Best Selection of Generative Adversarial Networks Hyper-Parameters using Genetic Algorithm

Fajr Alarsan (fajr.ib.alarsan@gmail.com)
Higher Institute for Applied Science and Technology https://orcid.org/0000-0001-6057-8339
Mamoon Younes
Damascus university

Research

Keywords: Generative Adversarial Networks (GAN), MNIST dataset, Image 1 synthesis, Generator, Discriminator, Genetic Algorithm

Posted Date: November 6th, 2020

DOI: https://doi.org/10.21203/rs.3.rs-95571/v1

Version of Record: A version of this preprint was published at SN Computer Science on May 20th, 2021. See the published version at https://doi.org/10.1007/s42979-021-00689-3.

RESEARCH

Best Selection of Generative Adversarial Networks Hyper-Parameters using Genetic Algorithm

Fajr Ibrahem Alarsan* and Mamoon Younes

Fajr Ibrahem Alarsan: fajr.ib.alarsan@gmail.com, Informatics and Decision Supporting Systems, Higher Institute for Applied Sciences and Technology
Mamoon Younes: Myyounes60@gmail.com, Faculty of Computer and Automation Engineering, Damascuse University

*Correspondence: fajr.ib.alarsan@gmail.com
Informatics and Decision Supporting Systems, Higher Institute for Applied Sciences and Technology, Damascus, Syria
Full list of author information is available at the end of the article

Abstract

Generative Adversarial Networks (GANs) are most popular generative frameworks that have achieved compelling performance. They follow an adversarial approach where two deep models generator and discriminator compete with each other In this paper, we propose a Generative Adversarial Network with best hyper-parameters selection to generate fake images for digits number 1 to 9 with generator and train discriminator to decide whereas the generated images are fake or true. Using Genetic Algorithm technique to adapt GAN hyper-parameters, the final method is named GANGA:Generative Adversarial Network with Genetic Algorithm. Anaconda environment with tensorflow library facilitates was used, python as programming language also used with needed libraries. The implementation was done using MNIST dataset to validate our work. The proposed method is to let Genetic algorithm to choose best values of hyper-parameters depending on minimizing a cost function such as a loss function or maximizing accuracy function. GA was used to select values of Learning rate, Batch normalization, Number of neurons and a parameter of Dropout layer.

Keywords: Generative Adversarial Networks (GAN), MNIST dataset, Image synthesis, Generator,Discriminator, Genetic Algorithm

1 Introduction

Many machine learning systems look at some kind of complicated input (say, an image) and produce a simple output (a categorical label like, "cat" or numeric label like 1, 2, or any other number that represent a class). By contrast, the goal of a generative model is something like the opposite: take a small piece of input-perhaps a few random numbers or vector of noise-and produce a complex output, like an image of a realistic-looking face. A generative adversarial network (GAN) is an especially effective type of generative model, introduced only a few years ago, which has been a subject of intense interest in the machine learning community [1].

The idea of a GAN is creating realistic images from scratch can seem like magic, but GANs use special method to turn a vague, seemingly impossible goal into reality. The method is to use randomness as an ingredient. At a basic level, it would not be very exciting if we build a system that produce the same face each time it ran. Thinking in terms of probabilities, it also helps us translate the problem of generating realistic images into a natural mathematical framework. The system should learn about which images are likely to be faces, and which are not. Mathematically, this involves modeling a probability distribution on images, that is, a function that tells us which images are likely to be faces and which are not. This type of problem modeling a function on a high dimensional space is exactly the sort of thing neural networks are made for. GAN must set up this modeling problem as a kind of contest. This is where the "adversarial" part of the name comes from. The key idea is to build not one, but two competing networks: a generator and a discriminator. The generator tries to create random synthetic outputs, while the discriminator tries to tell these apart from real outputs. The hope is that as the two networks will both get better and better with the end result being a generator network that produces realistic outputs. Figure 1 below explains GAN simply:

[Figure 1 about here.]

Generative adversarial networks are neural networks that learn to get samples from a special distribution (the "generative" part of the name) as input, and they do this by setting up a competition (hence "adversarial"). So the main concept behind this project is the generative adversarial network. GAN is about creating stuff and this is hard to compare other deep leaning fields. In other words, Generative adversarial networks (GANs) are deep neural net architectures included of two nets, pitting one against the other

In machine learning, a hyper-parameter is a parameter whose value is used to control the learning process. By contrast, the values of other parameters (typically node weights) are derived via training. Hyper-parameters can be classified as model hyper-parameters, that cannot be inferred while fitting the machine to the training set because they refer to the model selection task, or algorithm hyper-parameters, that in principle have no influence on the performance of the model but affect the speed and quality of the learning process. The choice of hyper-parameters can significantly affect the resulting models performance, but determining good values can be complex [2].

Figure 2 below explains hyper-parameters and default model parameters

[Figure 2 about here.]

Hyper-parameter Optimization or Hyper-parameter Tuning can be defined as choosing the right set of values in building a machine learning model, in our case the machine learning model is GAN.

Any GAN hyper-parameters can be summarized to:

- Learning rate
- Batch size
- Number of epochs
- Generator optimizer
- Discriminator optimizer
- Number of layers
- Number of units in a dense layer
- Activation function
- Loss function

- Properties such as: keep probability of dropout layer and Batch Normalization momentum

GANs potential is very huge because they can learn to mimic any data. So use of GAN we create worlds similar to our own in any domain: image, anime, news anchor, speech.

1.1 GAN Applications

GAN has interesting applications that are commonly used in the industry right now.

- GANs for Image Editing: Most image editing software these days do not give us much flexibility to make creative changes in pictures. For example, let us say we want to change the appearance of a 90-year-old person by changing his/her hairstyle. This can not be done by the current image editing tools out there. Another similar application is image de-raining (or literally removing rainy texture from images [3].
- Using GANs for Security: A constant concern of industrial applications is that they should be robust to cyber attacks. There is a lot of confidential information on the line! GANs are proving to be of immense help here, directly addressing the concern of adversarial attacks. These adversarial attacks use a variety of techniques to fool deep learning architectures. GANs are used to make existing deep learning models more robust to these techniques. How? By creating more such fake examples and training the model to identify them. Pretty clever stuff [4].
- Generating Data with GANs: The availability of data in certain domains is a necessity, especially in domains where training data is needed to model supervision deep learning algorithms. The health care industry comes to mind here. GANs shine again as they can be used to generate synthetic data for supervision. That is right! You know where to go next time you need more data [5].
- GANs for 3D Object Generation: Game designers work countless hours recreating 3D avatars and backgrounds to give them a realistic feel. It certainly takes a lot of effort to create 3D models by imagination. GAN has incredible power to be used to automate the entire process and create 3D models. [6].

There are also other applications, So Gan is very important, interesting, and useful tool to be understood and studied well.

2 Background and Related work

There are many works related to GAN hyper-parameters tunning.

In [7], the authors tried to find the appropriate structure more conveniently and efficiently. A method with multi-objective algorithm was proposed to obtain the optimal structure for the GANs. In the proposed method, the non dominated sorting genetic algorithm II (NSGA II) is utilized to optimize the hyper-parameters and structure of deep convolution generative adversarial network (DCGAN). The experiments are conducted on MNIST and Malware datasets demonstrate the efficiency and high performance of proposed method.

In [8], authors proposes the use of Conditional Generative Adversarial Networks (cGANs) for trading strategies calibration and aggregation. They provide a full methodology on: (i) the training and selection of a cGAN for time series data; (ii) how each sample is used for strategies calibration; and (iii) how all generated samples can be used for ensemble modeling. They have designed an experiment with multiple trading strategies, encompassing 579 assets. They compared cGAN with an ensemble scheme and model validation methods, both suited for time series. The results suggest that cGANs are a suitable alternative for strategies calibration and combination, providing out performance when the traditional techniques fail to generate any alpha. Their problem can be decomposed into two tasks model validation and hyper-parameter optimization. For each hyper-parameter, they have a space of values, they hope the desire that this space contains the best value of hyper-parameter. Best value will give a max value of accuracy or min value of loss function or error.

In [9], conditional version of Generative Adversarial Networks (cGAN) is used to approximate the true data distribution and generate data for the minority class of various imbalanced datasets. The performance of cGAN is compared against multiple standard oversampling algorithms. They present empirical results that show a significant improvement in the quality of the generated data when cGAN is used as an oversampling algorithm. The hyper-parameters of cGAN are the dimension of the noise space, the hyper-parameters related to the G (Generator) and D (Discriminative) networks architecture as well as their training options.The hyper-parameter tuning of the classifiers and the various oversampling algorithms was done in order to maximize the AUC: Area Under the Curve of the validation set.

In [10], authors propose and study an architectural modification (self-modulation), which improves GAN performance across different data sets, architectures, losses, regularizers, and hyper-parameter settings. They found that self-modulation allows the intermediate feature maps of a generator to change as a function of the input noise vector. While reminiscent of other conditioning techniques, it requires no labeled data. They also observe a relative decrease of 5% to 35% in FID (Frechet Inception Distanc). They made a modification to the generator and that leads to improved performance (86%) of the studied settings.

authors found that most models can reach similar scores with enough hyper-parameters optimization and random restarts. They suggested that improvements can arise from a higher computational budget and tuning more than fundamental algorithmic changes. To overcome some limitations of some metrics, they also proposed several data sets on which precision and recall can be computed. Their experimental results suggested that future GAN research should be based on more systematic and objective evaluation procedures.

3 Our Work

In this paper, we studied the effects of hyper-parameters in GAN, how the affect the generator and discriminator. So our work is distinguished by:

- Choosing best Learning rate for GAN
- Choosing best dropout keep probability
- Choosing best batch size
- Choosing best number of neurons in dense layers

4 Methods

Here, the concept of our work is explained in details, Genetic algorithm was used to get best values of some hyper-parameters.

4.1 Genetic Algorithm

According to [11]. Genetic algorithm (GA) is a metaheuristic [a] inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on biologically inspired operators such as mutation, crossover and selection. Genetic Algorithms are used to provide quality solutions for optimization problems and search problems. To get more details about GA, see [12].

4.2 GAN

GAN consists of two main part Generator and Discriminator. The training phase of the discriminator and generator are kept separate. In other words, the weights of the generator remain fixed while it produces examples for the discriminator to train on, and vice versa when it is time to train the generator

The discriminator training process is comparable to that of any other neural network. The discriminator classifies both real samples and fake data from the generator. The discriminator loss function penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real, and updates the discriminator's weights via back-propagation, discriminator has loss and accuracy function to be validated.

Similarly, the generator generates samples which are then classified by the discriminator as being fake or real. The results are then fed into a loss function which penalizes the generator for failing to fool the discriminator and back-propagation is used to modify the generator's weights, generator has loss function.

As the generator improves with training, the discriminator performance gets worse because the discriminator fails to distinguish between real and fake. If the generator succeeds perfectly, then the discriminator has a 50% accuracy (no better than random chance). The later poses a real problem for convergence of the GAN as a whole. If the GAN continues training past the point when the discriminator is giving completely random feedback, then the generator starts to train on junk feedback, and its own performance may be affected. The generator is typically a de-convolutional neural network, and the discriminator is a convolutional neural network [13] when generator is very accurate; i.e its loss function ends to zero, discriminator has bad accuracy and vice versa.

Figure 3 shows Generator (de-convolutional neural network) and Discriminator (convolutional neural network)

[Figure 3 about here.]

4.3 hyper-parameters tuning

Tuning process with GA needs a term to be minimization during GA running. From the above description of GAN, we have Generator loss: g_loss, Discriminator loss:

d_loss and Discriminator accuracy: d_acc
g_loss Minimization → d_acc Minimization ↔ d_loss Maximization
d_acc Maximization ↔ d_loss Minimization → 5 g_loss Maximization

5 Implementation and Results

5.1 Programming environment

According to many references, Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, data processing, predictive analytics, etc.), that aims to simplify package management and deployment. The distribution includes data science packages suitable for Windows, Linux, and macOS. It is developed and maintained by Anaconda, Inc., which was founded by Peter Wang and Travis Oliphant in 2012. Details about python version is show next

[Figure 4 about here.]

With tensorflow version 1.14.0 and python version 3.7.4. Keras version 2.2.5 was used for implementation. Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable error messages. It also has extensive documentation and developer guides [13].

5.2 Basic Model

Basic Generator in GAN model is show in Figure 5:

[Figure 5 about here.]

while each layer in the generator is shown in the Figure 6:

[Figure 6 about here.]

Basic Discriminator in GAN model is show in Figure 7:

[Figure 7 about here.]

The Optimizer was Adam: Adaptive Moment Estimation; Adam is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. It works better (faster and more reliably reaching a global minimum) when minimizing the cost function in training [14].

Discriminator and Generator loss function is "binary crossentropy"; Also called Sigmoid Cross-Entropy loss. It is a Sigmoid activation plus a Cross-Entropy loss, it is independent for each vector component (class), meaning that the loss computed for every Discriminator output vector component is not affected by other component values. That is why it is used for multi-label classification; MNIST dataset is multi-label classification , were the insight of an element belonging to a certain class should not influence the decision for another class. It is called Binary Cross Entropy Loss because it sets up a binary classification problem between C=2 classes for every class in C.

5.2.1 Layers Details

For LeakReLU activation function, Alpha was chosen to be 0.2, where LeakReLU activation function is as follow:

$$f(x) = \alpha * x, \quad if: \quad x < 0$$
$$f(x) = x, \quad if: \quad x >= 0$$

Default value of α is 0.3

For Batch Normalization, momentum is the importance given to the moving average or it is the lag in learning mean and variance, so that noise due to mini-batch can be ignored as described in the equation:

$$\mu_{new} = \beta * \mu_{old} + (1 - \beta) * \mu_{current}$$
$$\sigma^2_{new} = \beta * \sigma^2_{old} + (1 - \beta) * \sigma^2_{current}$$
$$\beta = momentum$$

By default, momentum would be set a high value about 0.99, meaning high lag and slow learning. When batch sizes are small, the no. of steps run will be more. So high momentum will result in slow but steady learning (more lag) of the moving mean. So, in this case, it is helpful. But when the batch size is bigger, as I have used, i.e 5K images (out of 50K) in single step, the number of steps is less. Also, the statistics of mini-batch are mostly same as that of the population. At these times, momentum has to be less, so that the mean and variance are updated quickly. Hence a ground rule is that:

- Small batch size =¿ High Momentum (0.9 to 0.99)
- Big batch size =¿ Low Momentum (0.6 to 0.85)

Dropout layer can be also added to Discriminator, dropout refers to dropping out units (both hidden and visible) in a neural network. Dropout refers to ignoring neurons during the training phase of certain set of neurons by a term named Keep Probability, which is chosen at random. By ignoring; i.e. these units are not considered during a particular forward or backward pass [15]

5.2.2 Genetic algorithm implementation

Using geneticalgorithm library to implement Genetic algorithm, Table 1 show the main code of implementation

[Table 1 about here.]

dimension refers to number of variables that the GA will find best values of them

variable_type refers to types of variables that the GA will find best values of them

variable_boundaries refers to min and max of area limits of variables that the GA will find best values of them
Other parameters can be set using algorithm_param [16].

6 Experiments Results

6.1 Tuning for Discriminator Loss

In this test, mission for GA was to minimize Discriminator Loss, so the returned value from the f(X) function in GA was the Discriminator Loss (Table 1); We can also minimize the value (1-Discriminator accuracy).

6.1.1 Learning Rate

First of all, learning rate was chosen to be tunned, after setting all needed parameters, results gave some values of learning rate that helped to get Discriminator Loss 0 and Discriminator accuracy 100%. Figure 8 show some results during training:

[Figure 8 about here.]

Filter results for only accuracy 100% are shown in Figure 9

[Figure 9 about here.]

max Generator loss was 12.89 and min value was 8.06.
For max loss of generator, learning rate had some values: 0.013,0.019,0.09,0.08,0.00019, most repeated value was: 0.00019. For min loss of generator, learning rate had value: 0.3.

6.1.2 Other parameters results

Repeating previous test to get best values of other parameters (keep probability, Dense neurons, batch size), best values are show in next table

[Table 2 about here.]

6.2 Tuning for Generator Loss

In this test, mission for GA was to minimize Generator Loss, so the returned value from the f(X) function in GA was the Generator Loss (Table 1).

6.2.1 Learning Rate

First of all, learning rate was chosen to be tunned, after setting all needed parameters, results gave some values of learning rate that helped to get Generator Loss 0. Figure 10 show some results during training:

[Figure 10 about here.]

Filter results for only loss 0.0 are shown in Figure 11

[Figure 11 about here.]

max Discriminator loss was 7.97 and min value was 4.78.
For max loss of Discriminator, most repeated value was: 0.171924. For min loss of generator, learning rate had value: 0.0154.

6.2.2 Other parameters results

Repeating previous test to get best values of other parameters (keep probability, Dense neurons, batch size), best values are show in next table

[Table 3 about here.]

Figure 12 below shows results during Generator learning when all best parameters were set

[Figure 12 about here.]

6.3 Discussion

From the results, it can be said that GAN with GA (GANGA) is more useful to be used in all fields. In our case, it is applied to MNIST dataset, but it can be implemented to any other dataset such as fingerprints dataset or any other datasets. Once the user has all details about application, GANGA can be applied tp get best parameters. Comparing with [17], the author validate the value 0.0001 for learning rate and 16 for batch size, with seam loss type, max Discriminator accuracy was 96.25% using ACGAN: Auxiliary classifier generative adversarial network. In our case the Discriminator accuracy was 100% for learning rate 0.00019984 and batch size 64 and keep probability 0.812.

7 Conclusion

A Generative Adversarial Network is presented in this study to to generate fake images for digits from 1 to 9, and train it to classify the results into fake or real. Genetic Algorithm was used to select best values for some hyper-parameters of GAN, results showed the importance of GA in selecting hyper-parameters. As a future work, the structure and design of GAN can be edited with best values of hyper-parameters to make GAN as robust as possible.

List of abbreviations

GAN: Generative Adversarial Network; GA: Genetic Algorithm;

Competing interests

The authors declare that they have no competing interests.

Author's contributions

FIA took the role of performing the literature review, finding best hyper-parameters for GAN. MY took on a supervisory role and oversaw the completion of the work, Proposed data set for implementation and validation. All authors read and approved the final manuscript.

Acknowledgments

Thanks for Mr. AA for his co-operation and help.

Availability of data and materials

The data set is available to public and can be found in python library.

Funding

The authors declare that they have no funding.

Ethics approval and consent to participate

The authors Ethics approval and consent to participate....

Consent for publication

The authors consent for publication.

Authors' information
FA holds bachelor degree in Business Informatics Engineering at Al-Wadee International University, Master degree in Informatics and Decision Supporting Systems.
MY is a Ph.D. in Computer and Automation Systems, University of Damascus, Syria

Endnote

a. metaheuristic is a higher-level procedure or heuristic designed to find, generate, or select a heuristic (partial search algorithm) that may provide a sufficiently good solution to an optimization problem, especially with incomplete or imperfect information or limited computation capacity.

References

1. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
2. Claesen, M., De Moor, B.: Hyperparameter search in machine learning. arXiv preprint arXiv:1502.02127 (2015)
3. Zhang, H., Sindagi, V., Patel, V.M.: Image de-raining using a conditional generative adversarial network. IEEE transactions on circuits and systems for video technology (2019)
4. Shi, H., Dong, J., Wang, W., Qian, Y., Zhang, X.: Ssgan: secure steganography based on generative adversarial networks. In: Pacific Rim Conference on Multimedia, pp. 534–544 (2017). Springer
5. Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2107–2116 (2017)
6. Wu, J., Zhang, C., Xue, T., Freeman, B., Tenenbaum, J.: Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In: Advances in Neural Information Processing Systems, pp. 82–90 (2016)
7. Du, L., Cui, Z., Wang, L., Ma, J.: Structure tuning method on deep convolutional generative adversarial network with nondominated sorting genetic algorithm ii. Concurrency and Computation: Practice and Experience, 5688 (2020)
8. Koshiyama, A., Firoozye, N., Treleaven, P.: Generative adversarial networks for financial trading strategies fine-tuning and combination. Quantitative Finance, 1–17 (2020)
9. Douzas, G., Bacao, F.: Effective data generation for imbalanced learning using conditional generative adversarial networks. Expert Systems with applications **91**, 464–471 (2018)
10. Chen, T., Lucic, M., Houlsby, N., Gelly, S.: On self modulation for generative adversarial networks. arXiv preprint arXiv:1810.01365 (2018)
11. Mitchell, M.: An Introduction to Genetic Algorithms. MIT press, ??? (1998)
12. Kramer, O.: Genetic Algorithm Essentials vol. 679. Springer, ??? (2017)
13. Karpathy, A.: Generative Models. OpenAI (2020). https://openai.com/blog/generative-models/
14. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
15. Gal, Y., Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: International Conference on Machine Learning, pp. 1050–1059 (2016)
16. geneticalgorithm. https://pypi.org/project/geneticalgorithm/
17. Cheng, K., Tahir, R., Eric, L.K., Li, M.: An analysis of generative adversarial networks and variants for image synthesis on mnist dataset. Multimedia Tools and Applications, 1–28 (2020)

Figures

Figure 1 . Simple GAN

Figure 2 . hyper-parameters vs default parameters

Figure 3 . Generator vs Discriminator

Figure 4 . Python version

Figure 5 . Generator design

Figure 6 . Generator details

Figure 7 . Discriminator details

Figure 8 . Dataset to Model stages.

Figure 9 . Results for Discriminator Loss minimization.

Figure 10 . Top results of Discriminator test.

Figure 11 . Results for Generator Loss minimization.

Figure 12 . Top results of Generator test.

Figure 13 . Generator results of final testing.

Table 1 Implementation of GAN with GA

Main code

```
def f(X):
    gan = GAN(X)
    g_loss=gan.train(epochs=10, batch_size=10, sample_interval=100)
    return g_loss
```

```
algorithm_param = 'max_num_iteration': None,
    'population_size':100
    'mutation_probability':0.1,
    'elit_ratio': 0.01,
    'crossover_probability': 0.5,
    'parents_portion': 0.3,
    'crossover_type':'uniform',
    'max_iteration_without_improv':20
```

```
if_name_ == '_main_':
    varbound = np.array([[0.01,0.2]])
    model = ga(function=f, dimension=1, variable_type='real', variable_boundaries=varbound,function_timeout=300)
    model.run()
```

Table 2 Best parameters for minimization Discriminator loss

keep probability	Batch normalization	Neurons
0.812	64	Generator: 256,512,1024 Discriminator: 512,256
Learning Rate: 0.00019948		

Tables

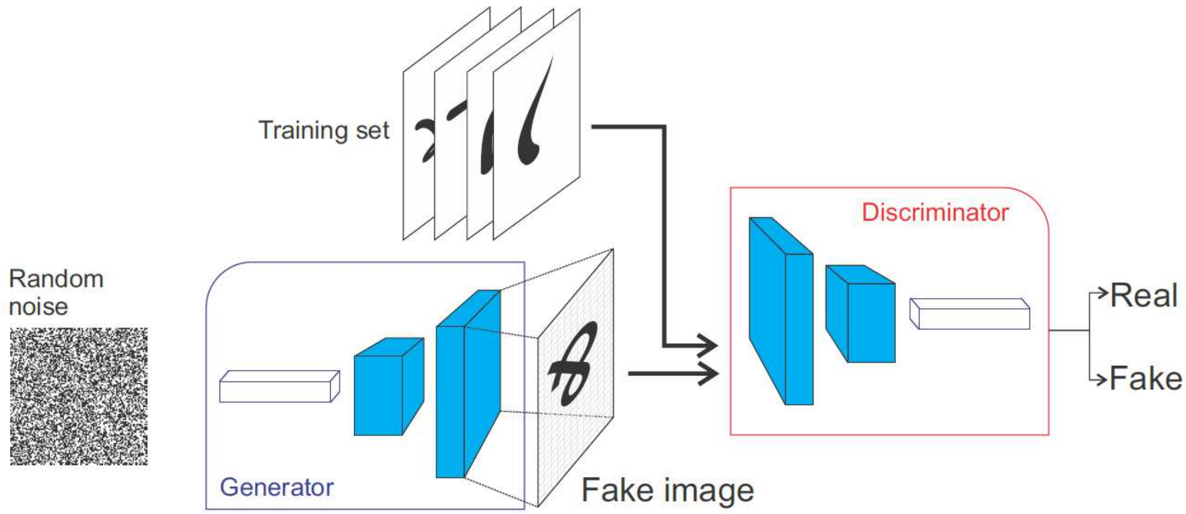
Additional Files

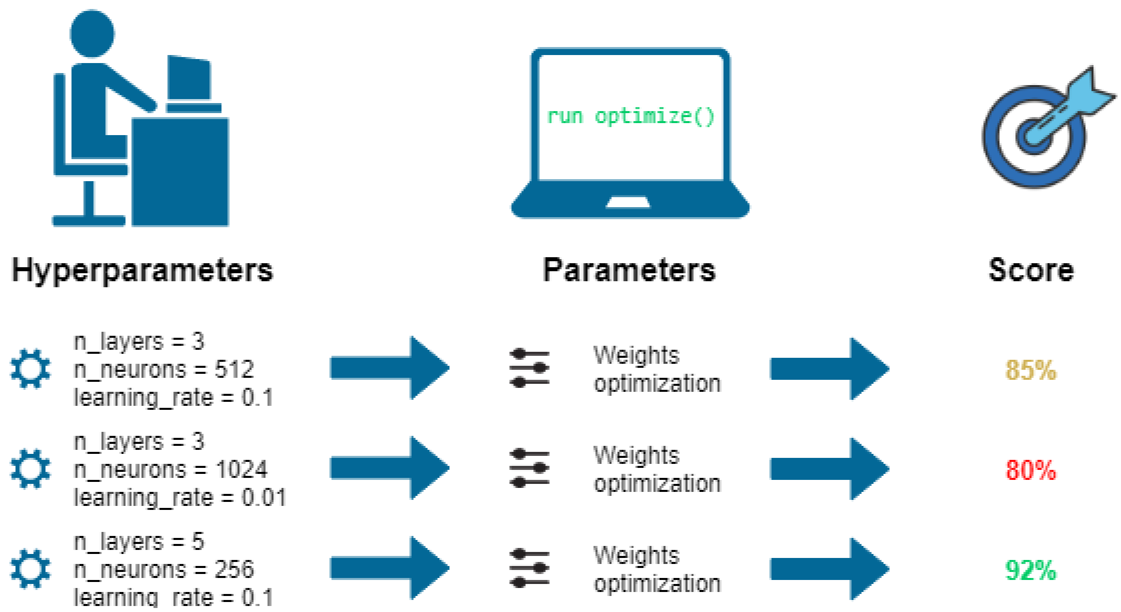
Additional file 1 — Figure 1.docx
Additional file 2 — Figure 2.docx
Additional file 3 — Figure 3.docx
Additional file 4 — Figure 4.docx
Additional file 5 — Figure 5.docx
Additional file 6 — Figure 6.docx
Additional file 7 — Figure 7.docx
Additional file 8 — Figure 8.docx
Additional file 9 — Figure 9.docx
Additional file 10 — Figure 10.docx
Additional file 10 — Figure 11.docx

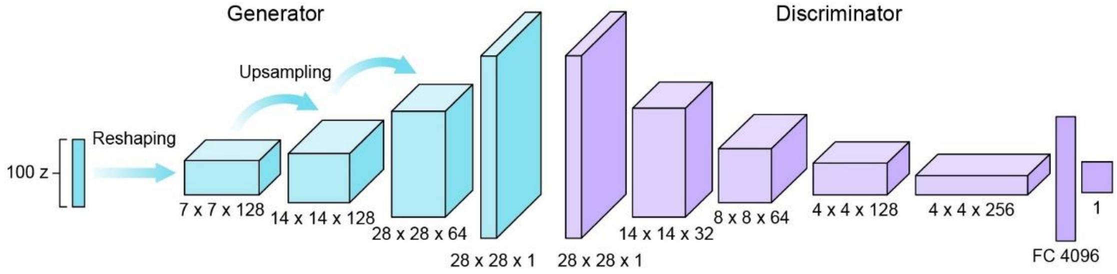
Table 3 Best parameters for minimization Generator loss

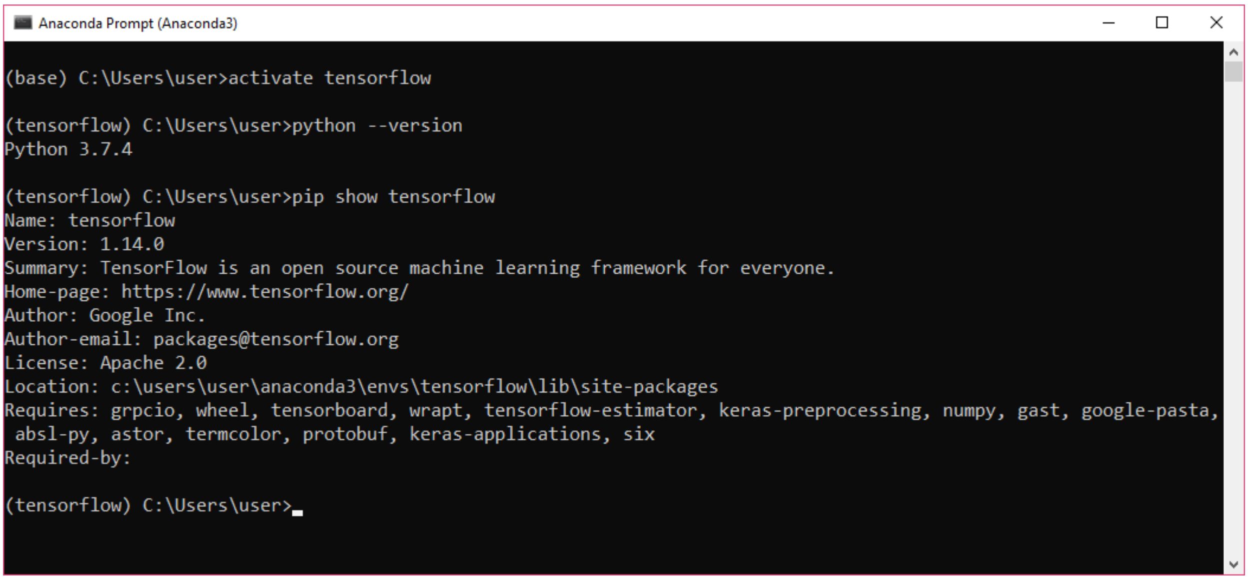
keep probability	Batch normalization	Neurons
0.63	64	Generator: 256,512,1024 Discriminator: 512,256

Learning Rate: 0.171924

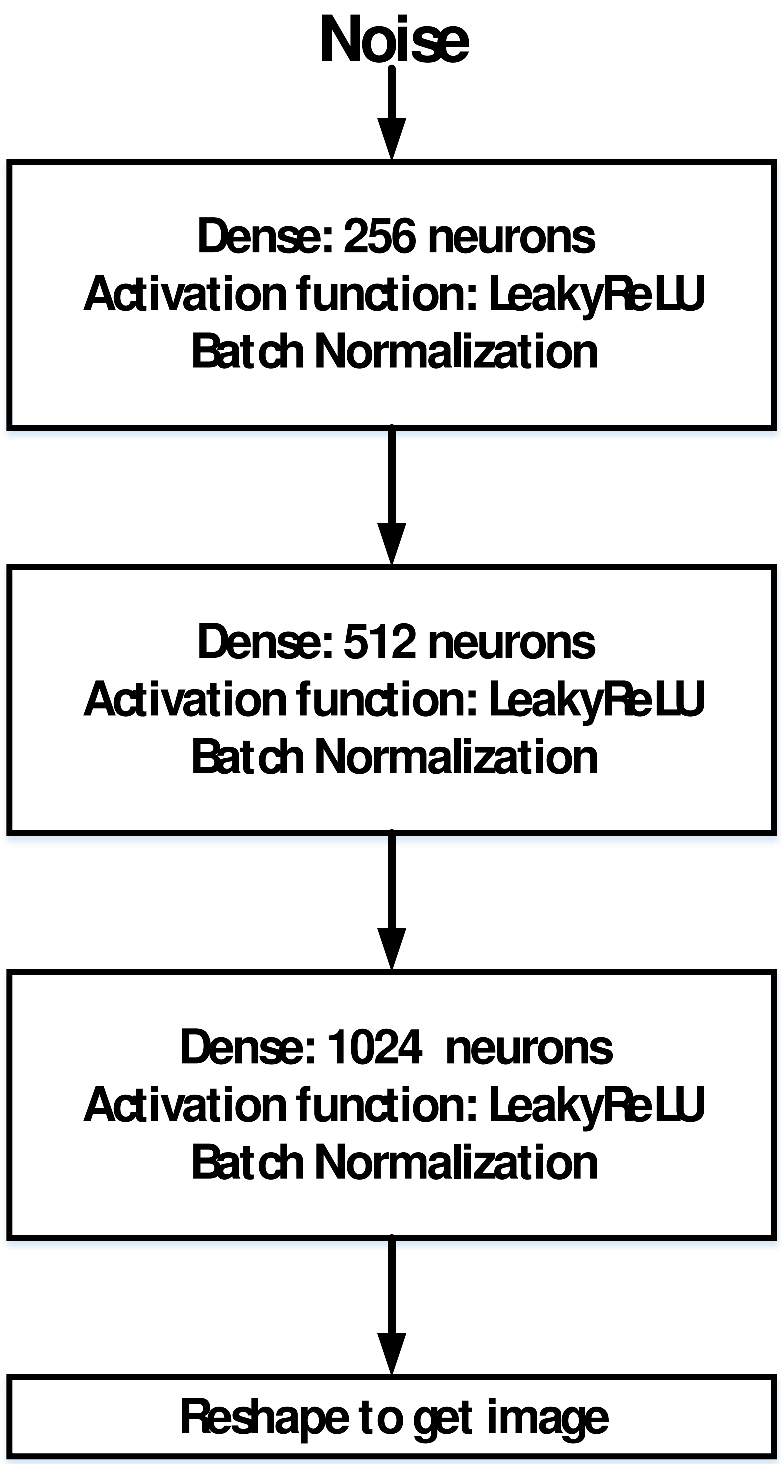

Training set
Random
noise
Generator
Fake image
Discriminator
Real
Fake

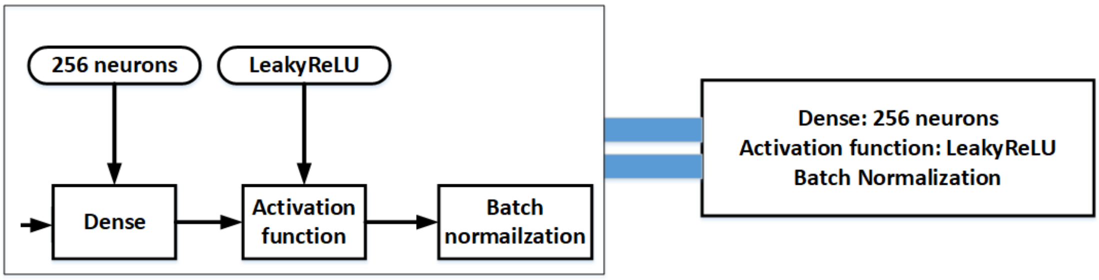

run optimize()
Hyperparameters
Parameters
Score
n_layers = 3
n_neurons = 512
learning_rate = 0.1
Weights optimization
85%
n_layers = 3
n_neurons = 1024
learning_rate = 0.01
Weights optimization
80%
n_layers = 5
n_neurons = 256
learning_rate = 0.1
Weights optimization
92%


Generator
Discriminator
Upsampling
Reshaping
100 z
7 x 7 x 128
14 x 14 x 128
28 x 28 x 64
28 x 28 x 1
28 x 28 x 1
14 x 14 x 32
8 x 8 x 64
4 x 4 x 128
4 x 4 x 256
FC 4096
1



```
Anaconda Prompt (Anaconda3)

(base) C:\Users\user>activate tensorflow

(tensorflow) C:\Users\user>python --version
Python 3.7.4

(tensorflow) C:\Users\user>pip show tensorflow
Name: tensorflow
Version: 1.14.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\users\user\anaconda3\envs\tensorflow\lib\site-packages
Requires: grpcio, wheel, tensorboard, wrapt, tensorflow-estimator, keras-preprocessing, numpy, gast, google-pasta,
 absl-py, astor, termcolor, protobuf, keras-applications, six
Required-by:

(tensorflow) C:\Users\user>
```


Noise
Dense: 256 neurons
Activation function: LeakyReLU
Batch Normalization
Dense: 512 neurons
Activation function: LeakyReLU
Batch Normalization
Dense: 1024 neurons
Activation function: LeakyReLU
Batch Normalization
Reshape to get image


256 neurons
LeakyReLU
Dense
Activation function
Batch normailzation
Dense: 256 neurons
Activation function: LeakyReLU
Batch Normalization

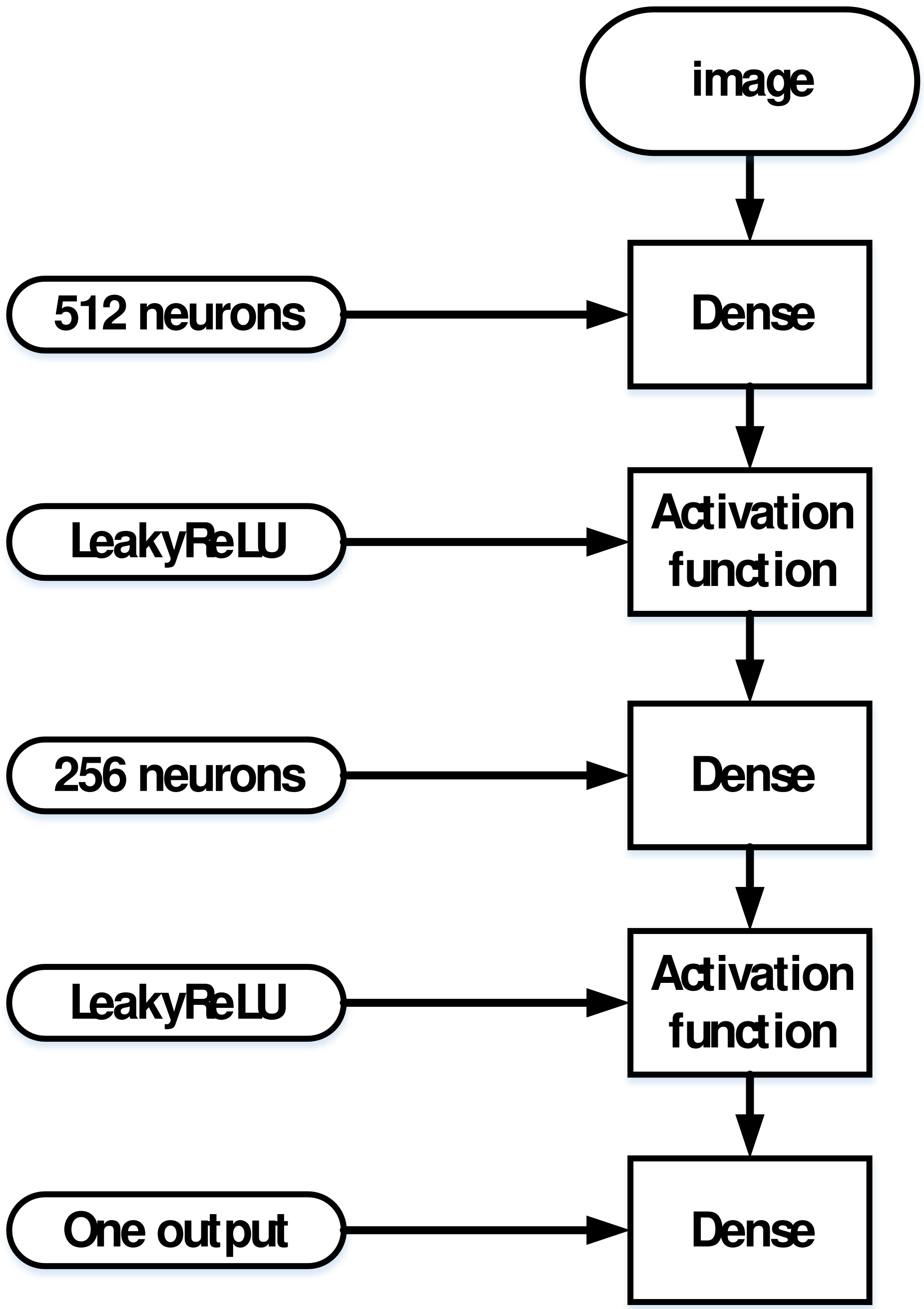


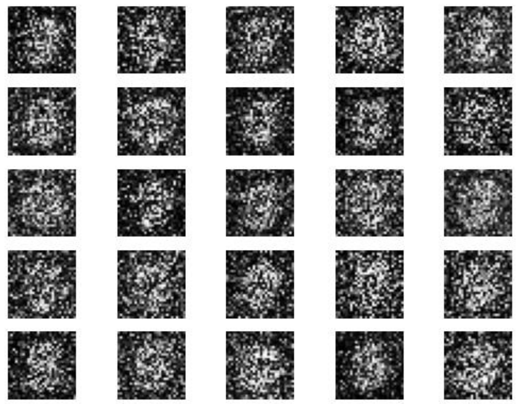
image
512 neurons
Dense
LeakyReLU
Activation function
256 neurons
Dense
LeakyReLU
Activation function
One output
Dense

```
G_Loss,      D_loss,      D_Acc,      LR
4.835429,    8.059048,    0.500000,   0.043611
9.805188,    9.653286,    0.400000,   0.043611
8.059048,    10.450405,   0.350000,   0.043611
6.160692,    13.638883,   0.150000,   0.043611
9.670857,    12.044644,   0.250000,   0.043611
11.282667,   11.247524,   0.300000,   0.043611
9.670857,    11.247524,   0.300000,   0.043611
12.894476,   11.247524,   0.300000,   0.043611
9.670857,    11.247524,   0.300000,   0.043611
11.282667,   11.247524,   0.300000,   0.028553
9.670857,    10.450405,   0.350000,   0.028553
11.282667,   11.247524,   0.300000,   0.028553
11.282667,   11.247524,   0.300000,   0.028553
9.670857,    8.856167,    0.450000,   0.028553
9.670857,    10.450405,   0.350000,   0.028553
11.282667,   10.450405,   0.350000,   0.028553
11.282667,   8.059048,    0.500000,   0.028553
9.670857,    10.450405,   0.350000,   0.028553
8.059048,    10.450405,   0.350000,   0.028553
8.059048,    11.247524,   0.300000,   0.097347
9.670857,    10.450405,   0.350000,   0.097347
6.447238,    10.450405,   0.350000,   0.097347
9.670857,    8.856167,    0.450000,   0.097347
8.059048,    11.247524,   0.300000,   0.097347
6.447238,    10.450405,   0.350000,   0.097347
12.894476,   12.044643,   0.250000,   0.097347
11.282667,   11.247524,   0.300000,   0.097347
12.894476,   10.450405,   0.350000,   0.097347
11.282667,   10.450405,   0.350000,   0.097347
11.282667,   11.247524,   0.300000,   0.065461
8.059048,    9.653286,    0.400000,   0.065461
```

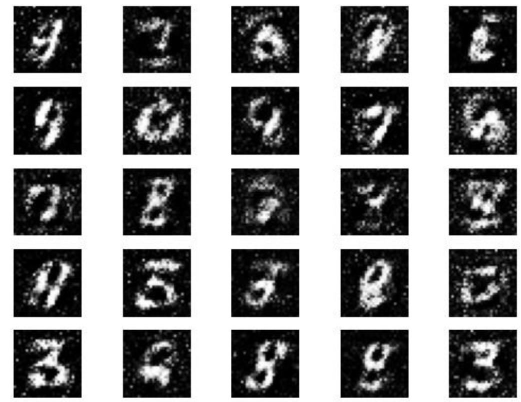
	G_Loss	D_loss	D_Acc	LR
698	11.282667	0.0	1.0	0.065273
720	9.670857	0.0	1.0	0.066043
750	11.282667	0.0	1.0	0.028301
824	11.282667	0.0	1.0	0.090876
874	12.894476	0.0	1.0	0.013635
881	8.059048	0.0	1.0	0.026079
898	11.282667	0.0	1.0	0.020439
899	11.282667	0.0	1.0	0.036774
943	11.282667	0.0	1.0	0.028957
952	12.894476	0.0	1.0	0.044514
971	11.282667	0.0	1.0	0.088288
986	11.282667	0.0	1.0	0.072956
995	11.282667	0.0	1.0	0.067904
1006	12.894476	0.0	1.0	0.019948
1040	9.670857	0.0	1.0	0.034972
1071	11.282667	0.0	1.0	0.013635
1095	9.670857	0.0	1.0	0.067284
1113	11.282667	0.0	1.0	0.034972
1122	11.282667	0.0	1.0	0.067284
1166	9.670857	0.0	1.0	0.068281
1182	11.282667	0.0	1.0	0.034972
1219	11.282667	0.0	1.0	0.034972
1229	8.059048	0.0	1.0	0.090502
1247	11.282667	0.0	1.0	0.085555
1337	11.282667	0.0	1.0	0.064713
1388	11.282667	0.0	1.0	0.032499
1406	11.282667	0.0	1.0	0.090502
1407	9.670857	0.0	1.0	0.090502
1465	12.894476	0.0	1.0	0.052773
1511	11.282667	0.0	1.0	0.085555
...	...	...	...	...
2277	12.894476	0.0	1.0	0.013635
2313	11.282667	0.0	1.0	0.090502
2333	9.670857	0.0	1.0	0.085555
2339	9.670857	0.0	1.0	0.083484
2340	11.282667	0.0	1.0	0.083484
2386	11.282667	0.0	1.0	0.019948
2404	9.670857	0.0	1.0	0.019948
2438	11.282667	0.0	1.0	0.090502
2439	11.282667	0.0	1.0	0.019948
2459	11.282667	0.0	1.0	0.085555
2468	12.894476	0.0	1.0	0.085555
2512	11.282667	0.0	1.0	0.044132

```
epoch,G_Loss,D_loss,D_Acc,LR
000001,   6.989230,    8.059048,    0.500000,    0.043419
000002,   11.282667,   12.044644,   0.250000,    0.043419
000003,   9.670857,    8.059048,    0.500000,    0.043419
000004,   11.282667,   8.856167,    0.450000,    0.043419
000005,   11.282667,   8.059048,    0.500000,    0.043419
000006,   11.282667,   10.450405,   0.350000,    0.043419
000007,   11.282667,   9.653286,    0.400000,    0.043419
000008,   9.670857,    12.044643,   0.250000,    0.043419
000009,   8.059048,    11.247524,   0.300000,    0.043419
000000,   11.282667,   10.450405,   0.350000,    0.144509
000001,   11.282667,   11.247524,   0.300000,    0.144509
000002,   11.282667,   8.856167,    0.450000,    0.144509
000003,   11.282667,   10.450405,   0.350000,    0.144509
000004,   11.282667,   11.247524,   0.300000,    0.144509
000005,   12.894476,   11.247524,   0.300000,    0.144509
000006,   11.282667,   12.044643,   0.250000,    0.144509
000007,   11.282667,   8.856167,    0.450000,    0.144509
000008,   9.670857,    10.450405,   0.350000,    0.144509
000009,   11.282667,   9.653286,    0.400000,    0.144509
000000,   9.670857,    8.059048,    0.500000,    0.084243
000001,   9.670857,    10.450405,   0.350000,    0.084243
000002,   9.670857,    8.059048,    0.500000,    0.084243
000003,   9.670857,    10.450405,   0.350000,    0.084243
000004,   11.282667,   8.856167,    0.450000,    0.084243
000005,   11.282667,   10.450405,   0.350000,    0.084243
000006,   11.282667,   8.059048,    0.500000,    0.084243
000007,   11.282667,   11.247524,   0.300000,    0.084243
000008,   11.282667,   8.856167,    0.450000,    0.084243
000009,   11.282667,   10.450405,   0.350000,    0.084243
000000,   12.894476,   12.044643,   0.250000,    0.151344
000001,   9.670857,    8.059048,    0.500000,    0.151344
```

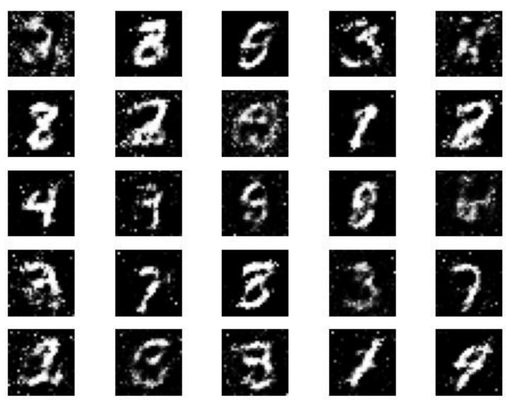
	epoch	G_Loss	D_loss	D_Acc	LR
1333	4	0.0	7.174073	0.55	0.171924
1334	5	0.0	7.971192	0.50	0.171924
1338	9	0.0	7.971192	0.50	0.171924
1339	0	0.0	7.971192	0.50	0.172570
1340	1	0.0	6.376954	0.60	0.172570
1344	5	0.0	6.376954	0.60	0.172570
1346	7	0.0	7.174073	0.55	0.172570
1349	0	0.0	7.174073	0.55	0.027164
1350	1	0.0	7.174073	0.55	0.027164
1351	2	0.0	7.971192	0.50	0.027164
1353	4	0.0	7.971192	0.50	0.027164
1359	0	0.0	7.174073	0.55	0.172846
1360	1	0.0	7.174073	0.55	0.172846
1362	3	0.0	7.174073	0.55	0.172846
1365	6	0.0	7.971192	0.50	0.172846
1366	7	0.0	7.971192	0.50	0.172846
1367	8	0.0	7.971192	0.50	0.172846
1368	9	0.0	7.174073	0.55	0.172846
1371	2	0.0	6.376954	0.60	0.015445
1373	4	0.0	7.971192	0.50	0.015445
1376	7	0.0	7.174073	0.55	0.015445
1378	9	0.0	4.782716	0.70	0.015445
1379	0	0.0	7.971192	0.50	0.015445
1381	2	0.0	7.971192	0.50	0.015445
1382	3	0.0	7.971192	0.50	0.015445
1384	5	0.0	7.971192	0.50	0.015445
1385	6	0.0	7.174073	0.55	0.015445
1386	7	0.0	7.174073	0.55	0.015445
1389	0	0.0	6.376954	0.60	0.171924
1390	1	0.0	7.971192	0.50	0.171924
...	...	...	...	...	...
4545	6	0.0	7.971192	0.50	0.171924
4546	7	0.0	7.174073	0.55	0.171924
4547	8	0.0	7.971192	0.50	0.171924
4549	0	0.0	7.174073	0.55	0.171924
4550	1	0.0	7.971192	0.50	0.171924



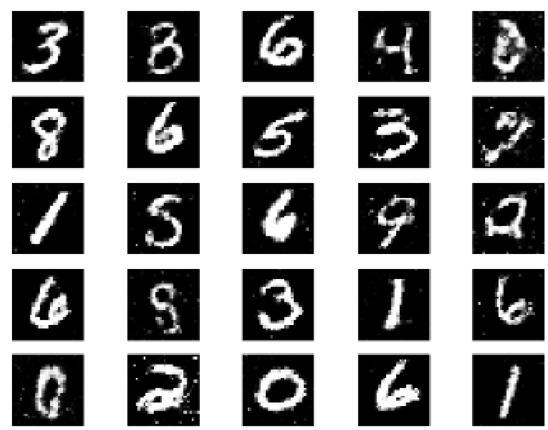
Generator step a



Generator step b



Generator step c



Generator final step

Figures

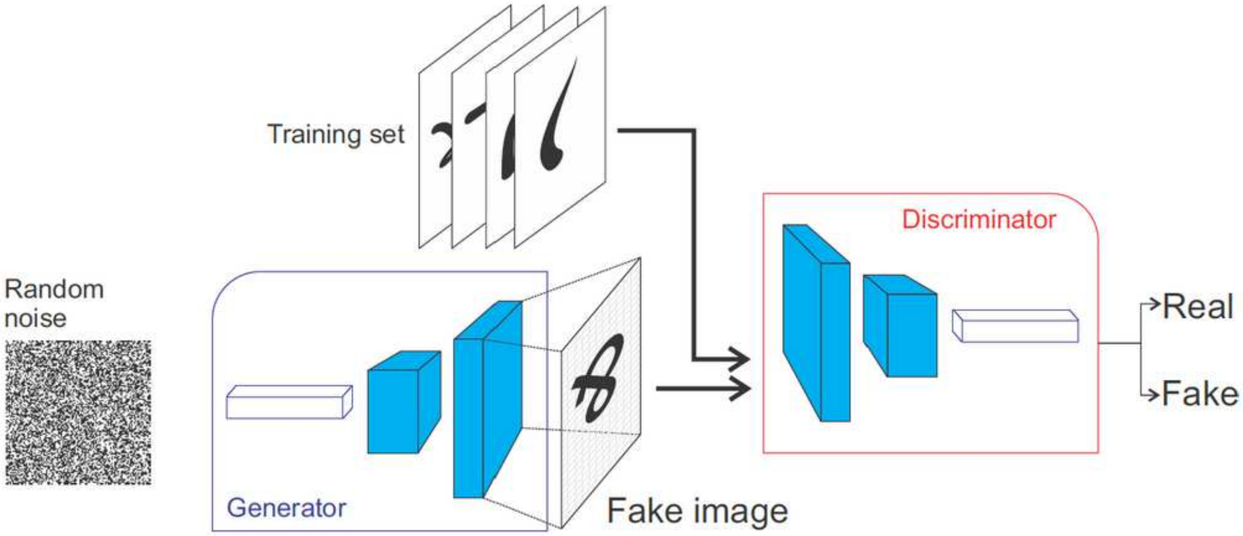


Figure 1

Simple GAN

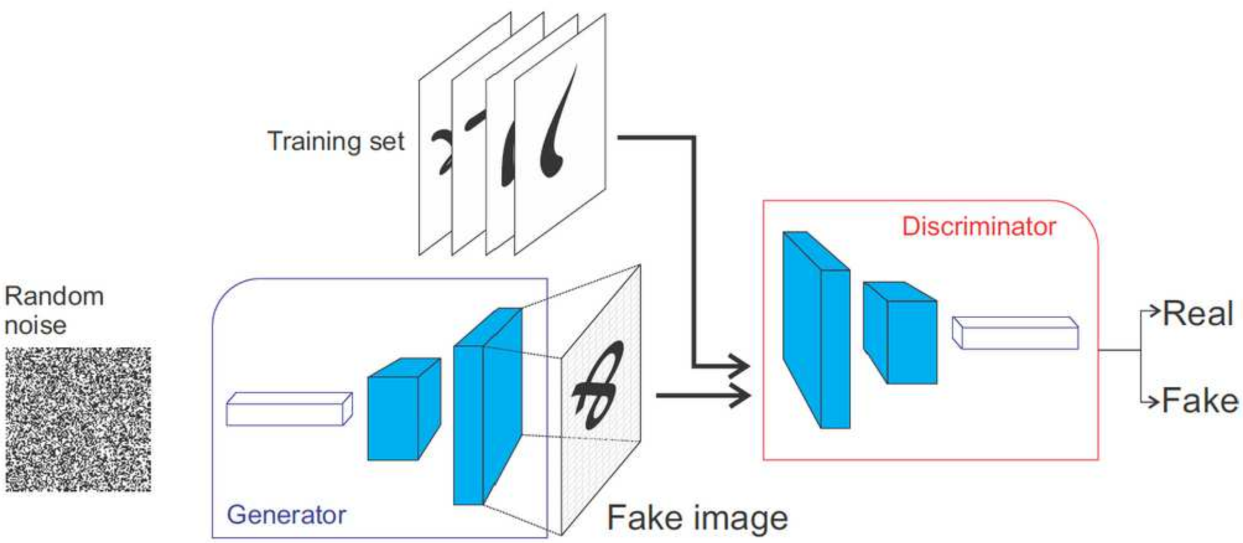


Figure 1

Simple GAN

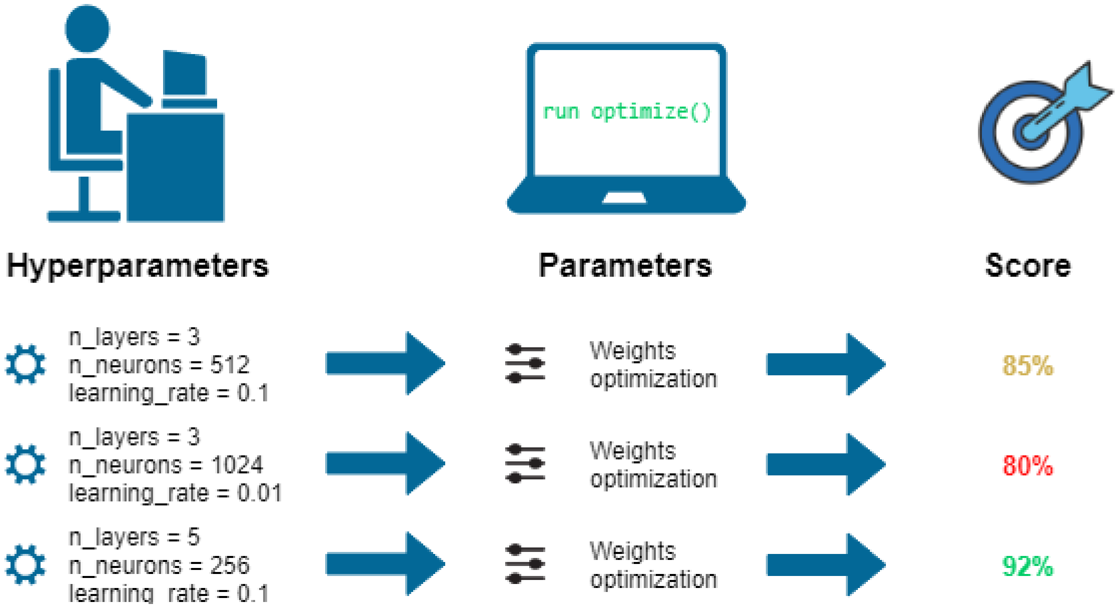


Figure 2

hyper-parameters vs default parameters

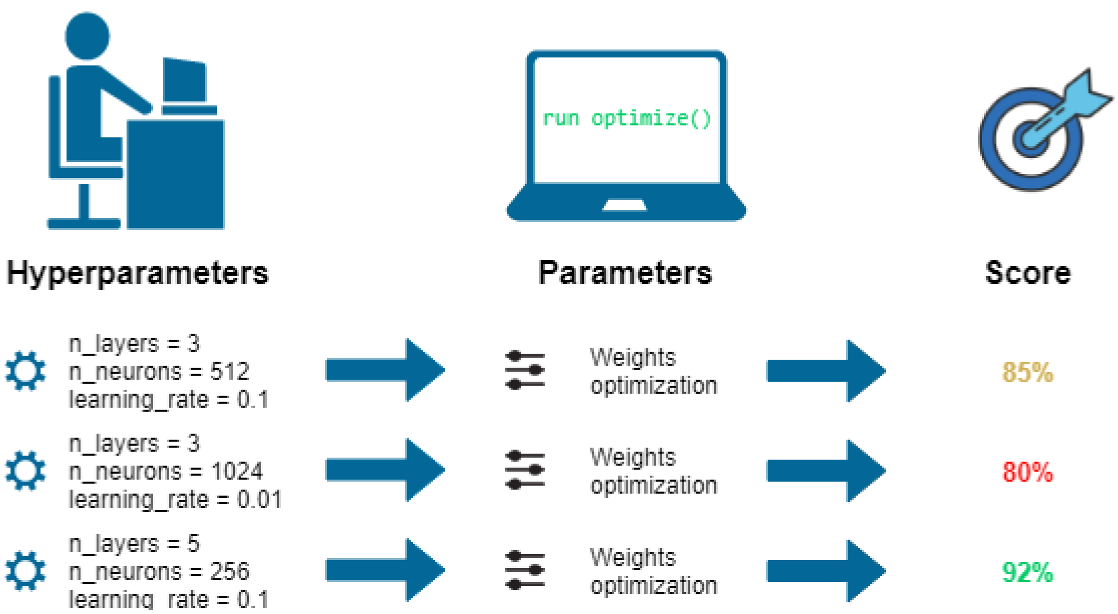


Figure 2

hyper-parameters vs default parameters

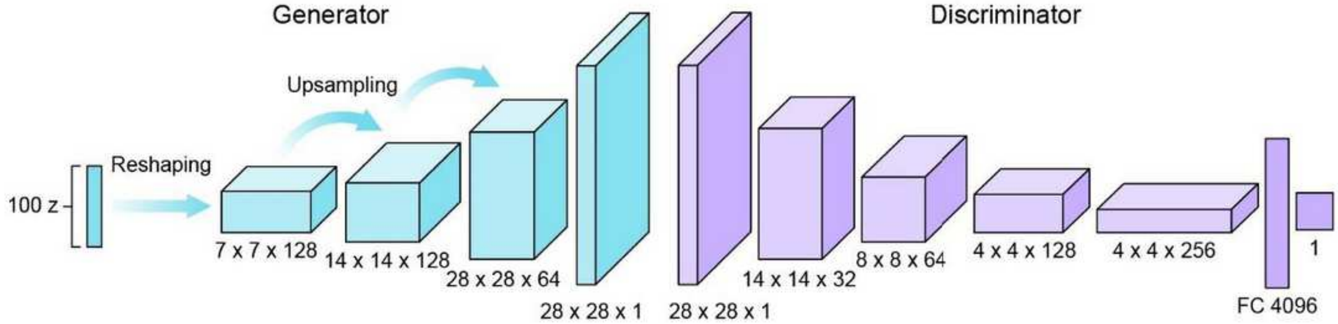


Figure 3

Generator vs Discriminator

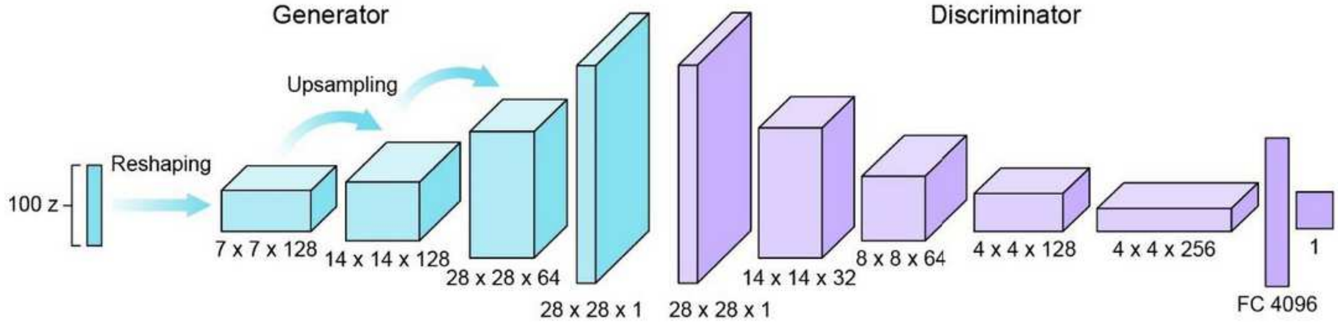
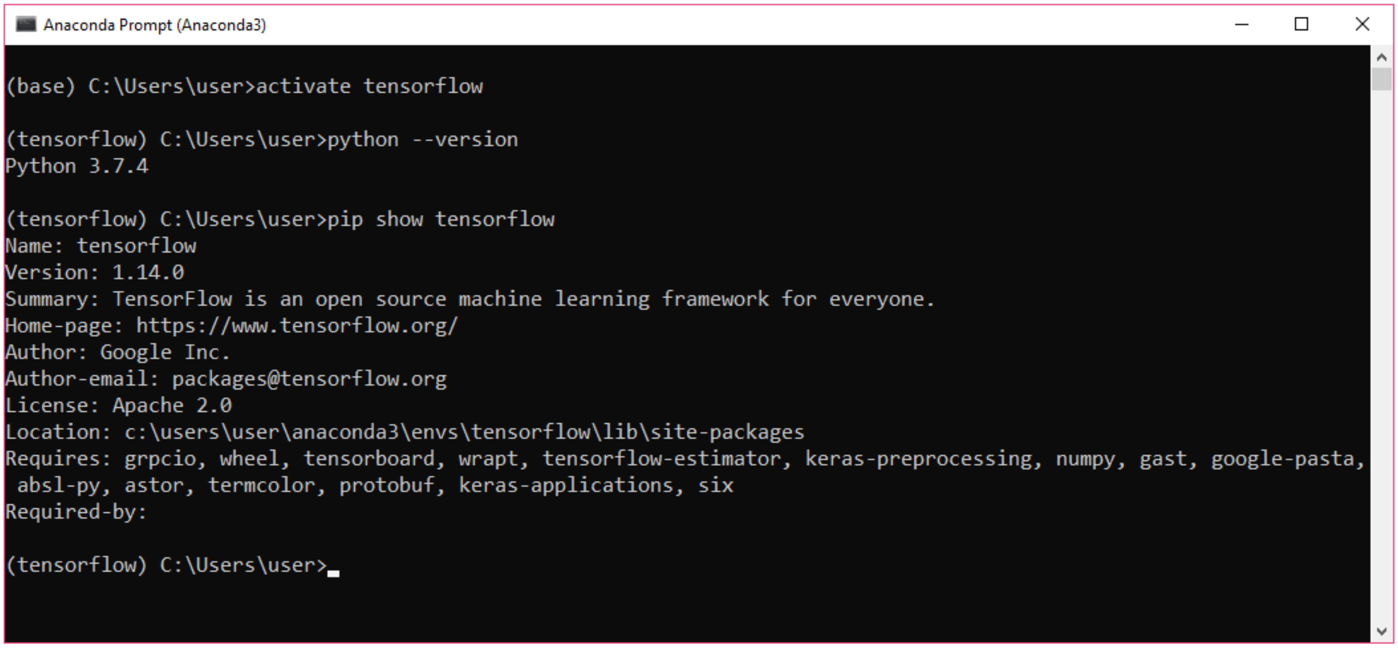


Figure 3

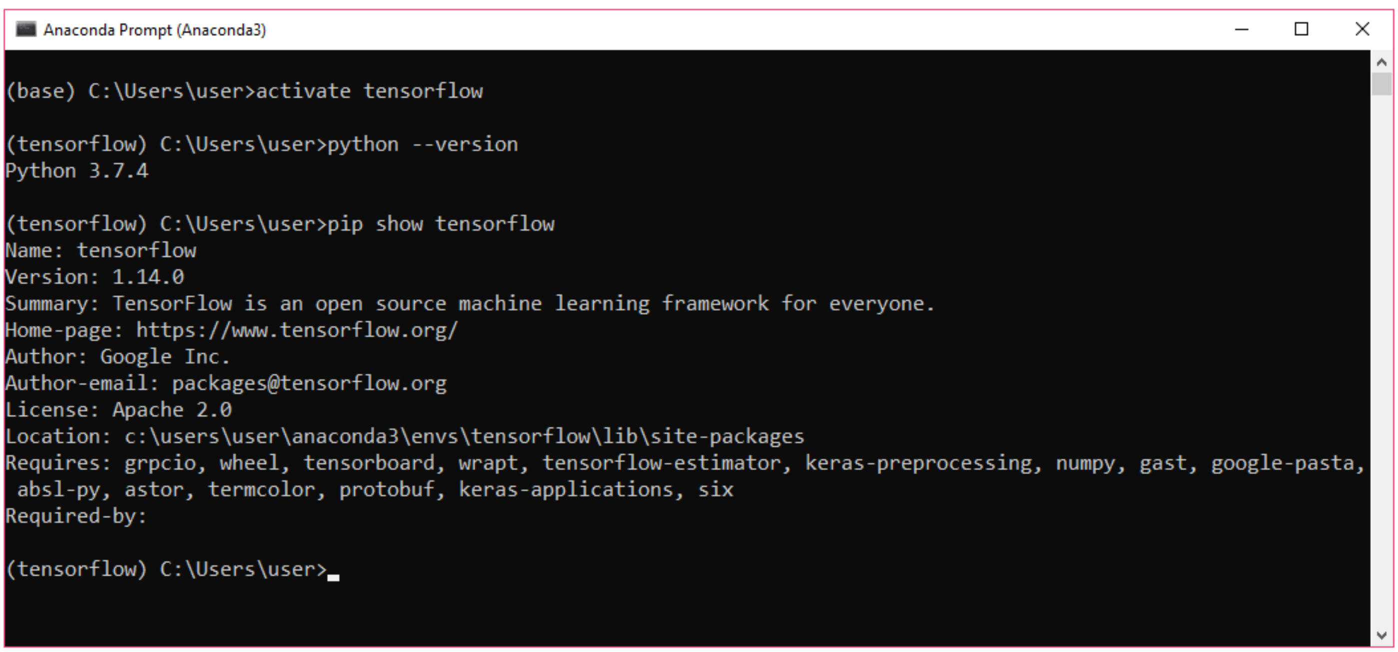
Generator vs Discriminator



```
Anaconda Prompt (Anaconda3)

(base) C:\Users\user>activate tensorflow

(tensorflow) C:\Users\user>python --version
Python 3.7.4

(tensorflow) C:\Users\user>pip show tensorflow
Name: tensorflow
Version: 1.14.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\users\user\anaconda3\envs\tensorflow\lib\site-packages
Requires: grpcio, wheel, tensorboard, wrapt, tensorflow-estimator, keras-preprocessing, numpy, gast, google-pasta,
 absl-py, astor, termcolor, protobuf, keras-applications, six
Required-by:

(tensorflow) C:\Users\user>
```

Figure 4

Python version



```
Anaconda Prompt (Anaconda3)

(base) C:\Users\user>activate tensorflow

(tensorflow) C:\Users\user>python --version
Python 3.7.4

(tensorflow) C:\Users\user>pip show tensorflow
Name: tensorflow
Version: 1.14.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: c:\users\user\anaconda3\envs\tensorflow\lib\site-packages
Requires: grpcio, wheel, tensorboard, wrapt, tensorflow-estimator, keras-preprocessing, numpy, gast, google-pasta,
 absl-py, astor, termcolor, protobuf, keras-applications, six
Required-by:

(tensorflow) C:\Users\user>
```

Figure 4

Python version

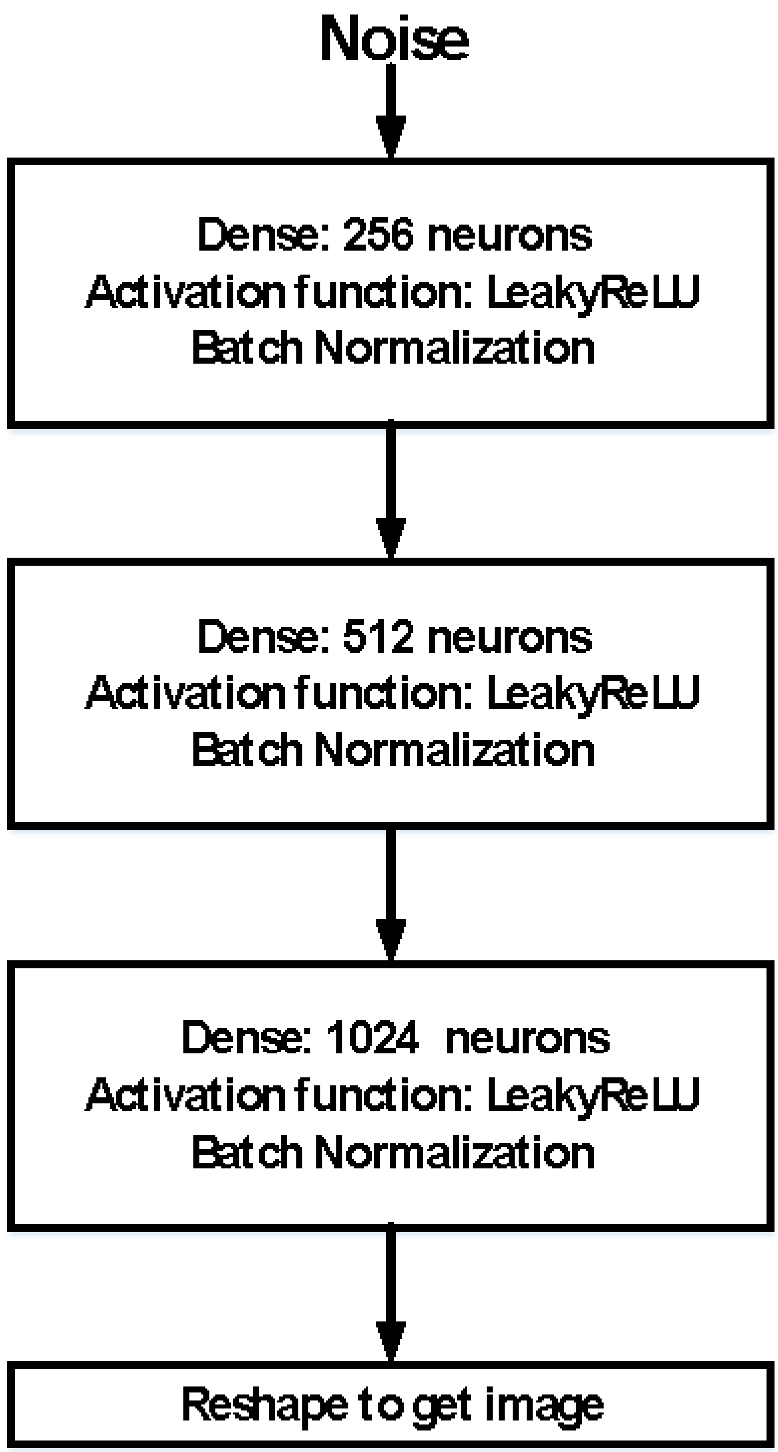


Figure 5

Generator design

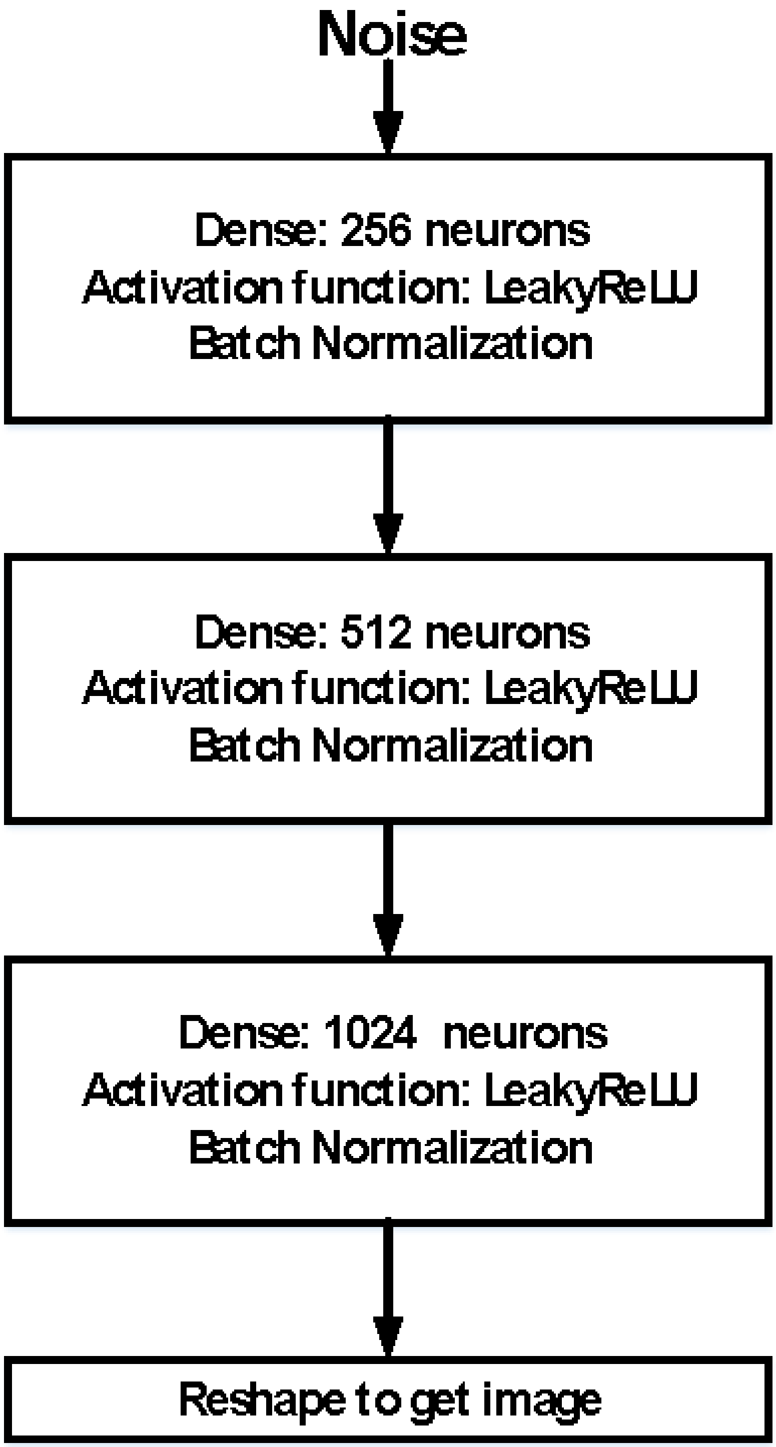


Figure 5

Generator design

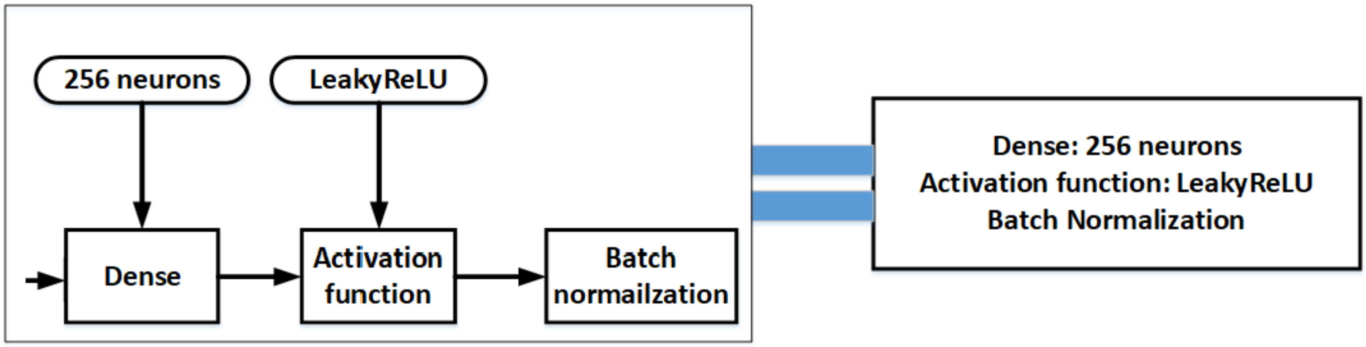


Figure 6

Generator details

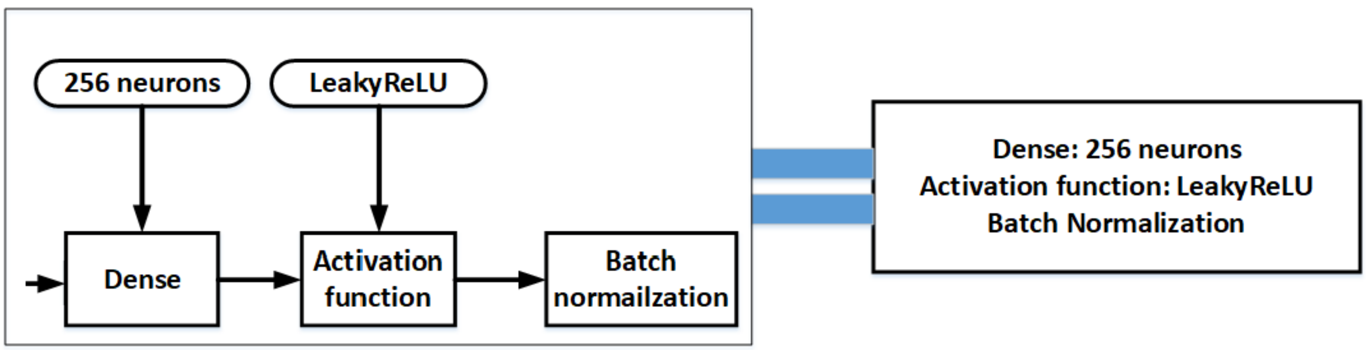


Figure 6

Generator details

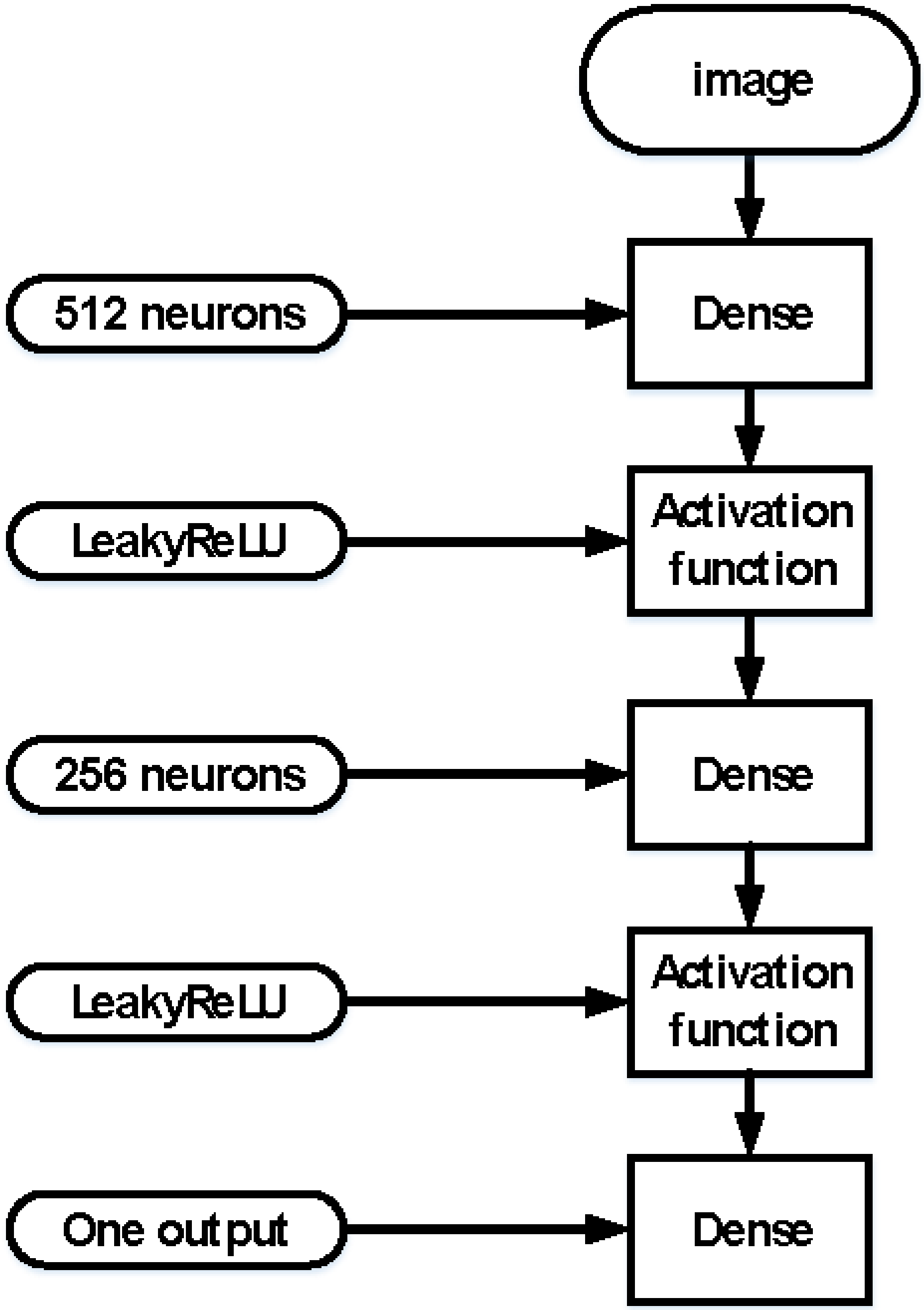


Figure 7

Discriminator details

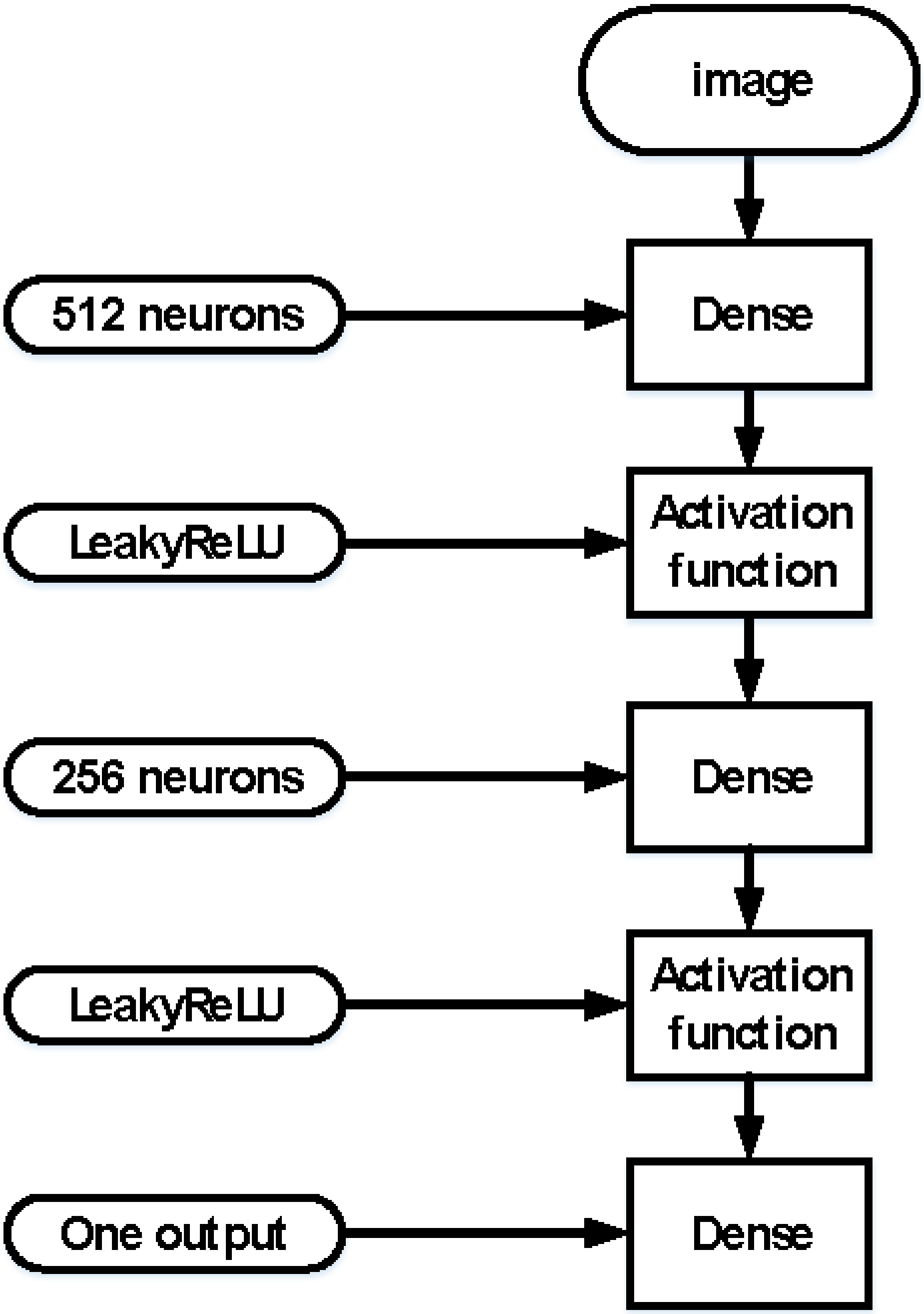


Figure 7

Discriminator details

```
G_Loss,      D_loss,      D_Acc,      LR
4.835429,    8.059048,    0.500000,   0.043611
9.805188,    9.653286,    0.400000,   0.043611
8.059048,    10.450405,   0.350000,   0.043611
6.160692,    13.638883,   0.150000,   0.043611
9.670857,    12.044644,   0.250000,   0.043611
11.282667,   11.247524,   0.300000,   0.043611
9.670857,    11.247524,   0.300000,   0.043611
12.894476,   11.247524,   0.300000,   0.043611
9.670857,    11.247524,   0.300000,   0.043611
11.282667,   11.247524,   0.300000,   0.028553
9.670857,    10.450405,   0.350000,   0.028553
11.282667,   11.247524,   0.300000,   0.028553
11.282667,   11.247524,   0.300000,   0.028553
9.670857,    8.856167,    0.450000,   0.028553
9.670857,    10.450405,   0.350000,   0.028553
11.282667,   10.450405,   0.350000,   0.028553
11.282667,   8.059048,    0.500000,   0.028553
9.670857,    10.450405,   0.350000,   0.028553
8.059048,    10.450405,   0.350000,   0.028553
8.059048,    11.247524,   0.300000,   0.097347
9.670857,    10.450405,   0.350000,   0.097347
6.447238,    10.450405,   0.350000,   0.097347
9.670857,    8.856167,    0.450000,   0.097347
8.059048,    11.247524,   0.300000,   0.097347
6.447238,    10.450405,   0.350000,   0.097347
12.894476,   12.044643,   0.250000,   0.097347
11.282667,   11.247524,   0.300000,   0.097347
12.894476,   10.450405,   0.350000,   0.097347
11.282667,   10.450405,   0.350000,   0.097347
11.282667,   11.247524,   0.300000,   0.065461
8.059048,    9.653286,    0.400000,   0.065461
```

Figure 8

Dataset to Model stages.

```
G_Loss,     D_loss,     D_Acc,     LR
4.835429,   8.059048,   0.500000,  0.043611
9.805188,   9.653286,   0.400000,  0.043611
8.059048,   10.450405,  0.350000,  0.043611
6.160692,   13.638883,  0.150000,  0.043611
9.670857,   12.044644,  0.250000,  0.043611
11.282667,  11.247524,  0.300000,  0.043611
9.670857,   11.247524,  0.300000,  0.043611
12.894476,  11.247524,  0.300000,  0.043611
9.670857,   11.247524,  0.300000,  0.043611
11.282667,  11.247524,  0.300000,  0.028553
9.670857,   10.450405,  0.350000,  0.028553
11.282667,  11.247524,  0.300000,  0.028553
11.282667,  11.247524,  0.300000,  0.028553
9.670857,   8.856167,   0.450000,  0.028553
9.670857,   10.450405,  0.350000,  0.028553
11.282667,  10.450405,  0.350000,  0.028553
11.282667,  8.059048,   0.500000,  0.028553
9.670857,   10.450405,  0.350000,  0.028553
8.059048,   10.450405,  0.350000,  0.028553
8.059048,   11.247524,  0.300000,  0.097347
9.670857,   10.450405,  0.350000,  0.097347
6.447238,   10.450405,  0.350000,  0.097347
9.670857,   8.856167,   0.450000,  0.097347
8.059048,   11.247524,  0.300000,  0.097347
6.447238,   10.450405,  0.350000,  0.097347
12.894476,  12.044643,  0.250000,  0.097347
11.282667,  11.247524,  0.300000,  0.097347
12.894476,  10.450405,  0.350000,  0.097347
11.282667,  10.450405,  0.350000,  0.097347
11.282667,  11.247524,  0.300000,  0.065461
8.059048,   9.653286,   0.400000,  0.065461
```

Figure 8

Dataset to Model stages.

	G_Loss	D_loss	D_Acc	LR
698	11.282667	0.0	1.0	0.065273
720	9.670857	0.0	1.0	0.066043
750	11.282667	0.0	1.0	0.028301
824	11.282667	0.0	1.0	0.090876
874	12.894476	0.0	1.0	0.013635
881	8.059048	0.0	1.0	0.026079
898	11.282667	0.0	1.0	0.020439
899	11.282667	0.0	1.0	0.036774
943	11.282667	0.0	1.0	0.028957
952	12.894476	0.0	1.0	0.044514
971	11.282667	0.0	1.0	0.088288
986	11.282667	0.0	1.0	0.072956
995	11.282667	0.0	1.0	0.067904
1006	12.894476	0.0	1.0	0.019948
1040	9.670857	0.0	1.0	0.034972
1071	11.282667	0.0	1.0	0.013635
1095	9.670857	0.0	1.0	0.067284
1113	11.282667	0.0	1.0	0.034972
1122	11.282667	0.0	1.0	0.067284
1166	9.670857	0.0	1.0	0.068281
1182	11.282667	0.0	1.0	0.034972
1219	11.282667	0.0	1.0	0.034972
1229	8.059048	0.0	1.0	0.090502
1247	11.282667	0.0	1.0	0.085555
1337	11.282667	0.0	1.0	0.064713
1388	11.282667	0.0	1.0	0.032499
1406	11.282667	0.0	1.0	0.090502
1407	9.670857	0.0	1.0	0.090502
1465	12.894476	0.0	1.0	0.052773
1511	11.282667	0.0	1.0	0.085555
...	...	...	...	...
2277	12.894476	0.0	1.0	0.013635
2313	11.282667	0.0	1.0	0.090502
2333	9.670857	0.0	1.0	0.085555
2339	9.670857	0.0	1.0	0.083484
2340	11.282667	0.0	1.0	0.083484
2386	11.282667	0.0	1.0	0.019948
2404	9.670857	0.0	1.0	0.019948
2438	11.282667	0.0	1.0	0.090502
2439	11.282667	0.0	1.0	0.019948
2459	11.282667	0.0	1.0	0.085555
2468	12.894476	0.0	1.0	0.085555
2512	11.282667	0.0	1.0	0.044132

Figure 9

Results for Discriminator Loss minimization.

	G_Loss	D_loss	D_Acc	LR
698	11.282667	0.0	1.0	0.065273
720	9.670857	0.0	1.0	0.066043
750	11.282667	0.0	1.0	0.028301
824	11.282667	0.0	1.0	0.090876
874	12.894476	0.0	1.0	0.013635
881	8.059048	0.0	1.0	0.026079
898	11.282667	0.0	1.0	0.020439
899	11.282667	0.0	1.0	0.036774
943	11.282667	0.0	1.0	0.028957
952	12.894476	0.0	1.0	0.044514
971	11.282667	0.0	1.0	0.088288
986	11.282667	0.0	1.0	0.072956
995	11.282667	0.0	1.0	0.067904
1006	12.894476	0.0	1.0	0.019948
1040	9.670857	0.0	1.0	0.034972
1071	11.282667	0.0	1.0	0.013635
1095	9.670857	0.0	1.0	0.067284
1113	11.282667	0.0	1.0	0.034972
1122	11.282667	0.0	1.0	0.067284
1166	9.670857	0.0	1.0	0.068281
1182	11.282667	0.0	1.0	0.034972
1219	11.282667	0.0	1.0	0.034972
1229	8.059048	0.0	1.0	0.090502
1247	11.282667	0.0	1.0	0.085555
1337	11.282667	0.0	1.0	0.064713
1388	11.282667	0.0	1.0	0.032499
1406	11.282667	0.0	1.0	0.090502
1407	9.670857	0.0	1.0	0.090502
1465	12.894476	0.0	1.0	0.052773
1511	11.282667	0.0	1.0	0.085555
...	...	...	...	...
2277	12.894476	0.0	1.0	0.013635
2313	11.282667	0.0	1.0	0.090502
2333	9.670857	0.0	1.0	0.085555
2339	9.670857	0.0	1.0	0.083484
2340	11.282667	0.0	1.0	0.083484
2386	11.282667	0.0	1.0	0.019948
2404	9.670857	0.0	1.0	0.019948
2438	11.282667	0.0	1.0	0.090502
2439	11.282667	0.0	1.0	0.019948
2459	11.282667	0.0	1.0	0.085555
2468	12.894476	0.0	1.0	0.085555
2512	11.282667	0.0	1.0	0.044132

Figure 9

Results for Discriminator Loss minimization.

```
epoch,G_Loss,D_loss,D_Acc,LR
000001,  6.989230,   8.059048,   0.500000,   0.043419
000002,  11.282667,  12.044644,  0.250000,   0.043419
000003,  9.670857,   8.059048,   0.500000,   0.043419
000004,  11.282667,  8.856167,   0.450000,   0.043419
000005,  11.282667,  8.059048,   0.500000,   0.043419
000006,  11.282667,  10.450405,  0.350000,   0.043419
000007,  11.282667,  9.653286,   0.400000,   0.043419
000008,  9.670857,   12.044643,  0.250000,   0.043419
000009,  8.059048,   11.247524,  0.300000,   0.043419
000000,  11.282667,  10.450405,  0.350000,   0.144509
000001,  11.282667,  11.247524,  0.300000,   0.144509
000002,  11.282667,  8.856167,   0.450000,   0.144509
000003,  11.282667,  10.450405,  0.350000,   0.144509
000004,  11.282667,  11.247524,  0.300000,   0.144509
000005,  12.894476,  11.247524,  0.300000,   0.144509
000006,  11.282667,  12.044643,  0.250000,   0.144509
000007,  11.282667,  8.856167,   0.450000,   0.144509
000008,  9.670857,   10.450405,  0.350000,   0.144509
000009,  11.282667,  9.653286,   0.400000,   0.144509
000000,  9.670857,   8.059048,   0.500000,   0.084243
000001,  9.670857,   10.450405,  0.350000,   0.084243
000002,  9.670857,   8.059048,   0.500000,   0.084243
000003,  9.670857,   10.450405,  0.350000,   0.084243
000004,  11.282667,  8.856167,   0.450000,   0.084243
000005,  11.282667,  10.450405,  0.350000,   0.084243
000006,  11.282667,  8.059048,   0.500000,   0.084243
000007,  11.282667,  11.247524,  0.300000,   0.084243
000008,  11.282667,  8.856167,   0.450000,   0.084243
000009,  11.282667,  10.450405,  0.350000,   0.084243
000000,  12.894476,  12.044643,  0.250000,   0.151344
000001,  9.670857,   8.059048,   0.500000,   0.151344
```

Figure 10

Top results of Discriminator test.

```
epoch,G_Loss,D_loss,D_Acc,LR
000001,  6.989230,   8.059048,   0.500000,   0.043419
000002,  11.282667,  12.044644,  0.250000,   0.043419
000003,  9.670857,   8.059048,   0.500000,   0.043419
000004,  11.282667,  8.856167,   0.450000,   0.043419
000005,  11.282667,  8.059048,   0.500000,   0.043419
000006,  11.282667,  10.450405,  0.350000,   0.043419
000007,  11.282667,  9.653286,   0.400000,   0.043419
000008,  9.670857,   12.044643,  0.250000,   0.043419
000009,  8.059048,   11.247524,  0.300000,   0.043419
000000,  11.282667,  10.450405,  0.350000,   0.144509
000001,  11.282667,  11.247524,  0.300000,   0.144509
000002,  11.282667,  8.856167,   0.450000,   0.144509
000003,  11.282667,  10.450405,  0.350000,   0.144509
000004,  11.282667,  11.247524,  0.300000,   0.144509
000005,  12.894476,  11.247524,  0.300000,   0.144509
000006,  11.282667,  12.044643,  0.250000,   0.144509
000007,  11.282667,  8.856167,   0.450000,   0.144509
000008,  9.670857,   10.450405,  0.350000,   0.144509
000009,  11.282667,  9.653286,   0.400000,   0.144509
000000,  9.670857,   8.059048,   0.500000,   0.084243
000001,  9.670857,   10.450405,  0.350000,   0.084243
000002,  9.670857,   8.059048,   0.500000,   0.084243
000003,  9.670857,   10.450405,  0.350000,   0.084243
000004,  11.282667,  8.856167,   0.450000,   0.084243
000005,  11.282667,  10.450405,  0.350000,   0.084243
000006,  11.282667,  8.059048,   0.500000,   0.084243
000007,  11.282667,  11.247524,  0.300000,   0.084243
000008,  11.282667,  8.856167,   0.450000,   0.084243
000009,  11.282667,  10.450405,  0.350000,   0.084243
000000,  12.894476,  12.044643,  0.250000,   0.151344
000001,  9.670857,   8.059048,   0.500000,   0.151344
```

Figure 10

Top results of Discriminator test.

	epoch	G_Loss	D_loss	D_Acc	LR
1333	4	0.0	7.174073	0.55	0.171924
1334	5	0.0	7.971192	0.50	0.171924
1338	9	0.0	7.971192	0.50	0.171924
1339	0	0.0	7.971192	0.50	0.172570
1340	1	0.0	6.376954	0.60	0.172570
1344	5	0.0	6.376954	0.60	0.172570
1346	7	0.0	7.174073	0.55	0.172570
1349	0	0.0	7.174073	0.55	0.027164
1350	1	0.0	7.174073	0.55	0.027164
1351	2	0.0	7.971192	0.50	0.027164
1353	4	0.0	7.971192	0.50	0.027164
1359	0	0.0	7.174073	0.55	0.172846
1360	1	0.0	7.174073	0.55	0.172846
1362	3	0.0	7.174073	0.55	0.172846
1365	6	0.0	7.971192	0.50	0.172846
1366	7	0.0	7.971192	0.50	0.172846
1367	8	0.0	7.971192	0.50	0.172846
1368	9	0.0	7.174073	0.55	0.172846
1371	2	0.0	6.376954	0.60	0.015445
1373	4	0.0	7.971192	0.50	0.015445
1376	7	0.0	7.174073	0.55	0.015445
1378	9	0.0	4.782716	0.70	0.015445
1379	0	0.0	7.971192	0.50	0.015445
1381	2	0.0	7.971192	0.50	0.015445
1382	3	0.0	7.971192	0.50	0.015445
1384	5	0.0	7.971192	0.50	0.015445
1385	6	0.0	7.174073	0.55	0.015445
1386	7	0.0	7.174073	0.55	0.015445
1389	0	0.0	6.376954	0.60	0.171924
1390	1	0.0	7.971192	0.50	0.171924
...	...	...	...	...	...
4545	6	0.0	7.971192	0.50	0.171924
4546	7	0.0	7.174073	0.55	0.171924
4547	8	0.0	7.971192	0.50	0.171924
4549	0	0.0	7.174073	0.55	0.171924
4550	1	0.0	7.971192	0.50	0.171924

Figure 11

Results for Generator Loss minimization.

	epoch	G_Loss	D_loss	D_Acc	LR
1333	4	0.0	7.174073	0.55	0.171924
1334	5	0.0	7.971192	0.50	0.171924
1338	9	0.0	7.971192	0.50	0.171924
1339	0	0.0	7.971192	0.50	0.172570
1340	1	0.0	6.376954	0.60	0.172570
1344	5	0.0	6.376954	0.60	0.172570
1346	7	0.0	7.174073	0.55	0.172570
1349	0	0.0	7.174073	0.55	0.027164
1350	1	0.0	7.174073	0.55	0.027164
1351	2	0.0	7.971192	0.50	0.027164
1353	4	0.0	7.971192	0.50	0.027164
1359	0	0.0	7.174073	0.55	0.172846
1360	1	0.0	7.174073	0.55	0.172846
1362	3	0.0	7.174073	0.55	0.172846
1365	6	0.0	7.971192	0.50	0.172846
1366	7	0.0	7.971192	0.50	0.172846
1367	8	0.0	7.971192	0.50	0.172846
1368	9	0.0	7.174073	0.55	0.172846
1371	2	0.0	6.376954	0.60	0.015445
1373	4	0.0	7.971192	0.50	0.015445
1376	7	0.0	7.174073	0.55	0.015445
1378	9	0.0	4.782716	0.70	0.015445
1379	0	0.0	7.971192	0.50	0.015445
1381	2	0.0	7.971192	0.50	0.015445
1382	3	0.0	7.971192	0.50	0.015445
1384	5	0.0	7.971192	0.50	0.015445
1385	6	0.0	7.174073	0.55	0.015445
1386	7	0.0	7.174073	0.55	0.015445
1389	0	0.0	6.376954	0.60	0.171924
1390	1	0.0	7.971192	0.50	0.171924
...	...	...	...	...	...
4545	6	0.0	7.971192	0.50	0.171924
4546	7	0.0	7.174073	0.55	0.171924
4547	8	0.0	7.971192	0.50	0.171924
4549	0	0.0	7.174073	0.55	0.171924
4550	1	0.0	7.971192	0.50	0.171924

Figure 11

Results for Generator Loss minimization.

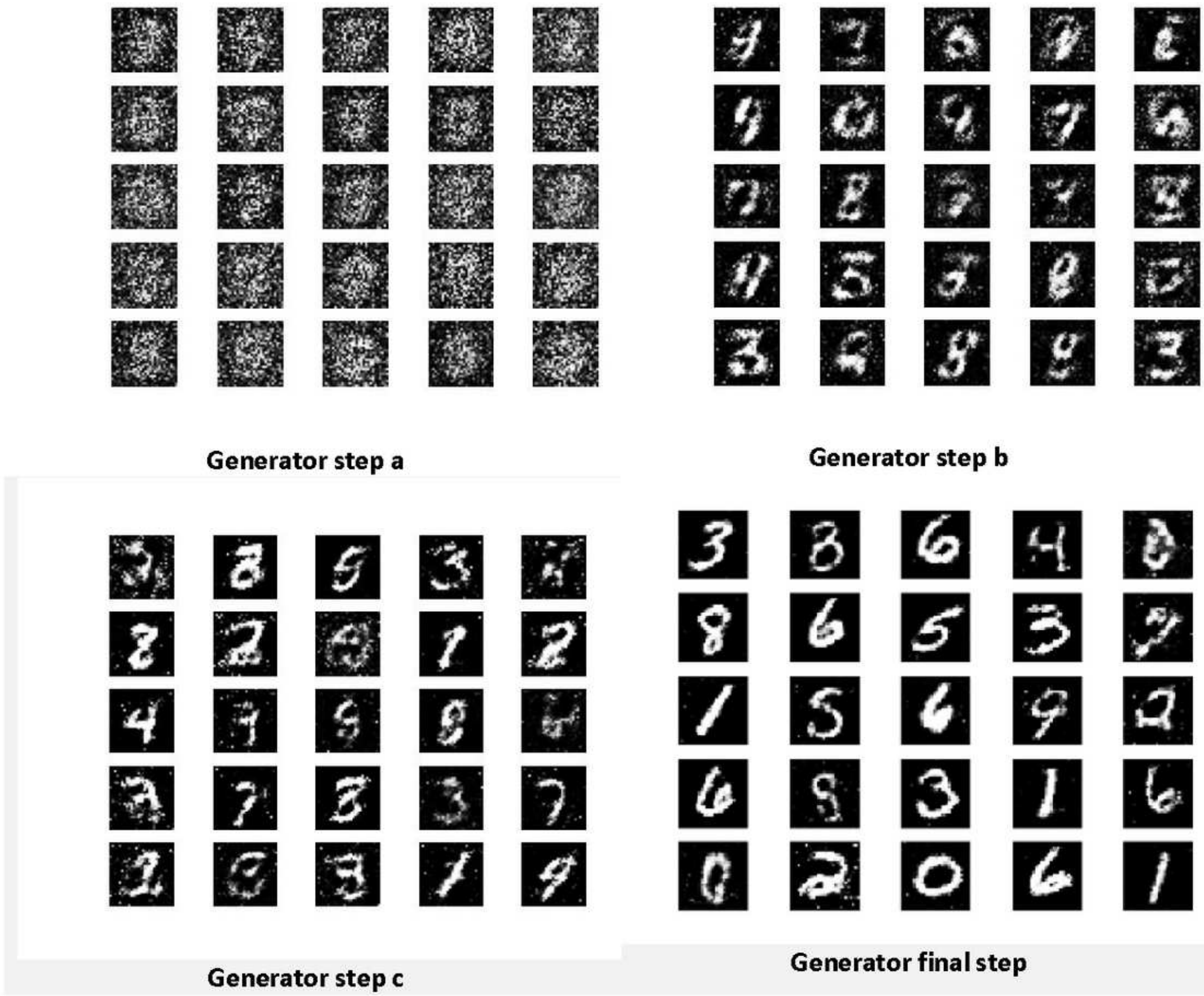


Figure 12

Top results of Generator test.

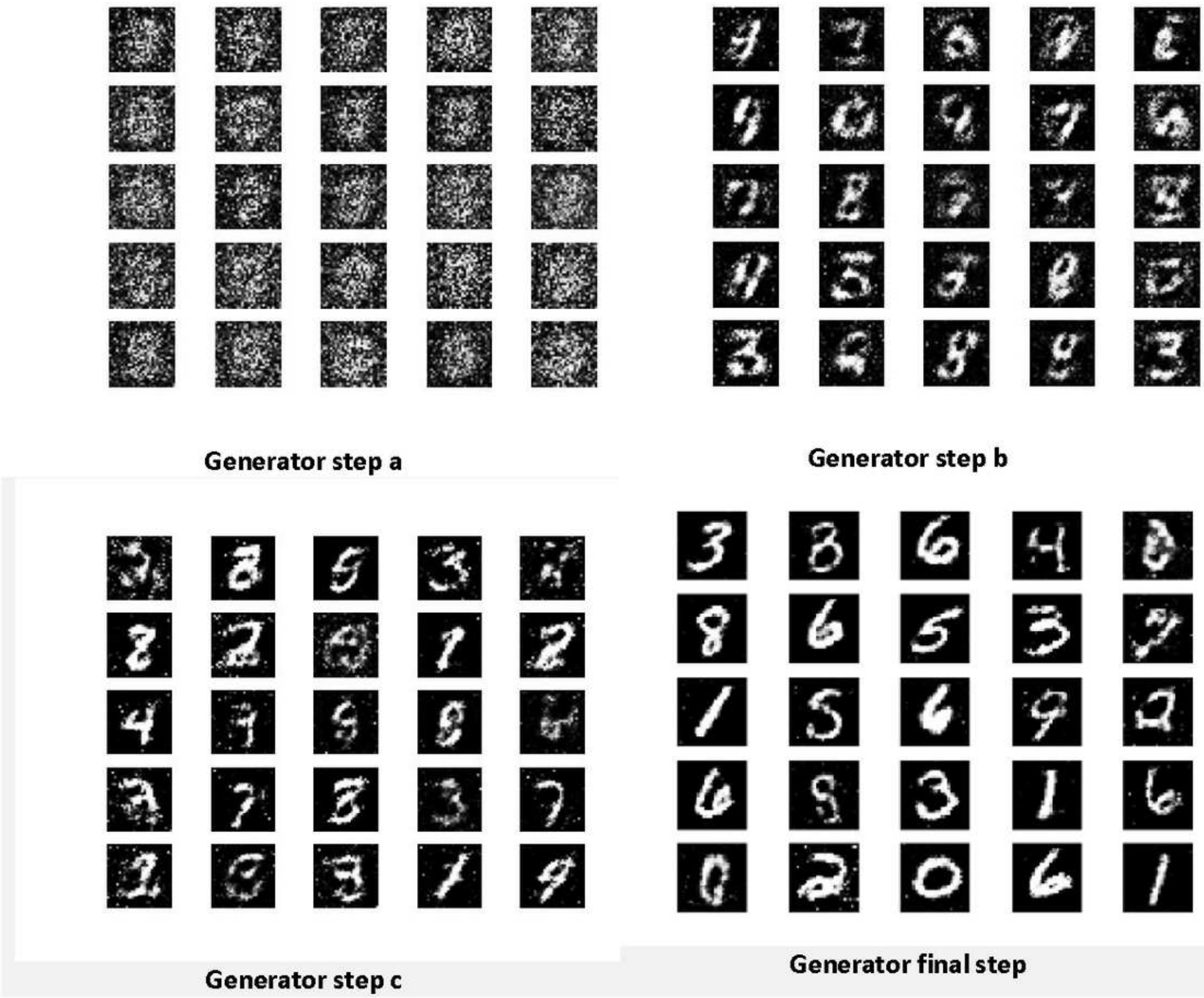


Figure 12

Top results of Generator test.

Image not available with this version

Figure 13

Generator results of final testing.

Image not available with this version

Figure 13

Generator results of final testing.