DCGAN and Transfer Learning for Image Generation and Classification

Wu Xianzong, 3021208007

Abstract

In recent years, Generative Adversarial Network (GAN) models have made significant strides in the field of image generation and transformation. This study implements an enhanced GAN model on the MNIST and CIFAR-10 datasets, achieving more efficient generation of handwritten digit images through adjustments and optimizations of model parameters and structure. Furthermore, the enhanced CNN GAN model is integrated into a Deep Convolutional Generative Adversarial Network (DCGAN), which undergoes exploratory analysis and improvements. By applying transfer learning to the DCGAN model, knowledge is transferred from the source domain to the target domain, resulting in noticeable performance enhancements.

Key Words— CNN, DCGAN, Transfer Learning, Handwritten Image Generation, Performance Improvement

1. Introduction & Related works

Image generation and classification, as crucial branches in the field of computer vision, play a vital role, especially in applications like image super-resolution recovery and image style transfer. The core task involves creating entirely new images by simulating the genuine distribution of training data. With the ongoing development of deep learning, there is an increasing pursuit of accuracy in image recognition. Various deep learning neural networks, emulating human thinking, have emerged, offering unprecedented opportunities in the field of image generation. The design of network architectures, parameter adjustments, and sample selection directly impact the final recognition results of neural networks. Currently, many studies focus on Convolutional Neural Networks (CNNs) to enhance the accuracy of image recognition. CNNs, known for directly using the original pixels of an image as input, eliminate the need for traditional methods of feature extraction. This approach outperforms early work that relied on manually extracted features [1]. In fact, CNNs have been successfully applied to classification tasks such as handwritten character and gesture recognition [2], operating directly on data streams without preprocessing or feature selection. CNN-trained models exhibit invariance to distortions like scaling, translation, and rotation, showcasing robust generalization capabilities. The significant advantage of CNNs lies in their ability to map high-dimensional data onto shared convolutional kernels. Through multi-layer training in end-to-end networks, convolutional kernels handle complex feature computations. This design significantly reduces the number of parameters in neural networks, simultaneously lowering model complexity and providing ample space for optimizing classification accuracy.

Most CNN image classifications are based on supervised learning. During the training process, a large amount of data is required as training samples to achieve more accurate classification. However, some samples are challenging to collect, such as radar profiles in specific weather conditions. Collecting these samples is extremely difficult due to conditional constraints. Ian Goodfellow [3] proposed Generative Adversarial Networks (GANs), a framework for generative models inspired by game theory. **GANs can generate images or image restorations, and their core idea involves the generator and discriminator engaging in mutual competition.** This process allows the generator to gradually learn to generate realistic images, while the discriminator becomes more accurate in distinguishing between real and generated images. In the adversarial training process, GANs progressively reduce the dissimilarity between generated and real data. The well-trained generator can synthesize lifelike generated images, making it challenging for the discriminator to distinguish between real and generated images. As an unsupervised deep generative model, GANs have gained widespread attention due to their lack of reliance on any prior assumptions and outstanding generative performance. In various applications such as virtual try-on, video object detection, and image super-resolution recovery, GANs demonstrate remarkable performance.

While Generative Adversarial Networks (GANs) have significant advantages over other methods, early GAN models still suffer from drawbacks such as slow convergence, gradient vanishing, and training instability. These issues often result in poor generation effects and excessively long training times, and in severe cases, mode collapse. To overcome these limitations, researchers have made improvements and innovations in various aspects of GANs, leading to a variety of enhanced GAN models. These enhancements involve innovations in network architecture, training algorithms, loss functions, aiming to enhance the stability, generation quality, and training efficiency of GANs. These efforts have made GANs perform more impressively in practical applications. Mao et al. [4] introduced the Least Squares GAN (LSGAN), which utilizes

the least squares method in the discriminator to assess whether the distribution of real and generated data is the same. They replaced the activation function in the last layer of the discriminator with a linear function, addressing the issue of gradient vanishing when using the original Sigmoid function as the activation function during GAN training. However, this work still employed the Jensen-Shannon divergence (JS divergence) to measure the difference between real and generated data, without proposing a more effective measurement method. fGAN provided a new perspective on this issue, derived from the mathematical principles behind GANs. It suggested that any divergence satisfying certain conditions could be used in the GAN framework, not limited to JS divergence. Inspired by this work, many researchers have explored suitable distance measures from this perspective to improve the GAN training process. Arjovsky et al. [5] proposed using Wasserstein distance as a replacement for JS divergence, initially addressing the problem of mode collapse. Wasserstein distance better measures the difference between the distributions of generated and real data. However, as Wasserstein distance lacks upper and lower bounds, it can lead to non-convergence of the discriminator. Wasserstein GAN (WGAN) used weight clipping to restrict parameter ranges and ensure convergence. Nevertheless, WGAN faced instability during training because it did not genuinely limit the discriminator to be 1-Lipschitz. WGAN-GP introduced gradient penalty as a replacement for weight clipping, solving the instability issue in the training process.

The above improvements in GANs primarily focused on loss variants. Now let's delve into some models that have been enhanced through architectural variants. Energy-Based GAN (EBGAN) [6] addresses the differentiation issue between real and generated samples by employing an auto-encoder as the discriminator, enabling discriminative sample reconstruction instead of direct authenticity scoring. The discriminator benefits from pre-training, and optimization strategies such as k-step discriminator optimization and 1-step generator optimization help prevent discriminator overfitting. Activation functions like maxout and ReLU are utilized to enhance performance, allowing the generator to rapidly gain "energy drive" and reducing training costs. Conditional GAN (CGAN) [7] optimizes the generator and discriminator by providing additional class label information, resolving issues of traditional GANs in handling detailed features. The encoding and fusion of class labels enhance the discriminator's classification ability, enabling CGAN to handle complex datasets containing multi-modal data, such as Flickr. Semi-supervised GAN (SGAN) [8], proposed within a semi-supervised learning framework, aims to address the high cost of supervised learning labels and the poor performance of unsupervised learning. SGAN's discriminator adopts a multi-head architecture, simultaneously performing real data classification and real vs. fake sample differentiation using softmax and sigmoid, providing better performance than the original GAN. StyleGAN [9] innovatively resolves limitations in generating images with specific features that were present in ProGAN [10]. StyleGAN fine-tunes visual features by modifying the input at each layer, utilizing style transfer techniques to separate high-level attributes from random variations and control synthesis intuitively by scale. Its mapping network maps latent vectors to a more flexible intermediate latent space, enhancing both generation quality and attribute decoupling. It introduces a novel method for quantized latent space decoupling and introduces the high-quality FFHQ face dataset. Self-Attention GAN (SAGAN) [11] overcomes limitations in traditional CNN-based GANs, such as limited information capture and finite receptive fields. SAGAN introduces a self-attention mechanism, enabling the generator and discriminator to capture global dependencies and more accurately generate key semantic parts of images, such as the correct positioning of facial features.

The Deep Convolutional GAN (DCGAN) [12], featured in this paper, represents a significant milestone in the evolution of Generative Adversarial Networks (GANs). It introduced the transposed convolutional neural network architecture, enhancing the generator's upsampling capability. By employing stride convolutions instead of traditional pooling layers, DCGAN effectively prevents information loss, particularly making significant strides in modeling high-resolution images. **Compared to the original GAN, DCGAN underwent crucial architectural modifications, including the introduction of batch normalization to facilitate more effective training.** It also adopted different activation function strategies, enabling the generator to better learn complex image features. This landmark model has had a profound impact on subsequent GAN research, especially with the widespread adoption of transposed convolutions. However, due to DCGAN's limitations in model capacity, challenges persist in generating high-resolution and diverse images.

In order to reduce the training cost of small-scale datasets, we adopted the approach of Transfer Learning [13] using feature vector extraction. Transfer Learning provides a method to leverage the powerful feature extraction capabilities of large neural networks. It aims to enhance the training efficiency of a new model by transferring the parameters from a pre-trained model. The motivation behind this method lies in the correlation between data or tasks. By sharing the learned model parameters, or knowledge, it effectively accelerates and optimizes the learning process of the new model, avoiding starting from scratch. There are different approaches to Transfer Learning, including Transfer Learning itself, Extract Feature Vector, and Fine-tune. In Transfer Learning, the convolutional layers of the pre-trained model are frozen, and only custom fully connected layers are trained. Extract Feature Vector calculates the feature vectors of the pre-trained model's convolutional layers for all data and then trains a

customized, simplified version of a fully connected network. **Fine-tune involves freezing some of the pre-trained model's convolutional layers and training only the remaining convolutional layers and fully connected layers.** Transfer Learning emphasizes how to better apply previously learned knowledge, and Fine-tune is one of the means often used in the later stages of Transfer Learning.

The main contributions of this report are summarized as follows:

1) Improvement of the original GAN network based on the MNIST dataset, considering aspects such as learning rate, activation function, optimizer, and network structure. The performance in terms of accuracy and D_loss is significantly better than the original algorithm.

2) Comparative experiments on the MNIST and CIFAR-10 datasets, introducing an enhanced DCGAN algorithm. This improvement incorporates Wasserstein distance [14], spectral normalization, and noise input dimension estimation, resulting in significantly better performance compared to a standard GAN network.

3) Building upon the conclusions from the first two experiments, we further enhanced the DCGAN model by transforming its discriminator into a 10-class classifier. Through transfer learning and fine-tuning strategies, we optimized the model's performance, achieving accuracy levels very close to the original CNN model.

The project code is available at https://github.com/Paddingbuta/TransferLearning-DCGANs

2. Research Methodology

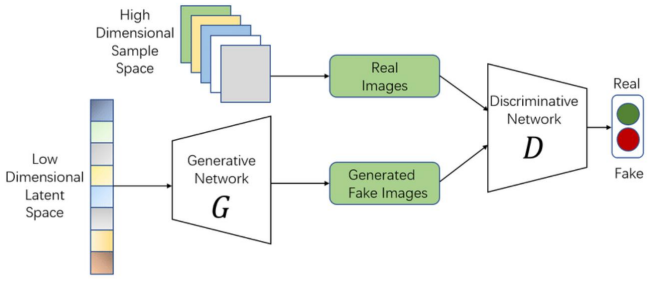
Convolutional Neural Network (CNN) is a type of deep learning model specifically designed for processing gridded data, such as images and videos. Its main components include convolutional layers, pooling layers, and fully connected layers. Convolutional layers perform feature extraction through convolutional operations, pooling layers are used for downsampling and reducing the size of feature maps, and fully connected layers are employed for classification tasks. CNNs have achieved significant success in image processing, effectively capturing local features in images.

Initially, we run a basic CNN model constructed using the Sequential API. This model comprises two convolutional layers, each followed by a max-pooling layer. It concludes with two fully connected layers, and the output layer utilizes the Softmax activation function for multi-class classification tasks. After building the model, it is compiled using the SGD (Stochastic Gradient Descent) optimizer and categorical cross-entropy loss function. Subsequently, image data undergo preprocessing, scaling pixel values to the range of [-1, 1], and one-hot encoding the labels. With data preparation and preprocessing complete, the model commences training. The model.fit() function is invoked, using the training set's images and labels for training, with specified batch sizes, epochs, and validation sets. During training, the model gradually adjusts weights through backpropagation and the gradient descent optimization algorithm to enhance accuracy in classifying handwritten digits.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 128)	204,928
dense_1 (Dense)	(None, 10)	1,290

Table 1: Description of the CNN Model

The abstract of Goodfellow's paper [3] states that Generative Adversarial Networks (GANs) simultaneously train two models through an adversarial process: a generative model G, tasked with capturing the data distribution, and a discriminative model D, aimed at estimating the probability that a sample comes from the training data rather than G. In this training process, the objective of G is to maximize the probability of D making errors. This framework corresponds to a minimax two-player game. Through adversarial training, these two networks continually optimize, ultimately leading the generator to produce realistic data, while the discriminator becomes increasingly challenged to distinguish between real and generated samples. GANs find extensive applications in tasks such as image generation and image editing.



Figue 1: Basic Structure of GAN

In this paper, the generator G of the GAN network takes random noise z as input and maps it to a 784-dimensional MNIST image space through a multi-layer fully connected network, ultimately producing a generated handwritten digit image. The generator network consists of an input layer, three hidden layers, and an output layer. ReLU activation functions are employed in the hidden layers, and the output layer uses Tanh to map the results to the range of -1 to 1. The discriminator D is also a multi-layer fully connected network that takes an MNIST image as input. It extracts image features through three hidden layers and outputs a probability between 0 and 1, indicating whether the input image is real or generated. ReLU activation functions are used in the hidden layers of the discriminator, and dropout is incorporated. The output layer uses sigmoid to map the results to a probability range of 0-1. During GAN training, the discriminator and generator are alternately trained. The discriminator aims to

classify real images as 1 and generated images as 0, while the generator aims to produce images classified as 1 by the discriminator. Their respective loss functions are designed to reflect these objectives.

The Deep Convolutional Generative Adversarial Network (DCGAN) is an improvement upon the traditional GAN, specifically designed for image generation. **DCGAN introduces convolutional layers and transpose convolutional layers, enhancing the ability of the generator and discriminator to handle image data effectively.** After the training starts, the random noise z is input to G to learn the model parameters. At this point, the desired probability of the sample data $G(z)$ generated by G after being judged by D is 1, mathematically represented as $D(G(z)) = 1$, or $1 - D(G(z)) = 0$, that is, minimizing the G model. In the training of model D, if the input sample data is real data x, the expected target is that the probability after D judgment is 1, mathematically represented as $D(x) = 1$; if the input sample data is $G(z)$, then the desired probability after D judgment is 0, that is, $D(G(z)) = 0$, so $1 - D(G(z)) = 1$. Therefore, maximizing the D model. The training problems of G and D can be represented by the value function $V(G, D)$, turning into the maximization-minimization problem of $V(G, D)$. It can be expressed as equation (1):

$$\min_G \max_D V(G,D) = E_{x\sim p_{\text{data}}(x)}[\lg D(z)] + E_{z\sim P_g(z)}[\lg(1 - D(G(z)))] \quad (1)$$

In comparison to a regular GAN, the DCGAN in this paper incorporates the following improvements:

1. Both the discriminator and generator use convolutional networks instead of fully connected networks. This enables the model to handle images of arbitrary sizes, not limited to the fixed size of MNIST (28x28).

2. The generator utilizes transpose convolutional layers to upsample feature maps, generating higher-resolution images compared to fully connected layers, resulting in more realistic outputs.

3. Batch Normalization is applied in both the generator and discriminator, contributing to the stability of GAN training.

4. The discriminator's final output employs a linear layer for logits instead of sigmoid, enhancing training stability.

5. The Wasserstein GAN's loss function is used instead of the original GAN's log loss, improving training stability.

Transfer learning is a machine learning strategy that revolves around applying knowledge learned in one task to another related task. This approach is particularly useful when dealing with limited data in the target domain, aiming to enhance the model's performance on the target task by leveraging prior knowledge from a source domain. Transfer learning methods primarily include feature extraction and model fine-tuning. In feature extraction, the early layers of the model act as feature extractors, are frozen, and a new classifier is trained on the target domain. In model fine-tuning, not only are the early layers frozen, but some top layers are also fine-tuned. Additionally, transfer learning can be categorized into same-domain transfer, cross-domain transfer, and multi-task transfer learning, depending on the relationship between the source and target domains.

We modified the discriminator in the DCGAN model into a 10-class classifier to address a multi-class problem. To achieve this, we removed the last dense layer of the original GAN discriminator, which was responsible for distinguishing between real and generated images. Based on this, we have several options regarding how to add dense layers. The first option is to add only one dense layer with 10 units and a softmax activation function. The second option is to add a dense layer with many units and then add another dense layer with 10 units and a softmax activation. We conducted comparative experiments in each of these four scenarios:

Experiment 1: Transfer Learning Only - Replace only the last dense layer with a trainable layer.

Experiment 2: Transfer Learning and Fine-tuning - Build upon Experiment 1 by introducing fine-tuning. Freeze all layers for the first 10 epochs, and then unfreeze them for the subsequent 10 epochs.

Experiment 3: Transfer Learning and Fine-tuning - Extend Experiment 2 by adding two additional dense layers. Based on experimental results, freeze all layers except the last two added layers for the first 10 epochs, and then unfreeze them for the next 10 epochs.

Experiment 4: Transfer Learning and Fine-tuning - Further modify Experiment 3 by freezing all layers except the last two added layers for the first 6 epochs, and then unfreezing them for the following 14 epochs.

3. Experiments and Results Analysis

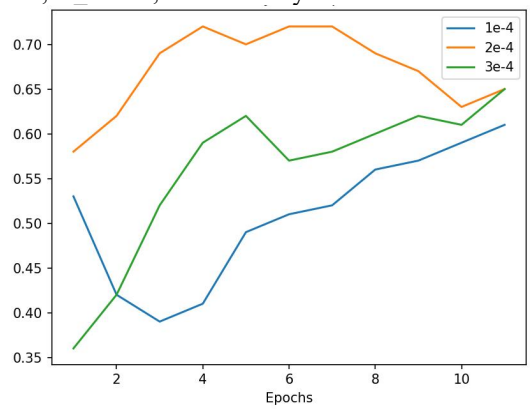
This experiment is primarily based on the MNIST dataset and validated on CIFAR-10, implemented using the Python+Keras platform. The initial parameters for this experiment are as follows:

- **Batch size:** 256
- **Number of epochs:** 20, indicating the number of times the model will iterate through the entire training dataset
- **Image scaling factor:** 255.0 / 2, used to scale pixel values to the [-1, 1] range
- **Activation function:** ReLU
- **Optimizer:** Adam
- **Dimension of the noise vector:** 64, used as input for the generator
- **Size of the test set:** 10,000
- **Model learning rate:** 1e-4

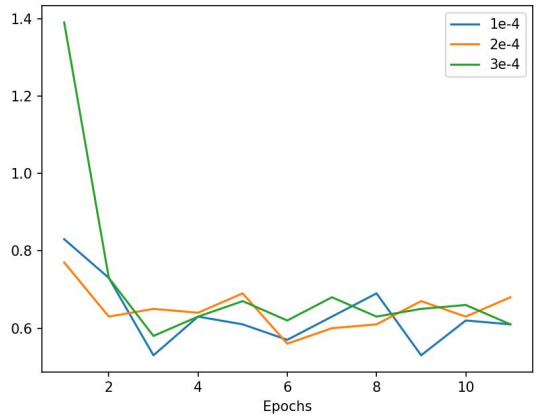
3.1 Improving GAN Network

Comparative experiments are conducted exclusively on the MNIST dataset to **investigate the impact of the learning rate on the effectiveness of the GAN (Generative Adversarial Network).** While keeping other

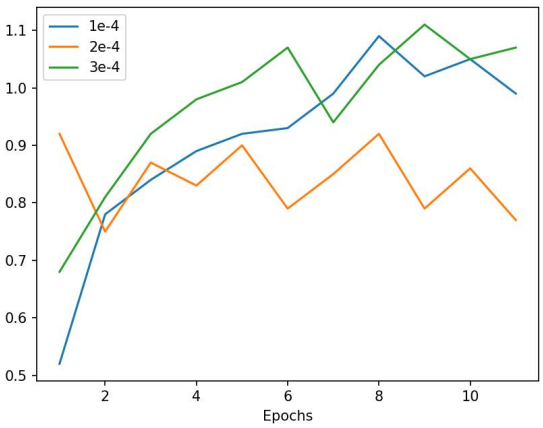
parameters constant, the learning rates are set to 1e-4, 2e-4, and 3e-4, respectively. The objective is to compare the G_LOSS, D_LOSS, and accuracy under different scenarios.



Figue 2: The Impact of Different Learning Rates on Accuracy



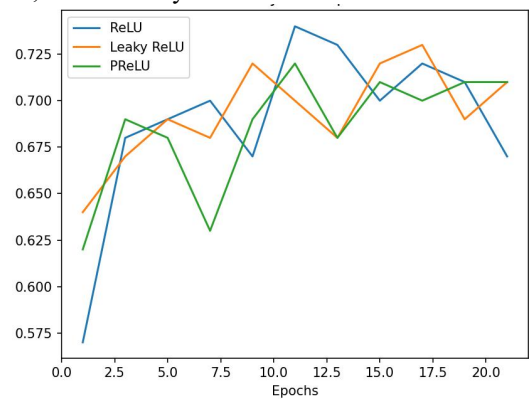
Figue 3: The Impact of Different Learning Rates on D_LOSS



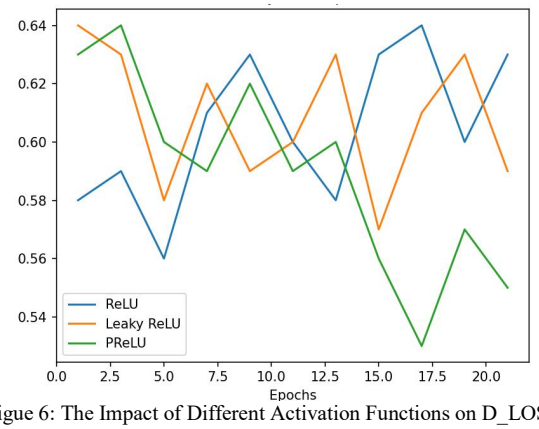
Figue 4: The Impact of Different Learning Rates on G_LOSS

Increasing the learning rate is advantageous for accelerating the convergence speed of the model, prompting it to learn data features and patterns more quickly. In these three sets of data, by increasing the learning rate to 3e-4, it is observed that the model achieves optimal performance more rapidly during the training process, showing slightly better results compared to smaller learning rates (1e-4 and 2e-4). This phenomenon suggests that increasing the learning rate may accelerate the model's convergence to some extent and achieve better results within the same number of training iterations.

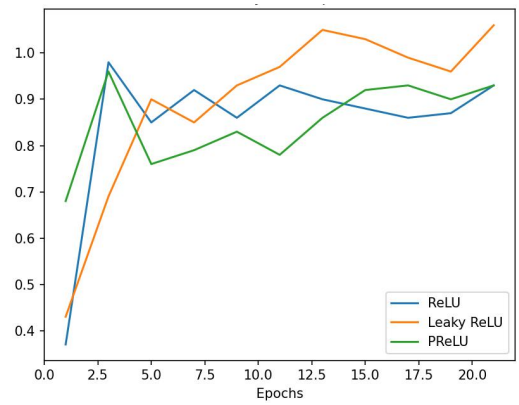
Next, we explore **the impact of different activation functions on the performance of the GAN.** While keeping other parameters constant, the activation functions are set to ReLU, Leaky ReLU, and Parametric ReLU (PReLU) respectively. The objective is to compare the G_LOSS, D_LOSS, and accuracy under different scenarios.



Figue 5: The Impact of Different Activation Functions on Accuracy



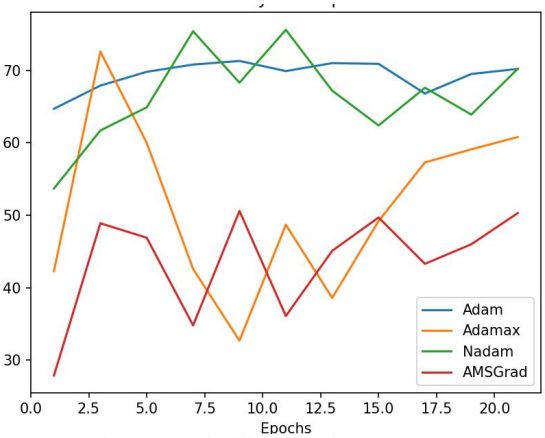
Figue 6: The Impact of Different Activation Functions on D_LOSS



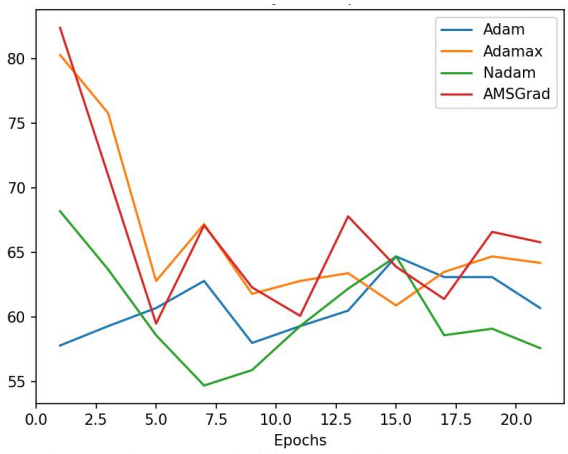
Figue 7: The Impact of Different Activation Functions on G_LOSS

From the results above, it appears that the Leaky ReLU activation function performs slightly better. The reason Leaky ReLU is more suitable for GAN networks lies in its introduction of a negative slope, allowing a small portion of negative values to pass through. This helps avoid the issue of neuron death that traditional ReLU may encounter with negative inputs. In GANs, the negative slope of Leaky ReLU allows gradients to propagate through the negative part during the backpropagation process, contributing to better handling of the complex data distribution between the generator and discriminator.

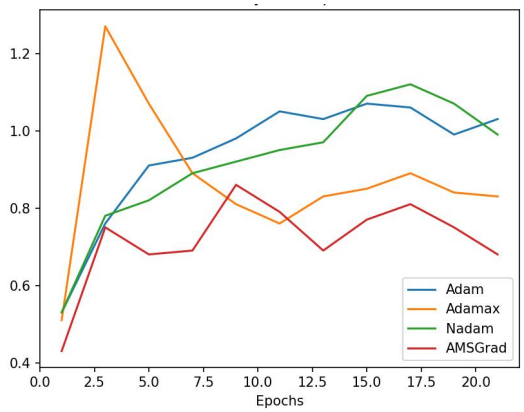
Next, we explore **the impact of different optimizers on the performance of the GAN.** While keeping other parameters constant, the optimizers are set to Adam, Adamax, Nadam, and AMSGrad, respectively. The goal is to compare G_LOSS, D_LOSS, and accuracy under different scenarios.



Figue 8: The Impact of Different Optimizers on Accuracy



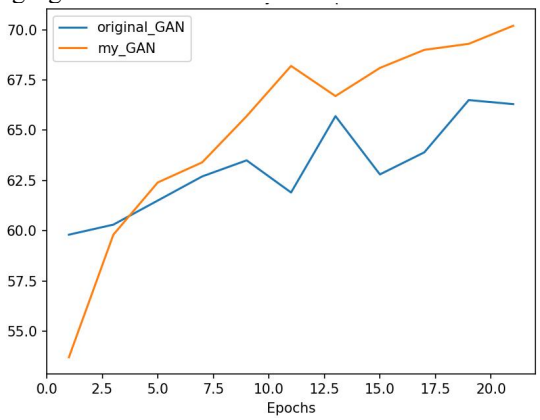
Figue 9: The Impact of Different Optimizers on D_LOSS



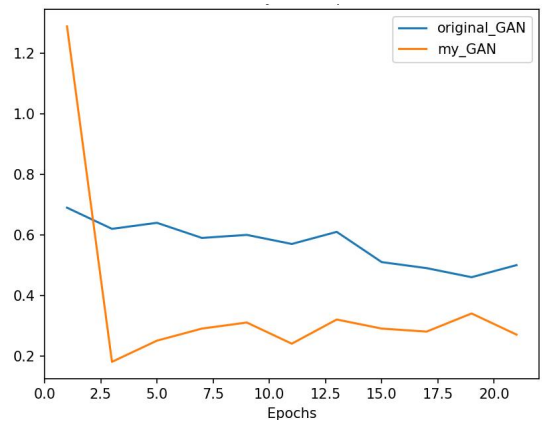
Figue 10: The Impact of Different Optimizers on G_LOSS

Adam and Nadam perform the best across various indicators, with Adamax showing slightly lower performance, and AMSGrad having the lowest average accuracy, significantly lower than the other algorithms.

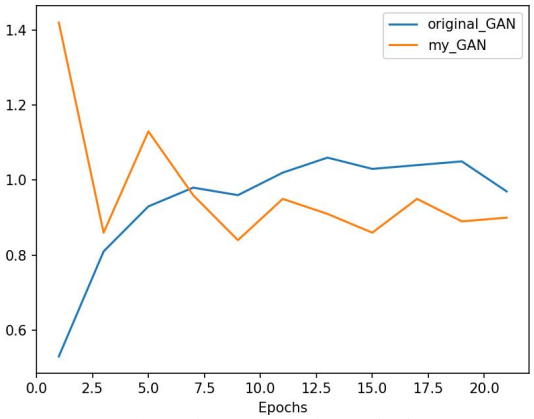
Based on the comprehensive results from the experiments, improvements are made to the original GAN model. **The changes include setting the learning rate to 3e-4, using Leaky ReLU as the activation function, and employing the Adam optimizer.** The comparison between the original and improved GAN models is shown in the following figures.



Figue 11: Comparison of Accuracy Before and After Improvement



Figue 12: Comparison of D_LOSS Before and After Improvement



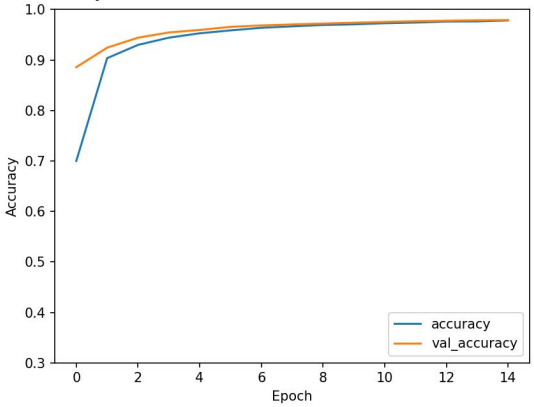
Figue 13: Comparison of G_LOSS Before and After Improvement

The performance in terms of accuracy and D_loss is significantly better than the original algorithm.

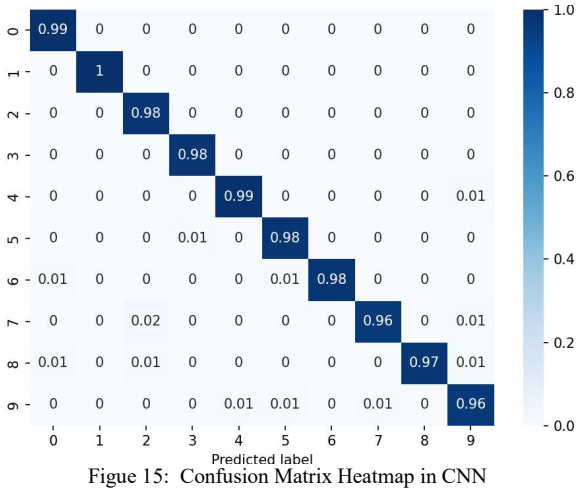
3.2 Comparative Experiments

In this section of the experiment, tests are conducted on two datasets, MNIST and CIFAR-10, with the main goal of comparing the performance of GAN and DCGAN. An improved version of the DCGAN algorithm is introduced, incorporating Wasserstein distance, spectral normalization, and noise input dimension estimation. These enhancements make the model easier to optimize, more stable, and better able to utilize limited small-sample data for training.

Initially, a simple CNN model is run, comprising two convolutional layers with ReLU activation functions, followed by max-pooling. Subsequently, there are three fully connected layers, with the first being a flattening layer, the second a dense layer with 128 units, and the last one a dense layer with 10 units as output using the Softmax activation function. Pixel values of training images are subtracted and then divided by 127.5 to normalize them within the range of [-1, 1]. Both training and testing labels undergo one-hot encoding conversion. The optimizer used is stochastic gradient descent, the loss function is categorical cross-entropy, and the metric is accuracy, achieving a final test accuracy of 98%.

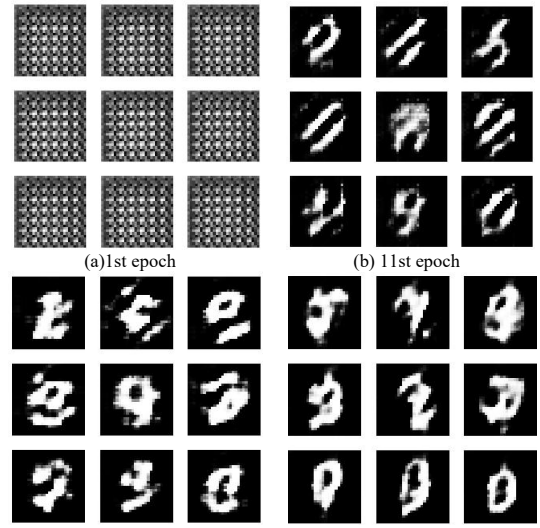


Figue 14: Accuracy Over Epoch in CNN

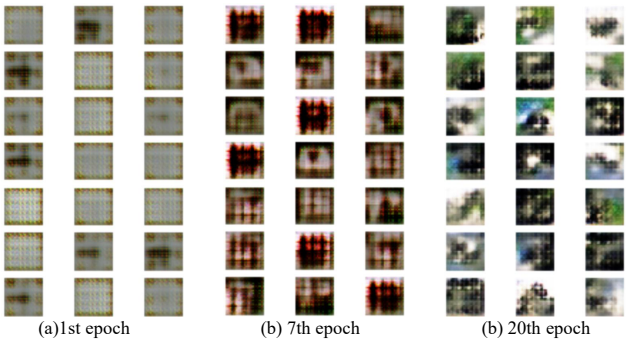


Figue 15: Confusion Matrix Heatmap in CNN

Here, both the generator and discriminator of GAN utilize deep convolutional neural networks. We employ checkpoints to store the GAN model, which can be later used for classification tasks. From the training results on the MNIST dataset, we observe that the generated images are initially random noise in the first few epochs. However, starting from the 10th epoch, these images gradually take shape and begin to resemble handwritten digits.

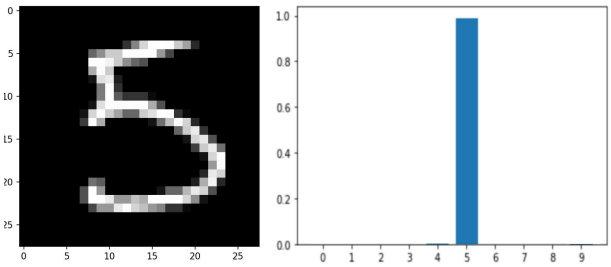

(a)1st epoch (b) 11st epoch (c) 25th epoch (d) 50th epoch

Figue 16: Generated Digits Evolution in GAN, MNIST dataset


(a)1st epoch (b) 7th epoch (b) 20th epoch

Figue 17: Generated Images Evolution in GAN, CIFAR-10 dataset

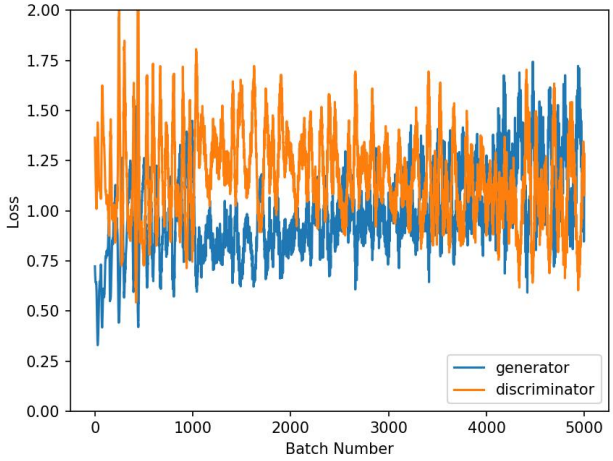
Visualizing the generated handwritten digits, the discriminator provides a reasonably accurate probability distribution for these images.



(a) A well-generated digit (b) Probability Distribution of (a)

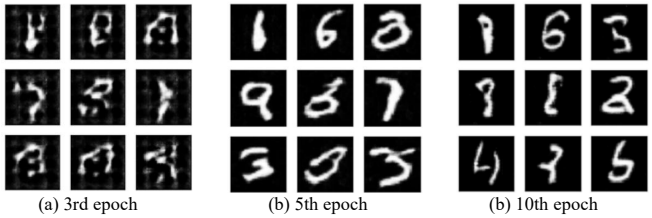
Figue 18: A Digit and Its Prediction

Below, we showcase the evolution of example digits and the changes in training losses for the generator and discriminator over time. Before the 30th epoch, the discriminator's loss is typically higher than the generator's loss. Afterward, the discriminator's loss decreases, while the generator's loss increases. Ultimately, their losses tend to converge.



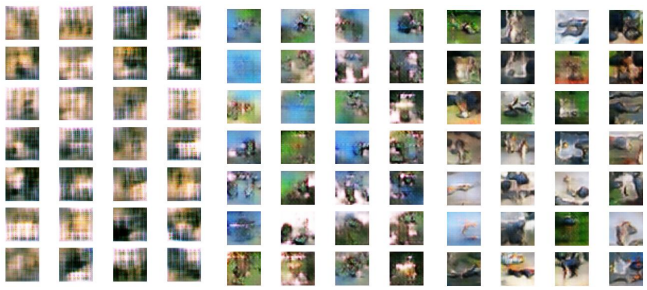
Figue 19: Training Loss Curves in GAN

Next, we run the improved DCGAN model, and the results on the MNIST and CIFAR-10 datasets are as follows.



(a) 3rd epoch (b) 5th epoch (b) 10th epoch

Figue 20: Generated Digits Evolution in DCGAN, MNIST dataset



(a) 3rd epoch (b) 8th epoch (b) 17th epoch

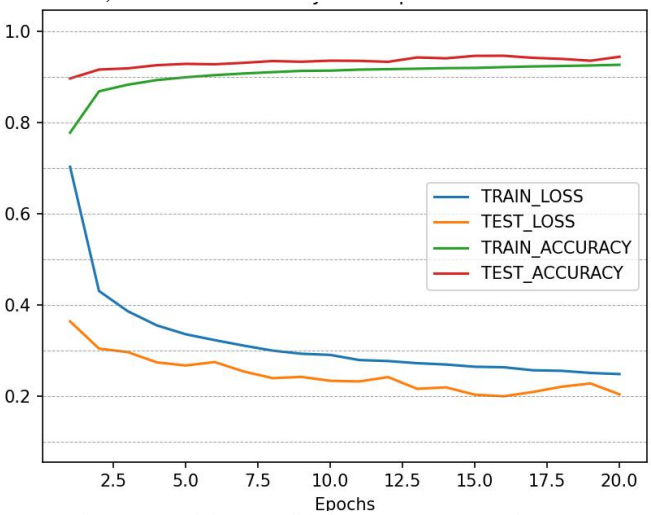
Figue 21: Generated Images Evolution in DCGAN, CIFAR-10 dataset

Clearly, compared to the regular GAN network, the improved DCGAN network achieves faster and clearer image generation with fewer epochs.

3.3 Transfer Learning and Fine-tuning

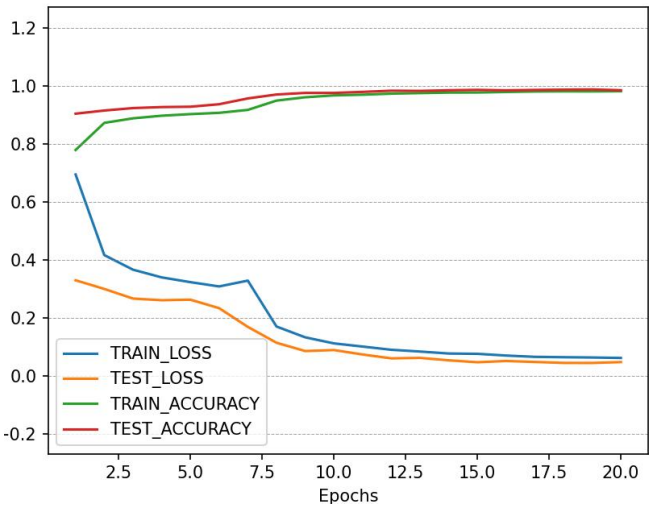
We continue to modify the discriminator of the DCGAN model, modifying the discriminator in our GAN model to a 10-way classifier. Specifically, we will run tests on the MNIST dataset. To achieve this, we remove the last dense layer of the original GAN discriminator, which is primarily used to distinguish between real and generated images.

Experiment 3.3.1: Transfer Learning Only - replacing the last dense layer with a trainable layer. The test accuracy can approach approximately 95%. (we will modify the discriminator in our GAN model to a 10-way classifier.)



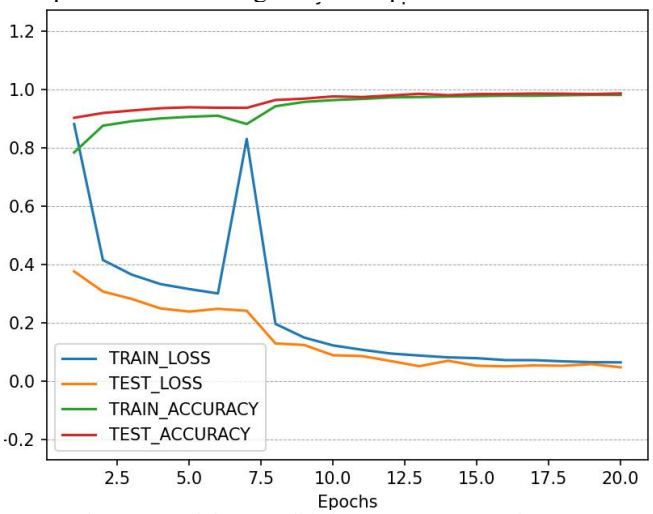
Figue 22: Training & Validation Accuracy & Loss in 3.3.1

Experiment 3.3.2: Transfer Learning and Fine-tuning - Building upon Experiment 1, fine-tuning is introduced. All layers are frozen for the first 6 epochs, and then unfrozen for the next 14 epochs, achieving a test accuracy of 98%.



Figue 23: Training & Validation Accuracy & Loss in 3.3.2

Experiment 3.3.3: Transfer Learning and Fine-tuning (Additional Layers) - Based on Experiment 3.3.2, two additional dense layers are added. Through experimentation, all layers except the last two added layers are frozen for the first 6 epochs, followed by unfreezing for the next 14 epochs. The test accuracy reaches 98.3%, demonstrating the best performance among the four experiments.



Figue 24: Training & Validation Accuracy & Loss in 3.3.3

Here is the hierarchical structure of the DCGAN network for Experiment 3.3.3.

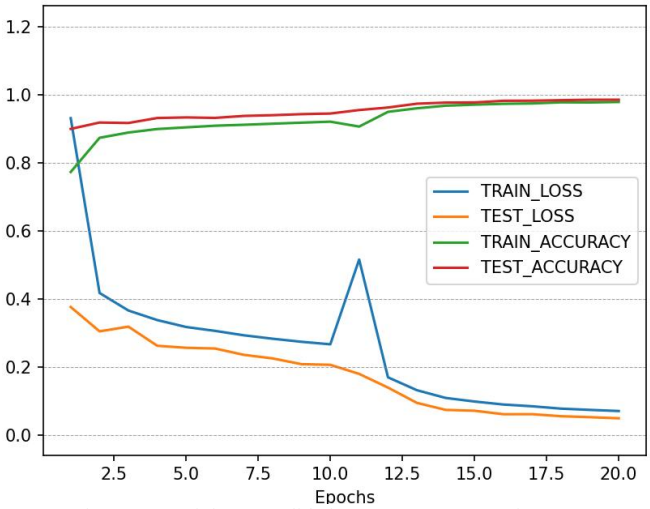
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 128)	3328
layer_normalization_5	(None, 14, 14, 128)	256
leaky_re_lu (LeakyReLU)	(None, 14, 14, 128)	0
dropout (Dropout)	(None, 14, 14, 128)	0
conv2d_1 (Conv2D)	(None, 7, 7, 256)	819456
layer_normalization_6	(None, 7, 7, 256)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 256)	0
dropout_1 (Dropout)	(None, 7, 7, 256)	0
conv2d_2 (Conv2D)	(None, 4, 4, 512)	2097664
layer_normalization_7	(None, 4, 4, 512)	1024
flatten (Flatten)	(None, 8192)	0
dense_1 (Dense)	(None, 1)	8193

Table 2: Description of the First DCGAN Model in Trial3

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 14, 14, 128)	3328
layer_normalization_5	(None, 14, 14, 128)	256
leaky_re_lu (LeakyReLU)	(None, 14, 14, 128)	0
dropout (Dropout)	(None, 14, 14, 128)	0
conv2d_1 (Conv2D)	(None, 7, 7, 256)	819456
layer_normalization_6	(None, 7, 7, 256)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 256)	0
dropout_1 (Dropout)	(None, 7, 7, 256)	0
conv2d_2 (Conv2D)	(None, 4, 4, 512)	2097664
layer_normalization_7	(None, 4, 4, 512)	1024
flatten (Flatten)	(None, 8192)	0
dense_2 (Dense)	(None, 128)	1048704
dense_3 (Dense)	(None, 10)	1290

Table 3: Description of the Second DCGAN Model in Trial3

Experiment 3.3.4: Transfer Learning and Fine-tuning (Change in Unfreezing Timing) - Building upon Experiment 3, the approach is altered to freeze all layers, except the last two added layers, for the first 10 epochs. Subsequently, they are unfrozen for the next 10 epochs, achieving a test accuracy of 97%.



Figue 25: Training & Validation Accuracy & Loss in 3.3.4

From the comprehensive analysis of the above experiments, it can be observed that the test accuracy exhibits a gradually increasing trend, especially reaching the highest point in Experiment 3.3.3. This indicates that adding two extra dense layers might allow the model to better adapt to image classification. The phenomenon of a sudden drop in accuracy followed by a subsequent rise may be attributed to the model undergoing fine-tuning across the entire network after unfreezing previously frozen layers. This phenomenon is commonly referred to as "warm-up" or "impact during unfreezing," where the performance may initially decline after unfreezing, but as training progresses, the model gradually adjusts to the new parameters of the unfrozen layers, ultimately demonstrating improved performance.

4. Conclusion

For the improvement of the GAN network, we adjusted the model's learning rate to 3e-4, used Leaky ReLU as the activation function, employed the Adam optimizer, and made appropriate adjustments to other structures and parameters. This led to a significant enhancement in performance, with a 4% increase in accuracy and improvement in D_loss on the MNIST dataset compared to the original algorithm.

Subsequently, **we enhanced the DCGAN algorithm and conducted comparative experiments** on the MNIST and CIFAR-10 datasets, introducing Wasserstein distance, spectral normalization, and noise input dimension estimation. The results indicated that the improved DCGAN model outperformed the regular GAN network significantly in terms of generation speed and clarity.

Ultimately, based on the conclusions drawn from the first two experiments, **we further improved the DCGAN model** by transforming its discriminator into a 10-class classifier. **Through transfer learning and fine-tuning strategies**, we optimized the model's performance, achieving an accuracy very close to that of the original CNN model (98%).

There are still several challenges, including the instability in the training process of GANs, especially the significant fluctuations in D loss and G loss. In the future, we might consider introducing additional convolutional operations and pooling techniques to enhance the quality of generated images. In fine-tuning, a common practice is to truncate the last layer of a pre-trained network and replace it with a new softmax layer. It is crucial to use a smaller learning rate during training and decide whether to freeze the weights of the pre-trained network's initial layers based on the dataset size.

Through this course experiment, we successfully improved the GAN and DCGAN models, achieving good performance in both image generation and classification tasks. We delved into the key factors of model fine-tuning and enhancements. This has laid a solid foundation for my future learning journey, and I am grateful for the guidance and efforts from the teacher and teaching assistants.

5. References

[1] Dixit, M.; Chen, S.; Gao, D.; Rasiwasia, N.; Vasconcelos, N. (2015): Scene classification with semantic fisher vectors. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2974-2983.

[2] Kim, H. J.; Lee, J. S.; Park, J. H. (2008): Dynamic hand gesture recognition using a CNN model with 3D receptive fields. IEEE Conference on Neural Networks and Signal Processing, pp. 14-19.

[3] Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D. et al. (2014): Generative adversarial nets. Advances in Neural Information Processing Systems, pp.2672-2680.

[4] Xudong Mao, Qing Li, Haoran Xie, Raymond Y.K. Lau, Zhen Wang, Stephen Paul Smolley; Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2794-2802

[5] Martin Arjovsky, Soumith Chintala, Léon Bottou Proceedings of the 34th International Conference on Machine Learning, PMLR 70:214-223, 2017.

[6] Zhao, Junbo, Michael Mathieu, and Yann LeCun. "Energy-based generative adversarial network." arXiv preprint arXiv:1609.03126 (2016).

[7] Mirza, Mehdi, and Simon Osindero. "Conditional generative adversarial nets." arXiv preprint arXiv:1411.1784 (2014).

[8] Odena, Augustus. "Semi-supervised learning with generative adversarial networks." arXiv preprint arXiv:1606.01583 (2016).

[9] Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.

[10] Karras, Tero, et al. "Progressive growing of gans for improved quality, stability, and variation." arXiv preprint arXiv:1710.10196 (2017).

[11] Zhang, Han, et al. "Self-attention generative adversarial networks." International conference on machine learning. PMLR, 2019.

[12] Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

[13] Pan, Sinno Jialin, and Qiang Yang. "A survey on transfer learning." IEEE Transactions on knowledge and data engineering 22.10 (2009): 1345-1359.

[14] GAN, Lan, et al. "Data augmentation method based on improved deep convolutional generative adversarial networks." Journal of Computer Applications 41.5 (2021): 1305.