easily trained). We define objective functions to minimize for **D** and **G** as below:

$$loss\_D = BCE(D(x),\ 1) + BCE(D(G(z)),\ 0)$$

$$loss\_G = BCE(D(G(z)),\ 1),$$

BCE: binary cross-entropy

A fixed normal-distribution noise is used to generate images after finishing training. In this part, the model's behaviors are discussed with respect to different hyperparameters, architecture depths and several improvement techniques.

1.1. Hyperparameter for DCGAN

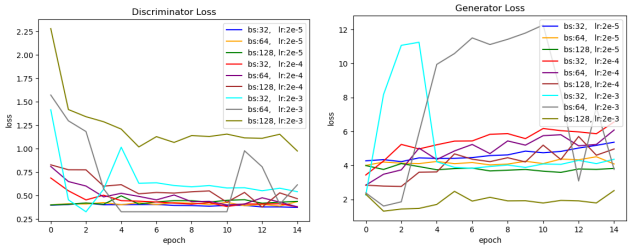

Fig 1. Discriminator and generator losses for different hyperparameters.

Multiple hyperparameters, namely *batchsize* and *learning rate*, are adjusted to conclude the best combination, which would be fixed to investigate other factors. From the results of training with 15 epochs shown in **Fig 1**, overall, learning rate of 2e-4 works relatively well regardless of batch size, among them batch size 128 showed the most stable curve. Using a larger learning rate causes large fluctuation of loss curves. A smaller learning rate leads to slow updates of models, and the curves almost remain unchanged. The generated images for the chosen hyper--parameters setting is shown in **Fig 2.** The histogram on the right side is obtained by our human recognition ability. It can be seen that **G** can generate like-digit images. Overall the digits are easy to recognize, however, some of them can not be classified (row 5, column 6). Also, the result is not diversified when there are only small amounts of images for digits 2, 5, 6, 9.

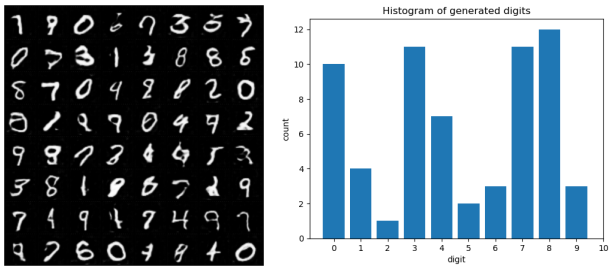

Fig 2. Generated images and the histogram of them.

1.2. Architecture depth for DCGAN

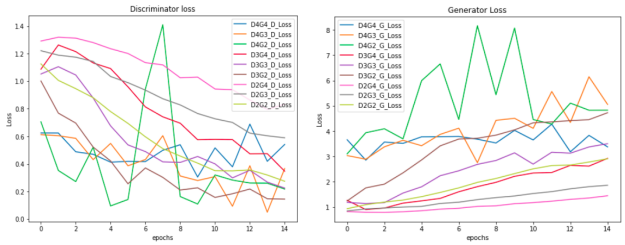

Fig 3.1. Discriminator and generator losses (DxGy accounts for depth of D and G are x,y).

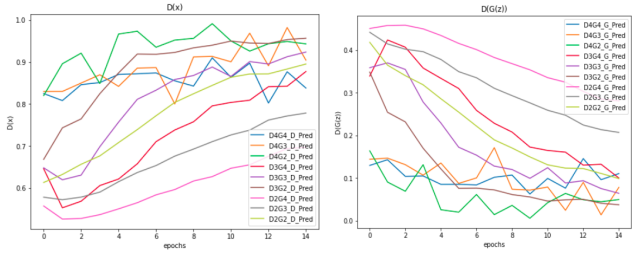

Fig 3.2. Discriminator and generator discriminator prediction (DxGy accounts for depth of D and G are x,y).

We fix the hyperparameter that yielded the optimal result (batch size: 128, learning rate 2e-4) in part 1.1 to investigate models' performance with respect to architecture depth. This figure is measured using **the number of convolution layers** in the **generator** and discriminator. We evaluate the generative model's performance by varying the number of convolution layers in generator and discriminator among three values 2,3,4.

From **Fig 3.1**, it is observable that when the generator architecture is deeper than that of discriminator, the discriminator loss is relatively higher, whereas the generator loss is lower (discriminator is inadequately trained, thus generator learns from inadequate information) along with low D(x) prediction (**Fig 3.2**). This situation should be avoided as discriminator acts as a guidance for generator, then it should be properly trained to learn expressive features of real objects

and boundaries between classes to guide generator on how images should be created. The same trend takes place when both depth of D and G equal 2 (with poor outputs despite plausible figures' graphs), which can also be attributed to both models being insufficiently trained. **Fig 4.a** illustrates an instance (D2G3) of these cases, when images created show little expressive features. With 3 convolution layers in discriminator, mode collapse is observed with generated images (**Fig 4.b**). This happens due to the fact that generator tries to "please" discriminator by outputting the same high-performance images and ignores, desensitizes the impact of noise z (in four illustrated plots, this situation can be observed in higher D(G(z)), higher confidence of these models compared to that of, for example D=4). With D equals 4, the generative model learns relatively well, thus generating fairly discriminative outputs (**Fig 4.c**), except for the case when G equals 2 with drastic changes during 15 epochs, and generating poor, noisy outputs (**Fig 4.d**).

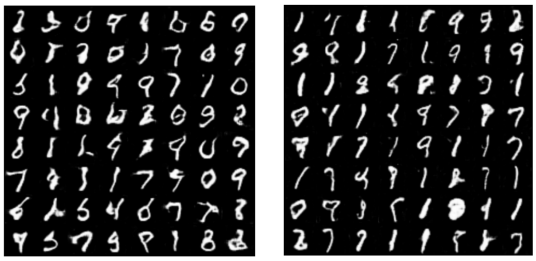

Fig 4.a, 4.b . Generated images of D2G3 and D3G3 respectively.

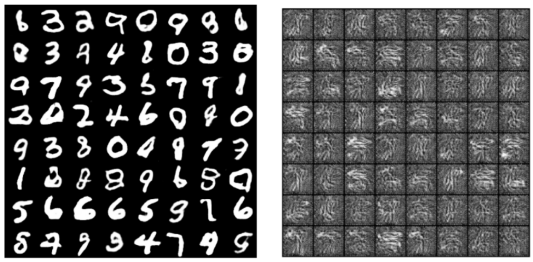

Fig 4.c, 4.d . Generated images of D4G4 and D4G2 (epoch 8).

1.3. Improvement Techniques

a. Virtual Batch Normalization.

In order to guarantee the diversification of outputs over batches from the generator, we use *virtual batch normalization (VBN).* Each image in a training minibatch is processed independently from others, using the mean and variance of the reference batch. The first minibatch feeded into the generator is used as the reference batch. The statistic for an input *x* using the reference batch is computed as below:

$$mean = \frac{x + n_{ref} * mean_{ref}}{1 + n_{ref}}, \; std = \sqrt{mean\_sq - mean^2}$$

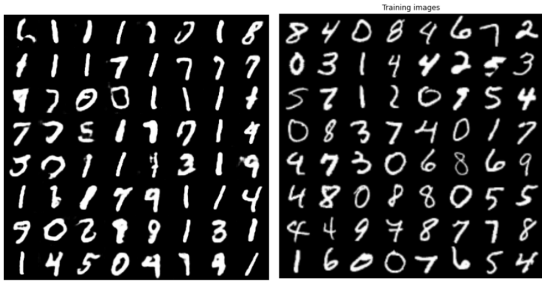

Fig 5.a, 5.b . Generated images and reference images using VBN on **G, D**.

Fig 5.a shows the generated images when applying VBN to both **G** and **D**. *Mode collapse* occurs when **G** produces a lot of images of digit 1 even when the reference set of VBN in **D** is diversified (**Fig 5.b**). Our hypothesis about this issue is that the reference statistic on **D** gives a guidance signal to **G** to improve the ability to generate high quality images of digit 1, which can trick **D** easily. We decide to use the Pytorch BatchNorm2d in the discriminator to remove the bias source, and obtain the result in **Fig 6** , with the same input noise. **G** still biases when generating more images of digit 0, 1, 3, 7, 9 than others, however the distribution is more diversified.

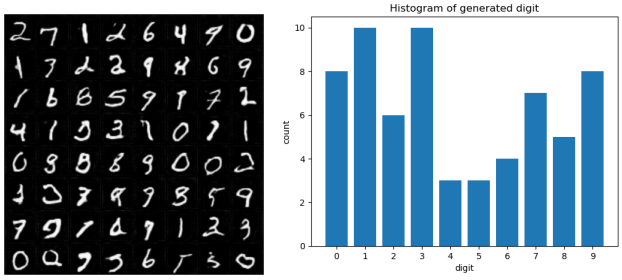

Fig 6. Generated images and digit histogram when using VBN on **G.**

b. Balance G and D

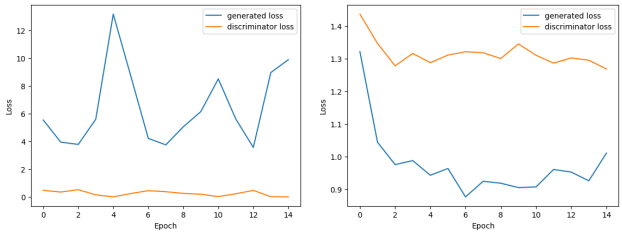

Fig 7. Losses of hypothesis 1 and 2 training are from left to right respectively.

Fig 8 . Generated images for hypothesis 1 and 2 are from left to right respectively.

We conduct 2 training schemes for 2 hypotheses. First hypothesis, the **discriminator** should be good at categorizing fake, real images to give good guidance for the **generator**, so after **D** is trained 5 steps, train **G** 1 step. Second hypothesis, discriminator is easy to overpower generator**,** which leads to a large gradient of **G**, so **G** should be trained more than **D**. From the results shown in **Fig 7** and **Fig 8,** the second hypothesis work well when both $loss\_D$ and $loss\_G$ are small, and the generated images look much clearer than those of hypothesis 1 because the discriminator is already very good from the very early stages. Due to this reason, we do not try deeper architecture for **D**. This design choice (training iteration) is worth testing when encountering new GAN problems.

1.4. Discussion

Weight initialization is essentially important for both G and **D** to avoid vanishing gradient problems and facilitate better learning (prevent converging to local minimums or oscillating over plateaus). **Fig 16 (Appendix)** describes the result when the model is poorly trained without weight initialization.

D should be deep enough to extract enough expressive information for G to learn. Similar depth of D leads to relatively the same behavior of learning.

During the training process, as plausible plots can be the result of both underperformed models or mode collapse, visual evaluation is necessary. Also, as the black-box manner of GAN during G and D in the zero sum game, its behaviors can be sensitive to small or random on-training signals and yield strange results (**Fig 4.c)** thus multiple rounds of a single setting are necessary to truly comprehend the performance.

2. CGAN on MNIST

Our CGAN model is also designed mostly based on the architecture described in the 49st slide in the lecture 'Generative Networks'. In this architecture, both noise vector and one-hot label are projected into 4*4*256 dimension space in the generator firstly. These representations are then reshaped to be concatenated before going into the same process as DCGAN containing transpose convolution layers and fully connected layers. Similarly, one-hot labels are also embedded to concatenate with outputs from generators, then go through convolution layers. To evaluate the model's performance, besides loss function and real/fake prediction confidence, *Inception Scores* (IS) and *TARR* are utilized based on the LeNet pre-trained classifier on MNIST dataset, which has achieved 0.990 accuracy in the test set. The IS calculated on MNIST test data is 9.81. We use 10,000 labels, each digit has 1000 occurrences to perform evaluation. This evaluation concerns various hyper-parameters, architecture depths and improvement techniques.

2.1. Hyperparameter for CGAN

Setting	TARR	IS
BS: 32; Lr: 2e-5	0.977	9.71
BS: 64; Lr: 2e-5	0.969	9.56
BS: 128; Lr: 2e-5	0.961	9.47
BS: 32; Lr: 2e-4	0.988	9.82
BS: 64; Lr: 2e-4	0.989	9.81
BS: 128; Lr: 2e-4	0.988	9.81
BS: 32; Lr: 2e-3	0.100	1.00
BS: 64; Lr: 2e-3	0.6192	4.02
BS: 128; Lr: 2e-3	0.100	1.09

Table 1. TARR and IS of different hyperparameter settings.

Once again, *batch size* and *learning rate* are tweaked to choose the optimal values. Training results with 10 epochs are in **Table 1** show that *batch size* 128 and *learning rate* 2e-4 still work well and we use this setting for consistency with part 1. In addition, smaller learning rate causes slow learning, while large learning rate leads to explosive gradient update and the model can not learn, which leads to smaller and poor *TARR* and *IS*. Moreover, smaller batch size leads to better results, but the training process is slower. The generated images for this chosen setting when passing fixed noise and an array of evenly--distributed labels to the generator are in **Fig 9.** It is easy to recognize that the generator incorporated the labels' signal well when images produced matching with given labels. Also, the quality of images is much better than that of DCGAN when there is no noise and all digits can be recognized easily by humans. This can be explained that the labels' signal gives a clear guidance to the generator to produce images of the given category. The *IS, TARR* on fake images of the chosen setting are similar to those of test data.

Fig 9. Generated images for batchsize 128, learning rate 2e-4.

Another key difference between results of CGAN and DCGAN is that *loss_D* and *loss_G,* D(x) and D(G(z)) are more balanced (**Fig 10**), when they fluctuate around 1.2 and 0.5 respectively. The high quality generated images can explain this balance that **G** learns well and can trick **D**.

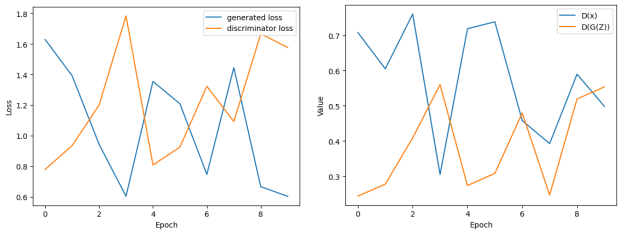

Fig 10. Training loss and probability to realize images as real or fake.

2.2. Architecture depth for CGAN

The same settings as part 1 is also used in this part regarding considering the number of convolution layers as architecture depth of generator/discriminator (values range from 2 to 4).

In general, similar to findings concerning different hyperparameters, plausible convergences are obtained in CGAN with a wide range of architecture depths. There are still outlier cases (with (depth of D, depth of G) = (4,4); (4,2); (3,2)) when discriminator is constructed deeper than generator, thus leading to significantly lower discriminator and higher generator losses (**Fig 17 Appendix**), overconfident D(x) predictions (**Fig 18 Appendix**) or great fluctuations of inception and accuracy over epochs (**Fig 12**). In spite of this, generated images show great expressiveness over classes even from above outliers. This trend indicates the excessive power of feeding labels to learning models, which diminish the effect of architecture depth. That being said, there are a number of problems that are noticeable through generated images such as poor performance in specific classes (**Fig 11.a**), misclassified generation (**Fig 11.b**), **in-class** mode collapse (**Fig 11.c**), which happens arbitrarily and regardless of model depths. This again illustrates the “black-box” nature of GAN.

Fig 11.a. Model failed to generate images of specific classes (D=3, G=4).

Fig 11.b. Model generated images in a wrong class (3 to 2) (D=4, G=4).

Fig 11.c. Mode Collapse generated images (D=2, G=3).

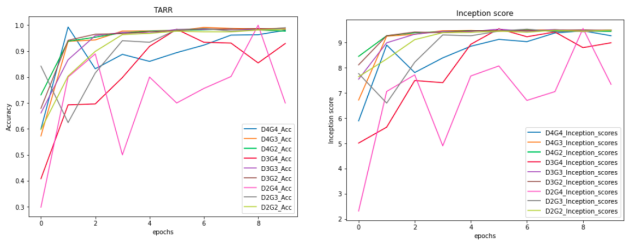

Fig 12. TARR and Inception Score of different architecture (DxGy denotes depth of D, G are x,y)t

2.3.Improvement Techniques

Setting	TARR	IS
CGAN base (1)	0.988	9.81
part 2.3.a	0.988	9.79
part 2.3.b more update on **D**	0.977	9.73
part 2.3.b Deep **D** (2)	0.989	9.82
part 2.3.c (3)	0.991	9.83

Table 2. TARR and IS from different techniques.

a. Virtual Batch Normalization

VBN is used to increase the diversity over a batch of digit shapes. From **Table 2**, the results of training with VBN in the generator are a bit worse. The fake images have some minor noises. In conclusion, because CGAN can assure diversification via label signal and careful design of architecture, in other experiments with CGAN, we use the normal BatchNorm.

b. Balance G and D

Intuitively, as mentioned in the first hypothesis of part 1.3.b, **discriminator** should be good at classifying fake/real images. Thanks to label feeding, the balance between G and D is, to some extent, obtained. Yet, we would try to train D more in an attempt to extract more information for G to learn.

Firstly, we tried to train **D** with more steps than **G**, and obtain worse results. (**Table 2**, part 2.3.b more update on **D**). One reason might be that the **G** is only updated at certain iterations during in one epoch, so **G** can not learn well.

The trend is observed when increasing depth of D to 4 (as discussed in 2.2). Yet, we still reported the result of the most stable architecture when depth D equals 4 (D=4, G=3) in **Table 2** for further observation in the following.

c. One-sided Label Smoothing

Stick with the architecture of **D** with depth 4 in part b, in order to improve gradient of *loss_G*, we use one-sided label smoothing as suggested in the lecture, and get a new formula for *loss_D:*

$$loss\_D = BCE(D(x),\ alpha) + BCE(D(G(z)), 0)$$

We set *alpha=0.9* and obtain the best result at the last row of **Table 2**. That being said, the difference is significant enough to claim the advantage of the technique. From **Fig 13,** the images generated are clear and diversified in shape, while *loss_D* is a bit higher at last epochs in comparison to **Fig 10** due to the effect of one-sided label smoothing. When *alpha*=0.7 is used, the variation range of training loss does not change much, and we acquire a closed result: *IS* 9.83, *TARR* 0.990.

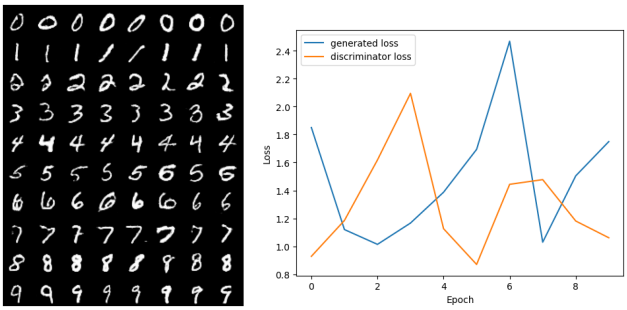

Fig 13. Generated images and training loss using deeper **D** with label smoothing.

2.4.Different metrics

We propose an approach to measure the performance of the model by calculating classifier accuracy on data of each digit. The model might be optimized to produce high quality images of some categories while performing worse on the other types. We audit the generalization of the model on every digit. From **Fig 14**, it can be seen that the overall trend is that the classifier works better on generated images of digits 0,1,6,9 than those of other digits. This can be reasoned by firstly, the ability of the classifier, and secondly, the quality of generated images. However, the difference of accuracy is negligible, and the geometric mean is close to *TARR* in **Table 3**, so we can conclude both generator and classifier work well.

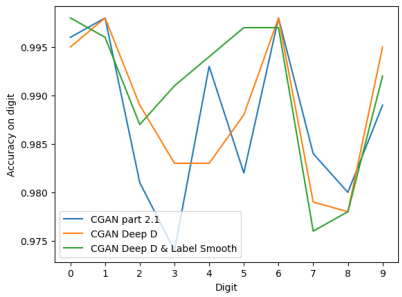

Fig 14 . Classify accuracy on each digit of different settings.

2.4. Conclusion

With label information, the model learned much better in CGAN compared to that of DCGAN, which actually has overshadowed other techniques' impact (as deeply discussed in 2.2 and 2.3). Yet, even with the introduction of inception score and TARR, visual evaluation is still needed to detect mode collapse and assess the sensitive behavior of GAN when the same setting can behave differently in separate training times.

Experiment	**Geometric Mean Acc**	**TARR**
CGAN base (1)	0.987	0.988
CGAN Deep **D** (2)	0.989	0.989
CGAN Deep **D** vs Label Smooth (3)	**0.991**	**0.991**

Table 3 . Geometric mean of accuracy on each digit class.

3. Re-training the handwritten digit classifier

The chosen classifier follows LeNet architecture, which contains two convolution layers with maxpool and three fully connected layers. Adam optimizer with learning rate 0.001 is used for the training process. The performance of the classifier on different training data settings is a metric to evaluate the quality of generated images.

3,1, Re-training with generated images

In this part, we use a mix of both real and synthetic data to train the classifier from scratch. There are four settings for the portion of the generated images: 10%, 20%, 50%, 100%. There are three independent sets of generated images from three models mentioned in **Table 3**. From the training result with 10 epochs in **Table 4**, a common trend can be witnessed that with more synthetic data, the classifier's performance goes down a considerable number, about 2% on test accuracy. This is a clear evaluation metric for the lowerer quality of generated images in comparison to the original data. With only 10-20% of real images, the model has yielded comparable results compared to that on the original model (0.99 accuracy).

Generator	**Test Accuracy** of four **synthetic** portion settings			
	10%	20%	50%	100%
CGAN base (1)	0.990	0.989	0.985	0.965
CGAN Deep **D** (2)	0.989	0.989	0.988	0.969
CGAN Deep **D** vs Label smooth (3)	0.987	0.986	0.987	0.969

Table 4. Classifier accuracy of different training sets.

3.2. Change training strategy

In this part, multiple training strategies regarding the use of synthetic data are tested with two purposes: further evaluate on the quality of generated images, and improve the performance of the classifier.

a. Initialize with synthetic data, finetune by real data

Firstly, the classifier is trained with all generated images in 7 epochs, then a part of real data is used to finetune in 3 epochs. The overall trend in **Table 5** is that with more real images, the performance of the model increases, but it can not surpass the real-image accuracy. That being said, it showed plausible improvement compared to only training with generated data (even only with 3 more epochs).

Generator	**Test Accuracy** of four **real**-portion settings			
	10%	20%	50%	100%
CGAN base (1)	0.980	0.984	0.986	0.988
CGAN Deep **D** (2)	0.981	0.986	0.985	0.991
CGAN Deep **D** vs Label smooth (3)	0.978	0.982	0.984	0.989

Table 5. Classifier accuracy on different sets of real data for fine tuning.

b. Initialize with real data, finetune by synthetic data

Firstly, the classifier is trained with all real data in 5 epochs, then a percentage of generated images is used to finetune in 5 epochs. Similar to part 3.1, the more generated images are used, the worse results. **Table 6** illustrates the process when feeding generated images into the model in the last first epoch actually has worsened the model performance.

Generator	**Test Accuracy** of two synthetic-portion settings	
	50%	100%
CGAN base (1)	0.984	0.979
CGAN Deep **D** (2)	0.984	0.986
CGAN Deep **D** vs Label smooth (3)	0.984	0.982

Table 6. Classifier accuracy on different sets of synthetic data for fine tuning.

c. Training with realistic synthetic data

Generator	**Test Accuracy** of different confident thresholds (t)	
	t=0.98	t=0.99
CGAN base (1)	0.986	0.987
CGAN Deep **D** (2)	0.986	0.985
CGAN Deep **D** vs Label smooth (3)	0.987	0.988

Table 7. Classifier accuracy on confident images.

The synthetic data is passed through the pre-trained classifier used in part 2 and only images having high confidence prediction are used for retraining the classifier. **Table 7** shows that even when only high quality synthetic images are chosen to train, the classifier's performance is still lower than when it is trained with real data, yet showed great improvement compared to when purely training on generated images (0.96)

d. Training models with more epochs.

As noisy generated images can slow down the learning process of a model, we would try if the classifier would eventually become better with more training epochs. As observed in Fig, the model does not improve much after 10 epochs. Thus, the model trained with real data strictly outperformed the others over 50 epochs.

Fig 15 . Validation accuracy of the classifier trained on different sets of data.

3.3. Conclusion

Plausible but still worse performance of the classifier being trained on generated images indicates the lower quality of synthetic data. A number of techniques are introduced in part 3.2 in an attempt to improve the classification model. Using confident thresholds showed its effectiveness when yielded the best result without using real data in the training process.

Appendix

1. DCGAN on MNIST

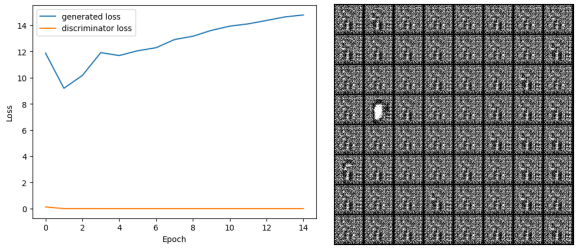

Fig 16 . Training loss and generated images without weight initialization.

2.CGAN on MNIST

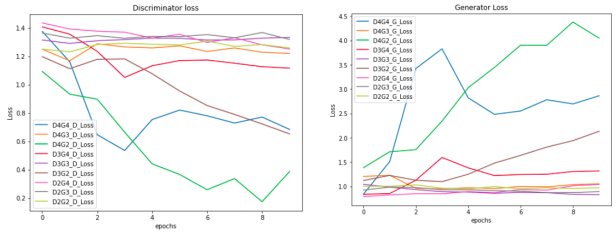

Fig 17. Discriminator and generator losses (DxGy accounts for depth of D and G are x,y).

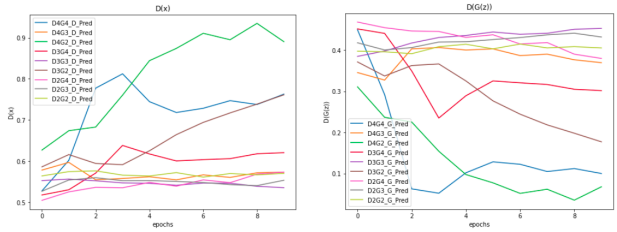

Fig 18. D(x) and D(G(x)) (DxGy accounts for depth of D and G are x,y).

Fig 19 . Generated images using VBN on **G** in part 2.3.a.