Abstract

Synthetic Aperture Radar (SAR) is a type of radar system widely used primarily for remote sensing and imaging applications. Combining SAR images with machine learning to achieve advanced tasks (e.g., target recognition) is one of the hotspots for research. However, due to the high cost of acquiring, the size of SAR image dataset for training machine learning models is usually small, which might lead to unsatisfactory model performances. One potential method to overcome this issue is generating SAR image by Generative Adversarial Networks (GAN), to attain the purpose of data augmentation. This project evaluates the availability and performance in SAR image synthesis of six classic GANs, which are DCGAN, WGAN-GP, WGAN-DIV, SNGAN, SNGAN(Enhanced) and VAE-GAN. Experiments are conducted based on MSTAR dataset. The results indicate that, in the ten-category target recognition task, when the training samples are insufficient (500 images), images generated by SNGAN(Enhanced) can improve the recognition accuracy from 81.03% to 86.27%. When the training samples are extremely insufficient (100 images), WGAN-DIV can improve the recognition rate from 44.39% to 55.32%. The project validates the effectiveness of conducting SAR data augmentation via WGAN-GP, WGAN-DIV, SNGAN, SNGAN(Enhanced) and VAE-GAN, among which WGAN-DIV and SNGAN(Enhanced) are two of the optimal options, which possess both generation diversity and fidelity to real SAR image, under different training sample conditions.

Keywords: Generative Adversarial Network, GAN, Synthetic Aperture Radar, SAR, image synthesis, image generation, machine learning, generative deep learning

Acknowledgements

Prior to embarking on this project, the author possessed no knowledge in the domain of generative deep learning. Therefore, I would like to express my sincere appreciation to my first supervisor, **Prof. XXX**, along with his postgraduate student **XXX**, for their invaluable guidance throughout this project. Additionally, **Prof. XXX** provided significant support in my application for postgraduate programmes, for which I am deeply grateful. Throughout the four-year undergraduate study, I have acquired extensive engineering knowledge and experience. I would like to express my gratitude to both the *University of Glasgow* and the *University of Electronic Science and Technology of China*, whose contributions have made these accomplishments possible. I am grateful to my family and my friends. Their unwavering support has consistently served as a driving force behind the advancement of my academic pursuits.

Contents

Abstract 4
Acknowledgements 5
1 Introduction 7
2 Preliminaries 10
2.1 Deep Learning 10
2.1.1 Training Neural Network via Backpropagation 10
2.1.2 Convolution / Deconvolution Layer 11
2.1.3 Activation Functions 12
2.2 Generative Adversarial Network (GAN) 13
2.2.1 What is GAN? 13
2.2.2 Pros and Cons of GAN 14
2.3 MSTAR Dataset 16
3 Implemented GANs and Background Theory 17
3.1 DCGAN 17
3.2 WGAN-GP 19
3.3 WGAN-DIV 20
3.4 SNGAN 22
3.5 SNGAN (Enhanced) 23
3.6 VAE-GAN 25
3.7 Algorithm for Training GANs 28
4 Model Performance Evaluation and Discussion 29
4.1 **Evaluation via Classification Performance** 29
4.1.1 Under Insufficient Training Samples Condition 30
4.1.2 Under Extremely Insufficient Training Samples Condition 32
4.1.3 Under Sufficient Training Samples Condition 33
4.2 Evaluation via Visual Effects and Statistics 36
4.3 Evaluation via Training Overhead 38
5 Conclusions 39
References 40
A Appendices 43
A.1 Supplementary Figures 43
A.2 Detailed Description of Implemented GAN Structures 50
A.3 Equipment, Platforms and Tools Involved 55

1 Introduction

Synthetic Aperture Radar, SAR, is a type of imaging radar that is potential to achieve ultra-high resolution [1]. This technology enables the acquisition of high-resolution images that are independent of weather conditions and available day and night [1]. The applications of SAR encompass a broad range of fields, including geoscience, research on climate change, environmental system monitoring, 2D/3D mapping, change detection, and even planetary exploration [1].

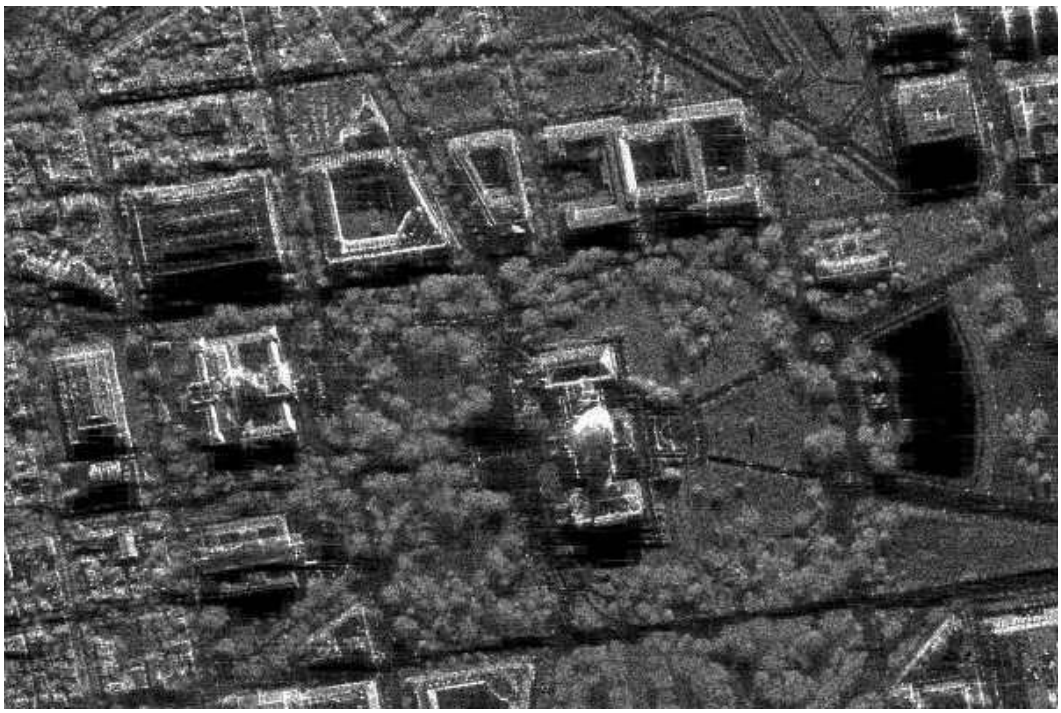


Figure 1 A SAR image of Washington, D.C., figure source: **Sandia National Laboratories**

As a type of remote sensing imagery, SAR images offer significant potential for integration with machine learning, particularly with deep learning [2]. Specific examples of combining SAR images with machine learning includes:

1. **Target Recognition**: Identifying the category of targets depicted in an SAR image [2].
2. **Object Detection**: Tracking the position of objects in SAR monitoring system [2].
3. **Semantic Segmentation**: Segmenting the contours of objects in SAR images [2].

A sufficient condition for a well-trained deep learning model is an adequate amount of training data. Take image recognition as an example, even the aged MNIST dataset [3] is composed of 60,000 training images and 10000 test images in total, divided into ten categories, let alone some of the modern datasets that are made up of tens of millions of samples. However, in comparison, the size of SAR datasets is considerably smaller, typically consisting of only a few thousand images. The main reason for this problem is largely due to the high cost involved in acquiring SAR images. From [4], “Acquiring Geocento's synthetic aperture radar (SAR) satellite data image costs 3,300 U.S. dollars

per scene for an image with a resolution of less than one meter. This cost is comparatively higher than the costs associated with optical satellite images commonly utilized today."

One possible solution for the lack of data problem in SAR images is data augmentation, a technique used to increase the size of a dataset by applying various transformations or manipulations to the original data. There has been an abundant amount of augmentation methods proposed for SAR images, including but not least to Geometric methods (i.e., rotation, shift, etc.), Linear Synthesis, and Computer-Aided Design (CAD) models [5-6].

In deep learning, there are models that are designed to synthesis data, especially images. Generative Adversarial Network, GAN, is a type of model for image synthesis. The idea of GAN was firstly proposed in [7] in 2013, and later became one of the most influential generative deep learning models. The aim of GAN is to generate images that have high fidelity to the images from training dataset.

After the first GAN, Vanilla-GAN, was invented, numerous related research was conducted to improve the generation ability and training stability of GAN. DCGAN [8] replaces the fully connected layers in Vanilla-GAN with convolution layers. WGAN-GP [9] uses Wasserstein Distance as the measurement gauge, instead of the Jenson-Shannon Divergence in Vanilla-GAN, to calculate the loss function, and adds a gradient penalty term to ensure the model stability. WGAN-DIV [10] expands Wasserstein Distance to Wasserstein Divergence, which improves the mathematical interpretability of WGAN. SNGAN [11] applies Spectral Normalization technique to maintain the stability of training process. GAN has potential to combined with other deep learning models. One example is VAE-GAN [12], which incorporates the advantages of both GAN and Variational Auto-Encoder, VAE, to achieve better performance.

It has been demonstrated that GANs are capable of synthesizing complex images. The application of GANs for generating SAR images is one of many research directions [2]. Due to the scarcity and rarity of SAR images, generating SAR images by GAN requires training GAN based on limited amount of data. Hence, the nature of this task is to train a deep learning model with an insufficient number of training samples [5].

In this project, the author tests the ability of six different GAN based models, in augmenting (generating) SAR images. They are **DCGAN**, **WGAN-GP**, **WGAN-DIV**, **SNGAN**, **SNGAN (Enhanced)** and **VAE-GAN**. The author mainly evaluates their performances under three conditions:

1. The training samples are insufficient.
2. The training samples are extremely insufficient.
3. The training samples are relatively sufficient.

Before this project, there have been several research focusing on using GAN to generate SAR images with high fidelity, including but not least to [5-6] [13-15]. These paper either use a large amount of

SAR images to train GAN or apply certain mechanisms to filter out some of the synthetic SAR images. The aim of this project is to assess the SAR image generation capacity of the above models under different training sample sizes, by unfiltered generated SAR images. The author intends to explore the availability of using each model to conduct data augmentation under different conditions and finds the best model for each condition.

The remainder of report are organized as follows. **Section 2** introduces the knowledge required to read this report and the dataset involved in this project. **Section 3** demonstrates the architectures and working principles of six GAN based models, including the theories behind each model. **Section 4** assesses the effectiveness of each GAN in SAR image synthesis from different perspectives. **Section 5** summarizes the results of this project.

2 Preliminaries

In this section, the author intends to clarify the concepts of **Deep Learning (DL)** and **Generative Adversarial Network (GAN)** to readers who have limited related knowledge about this field. The dataset involved is described. The imaging principle of synthetic aperture radar (SAR) will not be discussed, as it is not the focus of this project. If the reader is interested in the technical details of SAR, please refer to [1].

2.1 Deep Learning

As the most in-demand field in contemporary artificial intelligence (AI), deep learning focuses on creating algorithms that enable computers to learn from and process large amounts of data, often in the form of neural networks [16]. These algorithms are designed to mimic the way the human brain processes information, which is why they are often referred to as artificial neural networks. The very first artificial neural network, by the name of "Perceptron", is implemented in 1958 by Frank Rosenblatt [17]. In the early years, the structure of neural network is simple, which contains only 2 to 3 layers with a few thousand trainable parameters, due to the limited computational resource. The complexity of neural network has been consistently growing to achieve increasingly complicated task. The milestone conversational AI ChatGPT has 175 billion parameters (Version-3.5).

2.1.1 Training Neural Network via Backpropagation

Neural networks are designed to fit a "function" of a given task. This function can be either concrete or abstract [16]. Training a neural network is the process of adjusting its internal parameters so that it can learn to make accurate predictions or perform specific tasks based on input data. Here is the standard process of training a neural network:

1. **Feedforward**: Input data is passed through the network layer by layer, with each neuron applying mathematical operations to transform the input [16]. This process generates a final output or prediction.
2. **Compute Loss**: A loss function is utilized to quantify the disparity between the target values and the predictions inferred by the network [16-17]. In most cases, the purpose of training is to minimize the value of loss function, implying that the network makes correct predictions [16].
3. **Backpropagation**: The backpropagation algorithm calculates the gradients (i.e., the partial derivatives) of the loss function with respect to each parameter in the network [16-18]. This process helps to determine how much each weight and bias should be adjusted to reduce the overall loss [16].

4. **Update Parameters**: The calculated gradients are used to update the network's weights and biases, typically using an optimization algorithm (optimizer) like gradient descent or one of its variants [16-17]. The optimization algorithm adjusts the parameters in the direction that minimizes the loss function [16].

5. **Iterate**: Steps 1 to 4 are repeated for multiple iterations, usually called epochs, until the network converges to an optimal set of parameters, or a predetermined stopping criterion is met [16]. During each epoch, the network is usually trained on multiple batches of data, with parameter updates occurring after each batch [16].

After the training process is complete, the neural network should be able to make accurate predictions or perform the desired task effectively on new, unseen data. Apart from the above method, a neural network can be trained via genetic algorithms or reinforcement learning techniques [16].

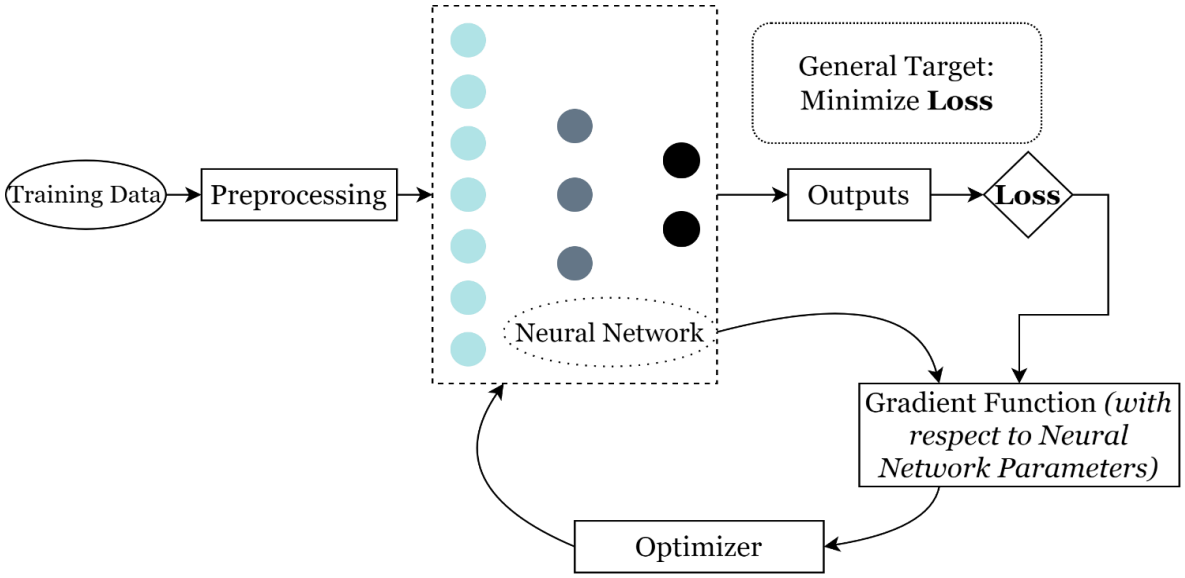


Figure 2 Standard process of training a neural network

2.1.2 Convolution / Deconvolution Layer

The original neural network is constructed on fully connected layer. Later, other types of network layer are invented to achieve higher model performance, typical ones include (de)convolution layers, recurrent layers, and attention blocks [16]. Convolution layer and deconvolution layer are two of the key components in all the GAN based models implemented in this project. It is proved by [8] that GAN based on convolution operations is superior to GAN based on fully connected layers, and by a very large margin. This fact can be approved by the author's early experiment, which is recorded in the midterm report of this project.

For Convolution Layer, given an input tensor **x** (with height **Hx**, width **Wx** and channel **C**), convolutional kernel tensor **w**, stride **s**, padding size **p**, the output **y** (with height **Hy**, width **Wy** and channel **K**) can be calculated as:

$$y_{i,j,k} = \sum_{c=1}^{C} \sum_{r=1}^{H_w} \sum_{l=1}^{W_w} w_{r,l,c,k} x_{(i-1)s+r+p,(j-1)s+l+p,c} \tag{1}$$

For Deconvolution (also known as Transposed Convolution) Layer, given an input tensor **x** (with height **Hx**, width **Wx** and channel **C**), convolutional kernel tensor **w**, stride **s**, padding size **p**, the output **y** (with height **Hy**, width **Wy** and channel **K**) can be calculated as:

$$y_{(i-1)s+r+p,(j-1)s+l+p,c} = \sum_{k=1}^{K} \sum_{r=1}^{H_w} \sum_{l=1}^{W_w} w_{r,l,k,c} x_{i,j,k} \tag{2}$$

The reader may find visual illustration of these two layers easier to understand:

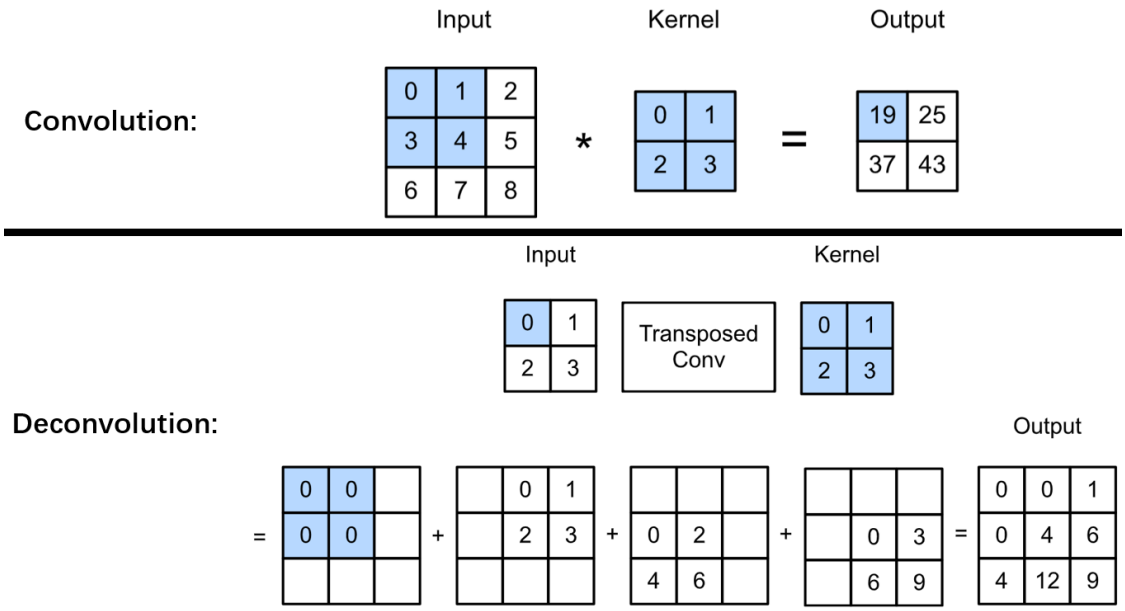


Figure 3 Convolution / Deconvolution operation, figure source: [16]

The key takeaway messages: Convolution layer is effective in down-sampling problems (e.g., classification). Deconvolution layer is effective in up-sampling problems (e.g., generation, denoising, …). Hence, they have been widely applied in the Discriminator and Generator of GAN, respectively.

2.1.3 Activation Functions

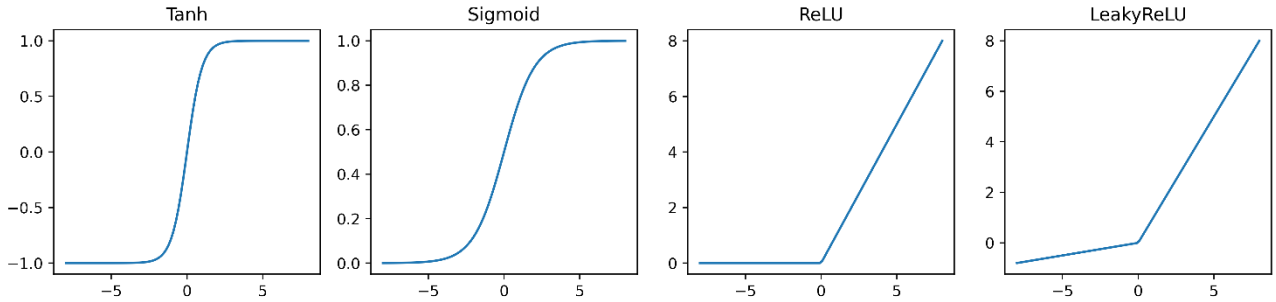


Figure 4 Activation functions involved in this project

Activation functions usually appears between two layers of a neural network [16]. The primary objective of activation functions is to offer a mechanism for a neural network to decide which neurons

should be activated (i.e., have non-zero outputs) in response to the input signal [16]. Non-linearity is introduced by activation functions, of which, the absence may lead to significant performance drop [16]. As shown in **Figure 4**, Tanh is used to mapping inputs to an output region of [-1, 1], which is usually added to the last layer of a Generator of GAN; Sigmoid is used to mapping inputs to [0, 1], usually applied to the final layer of a Discriminator of GAN; while ReLU (short for Rectified Linear Unit) and LeakyReLU are used as activations for intermediate layers.

$$\mathrm{Tanh}(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3}$$

$$\mathrm{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

$$\mathrm{ReLU}(x) = \max(x, 0) \tag{5}$$

$$\mathrm{LeakyReLU}(x) = \max(x, \alpha x)\,, \alpha \ll 1 \tag{6}$$

2.2 Generative Adversarial Network (GAN)

For readers who has no prior knowledge about Generative Adversarial Network (GAN), you may find this part useful.

2.2.1 What is GAN?

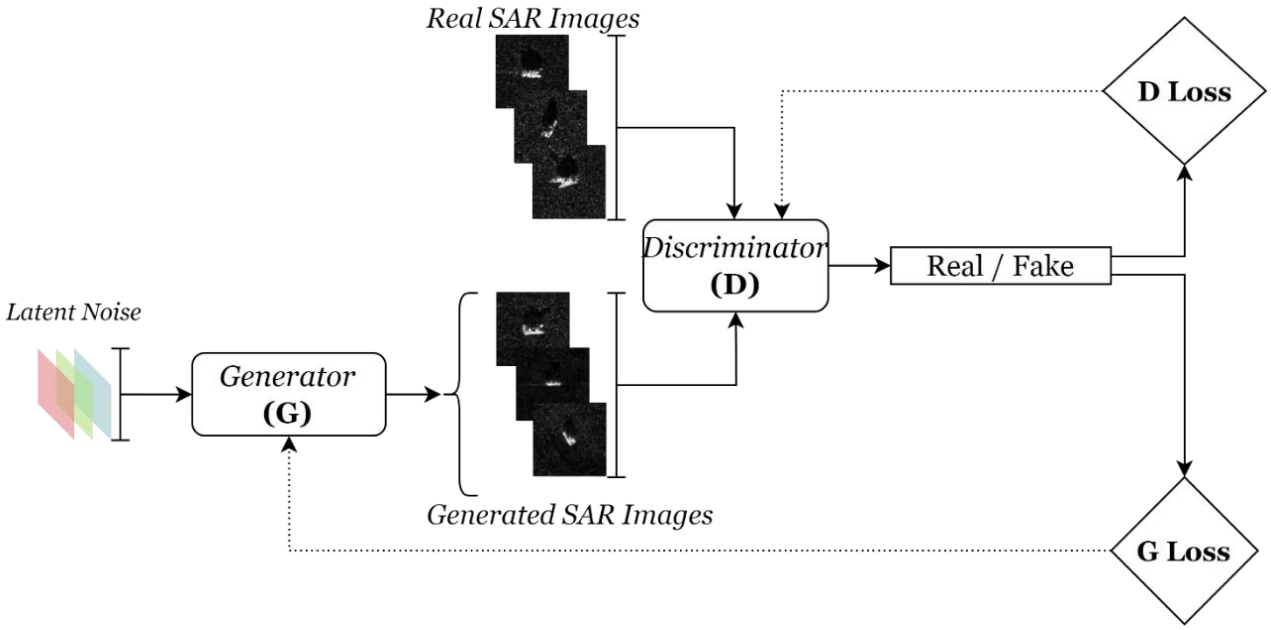


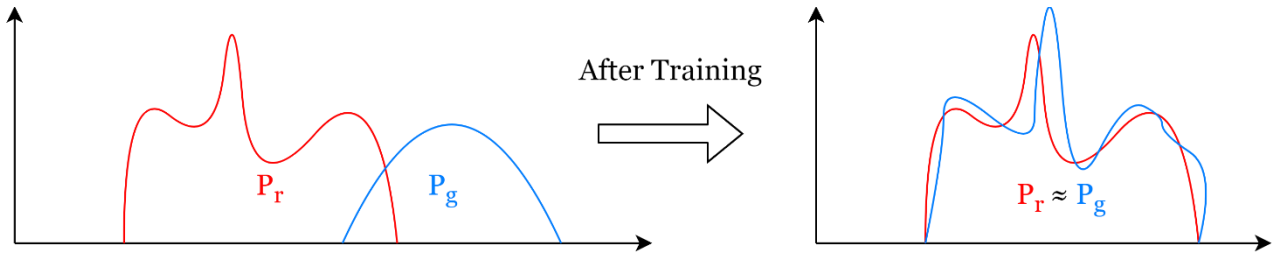
Figure 5 The workflow of GAN in image synthesis

GAN is the abbreviation term for Generative Adversarial Network(s). It was first introduced by Ian Goodfellow et al. [7] in 2013 and soon became the most dominant generative deep learning method of all time. In the field of image processing, GAN can be applied in image segmentation, image style transfer, text-to-image generation, etc. During the evolution of GANs, the fundamental problem that researchers put efforts on is image synthesis, which can be further described as: Generating images

that are similar to the training images. This problem is exactly what is focused on this project, in which the training images are SAR images.

A GAN has two equally important components: a **Discriminator (D)** and a **Generator (G)**. Training a GAN is training two neural networks simultaneously. From a simplified perspective to explain how GAN works, that is, the Generator creates new, synthetic data that is meant to resemble real data, while the Discriminator tries to determine which data is real and which is synthetic. These two networks are trained simultaneously, with the Generator trying to produce data that can fool the Discriminator [7]. And the Discriminator trying to correctly identify the real data from the synthetic data. Ideally, the result is a Generator that can produce data that is similar in distribution to the real data.

As shown in **Figure 5**, the input of Generator is latent noises, which are usually sampled from standard Gaussian distribution $\mathcal{N}(0,1)$ [7][32]. We want the output of Generator as close to the real data as possible. From a high-level perspective, the nature of Generator is to map a noise distribution onto a data distribution, while the nature of Discriminator is to measure the distance between the data distribution outputted by the Generator and the distribution of real data. Generator then tries to minimize the distance calculated from Discriminator. The ideal result is that the data distribution of Generator is very close to that of real data [7].



P_r is the distribution of real data, P_g is the distribution of data generated by **Generator**

Figure 6 The nature of GAN.

It is a common prejudice that Generator is more important than Discriminator in GAN. If the reader regards the Discriminator as a "distribution measurement gauge", instead of a "discriminator", you will have a better understanding of GAN and why it works. It is worth mentioning that most significant improvements in GANs have been achieved through modifications to the structure of the Discriminator.

2.2.2 Pros and Cons of GAN

The idea of GAN is revolutionary, it introduces adversarial training to deep learning. Compared to other generative deep learning models, such as autoregressive models and denoising diffusions, the structure of GAN is easy to implement. The training cost and the sampling (mapping an input noise

to a synthetic data) cost of GAN is relatively low, meanwhile, the generated data maintains a relatively high degree of fidelity.

On the other hand, there are some thorny issues with GANs:

1 Trade-off is of great importance to GAN since two networks are trained together. A stronger Generator or Discriminator may lead to convergence failure [28].

2 The training of GANs exhibits an overall lack of regularity, requiring real-time monitoring during the training process. Early stop [16] is often required.

3 Mode collapse, which is an everlasting problem in GANs (even the state-of-the-art ones), can be described as a lack of diversity in Generator's generated data [28]. This is the most common problem that are encountered in this project. In **Figure 7**, assume there are five clusters of real data, after training, if mode collapse does not occur, GAN will be able to generate data that are similar to all five clusters. If mode collapse occurs, the generated data will not cover all types of training data. In the worst case, Generator outputs only one cluster of data (**Figure 7(c)**) regardless of how the input noise varies. Mode collapse is the key reason that the generation diversity of GAN is no better than other deep generative models.

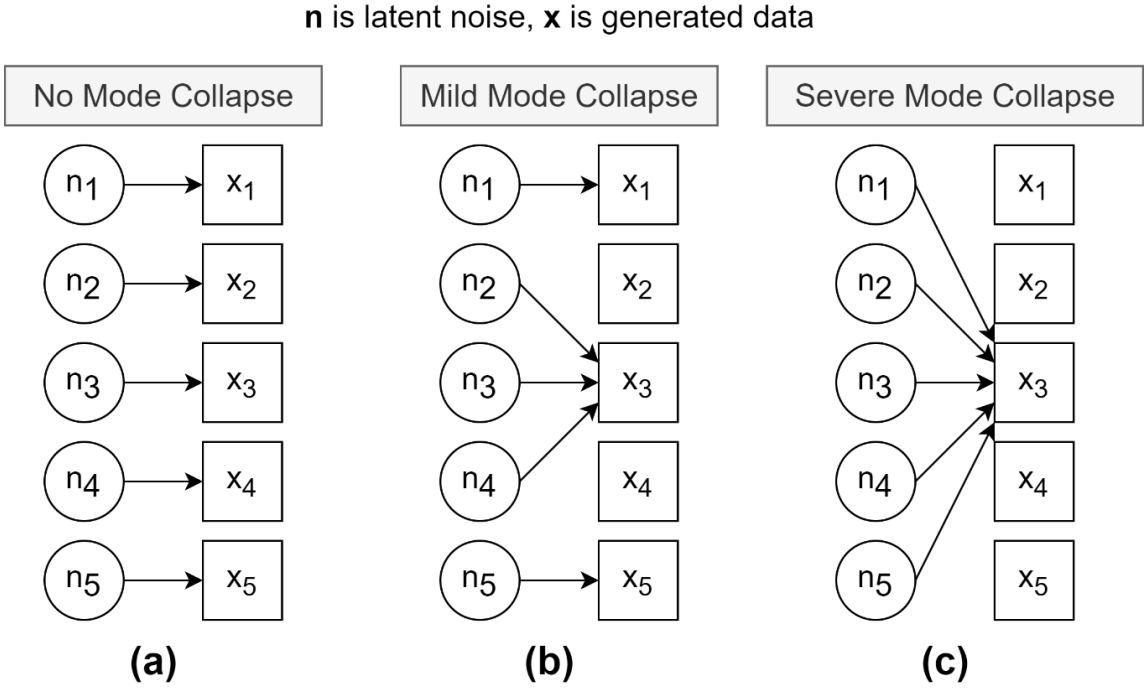


Figure 7 Mode collapse in GAN

4 Randomness of generated samples is a common issue in almost every deep generative model, including GAN. Due to the input of Generator, the latent noise, is randomly sampled from a prior distribution, the output of Generator is uncontrollable [7][28][32]. One cannot expect Generator to output a specific image by a certain input noise. The randomness, along with mode collapse issue, may cause a lack of diversity problem in the samples generated by GAN.

2.3 MSTAR Dataset

Published by US Defence Advanced Research Project Agency and Air Force Research Laboratory, the dataset involved in this project is MSTAR, short for Moving and Stationary Target Acquisition and Recognition. MSTAR consists of 10 classes of military vehicles, divided into training images (total 2746 images) and test images (total 2426 images). The training images are captured under depression angle 17° and test images are captured under 15°. The number of each class of vehicle is shown in **Table 1**. The images are in grayscale format (one-channel images). For convenience and consistency of this project, all SAR images in MSTAR dataset are cropped into the size of 128 (Height) × 128 (Width), meaning that each image contains 16384 valid pixels. For each class of vehicle, MSTAR contains its imagery of almost all azimuths, from 0° to 360°. **Figure 8** only shows one image out of every category of a certain azimuth. For more SAR image examples in MSTAR dataset, please refer to **Appx Figure 1**. All the GANs in this project are trained with the trainset or its subset.

Table 1 The distribution of data in MSTAR dataset

Name of Class	**Train Dataset** (17°)	**Test Dataset** (15°)
BMP2	232	196
T72	232	196
BTR70	233	196
BTR60	256	195
BRDM2	298	274
2S1	299	274
D7	299	274
T62	299	273
ZIL131	299	274
ZSU234	299	274
Total	**2746**	**2426**

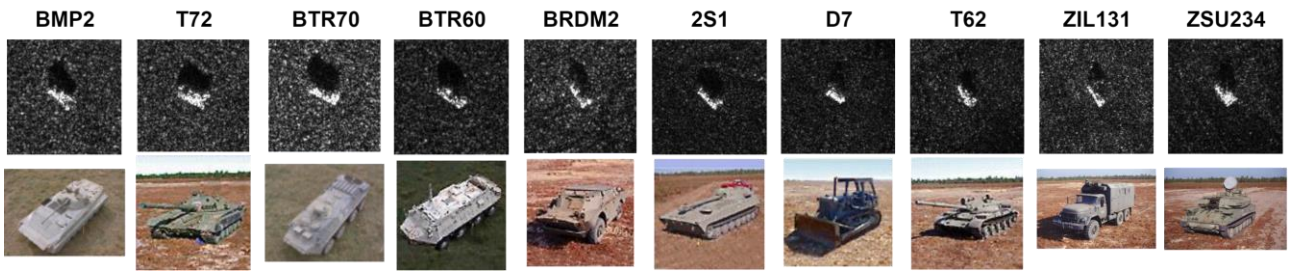


Figure 8 A glance of ten categories of vehicles in MSTAR dataset

3 Implemented GANs and Background Theory

The author has implemented 6 different types of Generative Adversarial Networks and evaluate each performance of them on generating SAR images in this project. In this section, the author intends to introduce the architecture of these GANs and their related theory. In the original paper [7], **Vanilla-GAN** (the first proposed GAN) is constructed on fully connected layers. In the midterm report of this project, the experiments have indicated that convolution-based GAN is better. Hence, all GANs in this project are convolutional neural networks.

3.1 DCGAN

DCGAN is the abbreviation for **Deep Convolutional GAN**. First proposed in [8], DCGAN only replace the fully connected layers with convolution and deconvolution layers, as compared to Vanilla-GAN, but the improvement in performance is significant.

In **section 2.1.1**, it can be concluded that choose an appropriate loss function for a neural network is critical. The loss function for DCGAN is identical to Vanilla-GAN. Recall in the previous section, the author mentions that Discriminator is the gauge to measure the distance between the distribution outputted by Generator and the distribution of real data [19]. Therefore, the loss function of nearly all types of GANs are calculated according to the outputs of Discriminator. The derivation of the loss functions in the original paper [7] is complex, but these loss functions can be understood intuitively.

Now, suppose there are n samples in the real dataset: $\boldsymbol{x} = x_1, x_2, x_3, \dots, x_n, \boldsymbol{x} \sim P_r$, where P_r represents the distribution of real data. $\boldsymbol{x} \sim P_r$ means the data $\boldsymbol{x}$ is sampling from distribution P_r. Suppose there are n noise samples sampled from a noise distribution: $\boldsymbol{z} = z_1, z_2, z_3, \dots, z_n, \boldsymbol{z} \sim P_{noise}$. Input the noise samples to the Generator, the generated data distribution, or the "fake" data distribution P_g can be got: $\boldsymbol{x} = G(z_1), G(z_2), G(z_3), \dots, G(z_n) = x_1, x_2, x_3, \dots, x_n, \boldsymbol{x} \sim P_g$, where G is Generator. There are two types of outputs for Discriminator, where D is Discriminator:

1. The input is real data:

$$D_{\boldsymbol{x} \sim P_r}(\boldsymbol{x}) \tag{7}$$

2. The input is fake data:

$$D_{\boldsymbol{x} \sim P_g}(\boldsymbol{x}), \text{equivalent to } D_{\boldsymbol{z} \sim P_{noise}}(G(\boldsymbol{z})) \tag{8}$$

(7) can be understood as "The position of real data in the Discriminator's metric" and (8) can be regarded as "The position of fake (generated) data in the Discriminator's metric". Now suppose that we want the Discriminator to output "1" if a real data is input and output "0" if a fake data is input. Hence, in order that Discriminator is powerful enough to distinguish fake data from real data, we want to:

$$Minimize\ \text{Distance}\left[D_{\boldsymbol{x}\sim P_r}(\boldsymbol{x}), 1\right] + \text{Distance}\left[D_{\boldsymbol{x}\sim P_g}(\boldsymbol{x}), 0\right] \tag{9}$$

In order that Generator can produce data that Discriminator can misjudge its authenticity, we want to:

$$Minimize\ \text{Distance}\left[D_{\boldsymbol{x}\sim P_g}(\boldsymbol{x}), 1\right] \tag{10}$$

Distance is a function that measures the distance between the outputs of Discriminator and target labels ('1' or '0'). If training Discriminator and Generator based on (9)(10), respectively, adversarial learning can be achieved, in which case, GAN will be function as expected. In practical, Distance function is chosen to be binary cross-entropy (BCE):

$$\text{BCE}(\boldsymbol{a}, \boldsymbol{b}) = -[\boldsymbol{b} \cdot log\boldsymbol{a} + (1 - \boldsymbol{a}) \cdot \log(1 - \boldsymbol{b})] \tag{11}$$

After proper transformation, the loss functions of DCGAN (9)(10) are written in expectation form:

$$\textbf{Discriminator}: Minimize\ -\mathbb{E}_{\boldsymbol{x}\sim P_r}[log\ D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim P_g}\left[log\left(1 - D(\boldsymbol{x})\right)\right] \tag{12}$$

$$\textbf{Generator}: Minimize\ -\mathbb{E}_{\boldsymbol{x}\sim P_g}[log\ D(\boldsymbol{x})] \tag{13}$$

The loss for Generator (13) only has one term since Generator cannot make influence on the output of Discriminator when the input is real data.

(References for equations and theories in **Section 3.1**: [7][8][19])

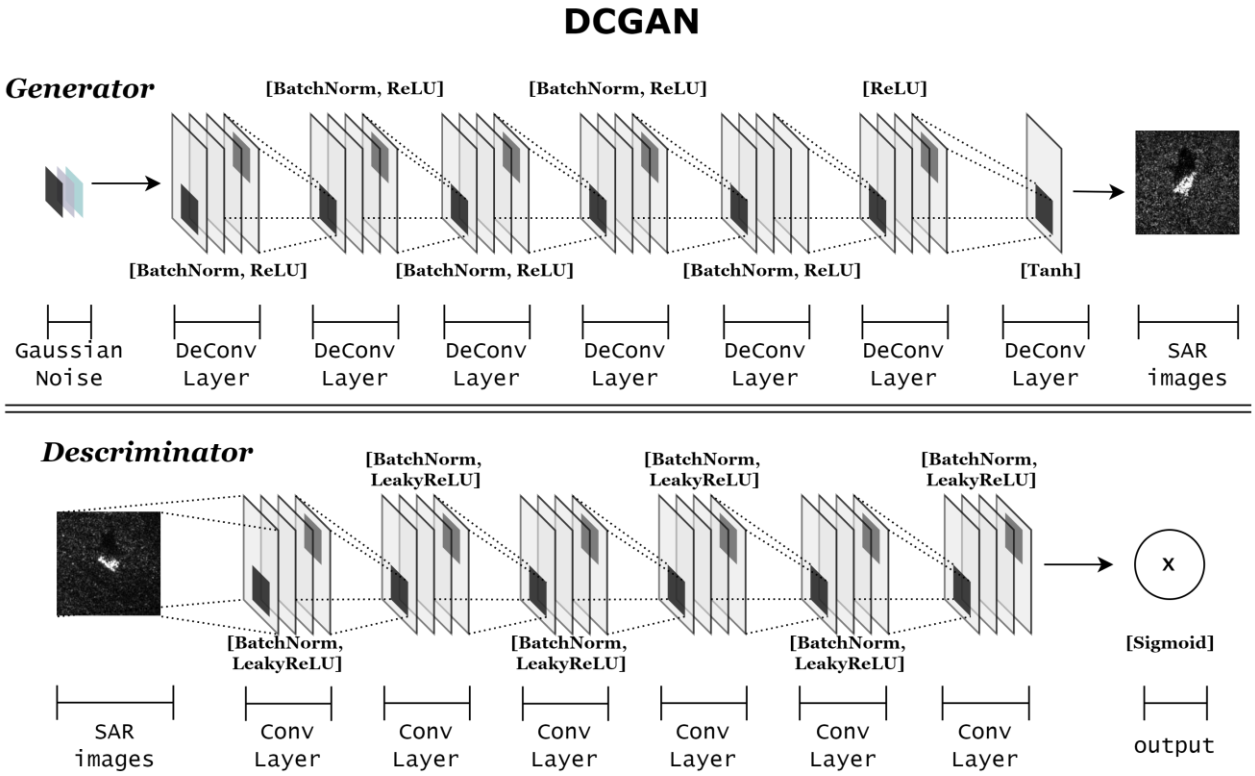


Figure 9 Visualisation of DCGAN structure

Figure 9 demonstrates the DCGAN architecture of generating SAR images. Both Generator and Discriminator are full convolutional neural networks. For detailed structure, please refer to **Appx Table 1**.

3.2 WGAN-GP

WGAN-GP is short for **Wasserstein GAN with Gradient Penalty** [9], which is an improved version of Wasserstein GAN (WGAN) [20]. WGAN uses different loss functions as compared to DCGAN. The distance calculation gauge is switched to Wasserstein Distance, or Earth-Mover Distance, which is a concept in Optimal Transport Theory.

Intuitively analysing, in (9) and (10), the objective function for DCGAN, we apply two auxiliary labels: "1" for real data, "0" for fake data, to help us "locate" the position where real data and fake data should be in the Discriminator's metric. GANs based on Wasserstein Distance consider the above process unnecessary. In WGANs, the two outputs of Discriminator are directly subtracted:

$$D_{\boldsymbol{x}\sim P_r}(\boldsymbol{x}) - D_{\boldsymbol{x}\sim P_g}(\boldsymbol{x}) \tag{14}$$

It is easy to understand that (14) can be roughly regarded as the distance between real data distribution and fake data distribution, measured by Discriminator. The Discriminator's role is to distinguish the fake data from real data. Therefore, to constantly improve the performance of Discriminator during training process, we need to maximize (14) during the training of Discriminator:

$$Maximize\ D_{\boldsymbol{x}\sim P_r}(\boldsymbol{x}) - D_{\boldsymbol{x}\sim P_g}(\boldsymbol{x}) \tag{15}$$

For Generator, intuitively speaking, it needs to conduct optimization in the opposite direction, as compared to Discriminator, under which condition adversarial learning is achieved. Since Generator has no influence on $D_{\boldsymbol{x}\sim P_r}(\boldsymbol{x})$, the objective for Generator has only one term:

$$Minimize\ -D_{\boldsymbol{x}\sim P_g}(\boldsymbol{x}) \tag{16}$$

In mathematical expectation form, the loss functions for WGAN are:

$$\textbf{Discriminator}: Maxmize\ \mathbb{E}_{\boldsymbol{x}\sim P_r \|D\|_L\leq 1}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim P_g \|D\|_L\leq 1}[D(\boldsymbol{x})] \tag{17}$$

$$\textbf{Generator}: Minimize\ -\mathbb{E}_{\boldsymbol{x}\sim P_g \|D\|_L\leq 1}[D(\boldsymbol{x})] \tag{18}$$

Note that above is the intuitive derivation. The math derivation of losses in WGAN is genius and complicated, refer to [20]. (17) and (18) are actually drawn from:

$$\mathcal{W}(P_r, P_g) = sup_{\|D\|_L=1}\mathbb{E}_{\boldsymbol{x}\sim P_r}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim P_g}[D(\boldsymbol{x})] \tag{19}$$

In (19), $\mathcal{W}(P_r, P_g)$ is the Wasserstein-1 Distance between distribution P_r and P_g [20]. To make formulas (17)(18)(19) hold, it is necessary to satisfy at least $\|D\|_L = 1$, which stands for 1-Lipschitz constraint for Discriminator.

Lipschitz constraint is an important concept for neural network, especially GANs. Networks under Lipschitz constraint condition generally have a stable training process and suffer less from vanishing

gradient problem. In WGANs, Lipschitz constraint is indispensable. For a given function $f(x)$, 1-Lipschitz constraint is:

$$\|f(\boldsymbol{a}) - f(\boldsymbol{b})\| \leq \|\boldsymbol{a} - \boldsymbol{b}\| \Leftrightarrow \frac{\|f(\boldsymbol{a}) - f(\boldsymbol{b})\|}{\|\boldsymbol{a} - \boldsymbol{b}\|} \leq 1 \Leftrightarrow \|\nabla f(\boldsymbol{x})\| \leq 1 \tag{20}$$

Recall that the goal of Discriminator is to measure the distance between real data distribution and fake distribution. Ideally, we want that a small change in the input will not cause the output of Discriminator to change dramatically, which is a basic requirement when calculating "distance". If not so, the Discriminator is considered to be "chaotic" and hence the training process becomes extremely unstable. Lipschitz constraint here is to help to stabilize the process of training. As a matter of fact, WGAN family outperforms DCGAN, since the latter does not include such a mechanism.

The original WGAN uses "weight clipping" to maintain 1- Lipschitz condition [20]. WGAN-GP replace this method by adding a gradient penalty term [9] to the loss function of Discriminator:

$$\boldsymbol{x_{inter}} = \varepsilon \cdot \boldsymbol{x} \sim P_r + (1 - \varepsilon) \cdot \boldsymbol{x} \sim P_g, \varepsilon \sim U(0,1) \tag{21}$$

$$\textbf{Discriminator}: Maximize\ \mathbb{E}_{\boldsymbol{x}\sim P_r}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim P_g}[D(\boldsymbol{x})] - \lambda \cdot \mathbb{E}[(\|\nabla D(\boldsymbol{x_{inter}})\| - 1)^2] \tag{22}$$

The loss functions in WGAN-GP are (18)(22). In (22), the gradient of Discriminator is approximated via linear interpolation. The last term forces $D(x)$ to obey 1-Lipschitz continuous condition by enforcing this term to zero during iteration. λ in (22) can be any positive numbers according to need. (22) is an empirical formula but works well. WGAN-GP is one of the most famous types of generative adversarial network.

(References for equations and theories in **Section 3.2**: [9][19][20][21][33])

The network structures of WGAN-GP and WGAN-DIV are completely identical in this project. For their structure, please **Figure 10**.

3.3 WGAN-DIV

WGAN-DIV is short for **Wasserstein Divergence for GAN**, proposed in [10]. WGAN-DIV expands the Wasserstein Distance in (19) to Wasserstein Divergence. The original paper claims that using Wasserstein Divergence will not only maintain the outstanding property of Wasserstein Distance in GAN, but also removes the 1-Lipschitz constraint requirement inside Discriminator [10]. What is needed to do is training Discriminator based on:

$$\textbf{Discriminator}: Maximize\ \mathbb{E}_{\boldsymbol{x}\sim P_r}[D(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim P_g}[D(\boldsymbol{x})] - k \cdot \mathbb{E}[(\|\nabla D(\boldsymbol{x_{inter}})\| - \boldsymbol{n})^p] \tag{23}$$

Train Generator according to:

$$\textbf{Generator}: Minimize - \mathbb{E}_{\boldsymbol{x}\sim P_g}[D(\boldsymbol{x})] \tag{24}$$

The optimizing result of (23), the $D(x)$, is the optimal solution for formula (19). The WGAN-DIV paper proves that (23) is a divergence, which is essential. A divergence can represents the mathematical distance between two distributions. The loss of WGAN-GP (22) is an empirical formula without strict mathematical proof. However, choosing (23) as the loss function, we are genuinely minimizing the distance between two distributions.

In engineering perspective, parameters in (23) are chosen to be $k = 2, p = 6, n = 0$ [10]. It is interesting that if we use $k = \lambda, p = 2, n = 1$, formula (23) is equivalent to formula (22). Under this condition, WGAN-DIV becomes WGAN-GP.

The derivation of WGAN-DIV is rather complicated, if interested, please refer to [10].

(References for equations and theories in **Section 3.3**: [10][22][33])

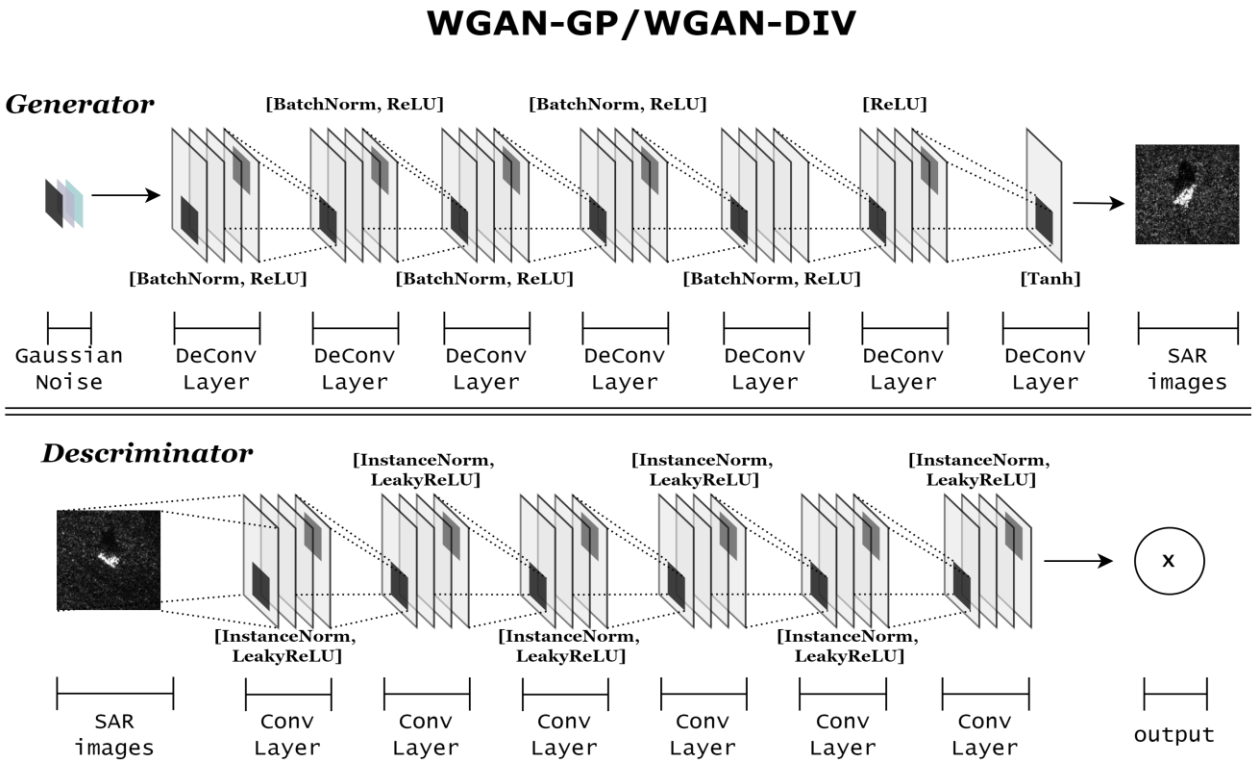


Figure 10 Visualisation of WGAN-GP/WGAN-DIV structure

Figure 10 demonstrates the architecture of WGAN-GP and WGAN-DIV. Compared to DCGAN, the structure of Generators is exactly the same. The only difference inside Discriminators is replacing *Batch Normalization* with *Instance Normalization*. For detailed structure, please refer to **Appx Table 2**.

3.4 SNGAN

As discussed previously, DCGAN lacks mechanism that guarantees 1-Lipschitz inside Discriminator, which leads to an unstable training process and a bad performance compared to WGAN-GP. However, the gradient penalty in WGAN-GP is not a sufficient condition for enforcing the 1-Lipschitz condition. The problem occurs because in (21)(22), we use a linear interpolation of real data and fake data to approximate the gradient of D, in which different samples of real data and fake data will lead to unequal values of $\nabla D(\boldsymbol{x_{inter}})$. Approximate solutions may not necessarily work globally. SNGAN, **Spectral Normalized GAN** [11], aims to force 1- Lipschitz in a way that differs from gradient penalty, in a stricter manner.

Now, consider a simple vector mapping relation: $y = \boldsymbol{\omega}x + \boldsymbol{b}$, where $\boldsymbol{\omega}$ is weight and $\boldsymbol{b}$ is bias. There is a function f. Under Lipschitz constraint condition, $\|f(\boldsymbol{\omega}x_1 + \boldsymbol{b}) - f(\boldsymbol{\omega}x_2 + \boldsymbol{b})\| \leq L \cdot \|x_1 - x_2\|$, if x_1 and x_2 are close enough, by first order Taylor Expansion, the approximation yields: $\left\|\frac{\partial f}{\partial y}\boldsymbol{\omega}(x_1 - x_2)\right\| \leq L \cdot \|x_1 - x_2\|$. If we want the left side is smaller than the right side, it requires that $\frac{\partial f}{\partial y}$ has upper and lower bounds. If satisfied, $\frac{\partial f}{\partial y}$ can be regarded as a constant. Hence, we got:

$$\|\boldsymbol{\omega}(x_1 - x_2)\| \leq L \cdot \|x_1 - x_2\| \tag{25}$$

Now, according to [11], $\boldsymbol{\omega}$ has its spectral norm form $\|\boldsymbol{\omega}\|_2$. The following inequality holds identically:

$$\|\boldsymbol{\omega}(x_1 - x_2)\| \leq \|\boldsymbol{\omega}\|_2 \cdot \|x_1 - x_2\| \tag{26}$$

Approximately, to make (25) hold identically, we can choose $L = \|\boldsymbol{\omega}\|_2$. To be specific, if we replace the weight of original mapping relation $\boldsymbol{\omega}$ to $\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|_2}$, 1- Lipschitz will be always guaranteed in f and y. The nature of neural network layers (e.g., fully connected layers, convolution layers) are just multiple combinations of variations of the simplest mapping function $y = \boldsymbol{\omega}x + \boldsymbol{b}$. Therefore, SNGAN just replaces every weight in discriminator from $\boldsymbol{\omega}$ to $\frac{\boldsymbol{\omega}}{\|\boldsymbol{\omega}\|_2}$ [11], meanwhile, we have to ensure that the first order derivative of every activation function inside Discriminator is bounded. Luckily, all the four activation functions in **2.1.3** own such feature.

Use the following iteration rules to calculate $\|\boldsymbol{\omega}\|_2$:

$$\boldsymbol{\nu} \leftarrow \frac{\boldsymbol{\omega}^T\boldsymbol{\mu}}{\|\boldsymbol{\omega}^T\boldsymbol{\mu}\|}, \boldsymbol{\mu} \leftarrow \frac{\boldsymbol{\omega}\boldsymbol{\nu}}{\|\boldsymbol{\omega}\boldsymbol{\nu}\|}$$

$$\|\boldsymbol{\omega}\|_2 \approx \boldsymbol{\mu}^T\boldsymbol{\omega}\boldsymbol{\nu} \tag{27}$$

Theoretically, if normalizing Discriminator as above, $\|D\|_L \leq 1$ holds identically.

The choose of the loss function of SNGAN can be identical to that of DCGAN, see (12)(13). The author uses an alternative that has been proved to work properly according to SNGAN paper. They are known as Hinge Loss:

$$\textbf{Discriminator}: Minimize\ \mathbb{E}_{\boldsymbol{x}\sim P_r}\left[\max\left(0, 1 - D(\boldsymbol{x})\right)\right] + \mathbb{E}_{\boldsymbol{x}\sim P_g}\left[\max\left(0, 1 + D(\boldsymbol{x})\right)\right] \quad (\mathbf{28})$$

$$\textbf{Generator}: Minimize\ -\mathbb{E}_{\boldsymbol{x}\sim P_g}[D(\boldsymbol{x})] \quad (\mathbf{29})$$

Hinge Loss can be understood as a restricted version of Wasserstein Loss. (29) is identical to (18), (28) is a variation of (17). Hinge Loss in Discriminator prevents itself from over-optimising, which might cause vanishing gradient problem, by the max $(0, x)$ term. This term introduces a threshold to the output of Discriminator. If output beyond that threshold, the optimization is truncated.

(References for equations and theories in **Section 3.4**: [11][21][24])

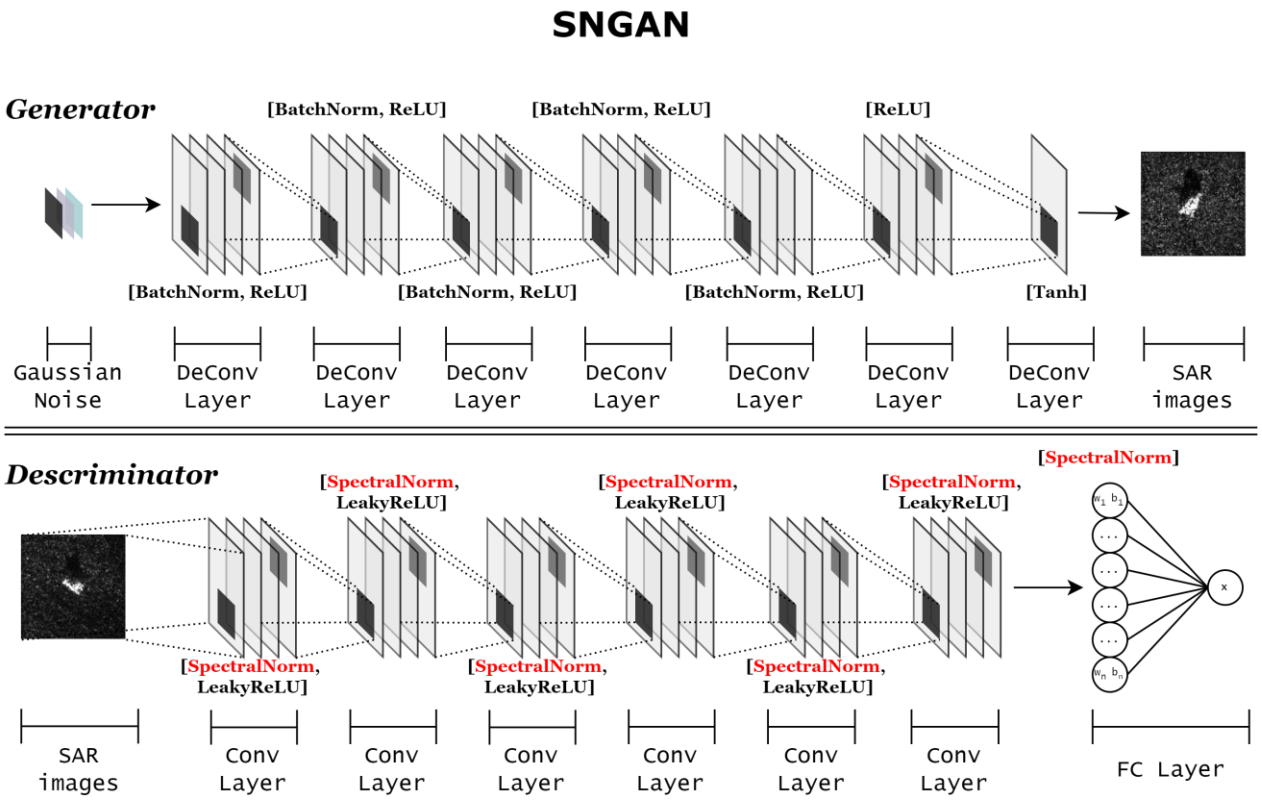
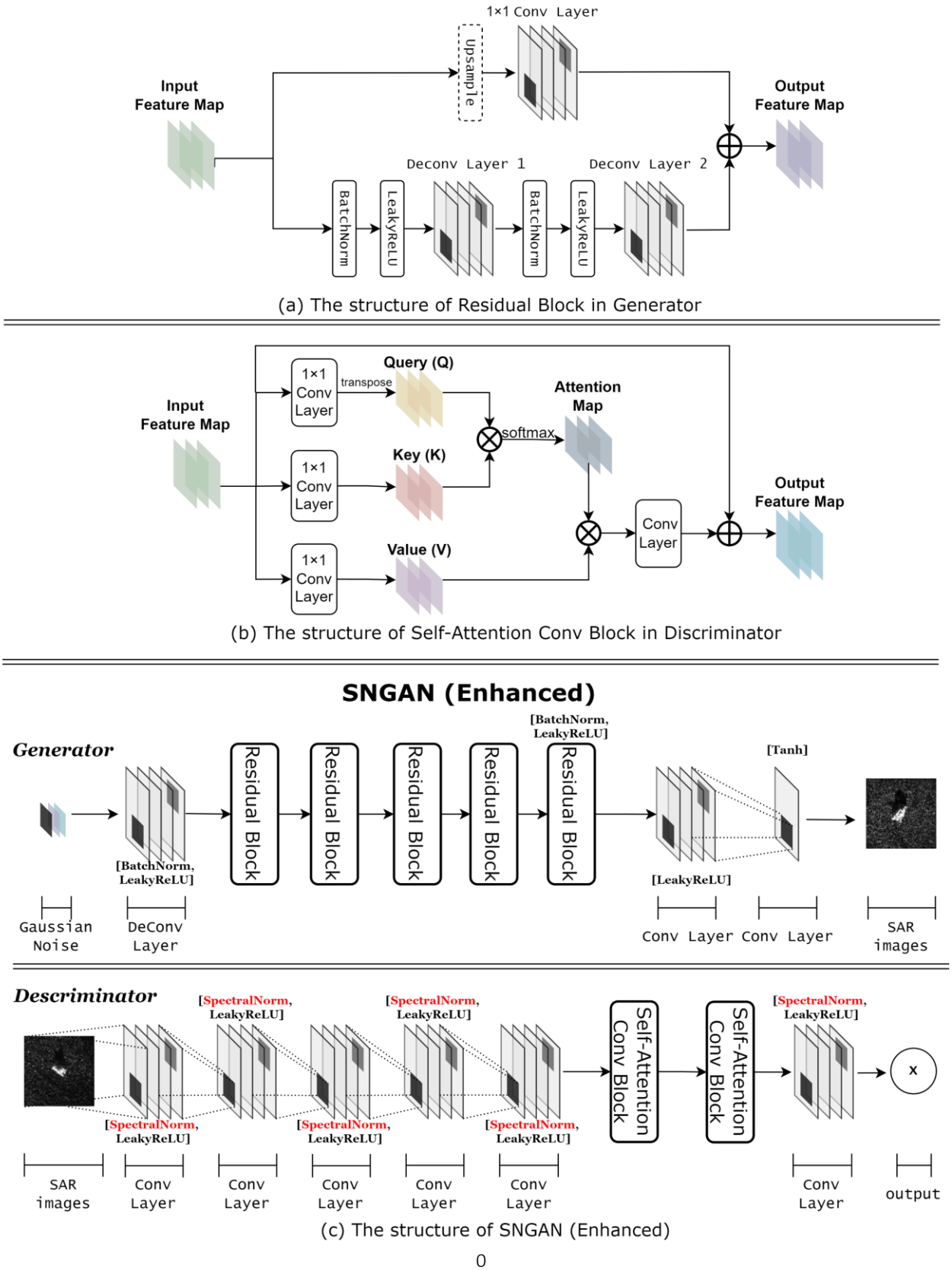


Figure 11 Visualisation of SNGAN structure

Figure 11 shows the structure of SNGAN in this project. The Generator is identical to DCGAN and WGAN series. In Discriminator, *Spectral Normalization* is applied to constraint the weight to satisfy Lipschitz condition. For detailed structure, please refer to **Appx Table 3**.

3.5 SNGAN (Enhanced)

Compared to WGAN-GP/DIV, the speed of training SNGAN is slightly swifter. Therefore, based on Spectral Normalization, the author constructs another GAN model, whose architecture (see **Figure 12**) is more complex, as compared to the previous SNGAN, to improve its performance.



0

Figure 12 Visualisation of SNGAN (Enhanced) structure

In Generator, the author replaces some of the Deconv layers by Residual Block (**Figure 12** (a)) to achieve up sampling. In a residual block, the output of a layer (the lower path in **Figure 12** (a)) is added to its input (the upper path, the output of an input passing through 1×1 Conv is equal to itself,

but with different data dimensions), which creates a residual connection that retains information from earlier layers. This helps prevent the problem of vanishing gradients and makes it easier for the network to learn more complex patterns in the data [25]. Residual Block has been proved to improve the generation fidelity of GAN and has been applied in the original paper of WGAN-GP [9] and SNGAN [11].

In Discriminator, before the last layer, the author adds two Self-Attention Conv Block, which is proposed in [26]. Self-Attention mechanism is a critical component in Transformer architecture. In short, Self-Attention Conv Block can be considered as a multiplication of a feature map (the input) and its own transpose, so that the pixels at any two locations are directly related [26]. The end result is that the dependencies between any two pixels can be learned, and the global features can be obtained [26]. It is worthing mentioning that there is also a residual connection (the upper path in **Figure 12** (b)) inside Self-Attention Conv Block to improve the stability of training. For detailed structure, please refer to **Appx Table 4**.

3.6 VAE-GAN

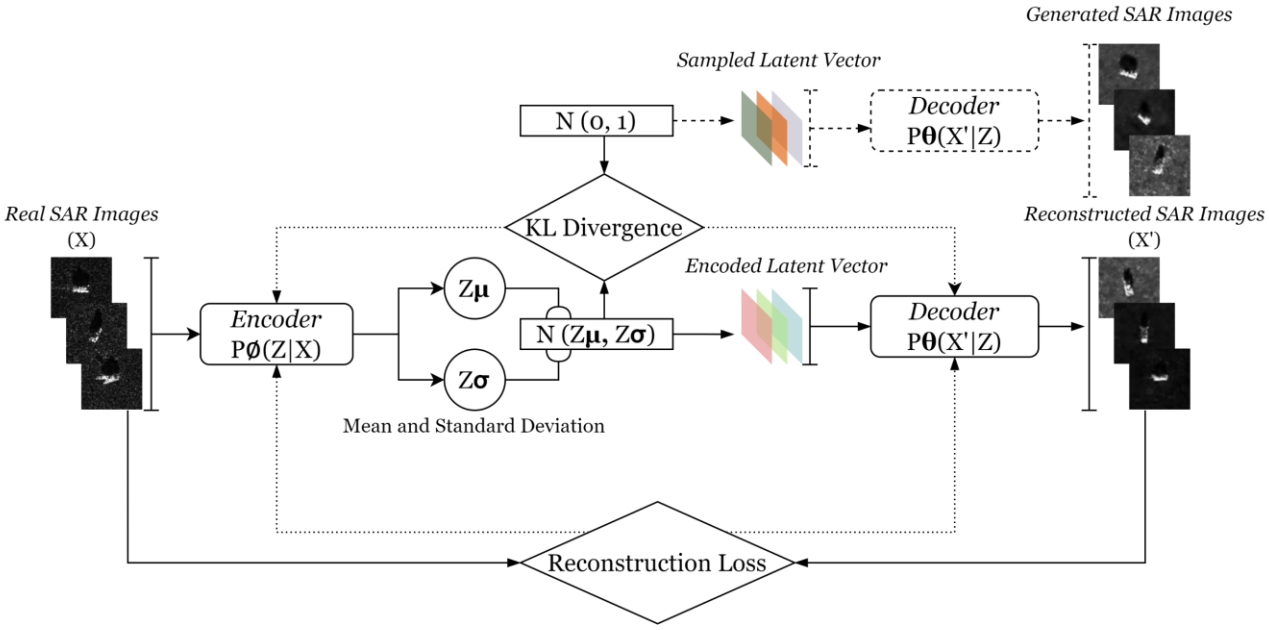


Figure 13 The workflow of VAE in image synthesis

GAN is one of the multiple types of deep generative models that can conduct image synthesis. One of GAN's alternative is Variational Auto-Encoder, or VAE, which is proposed in [23]. VAE owns a completely different working principle from GAN. A VAE has two components: an Encoder and a Decoder. For image generation, Encoder encodes an image into a latent vector (usually much smaller than the size of original image), which is called "bottleneck". Bottleneck is input to the Decoder, which can reconstruct the encoded image based on the features of the bottleneck. When training VAE, the distribution of the bottleneck is forced to approach a certain distribution, usually standard

Gaussian distribution $\mathcal{N}(0,1)$. Hence, if VAE is trained properly, we can simply sample a latent vector from that certain distribution and input it to Decoder, which outputs image that is fidelity to the original data. In that case, VAE can conduct image generation. From a high-level perspective, the nature of Encoder in VAE is to mapping a data distribution P_r to a target distribution $\mathcal{N}(0,1)$, while the nature of Decoder is to mapping $\mathcal{N}(0,1)$ to P_g, where P_g is as close to P_r as possible.

Figure 13 demonstrates the training of VAE. $P_\emptyset(\boldsymbol{z}|\boldsymbol{x})$ is the distribution of the Encoder's outputs according to the input (training) images. During training, $P_\emptyset(\boldsymbol{z}|\boldsymbol{x})$ should be as close to $\mathcal{N}(0,1)$ as possible. It is achieved by minimizing the Kullback-Leibler Divergence [23], or KLD between the above distributions. KLD can be applied to calculate the distance between two distributions. One the other hand, the output data of Decoder should be close to the training data, which can be satisfied by minimizing the mean square error (MSE) between the original data and the generated ones [23]. This is known as a reconstruction loss. Hence, the overall loss function of VAE has two terms:

$$Minimize\ \mathbb{E}_{\boldsymbol{x}\sim P_r}[\mathrm{KLD}(P_\emptyset(\boldsymbol{z}|\boldsymbol{x})\parallel\mathcal{N}(0,1))]+\mathbb{E}_{\boldsymbol{x}'\sim P_\theta(\boldsymbol{x}'|\boldsymbol{z}),\boldsymbol{x}\sim P_r}[\mathrm{MSE}(\boldsymbol{x}',\boldsymbol{x})] \tag{30}$$

$P_\theta(\boldsymbol{x}'|\boldsymbol{z})$ is the distribution of the outputs of Decoder, that is, the distribution of reconstructed images, according to bottleneck. Unlike GAN, VAE does not suffer from mode collapse and its training process is stable [32]. There exist blurry issues in the generated images of VAE, which can be overcome by introducing adversarial training.

Proposed in [12], VAE-GAN is the combination of VAE and GAN, which adds a Discriminator to VAE that help to improve the resilience of Decoder's generated samples, while maintains the excellent properties of VAE. In VAE-GAN, the Decoder is not only a "Decoder" in VAE, but also a "Generator" in GAN. Here are the loss functions of VAE-GAN:

Encoder: $Minimize$

$$\alpha\cdot\mathbb{E}_{\boldsymbol{x_r}\sim P_r,\boldsymbol{x_g}\sim P_g}\mathrm{MSE}(D_{\mathbf{INT}}(\boldsymbol{x_r}),D_{\mathbf{INT}}(\boldsymbol{x_g}))+\beta\cdot\mathbb{E}_{\boldsymbol{x_r}\sim P_r}[\mathrm{KLD}(P_\emptyset(\boldsymbol{z}|\boldsymbol{x_r})\parallel\mathcal{N}(0,1))] \tag{31}$$

Decoder (**Generator**): $Minimize$

$$\lambda\cdot\mathbb{E}_{\boldsymbol{x_r}\sim P_r,\boldsymbol{x_g}\sim P_g}\mathrm{MSE}(D_{\mathbf{INT}}(\boldsymbol{x_r}),D_{\mathbf{INT}}(\boldsymbol{x_g}))-\mathbb{E}_{\boldsymbol{x_g}\sim P_g}\left[log\ D\left(\boldsymbol{x_g}\right)\right] \tag{32}$$

Discriminator: $Minimize$

$$-\mathbb{E}_{\boldsymbol{x}\sim P_r}[log\ D(\boldsymbol{x})]-\mathbb{E}_{\boldsymbol{x}\sim P_g}\left[log\left(1-D(\boldsymbol{x})\right)\right] \tag{33}$$

In (31)(32)(33), to ensure the simplicity of formulas, $\boldsymbol{x_g}\sim P_g$ is used to replace:

$$\boldsymbol{x}'\sim P_\theta(\boldsymbol{x}'|\boldsymbol{z})\cdot P_{\emptyset,\boldsymbol{x}\sim P_r}(\boldsymbol{z}|\boldsymbol{x}) \tag{34}$$

(31)(32)(33), the loss functions of VAE-GAN are combinations of (12)(13)(30). The only difference is that the reconstruction loss switches from $\mathbb{E}_{\boldsymbol{x_r}\sim P_r,\boldsymbol{x_g}\sim P_g}\mathrm{MSE}(\boldsymbol{x_r},\boldsymbol{x_g})$ to $\mathbb{E}_{\boldsymbol{x_r}\sim P_r,\boldsymbol{x_g}\sim P_g}\mathrm{MSE}(D_{\mathbf{INT}}(\boldsymbol{x_r}),D_{\mathbf{INT}}(\boldsymbol{x_g}))$. $D_{\mathbf{INT}}$ refers to the output values of the intermediate layers of

Discriminator. The modified reconstruction loss both reduces the computation complexity and maintains the consistency in training the VAE-GAN model.

Compared to pure GAN model, VAE-GAN can generate images from two methods: direct sampling and reconstruction. The former is identical to previous GANs, that is, inputting a noise sampled from a prior distribution to the Decoder, which outputs images. The latter is simply inputting an image to the Encoder-Decoder, and it will output a batch of reconstructed images. Due to the reparameterization trick involved in sampling the bottleneck, the generated images exhibit similarity to the original images but are not identical, and there exists internal variation within the produced images. For convenience, in the rest of the report, the author uses "VAE-GAN-E" to represent the reconstruction mode of VAE-GAN, and "VAE-GAN-S" to represent the directly sampling mode.

(References for equations and theories in **Section 3.6**: [12][23])

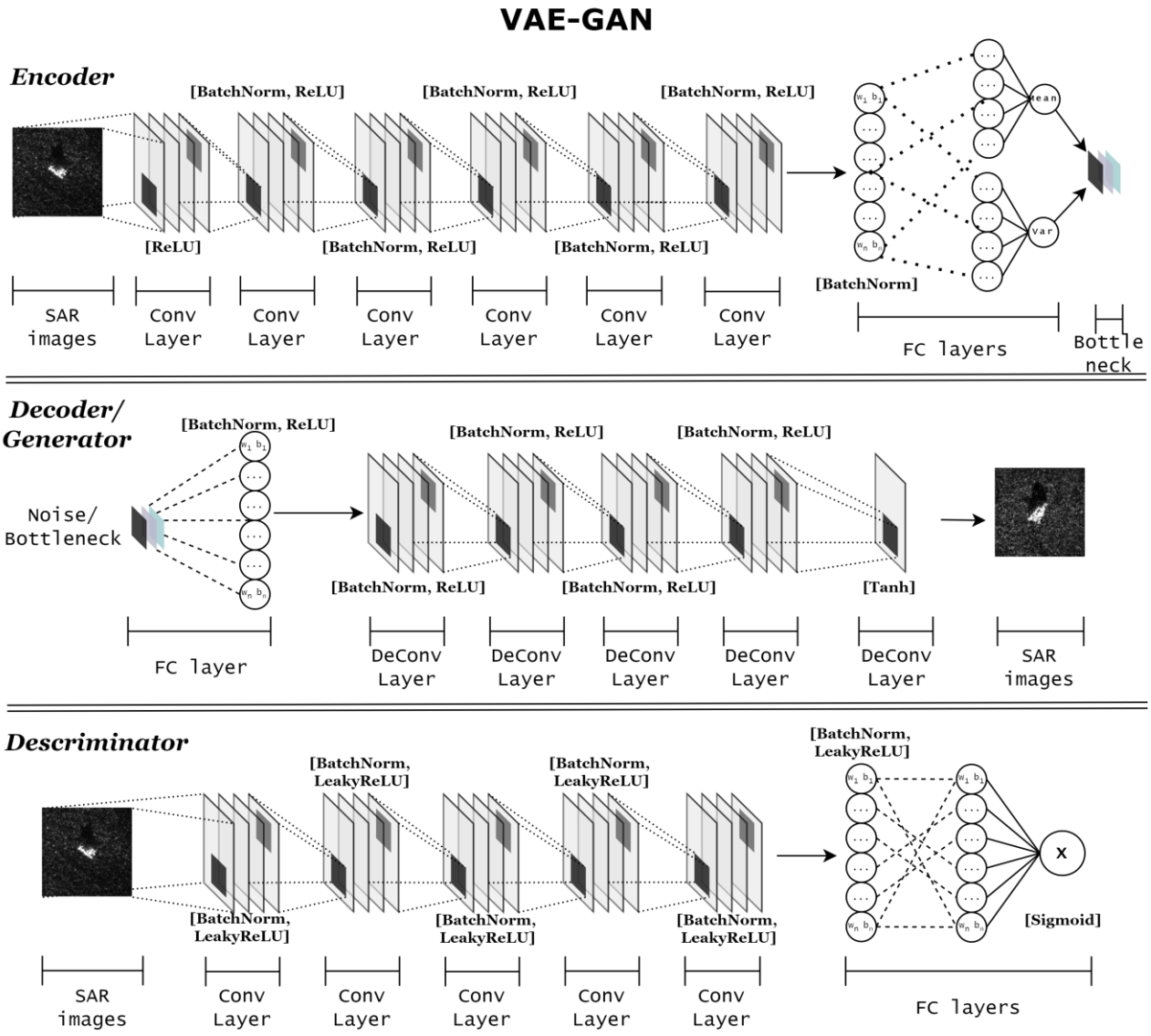


Figure 14 Visualisation of VAE-GAN structure

Figure 14 exhibits the structure of VAE-GAN. Training VAE-GAN is training three neural networks simultaneously. For detailed structure, see **Appx Table 5**.

3.7 Algorithm for Training GANs

The six GANs introduced above are roughly trained according to following algorithm:

Algorithm Training Generative Adversarial Network (GAN), references: [7][9][11]

Requirements Generator G; Discriminator D; Generator parameters θ_G; Discriminator parameters θ_D; Generator loss function $Loss_G$; Discriminator loss function $Loss_D$; optimizer Optim; batch size m; training epochs n; steps k.

1: Initialize θ_G, θ_D

2: **for** number of training epochs n **do**

3: **for** steps k **do**

4: Sample m noises from prior noise distribution, usually $\mathcal{N}(0,1)$:

$$z_1, z_2, \dots, z_m \Leftrightarrow \boldsymbol{z} \sim \mathcal{N}(0,1)$$

5: Calculate m generated samples by inputting the sampled noises to Generator:

$$G(\boldsymbol{z}) = x_1, x_2, \dots, x_m \Leftrightarrow \boldsymbol{x_g} \sim P_g$$

6: Sample m real samples from training data: $x_1, x_2, \dots, x_m \Leftrightarrow \boldsymbol{x_r} \sim P_r$

7: Update the parameters of Discriminator based on its loss function:

$$\theta_D \leftarrow \text{Optim}[\nabla_{\theta_D} Loss_D\big(D(\boldsymbol{x_g}), D(\boldsymbol{x_r})\big)]$$

8: **end for**

9: Sample another m noises from prior noise distribution $\mathcal{N}(0,1)$:

$$z_1, z_2, \dots, z_m \Leftrightarrow \boldsymbol{z'} \sim \mathcal{N}(0,1)$$

10: Calculate another m generated samples by inputting the sampled noises to Generator:

$$G(\boldsymbol{z'}) = x_1, x_2, \dots, x_m \Leftrightarrow \boldsymbol{x_g}' \sim P_g$$

11: Update the parameters of Generator based on its loss function:

$$\theta_G \leftarrow \text{Optim}[\nabla_{\theta_G} Loss_G\big(D(\boldsymbol{x_g}')\big)]$$

12: **end for**

Notes P_r refers to the distribution of real data; P_g refers to the distribution of fake data (data generated by Generator); $\boldsymbol{x_r} \sim P_r$ means $\boldsymbol{x_r}$ is sampled from distribution P_r; "$\Leftrightarrow$" is "equivalent to"; Steps $k = 1$ means training Discriminator and Generator equally in an epoch, $k \geq 1$ means training Discriminator more than Generator in an epoch.

4 Model Performance Evaluation and Discussion

In this section, the author evaluates the performance and availability in SAR image synthesis of each GAN model based on different schemes.

4.1 **Evaluation via Classification Performance**

Readers may wonder about how to validate the effectiveness of the generated SAR images. One simple but reliable method is to test that to what extent the generated SAR images can affect the recognition performance. In previous sections, it is mentioned that SAR image can be combined with machine learning to achieve target recognition task. The nature of target recognition is classification: Take MSTAR dataset as an example, a recognition model is trained on the train dataset, which includes ten categories of SAR images. The recognition performance of this model is the probability of classify the SAR images from test dataset into the correct category, namely, the classification accuracy.

Machine learning is a discipline based on big data [27]. Limited number of training data often leads to poor performance in recognition experiment. The cost of acquiring SAR images is relatively high, which sometimes causes the scarcity of data in dataset consists of SAR images. From this perspective, the author tests the effectiveness of generated SAR images when 1. using a small number of SAR images to train GANs (**section 4.1.1**) 2. using even fewer SAR images to train GANs (**section 4.1.2**), by recognition performance. It is worth noting that the ability of GANs is largely depended on how many data is fed in training process. Generally, "the more the better". A GAN may have a disappointing performance, or it may not converge at all when there are not enough training data. Compared to some of the famous dataset for evaluating GANs, such as LSUN and Cifar10, MSTAR has less than one-tenth the amount of data. Hence, to evaluate the full capability of the six implemented GANs in generating SAR images, the author trains them on the entire train dataset of MSTAR and conduct evaluation via classification performance (**section 4.1.3**).

The model for carrying classification experiment is **Support Vector Machine** (SVM), which is perhaps the most influential algorithm for classification task in classic machine learning field [27]. SVM aims to find the optimal hyperplane that maximizes the margin between any two classes of data from the train dataset, and it is proved to be of high efficiency in classifying high dimensional data [27]. To reduce the computational complexity, images used to train SVM is pre-processed by Principal Component Analysis (PCA). PCA can reduce the dimensionality in a group of data [27]. The idea behind PCA is to simplify complex datasets by identifying patterns and relationships between the different samples. The size of each original SAR image is 128×128, which is equivalent

to 16384 int8 numbers. After PCA, this number can be reduced to less than 100. For detailed working principle of PCA and SVM, please refer to [27].

Even though Convolutional Neural Network (CNN) can achieve a better recognition performance in SAR images than the PCA-SVM pipeline mentioned above, it requires efforts to build a well-performed CNN. Since there are six GAN models to be evaluated, if using CNN as recognition model, there shall be a considerable amount of time spending on training this CNN on different sets of SAR images. In terms of time cost, the author chooses PCA-SVM pipeline. Evaluation through PCA-SVM is swift. Support Vector Machine can be trained within a few seconds, by which the time saved can be spent on fine-tuning the GANs to achieve better generation ability.

4.1.1 Under Insufficient Training Samples Condition

In this section, all six GANs are trained on insufficient number of SAR samples. In train dataset of MSTAR (**Table 1**), there are 10 classes of SAR images, each with varying numbers of images ranging from 232 to 299. The author picks 50 images from each of ten classes, total 500 images, as the new train dataset to train GANs. For each class, the author guarantees that the 50 images are evenly distributed in azimuth angles as much as possible, which is to ensure the diversity of training images. This operation can effectively improve the classification accuracy.

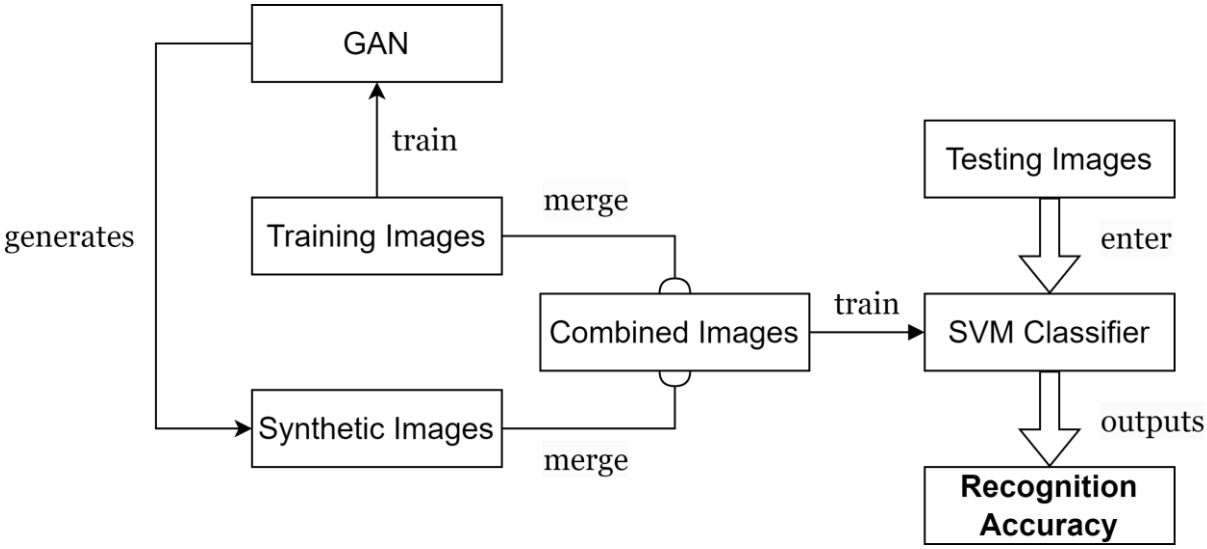


Figure 15 Evaluation pipeline 1

Figure 15 shows the evaluation pipeline: First train GANs on the selected 500 SAR images. Generate synthetic SAR images from the trained GANs. Combine the synthetic ones with the 500 real SAR images. Use the combined images to train the SVM classifier. Use the trained classifier to predict the class of SAR images in MSTAR test dataset (2426 images of 10 classes) and calculate the **Recognition Accuracy**.

The author intends to observe how will the synthetic images affect the recognition accuracy by setting the size of synthetic images to 1, 2, 3, 4, 5 times the size of training images, which is equivalent to

100, 150, 200, 250, 300 images per class for training SVM. As a comparison, the author tests the prediction accuracy of training SVM on only the training images, which is 81.03%, the baseline accuracy. Here are the experiment results:

Table 2 Classification results (Insufficient Training Samples)

	0	50	100	150	200	250
DCGAN	81.03%	-	-	-	-	-
WGAN-GP	81.03%	83.75%	84.28%	84.48%	84.70%	84.91%
WGAN-DIV	81.03%	**83.96%**	**84.49%**	84.88%	84.94%	85.20%
SNGAN	81.03%	83.02%	83.15%	84.20%	84.29%	84.61%
SNGAN (Enhanced)	81.03%	83.36%	84.46%	**85.30%**	**85.98%**	**86.27%**
VAE-GAN-S	81.03%	81.56%	82.35%	82.26%	82.27%	82.39%
VAE-GAN-E	81.03%	83.08%	83.19%	82.74%	83.03%	82.63%

The horizontal axis of **Table 2** is the number of synthetic images per class merged with training images. The vertical axis is the type of GAN. The values inside the table are how much percentage of SAR images in test dataset of MSTAR is correctly recognized. For example, "WGAN-GP/200, 84.70%" indicates that the combined images (**Figure 16**) contain 200 synthetic images and 50 training images per class, using the combined images to train SVM can obtain 84.70% **Recognition Accuracy** in classifying test images.

Inferred from **Table 2**, except for DCGAN, the other 5 types of GAN models can always improve the recognition performance when the generated images are added to the original training images. DCGAN cannot be properly trained when there are only 50 samples in each class, because severe mode collapse occurs, which results in the model can only generate 1 to 4 clusters of similar SAR images in every category. Hence, the classification accuracy values for DCGAN are "-".

It can be concluded that using WGAN-GP, WGAN-DIV, SNGAN, SNGAN(Enhanced) and VAE-GAN to conduct data augmentation for target recognition is valid under insufficient training samples condition, since classification accuracy is successfully improved. For WGAN-GP, WGAN-DIV, SNGAN and SNGAN(Enhanced), when the number of synthetic images increases, classification accuracy shows an increasing trend. WGAN-DIV outperforms others when synthetic images number is 50 and 100 per class. Under 250 generated samples, SNGAN(Enhanced) can achieve 86.27% accuracy, which improves the baseline accuracy 81.03% by 5%. VAE-GAN is less effective because its generated samples can only improve the accuracy by 2%, and accuracy does not constantly increase as the number of synthetic images increases.

As a comparison, using all SAR images (total 2746) in train dataset to train SVM can achieve an accuracy level of 94.27% in predicting test images.

4.1.2 Under Extremely Insufficient Training Samples Condition

The evaluation pipeline in this section is identical to **Figure 15**. The only difference is that instead of selecting 50 images per class, a smaller dataset is used. The new dataset only consists of 10 SAR images per class, total 100 images. Expanding the training images 1, 2, 3, 4, 5 times by adding synthetic images from GANs, the author records the classification accuracy of the SVM classifier on test images, in **Table 3**. As a comparison, the prediction accuracy of training SVM on only the 100 training images is 44.39%, which is the baseline accuracy.

Table 3 Classification results (Extremely Insufficient Training Samples)

	0	10	20	30	40	50
DCGAN	44.39%	-	-	-	-	-
WGAN-GP	44.39%	47.27%	50.66%	51.73%	53.81%	54.41%
WGAN-DIV	44.39%	**47.94%**	**51.35%**	**52.34%**	**54.44%**	**55.32%**
SNGAN	44.39%	-	-	-	-	-
SNGAN (Enhanced)	44.39%	46.77%	47.48%	46.22%	45.97%	46.10%
VAE-GAN-S	44.39%	46.87%	49.08%	51.80%	53.84%	55.27%
VAE-GAN-E	44.39%	46.75%	48.92%	51.09%	51.71%	51.66%

In General, the generated SAR images from WGAN-GP, WGAN-DIV, SNGAN(Enhanced), VAE-GAN are valid for data augmentation in extremely insufficient sample condition, as the classification accuracy improves when synthetic images are added to the original training images.

Due to the extreme scarcity, besides DCGAN, SNGAN cannot be properly trained due to the severe mode collapse, when there are only 10 training images per class. Even SNGAN(Enhanced), the optimal model in previous section, performs poorly in the classification experiment. It only has 3% advantage over baseline accuracy at most.

Two GANs based on Wasserstein distance, WGAN-GP and WGAN-DIV, perform well in this experiment. WGAN-DIV is undoubtably the best model under extremely insufficient training samples condition. It outperforms other models under every condition and can achieve an accuracy of 55.32%, which is 11% greater than the baseline accuracy, when there are 50 synthetic images in the combined images. VAE-GAN can achieve performance no less than WGAN-DIV. The synthetic

images via direct sampling from VAE-GAN can improved the accuracy up to 55.27%. This result demonstrates the superiority of Variational Auto-Encoder in small-sample training conditions.

4.1.3 Under Sufficient Training Samples Condition

To assess the full ability of the six GAN models in synthesizing SAR images, the author trained them on the entire trainset (total 2746 images) of MSTAR dataset. The evaluation pipelines in this section are different from that in **Figure 15**, which is because if the combined images contain all the training images in MSTAR, even a low-performing GAN model can achieve high classification accuracy. The author conducts two evaluation tests to assess the diversity and fidelity of generated SAR images, respectively.

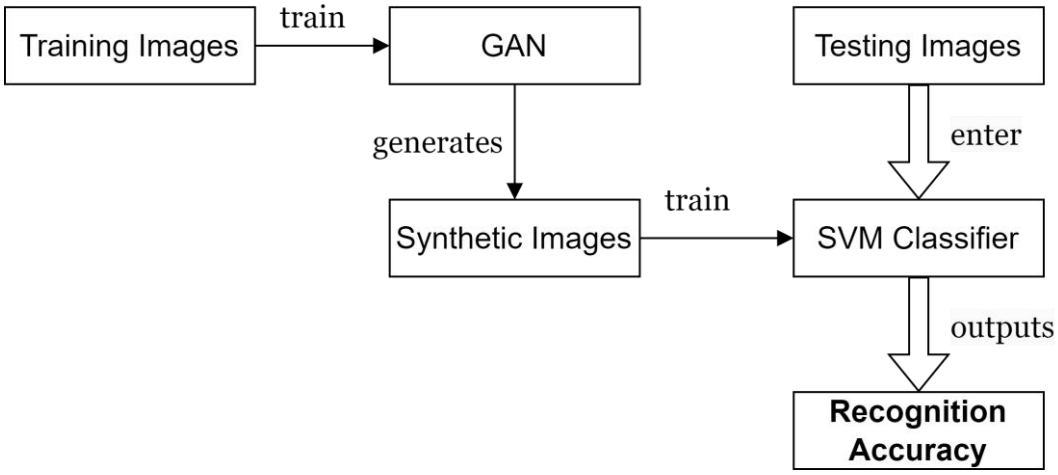


Figure 17 Evaluation pipeline 2

Table 4 Classification results (diversity)

Model	Recognition Accuracy
DCGAN	35.20%
WGAN-GP	82.27%
WGAN-DIV	77.99%
SNGAN	83.05%
SNGAN (Enhanced)	**86.64%**
VAE-GAN-S	76.50%
VAE-GAN-E	77.00%

High diversity indicates that a GAN model can generate images that reproduce as many trainset images as possible. Low diversity in GAN can be resulted from mode collapse and poorly generated samples. **Figure 16** demonstrates the pipeline to evaluate the diversity of generated SAR images. GANs are trained on the entire trainset (2746 images) of MSTAR. The SVM classifier is trained only

on synthetic images (1000 images per class, total 10000 images) generated by GANs, and performs classification task to acquire the recognition accuracy on test dataset (2426 images). The baseline accuracy is 94.27%, which is acquired by training SVM on the whole trainset of MSTAR. If the generation diversity of a GAN is very high, the recognition rate should be close to baseline accuracy.

The results in **Table 4** demonstrate that SNGAN (Enhanced) maintains the highest generation diversity if sufficient training images are provided. However, the highest recognition accuracy, 86.64%, is about 8% lower than the baseline accuracy, indicating that all six GAN models cannot reproduce every SAR image in the trainset.

The generation diversity of WGAN-GP and SNGAN is satisfactory since the corresponding accuracies are over 80%. DCGAN is the worst one, whose recognition accuracy is only 35.20%, which is caused by not only the low diversity, but also the low quality of generated SAR images.

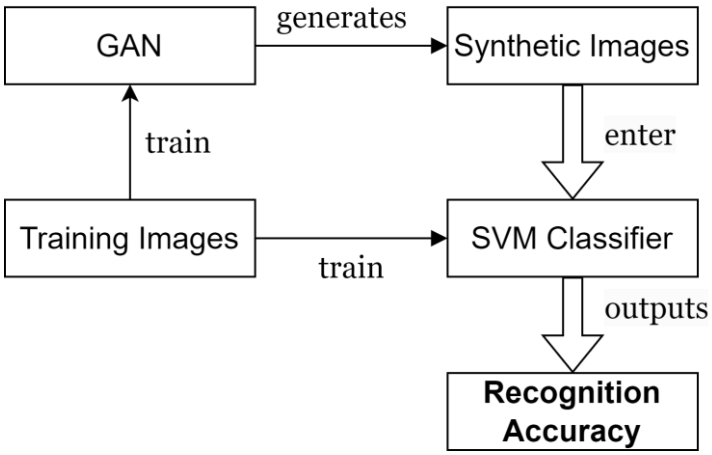


Figure 18 Evaluation pipeline 3

Table 5 Classification results (fidelity)

Model	Recognition Accuracy
DCGAN	74.99%
WGAN-GP	93.34%
WGAN-DIV	**96.40%**
SNGAN	94.33%
SNGAN (Enhanced)	**96.34%**
VAE-GAN-S	89.27%
VAE-GAN-E	94.61%

Fidelity refers to the degree of similarity between the generated images and the real images in trainset, which can be regarded as the quality of generated samples as compared to real samples. High fidelity means generated images owns a high resilience to training images. **Figure 16** demonstrates the pipeline to evaluate the fidelity. Training images contains all images in trainset of MSTAR (2746 images) and are utilized to train both GANs and SVM classifier. The synthetic images (1000 images per class, total 10000 images) from GANs enters SVM to acquire the recognition accuracy. The level of recognition accuracy can reflect the generation fidelity of GAN models. This is because only generated images that possess the features of training images can be correctly classified by the SVM trained on the same set of training images.

The results in **Table 5** show that WGAN-DIV and SNGAN (Enhanced) possess a good generation fidelity, as they own the highest recognition rate. The performances of WGAN-GP and SNGAN are satisfactory. For VAE-GAN, generation via direct sampling has lower fidelity than generation via encoding and reconstruction. This is because the latter is actually adding noise to the training images, which helps to maintain the features, while the former is mapping a noise distribution to training data distribution, resulting in some of the generated images be in low quality. Similar to the previous evaluation on diversity, DCGAN owns the worst performance in this experiment. 74.99% recognition accuracy means that one quarter of the generated samples are in low quality or chaotic.

In summary, considering the results obtained from both diversity and fidelity evaluation, SNGAN(Enhanced) is the most suitable model among the six types of GANs for generating SAR images under sufficient training samples condition.

4.2 Evaluation via Visual Effects and Statistics

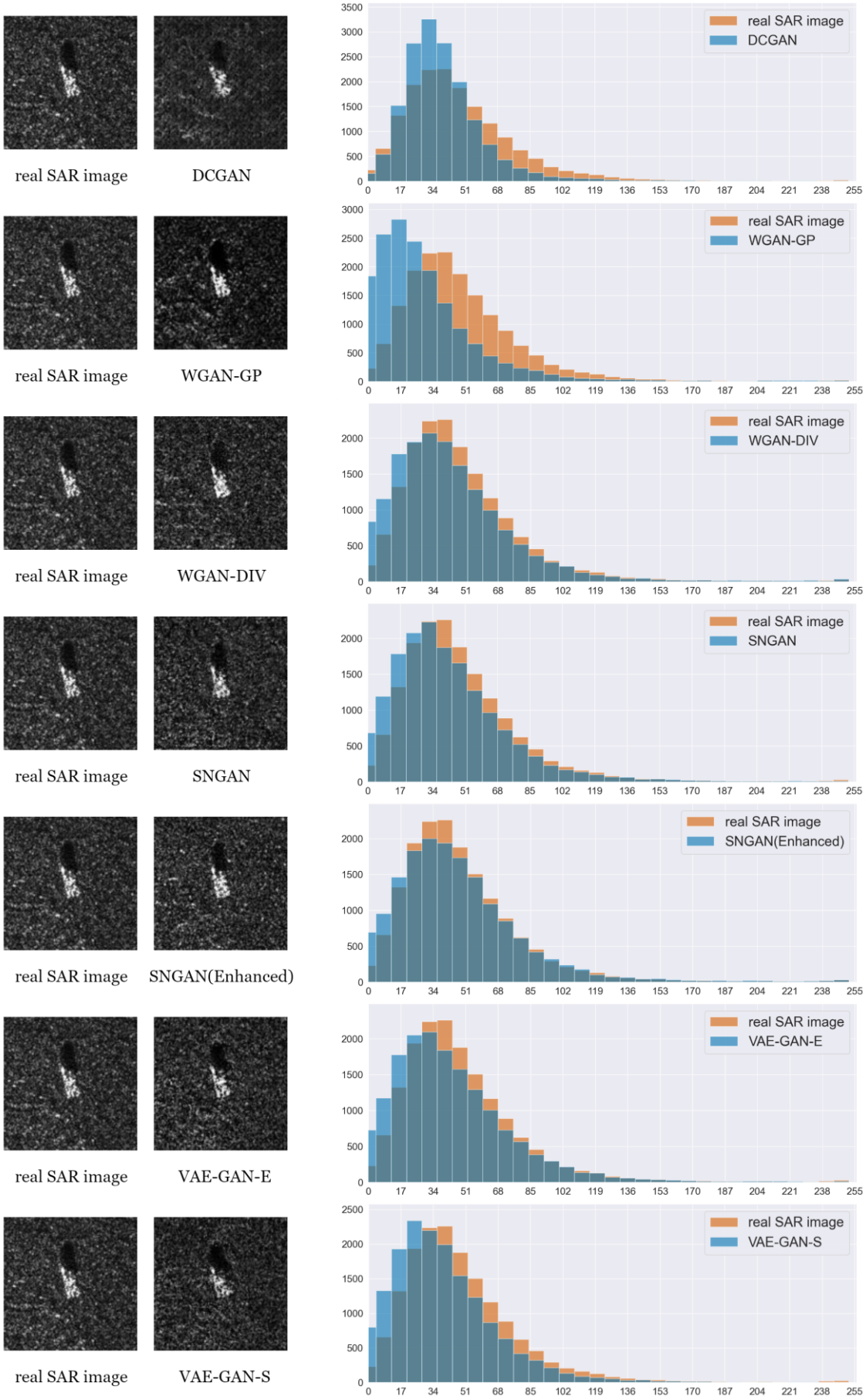


Figure 19 Comparison between real SAR image and SAR image from GANs

In this section, the generated SAR images of different six types of GANs are evaluated from visual perspective and statistics of grey scale values to assess the resilience (fidelity) to real SAR images. In **Figure 18**, the left column contains a real SAR image ('HB19782', class: 2S1) from trainset of MSTAR. The middle column contains the most similar synthetic SAR images from every GAN model to HB19872. All images are grayscale images that possess one image channel with 128×128 pixels, total 16384 pixels. The value of each pixel varies from 0-255 integer. The right column contains histograms used for statistically analyse the distribution of grayscale values in both real image and synthetic image. The horizontal axis of the histogram is "grayscale value". The vertical axis is "frequency", indicating how many times a certain range of grayscale values are presented inside an image.

From visual perspective, image from DCGAN is the worst since there are numerous artifacts presenting in the image. Image from VAE-GAN-S cannot accurately reconstruct the external features of 2S1. The appearance of the vehicle in WGAN-GP is roughly accurate. However, the image of WGAN-GP possesses a high contrast ratio compared to HB19782. Generally, the visual effects of WGAN-DIV, SNGAN, SNGAN(Enhanced) and VAE-GAN-E are roughly satisfactory, as the author cannot judge whether the generated images are real or synthetic if not zoom in.

Roughly speaking, the closer two gray value distributions are, the more similar two images are. The grayscale distribution of SNGAN(Enhanced) is the closest to the real SAR image. The distributions of images from WGAN-DIV, SNGAN and VAE-GAN-E varies larger, but they still maintain a good visual effect. The distribution of image by WGAN-GP has a noticeable centre of mass shift compared to HB19782, resulting in "high contrast ratio", which might be resulted from WGAN-GP is not well-trained.

To summarize, image from SNGAN(Enhanced) owns the highest quality and fidelity, compared to the real SAR image, while image from WGAN-DIV, SNGAN and VAE-GAN is capable of largely restoring the original image.

Due to the limitations in the length of the report, the author only exhibits the comparison between one set of images. Readers can refer to **Appx Figure 2-7** in **section A.1** for samples generated by different GANs in this project.

4.3 Evaluation via Training Overhead

This section is served as an additional test. Although training neural networks does not appear to be a sustainable activity due to the significant consumption of computational resources and electricity, the author tests the resource consumption of training six GAN models proposed in this report, from perspective of sustainable development. In this evaluation test, all the settings of six GAN models are the same:

1. Training dataset consists of 1024 SAR images with the size of 128×128
2. Batch size is 32, the dimensions for the input noise of Generator is 256
3. Total training epoch is 50, k=1 (train Discriminator and Generator equally in every epoch)
4. During the model training process, the computational precision is set to single-precision floating-point (FP32). The GPU used for training has 51.07 teraflops FP32 performance.

The results are:

Table 6 Training overhead of GANs

	Average Training Time per Epoch	Average GPU Memory Usage in Training Process	Average GPU Power in Training Process
DCGAN	0.704 sec	2.81 GB	293.85 W
WGAN-GP	1.215 sec	2.01 GB	272.49 W
WGAN-DIV	1.216 sec	2.02 GB	284.48 W
SNGAN	0.715 sec	2.57 GB	290.41 W
SNGAN (Enhanced)	1.519 sec	8.18 GB	285.92 W
VAE-GAN	2.763 sec	4.64 GB	307.50 W

The complexity of loss function in VAE-GAN leads to the extra time required in training. Therefore, the time cost per epoch is the longest. Residual connections in SNGAN(Enhanced) will occupy a large amount of GPU memory for cache, resulting in more than three times memory overhead compared to SNGAN, even with fewer trainable parameters in the network.

Taking energy efficiency, speed, and memory consumption into account, SNGAN and WGAN-DIV are two recommended choices for SAR image synthesis since they also maintain a satisfactory performance in previous evaluations.

5 Conclusions

In this project, the author implements six different Generative Adversarial Networks that are influential in the development of generative deep learning, and tests their corresponding generation ability of synthetic aperture radar (SAR) images based on different evaluation schemes.

The most important scheme, classification evaluation, is conducted to validate the availability of using GANs to generate SAR images to achieve data augmentation under different training sample conditions, and aims to find the optimal choice in each condition. The results show that when the training samples contain 50 SAR images per class, synthetic images by SNGAN(Enhanced) can maximumly promote the recognition accuracy in the ten-class classification test. When only use 10 SAR images per class to train GANs, images generated by WGAN-DIV and VAE-GAN can improve the recognition rate by 11%, which outperforms other models. When the training samples are relatively sufficient, SNGAN(Enhanced) maintains the highest diversity in image generation. Both the generated images of SNGAN(Enhanced) and WGAN-DIV possess a high degree of fidelity to the original training samples.

From visual perspective, SAR images generated by WGAN-DIV, SNGAN, SNGAN(Enhanced) and VAE-GAN are of high degree of similarity to real SAR images in MSTAR dataset. Further conclusion is drawn from statistics that image by SNGAN(Enhanced) owns the highest resilience to real SAR image.

From the perspective of resource consumption, SNGAN and WGAN-DIV can either maintain a good generation performance or maximum save the training overhead.

In summary, except for DCGAN, other five types of GAN models are proved to be available methods for augmenting SAR image. In the author's opinion, WGAN-DIV and SNGAN(Enhanced) are the overall best models. WGAN-DIV always possesses satisfactory performances, no matter how the size of training samples varies. SNGAN(Enhanced) demonstrates its superiority in diversity and fidelity in SAR image synthesis as the number of training samples gradually increases.

References

[1] Moreira, P. Prats-Iraola, M. Younis, G. Krieger, I. Hajnsek, and K. P. Papathanassiou, "A Tutorial on Synthetic Aperture Radar," *IEEE Geoscience and Remote Sensing Magazine*, vol. 1, no. 1, 2013.

[2] X. X. Zhu, S. Montazeri, M. Ali, Y. Hua, Y. Wang, L. Mou, Y. Shi, F. Xu, and R. Bamler, "Deep Learning Meets SAR: Concepts, Models, Pitfalls, and Perspectives," *IEEE Geoscience and Remote Sensing Magazine*, vol. 9, no. 4, pp. 143–172, 2021.

[3] Y. LeCun, C. Cortes, and C. Burges, "The MNIST Database of Handwritten Digits," 1998, [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: Apr. 10, 2023].

[4] L. S. Vailshery, "Geocento SAR Imagery Cost Worldwide 2022," Statista, 04-Mar-2022. [Online]. Available: https://www.statista.com/statistics/1293899/geocento-commercial-satellite-sar-imagery-resolution-cost-worldwide/. [Accessed: Apr. 10, 2023].

[5] Z. Cui, M. Zhang, Z. Cao, and C. Cao, "Image Data Augmentation for SAR Sensor via Generative Adversarial Nets," *IEEE Access*, vol. 7, pp. 42255–42268, Mar. 2019.

[6] S. Du, J. Hong, Y. Wang, and Y. Qi, "A High-Quality Multicategory SAR Images Generation Method with Multiconstraint GAN for ATR," *IEEE Geoscience and Remote Sensing Letters*, vol. 19, pp. 1–5, 2022.

[7] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," *Communications of the ACM*, vol. 63, no. 11, pp. 139–144, 2020.

[8] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks", *arXiv e-prints*, 2015. doi:10.48550/arXiv.1511.06434.

[9] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs", *arXiv e-prints*, 2017. doi:10.48550/arXiv.1704.00028.

[10] Wu, J., Huang, Z., Thoma, J., Acharya, D., and Van Gool, L., "Wasserstein Divergence for GANs", *arXiv e-prints*, 2017. doi:10.48550/arXiv.1712.01026.

[11] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida, "Spectral Normalization for Generative Adversarial Networks," *International Conference on Learning Representations (ICLR)*, 2018.

[12] Boesen Lindbo Larsen, A., Kaae Sønderby, S., Larochelle, H., and Winther, O., "Autoencoding Beyond Pixels Using a Learned Similarity Metric", *arXiv e-prints*, 2015. doi:10.48550/arXiv.1512.09300.

[13] S. R. Bhamidipati, C. Srivatsa, C. Kanakapura Shivabasave Gowda, and S. Vadada, "Generation of SAR Images Using Deep Learning," *SN Computer Science*, vol. 1, no. 6, 2020.

[14] X. Bao, Z. Pan, L. Liu, and B. Lei, “SAR image simulation by generative Adversarial Networks,” *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, 2019.

[15] Q. Song, F. Xu, X. X. Zhu, and Y.-Q. Jin, “Learning to Generate SAR Images with Adversarial Autoencoder,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1–15, 2022.

[16] A. Zhang, Z. Lipton, M. Li, and A. J. Smola, “Dive into Deep Learning”, *arXiv e-prints*, 2021.

[17] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Massachusetts: MIT Press, 2017.

[18] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[19] J. Su, “The Art of Adversarial Learning, from Zero to WGAN-GP,” June 2017. [Online]. Available: https://spaces.ac.cn/archives/4439. [Accessed: Apr. 10, 2023].

[20] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN”, *arXiv e-prints*, 2017. doi:10.48550/arXiv.1701.07875.

[21] J. Su, “Lipschitz Constraints in Deep Learning: Generalization and Generative Models,” Oct. 2018. [Online]. Available: https://spaces.ac.cn/archives/6051. [Accessed: Apr. 10, 2023].

[22] J. Su, “WGAN-DIV: the Culmination of Advancements in WGAN Family,” Nov. 2018. [Online]. Available: https://spaces.ac.cn/archives/6139. [Accessed: Apr. 10, 2023].

[23] Kingma, D. P. and Welling, M., “Auto-Encoding Variational Bayes”, *arXiv e-prints*, 2013. doi:10.48550/arXiv.1312.6114.

[24] J. Su, “GAN Models from an Energy Perspective (I): GAN = ‘Digging a Hole’ + ‘Jumping Into a Hole’,” Jan. 2019. [Online]. Available: https://spaces.ac.cn/archives/6316. [Accessed: Apr. 10, 2023].

[25] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[26] H. Zhang, I. Goodfellow, D. Metaxas and A. Odena, “Self-Attention Generative Adversarial Networks,” *arXiv e-prints*, 2018. doi:10.48550/arXiv.1805.08318.

[27] P. Harrington, *Machine Learning in Action*. Manning Publications, 2012.

[28] T. Rashid, Make your first GAN with pytorch a gentle introduction to generative adversarial networks, and a practical step-by-step tutorial on making your own with pytorch. Erscheinungsort nicht ermittelbar: Verlag nicht ermittelbar, 2020.

[29] Song, J., Meng, C., and Ermon, S., “Denoising Diffusion Implicit Models”, *arXiv e-prints*, 2020. doi:10.48550/arXiv.2010.02502.

[30] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B., “High-Resolution Image Synthesis with Latent Diffusion Models”, *arXiv e-prints*, 2021. doi:10.48550/arXiv.2112.10752.

[31] D. Foster, *Generative Deep Learning*. Sebastopol, CA: O'Reilly Media, Inc., 2019.

[32] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks, "Deep Generative Modelling: A Comparative Review of VAEs, GANs, Normalizing Flows, Energy-Based and Autoregressive Models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7327–7347, 2022.

[33] J. Su, "From Wasserstein Distance and Duality Theory to WGAN," Spaces, Jan. 2019. [Online]. Available: https://spaces.ac.cn/archives/6280. [Accessed: Apr. 10, 2023].

[34] N. Wale, "DCGAN Tutorial," PyTorch Tutorials. [Online]. Available: https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html. [Accessed: Apr. 21, 2023].

[35] E. Lindernoren, "PyTorch-GAN," GitHub. [Online]. Available: https://github.com/eriklindernoren/PyTorch-GAN. [Accessed: Apr. 21, 2023].

[36] W. Wang, "pytorch-gan-collections," GitHub. [Online]. Available: https://github.com/w86763777/pytorch-gan-collections. [Accessed: Apr. 21, 2023].

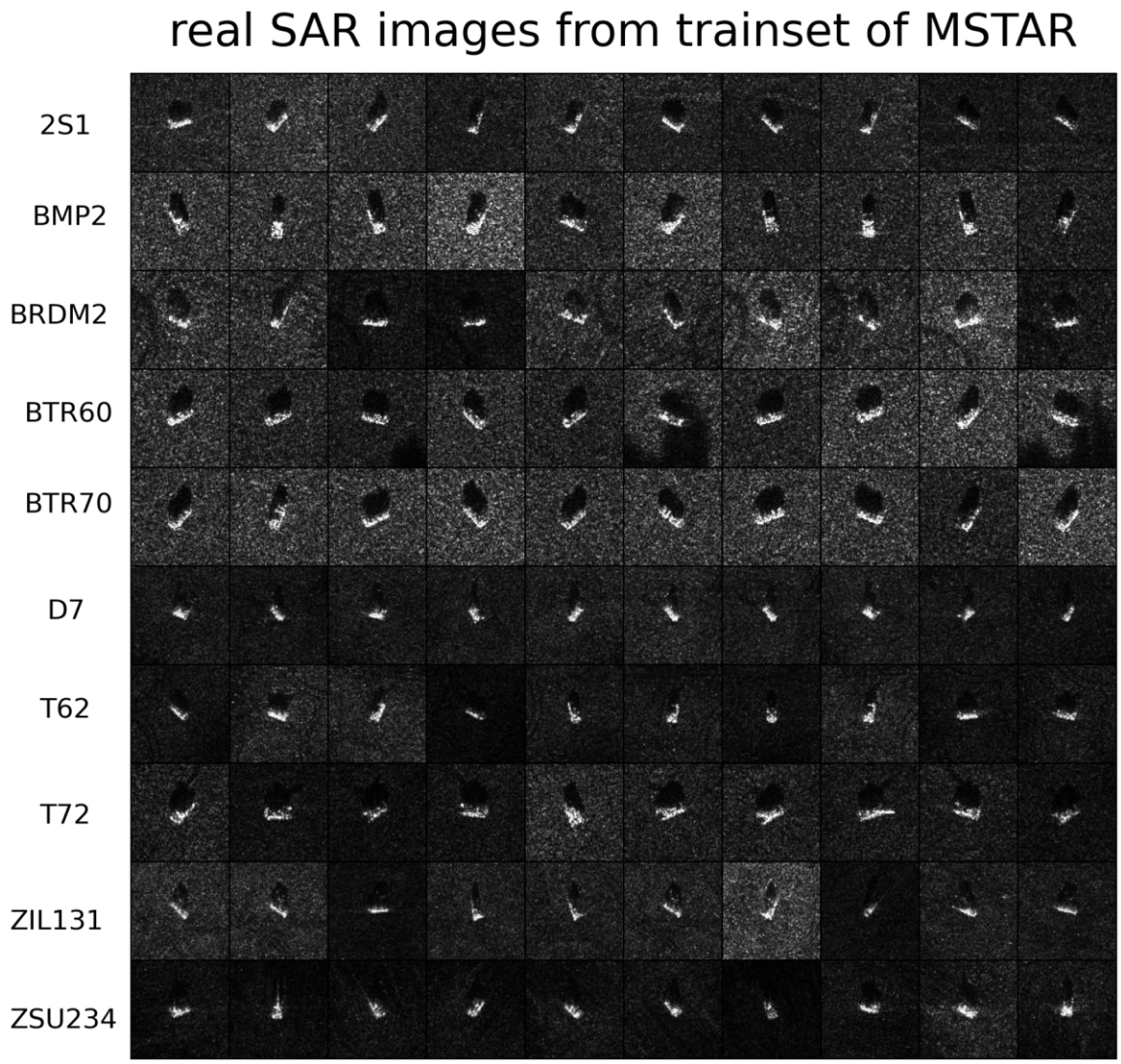
[37] R. Dutt, "VAE-GAN-PYTORCH," GitHub. [Online]. Available: https://github.com/rishabhd786/VAE-GAN-PYTORCH. [Accessed: Apr. 21, 2023].

[38] F. Sayah, "Support Vector Machine & PCA Tutorial for Beginner," Kaggle. [Online]. Available: https://www.kaggle.com/code/faressayah/support-vector-machine-pca-tutorial-for-beginner. [Accessed: Apr. 21, 2023].

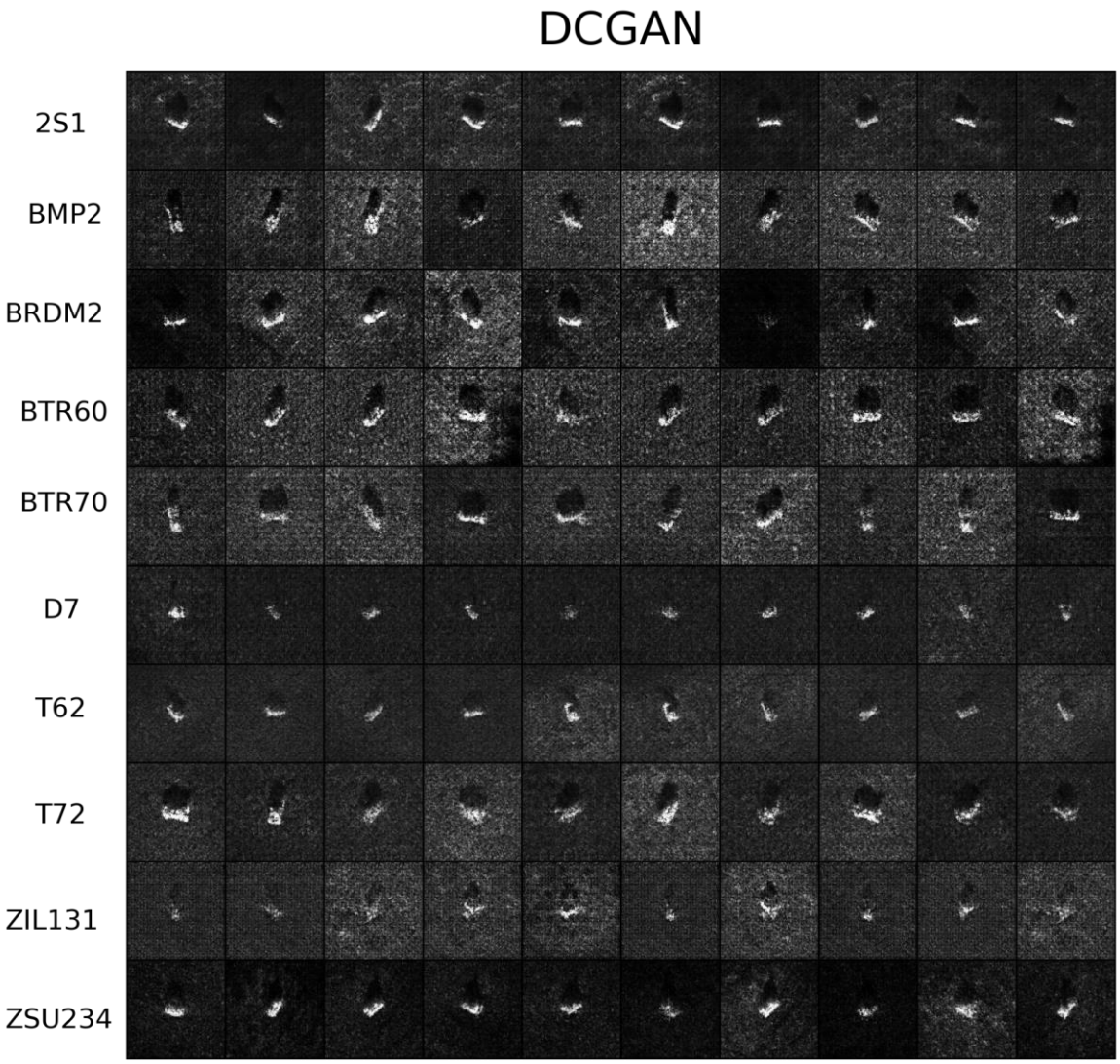
As a software project, the author has written several thousand lines of Python code to facilitate experiments, during which it is inevitable to encounter unfamiliar concepts and challenging problems. The author is deeply appreciated for [34-38], which are open-source codes and tutorials. The selfless sharing of the corresponding authors of [34-38] helps the author a lot to write concise code with high-efficiency. Besides, the author uses ChatGPT as the code assistant, which tremendously improve the working efficiency.

A Appendices

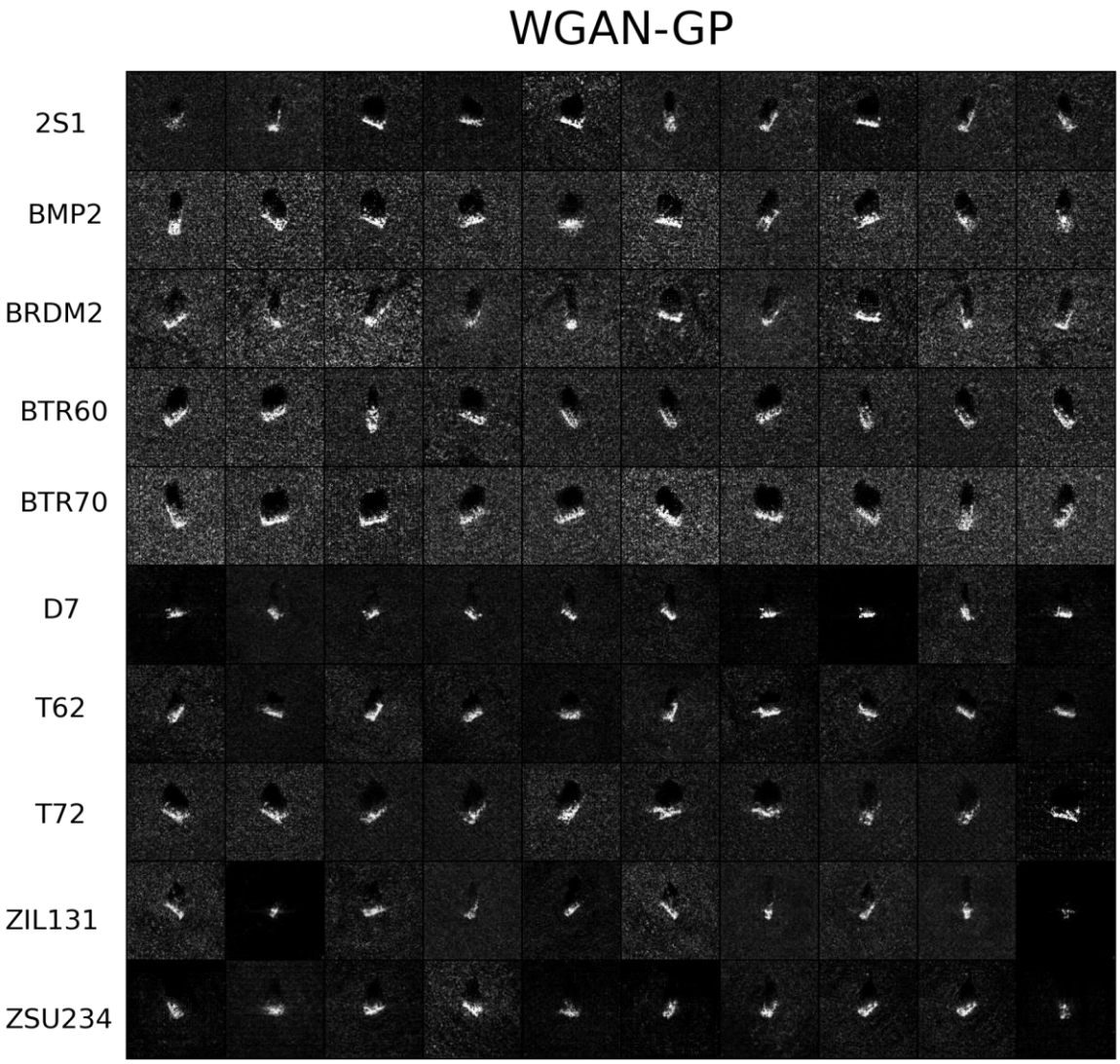
A.1 Supplementary Figures



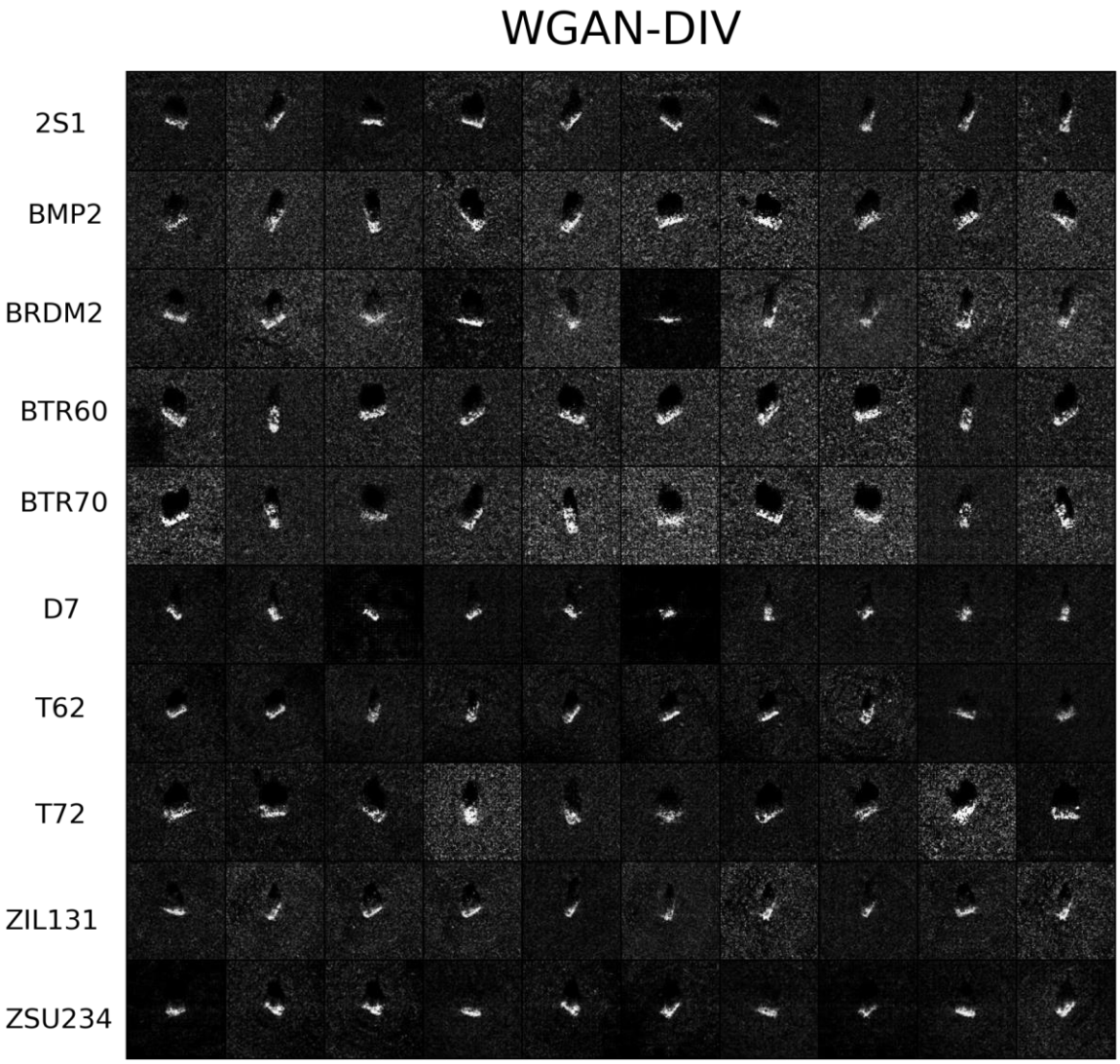
Appx Figure 1 A glance of real SAR images in trainset of MSTAR



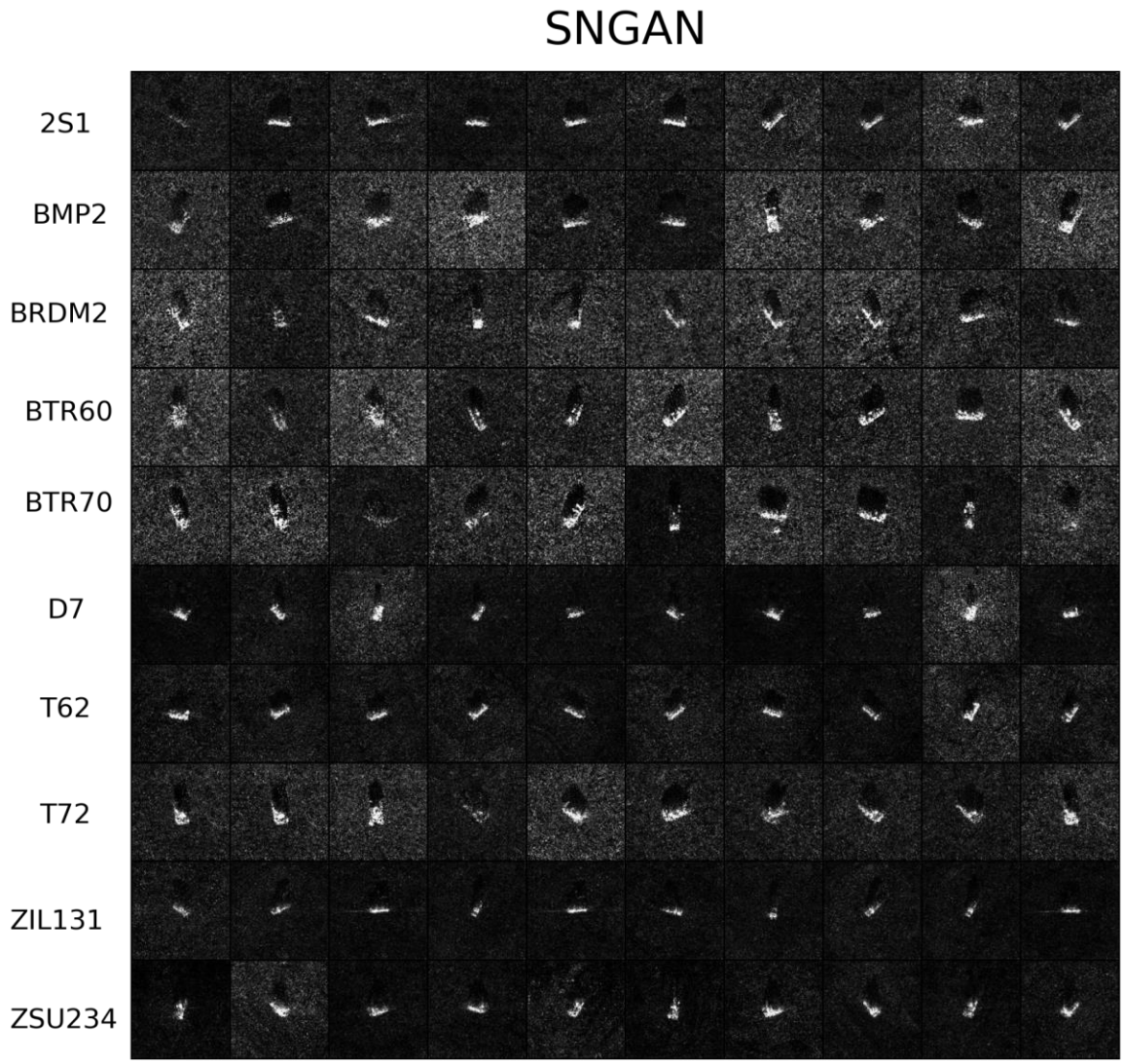
Appx Figure 2 A glance of SAR images generated by DCGAN



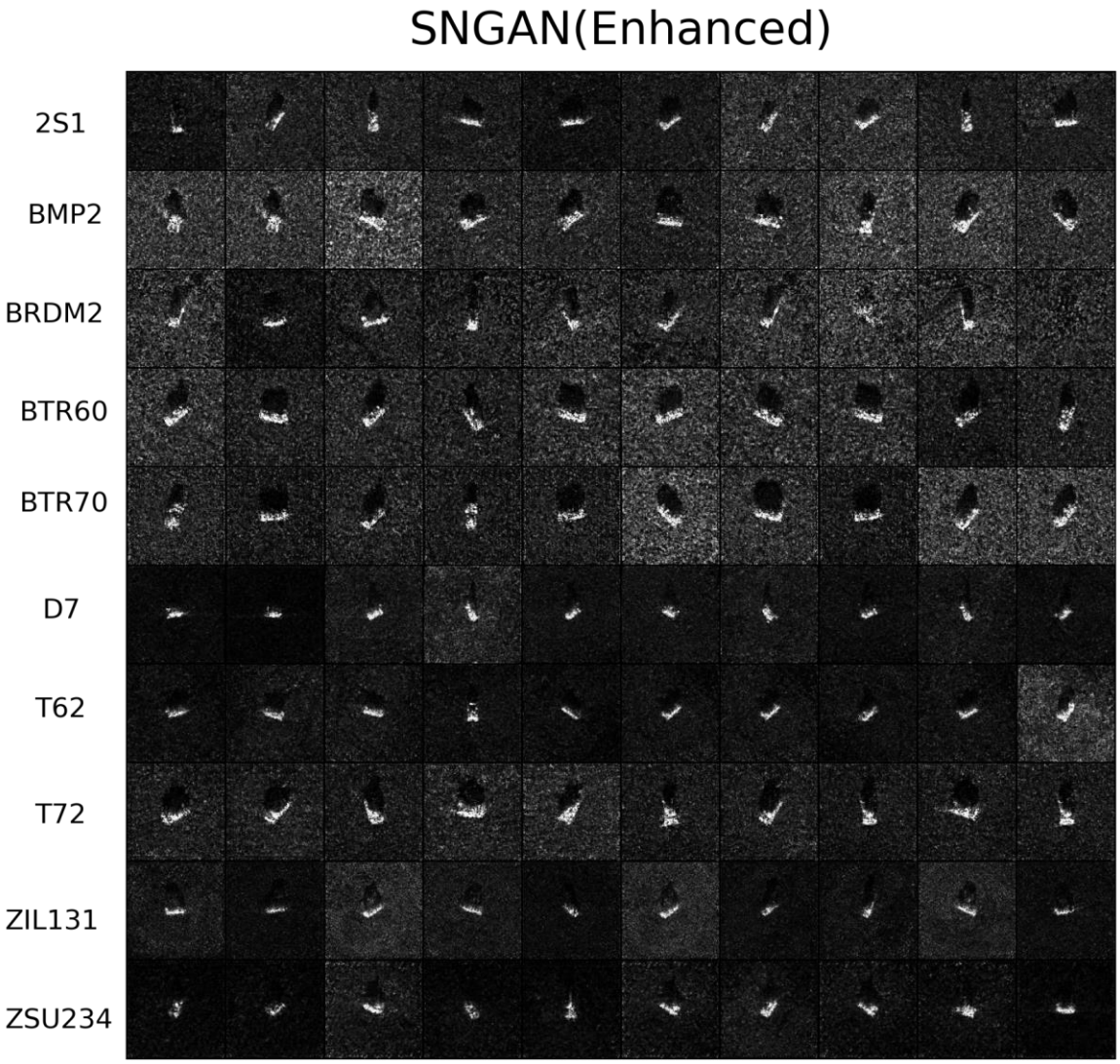
Appx Figure 3 A glance of SAR images generated by WGAN-GP



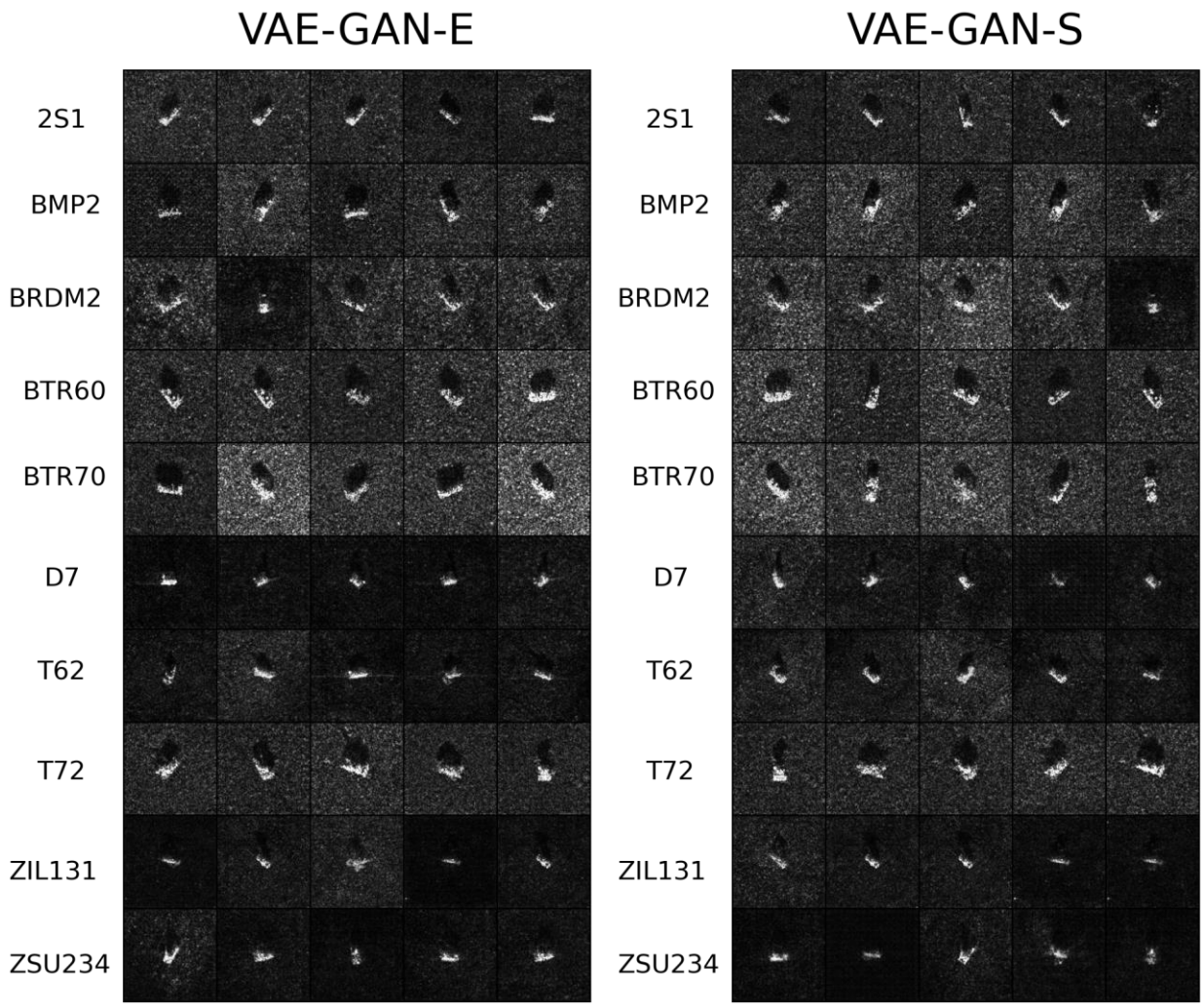
Appx Figure 4 A glance of SAR images generated by WGAN-DIV



Appx Figure 5 A glance of SAR images generated by SNGAN



Appx Figure 6 A glance of SAR images generated by SNGAN(Enhanced)



Appx Figure 7 A glance of SAR images generated by VAE-GAN

A.2 Detailed Description of Implemented GAN Structures

Appx Table 1 Architecture of DCGAN

DCGAN

Discriminator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Conv	1→64	(64,64)	4/2/1	BatchNorm	LeakyReLU
2	Conv	64→128	(32,32)	4/2/1	BatchNorm	LeakyReLU
3	Conv	128→256	(16,16)	4/2/1	BatchNorm	LeakyReLU
4	Conv	256→512	(8,8)	4/2/1	BatchNorm	LeakyReLU
5	Conv	512→512	(4,4)	4/2/1	BatchNorm	LeakyReLU
6	Conv	512→1	(1,)	4/1/0		Sigmoid

Generator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	ConvTranspose	N→512	(4,4)	4/1/0	BatchNorm	ReLU
2	ConvTranspose	512→512	(8,8)	4/2/1	BatchNorm	ReLU
3	ConvTranspose	512→512	(16,16)	4/2/1	BatchNorm	ReLU
4	ConvTranspose	512→256	(32,32)	4/2/1	BatchNorm	ReLU
5	ConvTranspose	256→256	(64,64)	4/2/1	BatchNorm	ReLU
6	ConvTranspose	256→64	(128,128)	4/2/1		ReLU
7	ConvTranspose	64→1	(128,128)	3/1/1		Tanh

Explanations Tables in **Section A.2**:

Neural network structures are shown in dynamic computational graph form. “Type” refers to the type of network layer. “Conv” is convolution layer, “ConvTranspose” is deconvolution layer and “Linear” is fully connected layer. “1→64” in “Dimensions” column refers to the input data dimension is 1 and the output dimension is 64. “Output Shape” is the shape of the output tensor for a network layer. “K/S/P” is short for “Kernel-Size/Stride/Padding”, which only presents in convolution and deconvolution layers, see formula (1) and (2).

Appx Table 2 Architecture of WGAN-GP/WGAN-DIV

WGAN-GP/WGAN-DIV

Discriminator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Conv	1→64	(64,64)	4/2/1	InstanceNorm	LeakyReLU
2	Conv	64→128	(32,32)	4/2/1	InstanceNorm	LeakyReLU
3	Conv	128→256	(16,16)	4/2/1	InstanceNorm	LeakyReLU
4	Conv	256→512	(8,8)	4/2/1	InstanceNorm	LeakyReLU
5	Conv	512→512	(4,4)	4/2/1	InstanceNorm	LeakyReLU
6	Conv	512→1	(1,)	4/1/0		

Generator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	ConvTranspose	N→512	(4,4)	4/1/0	BatchNorm	ReLU
2	ConvTranspose	512→512	(8,8)	4/2/1	BatchNorm	ReLU
3	ConvTranspose	512→512	(16,16)	4/2/1	BatchNorm	ReLU
4	ConvTranspose	512→256	(32,32)	4/2/1	BatchNorm	ReLU
5	ConvTranspose	256→256	(64,64)	4/2/1	BatchNorm	ReLU
6	ConvTranspose	256→64	(128,128)	4/2/1		ReLU
7	ConvTranspose	64→1	(128,128)	3/1/1		Tanh

Appx Table 3 Architecture of SNGAN

SNGAN

Discriminator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Conv	1→64	(64,64)	4/2/1	SpectralNorm	LeakyReLU
2	Conv	64→128	(32,32)	4/2/1	SpectralNorm	LeakyReLU
3	Conv	128→256	(16,16)	4/2/1	SpectralNorm	LeakyReLU
4	Conv	256→512	(8,8)	4/2/1	SpectralNorm	LeakyReLU
5	Conv	512→512	(4,4)	4/2/1	SpectralNorm	LeakyReLU
6	Conv	512→64	(2,2)	4/2/1	SpectralNorm	LeakyReLU
7	Linear	256→1	(1,)		SpectralNorm	

Generator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	ConvTranspose	N→512	(4,4)	4/1/0	BatchNorm	ReLU
2	ConvTranspose	512→512	(8,8)	4/2/1	BatchNorm	ReLU
3	ConvTranspose	512→512	(16,16)	4/2/1	BatchNorm	ReLU
4	ConvTranspose	512→256	(32,32)	4/2/1	BatchNorm	ReLU
5	ConvTranspose	256→256	(64,64)	4/2/1	BatchNorm	ReLU
6	ConvTranspose	256→64	(128,128)	4/2/1		ReLU
7	ConvTranspose	64→1	(128,128)	3/1/1		Tanh

Appx Table 4 Architecture of SNGAN(Enhanced)

SNGAN (Enhanced)

Discriminator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Conv	1→64	(64,64)	4/2/1	SpectralNorm	LeakyReLU
2	Conv	64→256	(32,32)	4/2/1	SpectralNorm	LeakyReLU
3	Conv	256→256	(32,32)	3/1/1	SpectralNorm	LeakyReLU
4	Conv	256→512	(16,16)	4/2/1	SpectralNorm	LeakyReLU
5	Conv	512→512	(16,16)	3/1/1	SpectralNorm	LeakyReLU
6	SelfAttention ConvBlock	512→256	(8,8)			LeakyReLU
7	SelfAttention ConvBlock	256→64	(4,4)			LeakyReLU
8	Conv	64→1	(1,)	4/1/0	SpectralNorm	

Generator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	ConvTranspose	N→256	(4,4)	4/1/0	BatchNorm	LeakyReLU
2	ResidualBlock	256→256	(8,8)			
3	ResidualBlock	256→256	(16,16)			
4	ResidualBlock	256→256	(32,32)			
5	ResidualBlock	256→256	(64,64)			
6	ResidualBlock	256→64	(128,128)		BatchNorm	LeakyReLU
7	Conv	64→64	(128,128)	3/1/1		LeakyReLU
8	Conv	64→64	(128,128)	3/1/1		Tanh

Appx Table 5 Architecture of VAE-GAN

VAE-GAN

Encoder

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Conv	1→8	(64,64)	4/2/1		ReLU
2	Conv	8→64	(32,32)	4/2/1	BatchNorm	ReLU
3	Conv	64→256	(16,16)	4/2/1	BatchNorm	ReLU
4	Conv	256→512	(8,8)	4/2/1	BatchNorm	ReLU
5	Conv	512→512	(4,4)	4/2/1	BatchNorm	ReLU
6	Conv	512→256	(4,4)	3/1/1	BatchNorm	ReLU
7	Linear	4096→1024	(1024,)		BatchNorm	ReLU
8-1	Linear	1024→32	(32,)			Reparameterization
8-2	Linear	1024→32	(32,)			Reparameterization

Decoder (Generator)

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Linear	N→16384	(16384,)		BatchNorm	ReLU
2	ConvTranspose	256→512	(16,16)	4/2/1	BatchNorm	ReLU
3	ConvTranspose	512→512	(32,32)	4/2/1	BatchNorm	ReLU
4	ConvTranspose	512→64	(64,64)	4/2/1	BatchNorm	ReLU
5	ConvTranspose	64→16	(128,128)	4/2/1	BatchNorm	ReLU
6	ConvTranspose	16→1	(128,128)	3/1/1		Tanh

Discriminator

Layer	Type	Dimensions	Output Shape	K/S/P	Normalization	Activation
1	Conv	1→64	(63,63)	5/2/1	BatchNorm	LeakyReLU
2	Conv	64→512	(32,32)	5/2/2	BatchNorm	LeakyReLU
3	Conv	512→256	(16,16)	5/2/2	BatchNorm	LeakyReLU
4	Conv	256→256	(8,8)	5/2/2	BatchNorm	LeakyReLU
5	Conv	256→64	(8,8)	3/1/1	BatchNorm	LeakyReLU
6	Linear	4096→512	(512,)		BatchNorm	LeakyReLU
7	Linear	512→1	(1,)			Sigmoid

A.3 Equipment, Platforms and Tools Involved

All the experiments and implementations in this project is carried on the author's personal computer. Key configuration is as follows:

CPU	Intel Core i7-12700K
Graphic Card	NVIDIA GeForce RTX 4080 16G, CUDA Version == 12.1
Memory	32G DDR4 4800MHz

All the programmes are written in **Python** and **IPython**, the former is run on **PyCharm**, and the latter is run on **JupyterLab**. The author has used several open-source packages and libraries in Python to conduct this project, including but not least to:

PyTorch	to implement GANs and other deep learning models
Scikit-Learn	to implement PCA-SVM classifier and conduct classification evaluation
Pillow/OpenCV	for image processing
Numpy	to assist data processing
Matplotlib	to plot graphs

Other tools involved in this project:

draw.io	to make flow chart and schematic of network architecture
Microsoft Office	to write this report and make PPTs