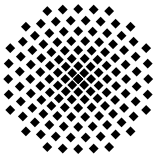


University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Masterarbeit D1513

Range-Doppler map upsampling for single channel chirp sequence radar using Deep Learning

Upsampling der Range-Doppler-Karte für einkanaliges Chirp-Sequence-Radar unter Verwendung von Deep Learning

Author: Zheming Yin

Date of work begin: 01.10.2024
Date of submission: 31.03.2025

Supervisor: Sven Hinderer

Keywords: Range-Doppler map, Image upsampling, Transformer, cGAN, Data collection, Data processing, Loss combination

Contents

Abbreviations

Adagrad	Adaptive gradient algorithm
Adam	Adaptive Moment Estimation
ADC	Analog-to-Digital Converter
API	Application Program Interface
BGD	Batch Gradient Descent
CFAR	Constant False Alarm Rate
cGAN	Conditional Generative Adversarial Networks
CMGAN	Conformer-Based Metric-GAN
CNN	Convolutional Neural Network
CPI	Coherent Processing Interval
CRI	Chirp Repetition Interval
DC	Direct Current
DFT	Discrete Fourier Transform
DP	Dual-Path
FFT	Fast Fourier Transform
FFW	Feed-Forward
FM	Frequency Modulated
FMCW	Frequency Modulated Continuous Wave
FOL	Frame Of Length
GAN	Generative Adversarial Networks
GPU	Graphics Processing Unit
HQ	High-Quality
IF	Intermediate Frequency
iFFT	Inverse Fast Fourier Transform
iRDP	Inverse Range-Doppler Processing
irFFT	Inverse Real-valued Fast Fourier Transform
ISS	Institut für Signalverarbeitung und Systemtheorie
I/O	Input/Output
LN	Layer Normalization
LSD	Logarithmic Spectral Distance
LQ	Low-Quality
MLP	Multi-Layer Perceptron
MIMO	Multiple input, multiple output

MSA Multi-head Self-Attention
MSE Mean Square Error

PLSD Phase-aware Logarithmic Spectral Distance
PMSE Pixel-wise Mean Squared Error
PSNR Peak Signal-Noise Ratio

RA Range-Azimuth
RDP Range-Doppler Processing
ReLU Rectified Linear Unit
rFFT Real-valued Fast Fourier Transform
RMSProp Root Mean Square Prop
RNN Recurrent Neural Networks
RSTB Residual Swin Transformer Blocks
Rx Receiver

SAR Synthetic Aperture Radar
SDR Signal-to-Distortion Ratio
SDK Software Development Kit
SD-SDR Scale-Dependent Signal-to-Distortion Ratio
SGD Stochastic Gradient Descent
SI-SDR Scale-Independent Signal-to-Distortion Ratio
SNR Signal-to-Noise Ratio
STFT Short-Time Fourier Transform
STL Swin Transformer Layers
Swin Shifted Windows
SwinIR Image Restoration using Swin Transformer

TE Transformer Encoder
TF Time-Frequency
TFRecord TensorFlow Records
Tx Transmitter

VGG Very Deep Convolutional Networks

WandB Weights and Biases
WMSE Weighted MSE

Abstract

During the extensive usage of chirp sequence radar, it is often affected by the cost, system limit or regulations, resulting in limited resolution of the range-Doppler map. Meanwhile, the range-Doppler map collected in the indoor environment also puts higher requirements on the model in terms of complexity and high amplitude fluctuation, whereas the current upsampling approaches cannot meet this requirement well. Therefore, this thesis will combine the current upsampling models in the state of the art, such as Transformer and cGAN models, with a series of data processing methods, such as logarithm and normalization operations, and the combination of multiple loss functions to improve the quality of the super-resolution range-Doppler map. In addition, the environmental conditions and data in the public datasets about the range-Doppler map are currently simple, so the dataset of this thesis is collected in the indoor environment by ourselves. In the evaluation phase, we obtain an optimized combination of model, processing methods and loss functions and tune the hyperparameters of the model, which significantly improves the result compared to the other image upsampling approaches.

Kurzfassung

Bei der umfangreichen Nutzung von Chirp-Sequence-Radar ist es häufig von den Kosten, Systembeschränkungen oder Vorschriften beeinflusst, was zu einer limitierten Auflösung der Range-Doppler-Karte führt. Gleichzeitig stellt die in Innenräumen erfasste Range-Doppler-Karte höhere Anforderungen an das Modell hinsichtlich Komplexität und hoher Amplitudenfluktuation, denen aktuelle Upsampling-Methoden nicht gut gerecht werden können. Daher kombiniert diese Masterarbeit aktuelle fortgeschrittene Upsampling-Modelle, wie Transformer- und cGAN-Modelle, mit einer Reihe von Datenverarbeitungsmethoden wie Logarithmierung und Normalisierung sowie die Kombination mehrerer Verlustfunktionen, um die Qualität der hochauflösenden Range-Doppler-Karte zu verbessern. Da die Umgebungsbedingungen und Daten in öffentlichen Datensätzen zur Range-Doppler-Karte derzeit noch einfach sind, wurde der Datensatz dieser Arbeit selbst in Innenräumen erfasst. In der Evaluierungsphase erzielen wir eine optimierte Kombination aus Modell, Verarbeitungsmethoden und Verlustfunktionen und optimieren die Hyperparameter des Modells, was das Ergebnis im Vergleich zu anderen Bild-Upsampling-Methoden deutlich verbessert.

1. Introduction

This chapter introduces the chirp sequence radar and the meaning of the range-Doppler map. The motivation and structure of this thesis are subsequent and the state-of-the-art models are explained in this chapter as well.

1.1. Chirp sequence radar

Chirp sequence radar is used in the radar system based on Frequency Modulated Continuous Wave (FMCW) technology, widely used in the field of automotive driving, target detection, and environmental sensing and so on [1]. Chirp sequence radar transmits multiple consecutive chirp signals, receives the echoes, and calculates target information such as range and velocity, which can be used to generate a range-Doppler map.

FMCW radar

In FMCW radar, the chirp frequency increases linearly with time, of which the signal is called the linear Frequency Modulated (FM) signal. Figure 1.1 illustrates the change of a linear FM chirp in amplitude and frequency over time. The signal frequency will increase from the initial value f_c by the bandwidth B during the duration of one chirp T_c. After a time interval, the other chirp will be transmitted at time T_p. The rate of the frequency change is shown as the slope, namely $S = B/T_c$.
With this property, we can get the range and velocity of the other targets related to the FMCW radar. There will be a delay between the signal being transmitted and being received by the radar, which is proportional to the range. Since the signal is a round trip motion, the range between the radar and the target can be defined as

$$r = c \cdot \frac{\tau}{2}, \tag{1.1}$$

where c represents the speed of light, namely 3×10^8 m/s, and τ represents the propagation delay. The Doppler effect refers to the phenomenon that the frequency of the received signal and the transmitted signal will be different when there is a relative velocity between the signal source and the target. The relationship between the frequency of the transmitted signal f and the received signal f' is

$$f' = f\left(\frac{c \pm v_o}{c \mp v_s}\right), \tag{1.2}$$

where v_0 and v_s represent the relative velocity of the radar and target. When they move towards each other, the frequency of the received signal increases, and vice versa. In the collected dataset, the radar as the signal source is moving, while the surrounding objects are stationary in most cases.
Meanwhile, as the reflected signal is captured by the radar, it will be mixed with the trans-

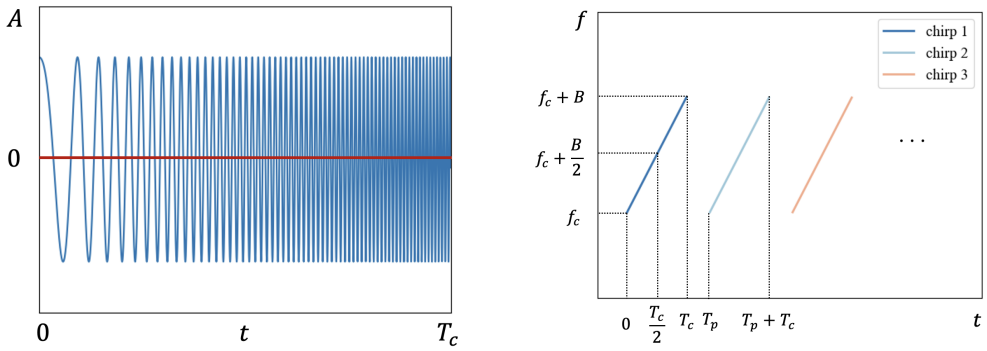


Figure 1.1.: The amplitude and frequency variation of FMCW radar in time respectively.

mitted signal in the system to obtain a new signal called Intermediate Frequency (IF) signal. The frequency of this IF signal is theoretically not only depending on the frequency shift f_r caused by the propagation distance but also the Doppler frequency f_d, according to [1] written as

$$f_{\text{IF}} = f_r - f_d = S \cdot \frac{2r}{c} - \frac{2v}{\lambda}, \tag{1.3}$$

but in our case, the Doppler frequency f_d in the beat frequency can be ignored due to the property of the chirp sequence radar, namely

$$f_{\text{IF}} = f_r = S \cdot \frac{2r}{c}, \tag{1.4}$$

where the reason will be shown in the next explanation of the chirp sequence radar.

Chirp sequence radar

The chirp sequence radar is a FMCW radar with the steep frequency ramps [2]. The dataset in this paper are collected by a single channel chirp sequence radar [3], which means only a single transmitter and a single receiver are used. In section 2.1, the hardware of the chirp sequence radar will be explained in detail, where there are a single transmitter and three receivers in the radar device. The advantages of using the single channel are following, on one hand, the Multiple input, multiple output (MIMO) radar will cause a higher cost, hence the single channel radar system reduce the cost to make the localization system cheaper. On the other hand, the signal needs to do the Fast Fourier Transform (FFT) to convert the signal from time domain into the frequency domain, and the computational complexity will be decreased accordingly.

Due to the difference in the rate of frequency change, the advantage of chirp sequence radar is removing the Doppler frequency part from the IF signal as mentioned above. The reason is that with the short chirp, i.e. the high value of the slope S, the frequency shift f_r will be much larger than the Doppler frequency f_d part. With this property, the radar system will be easy to de-couple the range and velocity estimation, where the range can be calculated by the IF signal frequency. The estimation of the range is within each chirp which is called the fast time axis, whereas the velocity is calculated by the frequencies of the transmitted and received signals according to the Doppler shift between the chirps which is called the slow time axis, since the time interval between the samples within the chirp is quite short.

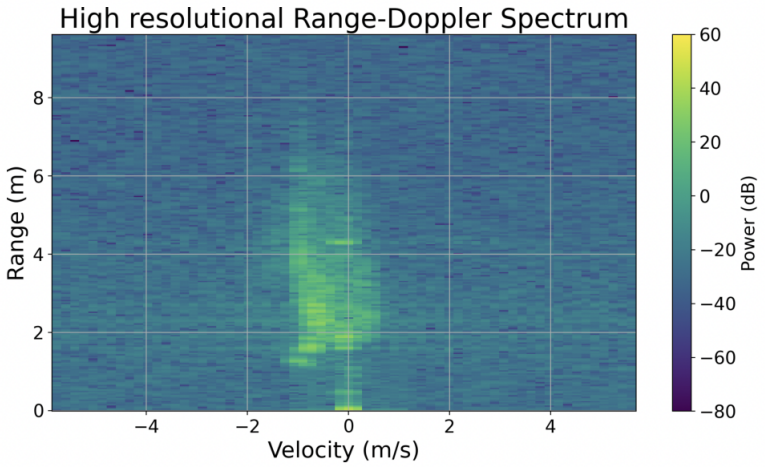


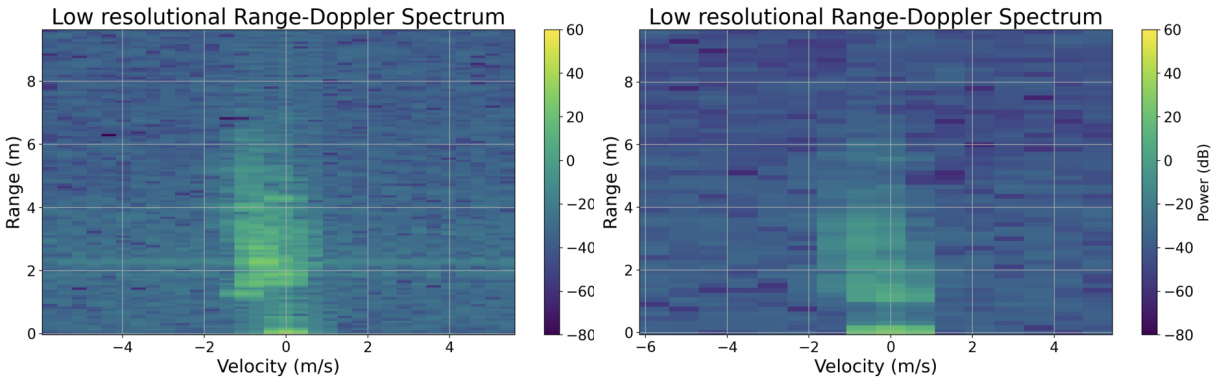
Figure 1.2.: An example of the range-Doppler map.

Range-Doppler map

According to the above, the chirp sequence radar based on the FMCW radar can also obtain information about the range and velocity. Meanwhile, there is a corresponding relationship between the range and velocity, which can be drawn through the range-Doppler map. The derivation and visualization process will be shown in detail in section 2.4, where the range-Doppler map looks like the one in Figure 1.2.
In the range-Doppler map, the horizontal and vertical axes denote velocity and range respectively. In our dataset, the range is calculated from 0 m to roughly 10 m, and the velocity changes from roughly -5 m/s to 5 m/s. At each point where the range and velocity intersect, the value represents the amplitude about the signal power. When the amplitude is converted to dB unit, the value range is between -80 dB and 60 dB, and the corresponding color in the bar gradually changes from dark to light.

1.2. Motivation

As mentioned above, the chirp sequence radar can achieve a larger bandwidth and higher range resolution, but in practical applications, many factors will limit the resolution, such as the hardware of the radar and Coherent Processing Interval (CPI). The carrier frequency and the bandwidth are fixed values in a radar system, since the large bandwidth or high frequency put a higher requirements on the hardware, such as Analog-to-Digital Converter (ADC), antennas and so on, if the cost of the radar system is limited, that is, the frequency or bandwidth are limited, both range and velocity resolutions are influenced. The carrier frequency used by the radar system is affected by many other factors as well, such as regulation. CPI refers to the time interval in which the radar continuously observes the targets and performs signal processing. The signal during this time period is coherent, that is, the phase of the signal remains stable and does not change significantly. CPI is inversely proportional to the Doppler frequency resolution. In this process, the longer the CPI, the more accurate the Doppler frequency resolution can be obtained which leads to a more accurate velocity estimation. However, in many cases, due to the movement of the targets or the ego-motion of the radar itself, the CPI is limited, which in turn affects the velocity resolution.



(a) In halved resolution

(b) In quarter resolution

Figure 1.3.: Examples of the low-resolution range-Doppler map with different downsampling factor related to the high-resolution range-Doppler map in Figure 1.2.

When Figure 1.2 is regarded as a high-resolution range-Doppler map, then in reality, the resolution of range and velocity may be only half or even a quarter of that, resulting in a lot of information loss, as shown in Figures 1.3, where Figure 1.3a shows the effect of the range-Doppler map with the halved resolution, whereas Figure 1.3b shows a quarter resolutional range-Doppler map.

However, high-resolution range-Doppler maps are crucial and have a wide range of applications in areas such as the indoor localization. In order to accurately process and utilize the range-Doppler map, such as Constant False Alarm Rate (CFAR) detection, a high-resolution range-Doppler map is wanted. A traditional, simple and parameter-free method for upsampling is (bi-)linear interpolation. With the development of deep learning, it has shown a strong capability to upsample data to the high resolution. The common structures and models include Convolutional Neural Network (CNN), Encoder-Decoder architecture, Transformer, Conditional Generative Adversarial Networks (cGAN) and so on [4]. Meanwhile, there are also multiple commercial super-resolution solution, such as the super-resolution system in video games provided by NVIDIA [5].

However, the current upsampling approaches are mainly based on images or videos generally, and rarely specifically based on the range-Doppler map of the indoor environment. Compared with the daily images, the range-Doppler map has several significant characteristics. One is that the data in the range-Doppler map have relatively wide dynamic range, especially in the complex environment such as the indoor environment. According to Figure 1.2, the amplitude can be as high as 40 dB in places with a closer range, while the noise will be lower than -60 dB in places farther away from the radar. Furthermore, the values in the range-Doppler map are not as visually coherent as those images in the daily life. In addition to the higher values in the parts with closer range, there are usually large differences between even two adjacent points in the surrounding area, which asks for the special requirements on data processing and structure of the model. We would like to process the data so that its dynamic range is relatively smaller. Meanwhile, since each point in the range-Doppler map contains a variety of information such as velocity, range and power, there is a certain correlation between these informations. We hope that the model can learn the connection and relationship between different areas and use deep learning to achieve better upsampling effect than the traditional methods, such as interpolation, and the models commonly used in

computer vision.
In addition, another challenge is that our goal is not just based on the closer part, i.e., the area with higher amplitude, but the entire range-Doppler map, including the signals with the lower amplitude. Therefore, in data processing, the purpose of making the range of amplitude smaller not only reduces the dynamic range of the data, but also allows the lower amplitude signals to receive attention.
This thesis will mainly focus on three parts: dataset, models, and loss functions. We will use the radar in the Institut für Signalverarbeitung und Systemtheorie (ISS) at the University of Stuttgart to collect the data of the range-Doppler maps and process the dataset. The pipeline will be built to load and process dataset, build and train models, and evaluate them. We will also choose the best data processing methods and the combination of loss functions as well to achieve better upsampling effects for range-Doppler maps.

1.3. Structure of the thesis

The thesis will consist of seven chapters. Chapter 1 focuses on the introduction of the content in the title and the explanation of some basic concepts, which are completed in section 1.1. Section 1.2 introduced the motivation of the upsampling task based on the range-Doppler map, the challenges, and the ultimate goal of this thesis. Section 1.3 shows the main structure of the entire thesis and each section. Section 1.4 shows the current upsampling approaches in the state of the art, including but not limited to upsampling models for range-Doppler maps. Chapter 2 will focus on the data part. Section 2.1 will introduce the information about the data collection, including the hardware of the radar used, the parameters of radar initialization, the settings during the collection process, such as the motion and environmental conditions, and the size of the collected dataset. Moreover, in order to meet our needs, the parameters of the radar as well as the resolution equations will be derived and calculated during the initialization. Section 2.2 will show the process of dataset loading, using TensorFlow Records (TFRecord) files to store data and speed up the loading and training process, and the setting of parameters while creating TFRecord files, such as the number of frames, where we use a sliding window for that and the augmentation. In addition, the resampling operation of the data will be demonstrated. Based on the collected high-resolution data, i.e., the ground truth, we create paired low-resolution and high-resolution tuples for training and evaluation by downsampling the high-resolution signals to their respective low-resolution representation. Meanwhile, since the visualization of the range-Doppler map is in the frequency domain, the conversion between the time domain and the frequency domain for the range-Doppler map will also be demonstrated. Section 2.3 will show the processing methods on the models' inputs and outputs. Since the model can only deal with the real value rather than complex value, it could be converted into the concatenation of the real and imaginary part or of the amplitude and phase part, and they can be also normalized or applied logarithm operation to reduce the dynamic range.
Chapter 3 is mainly about the deep learning models used, including a basic CNN model, a classical UNet structure and a UNet model that concatenates the decoder part with the encoder part, and two different architectures using Transformer, called Dual-Path (DP)-Time-Frequency (TF) Transformer model and Image Restoration using Swin Transformer (SwinIR) architecture, respectively in section 3.6 and 3.7. The Transformer architectures have also two types, DP-TF Transformer block and Shifted Windows (Swin) Transformer block. Section

3.8 will compare the differences between the Transformer architectures and blocks. Section 3.9 will use the above models to design a cGAN model which contains a generator and introduces a discriminator to implement the adversarial model. Furthermore, in the process of converting data from the time domain to the frequency domain, the range dimension will turn to an odd value, such as from 512 to 257, which will cause problems when the model divide the data into the patches or do the downsampling. Therefore, section 3.1 will focus on this problem and propose two methods. Additionally, the model must undergo an upsampling layer before the output. There are currently two main methods, which will be introduced in section 3.2.
Chapter 4 focuses on various loss functions in both training and evaluation phases, and based on the effects of different loss functions, section 4.5 will combine them to complement each other. Chapter 5 will show some methods used in the pipeline to optimize the entire training process, such as using multiple GPUs for distributed training to speed up the training process in section 5.1, some settings in the pipeline in section 5.2, and the hyperparameters which can be tuned by the sweep function of the tool WandB shown in section 5.3.
Finally, Chapter 6 shows the impact of multiple settings, such as different models, processing methods as well as loss functions, on the evaluation loss results, the effect of the super-resolution range-Doppler maps and analyzes them. Chapter 7 summarizes the results of the entire thesis and gives the outlook on the further research.

1.4. Overview of the state of the art

The state-of-the-art super-resolution methods mainly focus on the supervised learning algorithms [4]. Currently, the models and structures used in the super-resolution field mainly include CNN, residual-based methods, encoder-decoder structure, GAN model, etc [4]. In addition, Transformer has a strong ability in terms of translation task and wording understanding. The model uses Attention mechanism to understand the association and relationship between different words and phrases, so that it can understand longer sentences than Recurrent Neural Networks (RNN) [6]. Taking advantage of this feature, the Swin Transfomer designs a shifted window based on Transformer to divide the data patches and shift windows between them, then different windows can also learn the relationship between each other [7].

Encoder-decoder structure

The encoder-decoder structure includes two parts. First, it will use encoder to extract features, learn the extracted features, and then use the decoder part to restore the input based on those learned features, so that the model can get the required output. A common structure is called UNet. Olaf et al. apply UNet to segment medical images [8]. According to Figure 1.4, the structure of their model is U-shaped. The left side of "U" is the encoder part, that is, the features of the data are extracted and learned, and the right side is the part of the decoder which uses the learned features to restore the data back the original size, while segmenting the objects in the image.
In the field of super-resolution data, Akarsh et al. used UNet to upsample the point clouds of mmWave radar [9]. A structure similar to residual path is used in the UNet model, concating the decoder part with the corresponding position of the encoder to alleviate the information loss. The pixel-wise loss and dice loss are combined in the loss function to improve both

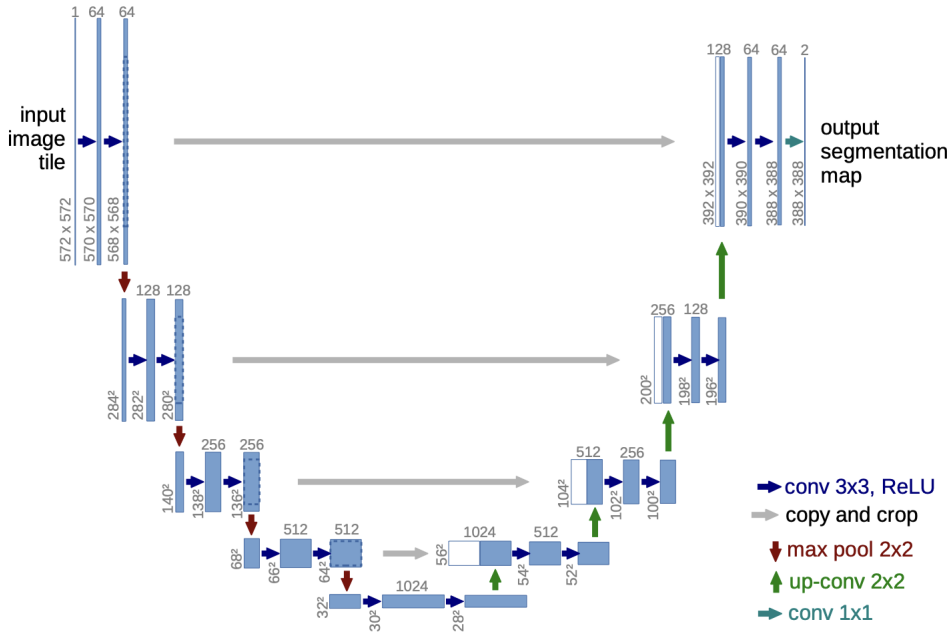


Figure 1.4.: UNet architecture [8]

pixel-wise accuracy and the clarity of the boundary. In the evaluation part, the paper used different environments to test the robustness of the model. Li et al. also applied UNet to the Range-Azimuth (RA) map collected by FMCW radar with MIMO [10]. They have much data processing, inclusive converting the data with the real and imaginary parts, using FFT to convert the data to a more informative map, and choosing the pixel shuffle method for upsampling.

Transformer & Swin Transformer

In Transformer, the encoder-decoder structure exists as well. There are six identical layers in the encoder part, each layer includes a multi-head attention and a forward propagation sub-layer. According to the attention layer shown in Figure 1.5, it will calculate the similarity between the query, key and value, thereby obtaining the relationship between different words. In the upsampling task, we will use the self-attention mechanism, that is, query, key and value all come from the same data, thereby obtaining the relationship between the patches. It is on this basis that the Swin transformer adds shifted windows to achieve that there can be overlapping between windows to improve the uncertainty brought about by fixed split [7]. Liang et al. proposed the SwinIR architecture based on the Swin transformer, adding shallow feature extraction and High-Quality (HQ) image restoration [11]. The Low-Quality (LQ) image will be downsampled once and then passed through multiple Swin transformers and residual blocks to improve the image resolution.
The author of [12] proposed a novel approach to separate two signals, which also uses Transformer model, but split the signals along the two coordinates of time and frequency respectively. In addition, due to two signals, it uses a dual-path structure to achieve better distinction between different signal sources. Inspired by this division method, since there are also two obvious dimensions in the range-Doppler map, namely the range and the Doppler effect, we can also try to apply Transformer along these two axes respectively, and the overlapping will exist in this segmentation approach as well.

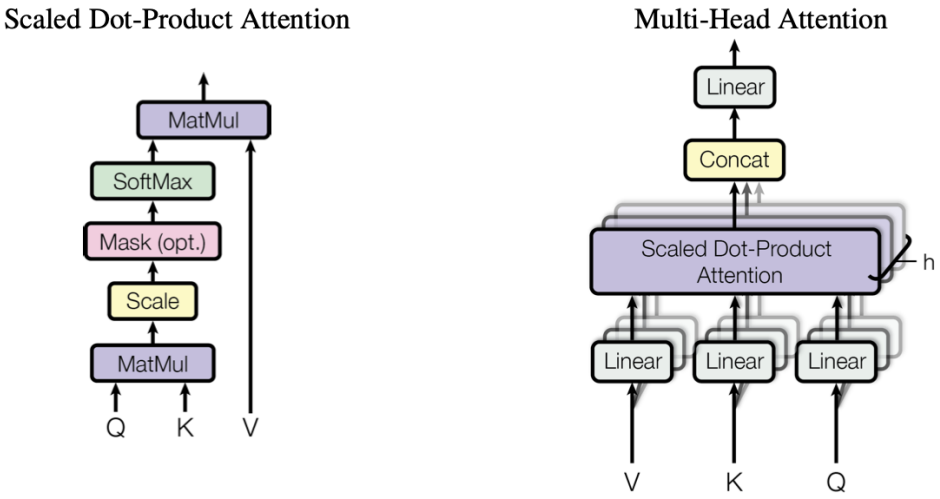


Figure 1.5.: Attention mechanism [6]

GAN & cGAN models

The Generative Adversarial Networks (GAN) model also contains two parts, called generator and discriminator. The generator part will extract and learn the data distribution of the input. After generating an output, the discriminator estimates the probability of real and generated samples belonging to the real class. The goal is to make the output of generator as visually close to the ground truth as possible. On the other hand, it is hoped that the discriminator can enhance its discrimination ability and jointly improve the learning ability of the model through adversarial learning between these two parts [13].

Based on the GAN model, Mehdi et al. proposed a conditional version of GAN model, called Conditional Generative Adversarial Networks (cGAN) [14]. cGAN will feed additional data in the generator and discriminator parts, which is called the condition, as the "y" demonstrated in the Figure 1.6. While in our case, the input of the generator is just the low-resolution data as the condition without the noise part. The discriminator will give the probability between the generated super-resolution and real high-resolution range-Doppler maps based on the conditional low-resolution range-Doppler map, which is well shown in Figure 1.7.

In the field of image super-resolution remedy, Karim et al. used the GAN model on the data of FMCW radar, which was based on the micro-Doppler signature of walking human targets [16]. The GAN model consists of two parts: the discriminator and the CasNet generator, where the CasNet generator is a stack of multiple UNet models. The loss function combines the pixel-wise L1 loss and the perceptual loss obtained by the discriminator. Compared with the task in this thesis, the content of our range-Doppler map is more complicated and the dynamic range of the data is more obvious.

With the usage of cGAN, Kong applied it to the super-resolution of Synthetic Aperture Radar (SAR) imagery [17]. The structure of UNet is used in the generator, downsampling is performed through the convolutional layer and upsampling is performed through transposed convolutional layer, and an attention mechanism is performed after each dimension change. Furthermore, multiple losses are combined in the loss function, including GAN loss, VGG perceptual loss and feature matching loss. Inspiration from this, multiple loss functions

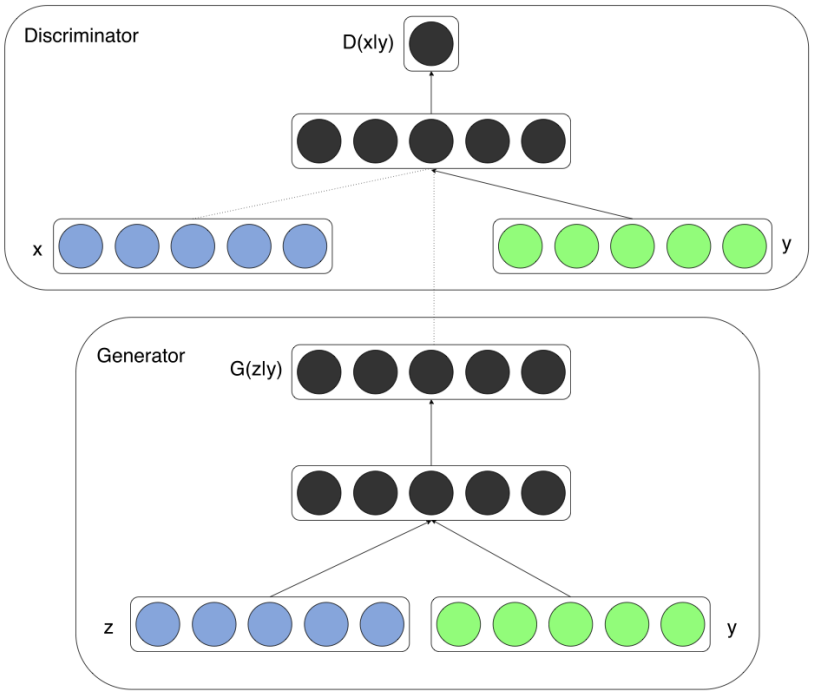


Figure 1.6.: cGAN architecture, where x is the ground truth or the generated output, z represents the noise and y is the condition of the model [14].

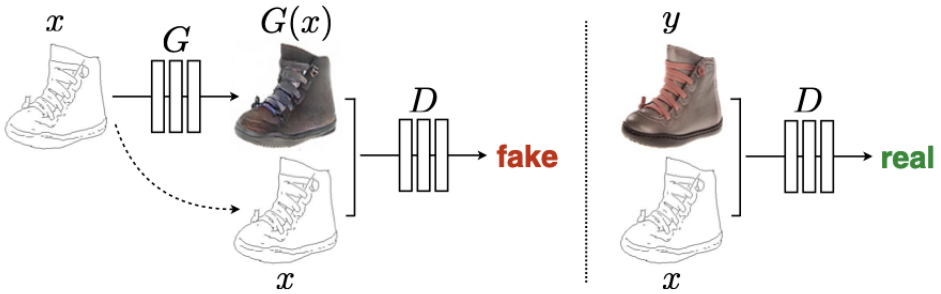


Figure 1.7.: cGAN architecture without noise input for the generator, where x is the conditional low-resolution data and y is the ground truth [15].

can also be combined in our training process to achieve better results. Sherif et al. used the Conformer-Based Metric-GAN (CMGAN) model, combining the Transformer with the GAN structure to learn long-distance dependencies and the convolutional layers to exploit local features, which can tackle a variety of tasks, including super-resolution processing of speech [18].
Also in the field of super-resolution range-Doppler map, Jeong et al. used the cGAN architecture, in which the generator was based on the UNet model [19]. In the evaluation, more metrics are used, such as Pixel-wise Mean Squared Error (PMSE), Peak Signal-Noise Ratio (PSNR), etc. However, since the data was collected mainly in open areas such as the playground, the content in the range-Doppler map is relatively simple. Ledig et al., in their SRGAN model, a GAN for super-resolution image, used a larger resampling rate, namely factor of four, which puts higher requirements on the ability of the model [20]. Meanwhile, they combine perceptual loss, adversarial loss and content loss to prevent the loss of high-frequency information and ensure that the boundaries of objects are clear.

2. Dataset

Dataset is very important for the training. Sufficient data that meets the requirement of the task is crucial, and the proper preprocessing can effectively help the model learn well. Based on the complexity of the indoor environment, which leads to the wide dynamic range of the data amplitude, we choose to collect the required dataset by ourselves so that the model can better adapt to complex range-Doppler map.

2.1. Dataset recording

Radar hardware

In the collection of the dataset, we used the type of BGT60TR13C radar from Infineon [3]. The BGT60TR13C type is a 60 GHz radar sensor with four integrated antennas, which includes single Transmitter (Tx) and three Receivers (Rxs), where the receivers are arranged in L-shape as shown in Figure 2.1. In the task, due to the single channel chirp sequence radar, one of the three receivers has to be chosen. Rx3 is used by default during the recording, since Rx1 and Rx2 are closer to the Tx than Rx3, it means that the interference between signals will increase, and the noise will increase accordingly. On the other hand, using different receivers can increase the variety of data to a certain extent, so Rx1 is also sometimes used in the collection process.
The advantage of the BGT60TR13C radar is that it can achieve ultra-wide bandwidth FMCW operation in a small package, which is conducive to provide higher resolution, retain more details and provide more possibilities in data, such as the downsample as the factor of 4. With a bandwidth of 5.5 GHz, the minimum range resolution is 3 cm. Furthermore, the high Signal-to-Noise Ratio (SNR) of this radar can achieve detection of up to 15 m, providing a good hardware foundation for the movement, identification and segmentation of objects within this range, where more parameters are given in Table 2.1.

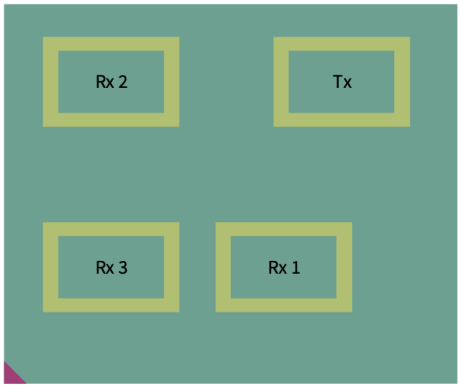


Figure 2.1.: The antennas arrangement of the radar device [3]

Table 2.1.: Parameters of the BGT60TR13C radar product [3]

Parametrics	BGT60TR13C	Parametrics	BGT60TR13C
Angle of Arrival	Yes	Current consumption	200 mA
Motion	Yes	Supply voltage	1.8 V
Frequency	60 GHz	Gain	5 dBi
Min. frequency	58 GHz	Max detection range	15 m
Max. frequency	63.5 GHz	Min detection range	0.2 m
Range	Yes	Speed	Yes
Number of Rx antennas	3	Number of Tx antennas	1

While using this BGT60TR13C radar, Infineon provides a corresponding radar Software Development Kit (SDK), which can access radar sensors and collect data through C/C++, Python and MATLAB, and can implement some radar algorithms such as range-Doppler map, simple segmentation, etc. with the SDK. In addition, Infineon also provides wrappers that adapt to multiple platforms, such as MacOS, Linux, etc. For specific instructions, the developer community and other platforms can be referred [21].

Initial device parameters

The parameters and settings during the data collection are mainly divided into three parts: the initial parameter settings of the radar, the time interval during radar operation, and the environmental conditions for collection. This part is about the parameters derivation of the radar.

As a member of Infineon's 60 GHz family, the wavelength λ of this radar is

$$\lambda = \frac{c}{f} = 5\,\text{mm}. \tag{2.1}$$

According to the information in Table 2.1, the bandwidth range is selected from 59 GHz to 63 GHz, that is, the bandwidth is 4 GHz. Since the collected range-Doppler map is mainly based on the indoor environment, the maximum velocity requirements are not that high, and it can be within ±5 m/s. Therefore, according to the relationship between Chirp Repetition Interval (CRI) T_p and maximum velocity v_{max}, the required CRI can be obtained as

$$T_{\text{p}} = \frac{\lambda}{4 \cdot v_{\text{max}}} = 2.5 \times 10^{-4}\,\text{s}. \tag{2.2}$$

Considering the requirements of velocity resolution, data processing capability, and power consumption, the chirp repetition interval is determined as T_p=220 µs. Conversely, the detectable velocity is

$$v_{\text{max}} = \frac{\lambda}{4 \cdot T_{\text{p}}} = 5.68\,\text{m/s}. \tag{2.3}$$

T_p is usually a little larger than the duration of the chirp T_c to make sure that there will a gap between chirps. This ratio is usually set as 1.1, that is, T_c=200 µs.

If the detection range is too large, the radar power and resolution will be more demanding, and the noise will also increase. Assuming that the maximum range is set at roughly 10 m, the relationship between the number of samples N_s and the maximum range r_{max} can be obtained

Table 2.2.: Initial configs of the radar device, where the resolutions are calculated in the section 2.2.

Parametrics	**Value**	**Parametrics**	**Value**
Chirp repetition time T_p	220 µs	Frequency f	60 GHz
#Chirps N_c	64	#Samples N_s	512
Start frequency f_c	59 GHz	End frequency f_{max}	63 GHz
Sample rate F_s	2.56 MHz	Gain G	33 dB
Receiver index	1 or 3	Transmitter index	1
Maximum range r_{max}	9.6 m	Maximum speed v_{max}	5.68 m/s
Range resolution Δr	0.0375 m	Velocity resolution Δv	0.178 m/s

as

$$N_{\mathrm{s}} = \frac{2B \cdot r_{\max}}{c} = \frac{2 \cdot 4\,\mathrm{GHz} \cdot 10\,\mathrm{m}}{3 \times 10^{8}\,\mathrm{m/s}} = 266.67. \tag{2.4}$$

Since the FFT operation is most efficient while processing data is the power of 2, the number of chirp $N_s = 256$. Due to the real-valued data collected by the radar, the negative frequencies will be discarded by the Real-valued Fast Fourier Transform (rFFT) processing, the required sampling number should be doubled to $N_s = 512$. Therefore, the maximum range is specified as

$$r_{\max} = \frac{c \cdot N_{\mathrm{s}}}{2B} = 9.6\,\mathrm{m}. \tag{2.5}$$

According to the number of samples and chirp time, the sampling frequency F_s can be deduced as

$$F_{\mathrm{s}} = \frac{N_{\mathrm{s}}}{T_{\mathrm{c}}} = \frac{512}{200 \times 10^{-6}\,\mathrm{s}} = 2.56\,\mathrm{MHz}. \tag{2.6}$$

The number of chirps is limited by the velocity and range resolution as well as the maximum velocity. If the required velocity resolution Δv and the range resolution Δr are set to be no less than 0.2 m/s and 0.2 m respectively, the number of chirps N_c will be limited by two values, namely between

$$N_{\mathrm{c}} \geq \frac{c}{2f_{\mathrm{c}}} \cdot \frac{1}{T_{\mathrm{p}} \cdot \Delta v} = \frac{3 \times 10^{8}\,\mathrm{m/s}}{2 \cdot 60\,\mathrm{GHz} \cdot 220\,\mu\mathrm{s} \cdot 0.2\,\mathrm{m/s}} = 56.82, \tag{2.7}$$

and

$$N_{\mathrm{c}} \leq \frac{\Delta r}{2 \cdot v_{\max} \cdot T_{\mathrm{p}}} = \frac{0.2\,\mathrm{m}}{2 \cdot 5.68\,\mathrm{m/s} \cdot 220\,\mu\mathrm{s}} = 80.03. \tag{2.8}$$

Similarly as the reason for N_s, the number of chirps will be determined as $N_c = 64$.
In summary, the initial parameters of the radar can be set according to Table 2.2. In the process of data collection, in order to mitigate the correlation between the scene and the surrounding objects, we increased the time interval. Specifically, the data collected once in the above initial setting is called a frame, the interval between frames will be a little bit increased, namely 0.1 second, and each group will include 8 frames. This setting can ensure the motion coherence of the object related to the radar while the similarity of the data can be mitigated.
To further reduce the similarity between groups, the time interval between groups is set as 1 second, and 5 groups are collected each time. The above process is considered to be

Table 2.3.: Recording settings

Settings	Value
Repetition interval between frames	0.1 s
#Frames	8
Interval between groups	1 s
#Groups	5
Interval between the measurements	1 s

Table 2.4.: Environment conditions

Type		Environment	Tempo
Dynamic	Indoor	Corridors of building V47	Medium tempo
			Slow tempo

called as the single measurement or called once measurement. The interval between each measurement is also set as 1 second. During the time interval between two measurements, the position and angle of the radar will change significantly. The direction of the radar is not limited to the ceiling, but also faces all surroundings, except the floor. This recording setting is summarized in the Table 2.3.

Environment

During the data collection process, the data was mainly recorded in the indoor environment, and the location was in the corridor of the V47 building at the University of Stuttgart. Compared with dynamic data, static data does not provide more information, but loses the information of the Doppler effect, which will not improve the model much. Therefore, only dynamic data exists in the data collection process. Data is collected by walking between the five floors from -1 to 4, and other students and scholars will also pass through the corridor at the same time, which will enrich the data content to a certain extent.
The equipments used in the recording are mainly the laptop and the radar. The dataset which is recorded in the first time, i.e. `dataset_1`, the laptop was moved by placing it on a chair with sliding wheels, while in subsequent data measurements, the laptop was directly carried in a backpack. Hence, the data collection process includes two tempos, called medium tempo and slow tempo. The medium tempo means the normal walking velocity of an adult, and the slow tempo includes the data measured when the walking speed is slowed down or the chair is pushed. In conclusion, the environmental conditions of the recording are summarized in Table 2.4.

Dataset structure

In the server, the data is stored in the path `/data/public/rd_sr/`. There are currently three folders, which also represent three types of data. As the earliest data, `dataset_sample` has most of its data in a static state, and the number of chirps and samples are relatively lower, which is not enough for resampling, so it is abolished. The above parameters in Table 2.2 are firstly used in `dataset_1`, but the data is still divided into dynamic and static, as well as the environments in different rooms. Subsequently, the sampling parameters and environment set in the above tables are used in `dataset_2`. On this basis, the dynamic and in corridor

collected part of dataset_1 is taken into the low tempo case of the dataset_2. Then, the following entire subsequent data processing and training parts will only use the data in dataset_2.
In dataset_2, the following structure is the subfolders of the environment and tempo, the next are the measurements, five groups and eight frames of data in each group. The naming convention for each measurement is, in order, the location (such as corridor), mode (such as moving radar), year, month, day, hour, minute and second, connecting with the underlines. Following illustrates the view of the data storage structure.

```
/data/public/rd_sr/
├── dataset_sample/
│   └── .../
├── dataset_1/
│   └── .../
└── dataset_2/
    └── corridor/
        ├── medium_tempo/
        │   └── .../
        └── slow_tempo/
            ├── Corridor_movingRadar_2024_12_17_16_50_3/
            ├── .../
            ├── .../
            └── Corridor_movingRadar_2024_12_17_16_52_4/
                ├── 0/
                │   └── ...
                ├── .../
                └── 5/
                    ├── 0.pickle
                    ├── 1.pickle
                    ├── ...
                    └── 8.pickle
```

Size

In dataset_2, the medium tempo includes 1431 measurements, and the low tempo includes 931 measurements. The size of dataset_2 reaches 16.4 GB totally.
In the way of splitting dataset, since the scene only occurs in the corridor, the training set, validation set and test set will be divided according to the tempo. Assuming that the training set and validation set contain both medium tempo and low tempo, and the test set only contains medium tempo, the medium tempo part will be divided into 80% for the training process and 20% for the test set, and of the 80%, 80% will be divided into the training set and 20% into the validation set. 80% of the low tempo part will be divided into the training set and 20% into the validation set. According to the above mentioned division method, the training set can contain 11.9 GB data, the size of validation set will be roughly 3 GB, while the size of test set is 1.5 GB.

Table 2.5.: Advantages of the TFRecord files about loading time and storage space, where the TFRecord files of resampling with the factor 2 or 4 contain both low-resolution and high-resolution data while in the original data folders only the high-resolution data is stored. Note that the durations is still depending on the usage of server resources.

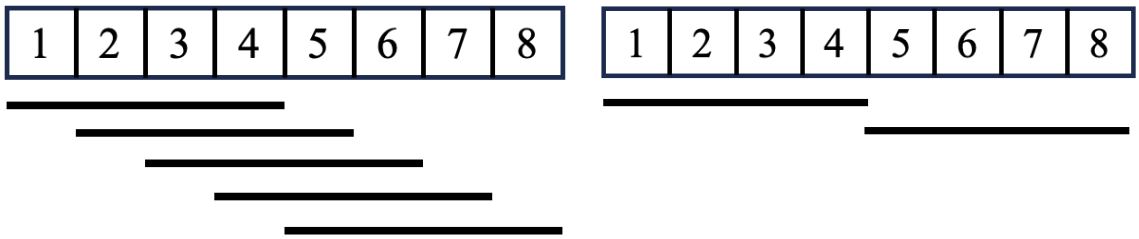
Duaration of the dataset loading			
Loading the subfolders		~3 hours	
Loading the TFRecord files		~5 minutes	
Storage of the dataset			
Type	High resolution data	Factor of 2	Factor of 4
Train set	11.9 GB	5.6 GB	3.5 GB
Validation set	3 GB	1.4 GB	888 MB
Test set	1.5 GB	717 MB	448 MB

2.2. Dataset loading

For dataset loading, we will read and store it in TFRecord files to effectively speed up the training process, and directly reload the TFRecord files in the subsequent training process. During creating TFRecord files, the number of frames and resampling operation will be done, and the conversion between the frequency domain and time domain will be performed in the subsequent reloading process.

TFRecord files

TFRecord is a file format based on the TensorFlow framework that stores data in binary format. This data format can bring many benefits, one of which is that it helps to speed up the training efficiency and reduce the waiting time for data loading of each training process. Since the subfolders must be loaded in sequence, the process of loading the subfolders needs to be completed by the CPU, which will increase the requirements for the CPU and CPU memory, as well as the time required, while the TFRecord files reloading process can be operated by the GPU which allows the parallel Input/Output (I/O) operations. According to a rough estimation, it will take nearly 3 hours to complete all data loading, while only costs roughly 5 minutes to reload it from the TFRecord files. However, note that the durations are still depending on the usage of server resources or settings.
Another advantage is that due to its binary format, it can effectively reduce the space of the data storage. According to the estimated size of the ground truth in the training set, validation set, and test set in section 2.1, it can reach 11.9 GB, 3 GB, and 1.5 GB respectively. On this basis, the corresponding low-resolution data will also be saved in the TFRecord files. Table 2.5 shows the space occupied by saving both low-resolution and high-resolution data after downsampling by the factor of two and four. The occupied storage space is much smaller.
There are two processes of the TFRecord files, namely creating and reloading. Firstly, in the process of creating files, the subfolders are read in sequence. In the code pipeline, a combination of dictionaries and lists is used to store data. The first-hierarchy of the dictionary records the environment and low- or high-resolution data. Although the current data only has the environment of indoor corridors, it is prepared for the case of new environments in the future. Each dictionary adds the same number of lists according to the



(a) Shift step as one (b) Stride same as the number of frames

Figure 2.2.: Sliding window while the number of the frames is 4, where the number 1-8 represent the frames in each group and the lines illustrate the selected frames within each window.

number of tempo types, which corresponds to the structure in Table 2.4, so as to facilitate the subsequent split according to the required types in different dataset. To clarify it, according to the current corridor environment, two dictionaries will be generated, called `corridor_GT` and `corridor_downsampled`, and then each dictionary will contain two lists, called `medium_tempo` and `slow_tempo`.
For any list in `corridor_GT`, its size will be `(#Sequences, 1, 64, 512)` while `corridor_downsampled` with factor of two has the size of `(#Sequences, #Frames, 32, 256)`. Therefore, in the current pipeline, two parameters in the TFRecord file will be determined, one is the number of frames and the other is the resampling rate. The significance of the number of frames is that due to the continuous motion of the radar, such as feeding 4 consecutive range-Doppler maps for the model, the model learns the temporal correlation between consecutive range-Doppler maps and creates more consistent outputs, while the input with single range-Doppler map doesn't have this information. Based on this goal, a sliding window approach will be introduced in loading for data augmentation which will be explained following. In the TFRecord files, except the low-resolution and high-resolution data being stored in byte format, the corresponding sizes of the two dictionaries are stored as int value for subsequent reloading. Additionally, the data cube will be operated with the de-biasing to remove the offset of each chirp.
The other part is to prepare the data from the TFRecord files, and after restoring the data according to the size stored in the TFRecord files, some additional processing will be performed. First of all, Range-Doppler Processing (RDP) will be used to convert the real-valued cube in the time domain into the complex-valued data in the frequency domain, and the minimum, mean as well as maximum values of the each dataset will be calculated for normalization, which will be shown in section 2.3.3, then they will be split into mini-batches, additionally the training set will be shuffled.

Sliding window for the number of frames

As mentioned above, due to the continuous movement of the radar, the changes in the range and relative velocity of the objects can have some relationships. When the data also contains previous information, the upsampling results of the model could be better. This part of the evaluation will be shown in the section 6.6. Two settings about the number of frames can be evaluated, namely a single frame without previous information and four frames. The quantity and quality of data will have a great impact on the training process of the model. In order to augment the data, the sliding window will be used, as shown in Figure 2.2.

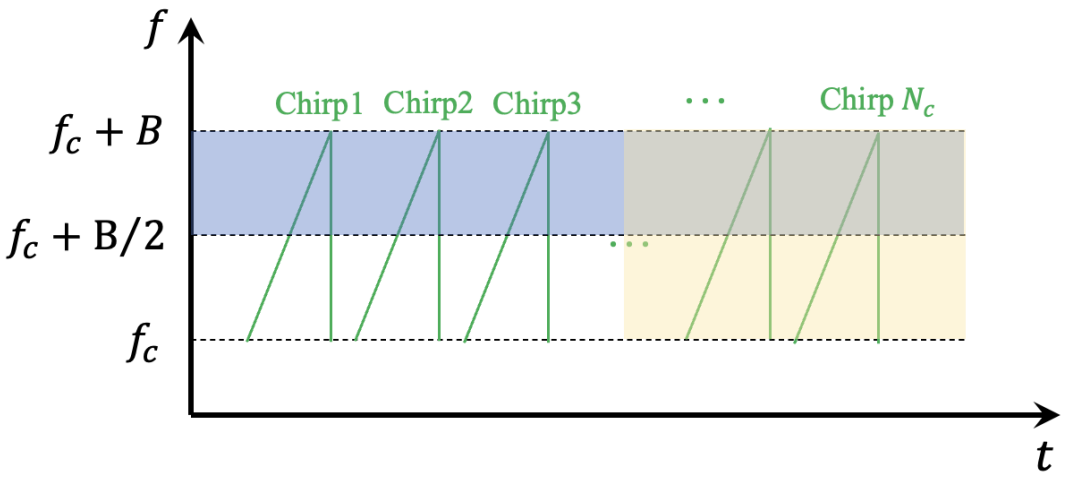


Figure 2.3.: Downsampling processing with the factor of 2 by cutting the bandwidth with only half samples within the chirp to obtain half range resolution and dropping half of the chirps to halve the velocity resolution.

According to the measurement, group and frame defined in Table 2.3, the sliding window will only be applied to the frames, that is, sliding within each group. That is to say, the integers from 1 to 8 in Figure 2.2 above represent the pickle files stored in eight frames, namely `1.pickle` to `8.pickle`. The reason for applying it only within the group is that in the above recording setting, in order to reduce the similarity of the scenery and data, the time interval between groups and collections is larger, so only the data within each group can be regarded as continuous. The lines at the bottom of the figures represent the frames covered by the window. The four frames covered will be used as one sequence input of the model, and the data to be upsampled is the last frame within each window, and the ground truth is only the high-resolution version of this last frame.
In addition, there are two ways as shown in Figure 2.2. The window in Figure 2.2a moves once a step, while the window in Figure 2.2b moves according to the number of frames. Compared with the approach that only loads the certain required number of frames once within a group, the method in Figure 2.2b will obtain twice the amount of data, while Figure 2.2a can obtain five times the amount. However, the shift method in the Figure 2.2a will cause the overlapping part, that is, the data to be upsampled in one input also appears in another input, which blows up the scale of the dataset but doesn't bring more information. Therefore, in order to ensure the quality of the data, subsequent data processing will only use the window sliding in the way as in Figure 2.2b.

Resampling

During creating the TFRecord files, high-resolution and low-resolution data will be stored in pairs. At this moment, the data is still in the time domain. Figure 2.3 draws a signal containing multiple chirps and the process of the downsampling with the factor of two.
Based on the radar settings selected above, the velocity and range resolutions can be calculated separately. Substituting the initial radar parameters from into the velocity resolution

formula below, the value is

$$\Delta v = \frac{c}{2f_{\mathrm{c}}} \cdot \frac{1}{T_{\mathrm{p}} \cdot N_{\mathrm{c}}} = 0.178\,\mathrm{m/s}, \tag{2.9}$$

while the range resolution can be derived as

$$\Delta r = \frac{c}{2B} = 0.0375\,\mathrm{m}. \tag{2.10}$$

First of all, regarding the resampling of velocity resolution, according to the derivation of formula 2.9, if in order to double the velocity resolution while the speed of light and carrier frequency are constant, the CRI T_p or the number of chirps N_c have to be reduced to half of the original parameters. Since the CRI have the influence on the shape of chirp and it's hard to build the pairs with different CRI, the later alternative is chosen, namely the number of chirps N_c turns to 32, corresponding to the abolishment of the signals under the yellow mask in Figure 2.3, and only keep the first 32 chirps. In terms of the range resolution, according to formula 2.10, since the speed of light is a constant, reducing the bandwidth increases the range resolution. Therefore, the blue mask in Figure 2.3 indicates that the upper half of the chirp has been removed, thereby halving the range resolution.
For the maximum values of both velocity and range, according to formulas 2.3 and 2.5, maintaining the wavelength λ and CRI T_p can ensure that the maximum velocity is the same as the high-resolution case. When the bandwidth is reduced by half, the number of sampling points within each chirp is also reduced by half, as plugging the formula 2.6 into the formula 2.5, the maximum value of the range is up to the slope only, hence, it remains.
For each high-resolution frame within the sliding window, known the shape of each high resolution data is `(64, 512)`, the pseudo-code of this downsampling process is written in Algorithm 1.

Algorithm 1 Pseudo code of resampling

Input: *High − resolution data as data, sampling rate*
Output: *Low − resolution data*

```
row = data.shape[0]
col = data.shape[1]
low_resolution_data = data[: row // sampling_rate, : col // sampling_rate]
```

Cube de-biasing

During creating the TFRecord files, another operation is cube de-biasing. Cube de-biasing refers to the mean removal of the data cube. The main purpose of this operation is to eliminate the constant offset caused by the hardware system or environment, so as to more clearly and fairly detect the reflected signal of the target, while ensuring no different offset between the chirps in the collection process.
The operation of cube de-biasing is to calculate the mean of each chirp, and then expand it to the size of N_s to remove the offset along the fast time axis. The pseudo code is as following Algorithm 2.

RDP

Algorithm 2 Pseudo code of cube de-biasing

Input: *Cube data in time domain*
Output: *Debiasing data*
1: $avgs = np.average(cube, 1)[:, None]$
2: $cube = cube - avgs$

After loading the TFRecord files, one processing to do is FFT, which is used to convert the data from time domain to frequency domain. This is a very important process in radar signal processing. Decomposing the data in the time domain into different frequency components helps to provide more useful information.
But in FFT, the frequency can be positive or negative, since the sine and cosine signals in the complex component can be in both directions. However, in our radar data, the signal in time domain is real-valued, which is different from the complex-valued signal as input. The real-valued input signal has a Hermitian symmetric spectrum. The reason is that in the definition of FFT, for a discrete signal $x(n)$ with the length of N, its Discrete Fourier Transform (DFT) conversion is as following

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi kn}{N}}. \tag{2.11}$$

Its conjugate is

$$X^*[k] = \sum_{n=0}^{N-1} x[n] e^{+j\frac{2\pi kn}{N}}, \tag{2.12}$$

then they both can be derived as conjugated pair, namely

$$X[N-k] = X^*[k]. \tag{2.13}$$

Therefore, there is no additional information between the positive and negative parts, in order to reduce the computational complexity and storage requirements, we can only save the information of the positive frequency part as well as one real value and discard the negative frequency part. Figure 2.4 shows well the finally obtained symmetrical data after the FFT process.
The data after rFFT processing contains two parts. One part is the Direct Current (DC) component, which can be regarded as the offset of the data and the other part is the Nyquist frequency component. In our data, when the dimension of a low resolution data obtained by double downsampling in time domain is `(256, 32)`, the dimension of the data in frequency domain will be in the shape of `(129, 32)`, while the dimension of the low-resolution data obtained by quadruple downsampling is `(65, 16)`. Similarly, the shape of the high-resolution cube data will changed from `(512, 64)` to `(257, 64)`.
For the specific transform process, since FFT can only process time domain data of limited length, it is necessary to truncate the signal, that is, add a window, and each transform process is only performed within one window. The commonly used window functions include but not limit to the Hamming windows, Hanning windows, and blackman windows. For the two window functions built in TensorFlow, Hamming windows and Hanning windows, since the end of the Hamming window has a smaller main lobe in frequency domain which leads to a shaper range-Doppler map [23]. Therefore, before performing rFFT, using the Hamming

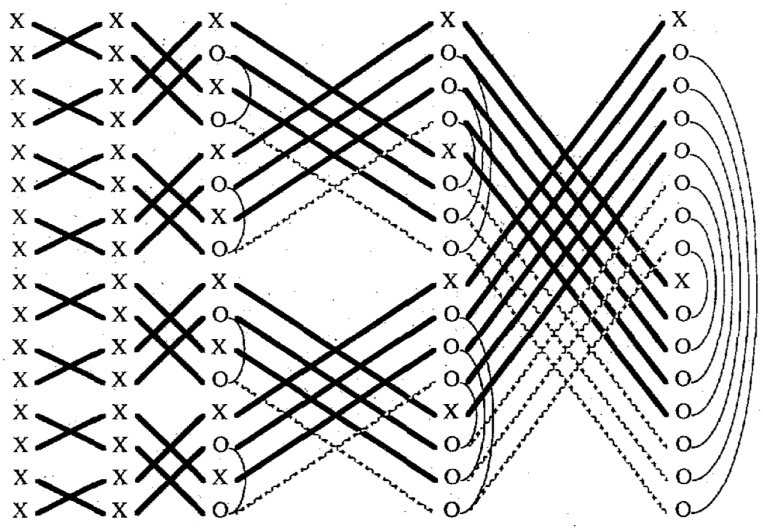


Figure 2.4.: An overview of the FFT in the case of real-valued inputs, where X indicates the real value and O represents the complex value. The figure shows the symmetric converted complex values [22].

window function on the data cube during the RDP is necessary.
After obtaining the windows of the range and Doppler effect axes respectively, we first apply the window on the range dimension and perform the rFFT operation. The dimension of the range will then change from 512 to 257 or from 256 to 129. The data after the rFFT operation on the range dimension is complex-valued. While applying the window on the velocity axis, it should be applied to the real and imaginary parts respectively. Since it is already complex-valued, only FFT is performed on the velocity dimension instead of rFFT. In addition, the FFT operation in TensorFlow only acts on the last dimension, so after finishing the transform on the range dimension, a transpose operation is required. Finally, since the range is between 0 and the maximum value, whereas the velocity can be in both positive and negative values, the data can be shifted so that the position where the velocity is zero centered, which is clear and convenient for the visualization. The pseudo code for this processing is shown in Algorithm 3.

Inverse Range-Doppler Processing (iRDP)

When the data needs to be converted from the frequency domain back to the time domain, the iRDP operation can be performed. The operation is the opposite of the above mentioned RDP operation. First of all, make sure that the shape of each range-Doppler map is as the format of `(1, #samples, #chirps)`. The reason for the extra dimension at the first position is to emphasize the number of frames. The number of frames may be 4 as input, but in the output it goes to one frame. Next, use Inverse Fast Fourier Transform (iFFT) to process the velocity axis and calculate the window functions of the range dimension as well as the velocity dimension. Furthermore, since the range dimension has not yet recovered its original size, the dimension must be calculated additionally in the range window function. Then, after removing the window function in the real and imaginary parts and transposing the shape, Inverse Real-valued Fast Fourier Transform (irFFT) and removing the range window function can be performed. The pseudo code is written in Algorithm 4 as well.

Algorithm 3 Pseudo code of RDP processing

Input: *Cube data in time domain as cube*
Output: *Range − Doppler map in frequency domain as rD*

```
range_window = tf.signal.hamming_window(tf.shape(cube)[2])
doppler_window = tf.signal.hamming_window(tf.shape(cube)[1])

cube_windowed = cube * range_window
rfft = tf.signal.rfft(cube_windowed)
rfft = tf.transpose(rfft, perm = [0, 2, 1])

re_rfft = tf.math.real(rfft) * doppler_window
im_rfft = tf.math.imag(rfft) * doppler_window
rfft = tf.complex(re_rfft, im_rfft)
rD = tf.signal.fft(rfft)

if whether_fftshift then
    rD = tf.signal.fftshift(rD, 2)
end if
```

Algorithm 4 Pseudo code of iRDP processing

Input: *Range − Doppler map in frequency domain as rD*
Output: *Cube data in time domain as cube*

```
num_chirps = tf.shape(rD)[2]
num_samples = tf.shape(rD)[1]
rD = tf.reshape(rD, (−1, num_samples, num_chirps))

if whether_fftshift then
    rD = tf.signal.fftshift(rD, 2)
end if

rfft = tf.signal.ifft(rD)
range_window = tf.signal.hamming_window((num_samples − 1) * 2)
doppler_window = tf.signal.hamming_window(num_chirps)

re_rfft = tf.math.real(rfft) / doppler_window
im_rfft = tf.math.imag(rfft) / doppler_window
rfft = tf.complex(re_rfft, im_rfft)

rfft = tf.transpose(rfft, perm = [0, 2, 1])
cube_windowed = tf.signal.rfft(rfft)
cube = cube_windowed / range_window
```

2.3. Pre- and post-processing of the model inputs and outputs

According to the type and characteristics of the data, a more appropriate preprocessing will greatly promote the training of the model. Furthermore, the convolutional layer is common in the models. Currently, after processing by RDP, the data is converted into complex value, whereas the convolutional layer can only be used to process the data in real value, so it needs to choose some preprocessing methods as well. In the pipeline, the following three processing steps are mainly used: the conversion from complex value to real value, that is, input data representation, the logarithm operation on the amplitude of the data, and the normalization of the data. The three processing steps contain different methods which can be combined in different ways.

2.3.1. Input data representation

In order to convert the range-Doppler map from complex value to real value, three different input data representations are introduced as real and imaginary representation, amplitude representation, as well as amplitude and phase representation. In addition, a very important preprocessing is about the dimensions. During the process of creating and loading TFRecord files, mini batch dimension is introduced. Therefore, the current shape of the low-resolution data is `(batch, #Frames, #samples//sampling_rate//2+1, #chirps//sampling_rate)`. According to the default setting of the convolutional layer in TensorFlow, the feature is always put in the last dimension, so the current data should be transposed, that is, transposed into `(batch, #samples//sampling_rate//2+1, #chirps//sampling_rate, #Frames)`.

Real and imaginary representation

In this representation type, the real and imaginary parts of the input are taken out separately and concatenated. According to the above input size, the last dimension will be converted from `#Frames` to `#Frames*2`, that is, `(batch, #samples//sampling_rate//2+1, #chirps//sampling_rate, #Frames*2)`. There are two options for the concating process. One option is to take features as the number of frames of the real part and features of the imaginary part separately and stack them directly on the last dimension, that is, `rrrriiii`, where `r` represents the real part and `i` indicates the imaginart part. Another is that separating each data and then stacking them, that is, `ririririi`. After a quick experiment, we used a small model and part of the data and found that there's no big difference between these two, since the convolution is invariant to the permutation with respect to the channels. Therefore, in order to focus on more important improvements later and reduce the complexity of different combinations, the first option is used by default, that is, directly stacking the real and imaginary parts.
In this representation type, the number of kernels in the last convolutional layer in the models will be set as two, where the first dimension represents real part and the second dimension indicates imaginary part, so the final output of the model will be `(batch, #samples//2+1, #chirps, 2)`. While calculating the loss, this prediction is still used in this shape. Only for the visualization, the value will be considered to convert back. TensorFlow also provides a function for directly merging real and imaginary parts into a complex value.

Amplitude representation

According to the definition of range-Doppler map, there is an amplitude at the position corresponding to the range and velocity, which is a required parameter in the visualization and range-Doppler map upsampling. Therefore, a real value can be obtained by calculating the amplitude of each complex value, and the size of the input will be hence `(batch, #samples//sampling_rate//2+1, #chirps//sampling_rate, #Frames)`.
In this case, the number of kernels in the last convolutional layer in all models will be set as 1, which represents the amplitude and the output shape would be `(batch, #samples//2+1, #chirps, 1)`. In the loss functions, only the difference between the amplitude can be calculated without any phase information. Meanwhile, this value is directly used while plotting the range-Doppler map, and the complex value or the signal in time domain cannot be restored.

Amplitude and phase representation

In order to retain as much information as possible, to facilitate the recovery of time domain signals and to add phase in the loss functions to supplement information, the amplitude and phase can also be concatenated. Its size will be `(batch, #samples//sampling_rate//2+1, #chirps//sampling_rate, #Frames*2)`, and the last dimension will be concated as `aaaapppp`, where `a` represents amplitude and `p` represents phase. The output unit of the last convolutional layer in the models is 2, where the first one represents amplitude and the second dimension denotes phase.

2.3.2. Logarithm operation

Compared to the upsampling image or signal approaches in the state-of-art papers mentioned in section 1.4, a significant difference is that the amplitude of our range-Doppler map in the indoor environment varies widely. As the detection range increases, the noise becomes greater and the SNR decreases. We hope that the model can also pay attention to areas with low signal power, so we need to reduce the dynamic range of the signal. One way is to perform logarithm operations on the data. For this part, there are currently two ways: no logarithm processing and logarithm processing with base 10.

2.3.3. Normalization types

In order to further reduce the dynamic range of the signal, the data can also be normalized. In addition to the amplitude, phase can also be normalized.
For amplitude normalization, there are two normalization ranges, namely (-1, 1) or (0, 1). First of all, we need to find the minimum, mean, and maximum values of the dataset. This part of the operation is performed when the TFRecord files are reloaded but not yet converted into the mini batches. By mapping a loop in the dataset, the corresponding values can be iteratively found. The value searching and normalization operation will be performed on the training set, validation set, and test set separately, not the entire dataset. In addition, since in reality, the model can only get the low-resolution data as the input, while the high-resolution data is unknown to the model. Therefore, while calculating these values, it is based on the low-resolution data rather than the high-resolution data. Another reason is that although the data has been downsampled, the overall value range, the minimum, mean as well as maximum values and data distribution will not change much. It can be considered

Table 2.6.: Combination of three different processing types, where ✘ represents the valid combination and - denotes no combination.

Combination	Real/imaginary	Amplitude	Amplitude/phase
Logarithm operation	-	✘	✘
Amplitude normalization	✘	✘	✘
Angle normalization	-	-	✘

that the data range and distribution at low-resolution case are basically consistent with the high-resolution data.
If the range is going to be (-1,1), the normalization operation can be done by

$$\text{normalized data} = \frac{\text{data}}{\text{max}}, \tag{2.14}$$

where max means the maximum absolute value among the dataset and in case of the range (0,1), the equation should be written as

$$\text{normalized data} = \frac{\text{data - min}}{\text{max - min}}. \tag{2.15}$$

In the case of real and imaginary representation, the normalization can be performed as

$$\text{normalized data} = \frac{\text{data}}{\text{amplitude}}. \tag{2.16}$$

Furthermore, since the phase is always in the range of ($-\pi$, π), it can be divided by π to make its range fall into (-1, 1). In conclusion, the dataset can be preprocessing in these three types, and their combination can be summarized as Table 2.6.

2.4. Range-Doppler map visualization

In order to intuitively see the effect of upsampling, we can plot the range-Doppler map, which mainly includes three steps: restoring the processed part, calculating the required parameters, and drawing.
First of all, some postprocessing should be performed to reverse preprocessing, and then the parameters that need to be calculated include the amplitude of each point, range resolution, and velocity resolution. The formulas for calculating velocity and range resolution can be applied in the pipeline according to formulas 2.9 and 2.10, and for the amplitude, it needs to be determined according to the input data representation. When the data is divided into real and imaginary parts, the square root of the sum of the squares should be calculated, and if it is separated by amplitude or amplitude and phase, the first value in the last dimension will be taken. In the visualization, the first range-Doppler map in the last batch of each epoch is used by default, that is, each epoch has one visualization plot. There are something also necessary to emphasize that while plotting low-resolution range-Doppler maps, the calculated resolution values must be changed according to the resampling rate. Meanwhile, since the data collected by radar device is in voltage unit, whereas the unit of range-Doppler map is in power, the above mentioned amplitude must be squared.

There are two additional settings in the range-Doppler visualization function. One is that the amplitude value can be drawn in linear scale or in dB unit. Since the data is postprocessed, the input of the visualization function is already in linear scale, but due to the large dynamic range in the data, in order to facilitate the observation of the low SNR part, the log scale is used by default, namely in dB unit. Another setting is that since the color bar in the range-Doppler map will produce different shades of colors as the amplitude in the range-Doppler map changes, in order to compare results clearly as well as according to the distribution of high-resolution data, the range in the color bar is set between -80 dB and 60 dB. When data exceeding this range exists, a warning will be displayed in logging. In the Weights and Biases (WandB) log, the corresponding low-resolution, predicted super-resolution, and high-resolution range-Doppler maps will be visualized together in this way.

3. Models

In this chapter, we will build the models for range-Doppler map upsampling. According to the introduction in section 1.4, there are mainly the following deep learning models, encoder-decoder architecture such as UNet architecture, Transformer model, cGAN, etc. Additionally, two layers are required in models, namely dimension processing layer and upsampling layer.

3.1. Dimension processing layer

According to the RDP introduced in section 2.2 and the preprocessing types in section 2.3, the input size of the model will be `(batch, #samples//sampling_rate//2+1, #chirps//sampling_rate, #Frames or #Frames*2)`. In order to explain the change of the shape easily and clearly, following will use the amplitude and phase representation type, batch size as 8, sampling rate as 2, the number of frames as 4 by default. In this combination, the shape of the low-resolution input will be `(8, 129, 32, 8)` and for high-resolution data as ground truth as `(8, 257, 64, 2)`. However, there is a problem with the current range dimension as an prime number. In the encoder-decoder architecture, it is difficult to choose a suitable stride for the current shape of the range axis because multiple times of the downsampling and upsampling layers will be performed. For the Swin Transformer, the current shape of range axis is also a troublesome size since the data needs to be split into multiple patches. Except the DP-TF Transformer model, since it calculates self-attention along the range and velocity axes separately, the problem of dimension does not need to be considered as well as our basic CNN model, because it will not downsample the data.
There are three solutions for dimension processing layer. The first is that only in the case of DP-TF Transformer model and the basic CNN model, it's possible to skip the additional dimension processing layer, and the model directly learns based on the input shape, but also ensure that there is no other downsampling process in the DP-TF Transformer model. In this case, since the model will perform an upsampling operation at the end, the dimension will become `(8, 258, 64, 2)`, so a cropping is required to select only the first 257 samples along the range axis.
Another option is that for models that need multiple downsampling layers such as in the encoder-decoder structure, it is more convenient when the range-Doppler map shape is in the power of 2, so the solution is to perform a convolutional layer along the range dimension, which uses a stride of 1 and the kernel size is set as `(2, 1)`, so that the input dimension of the model becomes `(8, 128, 32, 8)`. After passing through the model and subsequent upsampling layer, its dimension goes to `(8, 256, 64, 2)`. Therefore, there is always one less value in the range dimension. Then the transposed convolutional layer is needed, and also chosen stride as 1 and kernel size as `(2, 1)` to make its shape the same as ground truth. However, while applying the convolutional layer according to the second option, the reduction in one dimension means that information may be lost. In order to alleviate this problem,

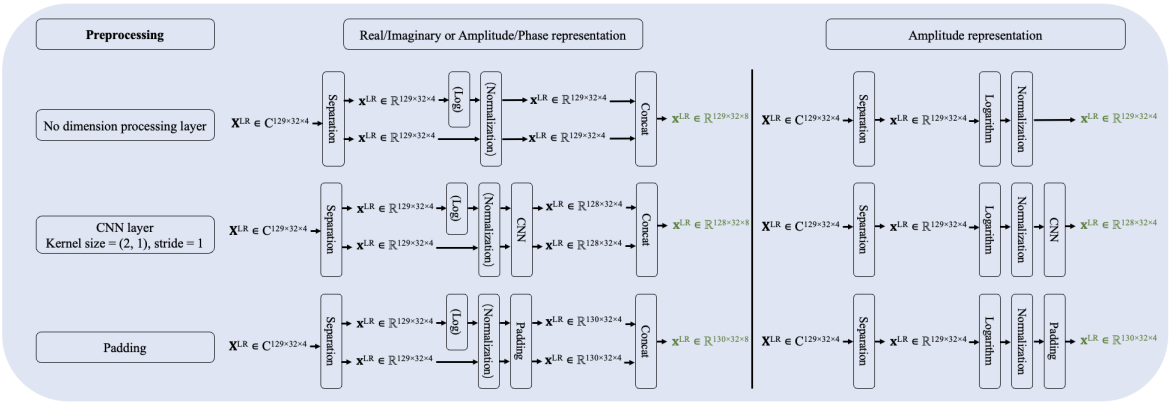


Figure 3.1.: Preprocessing block in the case of the number of frames as 4, where logarithm and normalization are not used in real and imaginary representation type and the green represents the model inputs.

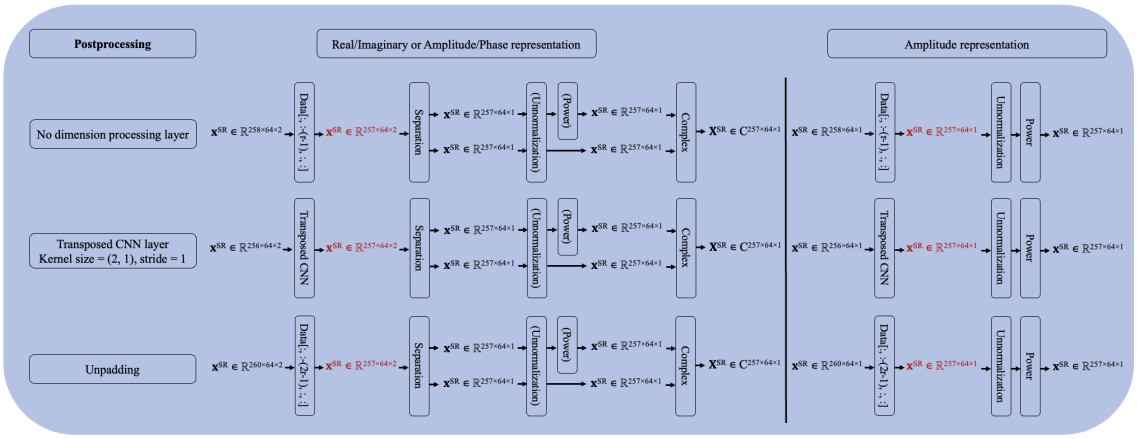


Figure 3.2.: Postprocessing block in the case of the number of frames as 4, where unnormalization and power of 10.0 are not used in real and imaginary representation type, the red represents the model outputs to calculate the training loss and r is the resample rate.

the third solution is padding. By filling a row of 0 values along the range dimension, it will not lose input information and can become an even number, that is, the input becomes `(8, 130, 32, 8)`. Meanwhile, since it is not a power of 2, the downsampling layer can be performed only once in the model at most. In this case, after the final upsampling layer, the size of the output will be `(8, 260, 64, 2)`. There are extra `(2*sampling_rate - 1)` samples in the range dimension, but since it was initially filled with 0, directly discarding this part will not cause information loss, that is, taking the values before the last `(2*sampling_rate - 1)` dimension.

In summary, according to the input data representation, logarithm, normalization type as well as the dimension processing type, the preprocessing block can be combined with six forms as shown in Figure 3.1. The corresponding postprocessing module has six forms as well, illustrated in Figure 3.2. The green part in the preprocessing block is the input data that the model starts to process, while the red part in the postprocessing block is the output of the model for calculating the training loss with the ground truth.

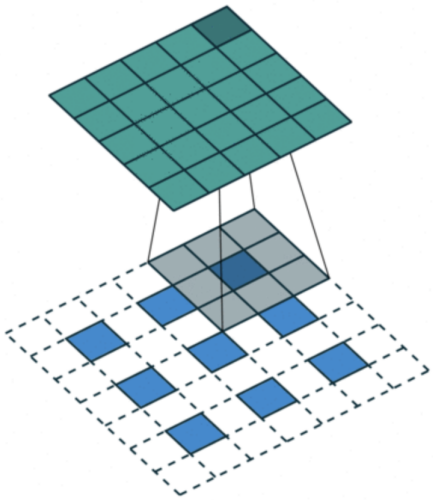


Figure 3.3.: View of the transposed convolutional layer in the case of the stride as 2, padding as 1 and kernel size as 3, where the blue is input and the green is output [24].

3.2. Upsampling layer

Before performing the upsampling layer, it's necessary to determine the times of upsampling layers required. Since the upsampling functions in TensorFlow or PyTorch usually doubles the resolution each time, the upsampling rate prefers to be a power of 2, so the times of upsampling operations in our case can be determined to be logarithmic sampling rate with the base as 2.
A common layer for upsampling is the transposed convolutional layer, whose operation is logically equivalent to the reverse convolutional layer as shown in Figure 3.3, but these two are not their respective inverse, that is, the transposed convolutional layer can not completely restore the range-Doppler map back to the original one before the convolutional layer. The shape relationship between the input and output of the transposed convolutional layer is

$$o = (i - 1) * s + k - 2 * p, \tag{3.1}$$

where s represents stride, k is the kernel size, p denotes the padding size, i and o are the shape of input and output, respectively.
TensorFlow has a built-in transposed convolutional module. To facilitate upsampling, there are two options in the padding parameter, either valid or same. When same is selected, upsampling will be performed according to the stride value. Furthermore, each time the transposed convolutional layer is used, it will pass through the additional layer normalization and Rectified Linear Unit (ReLU) activation layers as the other paper [12] did, which makes the training process more stable and converges faster as well as preventing the gradient explosion.
Another upsampling method is called pixel shuffle approach, also known as sub-pixel convolution or depth to space method, etc., proposed by Shi et al [25]. As shown in Figure 3.4, the principle is that the input data is processed by a series of hidden layers in the model, data of the shape $(\mathtt{N},\ \mathtt{C} \times \mathtt{r}^2,\ \mathtt{H},\ \mathtt{W})$ is obtained, where r denotes the resampling rate and all of the numbers are positive integers. Upsampling is achieved by splitting the size of $\mathtt{r}^2$ in depth and expanding its value into the dimensions of spaces, namely H and W.

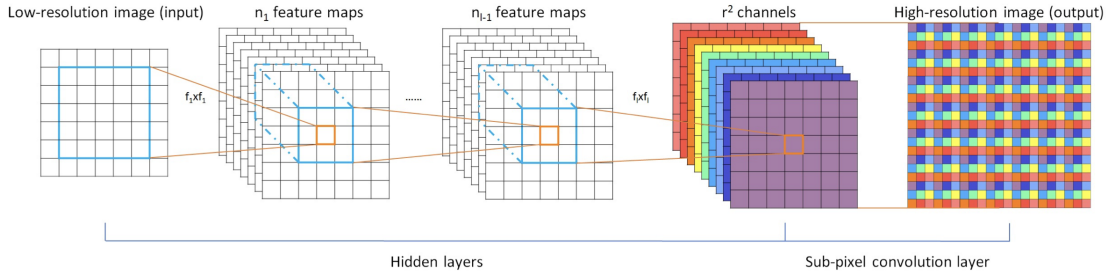


Figure 3.4.: View of the pixel shuffle approach, where r denotes the resample rate [25].

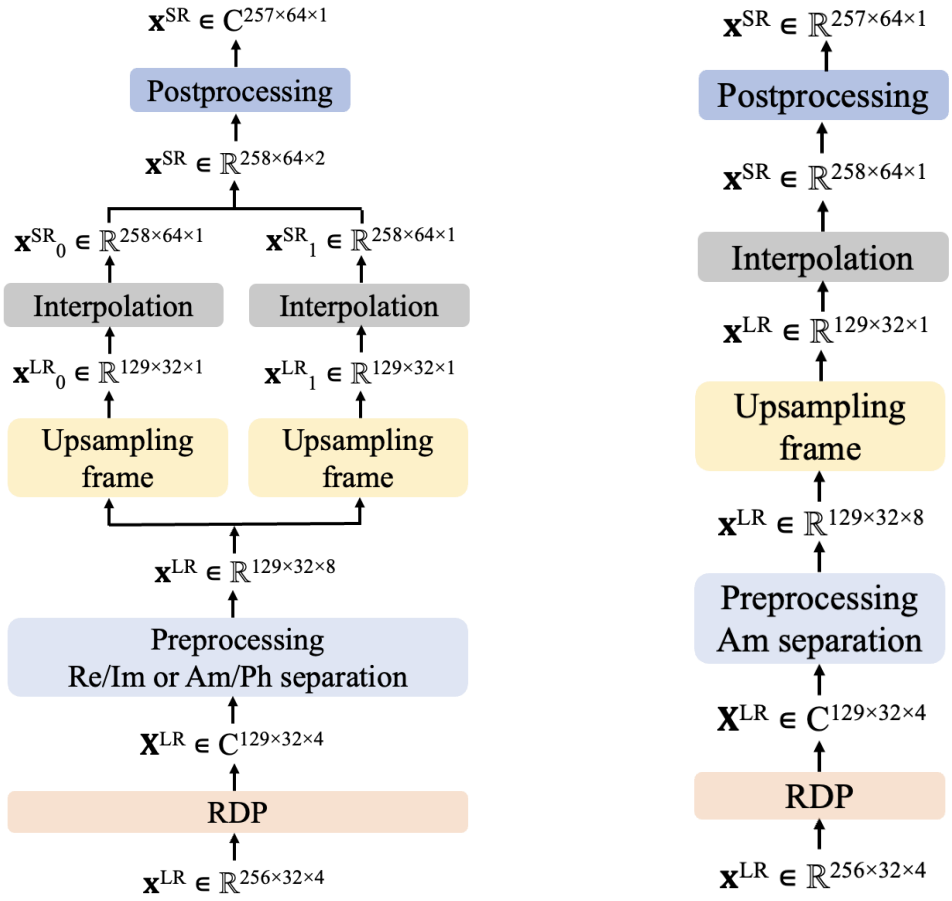
There are two main benefits of pixel shuffle approach. Firstly, the pixel shuffle method does not require any other parameters, which can reduce the number of parameters and computational complexity compared to the transposed convolutional layer [26]. On the other hand, according to the paper from Odena, etc., one reason for the checkerboard effect is that the transposed convolutional layer causes overlapping, while the lack of overlapping in the method of pixel shuffle approach can mitigate this problem to a certain extent [27].

3.3. Interpolation

As a method that does not require any parameters and training process, the interpolation method can be used as a baseline in the range-Doppler upsampling task. Its principle is to interpolate a value in the middle according to the values of the surrounding points. Figure 3.5 shows the interpolation model built in this thesis, which has two forms, Figure 3.5a and Figure 3.5b, according to the input data representation.
Assuming the same parameters as in section 3.1 are used, that is, the batch size is set as 8, the data resampling rate is 2, and the number of frames is 4, the shape of the input is `(8, 129, 32, 8)` in the case of the real and imaginary or amplitude and phase representation types after preprocessing block. Since the interpolation method has no parameters and training process, it will directly process the low-resolution range-Doppler map which is going to be upsampled. According to the order of the number of frames in section 2.2 and the input data representation mentioned in section 2.3, the last dimension of the real part or the amplitude part x_0^{LR} and the last dimension of the imaginary part or the phase part x_1^{LR} are the data that need to be upsampled, so only these two frames are taken out.
For the interpolation process, TensorFlow provides many approaches. By default, bilinear interpolation is used in this model, as illustrated in Figure 3.6, which is calculated based on the distance from the surrounding four points, namely

$$f(x,y) = \frac{1}{(x_2-x_1)(y_2-y_1)} \begin{bmatrix} x_2-x & x-x_1 \end{bmatrix} \begin{bmatrix} f(Q_{11}) & f(Q_{12}) \\ f(Q_{21}) & f(Q_{22}) \end{bmatrix} \begin{bmatrix} y_2-y \\ y-y_1 \end{bmatrix}, \tag{3.2}$$

where $f(x,y)$ is the interpolation value at the point (x,y). Furthermore, the default upsampling rate is 2 each time, so the sampling rate should be a power of 2. The model will be postprocessed according to the value of the resampling rate and as mentioned in section 3.1 to make it the same size as the ground truth after stacking along the last dimension.



(a) Interpolation model with the real and imaginary or amplitude and phase representation types.

(b) Interpolation model with the amplitude representation type.

Figure 3.5.: Interpolation models in terms of the representation types, in the case of the downsampling rate as 2 and the number of frames as 4.

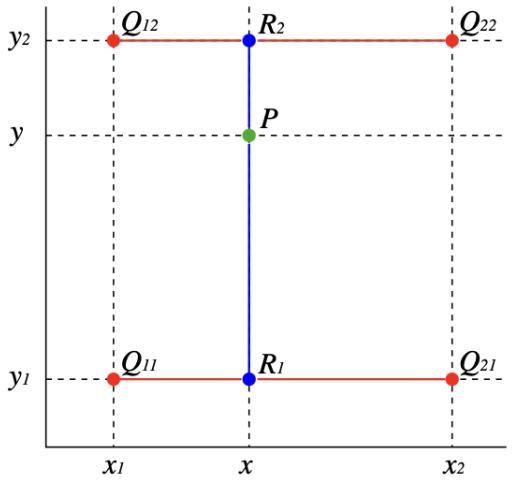


Figure 3.6.: Bilinear interpolation, where P is the interpolated point and Qs are the surrounding points [28].

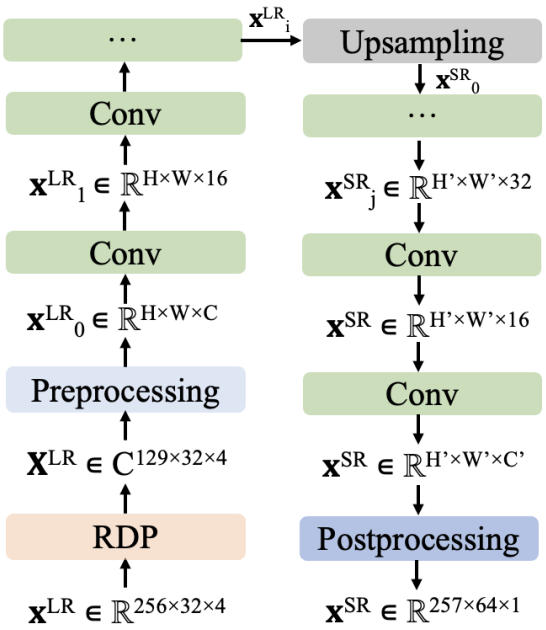


Figure 3.7.: CNN model, where the channel, height and width are depending on the processing methods.

3.4. CNN model

Compared with the bilinear interpolation model, the most common model for image processing is the Convolutional Neural Network (CNN). We introduced a relatively basic CNN model, which does not downsample the data and performs the upsampling operation after one specified convolutional layer. The overview of the CNN model is shown in Figure 3.7. As shown, since the preprocessing is a combination of representation, dimension processing, logarithm and normalization types, the height `H` and width `W` depend on the selected dimension processing type, while the channel `C` and `C'` depend on the input data representation. The value of `C` can be 4 or 8, and then enters the convolutional layers. In this model, two parameters are set to facilitate the change of model scale, namely the neuron list and upsampling layer index. The neuron list represents a list of the channel values in the case of stride as 1, and the upsampling layer index indicates the position of the convolutional layer after which the upsampling layer is performed. Similarly, `H'` and `W'` are the sizes of the range-Doppler map after upsampling, which are also up to the dimension processing type, and are then converted into the same shape as the ground truth while postprocessing. The process can be written as

$$x_i^{LR} = \text{Conv2D}^i(x_0^{LR}), \tag{3.3}$$

where i is the upsampling layer index with the upsampling operation

$$x_0^{SR} = \text{Upsampling}(x_i^{LR}), \tag{3.4}$$

then the final super-resolution data after j times convolutional layers is

$$x^{SR} = \text{Conv2D}(\text{Conv2D}^j(x_0^{SR})). \tag{3.5}$$

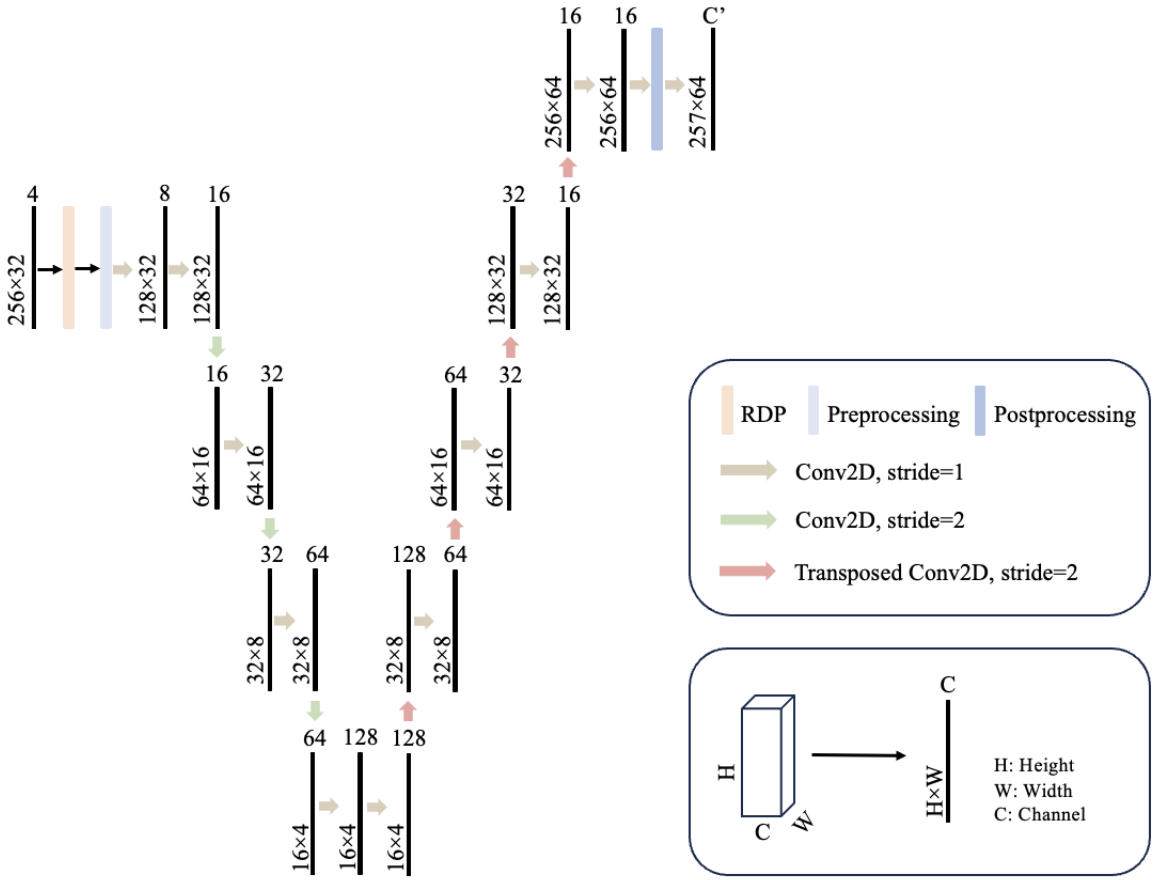


Figure 3.8.: UNet model

3.5. UNet architecture

As mentioned in section 1.4, the encoder-decoder structure is very common for the upsampling task in deep learning. During the encoder process, the model will do the downsampling operations and learn important features from the data, and then perform multiple upsampling layers based on the learned information. The most common model is the UNet model, so we built it based on the convolutional layers with the stride as 2. Compared with the CNN model in section 3.4, the downsampling process may cause information loss. Inspired by the paper [9] from Akarsh et al., by adding residual connections between the encoder and decoder parts, the information loss problem is alleviated. Therefore, in this section, there will be two parts, namely basic UNet model and UNet concat model.

3.5.1. UNet model

The overview of the UNet model is illustrated in Figure 3.8. Since there are multiple downsampling operations in the UNet model and the default downsampling rate is 2, the data size in height and width should be the power of 2. Therefore, the dimension processing layer is determined as the convolutional layer approach in most cases. Otherwise, only once downsampling can be done if using padding. Meanwhile, due to the representation type, the channel would be either 4 or 8, and while passing through the first convolutional layer, this

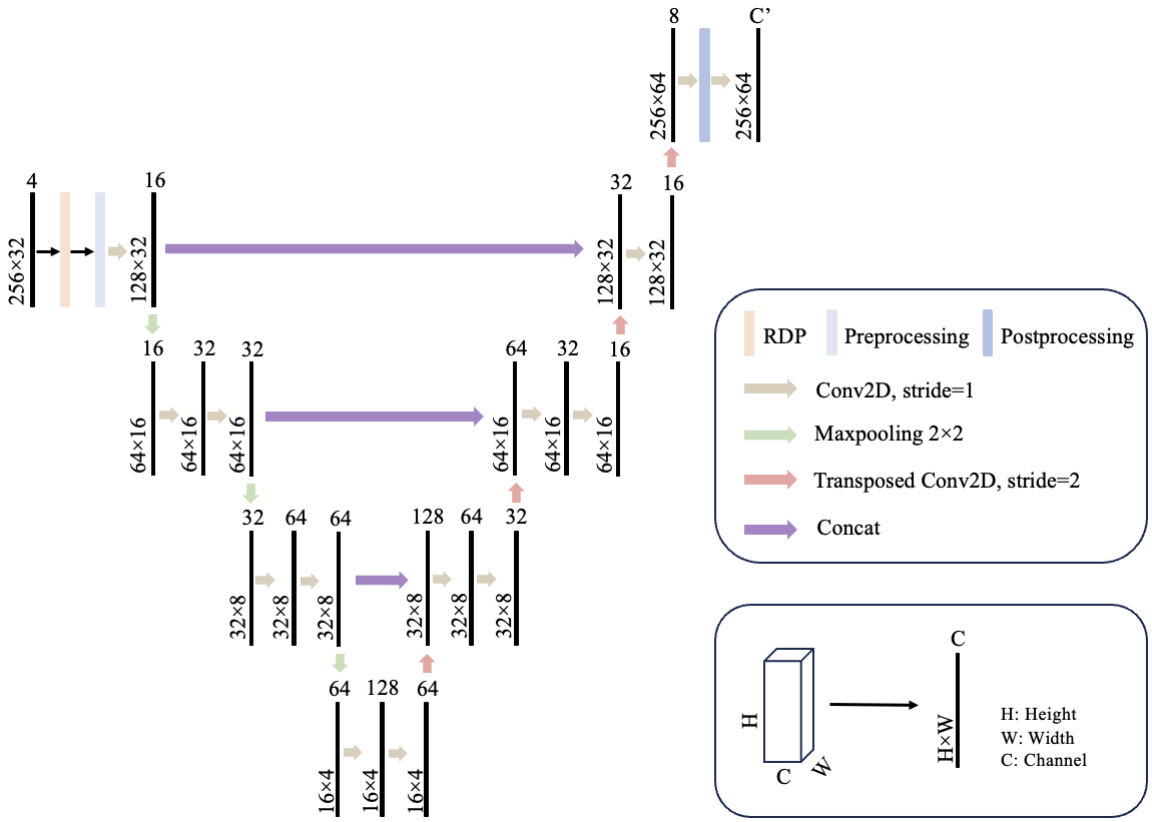


Figure 3.9.: UNet concat model

dimension is uniformly converted to 8.
In the UNet model, four times downsamplings are performed by default, and different numbers of upsampling layers are performed according to the resampling rate. When the resampling rate is 2, five times upsampling are performed. Before each upsampling layer, a convolutional layer with a stride of 1 is performed, and the number of channels is meanwhile modified. The upsampling operation uses the transposed convolutional layer by default. In postprocessing part, the corresponding unit parameters are set according to the data representation type.

3.5.2. UNet concat model

As mentioned above, to mitigate the information loss during downsamping operations, in UNet concat model the upsampling data will be concatenated with the downsampled data, as shown in Figure 3.9.
Compared with the UNet model in section 3.5.1, the UNet concat model has the following four main differences. First, the encoder part passes through two convolutional layers with a stride of 1 instead of just one layer. Moreover, in the decoder part, the UNet model only passes through the convolutional layer once to reduce the channel before the upsampling layer, while UNet concat model passes through two convolutional layers and reduces the channel dimension twice. The UNet model uses a convolutional layer with a stride of 2 for downsampling, while UNet concat model uses a 2×2 maxpooling layer. In addition, during each upsampling process, the data is concatenated with the corresponding downsampled data

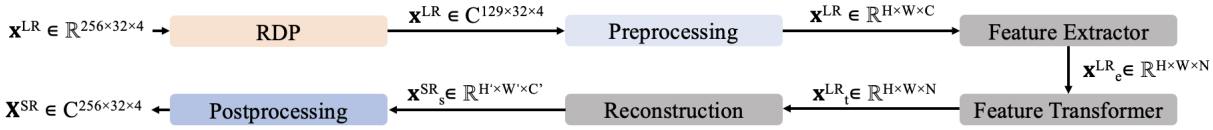


Figure 3.10.: DP-TF Transformer architecture, adapted from [12].

of the same size along the channel dimension.

3.6. DP-TF Transformer architecture

As mentioned in section 1.4, Transformer is a state-of-the-art model in the task of data upsampling. It uses the self-attention mechanism to learn the similarities and relationships between different blocks. In the range-Doppler map, the objects have a certain correlation in the dimension of the range and velocity related to the radar. Furthermore, the consecutive motion can also bring some additional motion information as well. In this model, it will mainly focus on the division along the two dimensions of range and velocity.
Figure 3.10 shows the overall structure of this model. Compared with the models mentioned previously, it mainly has three different blocks: feature extractor, feature Transformer, and reconstruction. According to [12], feature extractor is equivalent as an encoder, but since additional downsampling may cause information loss, the stride of the convolutional layer will be set as 1 in the featur extractor. The reconstruction is equivalent as the decoder, and the additional upsampling is performed according to the resample rate.

Feature Extractor

The feature extractor mainly includes two steps. The first step will perform a separable convolutional layer and a Layer Normalization (LN) to obtain an intermediate feature extractor representation x_{e1}^{LR} expressed as

$$x_{e1}^{LR} = \mathrm{LN}(\mathrm{SeparableConv2D}(x^{LR})), \tag{3.6}$$

where $x_{e1}^{LR} \in \mathbb{R}^{H\times W\times N}$ and H, W and N represent the height, width and channel size, respectively. In the second step, a SeparableConv2D layer and a LN are also performed, where SeparableConv2D layer still sets stride as 1 and does not perform downsampling in our case. In addition, a ReLU activation function layer will be used, written as

$$x_{e2}^{LR} = \mathrm{ReLU}(\mathrm{LN}(\mathrm{SeparableConv2D}(x_{e1}^{LR}))), \tag{3.7}$$

where $x_{e2}^{LR} \in \mathbb{R}^{H\times W\times N}$ still. According to the paper, another convolutional layer is used. Although the downsampling operation is not performed in our case, we can still keep it. The output of the feature extractor can be written as

$$x_{e}^{LR} = \mathrm{LN}(\mathrm{Conv2D}(x_{e2}^{LR})). \tag{3.8}$$

Feature Transformer

The feature Transformer mainly includes three hierarchies, as shown in Figures 3.11, 3.12, and 3.13. Figure 3.11 shows the overall structure of the feature Transformer, Figure 3.12

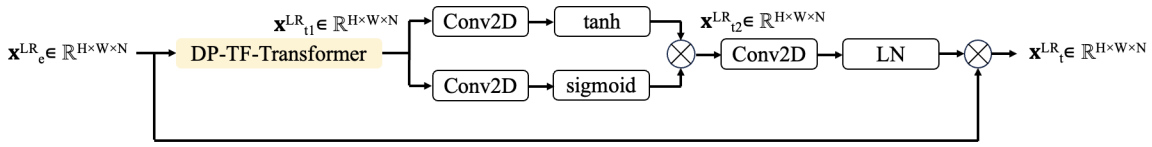


Figure 3.11.: Feature Transformer block, adapted from [12].

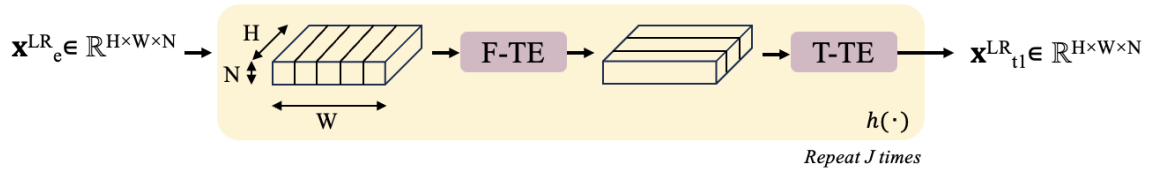


Figure 3.12.: DP-TF Transformer block, adapted from [12].

shows the structure of the DP-TF Transformer block and the way to do the data segmentation, and Figure 3.13 shows the structure of the Transformer Encoder (TE).
In the first hierarchy of the feature Transformer, the data will be firstly operated by the DP-TF Transformer block. Subsequently, the output passes through the dual path which can be seen as a gating mechanism. Each path includes a convolutional layer and an activation function. The activation function as tanh keeps the main content into (-1, 1) while the other activation function uses sigmoid that leads the output into (0, 1). Then the two values are multiplied elementwise which is beneficial for remaining the useful information and discarding the feature with lower importance, namely

$$\mathrm{x}^{\mathrm{LR}}_{\mathrm{t2}} = \tanh(\mathrm{Conv2D}(\mathrm{x}^{\mathrm{LR}}_{\mathrm{t1}})) \circ \mathrm{sigmoid}(\mathrm{Conv2D}(\mathrm{x}^{\mathrm{LR}}_{\mathrm{t1}})). \tag{3.9}$$

Furthermore, an additional convolutional layer and a layer normalization will be performed as well as a residual path, written as

$$\mathrm{x}^{\mathrm{LR}}_{\mathrm{t}} = \mathrm{x}^{\mathrm{LR}}_{\mathrm{e}} \circ \mathrm{LN}(\mathrm{Conv2D}(\mathrm{x}^{\mathrm{LR}}_{\mathrm{t2}})). \tag{3.10}$$

In the second hierarchy, that is, in the DP-TF Transformer block, the data are divided along the range and velocity axes and each division has a TE block respectively, described as

$$\mathrm{x}^{\mathrm{LR}}_{\mathrm{t1}} = h^{j}(\mathrm{x}^{\mathrm{LR}}_{\mathrm{e}}), \tag{3.11}$$

where $h^{j}(\cdot)$ denotes J times DP-TF Transformer block applications. Within the TE block, a sinusoidal positional encodings will be firstly added to the input as the representation of the

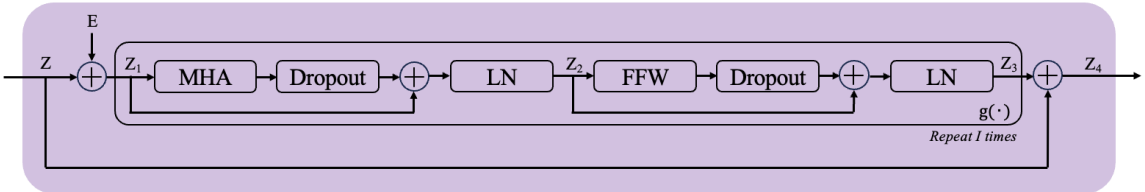


Figure 3.13.: TE block, adapted from [12].

order of the inputs, namely

$$Z_1 = Z + E, \tag{3.12}$$

then a module $g(\cdot)$ as shown in Figure 3.13 will be performed I times on Z_1 with a residual path with Z, written as

$$Z_4 = g^I(Z_1) + Z, \tag{3.13}$$

where within this module, a multi-head attention block and a dropout layer will be performed on Z_1, a residual path exists before the layer normalization, resulting in Z_2

$$Z_2 = \mathrm{LN}(Z_1 + \mathrm{Dropout}(\mathrm{MHA}(Z_1))). \tag{3.14}$$

Moreover, another similar part will be performed, replace the MHA with a point-wise Feed-Forward (FFW) network, resulting in Z_3

$$Z_3 = \mathrm{LN}(Z_2 + \mathrm{Dropout}(\mathrm{FFW}(Z_2))), \tag{3.15}$$

where FFW block contains two dense layers and a ReLU activation function, that is,

$$Z_2' = \mathrm{Dense}(\mathrm{ReLU}(\mathrm{Dense}(Z_2))), \tag{3.16}$$

Reconstruction

In the super-resolution data reconstruction, an upsampling layer will be performed firstly. If the transposed convolutional layer is used, it will combine with a layer normalization and a ReLU activation function, namely

$$x_{s1}^{SR} = \mathrm{ReLU}(\mathrm{LN}(\mathrm{Conv2DTranspose}(x_t^{LR}))). \tag{3.17}$$

According to the input data representation, the unit of the last convolutional layer could be determined as either 1 or 2 with the linear activation, written as

$$x_s^{SR} = \mathrm{Conv2D}(x_{s1}^{SR}). \tag{3.18}$$

3.7. SwinIR Transformer architecture

According to [11] mentioned in section 1.4, in addition to dividing the data along the two dimensions of range and velocity, it can also be divided into multiple quadratic range velocity patches. The structure is shown in Figure 3.14, which mainly includes three parts: shallow feature extraction, deep feature extraction, and super-resolution image reconstruction.
In the paper, the Swin Transformer is used in the SwinIR architecture, since we still have a DP-TF Transformer block, they can be combined as two models, namely SwinIR+Swin model and SwinIR+DP model.

3.7.1. SwinIR+Swin model

Shallow feature extraction

In the paper, the shallow feature extraction is seen as an encoder, which downsamples the

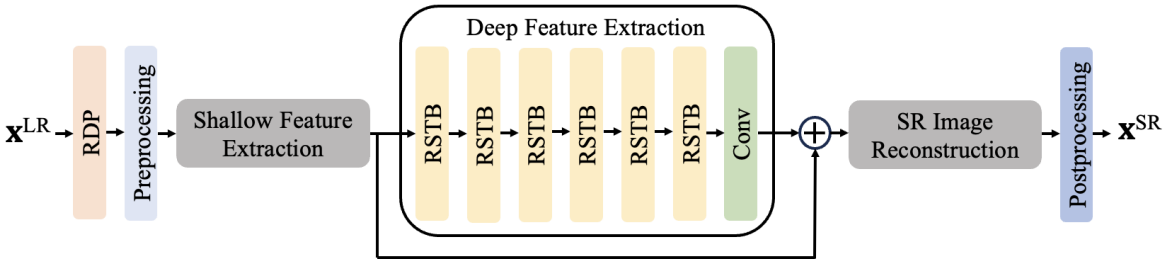


Figure 3.14.: SwinIR architecture, adapted from [11].

data once. In order to mitigate the impact of information loss, a convolutional layer is still performed here but the stride is determined as 1, then the data size remains unchanged while only the channel is increased, namely

$$x_s^{LR} = \text{Conv2D}(x^{LR}). \tag{3.19}$$

Deep feature extraction

In deep feature extraction, a dropout layer is performed first, then multiple Residual Swin Transformer Blocks (RSTB) and a convolutional layer are included, and the output of the deep feature extraction block is combined with the result of shallow feature extraction using the residual path, that is,

$$x_d^{LR} = \text{Conv2D}(\text{RSTB}^J(\text{Dropout}(x_s^{LR}))) + x_s^{LR}, \tag{3.20}$$

where J denotes the times of the RSTB applications. Figure 3.15a illustrates the structure of the RSTB, where it contains multiple Swin Transformer Layers (STL) blocks and a convolutional layer as well as the residual path, namely

$$Z_{R1} = \text{Conv2D}(\text{STL}^I(Z_R)) + Z_R, \tag{3.21}$$

where I represents the times of the STL applications. There are two parts inside the STL block, one part includes a layer normalization, a Multi-head Self-Attention (MSA) block as well as the residual part, described as

$$Z_{S1} = \text{MSA}(\text{LN}(Z_S)) + Z_S, \tag{3.22}$$

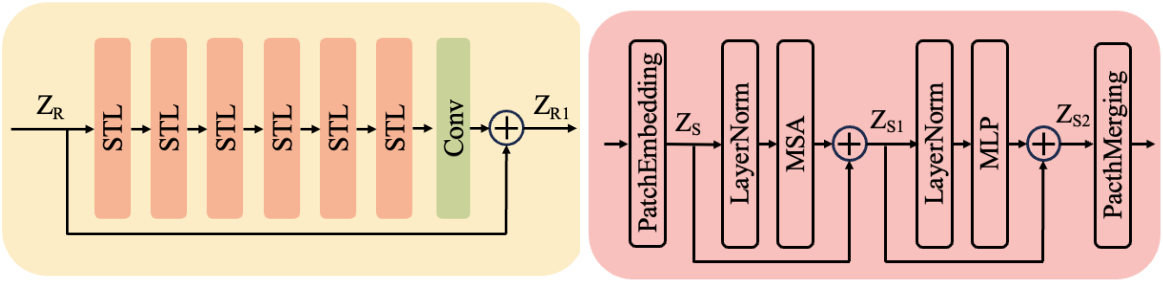
while another part contains a layer normalization, a Multi-Layer Perceptron (MLP) and the residual path, namely

$$Z_{S2} = \text{MLP}(\text{LN}(Z_{S1})) + Z_{S1}. \tag{3.23}$$

In the MLP block, five steps are carried out in sequence, namely dense layer, GeLU activation function, dropout layer, dense layer with linear activation function and dropout layer, denoted as

$$Z_{M1} = \text{Dropout}(\text{GeLU}(\text{Dense}(Z_M))), \tag{3.24}$$

$$Z_{M2} = \text{Dropout}(\text{Dense}(Z_{M1})). \tag{3.25}$$



(a) Structure of the RSTB block (b) Structure of the STL block

Figure 3.15.: The structure of RSTB and STL blocks, adapted from [11].

SR Image Reconstruction

The SR image reconstruction block includes three parts: the upsampling part as well as convolutional part before and after upsampling. There is a convolutional layer with the LeakyReLU activation function before the upsampling part, that is,

$$x_r^{LR} = \text{LeakyReLU}(\text{Conv2D}(x_d^{LR})). \tag{3.26}$$

In the next step, the upsampling part is same as the reconstruction part in the section 3.6, namely the equation 3.17 and 3.18 resulting in x_{rl}^{SR} if using the transposed convolutional layer. After the upsampling part, another convolutional layer with the linear activation function is performed to make sure that the channel dimension of the output is according to the input data representation, namely the unit as either 1 or 2, written as

$$x_r^{SR} = \text{Conv2D}(x_{rl}^{SR}). \tag{3.27}$$

3.7.2. SwinIR+DP model

Combining with DP-TF Transformer block, the difference between SwinIR+Swin model is the STL block. The structure of the STL block as shown in Figure 3.15b can be replaced by the DP-TF Transformer block which is illustrated in Figure 3.12.

3.8. Comparison between the architectures

According to all the models mentioned above, the differences between most models are relatively clear. For example, there is no downsampling process in the CNN model. The difference between UNet and UNet concat models is mainly whether to combine the data of the encoder part with the corresponding decoder part. However, for the DP-TF Transformer model, the SwinIR+DP and SwinIR+Swin models, the data division is the main difference, but there are still some other small differences. Therefore, this section will focus on two parts, one is the difference between the DP-TF Transformer and SwinIR architectures, and the other is the difference between the DP-TF and STL Transformer blocks. The evaluation of this section will shown in the appendix.

3.8.1. Differences between DP-TF and SwinIR Transformer architectures

The main structures of the two architectures are very similar, both of which contain three parts: encoder, deep feature extraction, and decoder, so the differences will be listed in this order.

Encoder

They both have the encoder part, in DP-TF Transformer architecture it's called the feature extractor block, whereas in SwinIR it's called shallow feature extraction.

1. The feature extractor block of the DP-TF Transformer architecture uses the separable convolutional layer while in SwinIR the normal convolutional layer, as written in formulas 3.6 and 3.19.

2. After the first separable convolutional layer, there's another layer normalization in the DP-TF Transformer architecture.

3. In the feature extractor block, DP-TF Transformer architecture has an additional convolutional layer, layer normalization as well as the ReLU activation function layer, namely formula 3.7.

Transformer blocks

Both of the architectures use some blocks to loop the Transformer modules, feature Transformer in DP-TF architecture and deep feature extraction in SwinIR architecture.

4. SwinIR architecture has a dropout layer before the first RSTB block in the deep feature extraction block, as shown in formula 3.20.

5. Before the DP-TF Transformer block in DP architecture, another convolutional layer with layer normalization exists, that is, formula 3.8.

6. SwinIR architecture doesn't use dual path after the RSTB blocks as in DP-TF Transformer architecture shown in Figure 3.11.

7. SwinIR sets a convolutional layer after RSTB and STL blocks, whereas DP-TF architecture has convolutional layer with layer normalization after the dual path, i.e. in Figures 3.14, 3.15a and 3.11.

8. In DP-TF Transformer architecture, the residual path in the feature Transformer uses multiply operation rather than adding, namely formula 3.10.

9. SwinIR architecture has an overall residual path, combining the output of the shallow feature extraction with the output of the deep feature extraction, as shown in Figure 3.14.

Decoder

The decoder part is called reconstruction in DP-TF Transformer architecture while called SR image reconstruction in SwinIR architecture. In this part, they don't have many differences, only one is that the kernel size in reconstruction is set as 1 by default while in SR image reconstruction block as a hyperparameter.

3.8.2. Differences between DP-TF and Swin Transformer blocks

Compared with the Transformer block as shown in Figure 3.13 and 3.15b, some differences are existing as following:

10. According to the formulas 3.16 and 3.24, the activation functions are different, ReLU in FFW block while GeLU in MLP block.
11. The order of the items in the TE and STL blocks is different.
12. The residual path in both Transformer blocks are not the same, in Swin Transformer block there's no overall residual path but the layer normalization is inside the residual paths while in DP-TF Transformer block the layer normalization is after each residual paths.

3.9. cGAN architecture

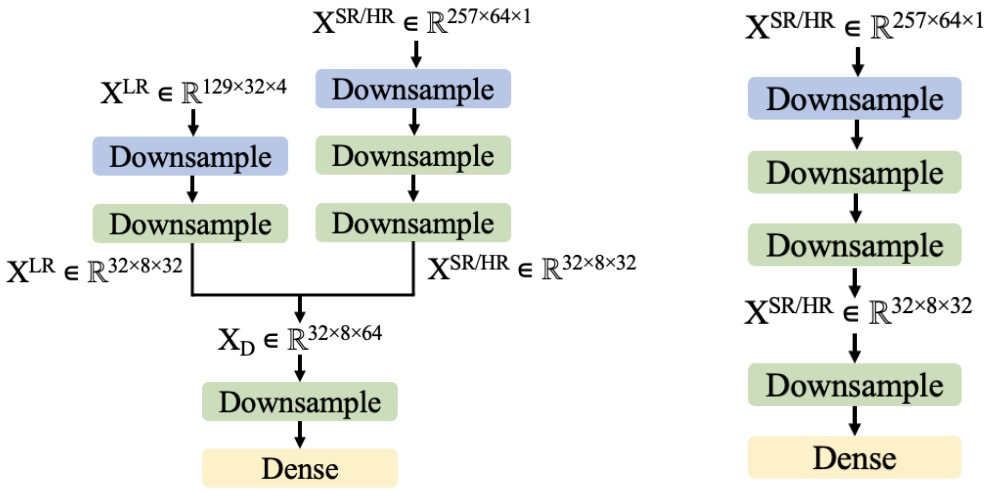
As mentioned in section 1.4, some image upsampling papers use cGAN model. Although it may cause some loss functions to increase in value, the generator and discriminator can be trained mutually and the super-resolution range-Doppler map could be visually better.

3.9.1. Generator

In our task, generator refers to the model that learns from the source domain input as the low-resolution range-Doppler maps and generates super-resolution range-Doppler maps. All the models mentioned above can be used as the generator, but in order to reduce the complexity of combinations and evaluations, in the section 6.1, we will compare the performance of different models under various evaluation loss functions and select the good one as the generator.

3.9.2. Discriminator

The task of the discriminator is to distinguish between super-resolution range-Doppler map and truly high-resolution range-Doppler map. Through the learning and supervision of the discriminator, the output of the generator could look more realistic. The output of the discriminator will be a probability value, indicating the probability that the input data could be the truly high-resolution data. For the discriminator, its goal is to have the output probability approaching 0 when the input is the super-resolution data, and approaching 1 with the high-resolution data as input.
For the discriminator, we build two structures, depending on whether the input contains the corresponding low-resolution range-Doppler maps, as illustrated in Figure 3.16. The discriminator in Keras tutorial [29] uses both source domain input as the low-resolution range-Doppler map and the target domain input as the super- or high-resolution range-Doppler



(a) Conditional discriminator inclusive low-resolution range-Doppler map as condition

(b) Discriminator exclusive low-resolution range-Doppler map as input

Figure 3.16.: The structures of the discriminator in terms of the input

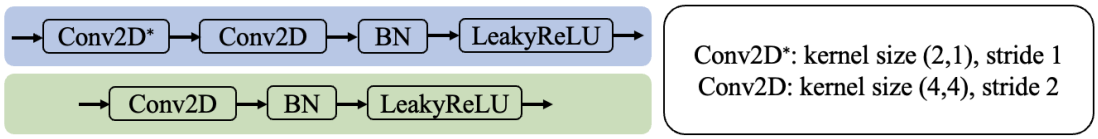


Figure 3.17.: Downsample block in the discriminator

map, while in some papers, such as [20], the low-resolution range-Doppler map is not input as additional information.

The structures of both discriminators are to obtain a probability value after a series of downsampling operations, and then performed by a dense layer with sigmoid as the activation function. The difference is that there is an extra concatenating operation in Figure 3.16a. Furthermore, as the range-Doppler maps are given into the discriminator in the first step, the dimension processing type is determined as the convolutional layer, since the height is an odd value and the later convolutional layers set stride as 2 as well as the batch normalization layer and leaky ReLU activation function, shown in Figure 3.17.

4. Loss functions

In order to train the model with different settings, we tried many loss functions. This chapter will show all the loss functions used. The loss functions will be slightly different depending on the process, namely for the training and evaluation phases. There are a total of eight loss functions and their combined loss functions.

4.1. MSE loss

Mean Square Error (MSE) is a common loss function in machine learning. Its formula is written as

$$\mathcal{L}_{\text{MSE}} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \tag{4.1}$$

where Y_i represents the ground truth, $\hat{Y}_i$ denotes the prediction and n is the total number of the samples. $\mathcal{L}_{\text{MSE}}$ represents the MSE loss between the ground truth and prediction, and the goal of the model is to reduce this value.
But in specific cases, the meaning of Y_i and $\hat{Y}_i$ and the usage of this loss function will be different, that is, the format of the values in ground truth and prediction is depending on the processing methods, namely the red part shown in Figure 3.2. Furthermore, it can be formulated into two loss functions according to the needs in the range-Doppler map upsampling task, namely MSE and Weighted MSE (WMSE).

4.1.1. MSE

In normal MSE, no additional weights are added to the loss function, but it is still performed depending on the different processes, training or evaluation phase.

Training&validation phases

In the phase of training and validation, the meaning of the Y_i and $\hat{Y}_i$ is firstly up to the input data representation, that is, in the case of real and imaginary representation type, the loss of both real and imaginary parts will be calculated separately, where Y_i refers to them of the high-resolution data and $\hat{Y}_i$ denotes that of the super-resolution data accordingly, then both will be combined in the way of

$$\mathcal{L}_{\text{MSE}} = \mathcal{L}_{\text{MSE, Re}} + \mathcal{L}_{\text{MSE, Im}}, \tag{4.2}$$

whereas in the case of amplitude and phase representation type, the loss of amplitude and phase will be calculated respectively and combined as

$$\mathcal{L}_{\text{MSE}} = \mathcal{L}_{\text{MSE, Amp}} + \lambda \times \mathcal{L}_{\text{MSE, Ph}}, \tag{4.3}$$

where λ as a hyperparameter can be tuned according to the processing methods. But if only amplitude, then the phase MSE loss will be neglected.

Evaluation phases

Since a lot of settings and combinations exist, in order to make the evaluation fair for all the cases and due to the importance of the amplitude in the range-Doppler map upsampling, only the amplitude loss will be taken into consideration, that is,

$$\mathcal{L}_{\text{MSE, Amp}} = \frac{1}{n}\sum_{i=1}^{n}(A_i - \hat{A}_i)^2, \tag{4.4}$$

where A and $\hat{A}$ are the amplitude of the high-resolution and super-resolution data, respectively. Note that in the training phase, the training loss is calculated between ground truth Y and prediction $\hat{Y}$ which are before the postprocessing block as shown in Figure 3.2. However, in the evaluation phase, the amplitude of the ground truth A and prediction $\hat{A}$ are converted back after the postprocessing block.

4.1.2. WMSE

The large dynamic range of radar signal amplitudes causes relatively low MSE loss for weak signal parts in comparison to strong reflections. The model will not pay much attention on these signals. Therefore, we assign different weights according to the amplitude of the signal. In order to make the weaker signal also get enough attention and the weaker signals get a larger weight, and the formula of amplitude WMSE loss will become

$$\mathcal{L}_{\text{WMSE, Amp}} = \frac{1}{n}\sum_{i=1}^{n}\frac{(A_i - \hat{A}_i)^2}{A_i}, \tag{4.5}$$

while the angle loss in WMSE is still same as in the MSE and combine them with an additional weight λ as well in the case of the amplitude and phase representation type, namely

$$\mathcal{L}_{\text{WMSE}} = \mathcal{L}_{\text{WMSE, Amp}} + \lambda \times \mathcal{L}_{\text{MSE, Ph}}, \tag{4.6}$$

while in the evaluation process, only the amplitude loss part, namely formula 4.5, is used. The reason is same as the explanation in the section 4.1.1.

4.2. SDR loss

In the field of signal processing, a very important indicator is SNR. We hope to get a larger signal related to the noise. According to the paper [30] by Jonathan et al., the classical SNR is equivalent to the `bss_eval_images`'s SDR in the case that there's only noise between the ground truth A and prediction $\hat{A}$, that is,

$$\mathcal{L}_{\text{SDR}} = 10\log_{10}\left(\frac{\|A\|^2}{\|A - \hat{A}\|^2}\right). \tag{4.7}$$

As shown in Figure 4.1, Jonathan et al. proposed two loss functions as the variations of the

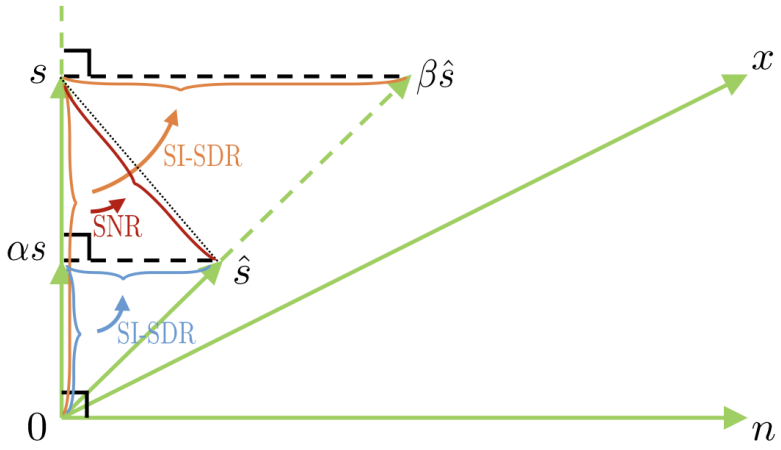


Figure 4.1.: Illustration of the SNR and SI-SDR [30]

Signal-to-Distortion Ratio (SDR), called Scale-Independent Signal-to-Distortion Ratio (SI-SDR) and Scale-Dependent Signal-to-Distortion Ratio (SD-SDR), which are respectively written as

$$\mathcal{L}_{\text{SI-SDR}} = 10\log_{10}\left(\frac{\|e_{\text{target}}\|^2}{\|e_{\text{res}}\|^2}\right) = 10\log_{10}\left(\frac{\left\|\frac{\hat{Y}^T Y}{\|Y\|^2} Y\right\|^2}{\left\|\frac{\hat{Y}^T Y}{\|Y\|^2} Y - \hat{Y}\right\|^2}\right), \tag{4.8}$$

$$\mathcal{L}_{\text{SD-SDR}} = 10\log_{10}\left(\frac{\|\alpha s\|^2}{\|s - \hat{s}\|^2}\right) = \mathcal{L}_{\text{SDR}} + 10\log_{10}\alpha^2, \tag{4.9}$$

where the optimal scaling factor is $\alpha = \hat{Y}^T Y/\|Y\|^2$. The advantage of these two variations is the faster convergence, but the SI-SDR is sensitive to the scaling and SD-SDR is not sure in the frequency domain. Therefore, we keep using the classical SDR loss, namely formula 4.7. Moreover, the current SDR is better when it's larger, so to make it decrease during the training process, a negative sign is added.

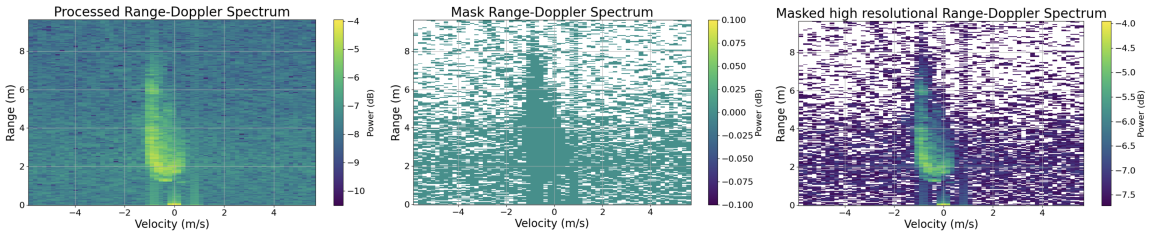
4.3. LSD loss

Braun et al. proposed two loss functions [31], Logarithmic Spectral Distance (LSD) and Phase-aware Logarithmic Spectral Distance (PLSD), which add logarithmic operation on the amplitude compared to classical MSE and are beneficial for the range-Doppler maps with the significant amplitude fluctuation.

4.3.1. LSD

LSD can be described as

$$\mathcal{L}_{\text{LSD, Amp}} = \left\langle \left|\log_{10}\hat{A} - \log_{10}A\right|^2 \right\rangle, \tag{4.10}$$



(a) Processed high-resolution range-Doppler map (b) The mask in the percentile of 50% (c) Masked high-resolution range-Doppler map

Figure 4.2.: The mask operation on the high-resolution range-Doppler map with logarithic amplitude

where A represents the amplitude of each point in the range-Doppler map and

$$\langle P(h,w)\rangle = \frac{1}{HW}\sum_{h,w} P(h,w). \tag{4.11}$$

To improve the stability further, the root mean square operation can combine with the LSD loss function, it turns to

$$\mathcal{L}_{\text{LSD, Amp}} = \left\{\frac{1}{N}\sum_{n=1}^{N}\left[\log_{10} A(n) - \log_{10}\hat{A}(n)\right]^p\right\}^{\frac{1}{p}}, \tag{4.12}$$

where p is set as 2. With the use of logarithmic operations, the amplitude change will be obviously reduced. However, since the data with high amplitude in the range-Doppler map is relatively only a small part while the part in the longer range or velocity in range-Doppler map occupy the majority, the model will pay much attention on the background noise during the training process, affecting the final super-resolution range-Doppler map. Therefore, a mask is added on the LSD loss to ignore much background noise during the training process. The threshold is set according to the median value or the percentile, only the loss of the part which has the amplitude over the threshold will be taken into consideration, otherwise will be neglected. The mask is still based on the logarithm amplitude. If the percentile is set as 50%, i.e. 50% data can over the threshold, the mask looks like Figure 4.2b, where Figure 4.2a shows the corresponding high-resolution range-Doppler map with the logarithmic amplitude and Figure 4.2c illustrates the masked high-resolution range-Doppler map.
In terms of the phase loss, the MSE can be used and combined with the scaling factor λ, namely

$$\mathcal{L}_{\text{LSD}} = \mathcal{L}_{\text{LSD, Amp}} + \lambda \times \mathcal{L}_{\text{MSE, Ph}}, \tag{4.13}$$

whereas in the evaluation phase, the scaling factor is set as zero, that is, only the amplitude LSD loss is taken into consideration and the mask will not be used. Note that, if in the processing methods the logarithm is performed, both MSE and LSD will have logarithmic operation on the amplitude, then the difference is only the square root in LSD loss function. Conversely, if no logarithm operation in the processing methods, the LSD loss function keeps using logarithm on the amplitude while the MSE doesn't have.

4.3.2. PLSD

Compared with the LSD loss function, the difference of the PLSD loss function is that it doesn't use the MSE phase loss, that is,

$$\mathcal{L}_{\text{PLSD}} = \left\langle \left| \log_{10} \left| \frac{\hat{Y}}{Y} \right| \right| \times \left(2 - \mathcal{R}\left\{ \frac{\hat{Y}}{Y} \right\} \right) \right\rangle, \tag{4.14}$$

where

$$\mathcal{R}\left\{ \frac{\hat{Y}}{Y} \right\} = \cos\left(\varphi_{\hat{Y}} - \varphi_{Y}\right). \tag{4.15}$$

PLSD loss function can be only used in training and validation phases if not amplitude representation type.

4.4. Perceptual loss

The Very Deep Convolutional Networks (VGG) [32] are common models for image reconstruction, which is implemented using multiple blocks of the convolutional layers. Simonyan et al. proposed six VGG configurations in different scales, each with the different number of layers and parameters, as shown in Table 4.1, where the configuration E represents VGG19, which has the most parameters and the highest accuracy.
Johnson et al. proposed a perceptual loss function based on the VGG model, using the features extracted from different layers of the VGG model to calculate the loss between super-resolution and high-resolution data [33]. Compared with the per-pixel loss function mentioned above, perceptual loss function can produce a more visually pleasing range-Doppler map. They proposed two types of the perceptual loss functions, one is about the feature reconstruction loss which is the Euclidean distance between the feature representation

$$\mathcal{L}_{\text{Perceptual}}(\hat{Y}, Y) = \frac{1}{HWC} \left\| \phi(\hat{Y}) - \phi(Y) \right\|_2^2, \tag{4.16}$$

where $\phi(\cdot)$ denotes the feature extraction of the VGG model and another type is based on the style of the image, namely

$$G^{\phi}(x)_{c,c'} = \frac{1}{HWC} \sum_{h=1}^{H} \sum_{w=1}^{W} \phi(\hat{Y})_{h,w,c} \phi(Y)_{h,w,c'}. \tag{4.17}$$

In our case, the upsampling data is specifically the range-Doppler map, therefore, the feature reconstruction perceptual loss is used rather than the style. In VGG model, with the deeper channel of blocks, the representation of each block changes. In the first two blocks, the shallow features are learned, such as margin, color and brightness. Since the third block, more deeper feature are represented, such as texture. And in fourth and fifth blocks, some distinguished features are extracted. Therefore, in the perceptual loss function of the pipeline, the last layer of the first four blocks in the pretrained VGG model provided by TensorFlow are used to extract the feature representation by default, since it can mitigate the loss computation to be much expensive. Furthermore, since the channel dimension of the range-Doppler map is just one, while in the VGG model, RGB image is used for pretraining, the range-Doppler

Table 4.1.: VGG configurations [32]

VGG Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

map has to extend the channel dimension.

4.5. Combination of the losses

According to the various loss functions mentioned above, their types and goals are somewhat different. For example, LSD loss function uses logarithmic operations to pay more attention on the signal with low amplitude, while the perceptual loss function improves the overall perception of the range-Doppler map using the feature representation. Therefore, different types of loss functions can be combined with each other, such as

$$\mathcal{L}_{\text{Combination}} = \mathcal{L}_{\text{LSD}} + \lambda_c \times \mathcal{L}_{\text{Perceptual}}, \tag{4.18}$$

where λ_c as a hyperparameter that can be tuned according to the importance or the values of both loss functions. Moreover, in order to improve the stability, the pretraining operation can be used, that is, the model can be trained with one loss of the loss combination and after a few epochs trained by both. Otherwise, at the very beginning of the training, if the perceptual loss is relatively large, it would cause too much attention on the perception rather than the target LSD loss per pixels.

4.6. Adversarial loss

In cGAN model, the generator and discriminator models will be trained mutually, whereas the loss functions for both are different. Therefore in this section, the generator loss function and discriminator loss function will be shown respectively.

4.6.1. Generator loss

For the generator model, on one hand, it should reduce the value of the loss combination, such as the LSD combining with perceptual loss, after multiple epochs of training, and on the other hand, it should make the discriminator believe that the super-resolution range-Doppler map is the real high-resolution range-Doppler map. Since the output of the discriminator model is a probability value, the adversarial loss for the generator part should be regarded as the difference between this probability and 100%, that is,

$$\mathcal{L}_{\text{GAN, Gen}} = \|1 - P_{\text{Super}}\|_2, \tag{4.19}$$

where P_{Super} represents the probability value of the super-resolution range-Doppler map as the real high-resolution range-Doppler map from the discriminator model. The total generator loss function should combine with the target loss function mentioned above, namely

$$\mathcal{L}_{\text{Gen}} = \mathcal{L}_{\text{Combination}} + \lambda_g \times \mathcal{L}_{\text{GAN, Gen}}, \tag{4.20}$$

4.6.2. Discriminator loss

For the discriminator model, its goal is to make the probability of identifying the super-resolution and ground truth as real high-resolution range-Doppler maps approaching 0 and 1 respectively. Therefore, the discriminator loss function also contains two parts, namely

$$\mathcal{L}_{\text{Disc}} = \|1 - P_{\text{High}}\|_2 + \|0 - P_{\text{Super}}\|_2, \tag{4.21}$$

where P_{High} is the probability value that the ground truth is really the high-resolution range-Doppler map.

5. Training optimization

In this chapter, the methods and settings to optimize the training process in the pipeline are shown, including but not limited to distributed training, early stopping, learning rate schedule and hyperparameter tuning.

5.1. Distributed training

In order to make full use of multiple Graphics Processing Unit (GPU)s for training and speed up the training process, TensorFlow provides the distributed training Application Program Interface (API) to achieve parallel computing across multiple GPUs, machines or TPUs. Depending on the device, the strategy has five types, as shown in Table 5.1.
According to the case in our pipeline, the custom training loop is chosen and multiple GPUs with the same model in the same machine for each training are used, so the mirrored strategy is more in line with our needs. The principle is illustrated in Figure 5.1, that is, it will divide the mini-batch data equally by the number of GPUs into each GPU, where each small divided data of the mini-batch is called a replica. The GPUs contain the same model, and each GPU uses the replica for training to update parameters, obtain loss and super-resolution range-Doppler maps. Once all GPUs have completed the training process, the obtained parameters and losses are averaged, and all super-resolution range-Doppler maps can be concatenated, then the training process of a batch of data is finished.
Compared with the original custom training loop, the updates in the pipeline are using strategy to load the model and the optimizer, as well as set up the checkpoint. Additionally, the custom training loop should be run as a new loop within the strategy loop, so that the strategy will automatically divide the batch and average the loss and parameters. It is noted that the operation on the loss should be selected as sum by default, since the strategy will then automatically calculate the average of the sum value.

Table 5.1.: Overview of the distributed training strategies in TensorFlow [34]

Training API	**Mirrored Strategy**	**TPUStrategy**	**MultiWorker-Mirrored Strategy**	**CentralStorage Strategy**	**ParameterServer Strategy**
Keras Model.fit	Supported	Supported	Supported	Experimental Supported	Experimental Supported
Custom training loop	Supported	Supported	Supported	Experimental Supported	Experimental Supported
Estimator API	Limited Support	Not Supported	Limited Support	Limited Support	Limited Support

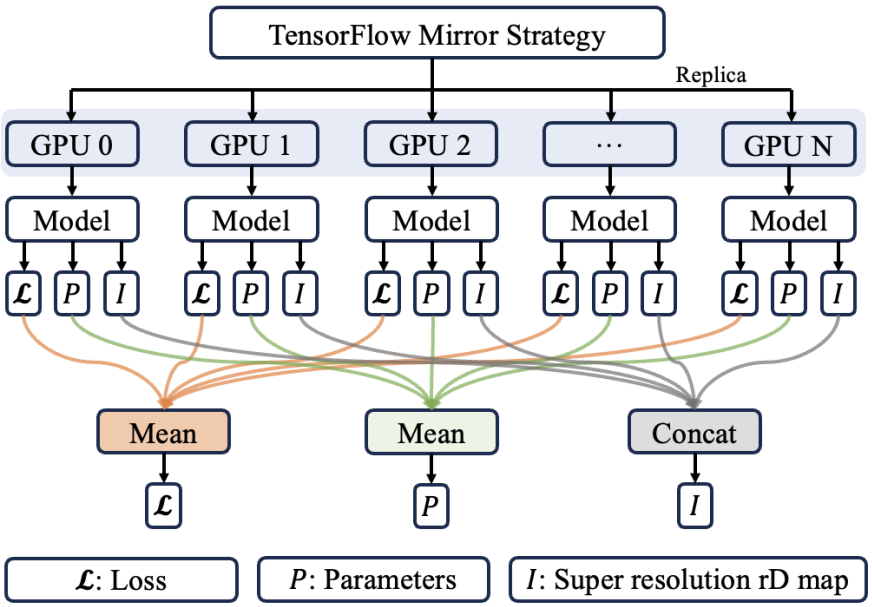


Figure 5.1.: Illustration of the mirrored distributed training strategy

5.2. Settings in the pipeline

In this section, some settings are introduced in order to speed up the training process as well as save the checkpoints for restoring the model in the evaluation phase.

Early stopping

In the training loop, the training and validation losses are obtained after each epoch if not pretraining. Otherwise, after pretraining phase, each epoch has a validation loss. By default, the training loop is terminated if the validation loss does not decrease after 10 consecutive epochs to avoid that the training process doesn't have progress due to overfitting but occupies the resources for a long time after setting a higher number of epochs.

Learning rate schedule

In order to speed up the training process and adjust the learning rate at different training stages to help the model converge better, the learning rate schedule is used in the pipeline. According to the size of low-resolution training set as `(28353, 4, 32, 256)`. By default, after every 10,000 batches, the learning rate lowers by factor 0.96 and the initial learning rate as a hyperparameter can be determined according to the different cases. The learning rate goes as shown in Figure 5.2 in the case that the number of the epochs is 200.

Checkpoint setting

In order to separate the training and evaluation phases and effectively prevent the interruption during the training process, the trainable parameters of the model can be saved in checkpoints. The strategy in the pipeline is by default to save the checkpoint immediately after the first epoch, and then save the current model parameters in the checkpoint as the validation loss decreases in the next epochs. To prevent storing too many checkpoints of the ealier stage, the default setting only retains the latest 5 checkpoints.

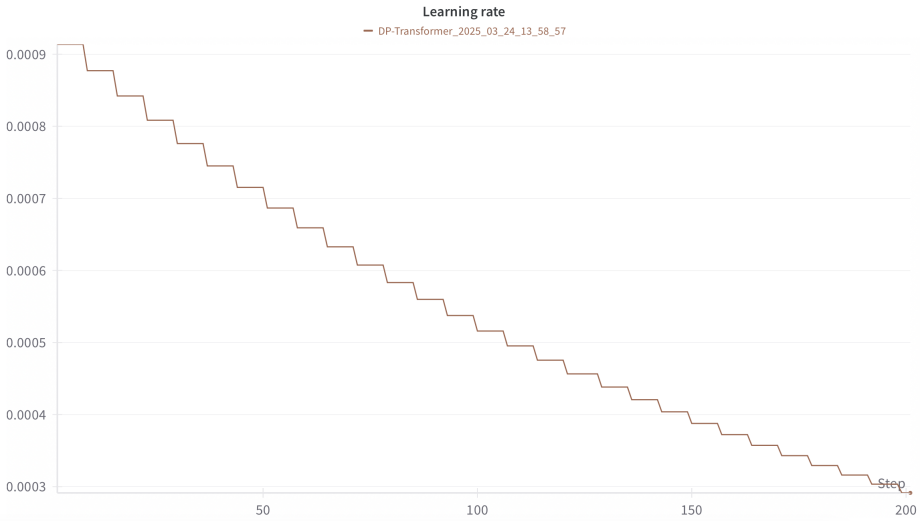


Figure 5.2.: Overview of the learning rate schedule in the case of 200 epochs, decay steps as 10,000, decay rate as 0.96 and initial learning rate as 9.1e-4.

WandB

WandB [35] is a tool that can be used to track, visualize, and collaborate on machine learning projects. It provides many functions, such as tracking indicators, storing data, tuning hyperparameters, and visualizing models. In our pipeline, WandB is mainly used for recording and tuning. In the training loop, WandB will record the low-resolution, super-resolution, and high-resolution range-Doppler map of both the training set and validation set in the last batch of each epoch, for a total of six range-Doppler maps. Meanwhile, it records the training loss, validation loss, current best validation loss, learning rate, and running time of each epoch, where Figure 5.2 is recorded by WandB. Furthermore, WandB will automatically record some system usage, such as memory usage, power usage, etc. Additionally, in our pipeline, WandB is also used for sweeping the hyperparameters to tune.

Optimizer

There are many choices for optimizers, among which Batch Gradient Descent (BGD) and Stochastic Gradient Descent (SGD) are the basic optimizers. Then the new methods were designed such as Momentum, which are used to speed up optimization and convergence. After that, optimizers such as Adaptive gradient algorithm (Adagrad) and Root Mean Square Prop (RMSProp) were proposed, which will optimize the parameters according to the importance, namely larger updates to parameters with low importance but smaller updates to parameters with high importance, which greatly improves robustness as well. The most commonly used optimizer at present is Adaptive Moment Estimation (Adam), which effectively combines the advantages of RMSProp and Momentum and achieves better results [36].
In our pipeline, the Adam optimizer is used by default. During the tuning process, the other above-mentioned adaptive optimizers could be tested, such as RMSProp and so on. Moreover, there are two APIs to load optimizer in TensorFlow, where the `tf.keras.optimizers.legacy` requires a higher version of TensorFlow but can provide faster speed, while the normal `tf.keras.optimizers` is suitable for lower versions, so the lower version is used by default in the server but can use the faster one in the local computer.

Pre-training

Since the training with VGG perceptual loss and cGAN is not stable in the early epochs and in order to make sure that the optimization of the model goes to the ideal direction, we introduced the pretraining phase. As a training loss function is combined with perceptual loss, or when cGAN is used, the training processing will be only affected by the LSD loss function in the first 50 epochs by default to ensure the stability in the early stage.

5.3. Hyperparameter optimization

WandB provides a sweep function that can be used to combine different hyperparameters and find the optimal solution based on the goal. In our pipeline, WandB will record the best validation loss during training, and hence, the goal is set to minimize this value during tuning. Sweep function provides a variety of combination methods, such as grid, random, and Bayes, among which random combination of the hyperparameters is the most common method. In the pipeline, multiple specified configuration files of each model can be created to save the hyperparameters which are going to be tuned, so that the settings can be modified easily during the tuning process. For the training loop, some settings or hyperparameters can be tuned, such as the type of optimizer, learning rate, batch size, etc. In terms of the models, such as DP-TF Transformer model, the hyperparameters to be tuned could be the number of filters, the kernel size of encoder and decoder, as well as the number of the blocks, etc.

6. Results

As mentioned in the previous chapters, compared with the upsampling approaches in other papers, this thesis has many updates in the model, data processing and loss function, so this chapter will mainly focus on the evaluation of these three aspects, but also some others. Since only its amplitude is useful in the visualization of range-Doppler map, the evaluation loss will be only based on the amplitude and only in the frequency domain as mentioned in chapter 4. Furthermore, during the training process, the input of the model may be processed in many different ways, and the pipeline uses the processed output to calculate the training loss, whereas in the evaluation phase, the data will be converted back to the complex value without logarithm and normalization to calculate the evaluation loss.
Since there are too many combinations, in order to make an intuitive and clear comparison, each evaluation will ensure that other parameters are the same, and only one difference will be compared each time, and the most suitable one will be determined for the subsequent evaluations. Due to the large number of evaluation cases, the size of the model and data are selected as a small one, whereas a large model will be trained with the whole dataset at the end. The evaluation indicators are mainly divided into two parts, one is the values of the evaluation losses. The smaller the evaluation loss, the better the model. The other is the visual effect of the generated super-resolution range-Doppler map.

6.1. Models comparison

Firstly, a model shall be determined to evaluate various subsequent processing methods and loss functions. Here, the classical loss function and common processing methods are used, namely the MSE loss function for training, convolutional layer as the dimension processing layer for the prime range shape and transposed convolutional layer as the upsampling layer, using the real and imaginary representation type, and without any logarithm or normalization operations, that is, the MSE loss function is calculated in linear scale.

Table 6.1.: Evaluation losses of different models in the case of the MSE and common processing methods.

Models comparison	#Params	MSE	SDR	LSD	WMSE	Perceptual
Interpolation	0	3.570	-3.620	0.750	0.774	28.010
CNN_simple	103,684	2.164	-5.028	**0.441**	3.599	25.133
UNet_simple	93,162	1.352	-2.872	0.600	1.034	22.014
UNet_concat	96,746	**1.275**	-3.609	0.552	0.968	**21.847**
DP	113,964	3.212	-4.926	0.611	**0.163**	23.583
SwinIR+DP	97,252	1.754	**-5.787**	0.503	0.314	22.764
SwinIR+Swin	111,624	1.541	-4.505	0.546	1.104	26.381

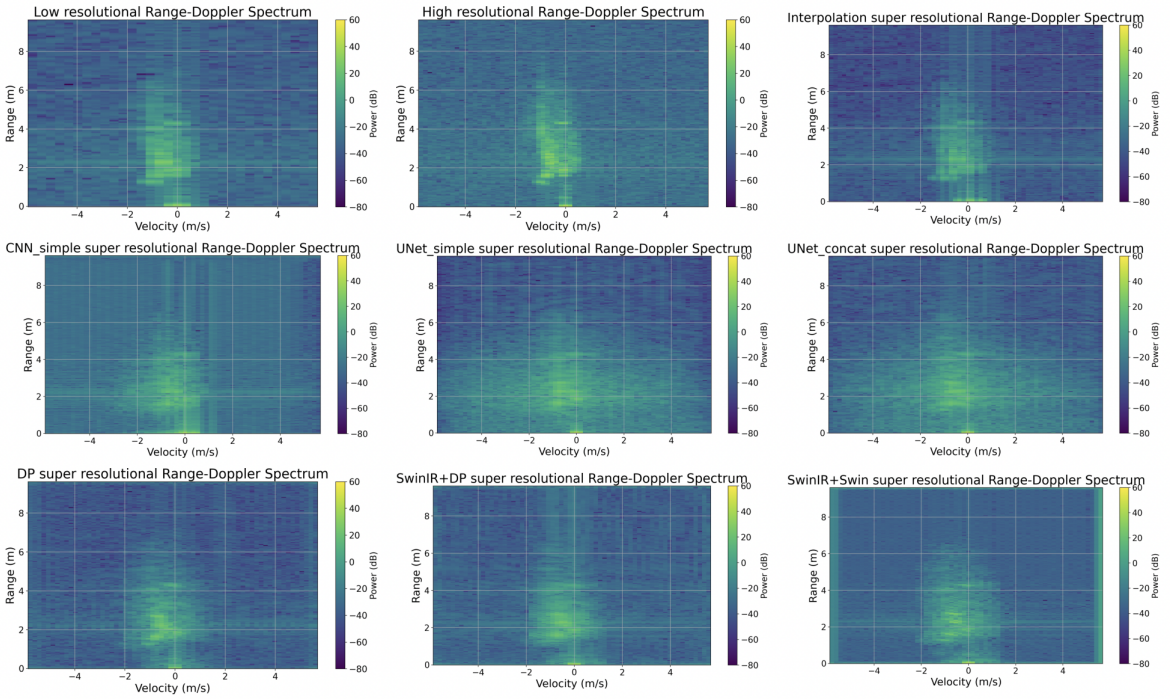


Figure 6.1.: Super-resolution range-Doppler maps of different models in the case of the MSE and common processing methods.

From the results of the evaluation losses in Table 6.1, most models have one good result, but from the perspective of super-resolution range-Doppler maps shown in Figure 6.1, the basic CNN model and both UNet models are not as good as the Transformer models, since there seems to be some information loss in the area of the strong signals. Among them, the DP-TF Transformer model and SwinIR architecture with DP Transformer block model seem to be better at present. The subsequent evaluations on the processing methods and training loss functions will temporarily use the DP-TF Transformer model.
According to the optimal processing methods evaluated in section 6.2 and the loss combination evaluated in section 6.3, we evaluate again the models. Table 6.2 and Figure 6.2 show the evaluation losses and super-resolution range-Doppler maps in the new case.
After using the new combination of the processing methods and the loss combination, most of the evaluation losses are optimized. Furthermore, visual effect of the super-resolution range-

Table 6.2.: Evaluation losses of different models in the case of the loss combination and new processing methods.

Models comparison	MSE	SDR	LSD	WMSE	Perceptual
Interpolation	2.485	-6.825	0.435	1.305	21.898
CNN_simple	3.574	-7.722	0.345	0.227	26.649
UNet_simple	3.126	-7.918	0.338	0.225	19.148
UNet_concat	1.992	**-8.355**	**0.320**	0.175	17.916
DP	1.731	-5.375	0.380	0.179	**12.320**
SwinIR+DP	1.662	-6.415	0.360	**0.128**	16.903
SwinIR+Swin	**1.619**	-6.392	0.360	0.134	15.662

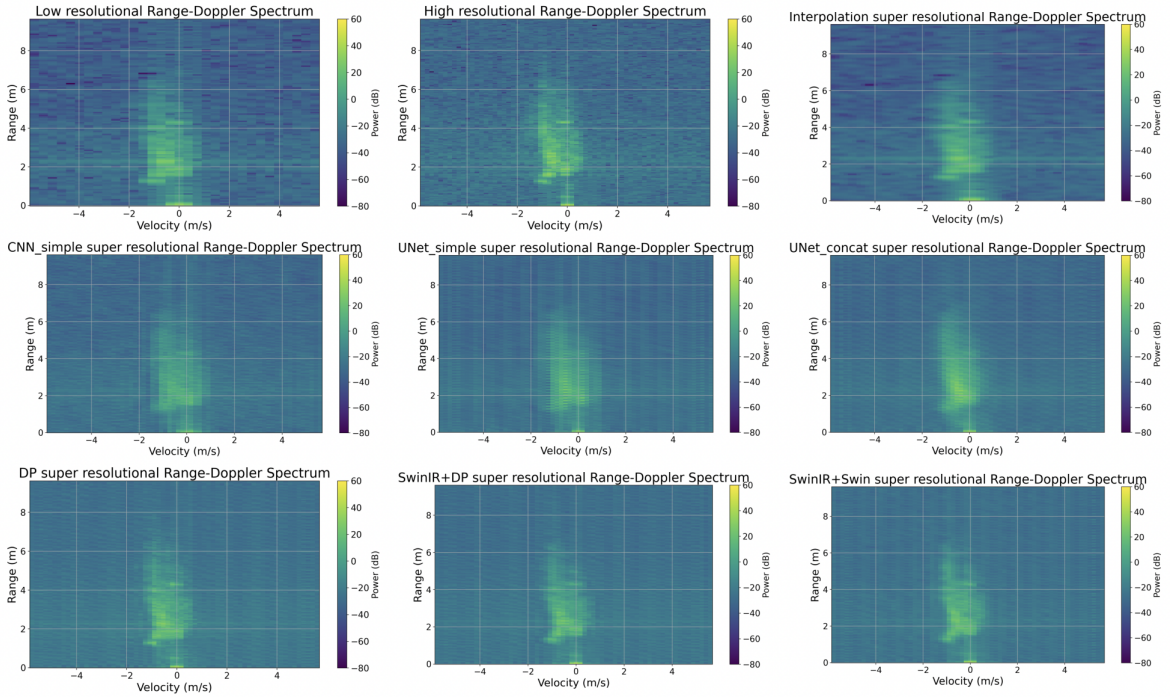


Figure 6.2.: Super resolution range-Doppler maps of different models in the case of the loss combination and the new processing methods.

Doppler map is improved as well. Some peaks in the high-resolution range-Doppler map can be clearly seen in the super-resolution range-Doppler map, especially in the Transformer models. Balanced by the evaluation losses and the visual effect, the DP-TF Transformer model and SwinIR+DP model are still better, where DP-TF Transformer model performs well in the perceptual loss while SwinIR+DP model is good at the other losses. Since the peaks look clearer in DP-TF Transformer model, following sections will keep using DP-TF Transformer model.

6.2. Processing methods

Based on the above DP-TF Transformer model, this section will compare the different combinations of the processing methods, namely input data representation, dimension processing types, upsampling types, logarithm and normalization operations.

Input data representation

While processing complex-valued data, there are three types of the input data representation, namely real and imaginary, amplitude as well as amplitude and phase. The other processing methods keep no dimension processing layer, the transposed convolutional upsampling layer, and do not use any logarithm and normalization operations, which means only the representation type is changed compared with the first case in section 6.1. The loss metrics and super-resolution range-Doppler map are shown in Table 6.3 and Figure 6.3 respectively.
Analyzed from the results, since the visualization of range-Doppler map is affected by amplitude, it is beneficial to use amplitude for training. From the perspective of evaluation losses, training with only amplitude is the best. However, since it loses the information of phase,

Table 6.3.: Evaluation losses of different types of the input data representation, where A1 represents the real and imaginary representation, A2 is the amplitude representation and A3 denotes the amplitude and phase representation.

Representation types	MSE	SDR	LSD	WMSE	Perceptual
A1	3.435	-4.958	0.619	**0.155**	24.682
A2	**3.136**	**-7.778**	**0.385**	0.181	**18.101**
A3	3.213	-5.068	0.493	0.241	20.126

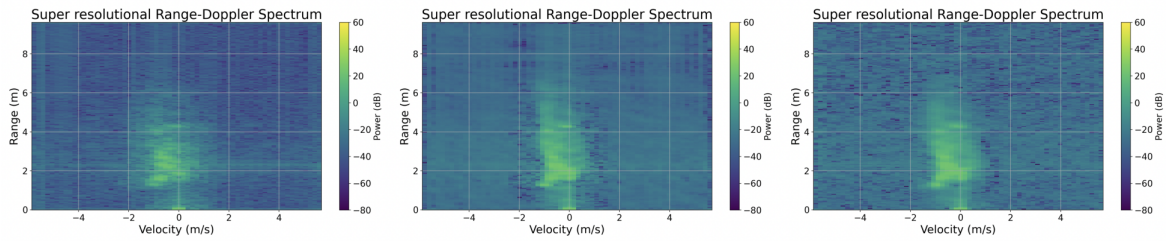


Figure 6.3.: Super-resolution range-Doppler maps of different types of the input data representation, from left to right real and imaginary representation, amplitude representation as well as amplitude and phase representation, respectively.

and the amplitude and phase representation type is mostly better than real and imaginary representation type in terms of the evaluation losses and visual effect, the amplitude and phase representation type will be maintained in the subsequent evaluations.

Dimension processing types

Operated by the rFFT, the shape along the range axis turns to an prime value, which will affect both the Transformer division and the downsampling process. However, since there is no downsampling operation and segmentation along the axes in the DP-TF Transformer model, it can still run normally without additional dimension processing. Therefore, there are three dimensional processing types: no processing, padding, and convolution. Based on the amplitude and phase representation type and keeping other settings unchanged, the loss metrics in Table 6.4 and the super-resolution range-Doppler maps in Figure 6.4 are obtained. While the convolutional layer as the dimension processing type leads to more parameters, the padding processing type can still obtain a better result in multiple evaluation losses. The reason could be that using convolution layer to remove a dimension along the range axis may result in information loss. Subsequent evaluations will be based on the padding dimension processing type.

Table 6.4.: Evaluation losses of different dimension processing types, where B1 represents no processing type, B2 is the padding and B3 denotes the convolutional layer as the processing type.

Dimension processing	MSE	SDR	LSD	WMSE	Perceptual
B1	3.100	-6.312	0.427	0.194	19.081
B2	3.113	**-6.611**	**0.420**	**0.190**	19.092
B3	**2.349**	-5.908	0.451	0.206	**16.259**

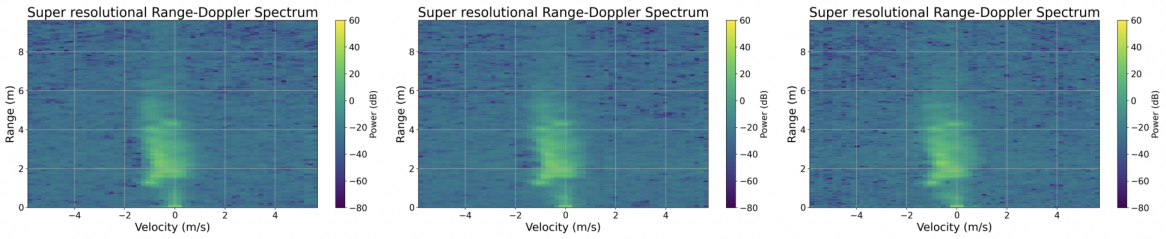


Figure 6.4.: Super-resolution range-Doppler maps of different dimension processing types, from left to right no processing, padding and convolution layer, respectively

Table 6.5.: Evaluation losses of different upsampling types, where C1 represents the transposed convolutional layer and C2 denotes the pixel shuffle approach as the upsampling layer.

Upsampling types	#Params	MSE	SDR	LSD	WMSE	Perceptual
C1	113,866	3.113	**-6.611**	**0.420**	**0.190**	19.092
C2	48,042	**2.560**	-4.206	0.506	0.269	**17.918**

Upsampling types

The two most common upsampling methods are transposed convolutional layer and pixel shuffle approach. The block, containing transposed convolutional layer, LN and ReLU activation function, will result in a lot of new parameters, while the pixel shuffle operation does not generate any new parameters. Table 6.5 and Figure 6.5 show the performance of two approaches in terms of evaluation losses and super-resolution range-Doppler maps.
Although the transposed convolutional layer brings a large number of new parameters, it does not surpass pixel shuffle layer in all loss metrics, especially the perceptual loss is higher. Visually, the right range-Doppler map in Figure 6.5 is even more reasonable for the dynamic range part. Therefore, the pixel shuffle layer will be used as the upsampling type in the subsequent evaluations.

Logarithm types

When the amplitude is part of the input, the range of the amplitude will affect the learning process, causing it to focus more on areas with higher signal power and ignore the loss caused by areas with less amplitude. Therefore, in this evaluation, the impact of the logarithm

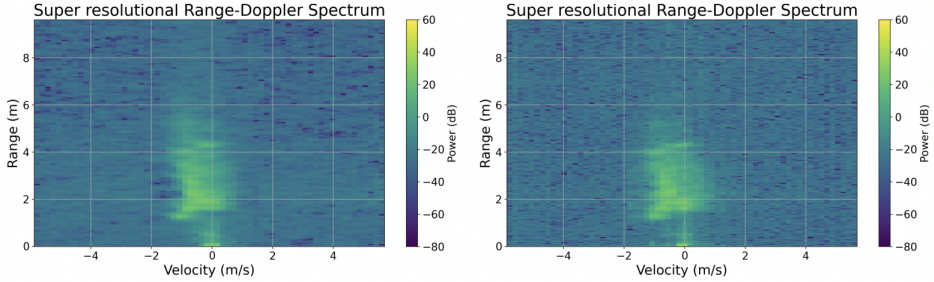


Figure 6.5.: Super-resolution range-Doppler maps of different upsampling types, left using the transposed convolutional layer and right with the pixel shuffle layer.

Table 6.6.: Evaluation losses of the effect of the logarithm, where D1 represents no logarithm operation, D2 uses the base 10 logarithm operation. Accordingly, the MSE training loss in D1 case uses the amplitude in linear scale while in D2 uses the logarithmic amplitude.

Logarithm types	MSE	SDR	LSD	WMSE	Perceptual
D1	2.560	-4.206	0.506	0.269	17.918
D2	**1.740**	**-9.251**	**0.295**	**0.130**	**17.636**

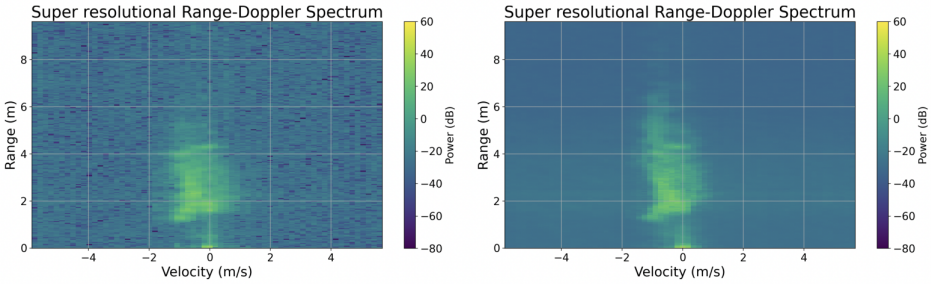


Figure 6.6.: Super-resolution range-Doppler maps of the cases without logarithm and with logarithm operation, from left to right, respectively.

operation will be evaluated.
From Table 6.6, it can be clearly seen that after using the logarithm operation with base 10, significant improvements have been achieved in all evaluation losses. However, compared with the left range-Doppler map in Figure 6.6, the right super-resolution range-Doppler map has a blurring effect in the background. The reason would be that in the left range-Doppler map the model pays less attention on the low amplitude signal as well as the noise and uses smaller values instead. After using the logarithm, the signals with low amplitude get more attention while the noise is still hard to learn, so that the noise is mostly predicted as relatively low amplitude and then look blurry.

Amplitude normalization types

Similar to the idea of logarithm operation, the normalization operation can reduce the dynamic range of the data. According to the maximum and minimum values of each dataset, the amplitude of each dataset can be accordingly converted to the range of (-1, 1) or (0, 1) . In addition, it can also be not normalized. Furthermore, since the normalization will result in a small gradient, the learning rate is tuned by the cases, only the case without normalization uses the learning rate as 1e-5, otherwise the learning rate as 1e-3. The results of the evaluation losses and the super-resolution range-Doppler maps are shown in Table 6.7 and Figure 6.7 respectively.
In general, the introduction of the normalization operation is conducive to reduce multiple evaluation losses, among which the optimization of perceptual loss is the most obvious, which represents visual optimization, since the VGG expects the normalized input, ideally in the (0, 1) range. The normalization in (0, 1) form obtains the most balanced result, so the amplitude normalization will be kept in the range of (0, 1) in subsequent evaluations.

Angle normalization types

Table 6.7.: Evaluation losses of different amplitude normalization types, where E1 represent no normalization operation, E2 is in the range of (-1, 1), E3 denotes in the range of (0,1).

Amplitude normalization	MSE	SDR	LSD	WMSE	Perceptual
E1	1.740	**-9.251**	**0.295**	0.130	17.636
E2	1.711	-6.841	0.349	**0.108**	14.770
E3	**1.073**	-6.327	0.358	0.119	**14.639**

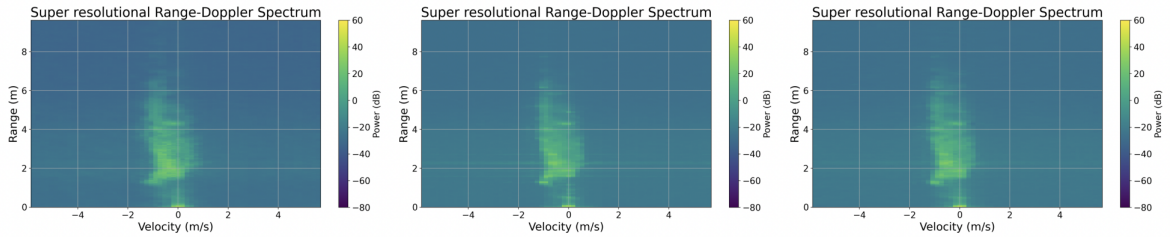


Figure 6.7.: Super-resolution range-Doppler maps of different amplitude normalization types, from left to right, no amplitude normalization, normalization in (-1, 1) and normalization in (0, 1), respectively.

In the amplitude and phase representation type, in addition to the amplitude being normalized, the angle can also be normalized and converted to (-1, 1) divided by π. The results are shown in Table 6.8 and Figure 6.8.
From the results, the normalization of the angle doesn't bring much improvement in the most evaluation losses. The reason is that the angle itself is within the range of ($-\pi$, π), and its dynamic range is not that large as amplitude. Therefore, the normalization operation will not cause many differences.
In conclusion, based on the above evaluations and analysis, the combination of processing methods tends to use padding for dimensional processing, pixel shuffle approach as the upsampling type, amplitude and phase representation determined as the conversion of the complex-valued data, logarithm operation and normalization operation in the range of (0, 1) on the amplitude, but no additional normalization operation performed on the phase.

6.3. Training loss functions

Different from the evaluation loss functions, there are six types of training loss functions. Based on the optimal combination of the processing methods combination obtained in the

Table 6.8.: Evaluation losses of the effect of the angle normalization type, where F1 represent no angle normalization operation, F2 denotes the angle normalization in the range of (-1, 1).

Angle normalization	MSE	SDR	LSD	WMSE	Perceptual
F1	**1.073**	**-6.327**	**0.358**	0.119	**14.639**
F2	1.330	-6.068	0.363	**0.117**	15.204

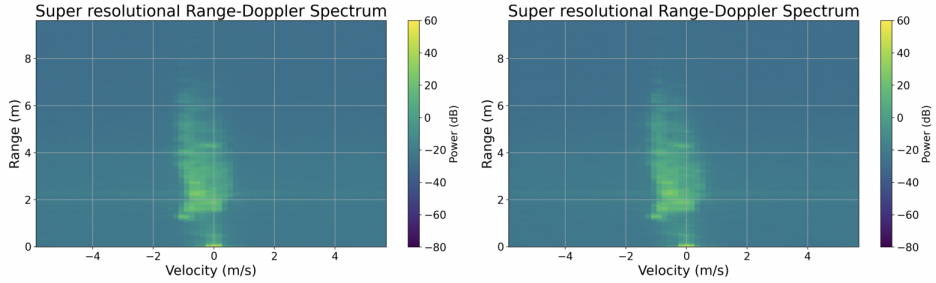


Figure 6.8.: Super-resolution range-Doppler maps of the effect of the angle normalization type, left without angle normalization and right with angle normalization.

Table 6.9.: Evaluation losses of different training loss functions, horizontal axis is the evaluation loss functions, vertical axis is the training loss functions.

Loss comparison	MSE	SDR	LSD	WMSE	Perceptual
MSE	**1.073**	-6.327	0.358	0.119	14.639
SDR	1.110	-5.683	0.370	0.121	14.163
LSD	1.232	**-6.736**	**0.351**	**0.108**	14.198
PLSD	1.424	-6.434	0.356	0.124	16.999
WMSE	1.279	-5.967	0.365	0.131	17.478
Perceptual	6.174	-4.129	0.417	0.393	**13.882**

section 6.2 and DP-TF Transformer model, different training loss functions are used to train the model, so as to obtain one training loss function that is most suitable for the range-Doppler map upsampling task. The hyperparameters such as the ratio λ between the amplitude loss and angle loss in the training loss function and learning rate will be adjusted according to their values according to a few batches at the very beginning. In addition, SDR and perceptual loss functions require a pretraining phase by the LSD training loss function, since they are not stable in the early stage. Table 6.9 and Figure 6.9 are the results obtained by different training loss functions.
From the results, it can be seen that the LSD training loss function can obtain better results in most of the evaluation losses, but from the visual perspective of super-resolution range-Doppler maps, the range-Doppler maps trained by perceptual loss effectively eliminate the blurring problem, that is, the amplitude difference in the background can be seen. Therefore, the training loss function LSD can be combined with the perceptual loss to take advantage of both pros.

Loss combination

According to the loss combination as written in formula 4.18, the relationship λ_c between LSD and perceptual loss should be determined. We compare a few loss values of the early batches and make sure that the main loss value comes from the LSD loss but still influenced by the perceptual loss to a certain content, then Table 6.10 and Figure 6.10 are obtained.
The overall relationship between the LSD loss function and perceptual loss function is that with the importance of the perceptual loss increasing in the loss combination, the evaluation perceptual loss decreases while the other evaluation losses is increasing. Since the loss combination is going to optimize the perceptual loss, the hyperparameter λ_C is determined as 0.5,

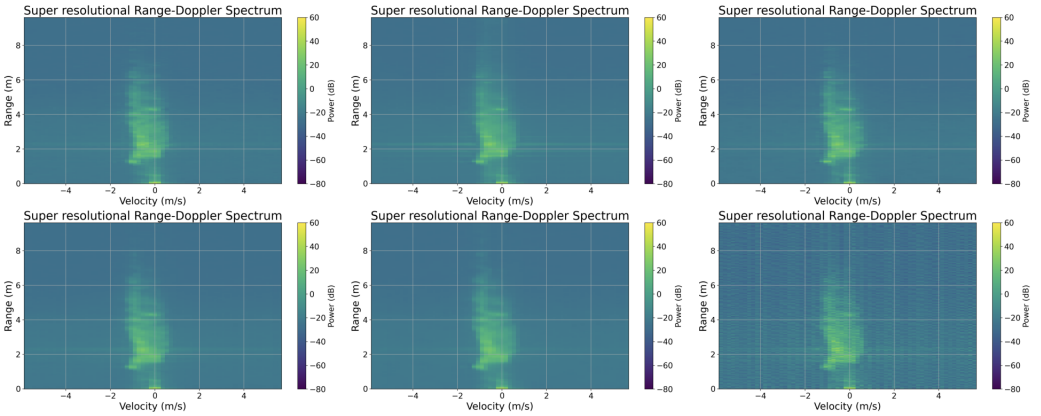


Figure 6.9.: Super-resolution range-Doppler maps of different training loss functions, in the first row from left to right MSE, SDR and LSD, in the second row from left to right PLSD, WMSE and perceptual loss functions.

Table 6.10.: Evaluation losses of the training loss combination with different ratios.

LSD combination	MSE	SDR	LSD	WMSE	Perceptual
Lambda=1	2.464	-5.270	0.383	0.220	13.779
Lambda=0.5	1.731	-5.375	0.380	0.179	**12.320**
Lambda=1e-1	1.347	-5.829	0.369	1.150	12.979
Lambda=1e-2	1.288	-6.514	0.355	0.116	13.213
Lambda=1e-3	1.486	**-7.268**	**0.341**	**0.108**	13.729
Lambda=1e-4	**1.265**	-6.405	0.357	0.115	14.138

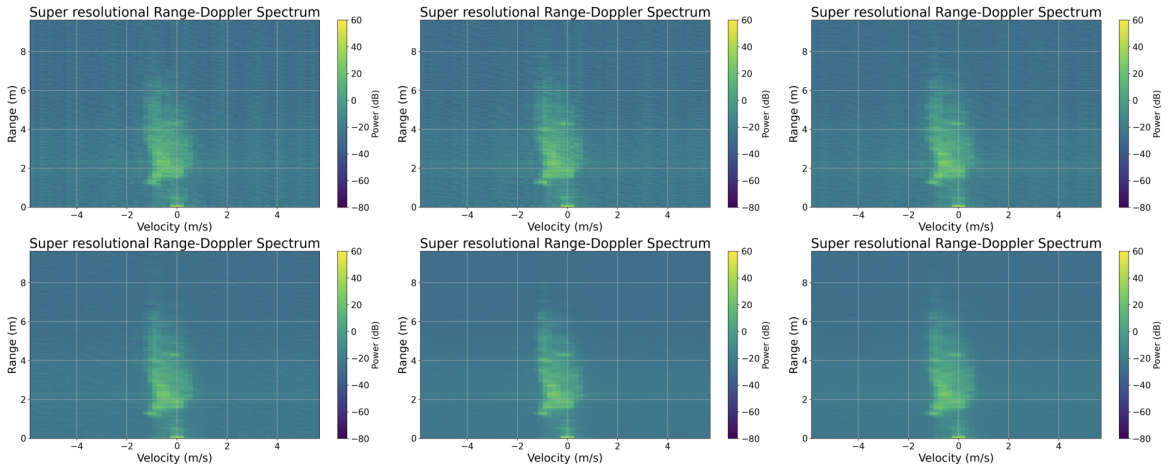


Figure 6.10.: Super-resolution range-Doppler maps of the training loss combination with different ratios, in the first row from left to right are lambda as 1, 0.5 and 0.1, while in the second row from left to right 1e-2, 1e-3, 1e-4.

Table 6.11.: Evaluation losses of the adversarial loss combination with different ratios in the case of the discriminator without low-resolution range-Doppler maps as input.

cGAN	MSE	SDR	LSD	WMSE	Perceptual
λ_g=5e-1	1.802	-5.216	0.383	0.186	13.543
λ_g=1e-1	1.621	-5.183	0.384	0.172	12.898
λ_g=5e-2	**1.447**	-5.463	0.379	0.181	13.311
λ_g=1e-2	1.640	-5.690	0.374	**0.169**	**11.881**
λ_g=5e-3	1.525	-5.305	0.382	0.193	12.956
λ_g=1e-3	2.072	**-5.845**	**0.370**	0.182	14.396

that is, the training loss function written as

$$\mathcal{L}_{\text{Combination}} = \mathcal{L}_{\text{LSD}} + 0.5 \times \mathcal{L}_{\text{Perceptual}}. \tag{6.1}$$

6.4. cGAN comparison

With the mutual supervision of the generator and discriminator, the output of generator can be visually more pleasing. Therefore, in this section, the cGAN model is evaluated, the generator keeps using the DP-TF Transformer model, and target training loss function is the loss combination from section 6.3. The generator loss is denoted as

$$\mathcal{L}_{\text{Gen}} = \mathcal{L}_{\text{LSD}} + 0.5 \times \mathcal{L}_{\text{Perceptual}} + \lambda_g \times \mathcal{L}_{\text{GAN, Gen}}, \tag{6.2}$$

where λ_g is going to be tuned. Furthermore, as said in section 3.9.2, the discriminator has two types, no low-resolution range-Doppler maps or with low-resolution range-Doppler maps as input. Therefore, both cases have been evaluated, where Table 6.11 and Figure 6.11 show the results in the case of the discriminator without any low-resolution information while Table 6.12 and Figure 6.12 are the results given the low-resolution range-Doppler maps.
It can be seen that cGAN model is able to improve the perceptual loss, no matter with or without the low-resolution range-Doppler map. Both cases are better, especially when the adversarial loss ratio sets as 1e-2, whereas the discriminator without the low-resolution range-Doppler map gets a relatively better perceptual evaluation loss. The reason could be that the concatenating operation of the low-resolution and high(super)-resolution range-Doppler map would confuse the discriminator, that is, the discriminator focuses more on the mapping relationship between low- and super-resolution range-Doppler map, rather than on the super-resolution range-Doppler map itself, and additionally, the loss calculation in the perceptual loss has no inputs of the low-resolution range-Doppler map. Perhaps the comparison directly between the super- and high-resolution range-Doppler maps is simple for the discriminator rather than with the low-resolution range-Doppler map as the medium. In conclusion, the discriminator without low-resolution range-Doppler map will be used and the generator loss turns to be

$$\mathcal{L}_{\text{Gen}} = \mathcal{L}_{\text{LSD}} + 0.5 \times \mathcal{L}_{\text{Perceptual}} + 0.01 \times \mathcal{L}_{\text{GAN, Gen}}. \tag{6.3}$$

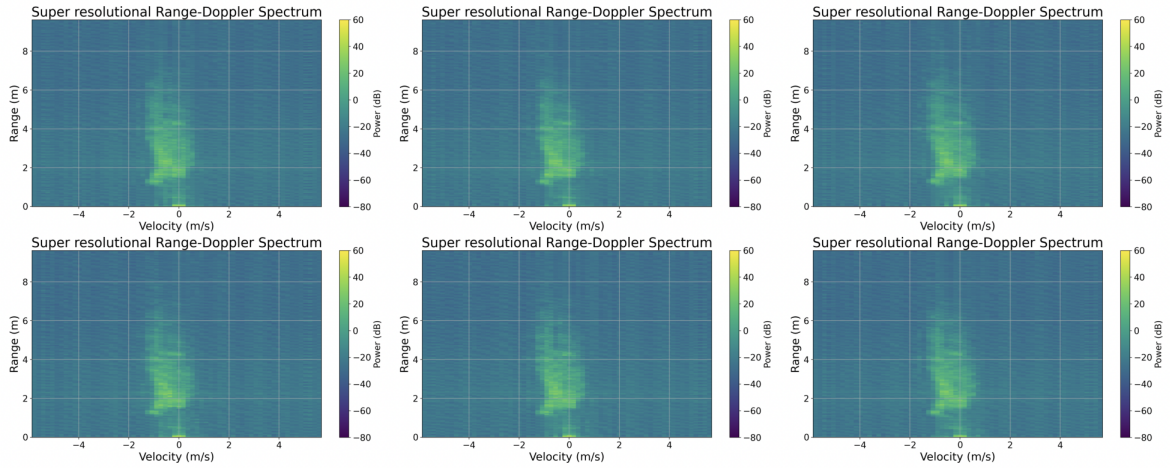


Figure 6.11.: Super-resolution range-Doppler maps of the adversarial loss combination with different ratios in the case of the discriminator without low-resolution range-Doppler maps as input, in the first row from left to right are λ_g as 5e-1, 1e-1 and 5e-2, while in the second row from left to right 1e-2, 5e-3, 1e-3.

Table 6.12.: Evaluation losses of the adversarial loss combination with different ratios in the case of the discriminator with low-resolution range-Doppler maps as input, where the number of parameters of the generator is 48,042 while that of the discriminator is 76,267.

cGAN	MSE	SDR	LSD	WMSE	Perceptual
λ_g=1e-1	1.592	-5.361	0.381	0.164	13.861
λ_g=5e-2	1.491	-5.628	**0.374**	**0.158**	12.126
λ_g=1e-2	**1.339**	-5.103	0.385	0.177	**11.926**
λ_g=5e-3	1.628	-5.461	0.379	0.165	13.511
λ_g=1e-3	1.920	**-5.648**	0.375	0.159	13.257
λ_g=1e-4	3.593	-5.331	0.381	0.190	14.148

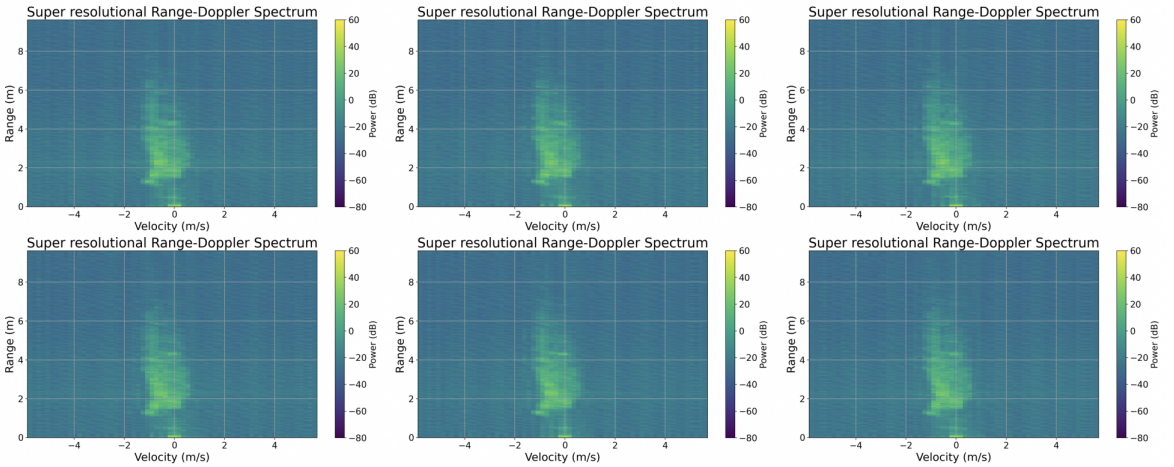


Figure 6.12.: Super-resolution range-Doppler maps of the adversarial loss combination with different ratios in the case of the discriminator with low-resolution range-Doppler maps as input, in the first row from left to right are λ_g as 1e-1, 5e-2 and 1e-2, while in the second row from left to right 5e-3, 1e-3 and 1e-4.

6.5. VGG layers in the perceptual loss

In this section, different combinations of the VGG layers are evaluated, by default the last convolutional layers of the first four blocks are used. Table 6.13 and Figure 6.13 show the results trained by the loss combination with different VGG layers used. With the deeper layer to be used, many evaluation losses decrease. However, in order to improve the perceptual loss, the default combination is still better one.

Table 6.13.: Evaluation losses of different combinations of the VGG layers in the training loss combination.

VGG layers comparison	MSE	SDR	LSD	WMSE	Perceptual
b1c2+b2c2+b3c4	1.975	-5.366	0.380	0.167	12.793
b1c2+b2c2+b3c4+b4c4	1.731	-5.375	0.380	0.179	**12.320**
b1c2+b2c2+b3c4+b4c4+b5c4	1.731	**-6.082**	**0.366**	**0.157**	13.714
b2c2+b3c4+b4c4	**1.730**	-5.530	0.379	0.186	14.857

6.6. Number of frames

Since the range-Doppler map is always collected dynamically, that is, the continuous frames will have some relationship and the model could get more motion information. In this thesis, everything discussed previously is based on the setting that the number of frames in the inputs is 4. In this section, the effects of different numbers of frames values on the model will be evaluated. Table 6.14 and Figure 6.14 illustrate the difference between the two cases.
From the evaluation losses from different cases, when the number of frames is chosen as 4, it can achieve a better result. Therefore, the continuous motion information is useful for the model and it verifies the previous assumption as well.

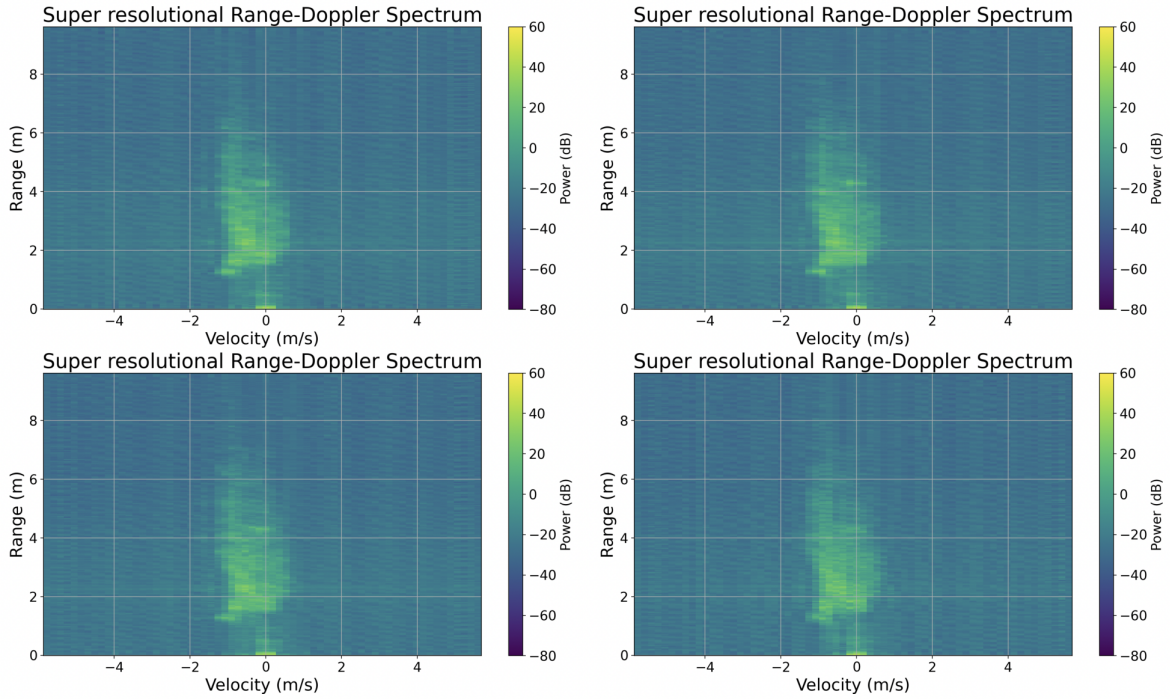
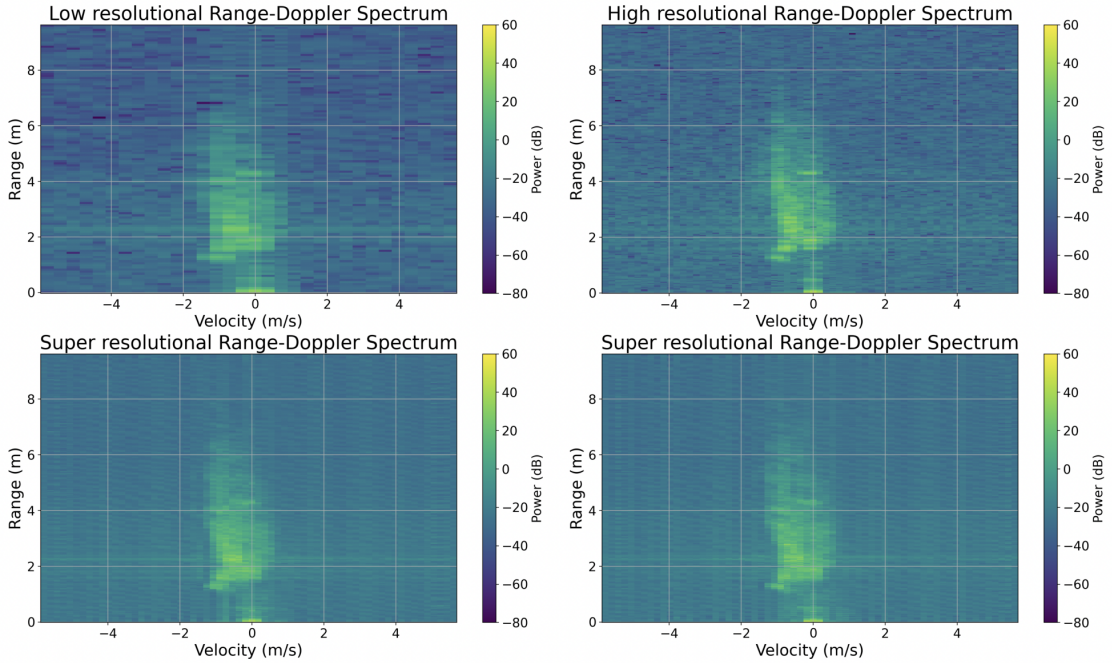


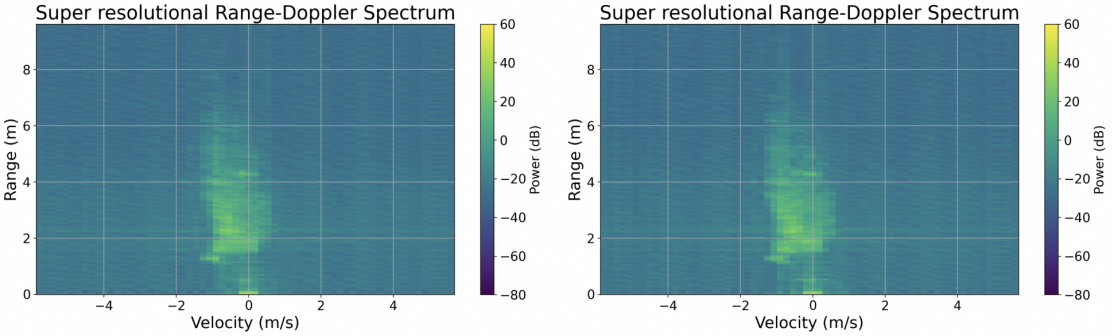
Figure 6.13.: Super-resolution range-Doppler maps of different combinations of the VGG layers in the training loss combination, in the first row from left to right as b1c2+b2c2+b3c4 and b1c2+b2c2+b3c4+b4c4, in the second row as b1c2+b2c2+b3c4+b4c4+b5c4 and b2c2+b3c4+b4c4.

Table 6.14.: Evaluation losses of different numbers of frames values.

#Frames comparison	MSE	SDR	LSD	WMSE	Perceptual
#Frames = 1					
DP	**1.379**	-5.239	0.381	0.191	12.778
cGAN	1.950	-3.930	0.409	0.236	13.063
#Frames = 4					
DP	1.731	**-5.375**	**0.380**	**0.179**	**12.320**
cGAN	**1.640**	**-5.690**	**0.374**	**0.169**	**11.881**



(a) Super-resolution range-Doppler maps with one frame as input, where in the second row the left from DP-TF Transformer model and the right from cGAN model.



(b) Super-resolution range-Doppler maps with four frames as input, where the left from DP-TF Transformer model and the right from cGAN model.

Figure 6.14.: Super-resolution range-Doppler maps of different numbers of frames

6.7. Resampling rate

With the optimal processing methods and loss combination, the case that the 4× downsampled low-resolution range-Doppler map is evaluated with both DP-TF Transformer model and cGAN model in this section. Table 6.15 and Figure 6.15 show the results. However, the evaluation losses as well as the super-resolution range-Doppler maps look not as good as the case of 2× downsampled low-resolution range-Doppler map. The reason is that too much information is lost, only the shape of the part with high signal power can be seen.

Table 6.15.: Evaluation losses while the resampling rate as 4.

Resampling rate	MSE	SDR	LSD	WMSE	Perceptual
DP	2.208	5.811	0.620	0.491	22.347
cGAN	2.307	5.166	0.600	0.429	22.162

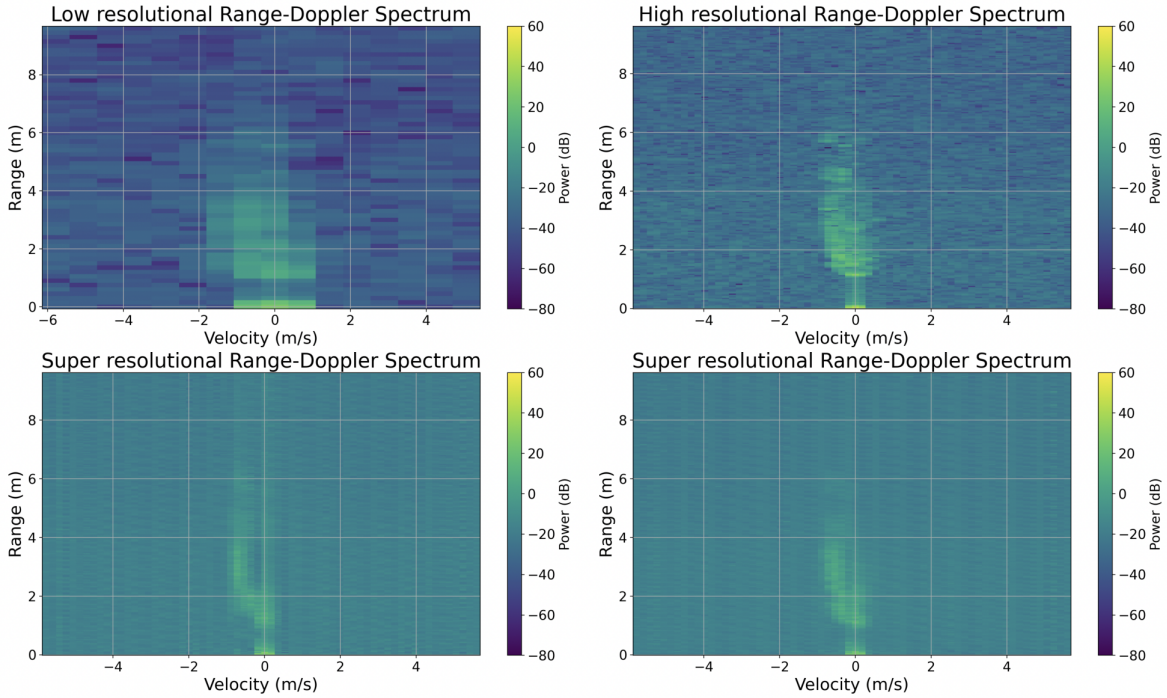


Figure 6.15.: Super-resolution range-Doppler maps while the resampling rate as 4, where in the second row the left from DP-TF Transformer model and the right from cGAN model.

6.8. Distributed training speed

To speed up the training process, the distributed training is used in the pipeline. The speed comparison is illustrated in Figure 6.16, which is recorded as the distributed training was successfully implemented, where DP-TF Transformer model is used in a large scale. The time taken in each epoch decreases from roughly 1800 seconds to under 400 seconds, that is, it uses multiple GPUs successfully.

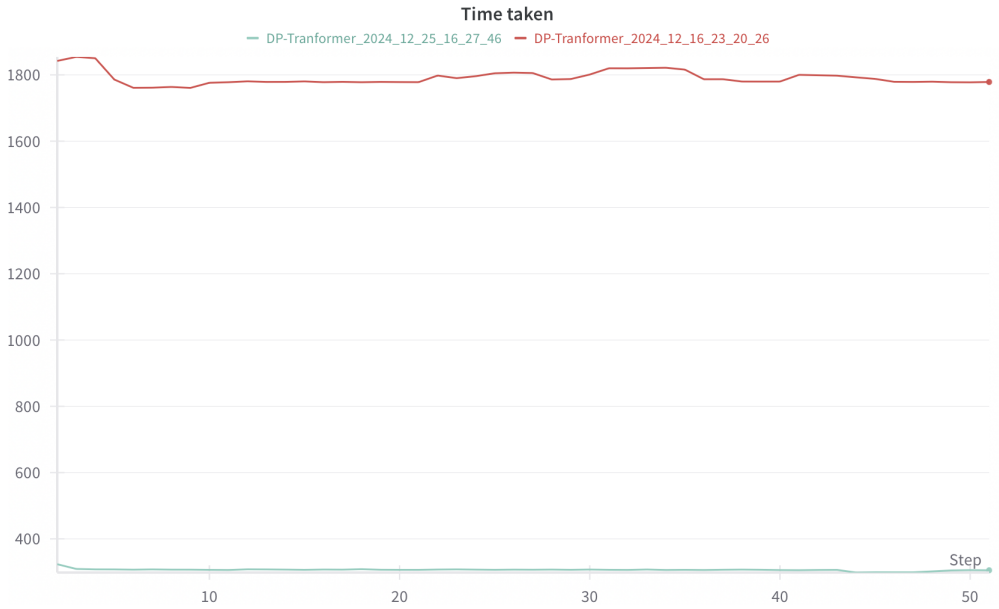


Figure 6.16.: Evaluation of the speed by the distributed training, where the red line represents the pipeline without distributed training while cyan line with.

6.9. Hyperparameters tuning

The hyperparameters of the model and training process have an impact on the result, such as the number of blocks, the number of filters, etc. Therefore, in this section, the sweep function in WandB is used to combine and evaluate some hyperparameters to find a better combination that minimizes the loss. Figure 6.17 shows the sweep process of WandB. With optimal combination of the hyperparameters, the evaluation losses and the visual effect of the super-resolution range-Doppler maps have improved as shown in Table 6.16 and Figure 6.18.

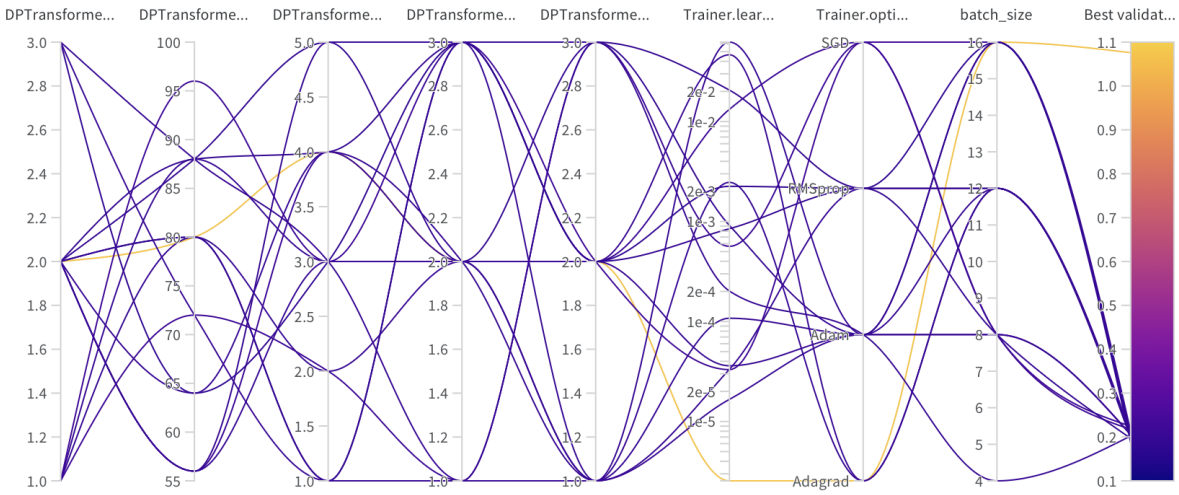


Figure 6.17.: Hyperparameter tuning by WandB

Table 6.16.: Evaluation losses after hyperparameters tuning

Hyperparameters tuning	MSE	SDR	LSD	WMSE	Perceptual
DP	1.225	-5.625	0.373	0.146	11.753
cGAN	1.223	-5.187	0.382	0.157	10.915

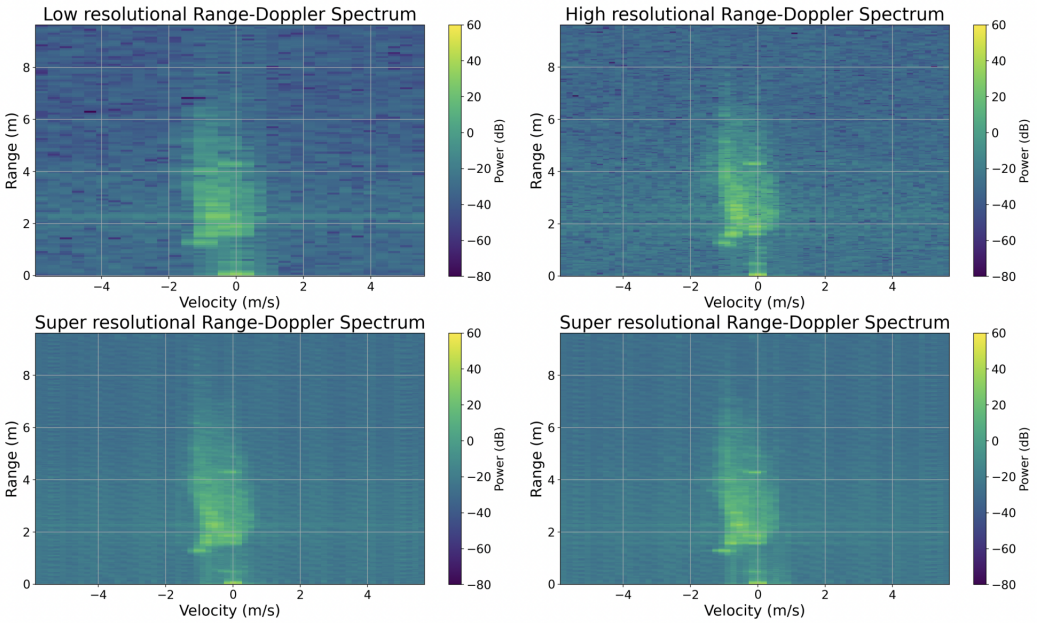


Figure 6.18.: Super-resolution range-Doppler maps after the hyperparameters tuning, in the second row left on trained by DP-TF Transformer model while right one trained by cGAN model.

6.10. Final result

Keeping the same tuned hyperparameters as evaluated in section 6.9, after running more epochs with the large model as well as the whole dataset, the evaluation losses and super-resolution range-Doppler maps are further improved as shown in Table 6.17 and Figure 6.19.

Table 6.17.: Evaluation losses with large models and whole dataset, where the number of parameters of the DP-TF Transformer model and the generator is 329,410 while that of the discriminator is still 766,187.

Final result	MSE	SDR	LSD	WMSE	Perceptual
DP	0.968	-6.008	0.363	0.123	9.079
cGAN	0.803	-5.619	0.371	0.120	8.740

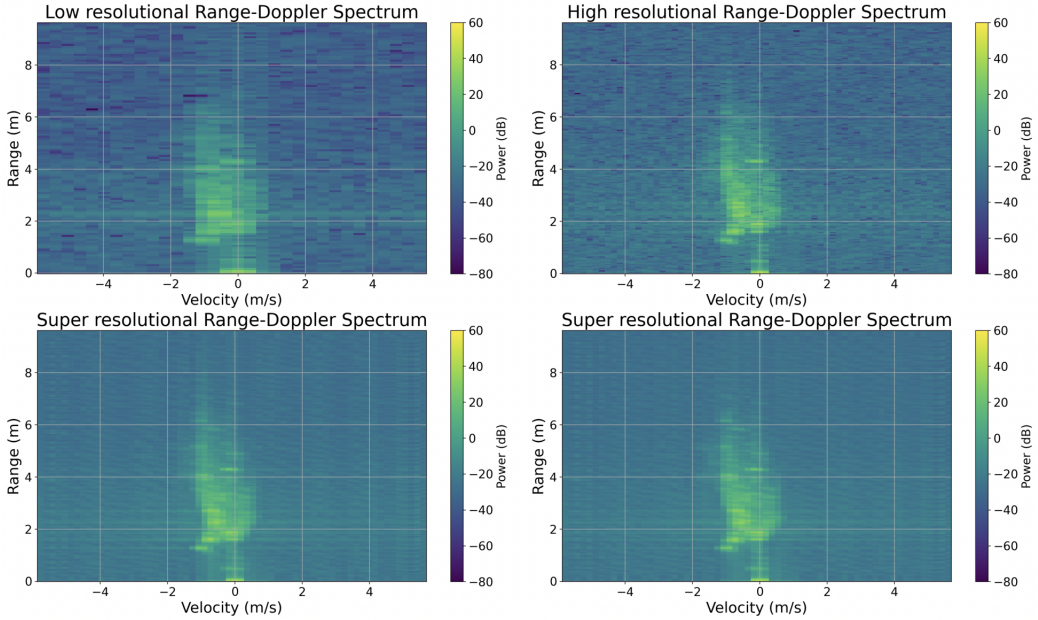


Figure 6.19.: Super-resolution range-Doppler maps trained by large models and whole dataset, in the second row left one trained by DP-TF Transformer model while right one trained by cGAN model.

7. Summary & Outlook

This chapter summarizes the results and improvements of the range-Doppler map upsampling task and give an outlook on the future research.

7.1. Summary

In summary, the thesis uses the combination of multiple data processing methods, models, and loss functions to upsample the range-Doppler map with lower resolution. The range-Doppler map is collected in the indoor environment, that is, it has more information and the dynamic range of the amplitude is wide, which poses the high requirement on the training process. However, through a series of optimizations, the final result is significantly improved compared to the general image upsampling approaches.
Firstly, the motivation of this task is introduced. Although chirp sequence radar can achieve high resolution, it cannot achieve the ideal resolution in practice due to the influence of the whole system or regulation. However, the range-Doppler map is an important part of the subsequent positioning and other operations, so a super-resolution one is needed. Meanwhile, we introduce some state-of-the-art upsampling models and papers.
Since the collection scenarios of many public datasets about the range-Doppler map are not sufficient to cope with complex environment, we choose to collect them by ourselves. Using the radar device of Infineon to dynamically collect data in the corridor, the range-Doppler map will contain both range and velocity information, and can ensure the complexity. Furthermore, by loading the data into the TFRecord files to speed up the training process, the data is also processed, such as downsampling, loading it with the certain number of frames, and converting it from the time domain to the frequency domain using RDP. Moreover, three methods for preprocessing the data are introduced, namely the input data representation on the complex-valued data in frequency domain, logarithm and normalization operation. According to the comparison and evaluation in section 6.2, we choose the amplitude and phase representation type, logarithm and normalization operations in the range of (0, 1) on the amplitude, but no additional operations are performed on the phase. The resolution and maximum values of the range and velocity are derived as well, which are necessary for the visualization of the range-Doppler map.
According to the image upsampling models in the state of the art introduced above, we built seven models, namely the basic CNN model, the basic UNet model, the UNet model with concatenating structure, three Transformer models according to different data division methods, and the cGAN model. Through the two times comparisons in section 6.1, the DP-TF Transformer model and SwinIR+DP model look better in evaluation loss and super-resolution range-Doppler maps, especially the DP-TF Transformer model. The reason is that division along the dimensions of range and velocity is conducive for the Transformer model to better learn the relationship between them. In addition, affected by RDP, there are prime numbers in the range dimension, so a variety of processing methods are introduced, among

which the padding method was determined according to the evaluation in section 6.2. For the upsampling layer, we also have two methods. In order to reduce the training complexity and ensure the training results, the pixel shuffle approach is selected as the upsampling type. Subsequently, the above optimal solution is combined with cGAN. According to the results obtained in section 6.4, the evaluation losses and the visual effect of the super-resolution range-Doppler map have been optimized to a certain extent through the supervision of the discriminator.
In order to effectively train the upsampling model based on range-Doppler map, a variety of loss functions are used, namely MSE, SDR, LSD, WMSE, PLSD and VGG perceptual loss functions. Through the comparison in section 6.3, LSD achieved a better effect. In order to improve its visual effect, it is combined with perceptual loss, and the ratio is 1:0.5. Furthermore, due to the instability of perceptual loss in the early steps, only LSD loss will be used for pretraining in the first 50 epochs. In the subsequent combination with cGAN, the adversrial loss will be combined. According to the evaluation results in section 6.4, 1:0.5:1e-2 is a relatively good loss ratio.
In addition, we also use a series of methods such as distributed training, learning rate schedule, early stopping, and using WandB for hyperparameters tuning to improve the training efficiency and final effect of the entire pipeline, which has been verified in section 6.8, 6.9 and 6.10, respectively.

7.2. Outlook

At the end, by combining multiple data processing methods and loss functions, the DP-TF Transformer model has made significant progress in upsampling the range-Doppler map in complex indoor environment. However, there are still some possible improvements in the future research.
The content and scale of the currently collected dataset can be further enriched, such as adding more environments. The current data is collected in the corridor of different floors, and the environment is relatively simple. It can be collected in different places such as laboratories, cafes, or subway stations. Although there are other students or scholars passing by in the corridor, areas with more people such as subway stations and cafes will bring more dynamic information. In addition, although there are two types of tempo in the dataset, the overall difference is not large. An alternative is collecting the data while running to make the range-Doppler map get a larger value in the velocity dimension.
For the models based on the Transformer, we can combine the ideas of Swin Transformer and DP-TF Transformer blocks. Swin Transformer divides the range-Doppler map into different small patches and windows, while DP-TF Transformer block divides the data into a column or a row along the dimensions of range or velocity. Therefore, we can try to divide data into different widths along two dimensions in DP-TF Transformer block and shift the window between them to see if the results can be improved. Furthermore, since the amplitude has been applied the logarithm and normalization, that is, the range of the amplitude is between 0 and 1, some activation function in the last convolutional layer of the decoder could be used to limit the output.
The selection of many hyperparameters can still be optimized. For example, the relationship between the amplitude loss and the phase loss in each loss function is currently determined based on a rough estimate of the results obtained in the first few batches to ensure that the

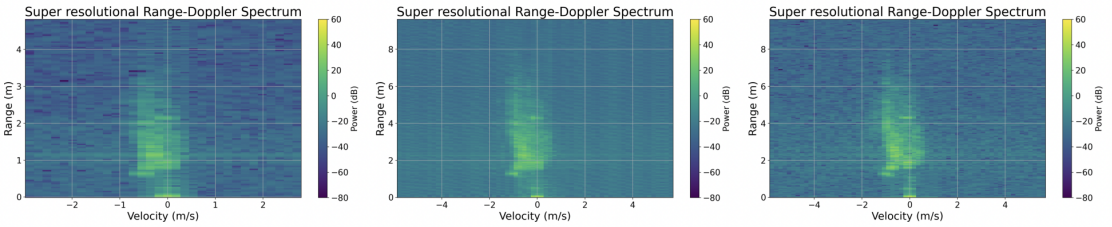
amplitude accounts for the larger proportion. In further research, different ratios can be tried to improve the selection of this hyperparameter. Meanwhile, although a series of parameters λ_c have been compared, more fine-tuning can still be done in the combination ratio between the LSD and perceptual loss functions as well as with adversarial loss.
Furthermore, since there are a large number of different settings and combinations in this thesis, in order to make it clear, the evaluation process is currently carried out step by step, that is, the next setting proceed in the case that the last setting is determined, but this may cause some potentially better combinations to be ignored. More combinations could be tried in future research.
In addition, the perceptual loss can be calculated by the features not only obtained by the VGG model, but also some other better models or other masking method such as CFAR. Meanwhile, more loss functions can be chosen to train the model as well.
Last but not least, although many hyperparameters in the model have been tuned, due to resource limit, many parameter ranges are not set very large to prevent the occurrence of out of memory situations. In the future, a relatively larger model or richer alternatives of the hyperparameters could be run on a server with stronger GPUs.

A. Appendix

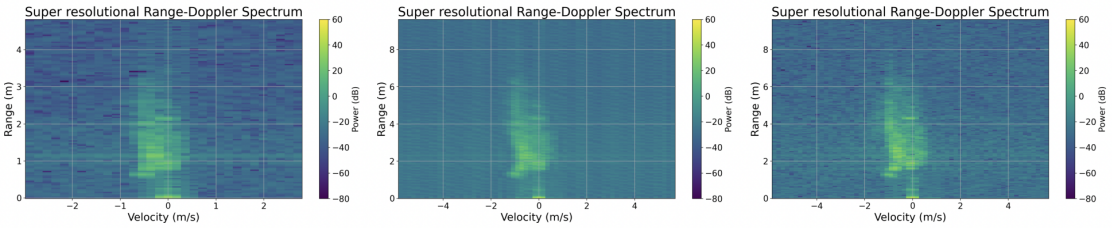
Following Table A.1 as well as Figure A.1 and A.2 show the evaluation of the difference mentioned in the section 3.8.

Table A.1.: Evaluation losses of the differences between architectures and Transformer blocks

Transformer difference	#Params	MSE	SDR	LSD	WMSE	Perceptual
DP	48,042	1.731	-5.375	0.380	0.179	12.320
SwinIR+DP	30,562	1.662	-6.415	0.360	0.128	16.903
Condition1	84,258	1.562	-5.784	0.372	0.155	12.866
Condition2	84,130	1.490	-5.367	0.380	0.363	13.102
Condition3	47,074	1.779	-5.333	0.382	0.176	13.343
Condition4	47,074	1.364	-4.986	0.388	0.170	13.436
Condition5	42,786	1.671	-5.215	0.384	0.164	14.410
Condition6	34,466	2.040	-5.181	0.387	0.206	15.945
Condition7	30,178	4.415	-5.718	0.377	0.272	17.011
Condition8	30,178	1.435	-6.523	0.359	0.150	15.112
Condition9	30,178	1.568	-5.805	0.373	0.166	14.807
Condition10	30.178	2.026	-5.980	0.369	0.169	15.887
Condition11	30.178	5.004	-7.837	0.338	0.256	20.567
Condition12	30.178	3.474	-7.513	0.343	0.265	19.121

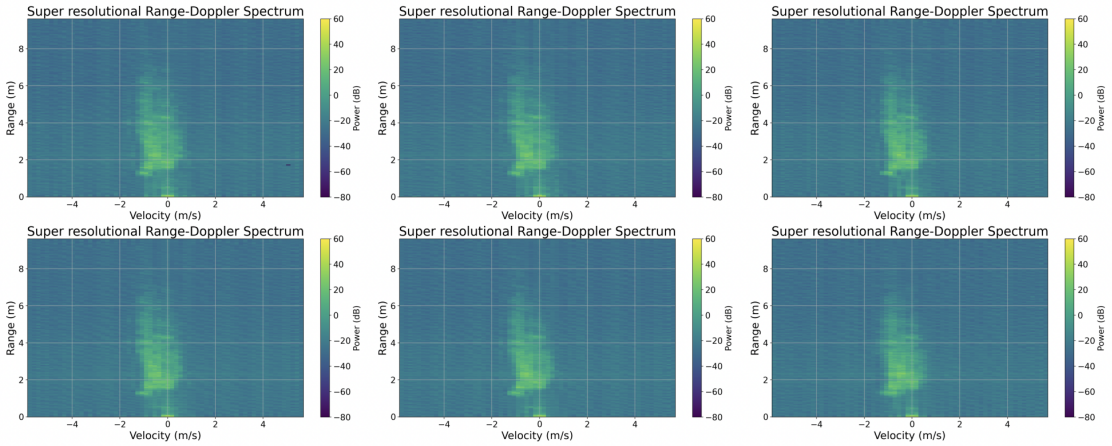


(a) Super-resolution range-Doppler map from the DP-TF Transformer model

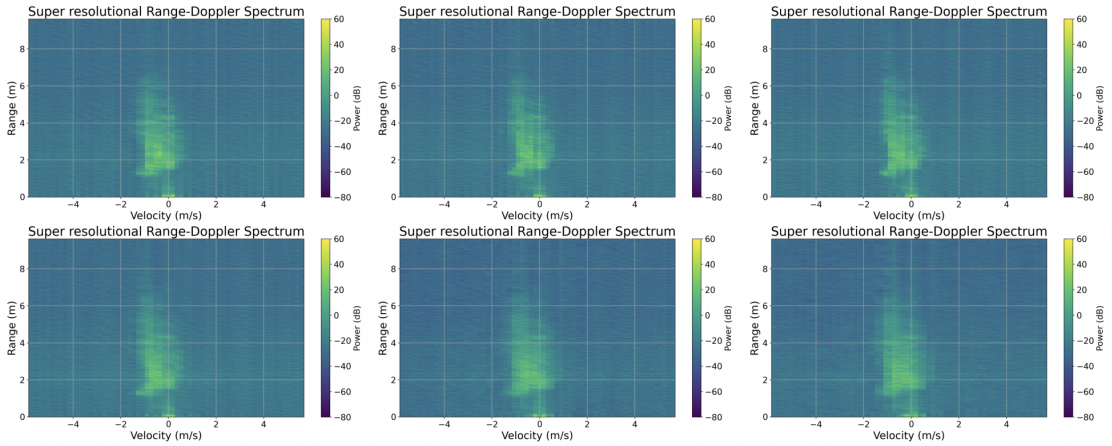


(b) Super-resolution range-Doppler map from the SwinIR+DP model

Figure A.1.: Super-resolution range-Doppler maps from both models



(a) Super-resolution range-Doppler maps as the condition 1 - 6 subsequently as true



(b) Super-resolution range-Doppler maps as the condition 7 - 12 subsequently as true

Figure A.2.: Super-resolution range-Doppler maps from the conditions subsequently as true

List of Figures

List of Tables

Bibliography

[1] Karsten Thurn et al. “Concept and Implementation of a PLL-Controlled Interlaced Chirp Sequence Radar for Optimized Range–Doppler Measurements”. In: *IEEE Transactions on Microwave Theory and Techniques* 64 (2016), pp. 3280–3289.

[2] Johannes Fink and Friedrich K. Jondral. “Comparison of OFDM radar and chirp sequence radar”. In: *2015 16th International Radar Symposium (IRS)*. IEEE, 2015, pp. 315–320. URL: http://ieeexplore.ieee.org/document/7226369/.

[3] Infineon Technologies AG. *BGT60TR13C | XENSIV 60GHz radar sensor for advanced sensing - infineon technologies*. URL: https://www.infineon.com/cms/en/product/sensor/radar-sensors/radar-sensors-for-iot/60ghz-radar/bgt60tr13c/.

[4] Zhihao Wang, Jian Chen, and Steven C. H. Hoi. “Deep learning for image super-resolution: A survey”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43 (2021), pp. 3365–3387. URL: https://ieeexplore.ieee.org/document/9044873/.

[5] Alexander Watson. “Deep learning techniques for super-resolution in video games”. In: *arXiv preprint arXiv:2012.09810* (2020).

[6] Ashish Vaswani et al. *Attention is all you need*. 2023. URL: http://arxiv.org/abs/1706.03762.

[7] Ze Liu et al. *Swin transformer: Hierarchical vision transformer using shifted windows*. 2021. URL: http://arxiv.org/abs/2103.14030.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-net: Convolutional networks for biomedical image segmentation*. 2015. URL: http://arxiv.org/abs/1505.04597.

[9] Akarsh Prabhakara et al. “High resolution point clouds from mmWave radar”. In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 4135–4142. URL: https://ieeexplore.ieee.org/document/10161429/.

[10] Yu-Jhe Li et al. “Azimuth super-resolution for FMCW radar in autonomous driving”. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2023, pp. 17504–17513. URL: https://ieeexplore.ieee.org/document/10205026/.

[11] Jingyun Liang et al. *SwinIR: Image restoration using swin transformer*. 2021. URL: http://arxiv.org/abs/2108.10257.

[12] Sven Hinderer. “Blind source separation of radar signals in time domain using deep learning”. In: *2022 23rd International Radar Symposium (IRS)*. IEEE, 2022, pp. 486–491. URL: https://ieeexplore.ieee.org/document/9904990/.

[13] Ian J. Goodfellow et al. *Generative adversarial networks*. arXiv:1406.2661 [stat]. 2014. URL: http://arxiv.org/abs/1406.2661.

[14] Mehdi Mirza and Simon Osindero. *Conditional generative adversarial nets*. arXiv:1411.1784 [cs]. 2014. URL: http://arxiv.org/abs/1411.1784.

[15] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. URL: http://arxiv.org/abs/1611.07004.

[16] Karim Armanious et al. "An adversarial super-resolution remedy for radar design trade-offs". In: *2019 27th European Signal Processing Conference (EUSIPCO)*. arXiv, 2019, pp. 1–5. URL: http://arxiv.org/abs/1903.01392.

[17] Yingying Kong and Si Liu. "DMSC-GAN: A c-GAN-based framework for super-resolution reconstruction of SAR images". In: *Remote Sensing* 16 (2023), p. 50. URL: https://www.mdpi.com/2072-4292/16/1/50.

[18] Sherif Abdulatif, Ruizhe Cao, and Bin Yang. "CMGAN: Conformer-based metric-GAN for monaural speech enhancement". In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 32 (2024), pp. 2477–2493. URL: https://ieeexplore.ieee.org/document/10508391/.

[19] Taewon Jeong and Seongwook Lee. "Resource-efficient range-doppler map generation using deep learning network for automotive radar systems". In: *IEEE Access* 11 (2023), pp. 55965–55977. URL: https://ieeexplore.ieee.org/document/10143630/.

[20] Christian Ledig et al. *Photo-realistic single image super-resolution using a generative adversarial network*. 2017. URL: http://arxiv.org/abs/1609.04802.

[21] *How to build Radar SDK from source code*. 2023. URL: https://community.infineon.com/t5/Knowledge-Base-Articles/How-to-build-Radar-SDK-from-source-code/ta-p/663602.

[22] H. Sorensen et al. "Real-valued fast fourier transform algorithms". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35.6 (1987), pp. 849–863. URL: http://ieeexplore.ieee.org/document/1165220/.

[23] Prajoy Podder et al. "Comparative performance analysis of hamming, hanning and blackman window". In: *International Journal of Computer Applications* 96 (2014), pp. 1–7.

[24] vdumoulin. *vdumoulin/conv_arithmetic*. 2016. URL: https://github.com/vdumoulin/conv_arithmetic.

[25] Wenzhe Shi et al. *Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network*. 2016. URL: http://arxiv.org/abs/1609.05158.

[26] Wenzhe Shi et al. *Is the deconvolution layer the same as a convolutional layer?* 2016. URL: https://arxiv.org/abs/1609.07009.

[27] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and checkerboard artifacts". In: *Distill* 1 (2016), e3. URL: http://distill.pub/2016/deconv-checkerboard.

[28] *Bilinear interpolation*. 2025. URL: https://en.wikipedia.org/w/index.php?title=Bilinear_interpolation&oldid=1276345325.

[29] Keras Team. *Keras documentation: Conditional GAN*. URL: https://keras.io/examples/generative/conditional_gan/.

[30] Jonathan Le Roux et al. *SDR - half-baked or well done?* 2018. URL: http://arxiv.org/abs/1811.02508.

[31] Sebastian Braun and Ivan Tashev. *A consolidated view of loss functions for supervised deep learning-based speech enhancement*. 2020. URL: http://arxiv.org/abs/2009.12286.

[32] Karen Simonyan and Andrew Zisserman. *Very deep convolutional networks for large-scale image recognition*. 2015. URL: http://arxiv.org/abs/1409.1556.

[33] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. *Perceptual losses for real-time style transfer and super-resolution*. 2016. URL: http://arxiv.org/abs/1603.08155.

[34] *Distributed training with TensorFlow | TensorFlow core*. URL: https://www.tensorflow.org/guide/distributed_training.

[35] *Weights & biases*. 2025. URL: httpss://wandb.ai/site.

[36] Diederik P. Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. 2017. URL: http://arxiv.org/abs/1412.6980.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 31.03.2025

Zheming Yin