МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Факультет аеронавігації, електроніки та телекомунікацій
Кафедра авіаційних комп'ютерно-інтегрованих комплексів

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ Віктор СИНЄГЛАЗОВ
"___"_____2025 р.

КВАЛІФІКАЦІЙНА РОБОТА (ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ

"БАКАЛАВР"

Спеціальність 151 «Автоматизація, комп'ютерно-інтегровані технології»
Освітньо-професійна програма «Інформаційні технології та інженерія авіаційних комп'ютерних систем»

Тема: Генеративно змагальні мережі. Структурно-параметричний синтез генеративної змагальної мережі Бігана

Виконавець: здобувач вищої освіти Заїка Анастасія Андріївна
Керівник: професор, доктор технічних наук Синєглазов Віктор Михайлович

Нормоконтролер: _____ Філяшкін М.К.
(підпис)

Київ – 2025

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
STATE UNIVERSITY "KYIV AVIATION INSTITUTE"
Faculty of Aeronautics, Electronics and Telecommunications
Department of Aviation Computer Integrated Systems

ADMISSION TO PROTECTION
Head of the Graduate Department
_____ Viktor SYNEGLAZOV
"___"_____2025 p.

QUALIFICATION WORK (EXPLANATORY NOTE)

EDUCATIONAL DEGREE GRADUATE

" BACHELOR "

Specialty 151 «Automation, computer-integrated technologies"
Educational and professional program " Information Technology and Aviation Computer Systems Engineering»

Тема: Generative adversarial networks. Structural-parametric synthesis of the Bigan generative adversarial network

Student: higher education student Zaika Anastasia Andreevna
Head: Professor, Doctor of Technical Sciences Syneglazov Viktor Mikhailovich

Norm controller: _____ Filyashkin M.K.
(signature)

Kyiv – 2025

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

Faculty of Aeronautics, Electronics and Telecommunications

Department of Aviation Computer Integrated Systems

Educational degree graduate: bachelor

Specialty 151 «Automation, computer-integrated technologies»

Educational and professional program «Information Technology and Aviation Computer Systems Engineering»

ADMISSION TO PROTECTION

Head of the Graduate Department

_____ Viktor SYNEGLAZOV

“ ____ ” _____2025 y.

TASK

for the completion of the student's qualifying work

Zaika Anastasiia Andriivna

1. Topic of work: «Generative adversarial networks. Structural-parametric synthesis of the Bigan generative network»

2. Deadline for completion of work: from 22.05.2025 y. for 14.06.2025 y.

3. Initial data for work: -

4. Contents of the explanatory note (list of issues to be developed): -

5. List of mandatory graphic material: -

6. Calendar plan-graphic:

№ п/п	Task	Term	Done
1.	Develop detailed content for the sections of the qualification work	19.05.25-20.05.2025	Done
2.	Introduction	21.05.2025	Done
3.	Section 1. Generative-competitive neural networks	22.05.25-25.05.2025	Done
4.	Section 2. Generative-competitive boundary equilibrium networks and their features	26.05.2025-31.05.2025	Done
5.	Section 3. Features of the BEGAN neural network	01.06.2025-05.06.2025	Done

6.	Section 4. Results and their analysis	06.06.2025-08.06.2025	Done
7.	Elimination of deficiencies and defense of qualification work	16.06.2025-08.06.2025	Done

7. Date of issue of the task _“05" May 2025 y._____

Head of qualification work: _____Viktor SYNEGLAZOV

The task was accepted for execution by:_____Anastasiia ZAIKA

____ ” _____2025 y.

Abstract

This bachelor's thesis is devoted to the construction of an intelligent image generation system based on the use of generative-adversarial networks of marginal equilibrium. The paper analyzes scientific works devoted to the development and application of generative-adversarial neural networks, The advantages of generative-adversarial networks of marginal equilibrium are substantiated.

The topology of the components of the generative-adversarial neural network, namely the generator and the discriminator, is developed, expressions for the metrics corresponding to the generator and the discriminator are given. A machine learning algorithm for generative-adversarial networks of marginal equilibrium is developed. The choice of the training sample is justified - the FashionMNIST dataset is a standard dataset for computer vision tasks. A computer program is developed and positive results are obtained.

Реферат

Ця бакалаврська дисертація присвячена побудові інтелектуальної системи генерації зображень на основі використання генеративно-змагальних мереж граничної рівноваги. У статті проаналізовано наукові праці, присвячені розробці та застосуванню генеративно-змагальних нейронних мереж. Обґрунтовано переваги генеративно-змагальних мереж граничної рівноваги.

Розроблено топологію компонентів генеративно-змагальної нейронної мережі, а саме генератора та дискримінатора, наведено вирази для метрик, що відповідають генератору та дискримінатору. Розроблено алгоритм машинного навчання для генеративно-змагальних мереж граничної рівноваги. Обґрунтовано вибір навчальної вибірки - набір даних FashionMNIST є стандартним набором даних для задач комп'ютерного зору. Розроблено комп'ютерну програму та отримано позитивні результати.

CONTENT

INTRODUCTION

For a long time, one of the main problems of machine learning algorithms with the teacher (English "Supervised Learning") is the data problem. In particular, training neural networks often requires the sampling of millions of elements. Considering as an example of learning to recognize images, millions of images should be of sufficient quality and should be noted, which requires manual labor of a large number of people. Thus indirect costs on

The training of the neural network recognition is huge. The way out of the situation can be an algorithm capable of generating such samples without human involvement. Over the past few years in this area, there has been obvious progress due to its appearance

generative competitive networks (GAN). However, such neural networks are extremely difficult to develop, and the question of synthesis of images in high resolution with them is still open. Recent studies in GAN are aimed at implementing methods that would bring greater stability in the learning process of this kind of networks: numerous types of data normalization, such as spectral normalization, pixel normalization and many others; Various errors, such as Wassertestein's distance, the distance of the library, and the like.

SECTION 1
GENERATIVE AND CONSISTENT NEURAL NETWORKS

1.1. Generatively competitive networks

Generative Adversarial Networks (GANs) are deep learning models that date back to 2014 (Goodfellow et al., 2014). These models belong to a set of generative models, a branch of unsupervised learning algorithms responsible for representing how data is generated. GAN models are trained to create synthetic data similar to a given real data set. Their operation consists of two types of neural networks that compete through opposing goals (hence the term "adversarial"). This adversarial training, similar to reinforcement learning, allows the network to learn to generate data that resembles real data distributions.

There are two networks that make up a GAN:

Generator (G): This network is responsible for generating data that closely resembles the real data set it was trained on. It takes a vector of random noise values (z) drawn from a simple prior distribution (pz) as input and processes it through the network's internal values until output data (G(z)) is generated, with the goal that these new data points closely match the real data distribution.

Discriminator (D): This model aims to classify between real samples (x∼pdata(x)), i.e., data belonging to the real data set, and generated samples (G(z)∼pg(G(z))), i.e., data artificially created by the generator. Its inputs are either real or fake data, and its output is classification probabilities for both classes through a logistic function.

This general structure is called Vanilla GAN, as it represents the basic version from which more complex architectures emerged but still respects the two fundamental blocks of GAN. A visual representation of this basic structure is shown in Figure 1.17.

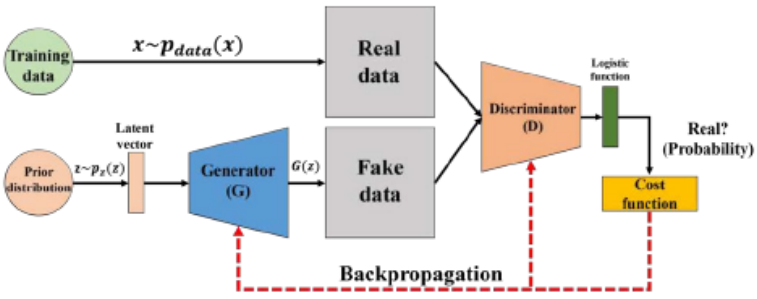

Fig.1.1. Training of Vanilla GAN

GAN training is described as a zero -sum game (also called minimax) between two players with opposite goals; It is a generator and Discriminator. Taking a vector of random noise taken from previous.

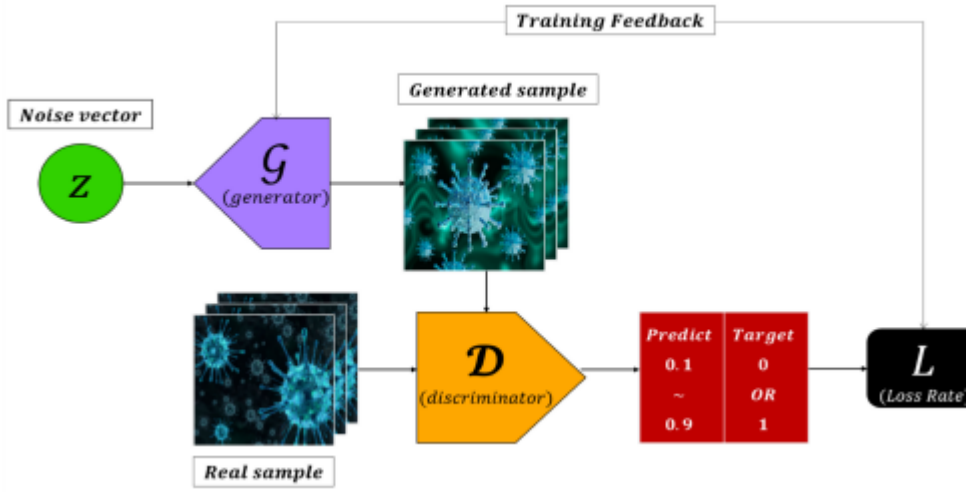

Fig.1.2. GAN (Generative Adversarial Networks) Training Algorithm

First, a real sample is randomly selected from the training set. Then the generator's output is combined into the training set, and the discriminator is trained. The target for a real image is "1," and for a generated image is "0." The real image outputs a value close to "1," while the synthetic image outputs a value close to "0."

Training the generator is difficult because a real image does not correspond to any single point in the latent space. When the generator's output is fed into the discriminator, it outputs the probability that the input is real. This probability is the GAN output. The input is a randomly generated vector from the d-dimensional latent space, and the output is "1" for training the GAN by generating a training batch. The output should be set to "1" to generate a real sample.

The loss function is binary cross-entropy between the discriminator output and the target "1." The target is a binary value, and the model uses a single output unit with a sigmoid activation function [12]. During GAN training, the discriminator's weights must be frozen so that only the generator's weights are updated. Otherwise, the discriminator would adjust to classify the generated image as real.

1.2. Applications of GAN neural networks – real inspiring examples

GANs have a range of different applications, most of which are related to creating images and image components. GANs are commonly used in tasks where image data is missing or limited in quantity, serving as a method to generate the necessary data. Let's look at some typical use cases of GANs.

Generating new examples for datasets

GANs can be used to create new examples for simple image datasets. If you have only a few training examples and need more, GANs can generate new training data for an image classifier by producing new training examples with various orientations and angles.

Creating unique human faces

Fig 1.3. The woman in this photo does not exist. The image was created by StyleGAN. Photo: Owlsmcgee via Wikimedia Commons, public domain.

After sufficient training, GANs can be used to create extremely realistic images of human faces. These generated images can be used to train facial recognition systems.

Image-to-image translation: GANs excel at image translation. They can be used to colorize black-and-white images, translate sketches or drawings into photographic images, or convert daytime scenes into nighttime ones.

Text-to-image translation: Text-to-image translation is possible with GANs. Given a text description of an image along with a corresponding example image, a GAN can learn to generate a new image based on the provided description.

Image editing and restoration: GANs can be used to edit existing photos. They can remove elements like rain or snow from an image, and also restore old, damaged, or corrupted images.

Super-resolution : Super-resolution is the process of taking a low-resolution image and increasing its pixel count to improve image quality. GANs can be trained to create higher-resolution versions of low-res images.

Generative adversarial networks (GANs) are applied in many fields: content and data creation — for example, generating images for online stores, avatars for games, automatically generated music videos based on a track's beat, or even virtual hosts for TV shows. Thanks to GANs and generative models, data synthesis emerges, which is then used to train other systems; automatic editing — this approach is already used in modern smartphones and some software. It allows changing faces in photos, reducing wrinkles, altering hairstyles, turning day into night, aging images, and more. Stunningly realistic "celebrity" images (actually of nonexistent people) can be created using NVIDIA's PG GAN. This network can also generate images of any other categories.

The model Everybody Dance Now, developed by a group of researchers from UC Berkeley, presents a simple method based on generative neural networks for motion transfer called "do as I do." Given: an input video of a dancing person. Result: making the image of another person dance. This is called "motion transfer."

Style transfer from one image to another allows neural networks to perform impressive tasks like "turning a horse into a zebra" or generating "anime portraits" from photos. This picture shows how different types of GANs handle these tasks.

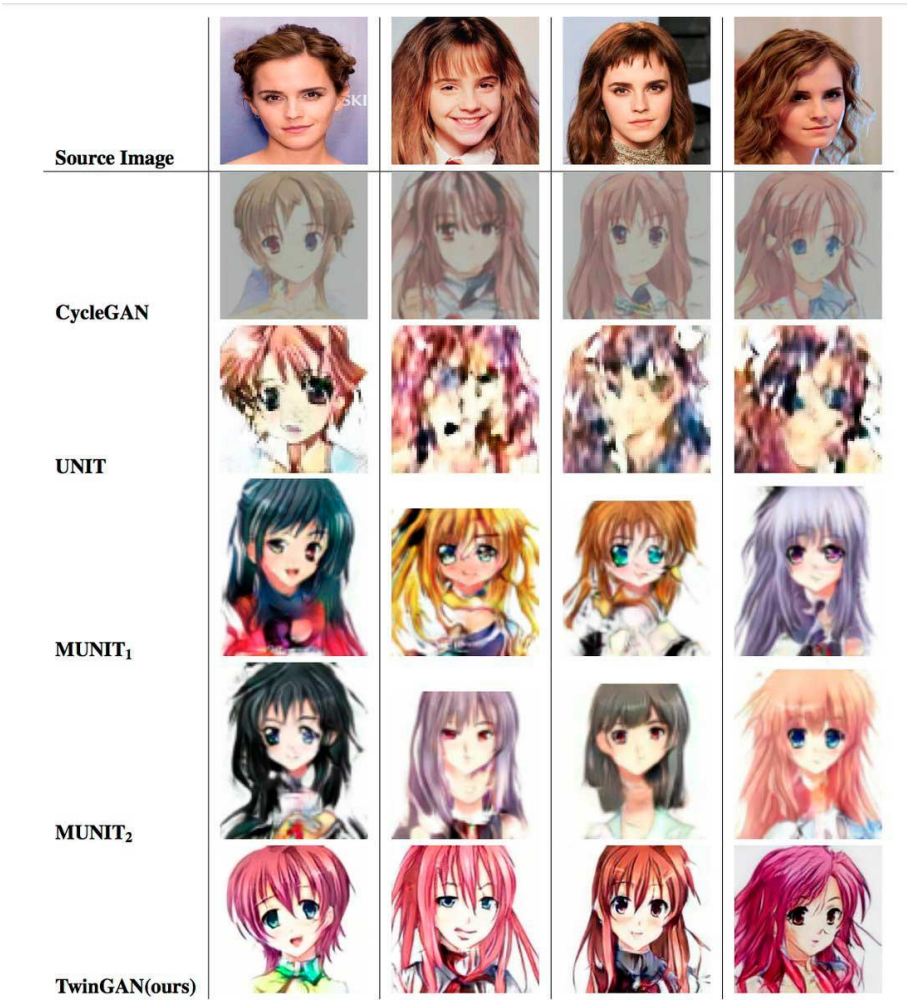

Fig.1.4. Different types of GANs generating photos in different styles.

Changing a person's emotions, age, or facial expression — all this can be achieved by properly training and programming GAN neural networks. In practice, it works like this: the model takes an input photo and a specified emotion to be displayed in the output image.

Fig1.5. Model takes photo with different emotion.

Neural networks GAN are also used to generate realistic videos of urban environments. For example, in filmmaking, games, and virtual reality.

Sketch and contour-to-photorealistic image translation using GAN works as follows: you draw a face, a bag, or, for example, a cat by hand, and receive a photorealistic image as output. GANs have a variety of applications, most of which are related to creating images and image components. GANs are typically used in tasks where image data is missing or

limited in quantity, serving as a method to generate the necessary data. Let's consider some typical use cases of GANs. Generating new examples for datasets.

GANs can be used to create new examples for simple image datasets. If you only have a few training samples and need more, GANs can generate new training data for image classifiers by producing new examples at different orientations and angles.

Creating unique human faces :

- **Graphics:** Creating realistic images, designs, and art concepts.
- **Video:** Generating new video frames, restoring old videos.
- **Medicine:** Generating 3D images of organs, aiding diagnosis.
- **Game Development:** Creating textures, characters, and environments without human involvement.

Face frontalization is the process of synthesizing frontal views of faces taking into account their angular poses. We implement a generative adversarial network (GAN) with spherical linear interpolation (Slerp) for frontalizing casual (natural) face images. Our particular focus is on generating approximate frontal faces from side views captured by surveillance cameras. Specifically, this work presents a comprehensive study on implementing an autoencoder-based boundary equilibrium GAN (BEGAN) for generating frontal faces using interpolation of side and mirrored face views. To improve the quality of the interpolated output, we implement BEGAN with Slerp. This approach can provide promising results along with faster and more stable training for the model. The BEGAN model also has a balanced generator-discriminator combination that prevents mode collapse while ensuring global convergence. Such an approximate face generation model is expected to replace composite face images used in surveillance and crime solving.

For various security and surveillance applications, human faces are considered one of the most universally used biometric identification tools, alongside other biometric features such as fingerprints, signatures, etc. [1]. However, there is a basic assumption that faces of two different subjects cannot be identical, which may not hold true for twins due to many biological reasons. The primary interest of this work is not to solve such biological difficulties, but rather to apply modern machine learning (ML) tools to generate faces from incomplete face images [2], [3].

Accurate face identification often requires a full frontal view of the subject's face, which may not be available in certain circumstances such as surveillance footage or when the subject is dynamic and changes face orientation. To aid face identification in these cases, face frontalization has been widely used on angular face views [4], [5].

Face frontalization is the process of synthesizing a frontal view from an inclined human face pose, where one side of the face is blurry or completely unavailable. In the rapidly growing field of automatic face and gesture recognition research, face frontalization is one of the most important and discussed problems, representing fundamental interest both for human and machine face processing and recognition.

Frontal face images can be automatically generated using one of the popular deep learning approaches called Generative Adversarial Networks (GAN) [6].

1.2. Overview of Generative Adversarial Neural Networks

The emergence of the deep learning model GAN (Generative Adversarial Networks) marks an important turning point in generative modeling. GANs are more powerful in learning features and representations compared to traditional machine learning-based generative model algorithms. Currently, they are also used for generating non-visual data such as voice and natural language. Typical technologies include BERT (Bidirectional Encoder Representations from Transformers), GPT-3 (Generative Pretrained Transformer-3), and MuseNet. GAN differs from machine learning-based generative models and target functions. Training is performed by two networks: the generator and the discriminator. The generator transforms random noise into realistic images, while the discriminator distinguishes whether an input image is real or synthetic. As training progresses, the generator learns more complex synthesis methods, and the discriminator becomes a more accurate differentiator. GANs face challenges such as mode collapse, training instability, and lack of evaluation metrics, and many researchers have attempted to address these issues. For example, solutions such as one-sided label smoothing, instance normalization, and mini-batch discrimination have been proposed. The application domain has also expanded.

Deep learning models directly learn high-level features from unstructured data [1]. The true power of deep learning lies in its ability to process unstructured data. In particular,

generative modeling generates unstructured data such as new images or text; therefore, deep learning has a significant impact on the field of generative models.

Generative modeling is the next frontier of machine learning. Deep learning has been applied to generative modeling only for a few years. At the 2014 NIPS (Neural Information Processing Systems) conference, Ian Goodfellow from Google Brain introduced GAN (Generative Adversarial Networks) [2]. GAN spawned a series of algorithms and advanced this field further.

Since mid-2018, significant progress has been made in sequence modeling and image-based generative modeling. Sequence modeling has mainly been performed using transformers [3], which use attention modules. Transformers are used instead of convolutional neural networks. Examples include Google's BERT (Bidirectional Encoder Representations from Transformers), GPT-3 (Generative Pretrained Transformer-3 — a type of artificial intelligence (AI) that uses machine learning algorithms to generate natural language text. The first version of GPT, released in 2018, was a revolutionary achievement in AI and natural language processing (NLP)), for language modeling, Parallel WaveGAN for speech synthesis, and MuseNet for music composition [4–7]. GAN-based technologies have been developed such as PGGAN (Progressive Growing of GANs), SAGAN (Self-Attention GAN), BigGAN, and StyleGAN; thus, improving the state of image generation [8–11].

A generative model can synthesize images by capturing and learning the statistical distribution of training data. In any case, the network weights are trained via backpropagation [13]. In GAN literature, multidimensional vectors are considered, and vectors in the probability space are italicized. Hidden vectors are usually denoted as z. In signal processing, vectors are represented by lowercase symbols to emphasize the multidimensional nature of the variables. Therefore, $p_{data}(x)$ data

(x) is the probability density function for a random vector x from $R^{|x|}$. The probability that a continuous random variable falls within a given interval is called the probability density, expressed as the probability density function. $p_G(x)$ represents the distribution of vectors generated by G. θ_D and θ_G are the weights extracted from G and D respectivel.

As with all deep learning algorithms, a target function is required for training. Currently, the loss function, objective function, and cost function are terms used interchangeably. More precisely, the cost function is the sum of loss functions over all training data, while the objective function is the target function optimized in a more general sense. However, in practice, these three terms are often used synonymously. When two objective functions are continuously updated, the objectives of G and D are represented as $J_G(\theta_G; \theta_D)$ и $J_D(\theta_D; \theta_G)$, to remind that the parameter sets θ_G and θ_D depend on each other. When the multidimensional gradient is updated, the gradient operator for the weights of G is expressed as $\nabla_{\theta G}$, and for D — as $\nabla_{\theta D}$. ∇ — is the differential operator where each component is represented as a formal vector relative to Cartesian coordinates *x, y, z*. In the case of an expected gradient, it is denoted as $E\nabla$.

Expected gradient refers to an approach in which gradient descent is used to train the parameter matrix W by optimizing the expected logarithmic likelihood function as the target. It involves computing the gradient as a sum over examples, where each example contributes a term representing the difference between the expectation of the product of the product of the latent variable and its transposed with the expectation of the product of the input and latent variable transposed. This approach allows the model to reconstruct the data as it ascends the gradient.

The target function is very important because it is related to GAN problems. If we use a target function that is not suitable for the problem we are trying to solve, GAN can get out of control during training. A typical example is when the losses are vibrating. The discriminator and oscillator loss functions do not show a steady state over a long period of time, and the GAN is highly vibrated.

As another example, although the image quality improves over time, the oscillator loss function may increase.

This is due to the lack of relationship between oscillator loss function and image quality. This lack of relationship makes it difficult to observe the learning process of the GAN. The target functions presented here are considered to be suitable methods for training complex GANs.

1.2.1. Wasserstein Generative Adversarial Networks WGAN (Wasserstein Generative Adversarial Networks)

The WGAN loss function is significant because it correlates the convergence of the generator and the sample quality. In the WGAN generative adversarial network, the Wasserstein loss is introduced into the loss function, which correlates the sample quality with the convergence of the generator [15]. The Wasserstein loss improved the stability in the optimization process. First, yi = 1, yi = −1 was used instead of yi = 1, yi = 0 for the binary cross-entropy loss function. In addition, the sigmoid activation function was removed from the last layer of the discriminator. Therefore, the prediction of pi is not limited to the range [0,1], but can be any number in the range [−∞, ∞]. For this reason, the WGAN discriminator is called a critic.

The Wasserstein loss function converges by training the discriminator so that the generator is updated correctly. This is different from the original GAN, where it is important to ensure that the discriminator does not become too strong. The Wasserstein loss can balance the training of the discriminator and the generator. WGAN trains the discriminator multiple times during the generator update for convergence. In general, for a single generator update, the discriminator is updated five times. Since WGAN prunes the weights from the critics, the learning rate is significantly reduced. If the gradient is incorrect, the generator cannot learn the direction of the weight update. For this reason, another method for the Lipschitz constraint, WGAN-GP (Gradient Penalty) [16], was introduced.

1.2.2. Generative Adversarial Wasserstein Network with Gradient Penalty WGAN-GP (Wasserstein GAN-Gradient Penalty)

WGAN-GP solves problems such as mode collapse and unstable learning and makes the GAN training predictable and robust. WGAN-GP includes a gradient penalty term in the critic (discriminator) loss function [17]. The weights of the critics are not pruned. Moreover, the batch normalization layer should not be used for critics. Batch normalization creates correlation between images in the same batch, so the gradient penalty loss has less effect [18]. WGAN-GP proposes another way to enforce the Lipschitz constraint for critics: adding a term to the loss function that penalizes when the gradient norm of the critic deviates significantly from “1”. As a result, the training process was significantly stabilized. The

gradient penalty loss is the squared difference between the gradient norm of the output and one. This model naturally finds the weights that minimize the gradient penalty term. In other words, the model is designed to satisfy the Lipschitz constraint. It is difficult to compute the gradient everywhere during the training process. WGAN-GP computes the gradient only at some point. To avoid being biased to one side, the real image-synthetic image pair is connected as shown in Figure 3, and images interpolated using randomly selected points along a straight line are used.

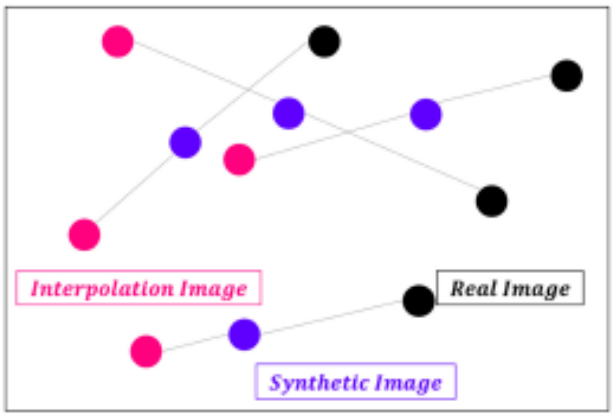

Fig.1.6. Interpolation between images

1.2.3. Generative adversarial networks with self-attention (Self-Attention Generative Adversarial Networks (SAGAN))

Attention is an algorithm used in sequence models such as transformers [3]. SAGAN is a model that applies the attention algorithm to GANs [9]. The self-attention algorithm is shown in Figure 4.

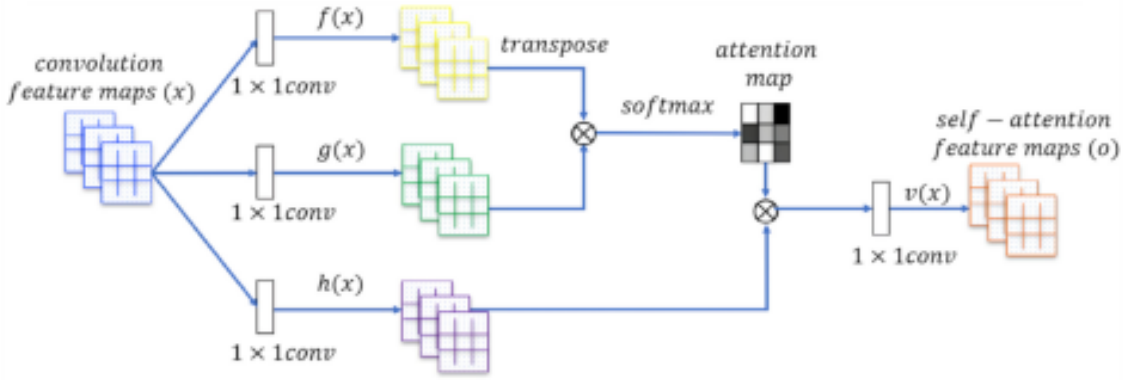

Fig.1.7. Self-attention algorithm SAGAN (self-attention generative adversarial networks)

In GAN without attention, the convolution feature map can only handle local information. To connect the pixel information on one side of the image to the other side, the channel must be increased to several convolution layers and the dimensionality of the image

space must be reduced. On the other hand, this process loses the precise location information instead of capturing high-level features. Thus, it is inefficient for the model to learn the relationship between distant pixels. SAGAN solved the above problem by applying the attention algorithm to GAN.

1.2.4. Deep Convolutional Generative Adversarial Networks (Deep Convolutional Generative Adversarial Networks (DCGAN)

In recent years, supervised learning using CNN (convolutional neural networks) has been widely applied in the field of computer vision [19]. In contrast, unsupervised learning using CNN has not received much attention.

Radford introduced DCGAN (Deep Convolutional Generative Adversarial Networks) in 2016 [20]. The internals of DCGAN are composed entirely of convolutional layers. The pooling layer of the discriminator is replaced by a convolutional layer, and the pooling layer of the generator is replaced by a transposed convolution (A transposed convolutional layer is an upsampling layer that generates an output feature map larger than the input feature map. It is similar to a deconvolutional layer. A deconvolutional layer inverts the layer into a standard convolutional layer.). After the convolutional layer, the fully connected classification layer is removed. Batch normalization is performed after each convolutional layer to advance the gradient flow.

The basic algorithm of DCGAN is the same as traditional GAN. There is a generator that generates 100-dimensional noise, and the noise is mapped and transformed through a convolutional layer. Figure 5 shows the structure of the DCGAN generator.

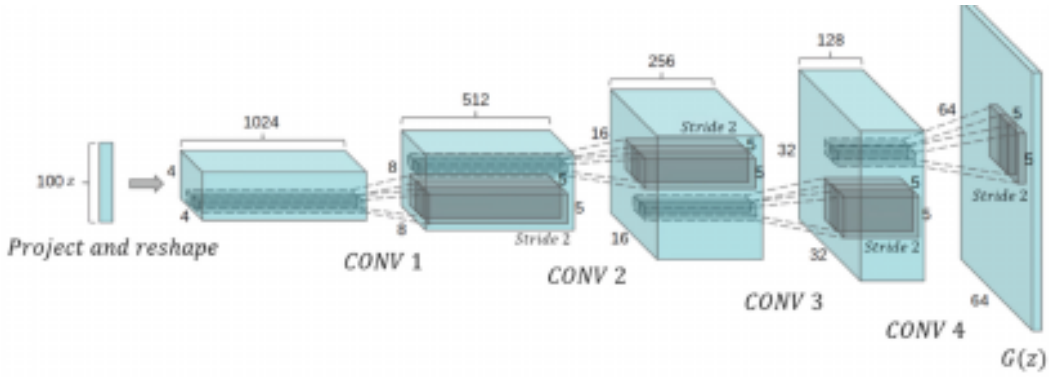

Fig.1.8. Structure of DCGAN (Deep Convolutional Generative Adversarial Networks) generator [19].

The generator serves the same purpose as the VAE (Variational Autoencoder) decoder [21]. The input to the generator is a vector drawn from a multivariate standard normal distribution. The generator freezes and trains the discriminator, and this process is repeated for thousands of epochs. The output image size is the same as the input image.

DCGAN is optimal when using the Adam (Adaptive Moment estimate) optimizer and a learning rate of "0.002" [22]. DCGAN learns different representations from objects to scenes. In addition, the learned features can be used for other tasks to be used as a general representation of the image. Raymond used DCGAN to fill in unwanted or missing parts of an image [23]. DCGAN is the first to use CNN as both the generator and discriminator of a GAN to improve performance. Nowadays, all GAN structures include a convolutional layer. Thus, GAN already implies the meaning of "DC".

1.2.5. Boundary Equilibrium Generative Adversarial Networks (BEGAN)

BEGAN (Boundary Equilibrium Generative Adversarial Networks) attracted attention because its discriminator is a CAE (Convolutional Autoencoder) and has convergence estimation properties that DCGAN does not have [24,25]. BEGAN learns the latent space of an image by maintaining and adjusting the balance between the generator and the discriminator. BEGAN uses AE (Autoencoder) as a discriminator instead of a classifier. The discriminator learns that the real image has a small reconstruction error and the image generated by the generator has a large reconstruction error. The generator learns such that the reconstruction loss of the discriminator is small. Unlike DCGAN, BEGAN is capable of convergent solution. The main problem of GAN, mode collapse, also occurs in BEGAN. Accordingly, BEGAN-CS (BEGAN with Constrained Space) was announced, which is space constrained, but it did not solve the mode collapse [26]. Sang-Wook changed the structure of the BEGAN-CS discriminator from AE to VAE to solve the mode collapse problem, and also changed the structure of the encoder and decoder [27]. The activation function was changed from ELU (Exponential Linear Unit) to LReLU (Leaky Rectified Linear Unit) [28,29]. The KLD (Kulback-Leibler Divergence) term was added to the existing discriminator loss function [30]. The BEGAN generative adversarial network will be discussed in more detail in Section 2.

1.2.6. Generative Adversarial Networks of Progressive Growth (Progressive Growing of Generative Adversarial Networks (ProGAN))

Generating high-resolution images is a big challenge. The larger the image, the easier it is for the network to make a mistake, as it needs to learn to generate more complex and delicate details.

The Progressive Network, which was built on top of the Improved Wasserstein Network, offers a smart way by gradually adding new high-resolution layers during training, which creates incredibly realistic images [6]. Each of these layers increases the resolution of the images for both the discriminator and the generator.

PGGAN is a model developed by NVIDIA to improve the speed and stability of GAN training [8]. PGGAN is a model that generates high-quality, high-resolution images by adding a new layer during the training of the generator and discriminator. PGGAN trains the generators and discriminators on low-resolution 4x4 pixel images. The added layer is not frozen, but continues learning. This algorithm was applied to the LSUN (large scale scene understanding) dataset image to obtain the results shown in Figure 8 [31]. The structural diagram of PGGAN is shown in Fig. 6.

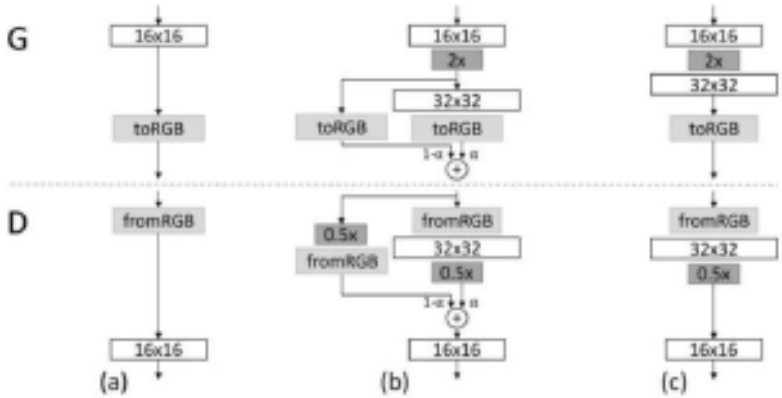

Fig.1.9. PGGAN Block Diagram.

The algorithm for constructing PGGAN can be represented as follows..

1. Train the generator and discriminator with low-resolution images.
2. At that point (for example, when they start to converge), the resolution needs to be increased. This is done using anti-aliasing. Instead of simply adding a new layer directly, it is added in small linear steps, controlled by α.

3. After completing the transition, we continue training the generator and discriminator and go to step 2 if the resolution of the images currently being generated is not the target resolution.

As a result of this progressive learning, the generated images in ProGAN are of higher quality and the training time is reduced by 5.4 on 1024x1024 images. The reason for this is that this network does not need to learn all the large-scale and small-scale representations at once. In ProGAN, the small-scale layers are learned first (i.e., the low-resolution layers converge), and then the model can focus on refining purely large-scale structures (i.e., new high-resolution layers converge).

1.2.7. Informational and Conditional GANs

The GAN model presented so far has almost no control over the generated image. Informational and conditional GANs control the generated images [32,33]. InfoGAN (Information Maximizing Generative Adversarial Networks) can control various properties of the generated image. It can apply the concept of information theory to predict the noise element for the output and convert it into a latent code responsible for control. The InfoGAN generator takes two inputs: a latent space z and a latent code c. The output of the generator is G(Z, c). The GAN is trained to maximize the mutual information between the latent code c and the generated image G(Z, c). Figure 9 shows the structure of InfoGAN.

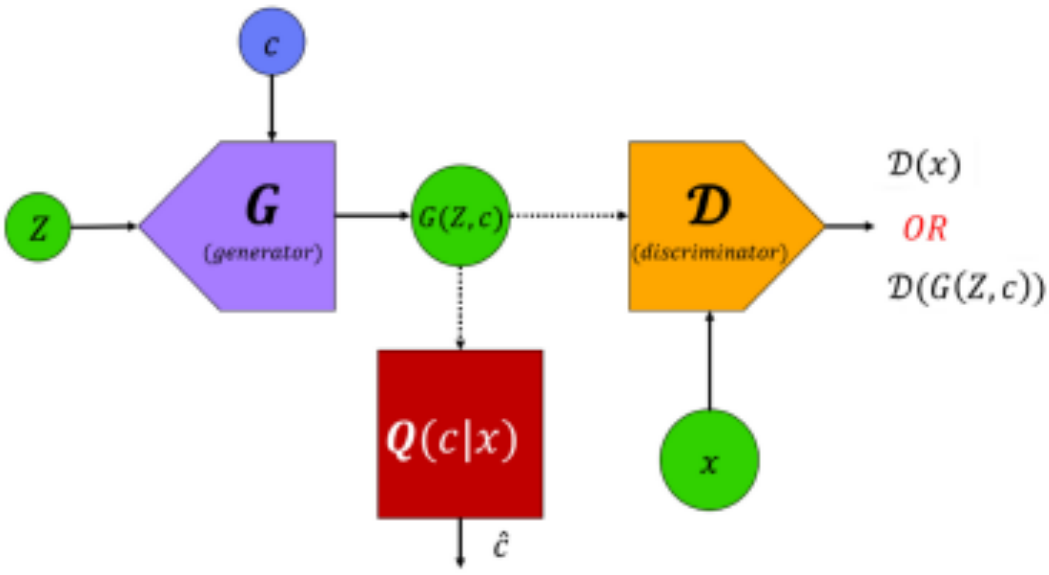

Fig 1.10. Structure and operating principle of InfoGAN (Information Maximization Generative Adversarial Networks).

The concatenated vector G(Z, c) is input as a generator. Q(c| x) is also a neural network. When combined with the generator, it forms a mapping of random noise Z and the hidden code $\hat{c}$. For InfoGAN, the training objective is to estimate c for a given X. This is done by adding a normalization term to the GAN objective function.

Figure 10 shows the structure of cGAN (Conditional Generative Adversarial Networks). cGAN performs conditional discrimination between real and synthetic images from discriminators, providing better performance than DCGAN when generating different data.

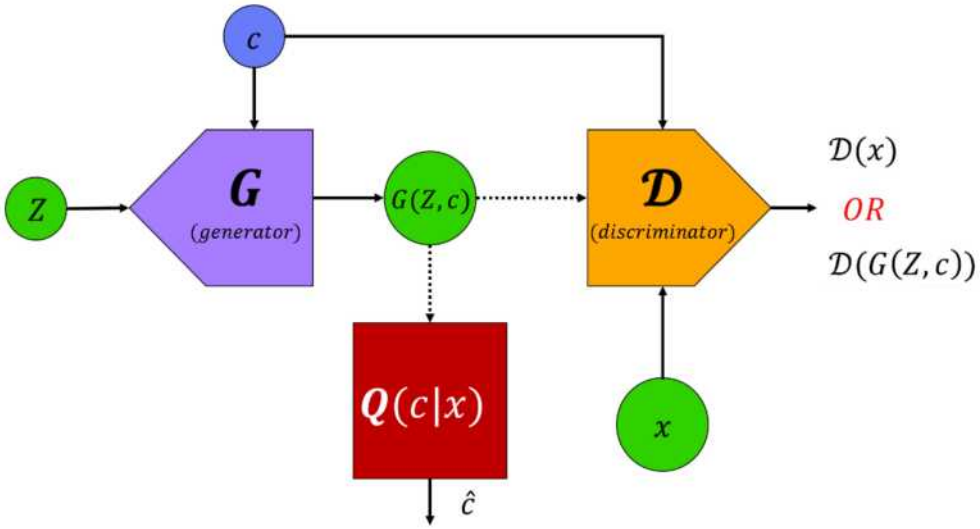

Fig. 1.11. Structure and operation principle of cGAN (conditional generative adversarial networks).

1.2.8. GAN based inference model

ALI (Adversarially Learned Inference) and BiGAN (Bidirectional Generative Adversarial Networks) are inference models where D inspects the data - latent pairs [36, 37]. G consists of an inference network, an encoder and a decoder. The input of D is a pair (z, x) to determine whether it is a real image and its encoding or a synthetic image and its latent vector. The quality of the synthetic image using ALI/BiGAN is not very good, but it can be improved with additional sampling and reconstruction distribution costs [38]. Figure 11 shows the structure of a GAN with an added inference network.

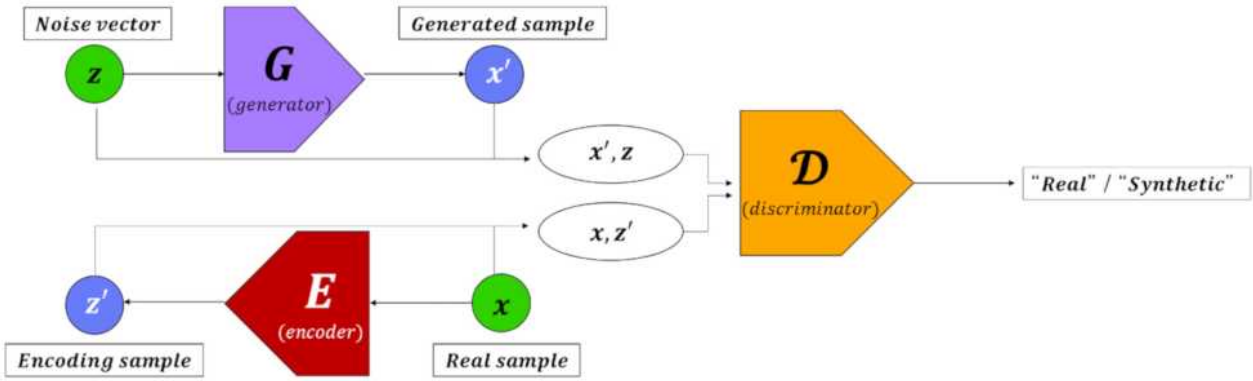

Fig 1.12. GAN structure with input network added angles.

2 SECTION
GENERATIVE-MAGICAL BOUNDARIES OF THE BOUNDARY LEVEL AND THEIR FEATURES

2.1. Topology of generative-magal boundaries

Generative-magnitude measures [7] (GAN) are a class of methods for generating a subdivision of data pmodel (x) and implementing a model for sampling from it. The GAN is driven by two functions: a generator G(z), which maps sample z into a uniform distribution of data, and a discriminator D(x), which determines which sample x belongs to the data. The generator and discriminator begin with a complete process of drawing up D and G based on the principles of the igor theory.

Generative-magnetic measures can generate even more recontextualized images, more clear ones, and lower ones that are generated by auto-encoders at a high cost per pixel. However, GANs still face many unknown difficulties: it is clear that it is important to start learning when a lot of tricks have stagnated [15, 16]. The correct choice of hyperparameters is of utmost importance. Managing the diversity of images of generated samples is difficult. Balancing the compatibility of the discriminator and generator is difficult: often the discriminator must easily move to the beginning [6]. GANs easily suffer from collapse mode, failure mode, in which more than one image is implemented [5]. Heuristic regularizers, such as batch discrimination [16] and repelling regularizer [21], have been developed to solve problems with varying degrees of success. That's why the bullets are placed on the foot:

• Generative-saving measures with a simple, but reliable architecture, a standard procedure for getting started and stable production.

• The concept of equalization, which is equal to the tension of the discriminator and generator.

• A new way of creating a compromise between image diversity and visual vibrancy.

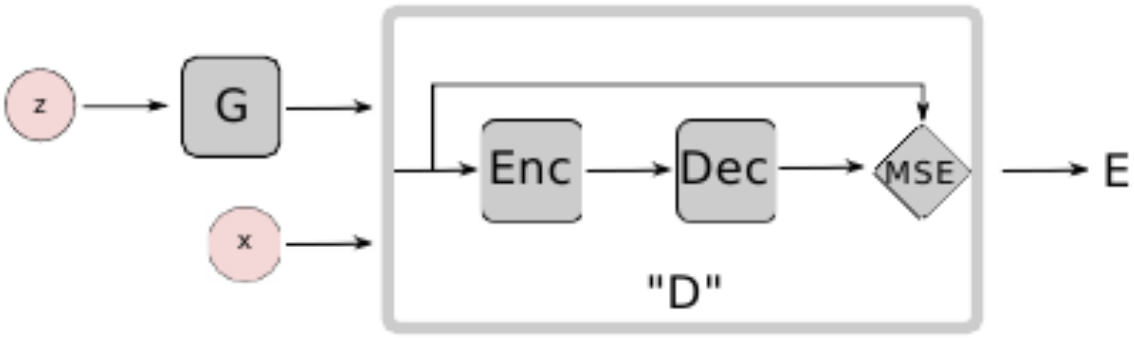

Fig.2.1. Block diagram of BEGAN

We use an auto-encoder as a discriminator. At that time, as typical GANs are able to directly establish the distribution of data, the method, which is seen in this work, aims to divide the expenses of the autocoder with the corresponding expenses, withdrawn from the Wasserstein subdivision. Use the standard GAN label with the addition of an equally important term to balance the discriminator and generator. This method has a simpler procedure and a more simple neural network architecture similar to typical GAN methods.

2.2. A look at the work connected with the generative-magal boundaries of the boundary river.

BEGAN (Generative Edge Measures of Boundary Equalization) has paid tribute to its discriminator CAE (Harth Autoencoder) and to the power of convergence estimation, which DCGAN does not have [24,25]. BEGAN enhances the acquisition of image space, preserving and adjusting the balance between the generator and discriminator. BEGAN uses AE (Auto Encoder) as a discriminator rather than a classifier. The discriminator recognizes that real images have a slight detriment to the reconstruction, while images generated by the generator have a great detriment to the reconstruction. The generator is recognized in such a way that the cost of reconstructing the discriminator is small. Under the control of DCGAN, BEGAN built before convergent evaluation.

The fundamental problem of GAN, the collapse of fashion, is also to blame for BEGAN. Apparently, BEGAN-CS (BEGAN with interconnected space) has been announced, which allows for interconnection of space without causing the problem of mode collapse [26].

Sung-Wook changed the structure of the BEGAN-CS discriminator from AE to VAE to overcome the mode collapse problem, and also changed the structure of the encoder and decoder [27].

The activation function was changed from ELU (Exponential Linear Unit) to LReLU (Rectified Linear Unit with Turn) [28,29]. The KLD (Kullback-Leibler Divergence) member was added to the main discriminator cost function [30]. The implementation model could solve the problem of fashion collapse.

2.3. Theoretical foundations of everyday life BEGAN

Whereas typical GANs are designed to immediately create divisions of data, in the BEGAN neural network there is a method of targeting divisions of auto-coder expenses with vicoristic expenses subtracted from Wasserstein's subdivision. Use the standard GAN label with the addition of an equally important term to balance the discriminator and generator. As a result, the procedure for starting is much simpler and the architecture of the neural network is more similar to standard GAN methods.

First, the input of the auto-coder is entered, then the lower boundary of the Wasserstein division between the divisions of the auto-coder of real and generated samples is calculated. To be introduced $L:\sim^{N_x} \to \sim^{+}$ spending to start a pixel-by-pixel auto coder

$$\mathcal{L}(v) = |v - D(v)|^{\eta} \text{ where } \begin{cases} D : \mathbb{R}^{N_x} \mapsto \mathbb{R}^{N_x} & \text{is the autoencoder function.} \\ \eta \in \{1, 2\} & \text{is the target norm.} \\ v \in \mathbb{R}^{N_x} & \text{is a sample of dimension } N_x. \end{cases}$$

In practice, it is extremely important to maintain a balance between the generator and discriminator losses; we consider them to be in equilibrium when:

$$\mathbb{E}[\mathcal{L}(x)] = \mathbb{E}[\mathcal{L}(G(z))] \quad (3)$$

f we generate samples that the discriminator cannot distinguish from real ones, their error distribution should be the same, including their expected error. This concept allows us to balance the effort allocated to the generator and the discriminator so that neither beats the other. We can relax the equilibrium by introducing a new hyperparameter $\gamma \in [0, 1]$, defined as $\gamma = \frac{\mathbb{E}[\mathcal{L}(G(z))]}{\mathbb{E}[\mathcal{L}(x)]}$ (4)

In our model, the discriminator has two competing goals: automatically encoding real images and distinguishing real images from generated ones. The γ term allows us to balance these two goals. Lower values of γ result in less diversity in images, since the discriminator focuses more on automatically encoding real images. We will call γ the diversity coefficient. There is a natural cutoff beyond which images are sharp and have detail. The BEGAN boundary equilibrium is defined as follows. We define this goal as

$$\begin{cases} \mathcal{L}_D = \mathcal{L}(x) - k_t.\mathcal{L}(G(z_D)) & \text{for } \theta_D \\ \mathcal{L}_G = \mathcal{L}(G(z_G)) & \text{for } \theta_G \\ k_{t+1} = k_t + \lambda_k(\gamma\mathcal{L}(x) - \mathcal{L}(G(z_G))) & \text{for each training step } t \end{cases}$$

where L_D and L_G - loss functions of the discriminator and generator, respectively, where x are the actual samples, $z \in [-1,1]^{N_z}$ – uniform random samples of dimension N_z.

The key part of BEGAN is the k parameter (k-tolerance), which adaptively regulates the balance between the generator and the discriminator. This ensures equal stability during the beginning and ensures the dominance of one of the components. The update of the parameter k follows the formula: $k_{t+1} = k_t + \lambda\,(\gamma\mathcal{L}_{\text{real}} - \mathcal{L}_{\text{fake}})$

where:

- $\mathcal{L}_{\text{real}} = \mathbb{E}_{x\sim p_{\text{data}}(x)}\,[L(x, D(x))]$ — average loss of reconstruction for reference images.
- $\mathcal{L}_{\text{fake}} = \mathbb{E}_{z\sim p_z(z)}\,[L(x, D(G(z)))]$ — average failure of reconstruction for generated images.
- γ— a hyperparameter that sets the balance of expenses.
- λ— tempo for updating for k.

BEGAN proposes a special metric for assessing the stability of progress -

convergence metric (convergence measure) M, which is based on comparisons between reconstructions for referenced and generated expressions: $M = \mathcal{L}_{\text{real}} + |\gamma \cdot \mathcal{L}_{\text{real}} - \mathcal{L}_G|$

Once again, this metric can be monitored for the progress of the past and presently make adjustments to hyperparameters.

We use proportional control theory to maintain balance $\mathrm{E}[L(G(z))] = \gamma \mathrm{E}[L(x)]$. This is implemented using a variable $k_t \in [0, 1]$ to control how much attention is given $L(G(z_D))$

during gradient descent. We initialize $k_0 = 0$. λ_k – is the proportional gain for k; in machine learning terms, this is the learning rate for k. We used 0.001 in our experiments. In essence, this can be thought of as a form of closed-loop feedback control, in which kt is adjusted at each step to maintain the equation (4).

In the early stages of training, G tends to generate easily recoverable data for the autoencoder, since the generated data is close to 0 and the real data distribution has not yet been accurately learned. This leads to L(x) > L(G(z)) early on, and this is maintained throughout the training process by the equilibrium constraint.

3 SECTION
FEATURES OF THE BEGAN NEURAL NETWORK

3.1. Architecture of the BEGAN neural network

The Boundary Equilibrium Generative Adversarial Network (BEGAN) is one of the variations of the generative network (GAN) architecture that involves a special approach to balancing the generator and discriminator.

Advantages of BEGAN

1. Stability: Based on the k-trend and the convergence metric, BEGAN will provide quick control over the developments compared to classic GANs.
2. Vibrancy of the image: Always respect the reconstruction of the generated images so they look natural and realistic.

Thus, BEGAN is distinguished by its unique approach to the balanced components of the generator and discriminator, the vicoristic and autoencoder architecture and innovative waste functions. This is the particularity of creating a model that is suitable for a wide range of tasks in the field of generative modeling.

The main features of the BEGAN neural network center on its architecture, input functions, and methods of stabilizing the process of initiation. BEGAN uses the autoencoder as the basis for the discriminator. This structure consists of two components: BEGAN uses an autoencoder as the basis for the discriminator. This structure consists of two components:

1. Encoder: Converts the input image into a compact latent space, visible key characters.
2. Decoder: Updates input from latent representation, generating image reconstruction.

BEGAN's generator begins to generate images that, after passing through the discriminator-autoencoder, result in the least damage to the reconstruction.

Discriminator. As part of this work, the implementation discriminator is an autoencoder, which is based on ResNet18. Main components:

1. Encoder (ResNet18Encoder):
    - Uses pre-trained ResNet18 modified to work with single-channel images.
    - The first convolutional layer is modified to process 1-channel images.

- The encoder calculates a latent vector of size 512.

2. Decoder (ResNet18Decoder):

- Starts with a linear layer that transforms the latent vector into an initial 4x4 map with 512 channels.
- Consists of a sequence of transposed convolutional layers with ReLU activations and BatchNorm normalization.
- The last layer is a transposed convolution with Tanh activation that reconstructs a 64x64 image.

Generator - the generator is created as a separate network for generating images from latent space. Main stages:

1. Linear layer:

- Converts a 64-bit latent vector into a 4x4 spatial map with 512 channels.

2. Transposed convolutional layers:

- Sequentially increases the image size to 64x64.
- Uses ReLU activation functions and BatchNorm normalization.

3. Final layer:

- Ends with a transposed convolution with a Tanh function that normalizes pixel values to the range [-1, 1]. The topology of the model is shown in Fig.1

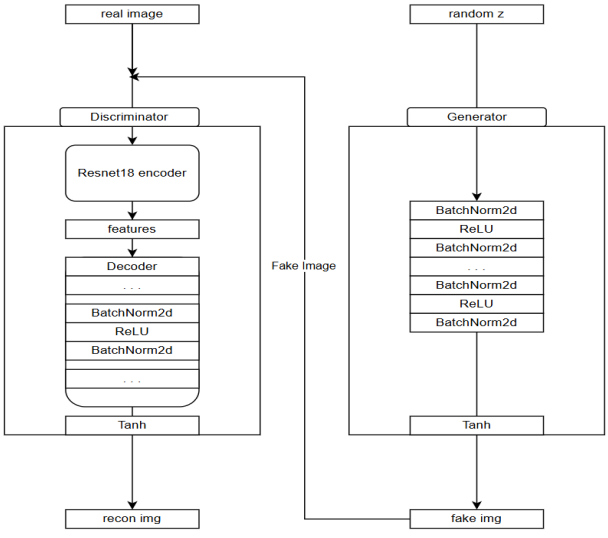

Fig.3.1. The topology of the model.

3.2. Loss function

BEGAN differs from classical GANs in that it uses reconstruction error as the main metric for evaluating the performance of the discriminator. The generator and discriminator are optimized with different objective functions:

- Discriminator: Minimizes the difference between the reconstruction of the real images and their generated counterparts. Formally, the loss function of the discriminator is defined as: $\mathcal{L}_D = \mathbb{E}_{x \sim p_{\text{data}}(x)} [L(x, D(x))] - k \cdot \mathbb{E}_{z \sim p_z(z)} [L(x, D(G(z)))]$ where:
  - $L(x, D(x))$— reconstruction function for real images.
  - $L(x, D(G(z)))$— reconstruction function for generated images.
  - k — balancing parameter between generator and discriminator.
- Generator: Learns to generate images that, after reconstruction by the discriminator, have minimal error. The generator loss function is defined as: $\mathcal{L}_G = \mathbb{E}_{z \sim p_z(z)} [L(x, D(G(z)))]$

These functions provide mutual optimization of the generator and discriminator, where the generator tries to minimize the reconstruction error of the generated images, and the discriminator strives to accurately reconstruct both the real and generated images, controlling the balance through the parameter k.

The algorithm of the BEGAN model can be divided into the following steps:

1. Initialization of the model:
   - Creation of a generator and a discriminator (autoencoder) with a given architecture.
   - Initialization of parameters such as k-trend and learning rate.
2. Data generation:
   - The generator creates samples based on random vectors from the latent space.
3. Discriminator evaluation:
   - The discriminator (autoencoder) evaluates both the real and generated samples, performing reconstruction for both.
   - Reconstruction errors for the real and generated images are calculated.
4. Discriminator update:

- The difference between the reconstruction errors of the real and generated samples is minimized using gradient descent.

5. Generator update:
   - The generator learns to create samples that minimize the reconstruction error of the discriminator.
6. K-trend adaptation:
   - The parameter k is updated to maintain a balance between the generator and the discriminator according to a given convergence metric.
7. Repetition:
   - The cycle is repeated until convergence or a given number of epochs are achieved.
8. Quality assessment:
   - The generated samples are analyzed for diversity, realism, and the absence of mode collapse.

This algorithm allows BEGAN to achieve training stability and high-quality generation while maintaining a balance between the generator and the discriminator. The BEGAN machine learning algorithm scheme is presented in Fig. 2.

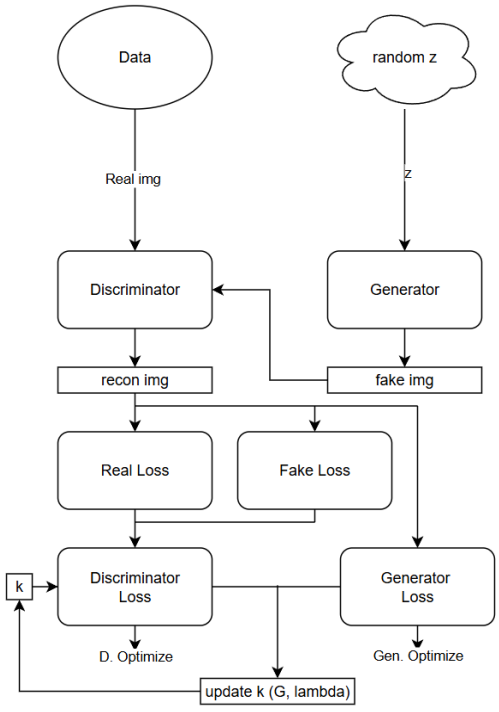

Fig.3.2. BEGAN machine learning algorithm

4 SECTION
RESULTS AND THEIR ANALYSIS

4.1. Dataset description

The FashionMNIST dataset is a standard dataset for computer vision tasks. FashionMNIST consists of:

- 60,000 images for training and 10,000 images for testing.
- Each image is presented in grayscale, has a size of 28x28 pixels and contains one of 9 classes (e.g., T-shirts, shoes, bags, etc.).

Advantages of using FashionMNIST

1. Simplicity: The small size of the images allows you to quickly train models even on limited hardware resources.
2. Diversity: The 9 classes provide a variety of data, which is useful for evaluating generative models.
3. Wide use: The dataset is a generally accepted standard, allowing you to easily compare results with other studies.

Disadvantages of using FashionMNIST

1. Low resolution: 28x28 images may not be detailed enough to generate high-quality images. Thus, FashionMNIST provides a good basis for testing image generation models, but its limitations should be taken into account when evaluating the results obtained. 4.2 Results and their analysis Figure 1 presents the images generated by the model.

Fig. 4.1. Images generated by the model

Based on the presented image of the results, the following conclusions can be drawn about the quality of the BEGAN model generation:

1. General appearance of the generated samples: The model is able to generate images that visually look like classes from the FashionMNIST dataset (T-shirts, shoes, etc.) and the contours of the objects are clearly defined in many samples.

2. Variety of generation: The model demonstrates a good level of variety, generating objects of all classes.

3. Detail: Due to the low resolution of the dataset (28x28), the detail is limited, and the generated samples have blurry textures.

4. Noise issues: Noise is observed in some samples

Solving the mode collapse problem

One of the key problems in generative adversarial networks is mode collapse, when the generator focuses only on a limited subset of the data distribution, creating samples with little variety. In this work, this problem was solved as follows:

Reducing the learning rate of the discriminator:

- Reducing the learning rate of the discriminator allows the generator to better adapt to changes in the discriminator, avoiding too rapid a penalty for imperfect samples.

Training logs:

Epoch [1/8], Step [900/938], D_loss: 0.1967, G_loss: 0.1152, k: 0.0770, M: 0.2177

Epoch [2/8], Step [900/938], D_loss: 0.1248, G_loss: 0.0733, k: 0.0660, M: 0.1383

Epoch [3/8], Step [900/938], D_loss: 0.1183, G_loss: 0.0754, k: 0.0480, M: 0.1362

Epoch [4/8], Step [900/938], D_loss: 0.0964, G_loss: 0.0706, k: 0.0320, M: 0.1198

Epoch [5/8], Step [900/938], D_loss: 0.0900, G_loss: 0.0507, k: 0.0191, M: 0.0962

Epoch [6/8], Step [900/938], D_loss: 0.0857, G_loss: 0.0442, k: 0.0112, M: 0.0873

Epoch [7/8], Step [900/938], D_loss: 0.0783, G_loss: 0.0470, k: 0.0073, M: 0.0862

Epoch [8/8], Step [900/938], D_loss: 0.0796, G_loss: 0.0359, k: 0.0060, M: 0.0839

CONCLUSIONS

1. It is shown that the developed generative-adversarial marginal equilibrium network is capable of generating images that visually look like classes from the FashionMNIST dataset and demonstrates a good level of diversity, generating objects of all classes.

2. Due to the low resolution of the dataset (28x28), the detail is limited, and the generated samples have blurred textures, and noise is observed in some samples.

3. To solve the mode collapse problem, it was proposed to reduce the learning rate of the discriminator, allowing the generator to better adapt to changes in the discriminator, avoiding too rapid a penalty for imperfect samples.

LIST OF SOURCES USE

1. Yann, L.; Yoshua, B.; Geoffrey, H. Deep Learning. Nature 2015, 521, 436–444.
2. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Bengio, Y. Generative adversarial nets. arXiv 2020,
3. arXiv:1406.2661. Available online: https://arxiv.org/abs/1406.2661 (accessed on 21 April 2020).
4. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Polosukhin, I. Attention is all you need. arXiv 2020,
5. arXiv:1706.03762. Available online: https://arxiv.org/abs/1706.03762 (accessed on 21 April 2020).
4. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-Training of Deep Bidirectional Transformers for Language Understanding.
6. arXiv 2020, arXiv:1810.04805. Available online: https://arxiv.org/abs/1810.04805 (accessed on 22 April 2020).
5. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.; Dhariwal, P.; Amodei, D. Language Models Are Few-Shot Learners.
7. arXiv 2020, arXiv:2005.14165. Available online: https://arxiv.org/abs/2005.14165 (accessed on 22 August 2020).
6. Payne, C. MuseNet. Available online: https://openai.com/blog/musenet (accessed on 23 April 2020).
7. Yamamoto, R.; Song, E.; Kim, J.M. Parallel WaveGAN: A fast waveform generation model based on generative adversarial
8. networks with multi-resolution spectrogram. In Proceedings of the ICASSP 2020–2020 IEEE International Conference on Acoustics,
9. Speech and Signal Processing, Barcelona, Spain, 4–8 May 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 6199–6203.
8. Karras, T.; Aila, T.; Laine, S.; Lehtinen, J. Progressive Growing of Gans for Improved Quality, Stability, and Variation. arXiv 2020,
10. arXiv:1710.10196. Available online: https://arxiv.org/abs/1710.10196 (accessed on 16 May 2020).

9. Zhang, H.; Goodfellow, I.; Metaxas, D.; Odena, A. Self-Attention Generative Adversarial Networks. arXiv 2020, arXiv:1805.08318.
11. Available online: https://arxiv.org/abs/1805.08318 (accessed on 17 May 2020).
10. Brock, A.; Donahue, J.; Simonyan, K. Large Scale Gan Training for High Fidelity Natural Image Synthesis. arXiv 2020,
12. arXiv:1809.11096. Available online: https://arxiv.org/abs/1809.11096 (accessed on 20 May 2020).
11. Karras, T.; Laine, S.; Aila, T. A style-based generator architecture for generative adversarial networks. In Proceedings of the
13. IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–20 June 2019; IEEE: Piscataway, NJ, USA,
14. 2019; pp. 4401–4410.
12. Nwankpa, C.; Ijomah, W.; Gachagan, A.; Marshall, S. Activation Functions: Comparison of Trends in Practice and Research for
15. Deep Learning. arXiv 2020, arXiv:1811.03378. Available online: https://arxiv.org/abs/1811.03378 (accessed on 22 May 2020).
13. LeCun, Y.; Touresky, D.; Hinton, G.; Sejnowski, T. A theoretical framework for back-propagation. In Proceedings of the 1988
16. Connectionist Models Summer School, San Mateo, CA, USA, 1 June 1988; Volume 1, pp. 21–28.
14. Triola, M.F. Bayes' Theorem. Available online: http://faculty.washington.edu/tamre/BayesTheorem.pdf (accessed on
17. 10 January 2020).
15. Arjovsky, M.; Chintala, S.; Bottou, L. Wasserstein Gan. arXiv 2020, arXiv:1701.07875. Available online: https://arxiv.org/abs/17
18. 01.07875 (accessed on 25 May 2020).
16. Gouk, H.; Frank, E.; Pfahringer, B.; Cree, M. Regularisation of Neural Networks by Enforcing Lipschitz Continuity. arXiv 2020,
19. arXiv:1804.04368. Available online: https://arxiv.org/abs/1804.04368 (accessed on 26 May 2020).

17. Gulrajani, I.; Ahmed, F.; Arjovsky, M.; Dumoulin, V.; Courville, A.C. Improved training of wasserstein gans. In Proceedings of the
20. Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017; Curran Associates Inc.: Red Hook, NY,
21. USA, 2018; pp. 5767–5777.
18. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv 2020,
22. arXiv:1502.03167. Available online: https://arxiv.org/abs/1502.03167 (accessed on 28 May 2020).
19. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In Proceedings of the
23. Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; Curran Associates Inc.: Red Hook, NY,
24. USA, 2013; pp. 1097–1105. Radford, A.; Metz, L.; Chintala, S. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial
25. Networks. arXiv 2020, arXiv:1511.06434. Available online: https://arxiv.org/abs/1511.06434 (accessed on 29 May 2020).
20. Kingma, D.P.; Welling, M. Auto-Encoding Variational Bayes. arXiv 2020, arXiv:1312.6114. Available online: https://arxiv.org/
26. abs/1312.6114 (accessed on 1 June 2020).
21. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. arXiv 2020, arXiv:1412.6980. Available online: https:
27. //arxiv.org/abs/1412.6980 (accessed on 2 June 2020).
22. Yeh, R.; Chen, C.; Lim, T.Y.; Hasegawa-Johnson, M.; Do, M.N. Semantic Image Inpainting with Perceptual and Contextual Losses.
28. arXiv 2020, arXiv:1607.07539. Available online: https://arxiv.org/abs/1607.07539 (accessed on 3 June 2020).
23. Berthelot, D.; Schumm, T.; Metz, L. Began: Boundary Equilibrium Generative Adversarial Networks. arXiv 2020, arXiv:1703.10717.
29. Available online: https://arxiv.org/abs/1703.10717 (accessed on 4 June 2020).

24. Baldi, P. Autoencoders, unsupervised learning, and deep architectures. In Proceedings of the ICML Workshop on Unsupervised
30. and Transfer Learning; Proceedings of Machine Learning Research, New York City, NY, USA, 19–24 June; 2016; pp. 37–49.
25. Chang, C.C.; Hubert Lin, C.; Lee, C.R.; Juan, D.C.; Wei, W.; Chen, H.T. Escaping from collapsing modes in a constrained space. In
31. Proceedings of the European Conference on Computer Vision, Munich, Germany, 8–14 September 2018; Springer: New York, NY,
32. USA, 2019; pp. 204–219.
26. Park, S.W.; Huh, J.H.; Kim, J.C. BEGAN v3: Avoiding Mode Collapse in GANs Using Variational Inference. Electronics 2020, 9,
33. 688. [CrossRef]
27. Clevert, D.A.; Unterthiner, T.; Hochreiter, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (Elus). arXiv
34. 2020, arXiv:1511.07289. Available online: https://arxiv.org/abs/1511.07289 (accessed on 5 June 2020).
28. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. Proc. ICML 2013, 30, 3.
29. Pйrez-Cruz, F. Kullback-Leibler divergence estimation of continuous distributions. In Proceedings of the 2008 IEEE International
35. Symposium on Information Theory, Toronto, ON, Canada, 8 August 2008; IEEE: New York, NY, USA, 2008; pp. 1666–1670.
30. Yu, F.; Seff, A.; Zhang, Y.; Song, S.; Funkhouser, T.; Xiao, J. Lsun: Construction of a Large-Scale Image Dataset Using Deep
36. Learning with Humans in the Loop. arXiv 2020, arXiv:1506.03365. Available online: https://arxiv.org/abs/1506.03365 (accessed
37. on 6 June 2020).
31. Chen, X.; Duan, Y.; Houthooft, R.; Schulman, J.; Sutskever, I.; Abbeel, P. Infogan: Interpretable representation learning by

38. information maximizing generative adversarial nets. In Proceedings of the Advances in Neural Information Processing Systems,
39. Barcelona, Spain, 5–10 December 2016; Curran Associates Inc.: Red Hook, NY, USA, 2016; pp. 2172–2180.
32. Mirza, M.; Osindero, S. Conditional Generative Adversarial Nets. arXiv 2020, arXiv:1411.1784. Available online: https://arxiv.
40. org/abs/1411.1784 (accessed on 9 June 2020).
33. Creswell, A.; Bharath, A.A. Inverting the Generator of a Generative Adversarial Network. IEEE Trans. Neural Networks Learn.
41. Syst. 2019, 30, 1967–1974. [CrossRef] [PubMed]
34. Lipton, Z.C.; Tripathi, S. Precise Recovery of Latent Vectors from Generative Adversarial Networks. arXiv 2020, arXiv:1702.04782.
42. Available online: https://arxiv.org/abs/1702.04782 (accessed on 10 June 2020).
35. Dumoulin, V.; Belghazi, I.; Poole, B.; Mastropietro, O.; Lamb, A.; Arjovsky, M.; Courville, A. Adversarially Learned Inference.
43. arXiv 2020, arXiv:1606.00704. Available online: https://arxiv.org/abs/1606.00704 (accessed on 11 June 2020).
36. Donahue, J.; Krдhenbьhl, P.; Darrell, T. Adversarial Feature Learning. arXiv 2020, arXiv:1605.09782. Available online: https:
44. //arxiv.org/abs/1605.09782 (accessed on 12 June 2020).
37. Li, C.; Liu, H.; Chen, C.; Pu, Y.; Chen, L.; Henao, R.; Carin, L. Alice: Towards understanding adversarial learning for joint
45. distribution matching. In Proceedings of the Advances in Neural Information Processing Systems, Long Beach, CA, USA, 4–9
46. December 2017; Curran Associates Inc.: Red Hook, NY, USA, 2017; pp. 5495–5503.

Addition

```
import torch
    import torch.nn as nn
    import torch.optim as optim
    from torch.utils.data import DataLoader
    import torchvision
    import torchvision.transforms as transforms
    import matplotlib.pyplot as plt
    import os

    from torchvision.models import resnet18
    from tqdm.auto import tqdm

    # ----------------------
    # Hyperparameters
    # ----------------------
    batch_size = 64
    z_dim = 48
    lr = 1e-4
    num_epochs = 8
    gamma = 0.5
    lambda_k = 0.001
    image_size = 64
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # ----------------------
    # Data Loading (Upscale MNIST to 64x64)
    # ----------------------
```

```
    transform = transforms.Compose([
        transforms.Resize((image_size, image_size)),
        transforms.ToTensor(),
        transforms.Normalize([0.5], [0.5])
    ])

    root = './data'
    train_dataset = torchvision.datasets.FashionMNIST(root=root, train=True, transform=transform, download=True)
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)



    # ----------------------
    # ResNet18-Based Autoencoder Discriminator
    # ----------------------
    # We'll modify ResNet18 to serve as an encoder, and create a decoder afterwards.
    class ResNet18Encoder(nn.Module):
        def __init__(self):
            super(ResNet18Encoder, self).__init__()
            # Load a pre-defined ResNet18
            self.model = resnet18(weights='DEFAULT')
            # Modify the first conv layer to accept 1-channel input
            self.model.conv1 = nn.Conv2d(1, 64, kernel_size=7, stride=2, padding=3, bias=False)

            # We will use the layers up to the avgpool to get a feature vector of size 512
            # The forward pass will end just before the fully connected layer.

        def forward(self, x):
```

```
        # x: (B, 1, 64, 64)
        x = self.model.conv1(x)
        x = self.model.bn1(x)
        x = self.model.relu(x)
        x = self.model.maxpool(x)

        x = self.model.layer1(x)
        x = self.model.layer2(x)
        x = self.model.layer3(x)
        x = self.model.layer4(x)

        x = self.model.avgpool(x)  # (B, 512, 1, 1)
        x = torch.flatten(x, 1)    # (B, 512)
        return x

    class ResNet18Decoder(nn.Module):
        def __init__(self, latent_dim=512):
            super(ResNet18Decoder, self).__init__()
            # We have a 512-d vector. We'll map it to a feature map and upsample back to
64x64
            self.fc = nn.Linear(latent_dim, 512*4*4)  # Start from a smaller spatial map, say
4x4

            self.deconv = nn.Sequential(
                nn.BatchNorm2d(512),
                nn.ReLU(True),
                nn.ConvTranspose2d(512, 256, 4, 2, 1),  # (B,256,8,8)
                nn.BatchNorm2d(256),
                nn.ReLU(True),
                nn.ConvTranspose2d(256, 128, 4, 2, 1),  # (B,128,16,16)
```

```
            nn.BatchNorm2d(128),
            nn.ReLU(True),
            nn.ConvTranspose2d(128, 64, 4, 2, 1),   # (B,64,32,32)
            nn.BatchNorm2d(64),
            nn.ReLU(True),
            nn.ConvTranspose2d(64, 1, 4, 2, 1),     # (B,1,64,64)
            nn.Tanh()
        )

    def forward(self, z):
        # z: (B,512)
        z = self.fc(z)       # (B,512*4*4)
        z = z.view(-1, 512, 4, 4)
        z = self.deconv(z)   # (B,1,64,64)
        return z

class ResNet18Autoencoder(nn.Module):
    def __init__(self):
        super(ResNet18Autoencoder, self).__init__()
        self.encoder = ResNet18Encoder()
        self.decoder = ResNet18Decoder(latent_dim=512)

    def forward(self, x):
        features = self.encoder(x)      # (B,512)
        recon = self.decoder(features)  # (B,1,64,64)
        return recon

# ----------------------
# Generator
# ----------------------
```

```
class Generator(nn.Module):
    def __init__(self, z_dim=64, base_dim=64):
        super(Generator, self).__init__()
        self.fc = nn.Linear(z_dim, base_dim*8*4*4)
        self.deconv = nn.Sequential(
            nn.BatchNorm2d(base_dim*8),
            nn.ReLU(True),
            nn.ConvTranspose2d(base_dim*8, base_dim*4, 4, 2, 1),  # (B,256,8,8)
            nn.BatchNorm2d(base_dim*4),
            nn.ReLU(True),
            nn.ConvTranspose2d(base_dim*4, base_dim*2, 4, 2, 1),  # (B,128,16,16)
            nn.BatchNorm2d(base_dim*2),
            nn.ReLU(True),
            nn.ConvTranspose2d(base_dim*2, base_dim, 4, 2, 1),    # (B,64,32,32)
            nn.BatchNorm2d(base_dim),
            nn.ReLU(True),
            nn.ConvTranspose2d(base_dim, 1, 4, 2, 1),             # (B,1,64,64)
            nn.Tanh()
        )

    def forward(self, z):
        out = self.fc(z)  # (B,base_dim*8*4*4)
        out = out.view(-1, 512, 4, 4)  # base_dim*8=512
        out = self.deconv(out)
        return out

# ----------------------
# Initialize Models
# ----------------------
G = Generator(z_dim=z_dim).to(device)
```

```
D = ResNet18Autoencoder().to(device)

# Initialize weights in generator and decoder part of D
def weights_init(m):
    classname = m.__class__.__name__
    if 'Conv' in classname or 'ConvTranspose' in classname:
        nn.init.xavier_normal_(m.weight.data, gain=0.02)
        if m.bias is not None:
            m.bias.data.fill_(0)
    elif 'Linear' in classname:
        nn.init.xavier_normal_(m.weight.data, gain=0.02)
        if m.bias is not None:
            m.bias.data.fill_(0)

G.apply(weights_init)
D.decoder.apply(weights_init)

# ----------------------
# Optimizers
# ----------------------
g_optimizer = optim.Adam(G.parameters(), lr=1e-4, betas=(0.5, 0.999))
d_optimizer = optim.Adam(D.parameters(), lr=1e-5, betas=(0.5, 0.999))

# ----------------------
# Training
# ----------------------
k = 0.0
fixed_z = torch.randn(64, z_dim, device=device)

def recon_loss(x, x_recon):
```

```
    return torch.mean(torch.abs(x - x_recon))

G.train()
D.train()

if not os.path.exists('samples'):
    os.makedirs('samples')

for epoch in range(num_epochs):
    for i, (real_images, _) in tqdm(enumerate(train_loader)):
        real_images = real_images.to(device)

        # ------------------------
        # Train Discriminator (Autoencoder)
        # ------------------------
        z = torch.randn(real_images.size(0), z_dim, device=device)
        with torch.no_grad():
            fake_images = G(z)

        real_recon = D(real_images)
        fake_recon = D(fake_images)

        d_real_loss = recon_loss(real_images, real_recon)
        d_fake_loss = recon_loss(fake_images, fake_recon)

        d_loss = d_real_loss - k * d_fake_loss

        d_optimizer.zero_grad()
        d_loss.backward()
        d_optimizer.step()
```

```
            # ------------------------
            # Train Generator
            # ------------------------
            z = torch.randn(real_images.size(0), z_dim, device=device)
            fake_images = G(z)
            fake_recon = D(fake_images)
            g_loss = recon_loss(fake_images, fake_recon)

            g_optimizer.zero_grad()
            g_loss.backward()
            g_optimizer.step()

            # ------------------------
            # Update k for equilibrium
            # ------------------------
            balance = (gamma * d_real_loss - g_loss).item()
            k = k + lambda_k * balance
            k = max(min(k, 1), 0)

            if (i+1) % 100 == 0:
                M = d_real_loss.item() + abs(gamma * d_real_loss.item() - g_loss.item())
                print(f"Epoch [{epoch+1}/{num_epochs}], Step [{i+1}/{len(train_loader)}], "
                      f"D_loss: {d_loss.item():.4f}, G_loss: {g_loss.item():.4f}, k: {k:.4f}, M:
{M:.4f}")

        # Save samples
        G.eval()
        with torch.no_grad():
            samples = G(fixed_z).cpu()
```

```
            grid = torchvision.utils.make_grid(samples, nrow=8, normalize=True, value_range=(-1, 1))
            torchvision.utils.save_image(grid, f'samples/epoch_{epoch+1}.png')
        G.train()
```