OPTIMIZATION THE TRAINING ALGORITHMS OF MACHINE LEARNING USING GAN NETWORKS

SEDAT AKEL

JANUARY 2022

ÇANKAYA UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER'S THESIS IN

COMPUTER ENGINEERING

OPTIMIZATION THE TRAINING ALGORITHMS OF MACHINE LEARNING USING GAN NETWORKS

SEDAT AKEL

JANUARY 2022

ABSTRACT

OPTIMIZING THE TRAINING ALGORITHMS OF MACHINE LEARNING USING GENERATIVE ADVERSARIAL NETWORKS

AKEL, Sedat

Computer Engineering Master

Supervisor: Assist. Prof. Dr. Roya CHOUPANI

January 2022, 68 pages

Artificial intelligence has started to be a part of our lives in many aspects over the past few decades. Developing new products without features using artificial intelligence is not reasonable in the contemporary world. It would not be possible if we were not using the deep learning techniques in machine learning algorithms. Traditional machine learning needs clever human design code that transforms raw data into input features for machine learning algorithms. But with deep learning, learning features from raw data directly are possible and this eases the requirement for subject-matter expertise. GANs are very recent advancement in the field of deep learning. They were not presented before 2014. Their capacity and quality of generating are far better than the other generative techniques in machine learning. Their philosophy is based on self-criticizing techniques for automatically learning representation of features. GANs can be used for generating photorealistic images, colorization, turning a simple sketch into a photorealistic image, increasing the resolution of an image, replacing photo defects with realistic patterns, predicting the next frames in a video, data augmentation, generating text, audio etc. data and more. GANs' architecture is very original in deep learning. They are made up of two neural networks that compete during training. Their structures are very clever and interesting but that leads us to very difficult training sessions. GANs are known as difficult to train, prone to failure and very difficult to hyper-tune.

In this thesis we focused on the optimization of some of the GANs. Their philosophies are the key reason to difficulties. For this we first explain the potential difficulties of GANs' trainings. After we retrain some known GANs and compare the results. We propose some structural designs and some optimization parameters to achieve better performant GANs.

Keywords: Generative Adversarial Networks (GANs), GANs' Optimization, Generative Models, Machine Learning, Neural Networks, Deep Learning

ÖZ

ÇEKİŞMELİ ÜRETİCİ AĞLAR İÇİN MAKİNE ÖĞRENMESİ EĞİTİM ALGORİTMALARINDA OPTİMİZASYON

AKEL, Sedat
Bilgisayar Mühendisliği Yüksek Lisans
Danışman:, Dr. Öğretim Üyesi Roya CHOUPANI
Ocak 2022, 68 sayfa

Yapay zekâ, son birkaç on yılda hayatımızda çok farklı yönleriyle yer bulmaya başladı. Bazı özelliklerinde yapay zekanın yer almadığı yeni bir ürün, günümüz dünyasında pek yer edinemez durumdadır. Bu, makine öğrenmesi algoritmalarında derin öğrenme tekniklerinin kullanılması ile mümkün olmuştur. Geleneksel makine öğrenmesi, ham veriyi algoritmalarda kullanılabilecek özelliklere çevirebilmek için insan akılının yer aldığı tasarım ve kodlamalara ihtiyaç duymaktadır. Fakat derin öğrenme ile doğrudan ham veriyi kullanarak özellikleri öğrenmek mümkündür. Bu da makine öğrenmesi sırasında alan uzmanı ihtiyacını oldukça azaltmaktadır. GANlar derin öğrenme alanında oldukça yeni bir ilerleme alanıdır. GANlar 2014'ten önce yoklardı. Onların makine üretmesi alanındaki kapasiteleri ve üretimdeki kaliteleri diğer üretici makine öğrenmesi tekniklerinden çok daha iyi durumdadır. Felsefeleri, verideki özellikleri tanımlamayı otomatik olarak öğrenen ve kendi kendini eleştirerek bunu yapan bir mantığa dayanmaktadır. GANlar, fotoğraf kalitesinde resimler üretmek, siyah-beyaz resimleri renklendirmek, basit bir çizimi gerçekçi bir resime dönüştürmek, resimlerin çözünürlüğünü artırmak, resimlerdeki hatalı-eksik yerleri onarmak, videolarda sonraki kareyi tahmin etmek, makine öğrenmesinde kullanmak üzere veri üretmek, geçekçi yazılar üretmek, gerçekçi müzik ve sesler üretmek için kullanılabilmektedir. GANların mimarisi derin öğrenme teknikleri arasında oldukça

orijinaldir. Temel olarak birbiriyle yarışan iki sinir ağından oluşmaktadır. Yapıları oldukça zeki tasarlanmış ve oldukça ilginçtir. Fakat bu durum makine öğrenmesini oldukça zorlu ve kırılgan yapmaktadır. GANlar oldukça zor öğretilen, hataya açık ve optimizasyonu oldukça zor olarak tanınmaktadır.

Bu tezde GANların optimizasyonuna odaklandık. GANların felsefeleri zorluklarda anahtar konumdadır. Bu sebeple öncelikle GANların eğitimindeki potansiyel zorlukları açıkladık. Daha sonra iyi olarak bilinen bazı GAN mimarilerini, bazı veri setleri ile eğitim sonuçlarını karşılaştırdık. Son olarak bazı temel yapısal öneriler ve optimizasyon parametreleri önerdik.

Anahtar Kelimeler: Çekişmeli Üretici Ağlar, GANların Optimizasyonu, Üretici Modelle, Makine Öğrenmesi, Yapay Sinir Ağları, Derin Öğrenme

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my supervisor Assist. Prof. Dr. Roya Choupani who always guided me with her knowledge and experience and gave me lots of advice. This thesis would not be possible without her criticism, encouragement, and insight throughout the research. My special gratitude also goes to the rest of the thesis committee for the encouragement and insightful comments.

I also would also like to express my special thanks to my wife for her support, and motivation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

GAN	Generative Adversarial Network
ML	Machine Learning
AI	Artificial Intelligence
DCGAN	Deep Convolutional GAN
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
WGAN	Wasserstein GAN
LSGAN	Least Square GAN
VAE	Variational Autoencoder
FID	Frechet Inception Distance
ReLU	Rectified Liner Activation Unit
LeakyReLU	Leaky Version of Rectified Liner Activation Unit
Adam	Adaptive Moment Estimation
SGD	Stochastic Gradient Descent

CHAPTER I

INTRODUCTION

Algorithms for machine learning are very good at classification and regression type of problems. Because they are designed to recognize patterns in existing data. But if the problem comes to generating new data the whole picture is changed. There are numerous techniques in machine learning to generate data, but they are very limited in creativity. This is where the GANs come to the scene.

GANs are a type of generative training architecture in deep learning. Ian Goodfellow et al. proposed GANs in a study published in 2014 [1]. Since then, they got very important attention within the field of AI, and their realistic generative capacity made them a far most interesting research area.

Generative Adversarial Networks (GANs) are a type of generative machine learning model. GANs are capable of both precise reproduction and approximate prediction. They can implicitly learn high-dimensional distributions of any data especially image and audio data which are difficult to model. At the start of the GANs' training a very basic latent distribution is given, but at the end of the training this turns to a very complicated and plausible data outputs like realistic images. They use two competing neural networks to find the real distribution of the data. The generator is used to generate fabricated data. On the other hand, the discriminator is used to distinguish generated data from actual samples.

Training GANs is known as very difficult process, because the generator and the discriminator interact each other and very sensible to the other's dynamics. The discriminator and the generator in a GAN architecture constantly try to outwit one other. This competition based on a zero-sum game. As training advances, the game may end up in a state that game theorists call a Nash equilibrium, named after the mathematician John Nash: this is when no player would be better off changing their

own strategy, assuming the other players do not change theirs. They have many issues to think about and decide to achieve a good result in generation paradigm. To generate more plausible instances its discriminator must also be as good as its generator. GANs have a very simple theoretical philosophy. But finding a balanced training strategy for the both neural networks is very difficult. In order to train both the generator and the discriminator at the same time, there must be healthy rivalry between them.

There are many issues to think about and decide to achieve a good result in generation paradigm. To generate more plausible instances GAN's discriminator must also be as good as its generator.

The biggest difficulty in GANs is mode collapse situation. According to Chen, H. [2], mode collapse is one of the primary causes of GAN training instability. Mode refers to an output distribution. When this happened, the generator starts to produce very similar outputs. This type of situation is not a good practice because lack of diversity is a barrier in front of advancement. Lack of advancement is a barrier in front of good quality outputs.

Another common failure in GAN training is non-convergence. In a normal neural network convergence failing means that the model loss doesn't go to a lower state. But in GANs' trainings it means very different. We have to find a balance between the discriminator and the generator. When this situation happens in the training, it may appear to be fine at first, but later begin oscillating or diverging (non-convergence) [2], and even worst the reason may not be obvious.

GANs are also very sensitive to the hyper-parameters. The discriminator and the generator have to learn at similar pace. If one of them learns more quicker than the other, this part of the GAN start to learn nothing. This is called diminished gradient.

There have been implemented many techniques to improve the quality and effectiveness of the GANs. Each of them focused some part of the problem, and none of them have achieved the ultimate success. This challenge of getting better results for GANs seems to remain quite a while.

1.1 SCOPE OF THE THESIS

After the introduction of original GAN in 2014, it has got a good attention and have proven that they are useful generative models. Since then, many research papers

proposed alternative GAN models to address the difficulties and limitations of the original GAN. Bhatnagar, S. et. al. in their paper [3] show the progress in GANs in human face generation. The results are very impressive. The quality has improved. But this is only limited the small images.

In just three years, advances in GANs made possible to produce high quality portrait photographs. But there are far more advances to be made to get adequate generative products. In this thesis we aimed to improve some GANs architectures with the optimization of training algorithms to get better results.

We used some types of GANs in this thesis as a base of GANs structures for optimization and comparison. One of them is DCGAN [4]. Wasserstein GAN (WGAN) [5] which proposes to use a different loss function like Least Squares GAN (LSGAN) [6] are two other GAN structures we tried as bases in our experiments.

One of the early successful implementations of GANs is DCGAN [4] which uses ConvNets (CNN) in its implementation. This approach then has become the de facto standard in GANs algorithms. Without the need to modify the underlying GAN architecture they use CNN to scale up to the full GAN framework.

As we said numerous times, GANs are known as very difficult in model training. Mode collapse is very common defect in their structures if they are not built wisely. When this happens, similar instances are produced by the generator and, it's wrongly assumed that the loss function is optimized. Substituting the GAN loss function with the Wasserstein GAN (WGAN)'s proposed loss function [5] is to address this problem. But another problem is the quality of the generated images. With the Least Squares GAN (LSGAN) [6], not only the mode collapse problem solved but also the perceptive quality is also improved.

This study proposes some optimization to DCGAN, WSGAN and LSGAN architectures. Hyperparameters which are subject to be changed are discussed and some advises are given for constructing the generator's and discriminator's network structures.

1.2 THESIS ORGANIZATION

Chapter 2 contains fundamental information about GANs to make clear the root causes of the potential problems. It also includes other background information

about optimization of GANs and a literature review. The structures of examined GANs are explained in this chapter.

In Chapter 3, we propose some optimization architectures and regularization tunings for DCGAN, WSGAN and LSGAN architectures.

In Chapter 4 we explain experiments and discuss the results of the proposed GANs' architectures. To try the architectures, we also introduce data sets used in the experiments. All information about experiments' environment, and experiments' structures are given. Comparison with original architectures and effects of optimizations are also explained.

Conclusions and future studies can be found in Chapter 5. Finally, all experiments which are conducted during this thesis are summarized with their GAN structures, example outputs during training sessions and loss function's plots can be found in appendices section.

CHAPTER II

BACKGROUND

2.1 INTRODUCTION

GANs are a clever way of training a generative model, instead of using unsupervised techniques like other generative models, they both use supervised and unsupervised techniques together. In their structures, there are two sub models namely the generator and the discriminator. These sub models are seemed to some kind of ordinary neural networks. But their objectives conflict with each other. The discriminator is a classifier neural network which aims to distinguish the generated samples from real (from training sets) samples. On the other hand, the generator aims to produce convincing samples from starting a really nonsense noise vector. Training sessions are held together until the discriminator is not sure about the classification results.

GANs have many practical application areas like:

- Generating artificial human face images
- Generating a cartoon character
- Image to image translation
- Human pose estimation
- Generating 3D objects from 2D images
- Photograph inpainting
- Photograph editing
- Text to image translation

Before we go into detail, we make some explanation to make GANs definition more understandable. What is generative modeling, what we mean by zero-sum game and how predictive and descriptive machine learning techniques are related in the GANs architecture.

2.2 GENERATIVE MODELS

A generative model is a machine learning model capable of generating random instances using a training set. The output is typically similar to it. Generative modelling involves automatically discovering and learning patterns in the input set using unsupervised techniques. After learning phase, the generative model can produce new plausible instances. GANs are models within the deep generative models which are aimed to describe the problem area with probability and statistics.

Naive Bayes, which is more commonly employed as a discriminative model, is an example of a generative model. To make a prediction, input and output data are generalized with the probability distribution. This classification can be reversed to generate independent features which means it is a generative function [7].

There are other examples of generative modelling like Latent Dirichlet Allocation (LDA), and The Restricted Boltzmann Machine (RBM). It is not in this thesis context that we only mention their names.

Generative modelling can be categorized in various way, but we decided that it can be as Variational Autoencoder (VAE) [8], Autoregressive Networks [9], and Generative Adversarial Networks (GANs).

The VAE is a good generative model, but generated samples are very dependent of the input samples and novelty is very limited. As Wolternik, J.M. et.al. stated their 2019 paper [10], generated samples of VAE are more blurred compared to GANs' outputs.

There are some autoregressive networks, for example Pixel RNN [11]. In this suggested architecture, pixel prediction is made. The generation is satisfactory but the prediction period is very long and comparing with the GAN models they are very low which means for high resolution problem it is nearly impossible to get good results.

The main objective of these paper is GANs and its optimization. So, the other generative models are only mentioned briefly and we study in detailed on GANs architecture.

2.3 PREDICTIVE vs. DESCRIPTIVE

There is a very common classification for machine learning types as predictive, descriptive and reinforcement learning. We are interested in first two types in the

concept of GANs. So, reinforcement learning is out of our scope at the moment. The predictive type of learning known as supervised learning which tries to learn from given labelled input and output pairs [12]. While the other type of learning which is descriptive, can be also called unsupervised learning. This type of learning aims to discovery the knowledge with the examination of the patterns in the input data [12].

The main difference between these two machine learning types is whether there are labels in the training data or not. A depict can be seen in Figure 2.1.

According to Kotsiantis [13], dataset used in a machine learning algorithm can have known labels or not, accordingly this algorithm is called supervised or not.

The predictive machine learning type can classify the given inputs or make regression type of predictions. On the other hand, the descriptive type learning learns from patterns in the input samples. GANS can be classified as the descriptive type of learning but using labelled data in the middle of the training sessions. According to Hofmann [14], descriptive type of ML can help to create labels before using in a supervised learning task.

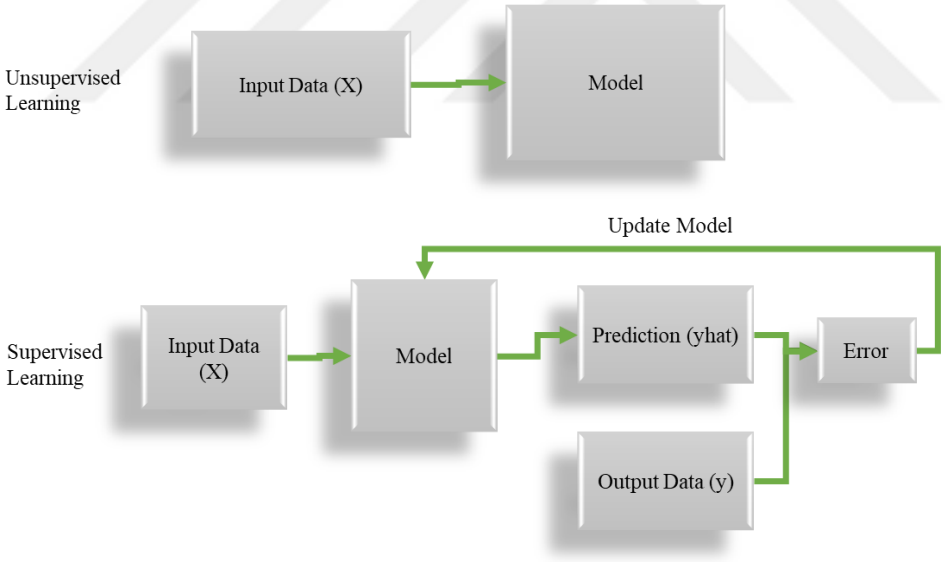


Figure 2.1: Unsupervised vs. supervised learning

Generative models are an unsupervised problem which involves the summarization of the distribution of input variables. On the other hand, predictive modelling is to predict classes with the training of labelled sample set. As it is

illustrated in Figure 2.2, differences between discriminative and generative modelling can be seen.

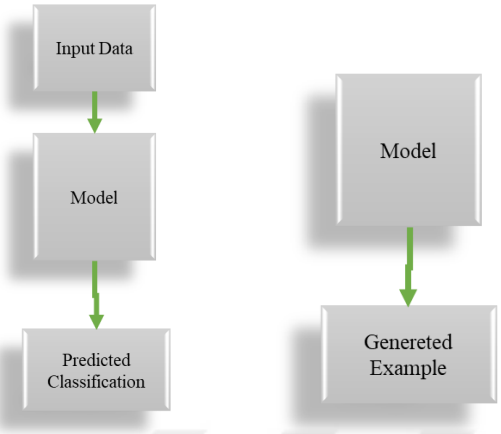


Figure 2.2: Discriminative and generative modeling

As in his book Bishop, C. M. [15] stated that, generative models are approaches that explicitly or implicitly represent the distribution of inputs and outputs, allowing synthetic data points to be generated in the input space by sampling from them.

It is not to be surprise that generative model should produce new instances which is indistinguishable from problem domain. GANs is a good model approach to achieve this goal.

In GAN algorithms supervised and unsupervised learning paradigm are used totally different. The discriminator is trained with fully supervised techniques with labeled data in the training set. The generator models are originally unsupervised learning models, but supervised techniques are also used in the GAN architecture. It can be thought as a decoder in an autoencoder, but is training approach is very different.

Briefly, the generator receives a random noise vector and produce a fake image. The discriminator model is fed with generated images in addition to training set. And this samples are labelled as fake or real to be trained using supervised learning to successfully discriminate fakes from reals. As a subsequent process, the generator learns how to convince the discriminator by minimizing its cost. During this process the discriminator's parameters are frozen and the back propagation tunes only the generator's parameter.

2.4 ZERO-SUM GAME

According to merriam-webster dictionary [16] zero-sum game is, in a situation only one side can win while at the same time the other side loses. Zero sum game is named after the game theory. If one person wins and the other person loses the total net gain is zero. In the GANs structures, it can be thought as when the generator generates very good samples which cannot be discriminated from real samples then its model parameters cannot be changed at all but the discriminator's model is penalized very hugely. In the same philosophy, if the discriminator successfully differentiates genuine samples from fakes, then it is rewarded, means that no model parameter changes are needed whereas the generator's model parameters will be penalized hugely. This zero-sum game penalization will eventually (and hopefully) reach to a point that generated samples by the generator network cannot be differentiated from real samples by the discriminator network.

2.5 GAN MODEL ARCHITECTURE

In this section, we are going to give more detailed information about GANs' structures. GANSs are composed of two trained models that uses a competitive dynamic between them. These models are actually neural networks having opposite objectives. The generator network aims to generate authentic samples with a given domain as training set, but not use these training samples directly. The discriminator aims to distinguish the generated samples and training samples from each other as a classic classification mission.

The generated data (instances) will vary depending on the choice of training set. For example, if we want to produce new human faces than we train the GAN with the samples of real human faces.

While the generator tries to generate authentic instances, the discriminator tries to classify them as real or not. They are trained one-by-one; not at the same time. The generator's input to the neural network is a random noise, On the other hand the discriminator's input to its neural network is samples half from training set half from previously generated samples. This process is not a one-time process. The two

networks try to outsmart each other until they excel in their jobs. The Generator learns through the feedback it receives from the Discriminator's classifications.

There are many varieties of GANs. Depending on the complexity of the domain or design of the GAN implementation, two neural networks can be as simple as densely connected networks, or they can be complex neural network which are used to solve other complex machine learning problems. As Saxena, D. et. al. explained in their paper [17] a sample GAN structure can be seen in Figure 2.3.

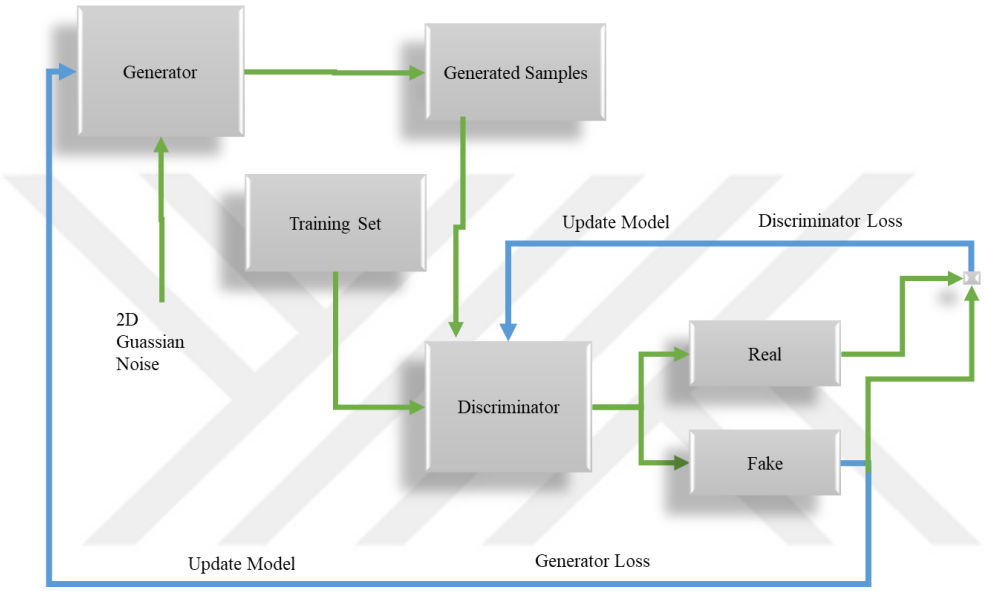


Figure 2.3: Basic GANs Architecture

The generator receives a random noise vector (typically a 2D Gaussian) as an input and produces a fake image as an output. During the training the generator is never fed with real samples. It receives these random noises as the latent space of the desired output domain. The latent space is the characteristics of the desired output domain. But it cannot be observed or discovered directly. It is supposed to be get with the help of deep learning techniques. At the beginning of the training iterations, generated images seem very far from the desired images, but it will gradually learn to produce more realistic images. Similarly, the discriminator has no trouble at learning to classify real ones from fake ones as the generated samples are far from to be so called real.

The discriminator and the generator are trained in turn. First the discriminator is trained. In this phase only its parameters are updated. The classification is ranged

between 0.0 to 1.0 in confidence. After this training session, now, it is the generator's turn to be trained. Actually, the generator is not trained alone, but the help of the discriminator. The produced images are fed into the discriminator as if they were real. At this stage the real samples are not included to the training. And the discriminator's parameters are not changed in this stage, that are frozen. The generated imaged in this stage are fakes but we want the discriminator believe that they are real. The backpropagation is applied to the generator only.

The generator never actually fed with any real images, but it can produce convincing images by only getting the gradients from the discriminator during the second phase of the GAN training. So, the better discriminator means that the better generator.

The basic algorithm for a GAN as follows:

Training of the discriminator

{Generate images with random noise vector as the latent space

Train the discriminator with real and generated images with equal number

Classify these images with the range of 0.0 – 1.0

Compute the classification errors

(With the aim of minimizing classification error)

Backpropagate it to only the discriminator}

Training of the generator

{Generate images with random noise vector as the latent space

Label all of them as real (to fool the discriminator)

Classify them with the help of the discriminator

Compute the classification errors

(With the aim of maximizing the discriminator's errors)

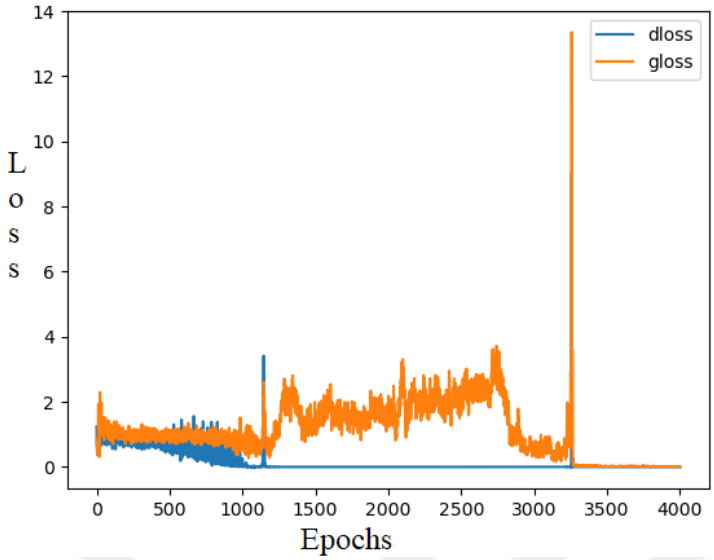
Backpropagate it to only the generator}

These two steps are repeatedly continued until the equilibrium is believed to be reached.

2.6 DIFFICULTIES OF GANs

As the last sentenced from the previous paragraph is stated, training will continue until the belief of the reach of the equilibrium. So, when the training should be stopped is a difficult question. The GANs' philosophy is like in the game theory as zero-sum game. If none of the sub networks are getting better by changing their situation then the equilibrium is reached. In a GAN, if produced samples and real samples have equal probability of (50%) realness, then the training reached its goal. This is theoretical situation. In real life the GANs are not perfect enough to reach to this state. And the GANs' difficulties start here.

One of the biggest difficulties is mode collapse. This is a situation when the trainings are continued, the generated samples are started to be similar even with the different latent spaces. This can be partial collapse and it can be recovered or totally collapsed. The easiest way to understand a mode collapse is to examine the generated samples during the training iterations. It can also be discovered by examining the model loss graphic. While discriminator's loss converges, the generator's is oscillating. When this happens, the produced samples are really garbage and the discriminator is very good at the classification, its loss value goes to zero. Because it is very easy to classify the realness. The experiment number 2 of our experiments resulted in such a situation. The results can be seen in Figure 2.4.



Losses over epochs (above) and generated images (below)

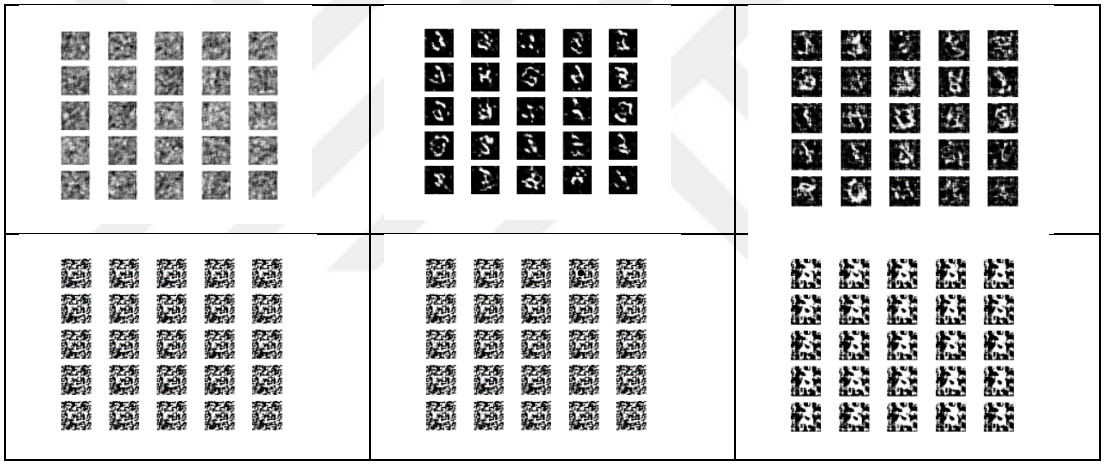
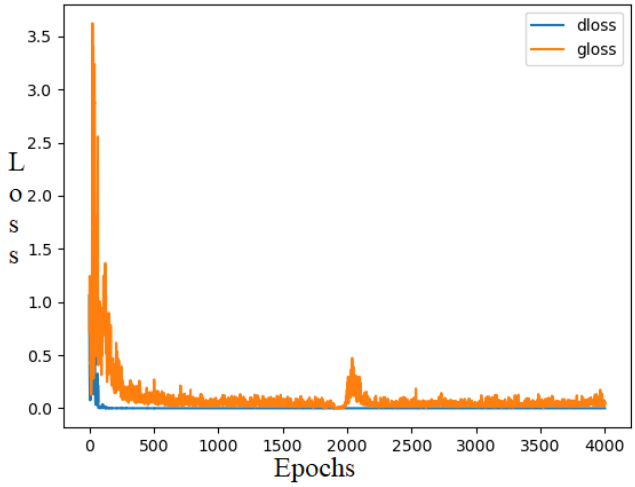


Figure 2.4: Suffering from mode collapse. Images are from experiment number 2

The underlying concept of GANs is straightforward. But achieving a stable training is very difficult for a GAN. For a normal neural network model loss settles down during the training. On the other hand, in a GAN we have not got an explicit loss function for the generator and convergence of the discriminator's loss is not a good thing without the loss of generator gets the same achievement. The generator and the discriminator should learn at the same pace. If the discriminator learns faster than the generator then the generator stars to learn nothing, means failing in the convergence. This situation is called non-convergence. In Figure 2.5 a typical convergence failure can be seen. Over the first iterations the loss of the discriminator drops significantly. In this stage, the generator is not good enough to generate, and we

do not want the discriminator classify them as fake so easily. These images are from experiment 12, and the generated images also can be seen.



Losses over epochs (above) and generated images (below)

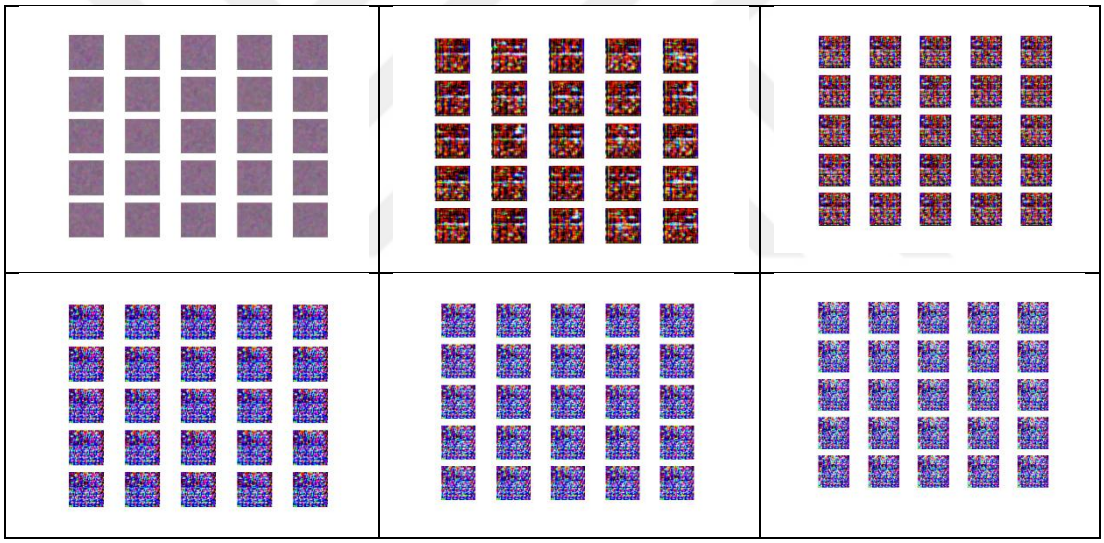


Figure 2.5: Suffering from non-convergence. Images are from experiment number 12

GANs are also very sensitive to the hyper-parameters. Both the discriminator and the generator have to be in a harmony. One of the others parameters can affect the others. And for each network each parameter has to be chosen very carefully. Sometimes finding the correct parameter is a try and evaluate process.

Generator can generate a wide range of data. Generating space is so huge that starting point of the generation becomes very important in the sense of elimination of unrelated distributions. In the real life, meaningful things like an image data, has related attributes in it. We fed the generator with a random noise vector as a

representation of latent space. So, this noise vector should have a specific distribution. We can think of the random numbers as the latent representation of the desired generated data. The multivariate normal distribution can be used for a set of approximately correlated random values that concentrate around a mean value.

2.7 LITERATURE REVIEW

Since the introduction of Generative Adversarial Networks in 2014 [1], their popularity has rapidly increased. GANs have seen as a very good generative model which can produce plausible new data. But their difficulties in training let many researchers propose alternative optimization to address the difficulties and limitations of the original GAN. They concentrated their studies around changing architecture of the GANs' network structures, finding a new loss function and change it and optimizing hyper-parameters of the GANs.

2.7.1 Deep Convolutional Generative Adversarial Networks

DCGAN [4] was one of the first successful implementations among proposed GANs for a better performant implementation. It is based on the original GAN architecture. The main difference is using convolutional neural networks in the hidden layers of the both generator and the discriminator's network. The basic idea is to learn from high dimensional space for image or image like data can only be achieved using convolutional layers in its structure. The common structure of the proposed GAN can be seen in Figure 2.6. It gets so high attention that, most of the other GANs are constructed on this proposal.

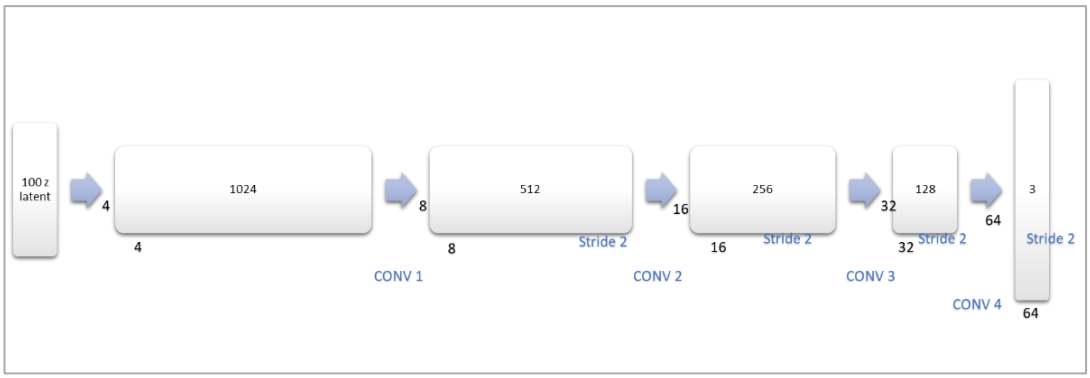


Figure 2.6: The proposed model architecture of DCGAN

The convolutional layers are a bit different from the used in a classical convolutional layered neural network. Upsample layers are used to double the dimensions when generating operation in the generator's structure. As the same manner, when using in the discriminator's network the reverse paradigm should be used which means using transpose convolutional layers. There are some other suggestions other than using convolutional layers. These are:

- Do not use pooling layers instead use strided convolutions
- Batch normalization should be used
- In generator use ReLU for the activation layer
- In discriminator use LeakyReLU
- Use Tanh for the last activation layer of the generator

2.7.2 The Wasserstein GAN (WGAN)

The mode collapse situation which is explained in the previous sections, is a one of the major difficulties in front of the training of the GANs. Researchers and implementors propose many solutions to address this issue in their studies. One of them is Wasserstein GAN (WGAN) [5]. Arjovsky, M. et.al. propose a new loss function to solve this problem. The main objective with their proposal is improvement in the stability on the training process. They also aim to judge the generated samples with their quality also. WGAN has a dense mathematical thinking. But with only a few minor modifications to the DCGAN architecture is adequate to implement in practice.

WGAN is a GAN variant that uses a different technique to train the generator model. Rather than using a discriminator, it introduces a critic model. This model scores the realness of the generated images. This critic model is based on a mathematical model that tries to minimize the distance of the distributions of the image sets: generated images and images in the training set.

WGAN implementation is based on the DCGAN, and main differences are replacement of the loss function with the Wasserstein loss function and replacing the discriminator with a more frequently updated version of critic model.

There are some differences in the WGAN algorithm. These are:

- Do not use sigmoid function in the output layer of the critic, use simple linear activation

- For the critic and the generator models use Wasserstein loss function
- Use RMSProp gradient descent with very small rates and with no momentum
- The critic model is updated more often
- Change the labels of fake and real images with 1 and -1

2.7.3 The Least Squares Generative Adversarial Network

LSGAN is proposed by Mao, X. et.al. in a 2017 paper [6]. The main motivation behind LSGANs is to solve the vanishing gradients problem. This problem occurs when the discriminator learns faster than the generator. To realize about bad samples which are produced by the generator, it tries to find unrelated samples by manipulating the weights according to the distance from the decision boundary of the discriminator. This is very important especially in the beginning of the training iterations. This is a kind of regularization. For more distances from decision boundary causes larger penalty for the generator. Least squares loss function is implemented to the output layer of the discriminator.

With these regularizations, LSGAN also leads to a more stable training which is very important in GANs' trainings. There are some differences in the LSGNA algorithm. These are:

- Change the labels of fake and real images with 0.0 and 1.0
- Apply mean squared error (L2 loss)
- The output layer of the discriminator should be linear

The model structure is shown in Figure 2.7.

Figure 2.7: The model architectures of a typical LSGAN

2.7.4 Progressive Growing GANs

An important technique was proposed in a 2018 paper [19] by Nvidia researchers Tero Karras et al. The model proposes to start from small images when generating samples. For later iteration more convolutional layers are added to generate larger images. So that learning gradually can be achieved to get convincing results. They also propose a new technique to variate the generated images to avoid mode collapse. The generator and the discriminator are trained with a 4x4 pixels. For later iterations gradually adding convolutional layers make the training images up to 1024x1024 resolution. This technique makes the training time less than usual. Because training iterations are done mostly with low resolution images. So, the time required for entire training is significantly reduced.

2.7.5 Evolutionary GANs

For a consistent training and more convincing outputs, Wang et al. [20] suggested a novel GAN architecture. They called this GAN as Evolutionary GAN (E-GAN). The main difference from other GANs is the objective function of the adversarial. The others use a non-changing one, during the training. But They use an alternating and evolving one. In their techniques they use different metrics to optimize the objective

function. The worst performing generator based on a score is removed and the rest are carried forward to the next iteration. They preserve the best generator at every iteration. The various generators are analyzed and selected for optimal generation.

CHAPTER III

OPTIMIZATION OF GANS

3.1 INTRODUCTION

Since its introduction, many GAN structures have been proposed by many researchers and implementors. In Figure 3.1, a narrow road map can be seen. It is not limited to this; there are many numbers of GANs. A main reason for so many different GANs is its difficulty to train. Some of them are focused to the mode collapse problem, the others to the speed of the training or quality the output result.

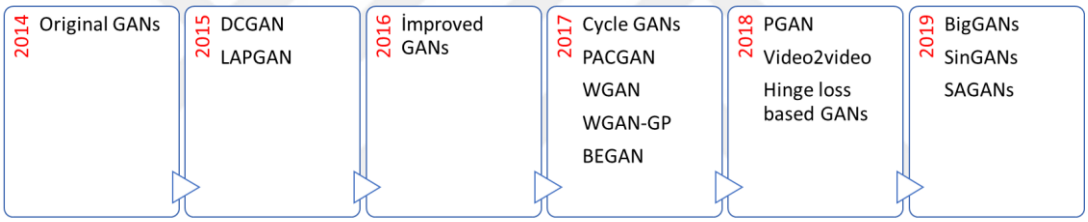


Figure 3.1: A road map of GANs.

There is not an agreement on the one “perfect” GAN structure. We also not introduced the “perfect” one. We show our experiments on the some known GANs and also optimize those algorithms with our hyper-parameters and some structural changes.

As Salimans et al. pointed out in their paper [22], finding an equilibrium is very difficult process, because the cost function of the GANs is non-convex and parameters are very large.

3.2 SOME OPTIMIZATION BASELINES

Before proceeding to the optimizations, we introduce some concepts and common approaches with related to GANs optimizations.

3.2.1 Downsampling using strided convolutions

In a classical deep convolutional network, the pooling layers are used to downsample the images. It seems to be natural to use the pooling layers in the discriminator part, but instead using strided convolutions are suggested.

3.2.2 Upsampling using strided convolutions

The generator model should be in a harmony with the discriminator part. So, when we use downsampling using strided convolutions, it is not a surprise to use opposite in the generator part as upsampling using strided convolutions. So, the generator model can scale a given input to the required output dimensions.

3.2.3 Using Batch Normalization

It is recommended to use batch normalization [23] after the activation layers of the discriminator and the generator models So that stabilization of the training process can be achieved. The activations have a zero-mean variance after batch normalization.

3.2.4 Using LeakyReLU

The rectified linear (ReLU) activation unit is a straightforward calculation that returns the input value directly if it is positive, and returns 0 otherwise. In “leaky” version of this activation function allows some negative values effect in a small negative percentage which makes the network layers much more optimized. For deep convolutional neural networks, using ReLU has become a standard protocol. In a GAN, instead of ReLU, usage of LeakyReLU makes some negative values take effect. Initially, ReLU was suggested for the generator model, while LeakyReLU was suggested for the discriminator model. But in our experiments, we also use LeakyReLU in both networks.

3.2.5 Using Adam Stochastic Gradient Decent

Stochastic gradient descent (SGD), takes one data point at random from the entire data set at each iteration to drastically simplify computations. Adam is a variant of SGD. It can be successfully used in DCGAN training, but with the value of 0.0002 for learning rate and 0.5 for the beta1 momentum value. This is recommended for both

the discriminator and the generator networks. In our experiments we change the parameters to see the results.

3.2.6 Using a Gaussian Latent Space

The DCGAN suggests to sample using a uniform distribution. But a standard Gaussian distribution is suggested to get better results.

3.2.7 Separating Batches of Fake and Real Images

The original GAN architecture combines fake and real images in the training sessions. But separating fake and real images and training them one after another, however, appears to be the ideal method.

3.2.8 Using Label Smoothing

The class label for images in the training set is 1 and for the generated images this value is 0. To get a regularization effect on training we can smooth these values slightly higher or lower (randomly) values. These are called soft labeling.

3.3 USING DIFFERENT LOSS FUNCTIONS

Like any other deep learning neural network, the discriminator model is updated. But when it comes to the generator it has no loss function explicitly. Its loss function comes from the discriminator indirectly. Minimax loss is the original GAN loss function. During the advancement of GANs two more loss functions are introduced and implemented successfully. These are Wasserstein and least squares loss functions.

3.3.1 Minimax GAN Loss

The discriminator and the generator models are optimized simultaneously according to the minimax loss optimization. In two sided games, minimax loss means minimizing the loss for one side while maximizing the others. In our scenario, the generator and discriminator are the two sides which take their turns to update their model weights. The minimax strategy in the GAN seeks to minimize the generator loss while increasing the discriminator loss. As stated in

$$\min \max (D, G) \tag{3.1}$$

$$\text{discriminator's loss: } \max \log D(x) + \log(1 - D(G(z))) \tag{3.2}$$

3.3.2 Least Squares GAN Loss

It is proposed by Xudong Mao, et al. [6]. They experimented that using binary cross entropy loss has some limitations. Gradients have gone to disappearing with this loss function when created images are considerably different from genuine ones. To correct this problem, they proposed increasing the penalty for larger errors, resulting in a significant model correction.

Their methods indicate that for generated and real photos, the class labels of 0 and 1 should be kept, and the least squares should be minimized, means L2 loss.

$$\text{discriminator: } \min(D(x)-1)^2 + (D(G(z)))^2 \tag{3.3}$$

$$\text{generator: } \min(D(G(z)) - 1)^2 \tag{3.4}$$

$$\text{L2 loss} = \text{Sum}(Y_{predicted} - Y_{true})^2 \tag{3.5}$$

3.3.3 Wasserstein GAN Loss

It is proposed by Martin Arjovsky, et al. [5]. They observed that in the traditional GAN architecture, the Kullback-Leibler divergence, or the difference between the actual and projected probability distributions, is not good enough. They proposed to change the philosophy to the Earth-Mover's distance (Wasserstein distance). Using this loss function also changed the role of the discriminator. Now it is updated five times more than the original one, and so its name is changed to "critic". The critic model does not use prediction probability, instead give scores to the images. The model weights are kept small (hypercube of [-0.01, 0.01]). The Wasserstein loss function allows a continuous training of the models. So, it can generate better quality images.

CHAPTER IV

OPTIMIZATION TRIALS

4.1 DATA SET USED

There are two datasets which are used in our experiments. These are MNIST dataset and CELABA dataset.

The MNIST database is the acronym of Modified National Institute of Standards and Technology database. It has a vast library of handwritten digits that is frequently used to train image processing systems. In the field of machine learning, the database is also commonly utilized for training and testing of training algorithms. The dataset contains 70,000 28x28 pixel grayscale images of handwritten digits. The mnist.load dataset() function in Keras gives access to the MNIST dataset. We used only 60,000 training set in our experiments. Figure 4.1 shows an example of data.

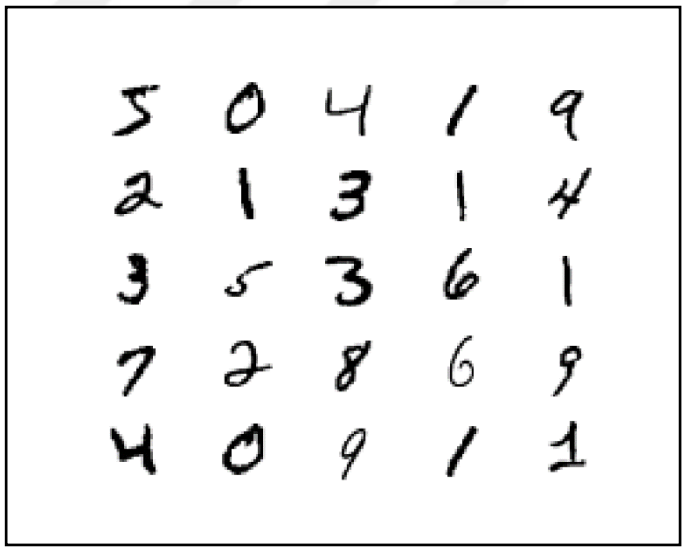


Figure 4.1: 25 sample digits from MNIST data

The CelebFaces Attributes Dataset (CelebA) is a large-scale face dataset with over 200K celebrity photos and 40 attribute annotations for each image. It can be downloaded from Kaggle webpage [24]. This dataset is very big. Because of the

limitations of the computer power, we prepared another version of this dataset using a predefined machine learning model which makes face detection [25]. After all we used 50,000 40x40 pixel colored face images in our experiments. Figure 4.2 shows an example of random data. And after face detection, Figure 4.3 shows the prepared data sample.

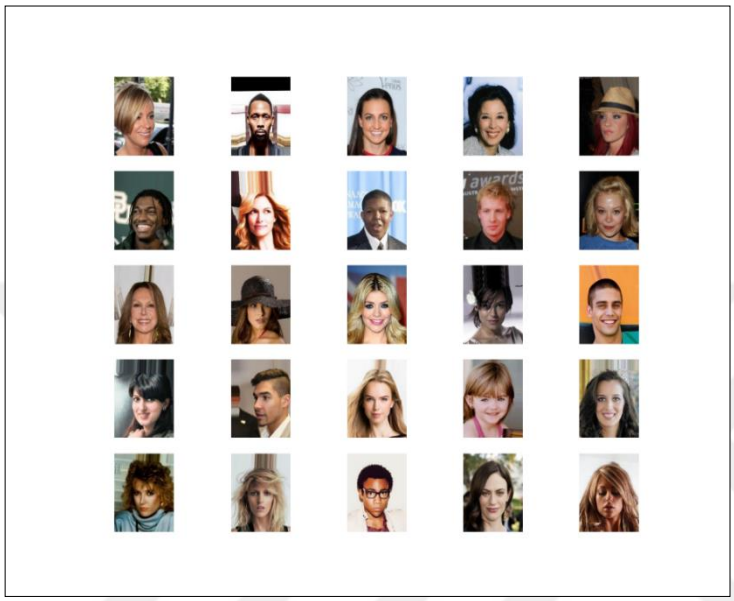


Figure 4.2: A sample from CELEBA data

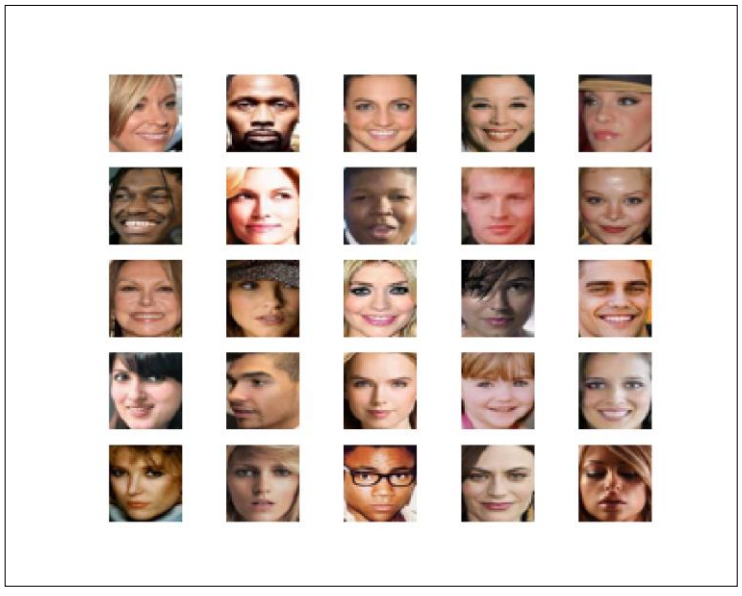


Figure 4.3: A sample from resampled and face detected CELEBA data

4.2 COMPUTING ENVIRONMENT

During experiments this computing environment is used:

- MacBook Pro
- System Memory: 16.00 GB
- System CPU: 8 Core
- System GPU: 8 GPU
- Chip Architecture: M1
- maxCacheSize: 5.33 GB

4.3 EXPERIMENT ENVIRONMENT

More than 30 experiments are conducted, but only 20 of them are included in this report. Because the rest was not controlled, so they are excluded from evaluations. To make experiments comparable, similar code structures are used for all experiments. At the same time only maximum 2 of them are run simultaneously. For some of the GANs same structure are used with both MNIST and CELEBA datasets. For each experiment these files are generated:

1. Model summary of the discriminator (Example: master_gan_mnist_v1_discriminator_modelsummary.txt see Figure 4.4)
2. Model summary of the generator (Example: master_gan_mnist_v1_generator_modelsummary.txt see Figure 4.5)
3. Model summary of GAN (Example: master_gan_mnist_v1_modelsummary.txt see Figure 4.6)
4. Graphical representation of the whole GAN structure with all layers, connection between them and input output layers. (Example: master_gan_mnist_v1_model_plot.png see Figure 4.7)
5. Plot diagram of the discriminator and the generator loss over iterations (Example: master_gan_mnist_v1_plot_loss.png see Figure 4.8)
6. Model parameters and structures are saved every 500 iterations for original GAN and LSGAN based models and, every 200 iterations for DCGAN and WGAN based models. (Example: master_gan_mnist_v1_generator_model_30000.h5)

7. Sample images generated with the current generator in training process are saved every 500 iterations for original GAN and LSGAN based models and, every 200 iterations for DCGAN and WGAN based models. (Example: master_gan_mnist_v1_30000.png see Figure 4.9 and master_lsgan_celeba_v3_30000.png see Figure 4.10)

```
Model: "discriminator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten (Flatten)            (None, 784)               0
_________________________________________________________________
dense (Dense)                (None, 512)               401920
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               131328
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 257
=================================================================
Total params: 533,505
Trainable params: 533,505
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.4: Example of model summary of the discriminator in the experiments

```
Model: "generator"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 256)               25856
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 256)               0
_________________________________________________________________
batch_normalization (BatchNo (None, 256)               1024
_________________________________________________________________
dense_4 (Dense)              (None, 512)               131584
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 512)               0
_________________________________________________________________
batch_normalization_1 (Batch (None, 512)               2048
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              525312
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 1024)              0
_________________________________________________________________
batch_normalization_2 (Batch (None, 1024)              4096
_________________________________________________________________
dense_6 (Dense)              (None, 784)               803600
_________________________________________________________________
reshape (Reshape)            (None, 28, 28, 1)         0
=================================================================
Total params: 1,493,520
Trainable params: 1,489,936
Non-trainable params: 3,584
_________________________________________________________________
```

Figure 4.5: Example of model summary of the generator in the experiments

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
model_1 (Functional)         (None, 28, 28, 1)         1493520
_________________________________________________________________
model (Functional)           (None, 1)                 533505
=================================================================
Total params: 2,027,025
Trainable params: 1,489,936
Non-trainable params: 537,089
_________________________________________________________________
```

Figure 4.6: Example of model summary of the GAN in the experiments

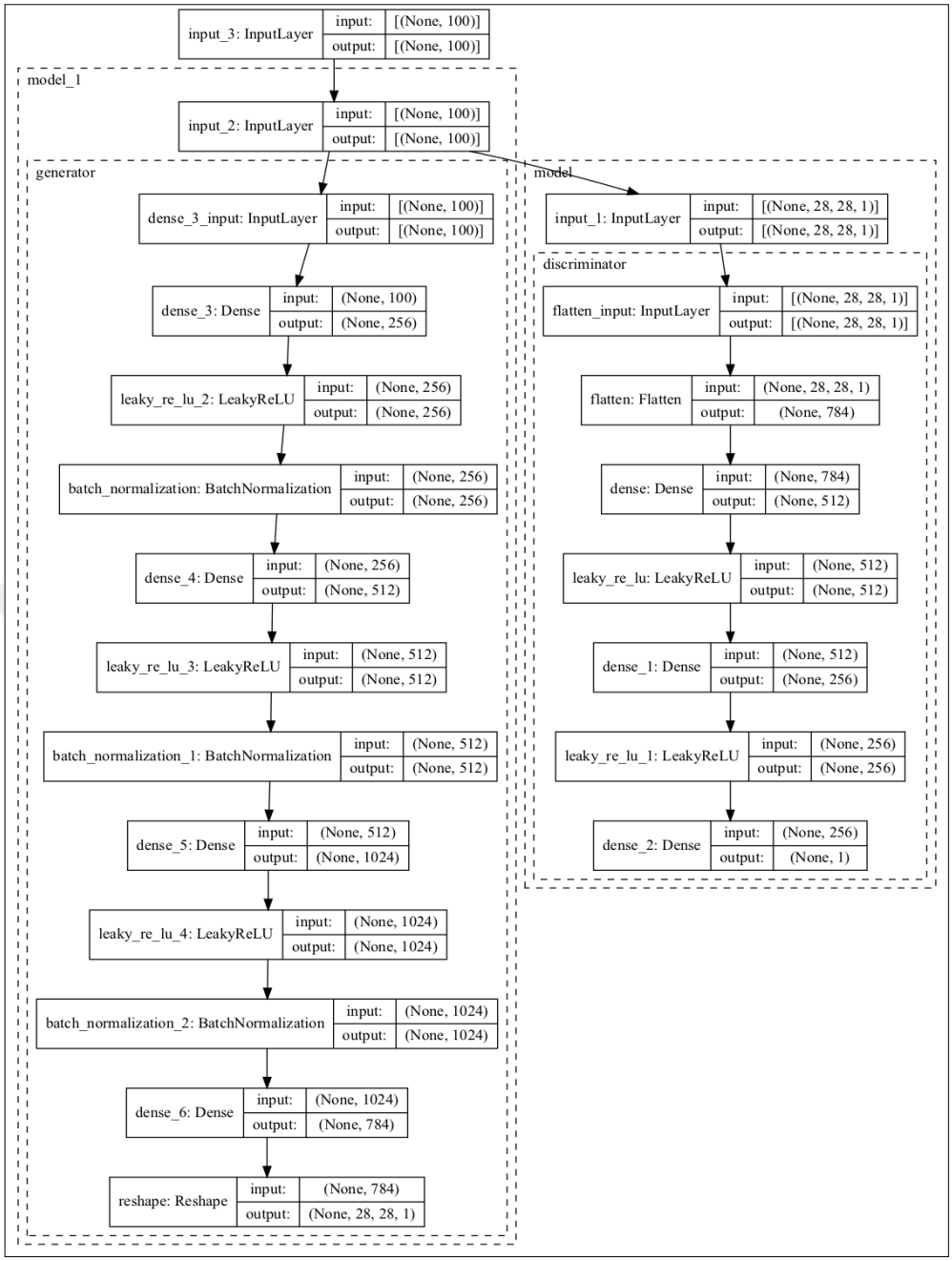


Figure 4.7: Example of model structure figure of the whole GAN in the experiments

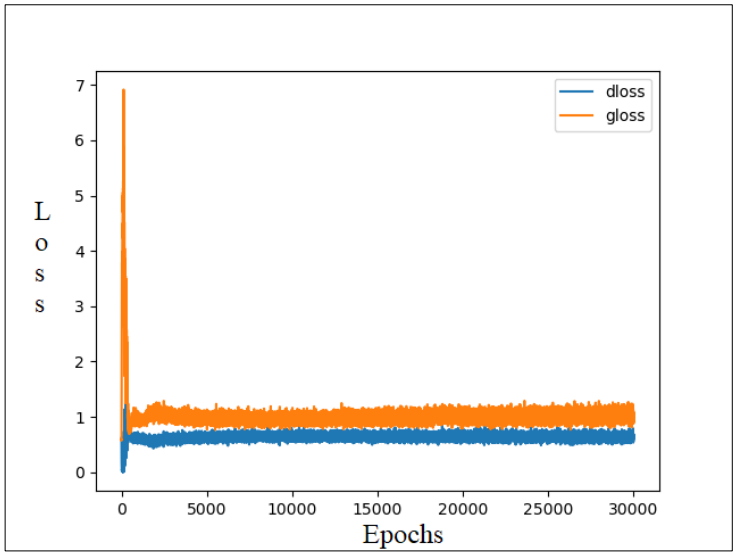


Figure 4.8: Example of plot diagram for the discriminator (blue plot) and for the generator (orange plot) losses, in a typical training session

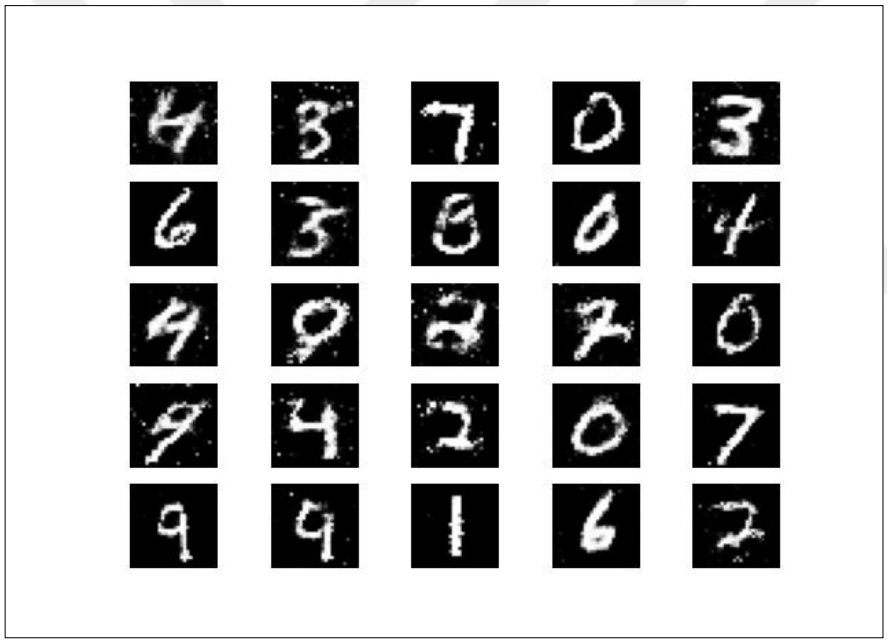


Figure 4.9: Example of generated images with the generator during experiments

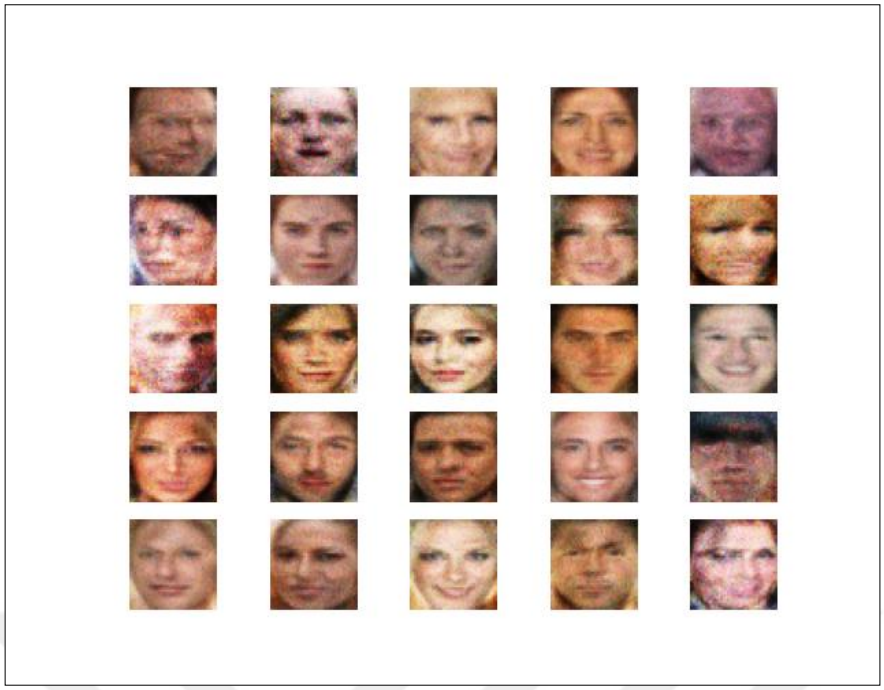


Figure 4.10: Another example of generated images with the generator during experiments

4.4 EXPERIMENT PARAMETERS

We used 4 models for the basis of our experiments. With those structures we changed some of the parameters and in some experiments, we changed the structure of the GANs. The parameters for the experiments are:

1. The GAN's Overall Parameters
   a. Experiment Number
   b. Derived From (inspiration of the GAN structure)
   c. Dataset Used
   d. Number of Epochs
   e. Dimension of Latent Space
   f. Bach Size
   g. Loss Function of the GAN
   h. Optimizer Name
   i. Optimizer Learning Rate
   j. Optimizer Beta1 Parameter (for Adam optimizer)
2. The Discriminator's Parameters
   a. Loss Function of the Discriminator

b. Optimizer Name

c. Optimizer Learning Rate

d. Optimizer Beta1 Parameter (for Adam optimizer)

3. The Generator’s Structure Parameters

a. Activation Functions of Layers

b. Alpha Value (for leakyReLU)

c. Whether Batch Normalization is used or not

d. Momentum Value (for Batch Normalization)

e. Whether UpSampling2d is used or not

f. Whether Conv2D is used or not

g. Kernel Size (for Conv2D)

h. Last Layer Activation Function

4. The Discriminator’s Structure Parameters

a. Activation Functions of Layers

b. Alpha Value (for leakyReLU)

c. Whether Batch Normalization is used or not

d. Momentum Value (for Batch Normalization)

e. Whether Conv2D is used or not

f. Number of Strides (for Conv2D)

g. Kernel Size (for Conv2D)

h. Name of the Regularization if it is used

i. Value of the Regularization if it is used

j. Last Layer Activation Function if it is used

4.5 EVALUATION

A loss function is used to train deep learning neural networks until they approach convergence. On the other hand, in a GAN model the generator learns from the discriminator. For the generator, there is no explicit loss function [22]. To achieve equilibrium, both the generator and the discriminator are trained together. The generated images are used to evaluate a GAN's performance. Manual evaluation of generated images is the first step in the evaluation. But there are some other quantitative measurements to support the visual inspections. In fact, there is no

universally accepted method for evaluating a GAN's generator model. This problem is still an open research area as Borji, A. [26] stated at his paper.

When GAN training is in progress, we have to find a way to stop the training process. While evaluation of the model is a visual inspection, we should save the models in an interval. Actually, we saved the generator models in some steps. These saved models can be used when the visual inspection is done, and correspondent saved model can be used for the image producing. But it be kept in mind that, this is a subjective evaluation and includes biases of the reviewers.

There are numerous quantitative evaluation metrics which can be used. As of today, there is no agreed technique about GAN's inspections. But some researchers suggest Frechet Inspection Distance (FID) score seems more plausible [26]. In our experiments we use FID score as the comparison unit.

4.5.1 Frechet Inception Distance (FID)

Martin Heusel and his colleagues proposed and applied the FID score technique [27]. The distance between feature vectors calculated for real and generated images is calculated using the FID. So, the similarity of the generated samples to the training set can be determined. Lower scores indicate a closer relationship between the two groups. To find the FID score, it is needed computing power. If the number of the samples in the compared sets are high in number than the generated score is more reliable, but it needs more computing power. In our experiments 5,000 samples from the generated images and 5,000 samples from training set are randomly chosen, then their FID scores are calculated. For the CelebA image set the FID scores are calculated in low values, but for MNIST data set the FID scores may seems to a bit higher. But we only concern with the comparison between different models, these scores are acceptable in our situation. The score reflects the similarity of the two groups of images calculated with the inception v3 image classification model.

4.6 EXPERIMENTS

Experiments are started by implementing advised algorithms of the GAN [1], DCGAN [4], LSGAN [6] and WGAN [5]. While proceeding, we took the visual inspection of the generated images in certain intervals. The generator and the

discriminator's losses are kept in a memory object and at the end of the iterations it is plotted in a file. After each interval period the generator produced sample images, and these images are kept in separate files to inspect. For the same interval the current model structure with its weights is kept in separate h5 files. Any time after the execution we can regenerate images with those h5 files. All important hyper-parameters which is subject to change in our experiments are noted in a file. The GANs architectures are kept in 4 separate files. One for the discriminator, one for the generator and one for overall GAN structure. The fourth structure file is the detailed version of overall GAN structure with all layers and activations and interactions between layers. The sample files are given in the previous chapter. (From Figure 4.4 to Figure4.10).

There are conducted 20 experiments. We used MNIST dataset in 10, and CELEBA dataset in the other 10 of the experiments. After implementing and running original GANs algorithms, FID scores are calculated and noted. After that the structures and hyper-parameters of these GANs are changed and the procedures are started from the beginning.

The complete list of experiments is in Table 4.1.

Table 4.1: Experiments

Exp.	Derived From	Dataset	Epochs	Latent Dimension	Batch Size
1	GAN	MNIST	30000	100	32
2	DCGAN	MNIST	4000	100	32
3	LSGAN	MNIST	30000	100	32
4	WGAN	MNIST	4000	100	32
5	DCGAN	MNIST	4000	100	32
6	DCGAN	MNIST	4000	100	32
7	LSGAN	MNIST	30000	200	32
8	LSGAN	MNIST	30000	200	128
9	LSGAN	MNIST	30000	200	128
10	WGAN	MNIST	5000	100	64
11	GAN	CELEBA	30000	100	32
12	DCGAN	CELEBA	4000	100	32
13	LSGAN	CELEBA	30000	100	32

Table 4.1 (continued)

14	WGAN	CELEBA	4000	100	32
15	DCGAN	CELEBA	4000	100	32
16	DCGAN	CELEBA	4000	100	32
17	DCGAN	CELEBA	4000	100	32
18	DCGAN	CELEBA	4000	100	128
19	LSGAN	CELEBA	30000	100	64
20	LSGAN	CELEBA	30000	200	128

As it can be seen from Table 4.1, original GAN is used only two times: one for MNIST data set and the other for CELEBA dataset. These can be used as a starting point for the rest of the experiments. Latent dimensions are used mostly as 100 dimensions and the batch sizes are mostly 32. Number of epochs are 30000 for GAN and LSGAN derived algorithms and 4000 and 5000 for DCGAN and WGAN derived algorithms. But these parameter changes are not the main differences (versions) of the experiments. All structural differences and hyper-parameters of the generators and the discriminators are given at Table 4.2, Table 4.3, Table 4.4, and Table 4.5.

Table 4.2: Hyper-parameters of the GANs

Exp.	**Loss Function**	**Optimizer**		
		Name	**Learning Rate**	**Beta1**
1	binary_crossentropy	Adam	0.0002	0.5
2	binary_crossentropy	Adam	0.0002	0.5
3	Mse	Adam	0.0002	0.5
4	wasserstein_loss	RMSprop	0.00005	-
5	binary_crossentropy	Adam	0.0002	0.5
6	binary_crossentropy	Adam	0.0002	0.5
7	Mse	Adam	0.0002	0.5
8	Mse	Adam	0.0002	0.5
9	Mse	Adam	0.0002	0.5
10	wasserstein_loss	RMSprop	0.00005	-
11	binary_crossentropy	Adam	0.0002	0.5
12	binary_crossentropy	Adam	0.0002	0.5

Table 4.2 (continued)

13	Mse	Adam	0.0002	0.5
14	wasserstein_loss	RMSprop	0.00005	-
15	binary_crossentropy	Adam	0.0002	0.5
16	binary_crossentropy	Adam	0.0002	0.5
17	binary_crossentropy	Adam	0.0002	0.5
18	binary_crossentropy	Adam	0.0002	0.5
19	Mse	Adam	0.0002	0.5
20	Mse	Adam	0.0002	0.5

Table 4.3: Hyper-parameters of the discriminators

Discriminator				
Exp.	**Loss Function**	**Optimizer**		
		Name	**Learning Rate**	**Beta1**
1	binary_crossentropy	Adam	0.0002	0.5
2	binary_crossentropy	Adam	0.0002	0.5
3	Mse	Adam	0.0002	0.5
4	wasserstein_loss	RMSprop	0.00005	-
5	binary_crossentropy	Adam	0.0002	0.5
6	binary_crossentropy	Adam	0.0002	0.5
7	Mse	Adam	0.0002	0.5
8	Mse	Adam	0.0002	0.5
9	Mse	Adam	0.0002	0.5
10	wasserstein_loss	RMSprop	0.00005	-
11	binary_crossentropy	Adam	0.0002	0.5
12	binary_crossentropy	Adam	0.0002	0.5
13	Mse	Adam	0.0002	0.5
14	wasserstein_loss	RMSprop	0.00005	-
15	binary_crossentropy	Adam	0.0002	0.5
16	binary_crossentropy	Adam	0.0002	0.5
17	binary_crossentropy	Adam	0.0002	0.5
18	binary_crossentropy	Adam	0.0002	0.5
19	Mse	Adam	0.0002	0.5

Table 4.3 (continued)

20	Mse	Adam	0.0002	0.5

Table 4.4: Layer structures of the generators and their hyper-parameters

Generator								
Exp.	**Layers Activation Function**		**Batch Normalization**		**Up Sampling 2D**	**Conv 2D**		**Last Layer Activation Function**
	Name	**Alpha**		**Momentum**			**Kernel Size**	**Name**
1	leakyReLU	0.2	+	0.8	-	-	-	Tanh
2	Relu	-	+	0.8	+	+	3	Tanh
3	leakyReLU	0.2	+	0.8	-	-	-	Tanh
4	Relu	-	+	0.8	+	+	4	Tanh
5	Relu	-	+	0.8	+	+	3	Tanh
6	leakyReLU	0.2	+	0.8	+	+	3	Tanh
7	leakyReLU	0.2	+	0.8	-	-	-	Tanh
8	leakyReLU	0.2	+	0.8	-	-	-	Tanh
9	leakyReLU	0.2	+	0.8	-	-	-	Tanh
10	Relu	-	+	0.8	+	+	4	Tanh
11	leakyReLU	0.2	+	0.8	-	-	-	Tanh
12	Relu	-	+	0.8	+	+	3	Tanh
13	leakyReLU	0.2	+	0.8	-	-	-	Tanh
14	Relu	-	+	0.8	+	+	4	Tanh
15	Relu	-	+	0.8	+	+	3	Tanh
16	leakyReLU	0.2	+	0.8	+	+	3	Tanh
17	leakyReLU	0.2	-	-	+	+	4	Tanh
18	leakyReLU	0.2	-	-	+	+	4	Tanh
19	leakyReLU	0.2	+	0.8	-	-	-	Tanh
20	leakyReLU	0.2	+	0.8	-	-	-	Tanh

Table 4.5: Layer structures of the discriminators and their hyper-parameters

Discriminator										
Exp.	**Layers Activation Function**		**Batch Normalization**		**Conv2D**			**Regularization**		**Last Layer Act. Function**
	Name	**Alpha**		**Momentum**		**Strides**	**Kernel Size**	**Name**	**Value**	**Name**
1	leakyReLU	0.2	-	-	-	-	-	-	-	Sigmoid
2	leakyReLU	0.2	+	0.8	+	2	3	Dropout	0.25	Sigmoid
3	leakyReLU	0.2	-	-	-	-	-	-	-	-
4	leakyReLU	0.2	+	0.8	+	2	3	Dropout	0.25	-
5	leakyReLU	0.2	-	-	+	2	3	Dropout	0.4	Sigmoid
6	leakyReLU	0.2	-	-	+	2	3	Dropout	0.4	Sigmoid
7	leakyReLU	0.2	-	-	-	-	-	-	-	-
8	leakyReLU	0.2	-	-	-	-	-	-	-	-
9	leakyReLU	0.2	-	-	+	2	3	Dropout	0.4	Sigmoid
10	leakyReLU	0.2	+	0.8	+	2	3	Dropout	0.25	-
11	leakyReLU	0.2	-	-	-	-	-	-	-	Sigmoid
12	leakyReLU	0.2	+	0.8	+	2	3	Dropout	0.25	Sigmoid
13	leakyReLU	0.2	-	-	-	-	-	-	-	-
14	leakyReLU	0.2	+	0.8	+	2	3	Dropout	0.25	-
15	leakyReLU	0.2	-	-	+	2	3	Dropout	0.4	Sigmoid
16	leakyReLU	0.2	-	-	+	2	3	Dropout	0.4	Sigmoid
17	leakyReLU	0.2	-	-	+	2	5	Dropout	0.4	Sigmoid
18	leakyReLU	0.2	-	-	+	2	5	Dropout	0.4	Sigmoid
19	leakyReLU	0.2	-	-	-	-	-	-	-	-
20	leakyReLU	0.2	-	-	-	-	-	-	-	-

4.7 RESULTS AND DISCUSSION OF THE EXPERIMENT

Table 4.6 and Table 4.7 show the FID scores of the all experiments. The scores for MNIST data set are higher than CELEBA data set experiments. One reason for it is the computational limitations. It is taken only 5,000 samples to compare. But it is not important, since the reference point of FID is also computed with the same

parameters, and we concern with the comparisons. So, it can be examined MNIST data set experiments and CELEBA data set experiments separately.

According to FID scores the winner configuration for MNIST data set experiments is experiment number 3 with an FID score of 52.4530, while the original implementation of GAN's FID score is 53.7039. Experiment number 8 has the closest FID score with 52.5877. A visual inspection can be done for experiment 3 in Figure 4.11.

According to FID scores the winner configuration for CELEBA data set experiments is experiment number 18 with an FID score of 0.0681, while the original implementation of GAN's FID score is 0.4570. Experiment number 15 has the closest FID score with 0.0793. A visual inspection can be done for experiment 18 in Figure 4.12.

Table 4.6: FID Scores of MNIST data set experiments

Experiment Number	Data Set	FID Score
1	MNIST	53.7039
2	MNIST	62.0163
3	MNIST	52.4530
4	MNIST	55.5659
5	MNIST	53.3753
6	MNIST	53.1102
7	MNIST	53.8863
8	MNIST	52.5877
9	MNIST	56.3247
10	MNIST	55.4045

Table 4.7: FID Scores of CELEBA data set experiments

Experiment Number	Data Set	FID Score
11	CELEBA	0.4570
12	CELEBA	18.5712
13	CELEBA	0.5618
14	CELEBA	0.1605

Table 4.7 (continued)

15	CELEBA	0.0793
16	CELEBA	0.1015
17	CELEBA	0.0844
18	CELEBA	0.0681
19	CELEBA	0.2844
20	CELEBA	0.2584

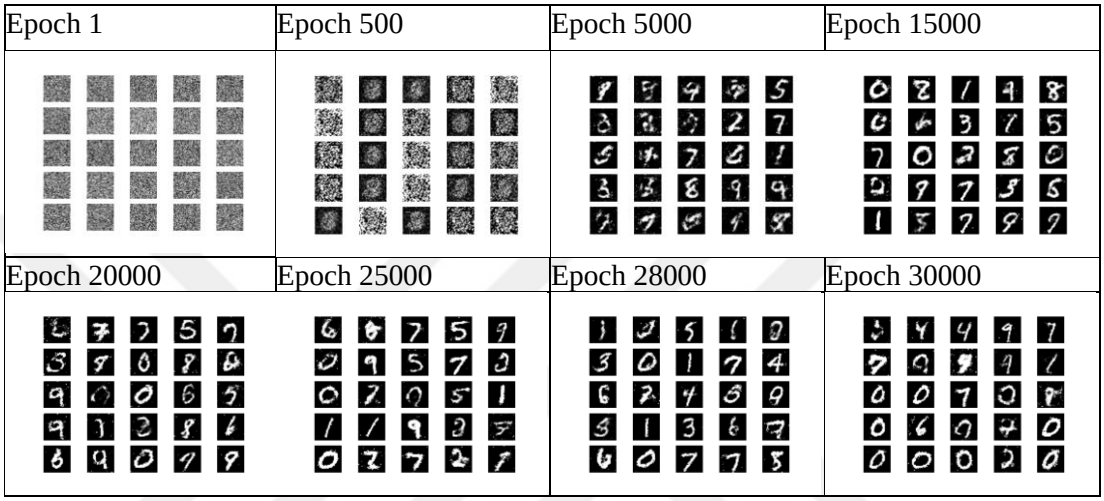


Figure 4.11: Visual inspection of experiment 3

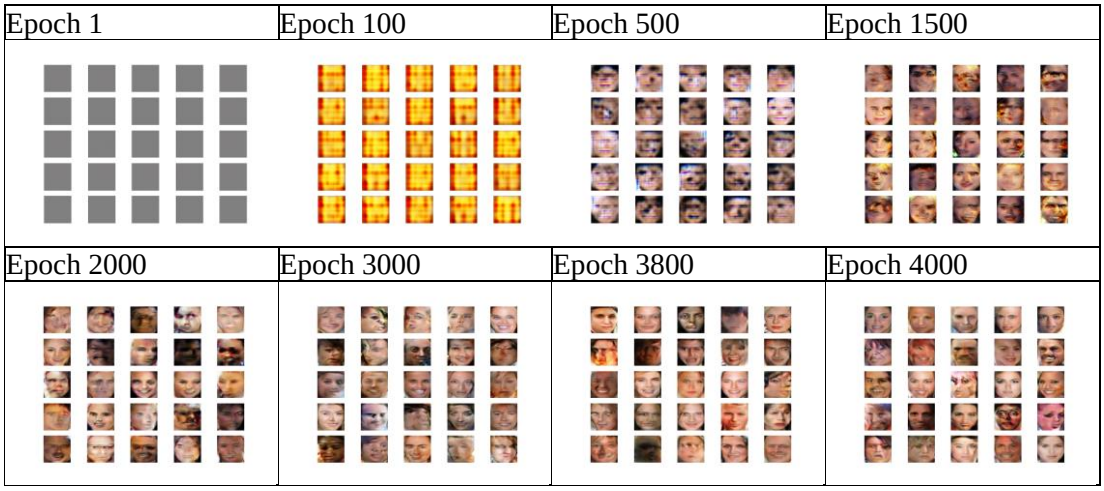


Figure 4.12: Visual inspection of experiment 18

According to these results latent dimension of 100 or 200 and the number of epochs is seen as not very important for the results. Bach size of 32, 64 or 128 are not made any significant changes in the results.

We can see that loss function mse is over binary cross-entropy for MNIST data set. But for CELEBA data set the winner optimizer is binary cross-entropy. The Adam optimizer is significantly dominant, with a learning rate of 0.0002 and a Beta1 value of 0.5 in the performant experiments. The results of leakyReLU with an Alpha of 0.2 and batch normalization with a momentum of 0.8 are excellent at the structures of the discriminator and the generators' network layer designs. Upsampling and downsampling are very important for DCGAN and WGAN derived configurations, but for LSGAN it is better not to use convolutional layers.

With the sigmoid function, the activation function for the discriminators' last layer is excellent. For the generator site, Tanh activation function seems to be a must.

Another important issue is seen from the results that, the discriminator's network structure should be in a harmony with the structure of the generator's network. For example, if convolutional layers are used in one network, it must be used in the other network too.

CHAPTER V

CONCLUSIONS

GANs are fascinating. Interest to this neural network is growing rapidly. Their ability of generating realistic examples makes them very important actor in the generative machine learning filed. As Goodfellow, I. et.al. indicated in their recent paper [28], it is indeed very hard to train them at this time. To make GANs a more credible technology, it will be necessary to provide models, costs, or training algorithms that can consistently and quickly locate good Nash equilibria. To achieve a successful result, its structure has to be built very carefully. The main conclusion of this thesis is, when building a GAN network its generator and discriminator has to be in a harmony in their structures. Similar structures have to be configured and connected. Learning process must be in the same pace. If one of the networks are learning faster than the other than this GAN network is bound to fail. The generalization is very important for the generative tasks. Because of that when building the generator network, for example when synthesize an image, to capture the main features, the network has to handle the problem of generalization as the nature of the problem field, i.e., in this example using convolutions is highly recommended. But for another field, the generalization may be changed. All structures, the discriminator and the generator's losses and sample generated images can be seen at appendices. So that all architectural differences and their effects on visual inspection can easily be discovered. According to the results we concluded that hyper-parameters are very sensitive. Even if the most performant model's a few parameters are changed the results affected very badly. It means hyper-parameters should not be tuned randomly. Tuning suggestions are in the section 4.7.

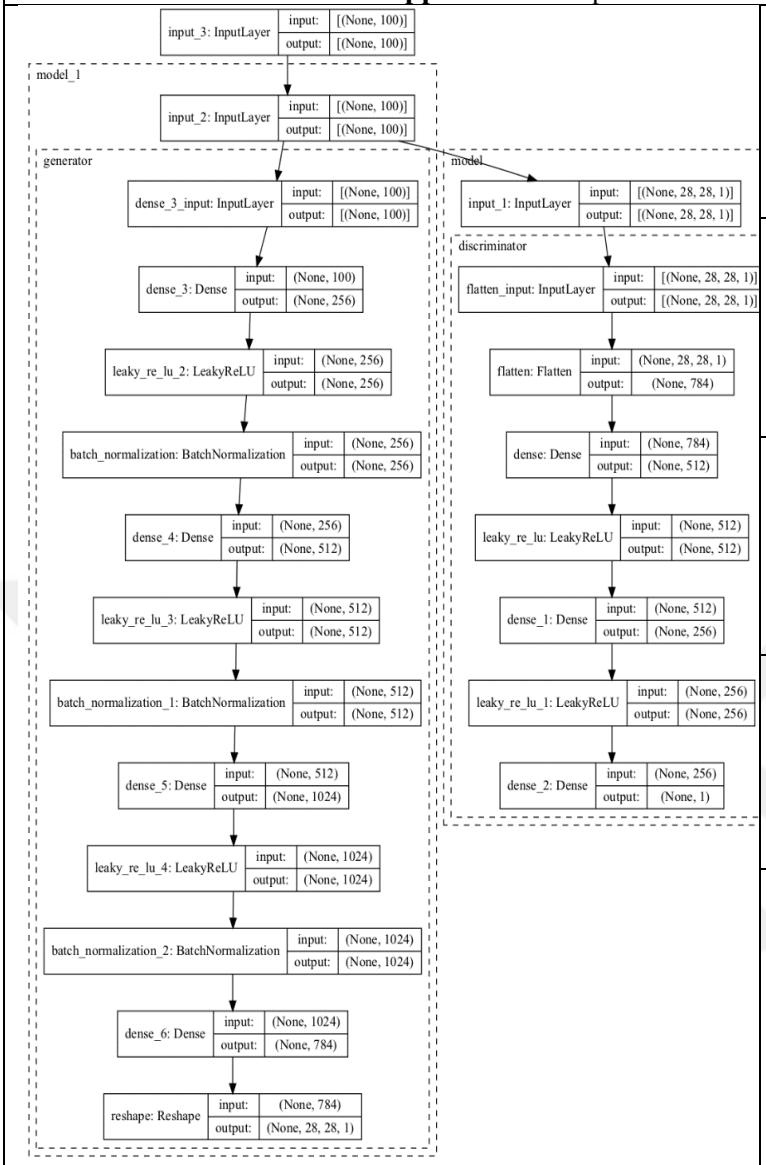
In this thesis we focused on mainly DCGAN, LSGAN and WGAN derived GAN structures. Although these GANs are very important pillars in the GANs' evolutions, there are some promising approaches like evolutionary GANs (E-GAN) [20] and progressive growing GANs [19]. Because of GANs' popularity many new GANs are introduced continuously. This study can be extended with newly introduced

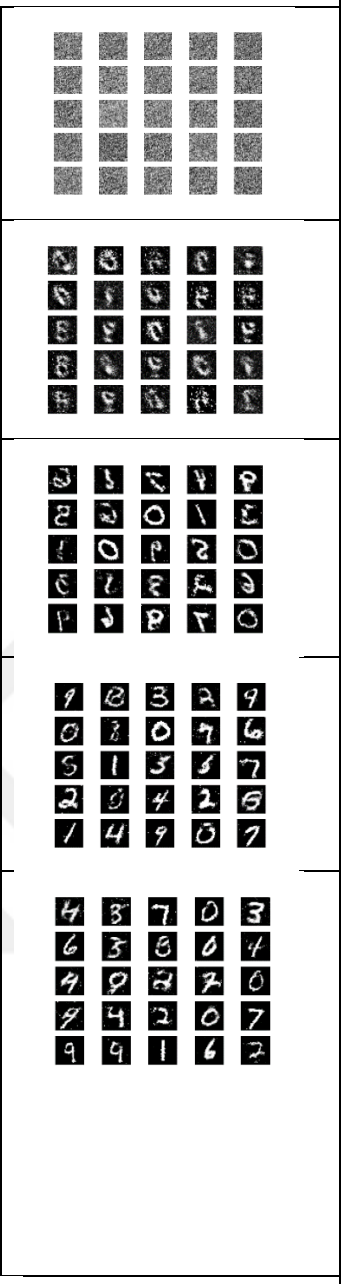
GANs. But more importantly when evaluating a new GAN, it would be very good if we had an evaluation framework and a fair metric.

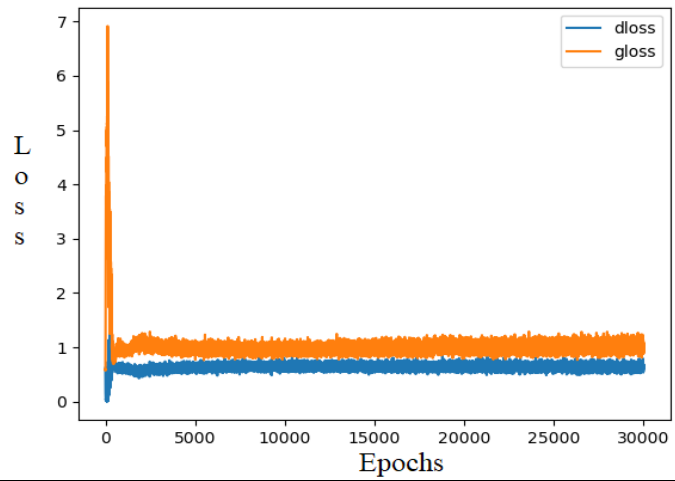
APPENDICIES

This page intentionally left blank

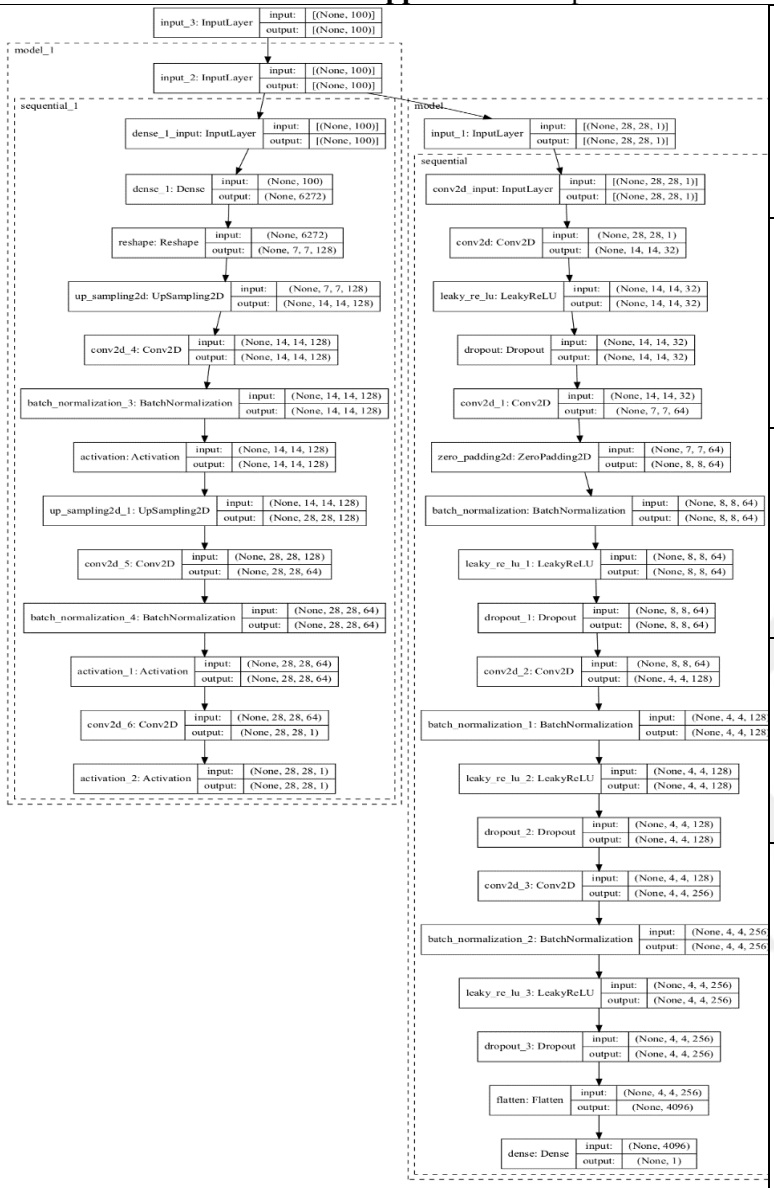
Appendix A - Experiment 1

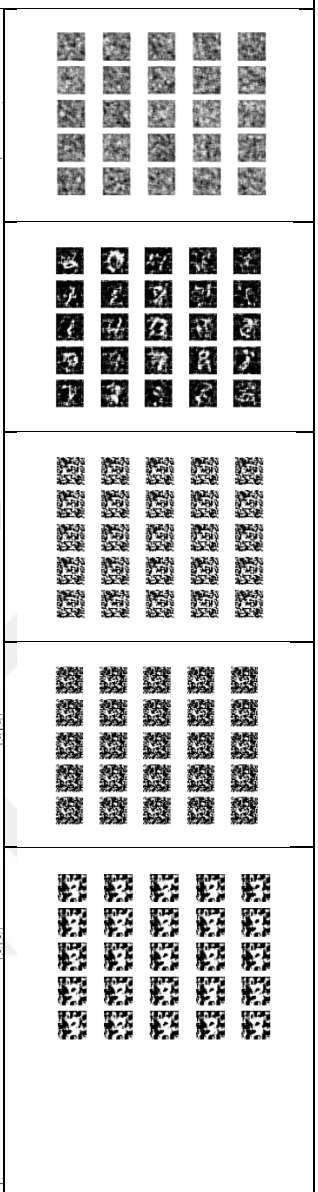


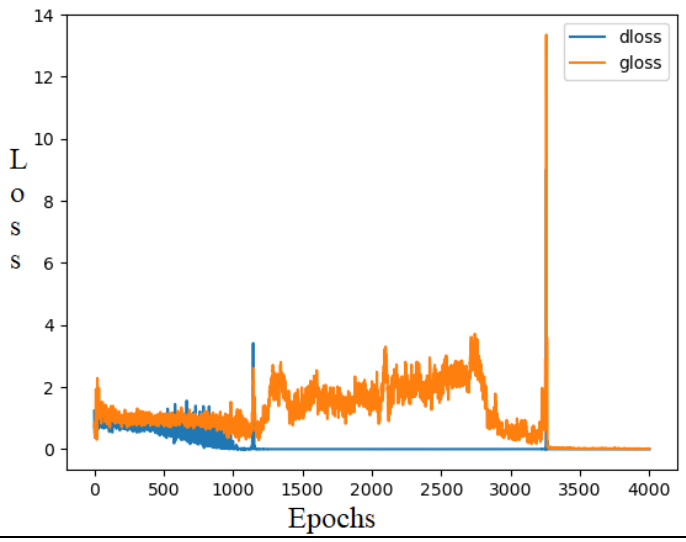




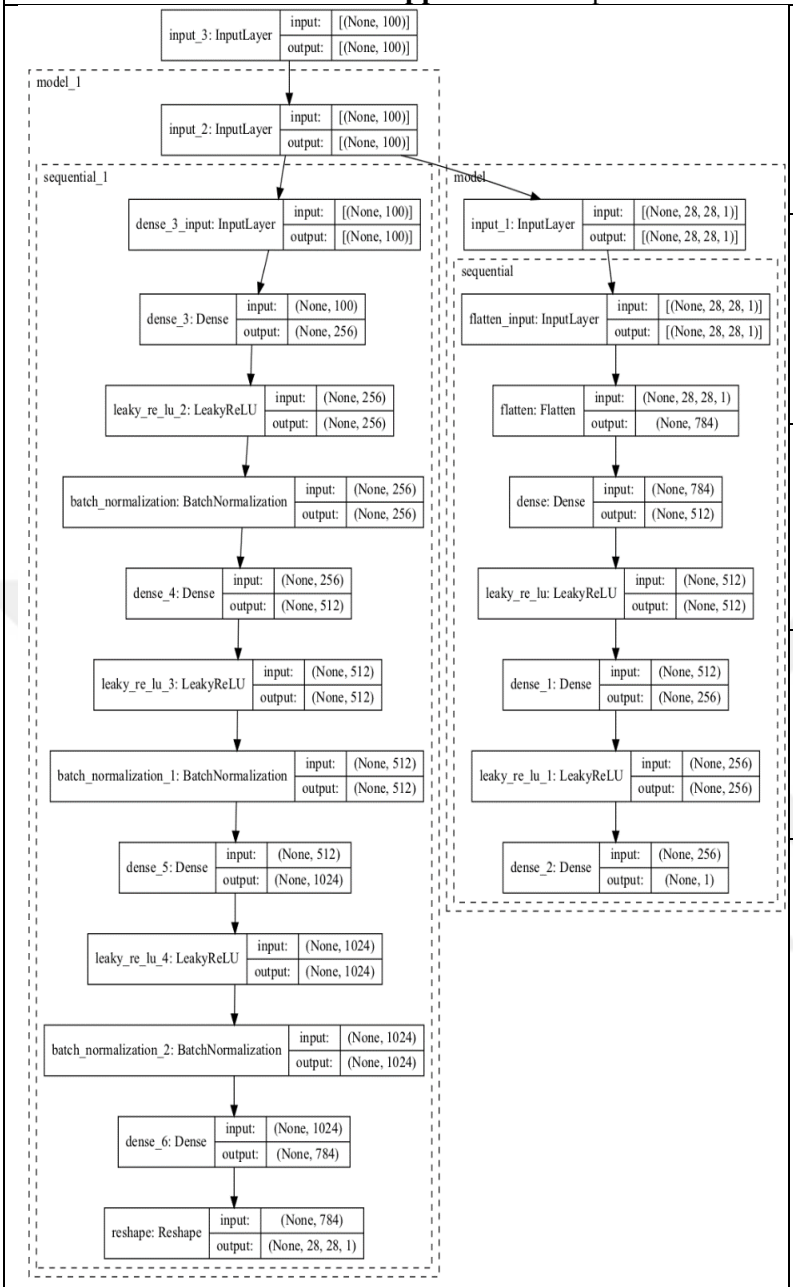
Appendix B - Experiment 2

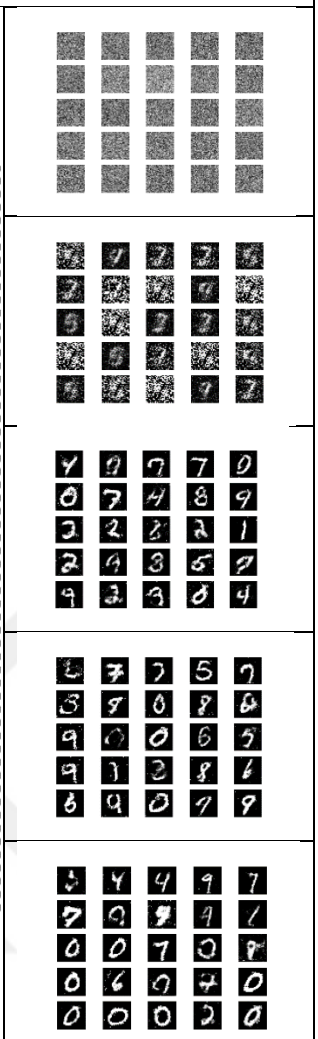


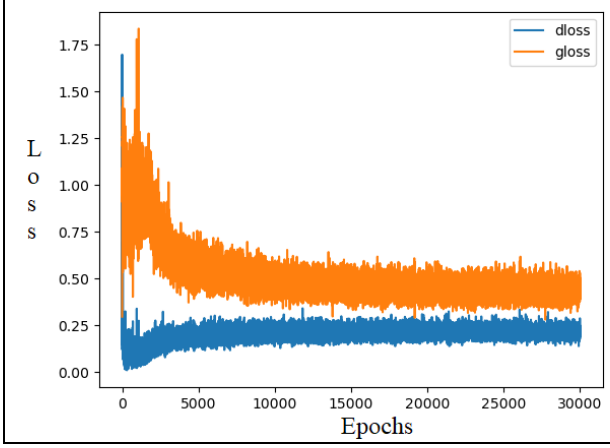




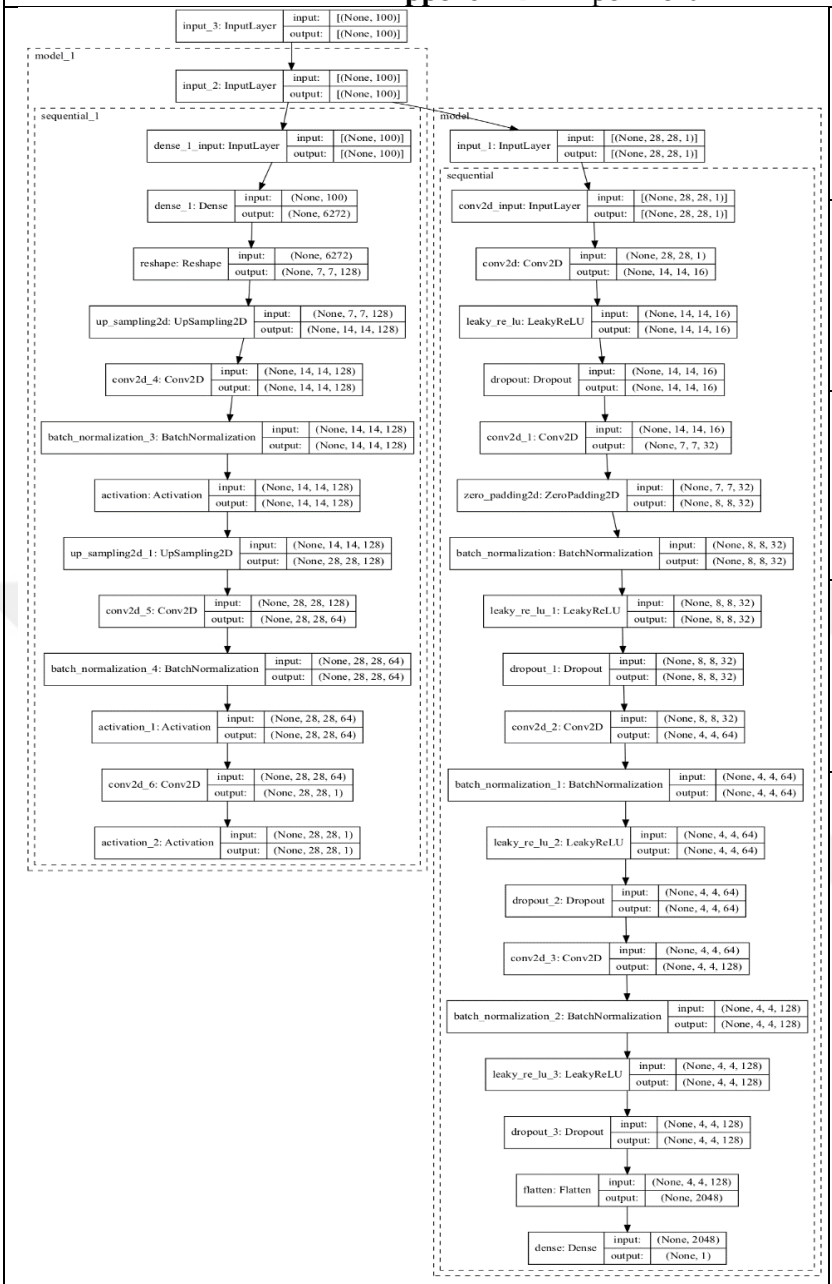
Appendix C - Experiment 3

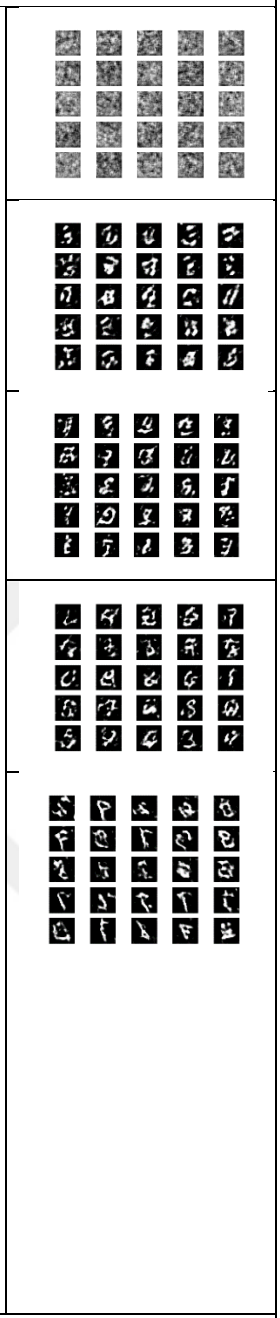


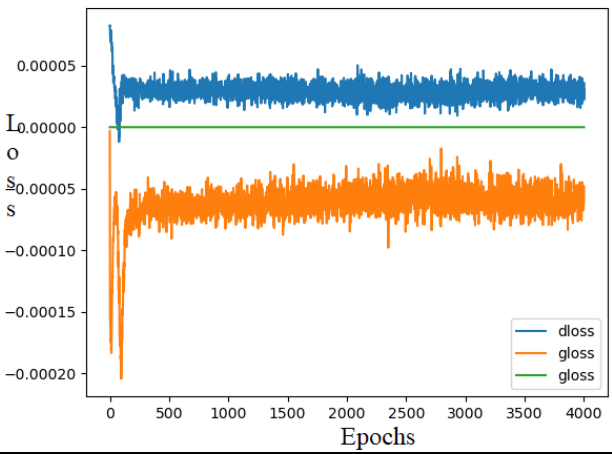




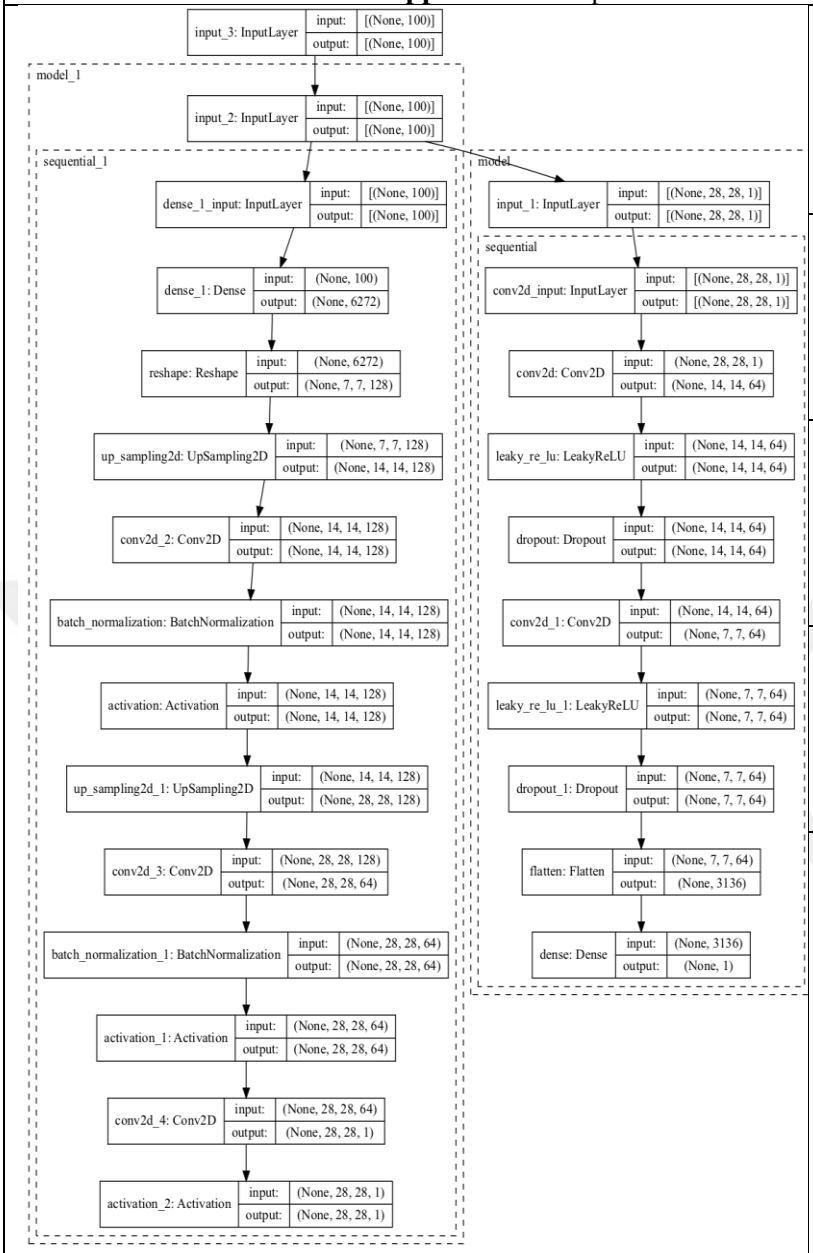
Appendix D - Experiment 4

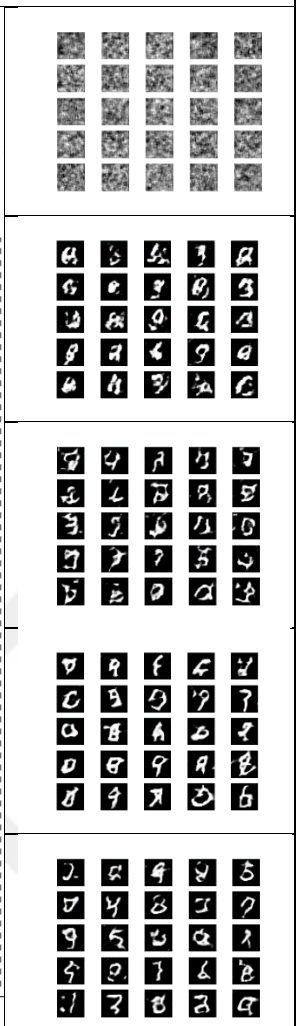


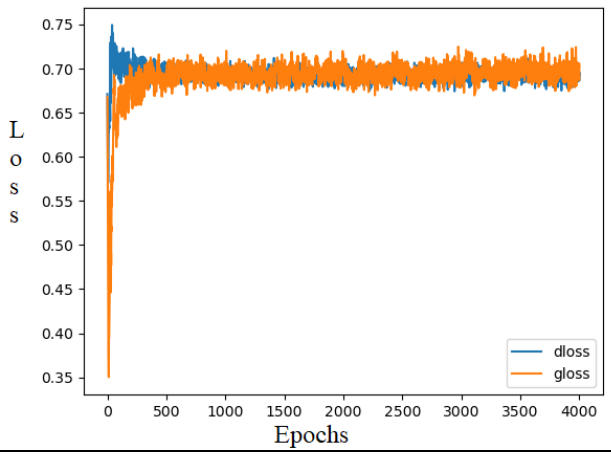




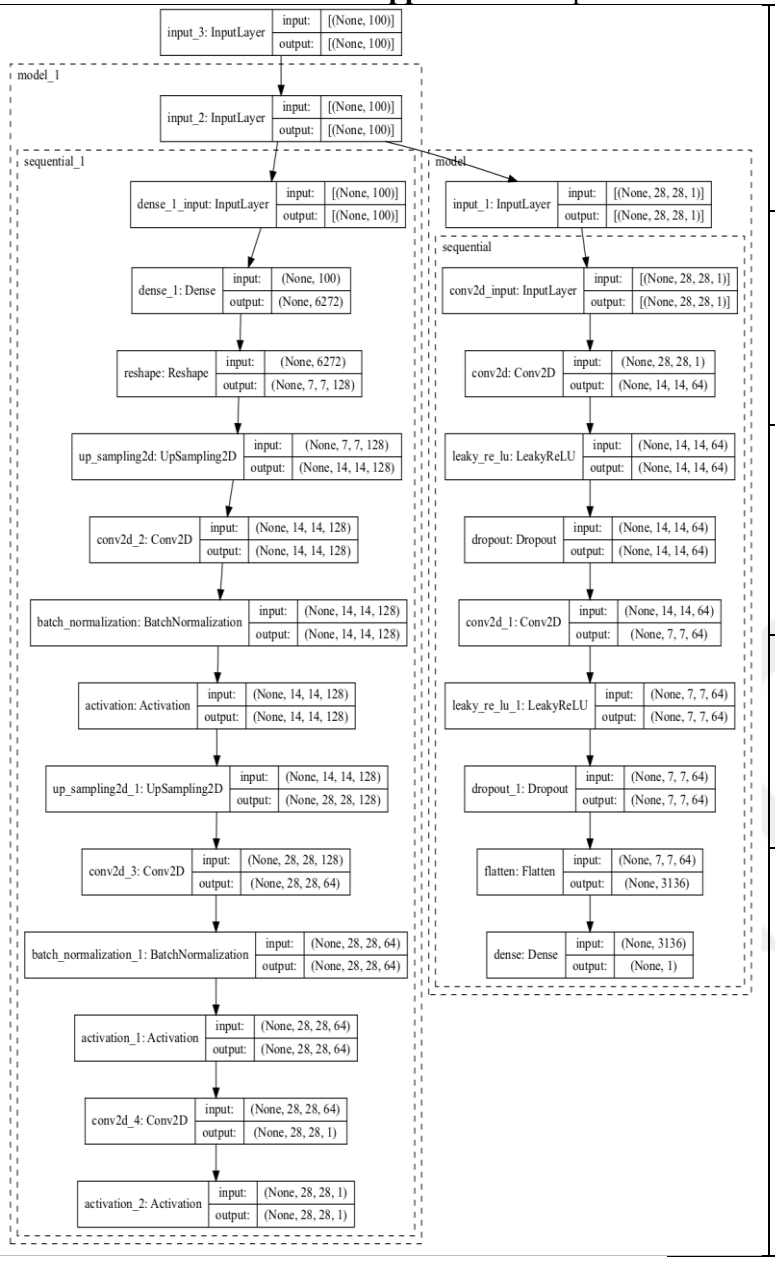
Appendix E - Experiment 5

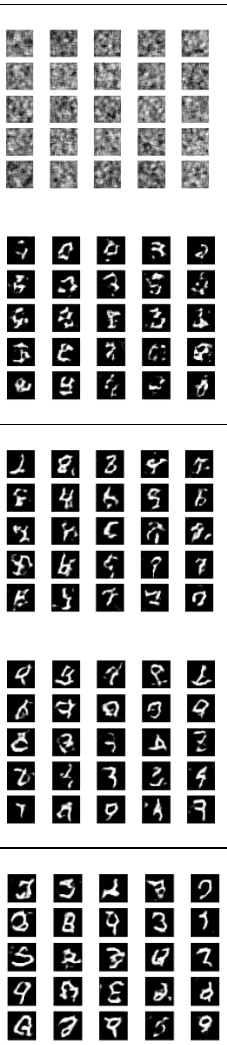


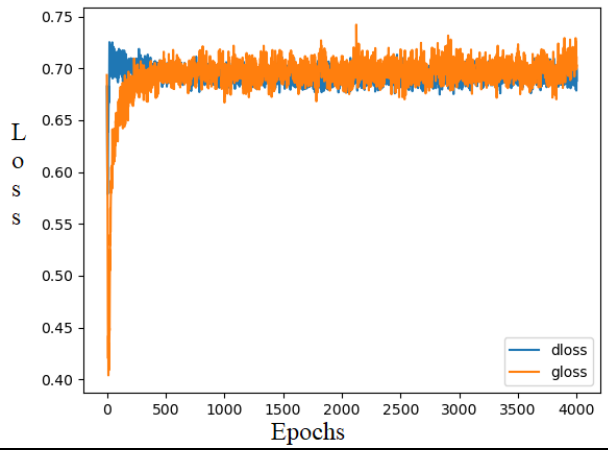




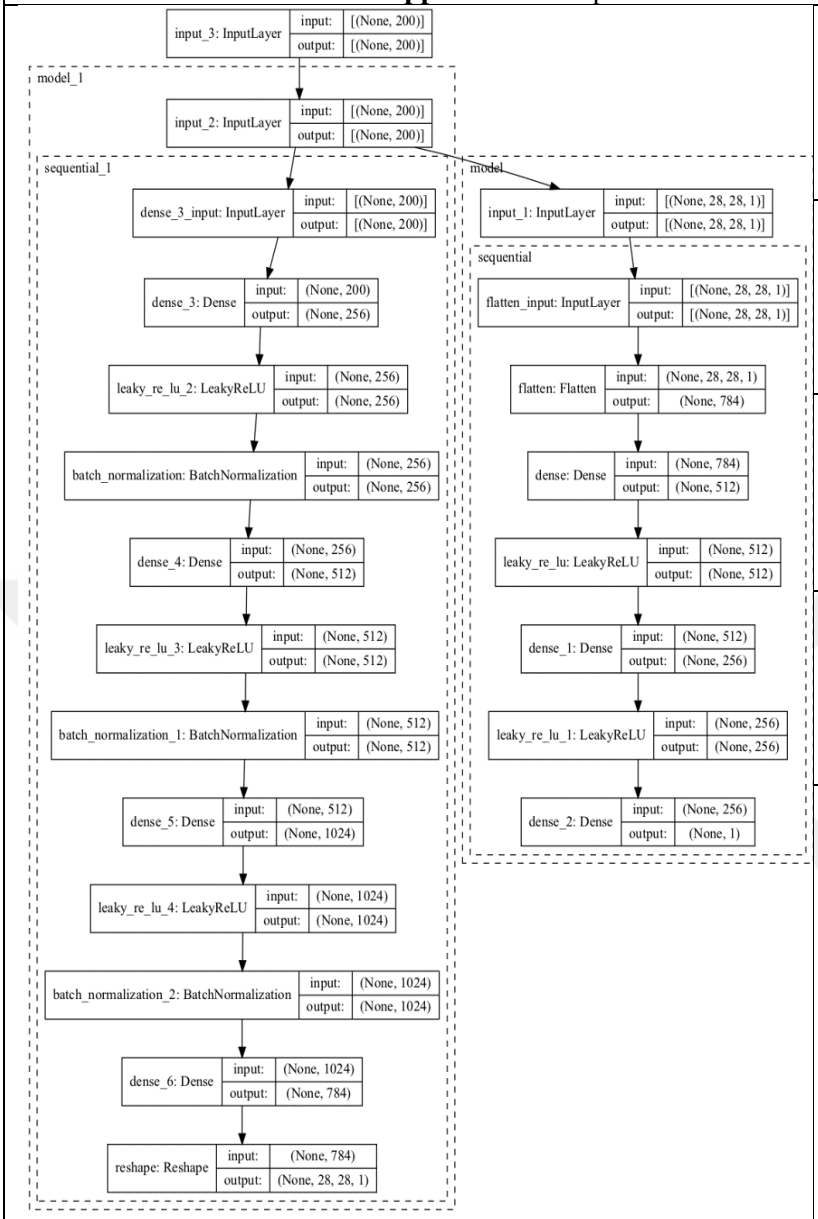
Appendix F - Experiment 6

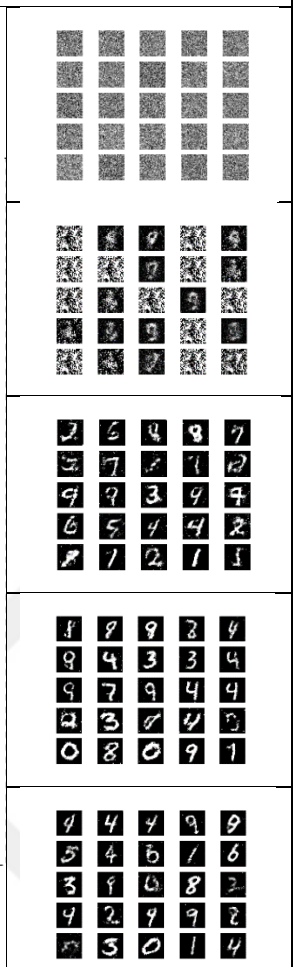


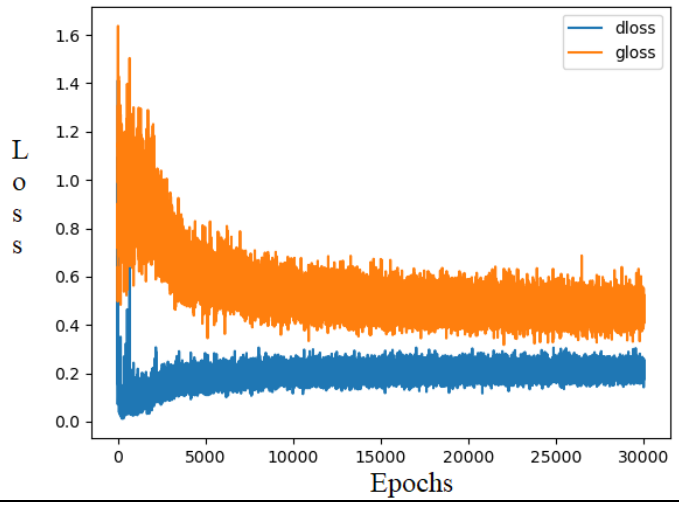




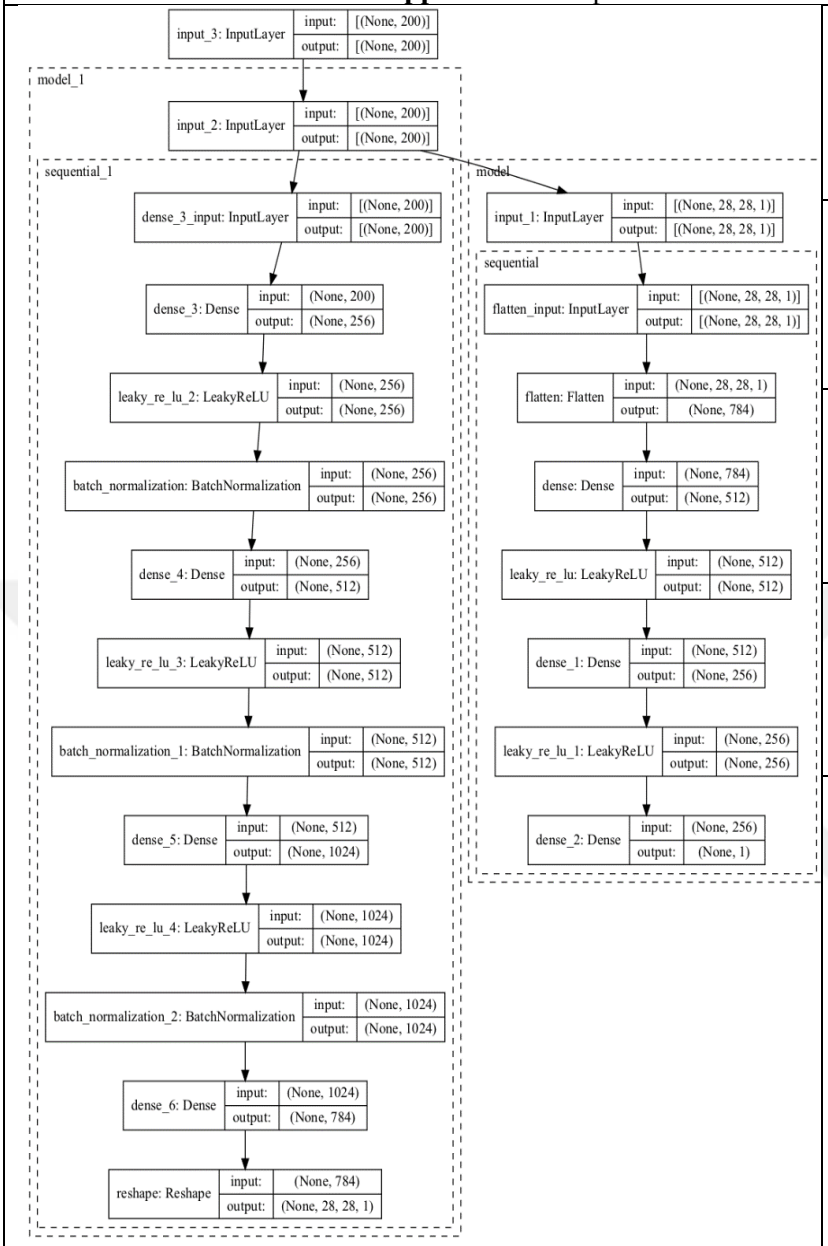
Appendix G - Experiment 7

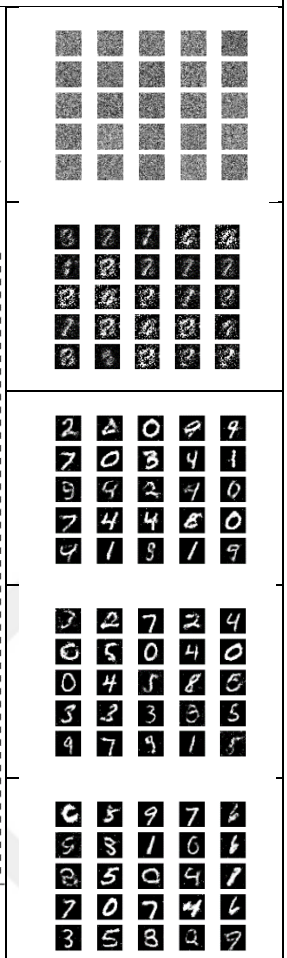


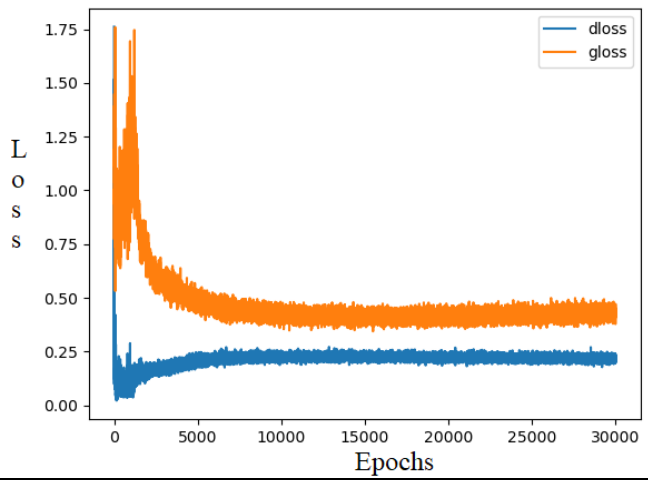




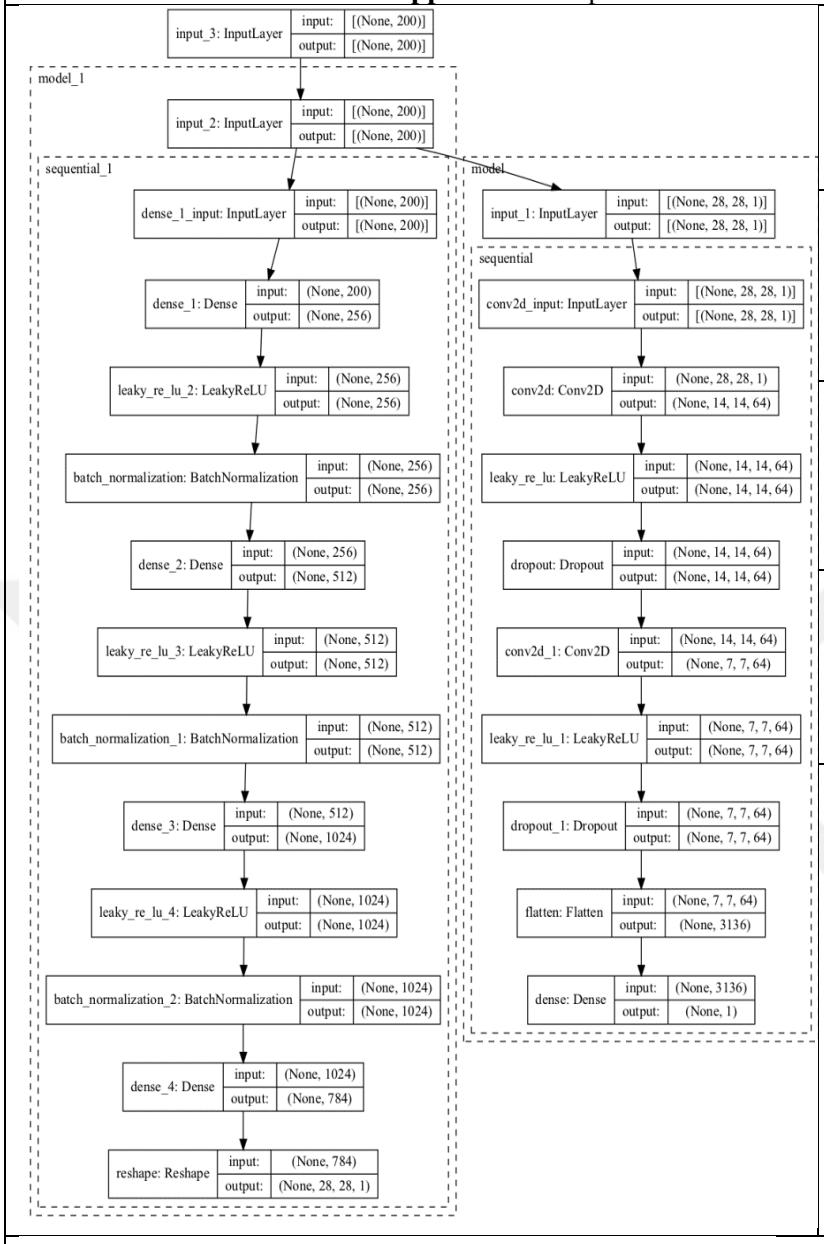
Appendix H - Experiment 8

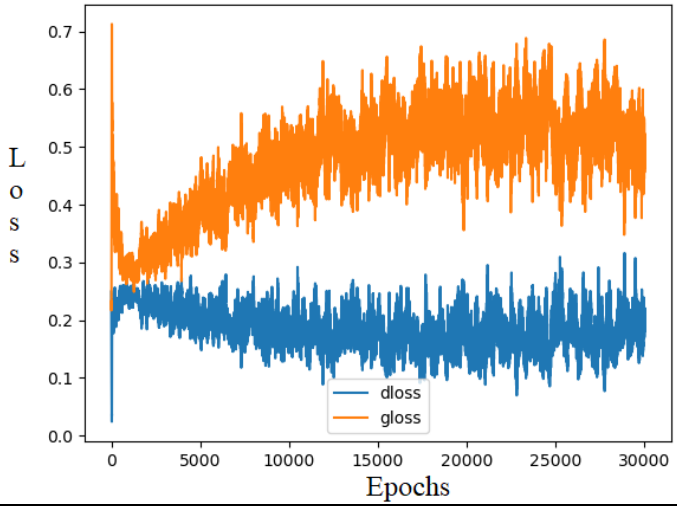




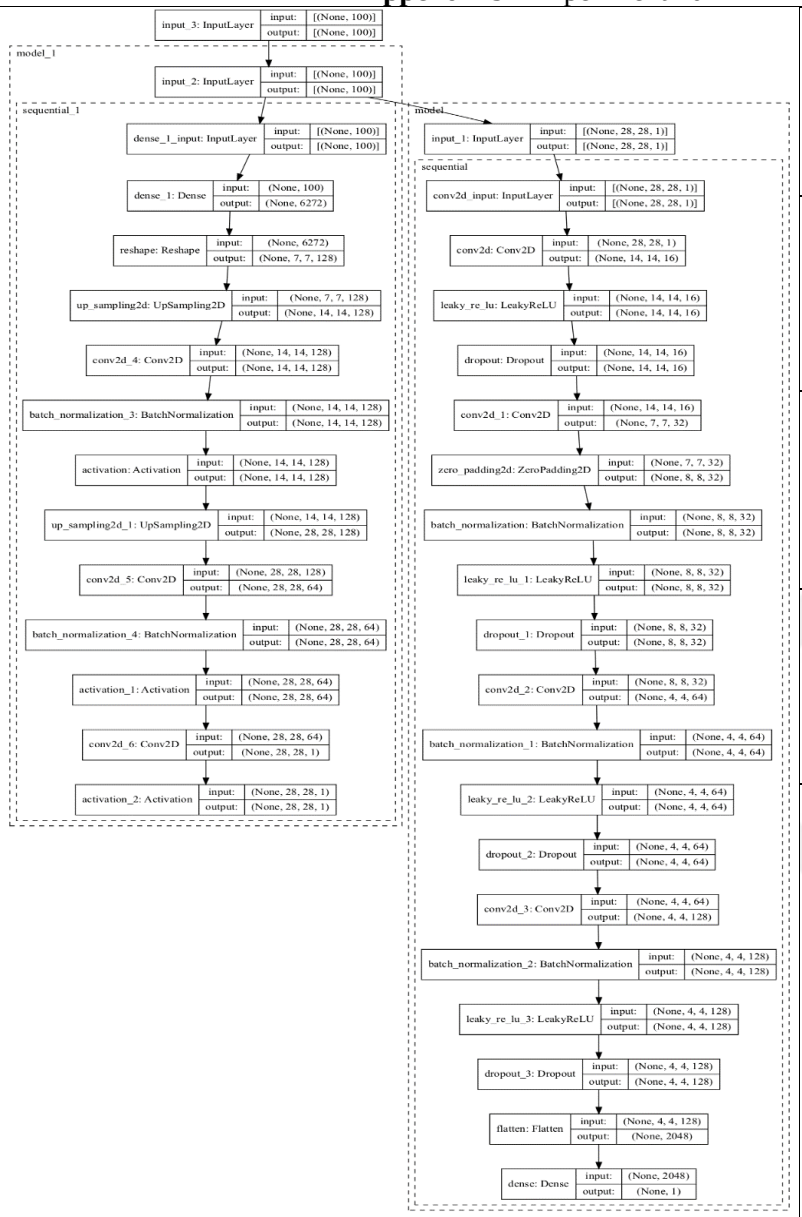


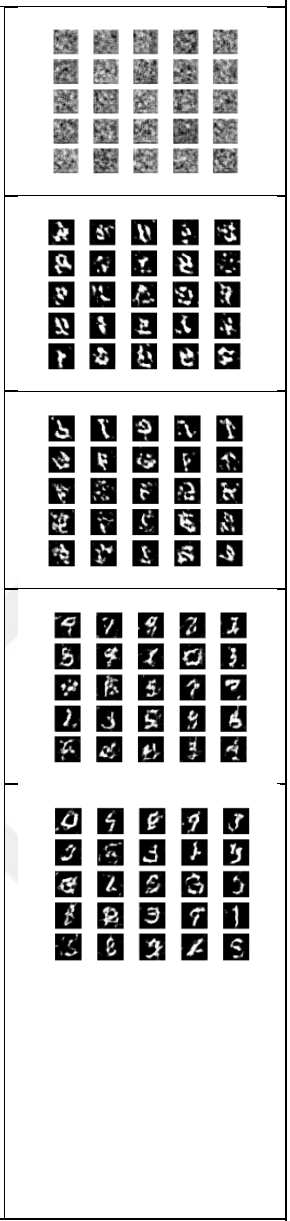
Appendix I - Experiment 9

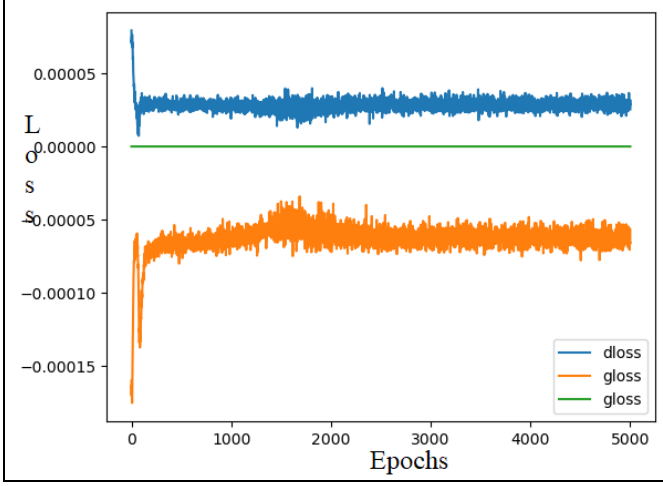




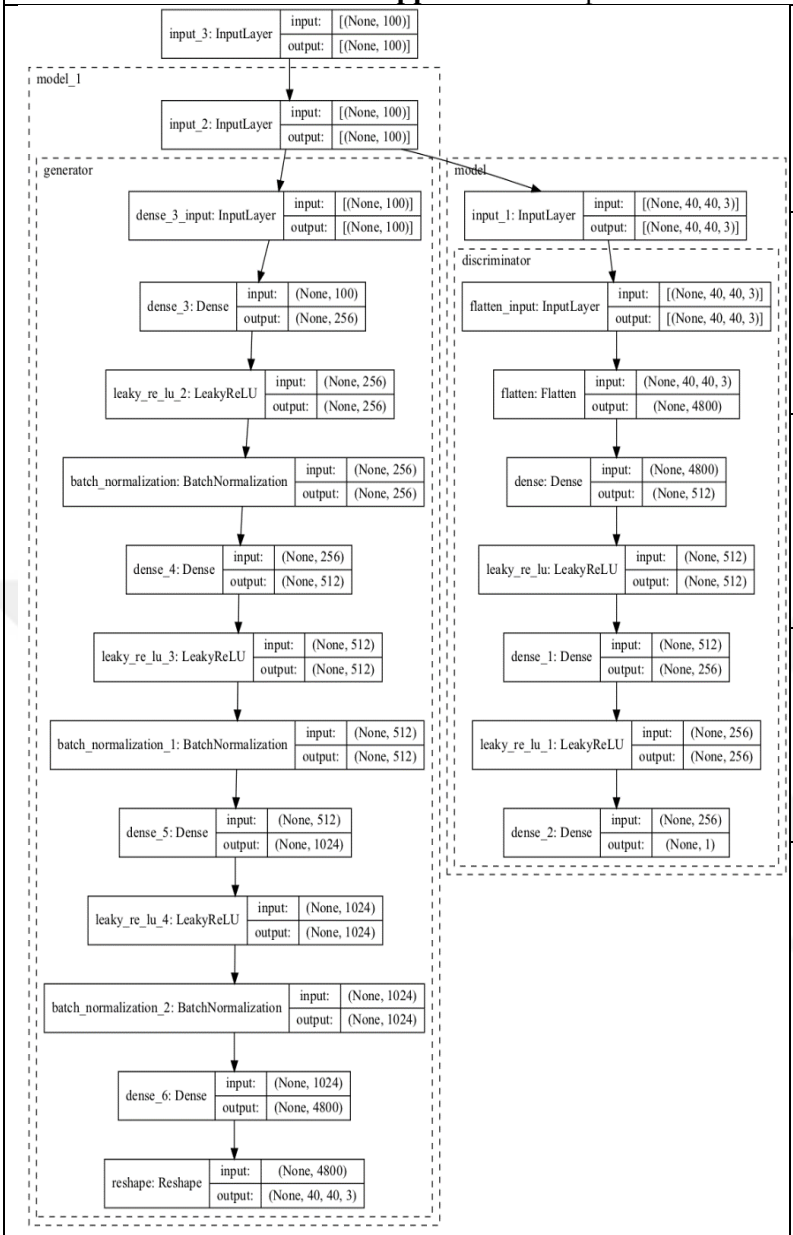
Appendix J - Experiment 10

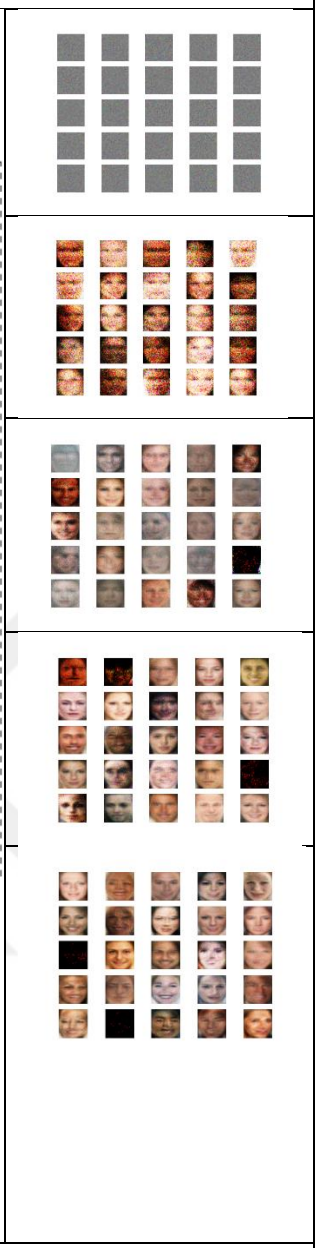


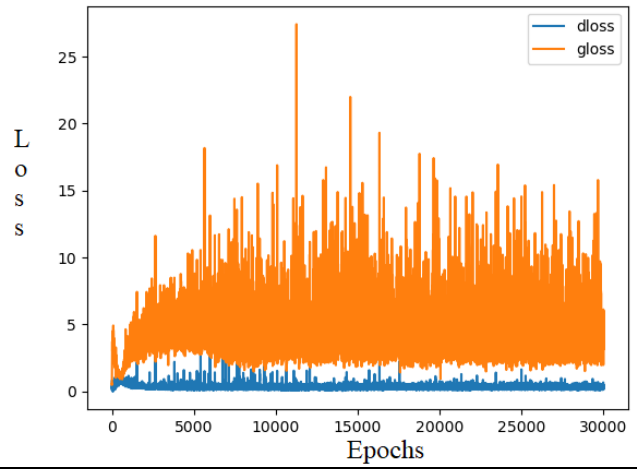




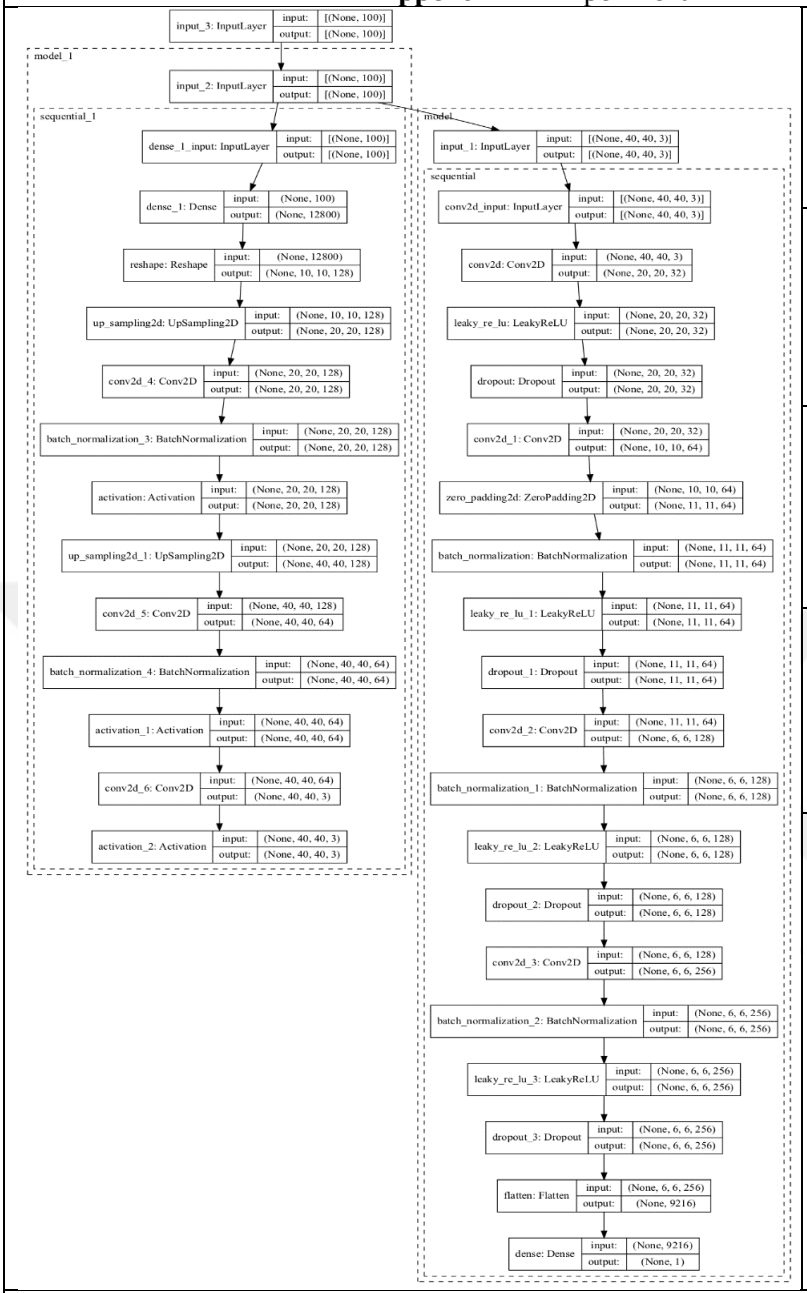
Appendix K - Experiment 11

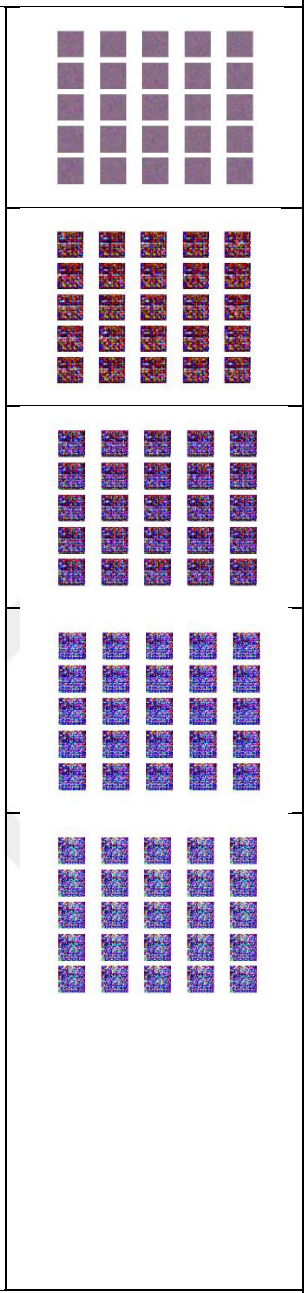


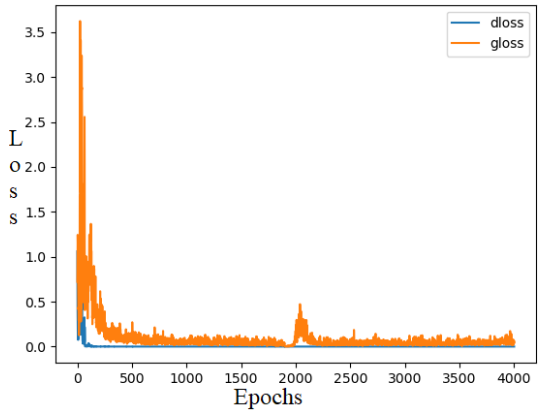




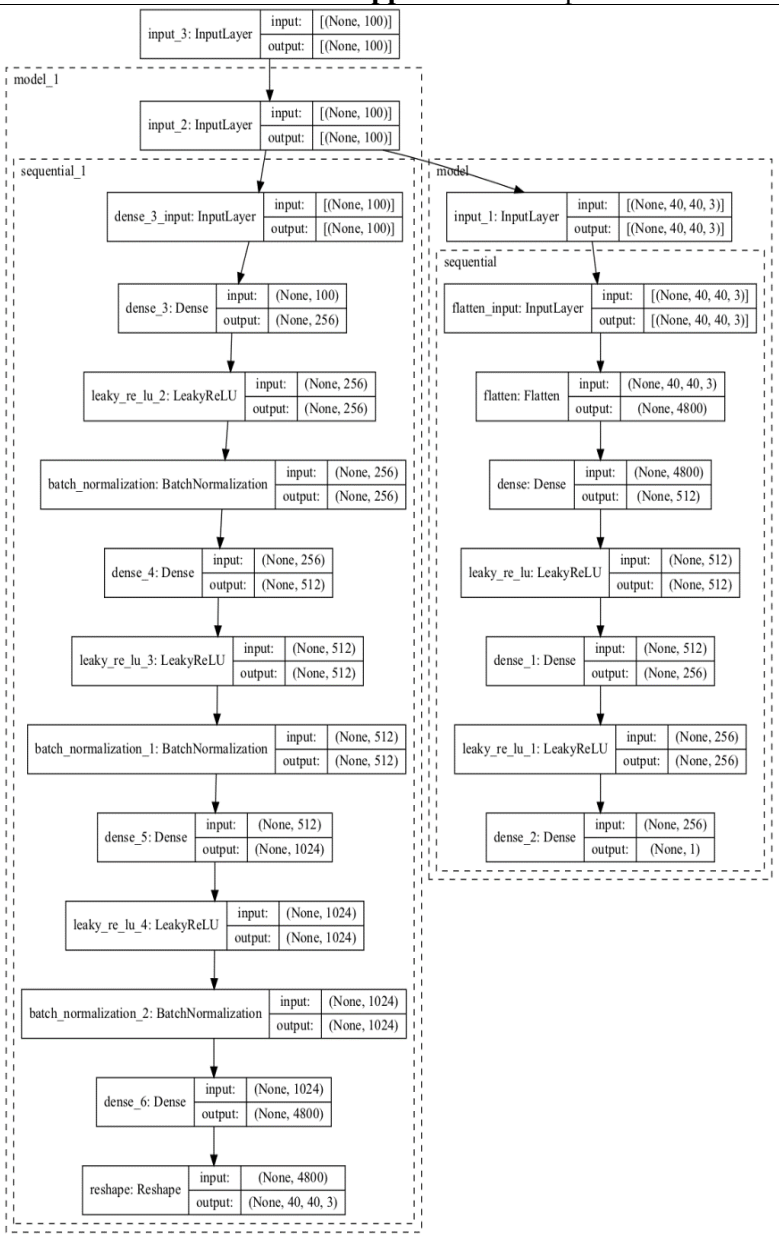
Appendix L - Experiment 12

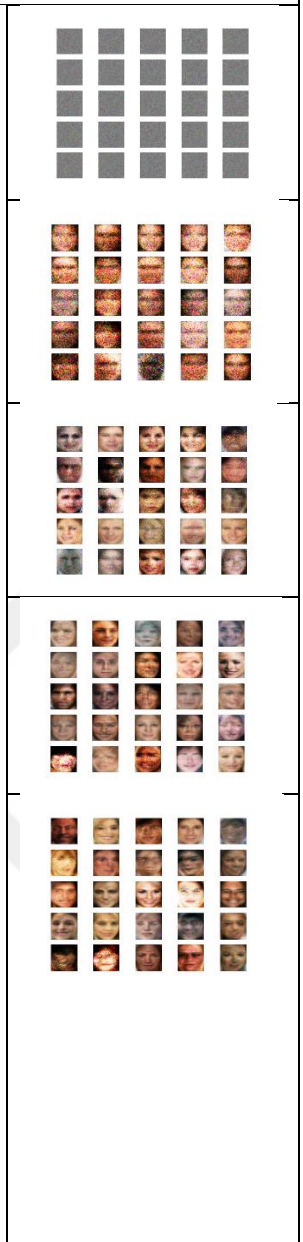


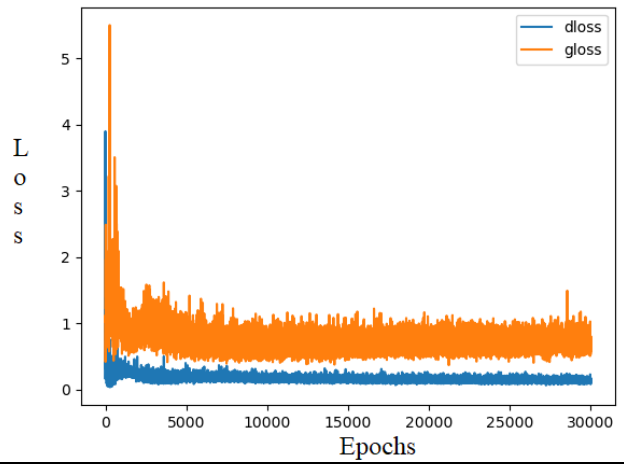




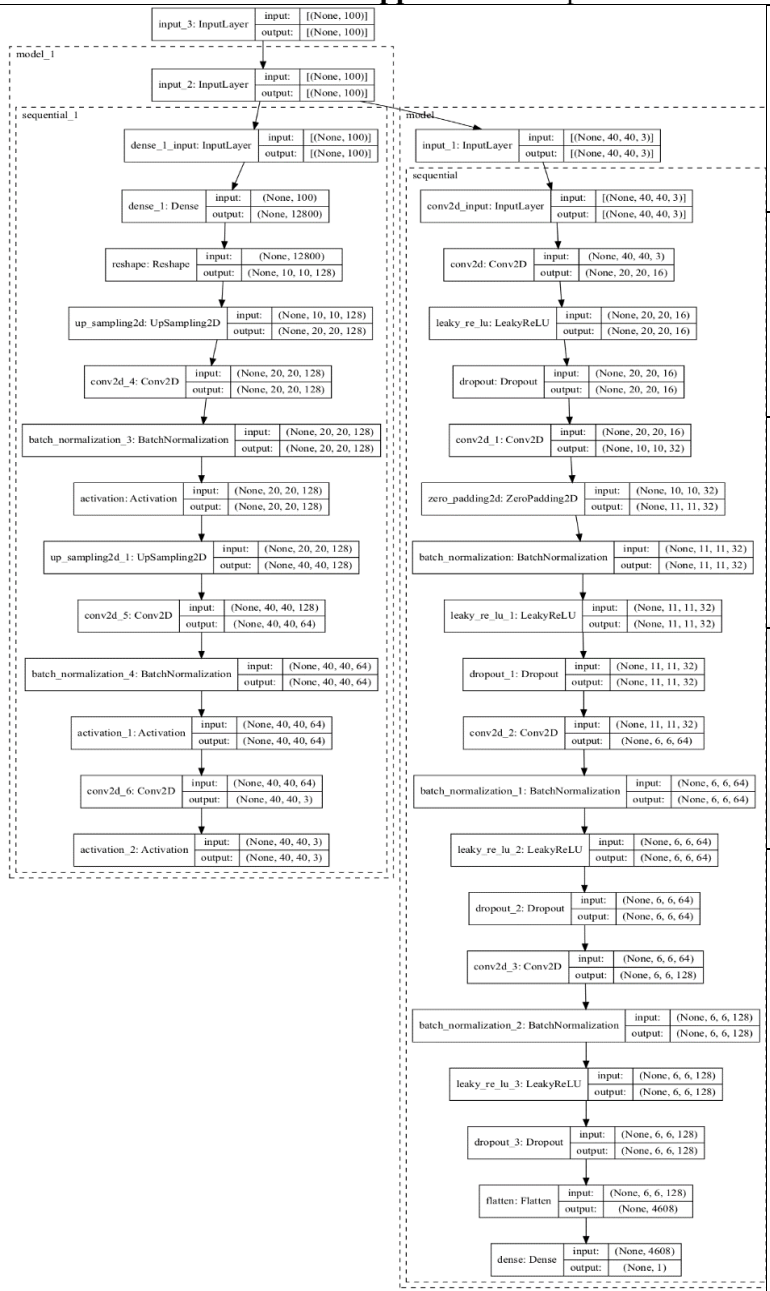
Appendix M - Experiment 13

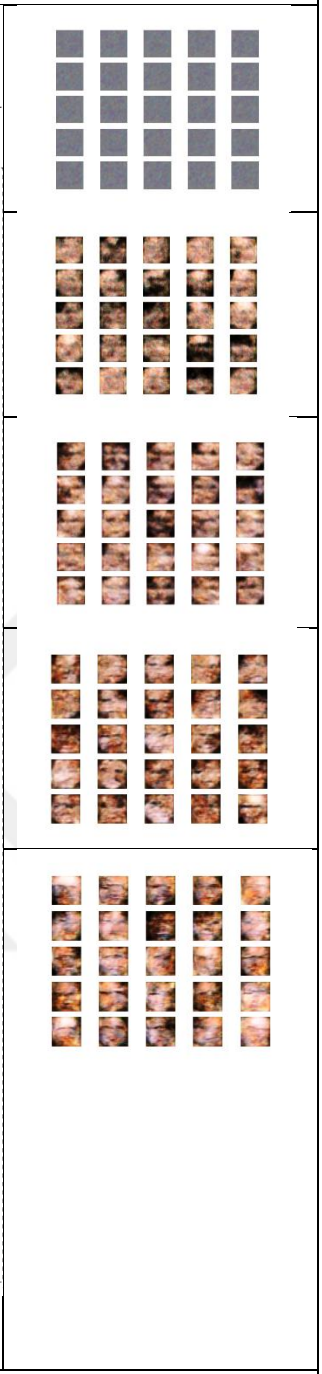


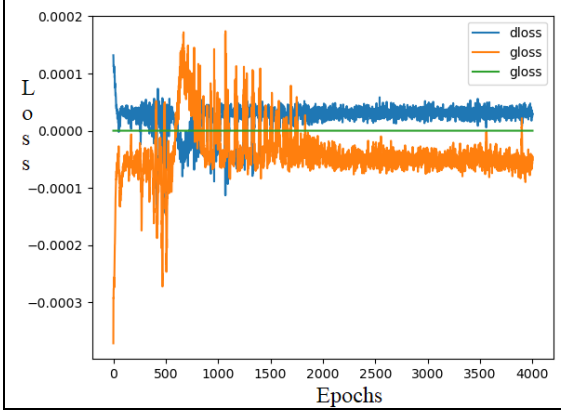




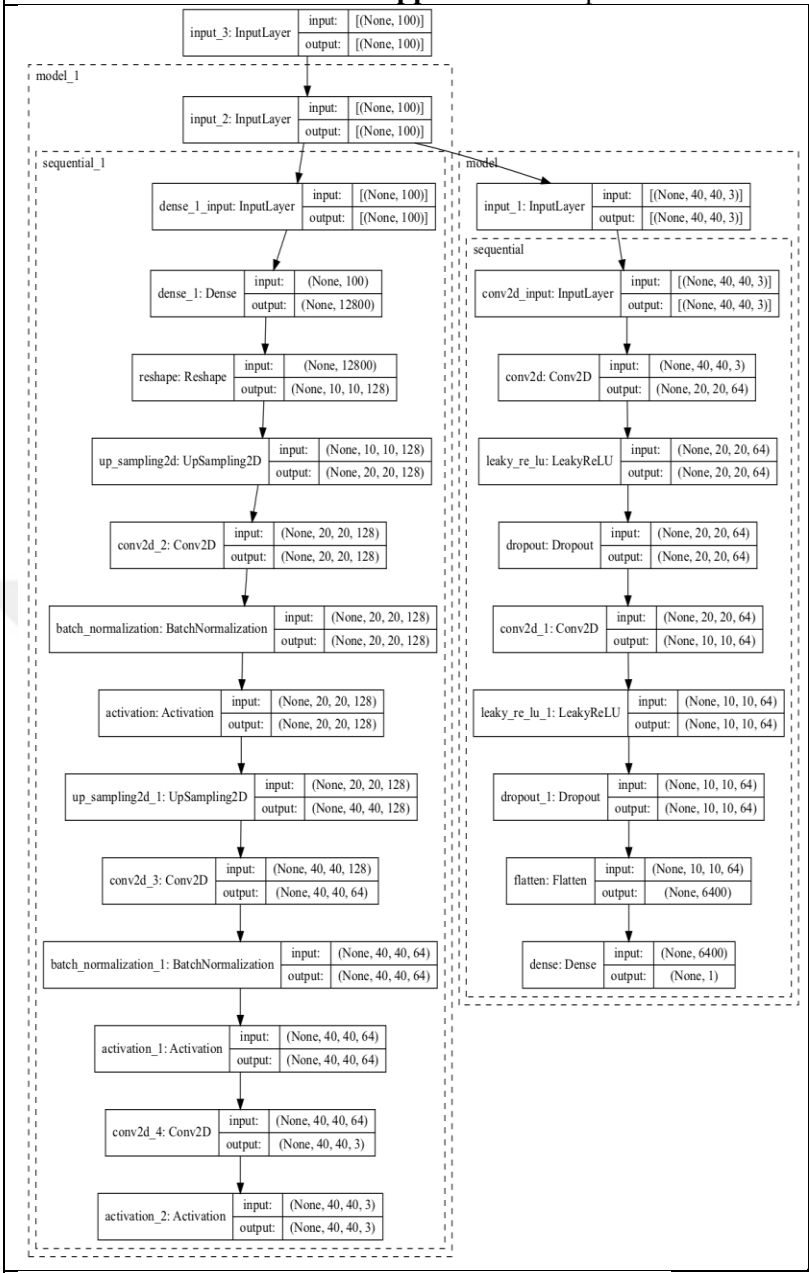
Appendix N - Experiment 14

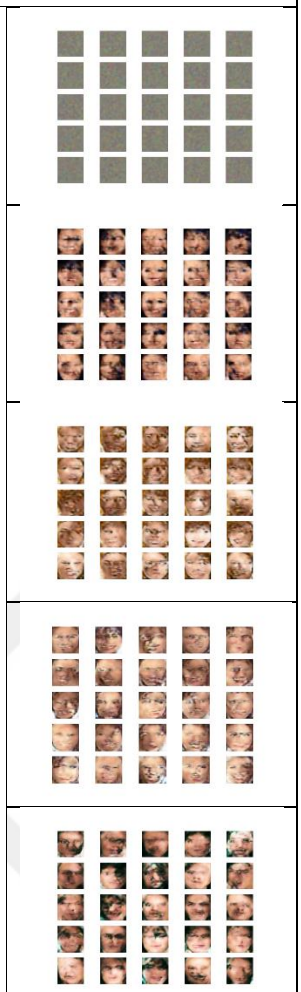


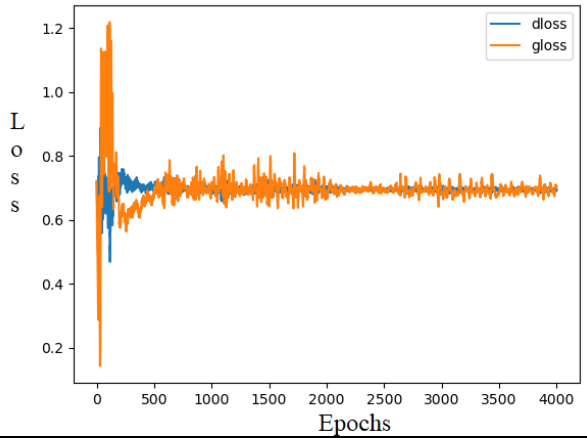




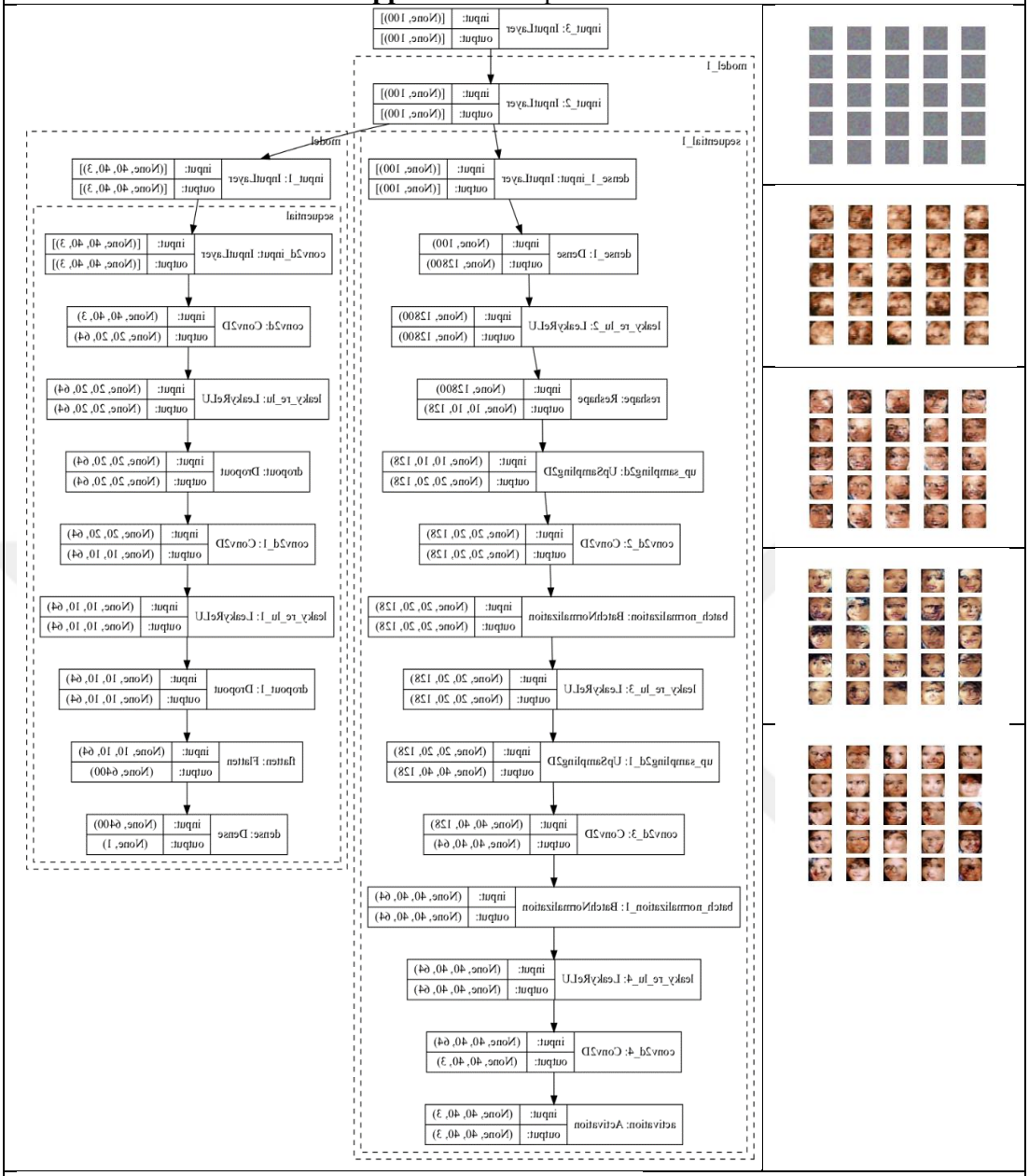
Appendix O - Experiment 15

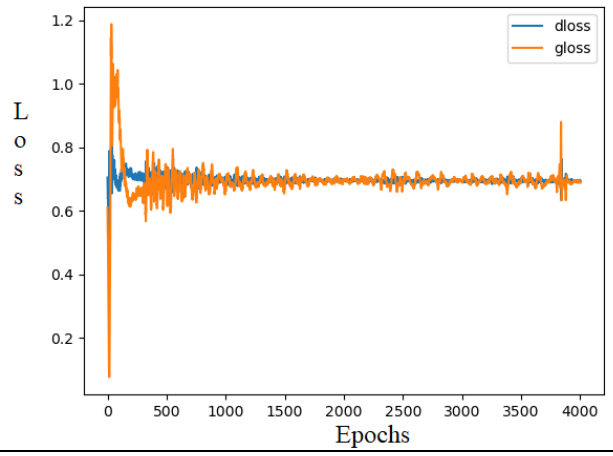




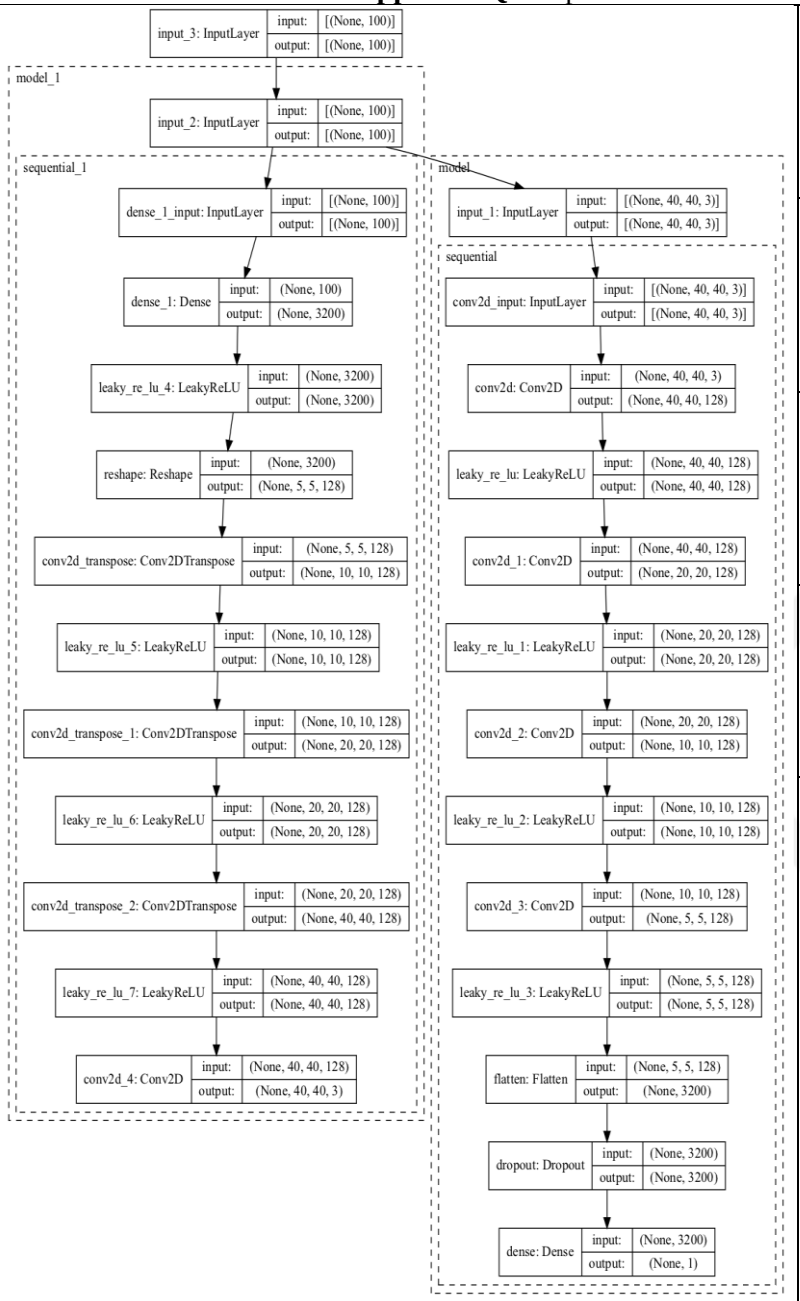


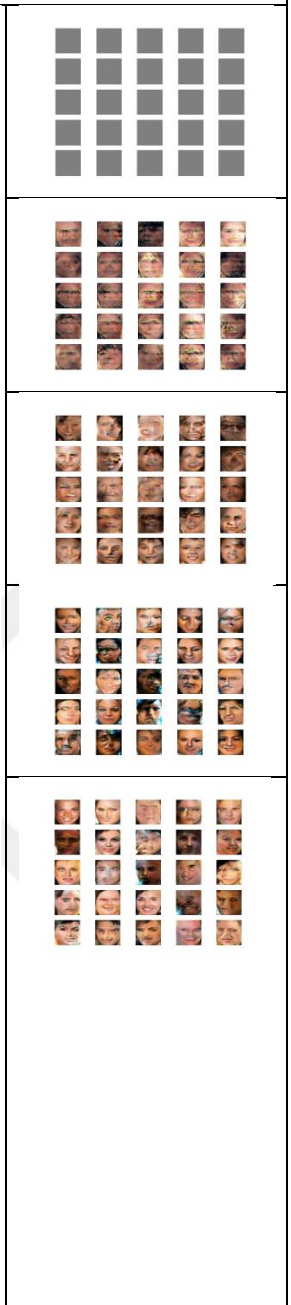
Appendix P - Experiment 16

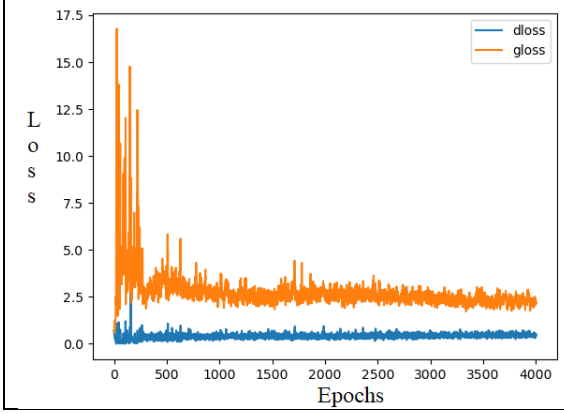




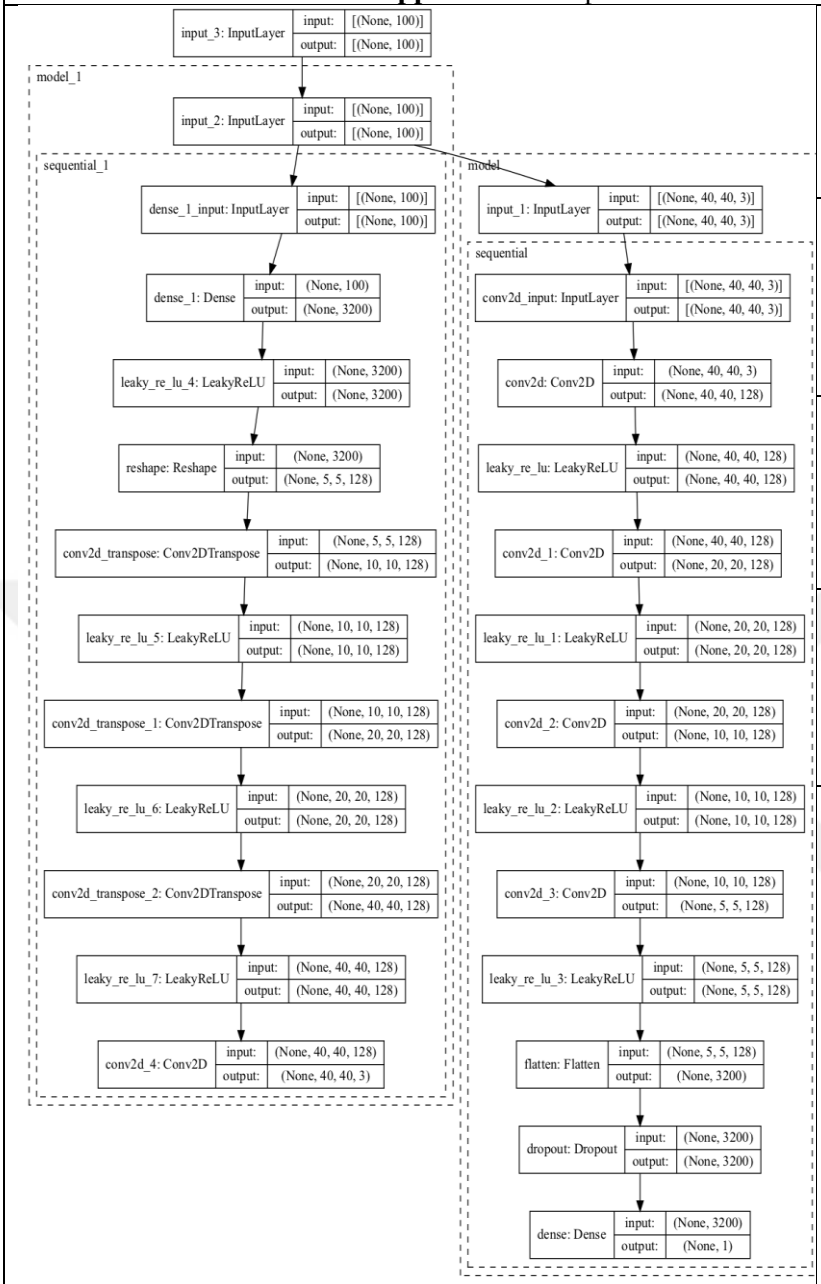
Appendix Q - Experiment 17

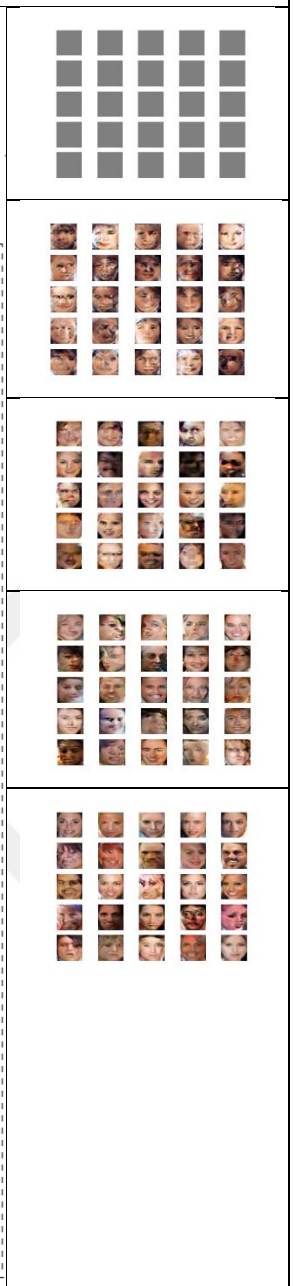


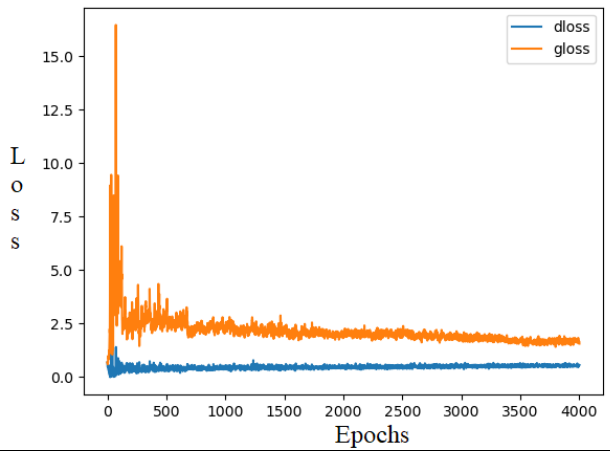




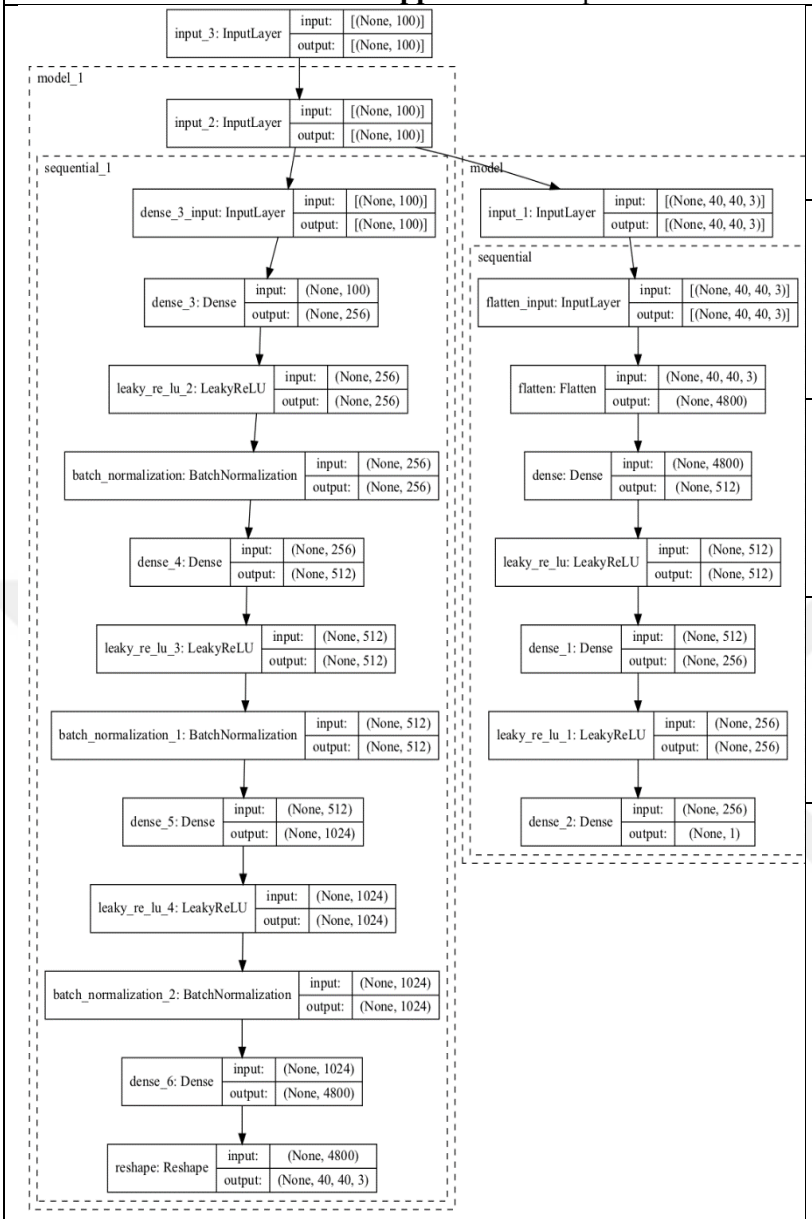
Appendix R - Experiment 18

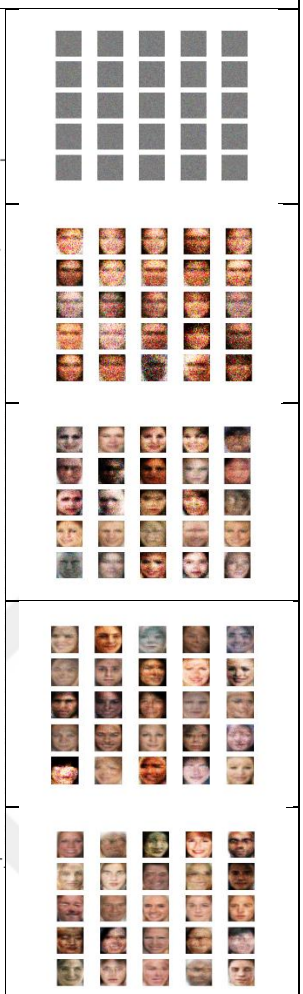


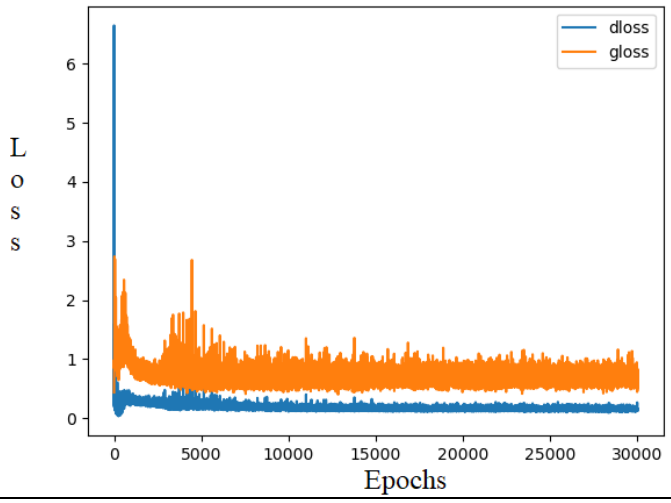




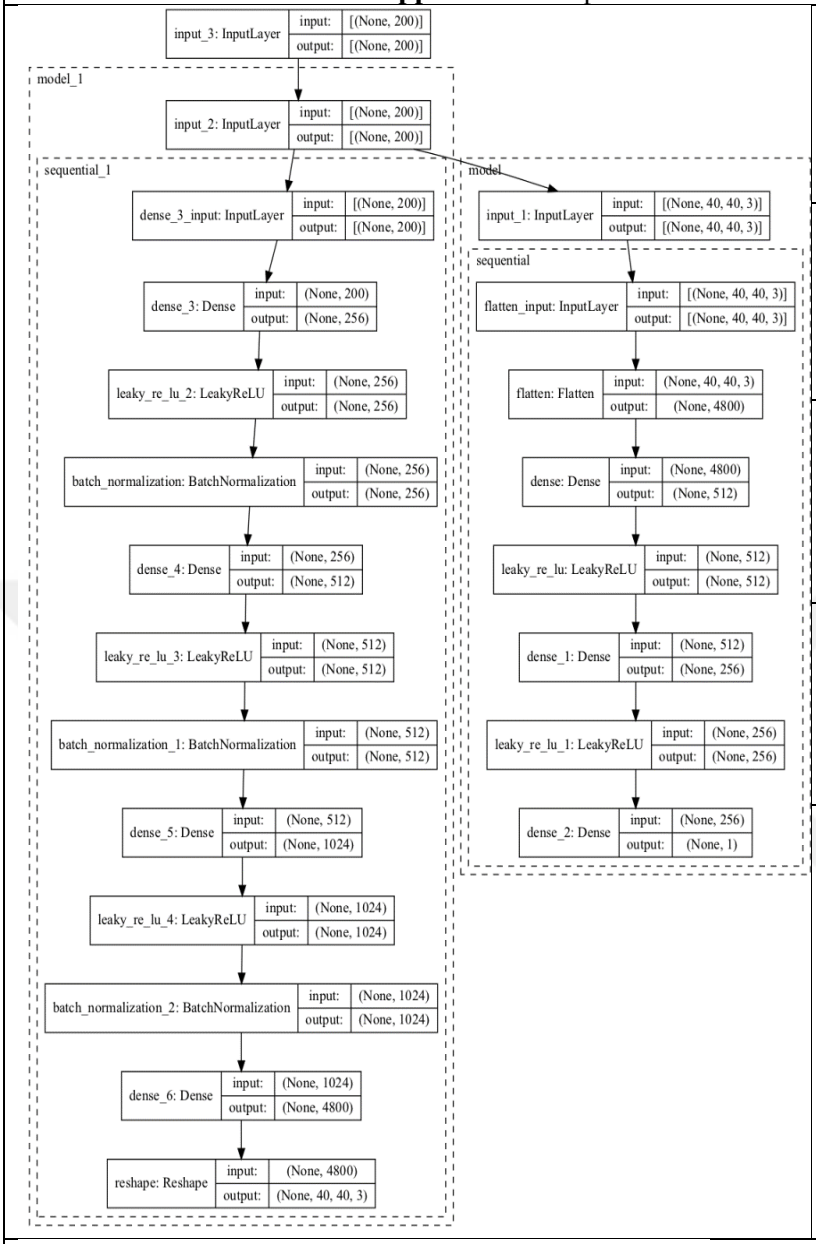
Appendix S - Experiment 19

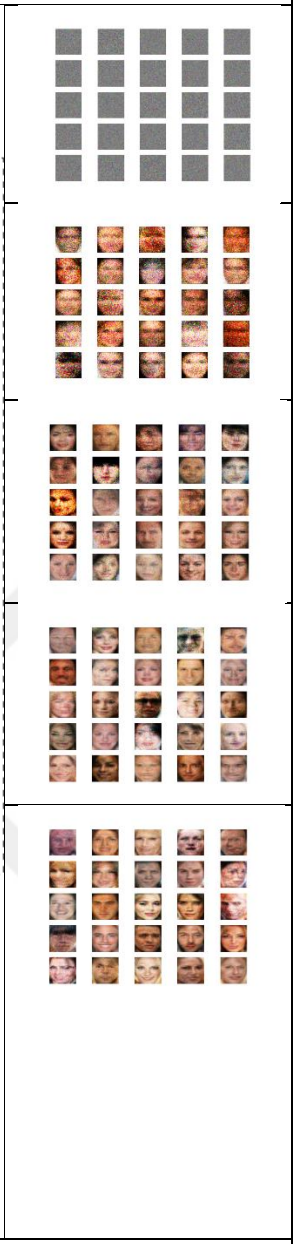


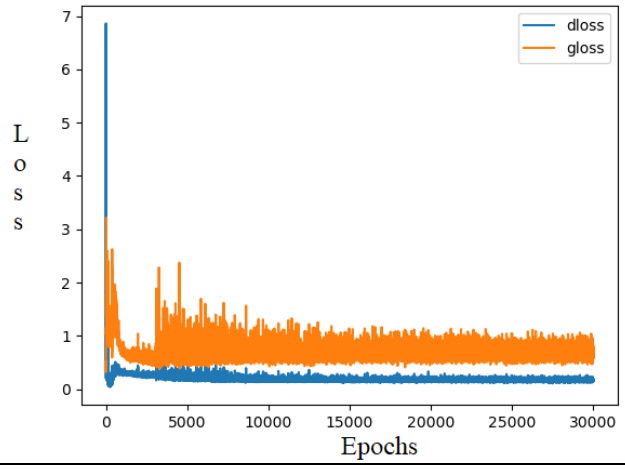




Appendix T - Experiment 20







REFERENCES

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). *Generative Adversarial Networks*. http://arxiv.org/abs/1406.2661

[2] Chen, H. (2021). Challenges and Corresponding Solutions of Generative Adversarial Networks (GANs): A Survey Study. *Journal of Physics: Conference Series*, *1827*(1). https://doi.org/10.1088/1742-6596/1827/1/012066

[3] Bhatnagar, S., Cotton, T., Brundage, M., Avin, S., Clark, J., Toner, H., Eckersley, P., Garfinkel, B., Dafoe, A., Scharre, P., Zeitzoff, T., Filar, B., Anderson, H., Roff, H., Allen, G. C., Steinhardt, J., Flynn, C., Héigeartaigh, S. Ó., Beard, S., … Amodei, D. (2018). *The Malicious Use of Artificial Intelligence: Forecasting, Prevention, and Mitigation Authors are listed in order of contribution Design Direction*.

[4] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. http://arxiv.org/abs/1511.06434

[5] Arjovsky, M., Chintala, S., & Bottou, L. (2017, July). Wasserstein generative adversarial networks. In *International conference on machine learning* (pp. 214-223). PMLR.

[6] Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., & Paul Smolley, S. (2017). Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2794-2802).

[7] Azeraf, E., Monfrini, E., & Pieczynski, W. (2020). Using the Naive Bayes as a discriminative classifier. http://arxiv.org/abs/2012.13572.

[8] Kingma, D. P., & Welling, M. (2013). *Auto-Encoding Variational Bayes*. http://arxiv.org/abs/1312.6114

[9] Akaike, H. Fitting autoregressive models for prediction. *Ann Inst Stat Math* **21,** 243–247 (1969). https://doi.org/10.1007/BF02532251

[10] Wolterink, J. M., Kamnitsas, K., Ledig, C., & Išgum, I. (2019). Deep learning: Generative adversarial networks and adversarial methods. In Handbook of Medical Image Computing and Computer Assisted Intervention (pp. 547-574). (The Elsevier

and MICCAI Society book series). Elsevier. https://doi.org/10.1016/B978-0-12-816176-0.00028-4

[11] Oord, A. van den, Kalchbrenner, N., & Kavukcuoglu, K. (2016). *Pixel Recurrent Neural Networks*. http://arxiv.org/abs/1601.06759

[12] Kevin P. Murphy, (2012), Machine Learning: A Probabilistic Perspective, 2012 ISBN 978-0-262-01802-9, p 2.

[13] Kotsiantis, S. B. (2007). Supervised machine learning: A review of classification techniques. *Informatica, 31*, 249–268.

[14] Hofmann, T. (2001). Unsupervised learning by probabilistic latent semantic analysis. *Machine Learning, 42*, 177–196.

[15] Bishop, C. M., (2006). *Pattern Recognition and Machine Learning*, p *43.*

[16] https://www.merriam-webster.com/dictionary/zero-sum%20game [Accessed: 21-Oct-2021].

[17] Saxena, D., & Cao, J. (2021). Generative Adversarial Networks (GANs) Challenges, Solutions, and Future Directions. *ACM Computing Surveys (CSUR), 54*(3), 1-42.

[18] Metz, L., Poole, B., Pfau, D., & Sohl-Dickstein, J. (2016). *Unrolled Generative Adversarial Networks*. http://arxiv.org/abs/1611.02163

[19] Karras, T., Aila, T., Laine, S., & Lehtinen, J. (2017). Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*.

[20] Wang, C., Xu, C., Yao, X., & Tao, D. (2018). *Evolutionary Generative Adversarial Networks*. http://arxiv.org/abs/1803.00657

[21] Gui, J., Sun, Z., Wen, Y., Tao, D., & Ye, J. (2020). A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications. http://arxiv.org/abs/2001.06937.

[22] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). *Improved Techniques for Training GANs*. http://arxiv.org/abs/1606.03498.

[23] Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. http://arxiv.org/abs/1502.03167

[24] https://www.kaggle.com/jessicali9530/celeba-dataset, [Accessed: 12-Sep-2021].

[25] Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, *23*(10), 1499-1503.

[26] Borji, A. (2018). Pros and Cons of GAN Evaluation Measures. http://arxiv.org/abs/1802.03446

[27] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. http://arxiv.org/abs/1706.08500.

[28] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2020). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139-144.