Joint Master's in Applied Geophysics

IDEA League

Delft University of Technology, The Netherlands
ETH Zürich, Switzerland
RWTH Aachen, Germany

Master Thesis

Low-to-high-quality seismic image translation using deep learning

written by

Ying Ting (Rhea) Lau

External Supervisors from SLB UK	**Internal Supervisor from ETH Zürich**
Dr. Rajiv Kumar Dr. Massimiliano Vassallo	Dr. Dirk-Jan van Manen

Academic year 2023/2024
August 5, 2024

Abstract

Is there a way do high-quality seismic interpolation fast and cheap? This thesis introduces a novel approach for translating low-to-high-quality seismic images using deep learning, employing three distinct models: Generative Adversarial Network (GAN), a modified Swin Transformer Convolutional Residual Network (SCRN) as developed by Gao et al. (2024), and their hybrid variant, the Swin Transformer Convolutional Residual Generative Adversarial Network (SCR-GAN). The model hyper-parameters are optimised through extensive analysis.

The two-step workflow is as follows: 1) Train the neural network model with paired images: standard-resolution images obtained from low-cost processing and high-quality images from high-cost processing. 2) Input the unseen low-cost standard-resolution datasets into the trained model to transform them into high-resolution counterparts. This workflow is pioneering in producing high-resolution output from a low-resolution processing sequence in an end-to-end manner while mitigating the computational bottleneck of high-resolution technology.

To demonstrate the feasibility of translating low-to-high-quality seismic images with deep learning, we used interpolation as an example. Matching Pursuit Fourier Interpolation (MPFI) and Time Domain Radon Interpolation (TDRI) are commonly used interpolators in the industry. TDRI, which applies basis functions in a higher-dimensional tau-p domain, provides higher accuracy but at a significantly higher computational cost—127,294 CPU hours to process 3700 km^2 areal data, compared to 13 CPU hours with MPFI. Our focus is on translating MPFI images (standard quality) to pseudo-TDRI images (state-of-the-art quality) using the two-step workflow. The resemblance between the deep-learning-generated pseudo-TDRI images and ground truth is assessed quantitatively and qualitatively to identify the best model. GAN, SCRN, and SCR-GAN achieve SNR values of 15.19 dB, 16.32 dB, and 16.25 dB, respectively, on reconstructions using unseen datasets. SCRN outperforms the others in terms of image resemblance and de-noising capability, even exceeding TDRI. Training SCRN requires 23.42 GPU hours, with translation taking just 11 GPU minutes. Once the model is trained, translating MPFI to TDRI quality requires less than 14 CPU hours, which is nearly four orders of magnitude faster than the traditional TDRI process.

Keyword: Seismic interpolation, image-to-image translation, generative adversarial networks (GANs), vision transformer (ViT), Swin transformer

Acknowledgements

I would like to express my sincere gratitude for the opportunity to undertake this internship at SLB Crawley, where I had the chance to work on this exciting deep learning project. I am especially grateful for the invaluable support and wisdom from the industry shared by my supervisors, Dr. Rajiv Kumar and Dr. Massimiliano Vassallo. I never imagined that I could contribute to two provisional patents and present my work at an internal conference. Their kindness and generosity were instrumental in these achievements.

I am also deeply grateful to my academic supervisor, Dr. Dirk-Jan van Manen, for his prompt and constructive feedback on my thesis.

A special thanks goes to my friends Anson, Ryan, Sargun, Karla, Ashley, and Winky for their enormous support through countless days and nights, especially during challenging times. Additionally, I would like to thank Konsta and my parents for their unwavering faith and constant support.

ETH Zurich Rhea Lau
August 5, 2024

Contents

CHAPTER 1

Introduction

Since Krizhevsky, Sutskever, and Hinton (2012) introduced the first deep Convolutional Neural Networks (CNNs) for image classification, deep learning methodologies, especially those employing CNNs, have increasingly dominated computer vision tasks. Goodfellow et al. (2014) established the framework of Generative Adversarial Networks (GANs), comprising a generator and a discriminator that are trained concurrently through adversarial processes, with the generator producing images and the discriminator evaluating their authenticity. CNNs are integral parts of GAN architectures for feature extractions. Isola, Zhu, Zhou, and Efros (2017) examined conditional adversarial networks (cGANs) as a versatile solution for image-to-image translation challenges, while Ledig et al. (2017) specifically applied GANs to enhance image resolution from low to high. The introduction of Transformers during the same period introduced a new class of deep learning architectures. Unlike CNNs, which capture local features, Transformers employ a self-attention mechanism that enables the modelling of long-range dependencies (Vaswani et al., 2017). Dosovitskiy et al. (2020) defined Vision Transformers (ViTs) tailored for computer vision applications, and Z. Liu et al. (2021) refined this with the Swin Transformer, a shift-window variant. Both architectures demonstrate capabilities in image processing tasks, including translation. Leveraging the strengths of both architectures, Torbunov et al. (2023) proposed UVCGAN, a general-purpose architecture that integrates ViTs and GANs for effective image-to-image translation.

These deep learning image-translation methods can be applied in the seismic context to make seismic processing more affordable and accurate, in this thesis we will focus particularly on seismic interpolation. In seismic acquisition, sampling data at the Nyquist rate $f_{Nyq} = \frac{1}{2\Delta t}$ is challenging due to high surveying and time costs, thus resulting in aliasing. To address this, seismic interpolation is used to spatially transform irregularly sampled traces to a desired grid before further processing (Claerbout, 1976). Seismic interpolation methods fall into two categories: wave-equation-based and signal processing methods.

- Wave-equation methods rely on seismic wave propagation and require a velocity model to interpret traces, allowing interpolation through complex geological structures and subsurface heterogeneities, recovering parabolic and hyperbolic events (Ronen, 1987). However, these methods need accurate velocity models,

which may not be available, and their straight-raypath approximation can fail for low-frequency waves (Stolt, 2002).

- Signal processing methods for seismic interpolation include adaptive domain transform and prediction-error filter methods, which do not require subsurface information. Studies have demonstrated their applications: F-X domain interpolation for regularly sampled data (Spitz, 1991); F-K domain interpolation and minimum weighted norm interpolation for regularly sampled data with optional missing traces (Naghizadeh, 2012; B. Liu & Sacchi, 2004); and matching pursuit Fourier interpolation (MPFI) and time domain Radon interpolation (TDRI), methods this thesis focusing on, for irregularly sampled data (Schonewille, Yan, Bayly, & Bisley, 2013; Schonewille, Dishberger, & Kapadia, 2014). MPFI and TDRI excel by assuming sparsity in higher-dimensional spaces, enabling the reconstruction of highly curved and dispersed events that cannot be achieved with the assumptions of linearity and sparsity in the TX domain. TDRI achieves higher accuracy and incurs more computational cost than MPFI because TDRI's basis functions operate in the two-dimensional τp domain, while MPFI iterates in the one-dimensional k_x domain for each frequency.

Deep learning methods, such as CNNs, GANs, and transformers, do not require velocity models and provide faster reconstruction (Kaur, Pham, & Fomel, 2019, 2021; Gao, Shen, & Min, 2024). CNNs combine adaptive domain transforms and prediction-error filter methods without linearity or sparsity assumptions, by learning and applying transforms and filters through iterative training (Oliveira, Ferreira, Silva, & Brazil, 2018). Gan, Wang, Chen, Zhang, and Jin (2015), Oliveira et al. (2018), and Kaur et al. (2019) have explored the application of GANs for interpolating missing seismic traces, demonstrating its effectiveness in reconstructing data where conventional methods might fall short. Additionally, more recent studies by Guo, Fu, and Li (2023) and Gao et al. (2024) have investigated the use of transformers for the same purpose, by using transformers to capture long-range dependencies and contextual information to accurately fill in missing seismic traces, While these methods can be applied to 3D data, they require large amounts of training data (Khosro Anjom, Vaccarino, & Socco, 2023). Figure 1.1 illustrates the various algorithms used for seismic trace interpolation and processing.

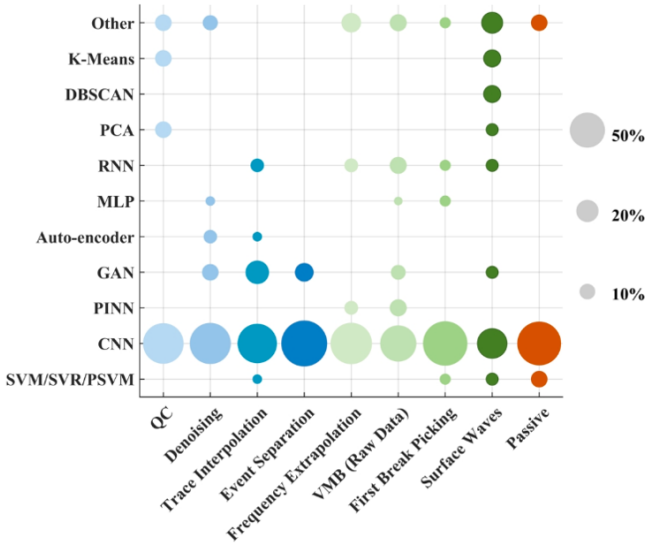


Figure 1.1: *Deep learning algorithms used for seismic processing applications (taken from Khosro Anjom et al. (2023)).*

Our aim in this thesis is to combine the strengths of deep learning networks with conventional seismic pre-processing technologies to achieve optimal results to get the best of both worlds. Returning to our focus on low-to-high-quality seismic image translation, we aim to enhance seismic images quality using deep learning, specifically with GANs and ViT. The two-step workflow is as follows (illustrated in Figure 1.2):

1. Train the neural network with paired images: standard-resolution images obtained from low-cost processing and high-quality images from high-cost processing.

2. Input the unseen low-cost standard-resolution datasets into the trained model to transform them into high-resolution counterparts.

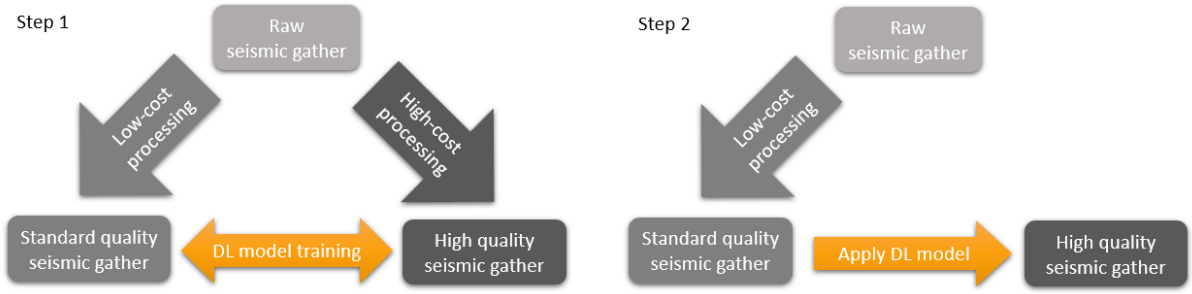


Figure 1.2: *2-Step deep learning workflow for low-to-high-quality seismic image translation.*

To demonstrate the feasibility of low-to-high-quality image translation with deep learning, we will use interpolation as an example, converting low-cost standard-resolution MPFI datasets into high-cost high-resolution TDRI solutions (see figure 1.3). Unlike previous deep learning methods for seismic interpolation that are applied on on raw traces, we are interested in transforming low-quality, conventionally interpolated traces into high-quality traces interpolated by expensive methods.

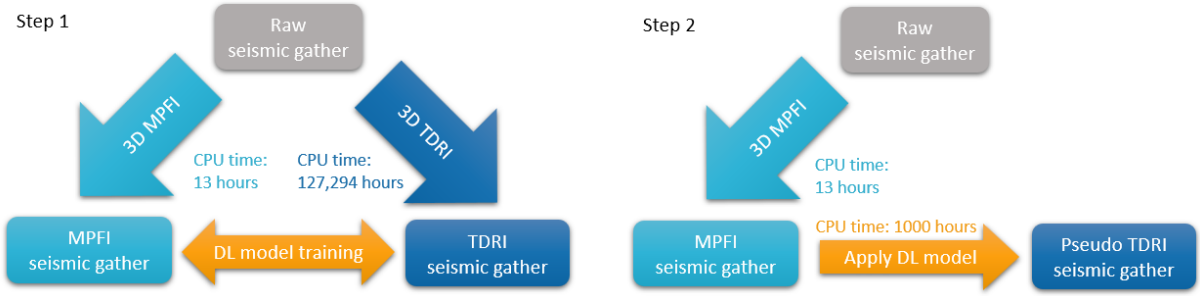


Figure 1.3: *2-Step deep learning workflow for MPFI-to-TDRI seismic image translation.*

1.1 Scope of study

Is there a way do high-quality seismic interpolation fast and cheap? In this thesis, we present a novel approach for translating low-to-high-quality seismic images using deep learning, specifically employing three different models: Generative Adversarial Network (GAN), Swin Transformer Convolutional Residual Network (SCRN),

and their hybrid variant, Swin Transformer Convolutional Residual Generative Adversarial Network (SCR-GAN). We have carefully selected the models' hyper-parameters through rounds of model optimisation testing to ensure optimal performance. We demonstrate the feasibility of this quality enhancement method by translating MPFI images (industrial standard quality) into their pseudo-TDRI counterparts (state-of-the-art quality) as outlined in Figure 1.3. The resemblance between the pseudo-TDRI images generated by deep learning and their ground truth will be evaluated quantitatively and qualitatively to identify the best model. This workflow is the first of its kind to produce high-resolution outputs from a low-resolution processing sequence in an end-to-end manner, effectively mitigating the computational bottleneck associated with high-resolution technology. **In recognition of its novelty and potential impact, we have filed two provisional patents for this technology during its development: one for the MPFI-TDRI translation purpose and the other for the UDD-MDD application.**

1.2 Structure of thesis

In Chapter 2, we will present both seismic processing theory and deep learning theory. The seismic processing section covers the workings of TDRI and MPFI, while the deep learning theory section introduces key concepts in deep learning and vision transformers, ensuring readers without a computer science background are well-prepared for more advanced topics later on. This is followed by descriptions and architectures of the three deep learning models used in our experiments: GAN, SCRN and SCR-GAN. Chapter 3 details our experimental methodology, beginning with data preparation, followed by model training and validation, model evaluation metrics, and finally, model optimisation for each of our three models. Chapter 4 presents both quantitative and qualitative results of image translation for our three models. Chapter 5 discusses the pros and cons of each model, their computation time taken, and identifies the best model for low-to-high-quality seismic image translation. Chapter 6 summarises the overall findings of the report.

CHAPTER 2

Theory

2.1 Seismic processing theory

In order to reconstruct the seismic signal from the samples collected in the field, it is ideal to follow the Nyquist criterion (Shannon, 1949), which states that the sampling rate f_s must be at least twice the highest frequency f_{max} present in the signal, i.e. $f_s \geq 2f_{max}$. If the criterion is met, we can reconstruct the original band-limited continuous-time signal $x(t)$ from the samples $x(nT)$ using Whittaker-Kotelnikov-Shannon (WKS) theorem (Shannon, 1949):

$$x(t) = \sum_{n=-\infty}^{\infty} x(nT) \cdot \text{sinc}\left(\frac{t - nT}{T}\right) \tag{2.1}$$

Where T is the sampling period ($T = \frac{1}{f_s}$), f_s is the sampling frequency, and $\text{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ is the sinc function. If the Nyquist criterion is not met, frequencies higher than $f_s/2$ (the Nyquist frequency) will alias into in the base bandwidth (the range from 0 Hz to $f_s/2$). These higher frequencies will appear as if they are lower frequencies within the base bandwidth, and they may overlap with the original signal components (see Figure 2.1(d)). This results in high-frequency components being misinterpreted as lower frequencies, leading to distortion in the reconstructed signal (Oppenheim, 1999).

Seismic data are typically irregularly and sparsely collected in spatial sense due to finite surveying and time resources (Schonewille et al., 2013). Therefore, interpolation is necessary to accurately reconstruct the original signal and address aliasing issues. Before processing the dataset, we require an interpolation method for filling gaps in the raw data to ensure data continuity and correcting acquisition artifacts to prepare data in a regular grid (Claerbout, 1976). The ideal interpolation technique should prioritise speed and affordability while maintaining good accuracy. This section discusses two SLB in-house interpolators, matching pursuit Fourier interpolation (MPFI) and time domain Radon interpolation (TDRI).

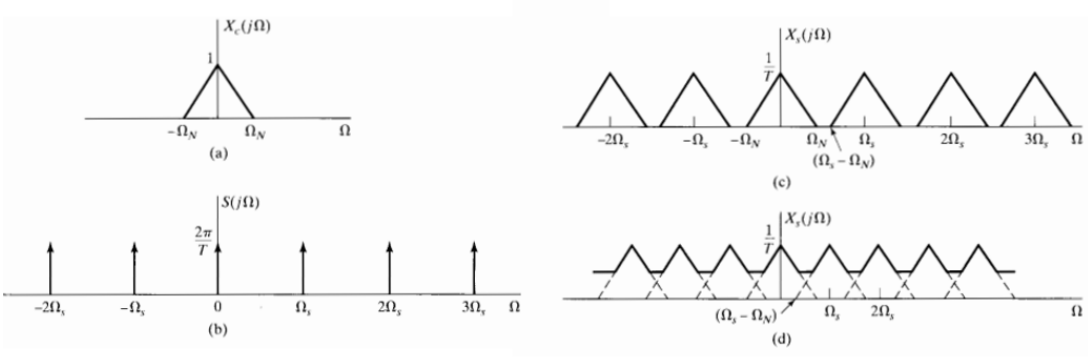


Figure 2.1: *Effect in the frequency domain of sampling in the time domain, where Ω_S denotes the sampling frequency and Ω_N is the Nyquist frequency. (a) Spectrum of the original signal. (b) Spectrum of the sampling function. (c) Spectrum of the sampled signal with $\Omega_S > 2\Omega_N$, no aliasing occurs. (d) Spectrum of the sampled signal with $\Omega_S < 2\Omega_N$, aliasing occurs. Illustration taken from Oppenheim (1999).*

2.1.1 Matching pursuit Fourier interpolation (MPFI)

Matching Pursuit Fourier Interpolation (MPFI) is a method used in signal processing to reconstruct signals in the wavenumber domain. "Fourier" in MPFI refers to the use of Fourier transform-based basis functions, specifically the non-uniform discrete Fourier transform (NDFT) given by Equation 2.4 (Bagchi & Mitra, 1996; S. Xu, Zhang, Pham, & Lambaré, 2005). MPFI integrates principles from Matching Pursuit, the signal to be interpolated is modelled as the sum of a finite number of complex sinusoids, selected from a large and redundant dictionary. MPFI iteratively calculates the wavenumber, amplitude and phase of such sinusoids to best match the data at the every iteration, approximating the overall spectrum. The process described by Schonewille et al. is as follows:

Initialisation:

1. Estimate the FX spectrum $S(f, x)$ of the original irregularly sampled signal $s(t, x)$ using Fast Fourier transform (FFT). $S(f, x)$ could be decomposed into a sum of basis functions:

$$S(f, x) = \sum_{i=1} A_i e^{j(\varphi_i + k_i x)} \tag{2.2}$$

2. Set the residual S_{res} as the transformed original signal S itself and the interpolated signal S_{int} to 0.

For each iteration i:

MPFI looks for three parameters: amplitude A_i, phase φ_i and wavenumber k_i, by identifying the basis function $\text{BF}_i(A_i, \varphi_i, k_i) = A_i e^{j(k_i x + \varphi_i)}$ that best match S_{res} at the current iteration. The cost function to minimise the difference is:

$$[A_i, \varphi_i, k_i] = \min_{A_i, \varphi_i, k_i} [CF] = \min_{A_i, \varphi_i, k_i} \left[\sum_n \left\| S_{\text{res}}(x_n) - \underbrace{A_i e^{j(\varphi_i + k_i x_n)}}_{\text{Basis function}} \right\|^2 \right] \tag{2.3}$$

and can be solved linearly for the amplitude and phase as a function of wavenumber:

$$A_i\,(k_i)\, e^{-j(\varphi_i(k_i))} = \frac{1}{X} \sum_{n=1}^{N_X} S_{\text{res}}\,(x_n)\, e^{-j(k_i x_n)} \Delta x_n \tag{2.4}$$

where X is the total distance in the x direction, N_x is the total number of measurements in the x direction, and Δx_n is the spacing at sample number n. This simplifies the cost function to vary solely with the wavenumber parameter:

$$[k_i] = \min_{k_i}[CF] = \min_{k_i}\left[\sum_n \left\| S_{\text{res}}\left(x_n\right) - A_i\left(k_i\right) e^{j(\varphi_i(k_i)+k_i x_n)} \right\|^2\right] \tag{2.5}$$

1. Equation 2.5 is calculated for every wavenumber k_i in the dictionary. This results in a representation of the spectrum of the residual in the wavenumber domain.

2. The basis function with the optimal k_i that give the smallest residual is selected, which is the one that correspond to the maximum amplitude of the spectrum. This process may involve differentiating between aliased replicas using prior knowledge.

3. This basis function describe a component of our estimated wavefield and can be added to the output at a generic position x:
$$S_{\text{int}}(x) = \sum_i A_i e^{j(\varphi_i+k_i x)} = S_{\text{int},i-1}(x) + A_i e^{j(\varphi_i+k_i x)} \tag{2.6}$$

4. The basis function is then subtracted from the residual $S_{\text{res}} = S_{\text{res},i-1} - \text{BF}_i(A_i, \varphi_i, k_i)$.

Convergence:

1. Continue the iterative process until S_{res} is sufficiently small or a predefined number of iterations is reached. The final interpolated signal (total spectrum) S_{int} is the sum of all selected basis functions (high energy spectral components) obtained repeatedly by Equation 2.6.

2. Transform the S_{int} back to the TX domain using inverse FFT.

MPFI models the signal as a summation of Fourier components, thus performing spatial interpolation in the wavenumber domain. This method is not only dependent on the sparseness of the original signal in the FK domain, but also incorporating any available prior information in the FK domain, such as a more densely sampled secondary dataset for better reconstruction of dipping events or a prior derived from lower frequencies in the same dataset to de-alias higher frequencies (Schonewille et al., 2013). These priors assist in distinguishing the primary energy component from its aliased replicas along the wavenumber axis.

While MPFI offers affordability and computational efficiency, it has limited robustness in reconstructing steeply dipping and low-amplitude events, due to several theoretical factors rooted in the nature of seismic data and the methods employed. First, seismic data is primarily collected in TX domain, where low-amplitude signals (e.g. reflections, refractions and diffractions) are overlaid by the high-amplitude, low-frequency dispersive noises (e.g. ground roll and guided waves). These low-amplitude signals may be inadequately represented when NDFT is applied (Bilsby, Kumar, Vassallo, & Zarkhidze, 2023). In high dynamic range situations, MPFI can generate artifacts due to its focus on matching the strongest events, which can distort weaker signals. Even with optimal interpolation, some errors are unavoidable. If these errors are small compared to a very strong event, they may still overshadow weaker underlying signals.

Moreover, the irregular sampling of seismic data can lead to spectral leakage, where energy from desired signals

spreads into adjacent frequency bins with DFT, further complicating accurate reconstruction. Perfect interpolation in FK domain is impossible for data overlapping condition (Soubaras, 1997). For instance the overlaying signals in Figure 2.2(b) cannot be reconstructed in numerical perfection by the basic MPFI algorithm, and more advanced algorithms or more complex workflows may be needed. While modern technologies can interpolate these three linear events with high accuracy, real field seismic data presents far greater complexity.

Another factor is the effectiveness of MPFI heavily relies on the assumption of signal sparsity in the FK domain (S. Xu, Zhang, & Lambaré, 2010). If the original signal does not exhibit significant sparsity or if the prior information about the signal structure is not sufficiently accurate, MPFI may struggle to accurately identify and reconstruct the relevant components from the noisy data. This limitation becomes more pronounced when dealing with complex geological structures or signal-to-noise ratios are inherently low.

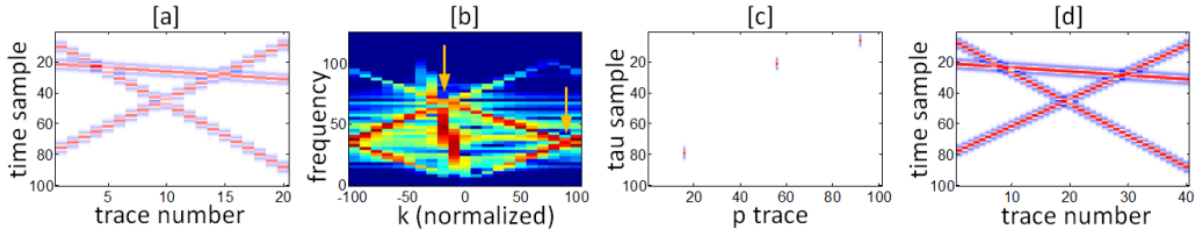


Figure 2.2: *Illustration of synthetic traces with missing data in (a) TX domain, (b) FK domain, (c) τp domain, and (d) the interpolated traces using TDRI. The orange arrows mark overlapping data in the FK domain (taken from Schonewille et al. (2014)).*

2.1.2 Time domain Radon interpolation (TDRI)

Serving a similar functionality as MPFI, Time domain Radon interpolation (TDRI) also operates as an iterative solver but relies on sparsity in the zero-offset intercept time/ dip (τp) domain. τ is the intercept time obtained by projecting the slope back to x = 0, and p is the horizonal ray parameter, also known as the slowness, $p = \frac{1}{c}$ for a 2D vector where c is the velocity (Diebold & Stoffa, 1981). While in 3D space, the slowness p only considers the horizontal projection of the vector. This domain can be considered an intermediary between the time-space (TX) and frequency-space (FX) domains, as it involves a combination of linear moveout corrections and stacking with various slownesses. τp transform is a tool used to isolate linear events in seismic data by transforming the data from the TX domain to the τp domain by Equation 2.7 (Beylkin, 1987). A coherent seismic event, such as a reflection, will have a consistent moveout pattern across multiple traces. By integrating along the lines of constant moveout defined by $t = \tau + px$, the slant stack maps the distributed energy of seismic events in the TX domain into concentrated points in the τp domain (see Figure 2.2(c)). This transformation allows for the separation of signals based on their dip, which is particularly useful for dealing with complex subsurface structures. By applying linear moveout corrections, TDRI aligns seismic events of different dips, effectively enhancing signal coherence and suppressing noise.

Discrete Radon Transform (DRT):

Let a seismogram $u(t,x)$ contain $2L+1$ traces, i.e., we have $u\left(t,x_l\right)$, where $l=0,\pm1,\cdots,\pm L$. Assuming that $x_{-L}<x_{-L+1}<\cdots<x_{L-1}<x_L$, the DRT equation is given by Beylkin (1987) is:

$$(Ru)(\tau,p)=\sum_{l=-L}^{l-L}u\left(\tau+px_l,x_l\right)\Delta x_l \tag{2.7}$$

where $\Delta x_l=\left(x_{l+1}-x_{l-1}\right)/2$ for $l=0,\pm1,\cdots,\pm(L-1)$, and $\Delta x_L=x_L-x_{L-1},\Delta x_{-L}=x_{-L+1}-x_{-L}$.

Inverse Discrete Radon Transform (Inverse DRT):

Given the vector sequence $y(n)$ (DRT of $x(n)$), we compute the following (Beylkin, 1987).

1. The adjoint transform of $y(n)$ by using $z(n)=\sum_{m=-M}^{m-M}R_{-m}^{*}y(n+m)$ to obtain $z(n),n=0,\cdots,N-1$.
2. The Fast Fourier Transform (FFT) of $z(n)$.
3. A solution to a linear $(2L+1)\times(2L+1)$ system:

   $$\hat{z}(k)=\hat{H}(k)\hat{x}(k),$$

   for $k=0,1,\cdots,N-1$, where $\hat{z}(k)$ and $\hat{x}(k)$ are DFT's of $z(n)$ and $x(n)$, and $\hat{x}(k)$ is a computational matrix (refer to Beylkin (1987) for more details), for $k=k_{\min},\cdots,k_{\max}(\det\hat{H}(k)\neq0)$ to obtain $\hat{x}(k)$. We set $\hat{x}(k)=0$ outside the frequency band, i.e., for $k=0,\cdots,k_{\min}-1$ and $k=k_{\max}+1,\cdots,N/2$.
4. Inverse FFT of $\hat{x}(k)$ to obtain $x(n)$.

If (2) holds for all $k=0,\cdots,N/2$, then the DRT is invertible for all frequencies.

The iterative approach of TDRI involves repeatedly adjusting the model to minimise the difference between the observed and predicted data, same as the approach used in MPFI. However, TDRI's reliance on the τp domain enables it to more effectively reconstruct seismic data, particularly when dealing with very sparsely sampled datasets. This is because the seismic wavelet's energy is compact in time and spreads across frequencies, such that the signal often appears as sparse, concentrated in a narrow range of τ values and specific dip, making it easier to distinguish from noise than in the frequency domain. The process described by J. Wang, Ng, and Perz (2010) is as follows:

Initialisation: Set the residual $s_{\text{res}}(t,x)$ as the transformed original signal $s(t,x)$ itself and the interpolated signal s_i to 0.

For each iteration i:

1. Determine the basis function $\mathrm{BF}(\tau,p)$ for every zero-offset intercept time τ and dip p that best represent s_{res}.
2. Find the τ_i and p_i that corresponds to the maximum energy component in BF. The τ_i,p_i can be identified by minimising the cost function CF using the conjugate gradient (CG) algorithm:

   $$\mathrm{CF}=\left\|s_{\text{res}}-\mathrm{Randon}^{-1}(\mathrm{BF}(\tau_i,p_i))\right\|^2 \tag{2.8}$$

3. Add the selected basis function $\mathrm{BF}_i(\tau_i, p_i)$ as part of the the cumulative spectrum S_i, i.e. $S_i = \sum^i \mathrm{BF}_i(\tau_i, p_i)$.

4. Subtract the transformed component from the residual by $s_{\text{res}} = s_{\text{res}} - \mathrm{Randon}^{-1}(\mathrm{BF}(\tau_i, p_i))$.

Convergence: Continue the iterative process until s_{res} is sufficiently small or a predefined number of iterations is reached. The final interpolated signal (total spectrum) s_i is the sum of all selected basis functions (high energy spectral components) obtained repeatedly by step 2.

TDRI is a more powerful interpolator than MPFI, giving better reconstruction for very sparsely sampled data and overlapping events (Schlumberger, 2016). TDRI overcomes the limitation of MPFI in reconstructing overlapping signals by utilising a significantly larger number of measurements per unknown coefficient (Schonewille et al., 2014). This increased measurement density in TDRI enhances the ability to isolate and accurately reconstruct signals that may be obscured or mixed together with noises, such that precise reconstructions are possible for complex structure even when the available data is limited (Gu & Sacchi, 2009). However, this increased performance comes at a significantly higher computational cost. The higher computational cost of TDRI compared to MPFI arises because TDRI's basis functions scan a higher-dimensional space, iterating in the two-dimensional τp domain, whereas MPFI iterates along the one-dimensional k_x domain for each frequency and excluding the frequency from the inversion process. As noted in our introduction, processing our Gulf of Mexico dataset using TDRI required 9,792 times more CPU time compared to using MPFI. As the substantial computational expense of TDRI is heavily dependent on the maximum processing frequency, it is recommended to apply frequency filtering to the input data, utilising TDRI for the lower frequency components while reserving the higher frequency components for MPFI (Schlumberger, 2016). By strategically dividing the frequency spectrum for TDRI (low frequencies) and MPFI (high frequencies) respectively, we can achieve a cost-effective solution that maintains high-quality interpolation results from TDRI at an acceptable cost. Similarly, the objective of this thesis is to explore how deep learning can overcome the computational cost of TDRI while maintaining its high quality.

2.2 Deep learning theory

Deep learning is a subset of machine learning that employs artificial neural networks with multiple layers to model and learn complex patterns in large datasets (Dargan, Kumar, Ayyagari, & Kumar, 2020). It involves training these networks to automatically extract features and make predictions or decisions based on raw data, such as images, text, or sound. In particular, deep learning algorithms excel in image recognition and recreation, which can be applied in the seismic fields such as interpolation, de-noising and even inversion (Siahkoohi, Kumar, & Herrmann, 2018; S. Li et al., 2019). This section introduces foundational deep learning concepts and three models we used for low-to-high-quality seismic image translation. Generative Adversarial Networks (GANs) advance data generation by leveraging adversarial training to produce realistic samples. The Swin Transformer Convolutional Residual Network (SCRN) and its hybrid variant SCR-GAN further showcase deep learning's image reconstruction capabilities with the aid of vision transformers.

2.2.1 Introduction to deep learning

This section offers a comprehensive overview of deep learning fundamentals, including Convolutional Neural Networks (CNNs), activation functions, loss functions, learning algorithms, and the model training process. Readers already familiar with deep learning terminology and concepts may proceed directly to Section 2.2.2.

Convolutional neural networks (CNNs)

Convolutional Neural Networks (CNNs) are a cornerstone of modern deep learning, particularly renowned for their effectiveness in solving complex image-driven pattern recognition tasks and with their precise yet simple architecture (O'shea & Nash, 2015). CNNs are composed of convolutional layers, pooling layers and linear layers. Pooling layers can be omitted when we perform strided convolution which has the same spatial reduction effect (O'shea & Nash, 2015; Alzubaidi et al., 2021).

In a convolutional layer, the convolution operation involves sliding a kernel over the input data and computing the dot product between the kernel and the covered input area. When applying a convolution to an input vector $\mathbf{x} \in \mathbf{R}^n$, a convolution layer transforms it into an output vector (also known as the feature map) $\mathbf{y} \in \mathbf{R}^m$. The dimensions of the output vector are determined by several factors: the kernel size $\mathbf{R}^k$, the stride s, and the padding p. The general formula for calculating the size of the output vector m is (O'shea & Nash, 2015):

$$m = \left\lfloor \frac{n + 2p - k}{s} \right\rfloor + 1 \tag{2.9}$$

where $\lfloor \cdot \rfloor$ denotes the floor operation. A convolutional layer extracts spatial features, while a linear layer (fully connected layer) combines these features to make final predictions (Wu, 2017). In a linear layer, the input vector $\mathbf{x} \in \mathbf{R}^n$ is transformed into an output vector $\mathbf{y} \in \mathbf{R}^m$ using the following equation:

$$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{2.10}$$

where $\mathbf{W} \in \mathbf{R}^{m \times n}$ is the weight matrix and $\mathbf{b} \in \mathbf{R}^m$ is the bias vector. Understanding these components is crucial for designing and optimising CNN architectures to capture patterns hidden in seismic data.

Activation functions

Activation functions are essential in CNNs as they introduce non-linearity into the model, enabling it to learn complex patterns (Wu, 2017). In CNNs, the rectified linear unit (ReLU) and its variants like Leaky ReLU and exponential linear unit (ELU) are typically used after convolutional layers. These activation functions regulate negative values, which is crucial because the subsequent weight could involve negative elements. The equation for each activation function is as follows:

$$\mathrm{ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.11}$$

$$\text{Leaky ReLU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha x & \text{if } x < 0 \end{cases} \tag{2.12}$$

$$\mathrm{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha\left(e^{x} - 1\right) & \text{if } x < 0 \end{cases} \tag{2.13}$$

where α is a scaling factor. By using these functions, especially in deep networks, the vanishing gradient problem can be mitigated. Figure 2.3 illustrates how leaky ReLU and ELU handle negative values differently compared to ReLU. While leaky ReLU and ELU scale negative values and pass them on, ReLU outputs zero. If the weight of a specific layer is negative, ReLU will not propagate the gradient to the next weight.

Different activation functions are used in the output layer, such as the hyperbolic tangent (tanh) and sigmoid functions. The tanh function outputs values between -1 and 1 (Equation 2.14), which is suitable for tasks like image generation, while the sigmoid function outputs values between 0 and 1 (Equation 2.15), often used for binary classification tasks, such as in a discriminator (discussed later in section 2.2.2).

$$f(x)_{\tanh} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.14}$$

$$f(x)_{\text{sigm}} = \frac{1}{1 + e^{-x}} \tag{2.15}$$

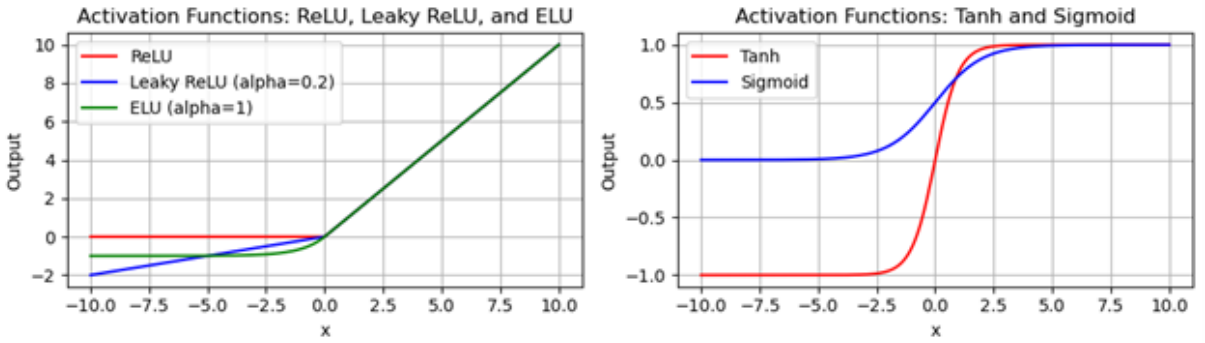


Figure 2.3: *Activation Functions: ReLU, Leaky ReLU ($\alpha = 0.2$), and ELU ($\alpha = 1$) on the left; Tanh and Sigmoid on the right.*

Loss functions

Loss functions, or criterions, measure the difference between the predicted value generated by the model and the actual values, guiding the model's learning process by quantifying prediction errors. In supervised deep learning (the input training data has labels), tasks fall into two main categories: classification and regression. Classification predicts categorical outcomes, while regression predicts continuous values.

Our seismic image translation task is a regression problem, where we predict the value of each pixel for a given input. In model training, we utilise Mean Squared Error (MSE) and Mean Absolute Error (MAE) as components of our loss function. MSE penalises larger errors more heavily due to its squaring term (Equation 2.16), making it sensitive to outliers (Chen et al., 2022). Conversely, MAE is more robust to outliers as it treats all errors linearly (Equation 2.17).

Mean squared error (MSE):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{2.16}$$

Mean absolute error (MAE):

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right| \tag{2.17}$$

Additionally, our model includes a binary classification task, for which we use Binary Cross-Entropy (BCE) or Least Squares Error (LSE) as the criterion. BCE evaluates the model's performance based on the probability values between 0 and 1 (Equation 2.18). LSE can also be applied by treating class labels as numerical values 0 and 1 (Equation 2.19), however it does not handle the probabilistic nature of classification problems as effectively as BCE.

Binary cross-entropy (BCE):

$$\mathrm{BCE} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\left(\hat{y}_i\right) + (1 - y_i)\right] \tag{2.18}$$

Least squares error (LSE):

$$\mathrm{LSE} = \sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \tag{2.19}$$

Optimisers

Learning algorithms, or optimisers, adjust the weights of a neural network to minimise the loss function by iteratively updating the weights based on gradients computed via back-propagation (Shrestha & Mahmood, 2019). Popular optimisers include Adaptive Moment Estimation (Adam) and Root Mean Square Propagation (RMSprop), each offering different strategies to enhance convergence and performance.

RMSprop (Root Mean Square Propagation) adapts the learning rate for each parameter based on the average of recent magnitudes of the gradients for that parameter. Instead of using the entire history of gradients, RMSprop uses an exponential moving average to keep track of the squared gradients, which is ideal to cope with non-stationary settings and large-scale stochastic optimisation (Zou, Shen, Jie, Zhang, & Liu, 2019).

Adam combines the ideas of RMSprop to deal with non-stationary objectives and momentum to deal with sparse gradient (Kingma & Ba, 2014). It uses moving averages of both the gradients and the squared gradients to adapt the learning rate for each parameter, and includes bias correction terms to account for the initialisation of the moment estimates. Overall, Adam is robust and suitable for non-convex optimisation problems (Kingma & Ba, 2014).

Training, validation, and testing

Machine learning workflows typically consist of three stages: training, validation, and testing. Before training a model, we split the data into a training set, a validation set, and a testing set. The data splitting proportion is detailed in Chapter 3.1.2. In each epoch, which refers to the one entire passing of the dataset through the algorithm, the model undergoes both training and validation.

During the training step, the training dataset is fed into the model so it can learn the features and iteratively update its weights. Dropout, a regularisation technique, is applied which randomly disables a fraction of neurons during training to prevent overfitting. After training, the validation set is processed by the model to assess its performance and generalisation to unseen data. If the model uses dropout during training, all neurons are reactivated during this phase.

After a fixed number of epochs, we analyse the convergence trend and stability of the validation loss, and fine-

tune the model's hyper-parameters and architecture. Once the best model is selected, we apply the testing dataset to this final model to unbiasedly evaluate its performance on unseen data.

2.2.2 Model 1: Generative Adversarial Network (GAN)

Invented by Goodfellow et al. in 2014, Generative Adversarial Networks (GANs) represent a powerful class of deep learning models. GANs comprise of two main components: generator(s) (G) and discriminator(s) (D). They often consist of CNNs (Siahkoohi et al., 2018). While G generates synthetic samples from the embedding space, D distinguishes between generated images ($\hat{y}$) and real dataset samples (Lau, 2024). The training of GANs is an adversarial process where G and D are competing against each other: G tries to fool D by producing increasingly realistic samples, D improvises at distinguishing real samples from synthetic ones. Through the adversarial learning, both G and D evolve by computing the losses and updating model weights through backpropagation (Figure 2.4).

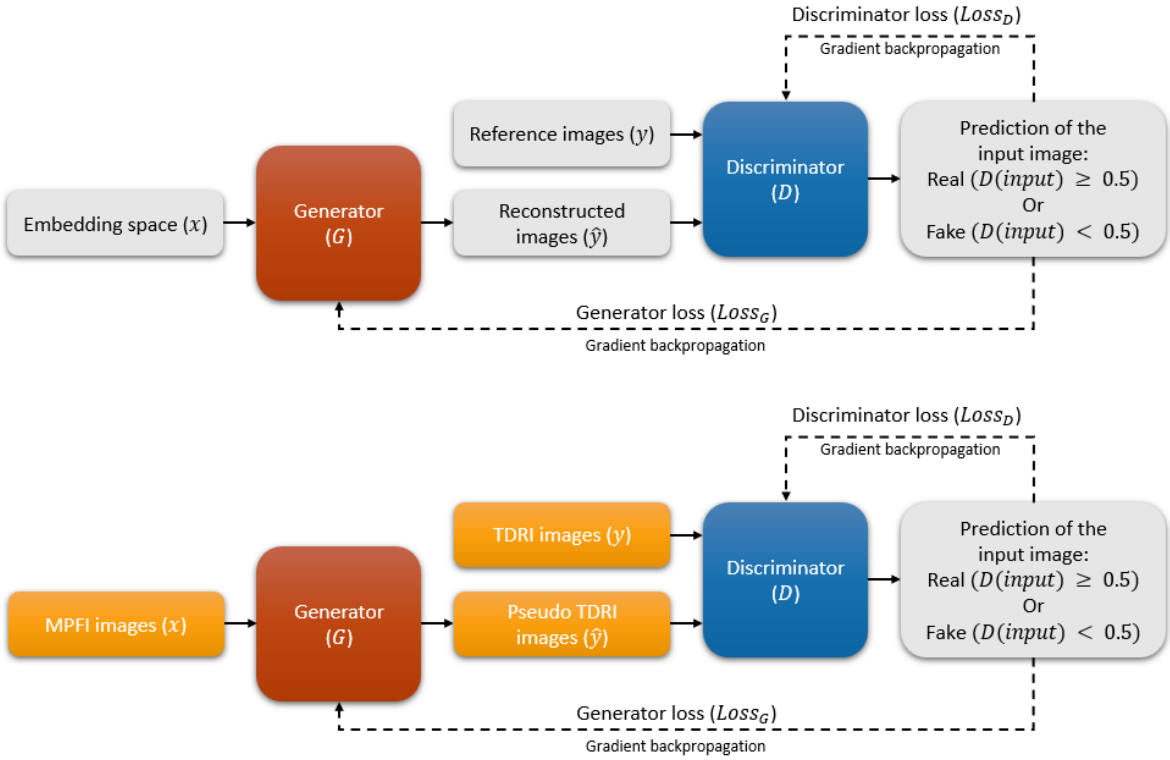


Figure 2.4: *General GAN architecture (modified from Saxena and Cao (2021)) on the top; low-to-high-quality seismic image translation GAN architecture at the bottom.*

In the context of low-to-high-quality seismic image translation, our application involves the following steps: 1) Presents MPFI images processed by SLB in-house software Omega, denoted as x, to our generator G ; 2) G creates pseudo-TDRI images from x, denoted as $\hat{y}$; 3) D assesses whether the given sample is an Omega TDRI image (y) or a pseudo-TDRI counterpart ($\hat{y}$), assigning a scalar value between 0 (indicating a reconstructed image) and 1 (indicating a real dataset image); 4) The generator loss ($loss_G$) and discriminator loss ($loss_D$) are computed based

on Equation 2.20 and 2.21; 5) The gradients of these losses are back-propagated to iteratively update the weights of G and D (Saxena & Cao, 2021).

$$\text{Loss}_{\text{G}} = \underbrace{\text{Criterion}(\text{D}(\hat{\text{y}}), 0)}_{G \text{ performance on fooling } D} + \underbrace{\lambda_1 \cdot \text{MSE}(\hat{y}, y) + \lambda_2 \cdot \text{MAE}(\hat{y}, y)}_{\hat{y} \text{ resemblance to } y \text{ in TX domain}} + \underbrace{\lambda_3 \cdot \text{MAE}(\text{fft2}(\hat{y}), \text{fft2}(y))}_{\hat{y} \text{ resemblance to } y \text{ in FK domain}} \tag{2.20}$$

$$\text{Loss}_{\text{D}} = (\underbrace{\text{Criterion}(\text{D}(\text{y}), 1)}_{D \text{ performance on classifying real image}} + \underbrace{\text{Criterion}(\text{D}(\hat{\text{y}}), 0)}_{D \text{ performance on classifying synthetic image}})/2 \tag{2.21}$$

G aims to maximise the probability of D being mistaken by generating high resemblance images. This can be achieved by minimising $loss_G$, which is computed through a combination of a criterion (BCE or LSE) between $D(\hat{y})$ and 1, MSE and MAE losses between $\hat{y}$ and y, as well as the MAE loss between $\hat{y}$ and y in the FK domain (Equation 2.20) (Kumar, Dancer, Rayment, Hampson, & Burgess, 2022). The first term measures $\hat{y}$'s ability to confuse D, second and third terms quantify the resemblance of $\hat{y}$ and y, and the last term ensure frequency consistency in the Fourier domain. In contrast, D aims to correctly classify real and fake samples. This involves minimising the criterion for real samples $D(y)$ and maximising the criterion for generated samples $D(\hat{y})$ (Kumar et al., 2022).

U-Net Architecture

Introduced in 2015 by Ronneberger, Fischer, and Brox, U-Net is a CNN architecture specifically designed for image segmentation tasks. Its distinctive U-shaped architecture comprises two key segments: the contracting path and the expanding path (Ronneberger et al., 2015; Long, Shelhamer, & Darrell, 2015). The design's main advantage lies in its ability to capture context and precise localisation, making it effective for learning local features in seismic images.

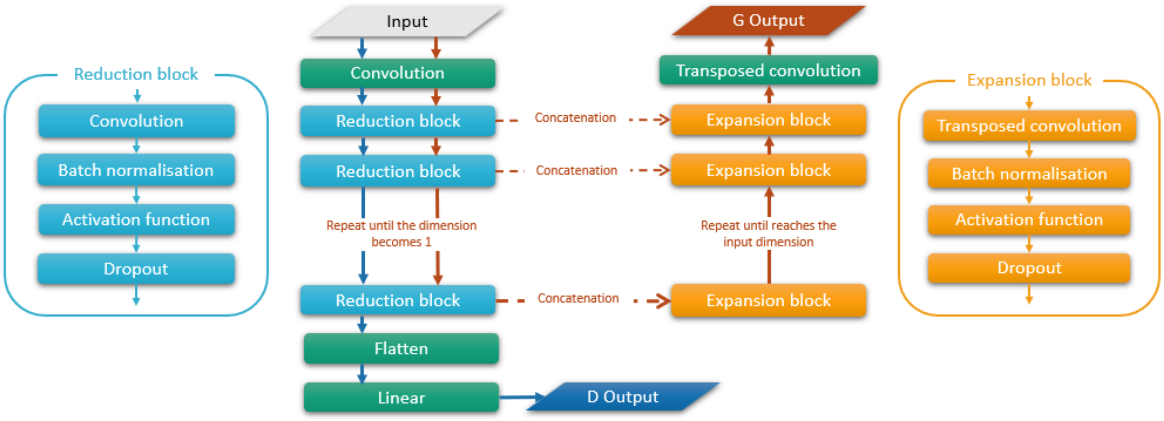


Figure 2.5: *U-Net GAN architecture.*

In our generator model, which adopts the U-Net architecture, the contracting path commences with a convolutional layer, followed by a series of reduction blocks. The number of these blocks depends on how many are needed to reduce the image height and width to 1. Each reduction block contains four layers: a convolutional layer, batch normalisation, an activation function, and dropout (Figure 2.5 left). This configuration aims to decrease the spatial resolution of the input image while simultaneously increasing the number of feature channels.

The contracting path is crucial for extracting essential features from the input image. The convolutional layers perform spatial convolutions to detect patterns such as edges and textures. Batch normalisation stabilises and accelerates training by normalising the output of the convolutional layers. The activation function introduces non-linearity, allowing the network to learn complex patterns. Dropout is employed to prevent overfitting by randomly and temporarily turning off some of the neurons during training.

Conversely, the expanding path in our model consists of multiple expansion blocks and a final layer comprising a transposed convolutional layer and a Tanh function. Each expansion block also consists of four layers: a transposed convolutional layer, batch normalisation, an activation function, and dropout (Figure 2.5 right). This part of the network is designed to elevate the spatial resolution of the feature maps.

The expanding path's role is to upsample the feature maps, effectively reversing the spatial reduction performed by the contracting path. Transposed convolutional layers perform the upsampling, while batch normalisation, activation functions and dropout are applied similarly to the contracting path. The expanding path also concatenates corresponding feature maps from the contracting path, using skip connections to merge high-resolution features with the upsampled output. This integration of features helps preserve spatial information, crucial for accurate image translation.

In our discriminator model, we adopt the contracting path of the U-Net architecture. The output from this path is then processed through a flattening layer, transforming the multi-dimensional feature maps into a one-dimensional vector. This vector is subsequently passed through a linear layer, which applies a learned weight matrix to transform the input into the desired output size. Finally, a sigmoid function is applied to limit the scalar value between 0 and 1.

2.2.3 Introduction to Vision Transformer (ViT)

A Transformer is a powerful deep learning architecture introduced by Vaswani et al. (2017). While the features captured by CNNs are small-scale and limited by the size of the convolutional kernels (Equation 2.9), transformers utilise a self-attention mechanism that enables capturing long-range dependencies (Vaswani et al., 2017). Early transformers are designed for natural language processing (NLP) purposes, such as translation and reading comprehension. For detailed information on the original NLP-oriented transformer mechanisms, refer to Vaswani et al. (2017). Dosovitskiy et al. (2020) introduced Vision Transformers (ViT) for computer vision applications, suitable for our image translation objective. In our models 2 and 3, we employ a special class of ViT, Swin Transformers, which use a shifted windowing scheme to increase efficiency in feature extraction (Z. Liu et al., 2021). This section will provide the basics of the attention mechanism, ViT, and Swin Transformers.

What is Attention?

In the context of picture-translation application, attention mechanism allows our model to relate different parts of the input image when generating the output picture. Self-attention mechanism calculates the dependencies of different pixels groups using Key (K), Value (V) and Query (Q) vectors, where Q is the information that is being looked for, K is the context or reference, and V is the content that is being searched (Nabil, 2021).

The self-attention function output is a weighted sum of V, and the weight is a softmax function of Q and K (Vaswani et al., 2017):

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.22}$$

Where d_k is the dimension of the keys. Multi-head self-attention extended the self-attention mechanism by using multiple attention heads. Each head performs self-attention (Equation 2.22) independently (Vaswani et al., 2017):

$$\text{head}_{\text{i}} = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{2.23}$$

The single head results are concatenated and linearly transformed to produce the final output.

$$\text{MultiHead}(Q, K, V) = \text{Concat}\left(\text{head}_{1}, \ldots, \text{head}_{\text{h}}\right) W^O \tag{2.24}$$

Where the projections are parameter matrices $W_i^Q \in R^{d_{\text{model}} \times d_k}, W_i^K \in R^{d_{\text{model}} \times d_k}, W_i^V \in R^{d_{\text{model}} \times d_v}$ and $W^O \in R^{hd_v \times d_{\text{model}}}$. By using parallel attention heads, the model is able to capture information from different representation subspaces at various positions efficiently.

Vision transformer (ViT)

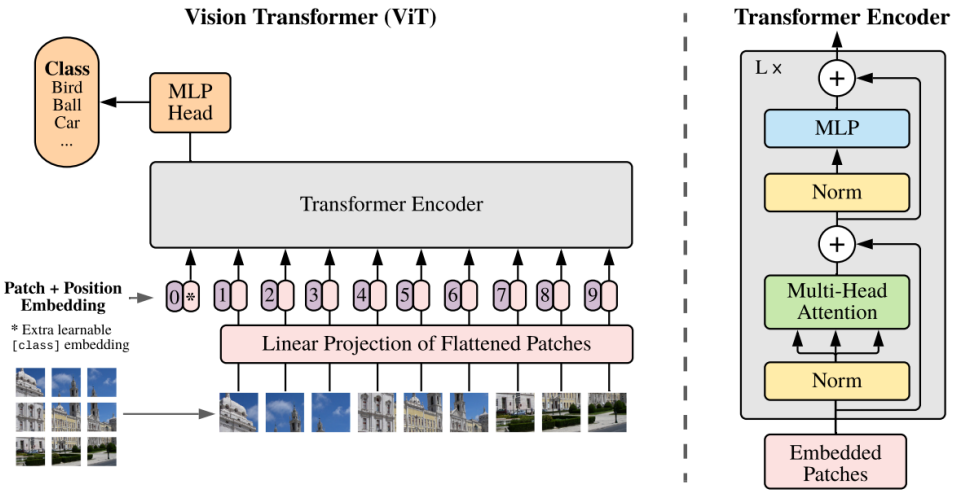


Figure 2.6: *ViT model overview (taken from Dosovitskiy et al. (2020)).*

In NLP, transformers calculate attention between words. In image processing, ViT calculates attention between image patches. Similar to how letters are grouped into words before being fed into a classic transformer, pixels are grouped into patches before being fed into ViT. The ViT flow described by Dosovitskiy et al. (2020) contains the following steps, also depicted in Figure 2.6:

- **Patch embedding:**
  The image is divided into fixed-size patches, which are flattened into vectors. Each image patch is linearly embedded into a vector of a fixed dimension using a trainable linear projection. Unlike words in a sentence that inherently have positional information, image patches require positional encodings to retain spatial information. These encodings, which can be either learnable parameters or fixed values, are added to the embeddings to indicate the position of each patch in the original image.

- **Transformer encoder:**
  The sequence of patch embeddings with positional encodings, is fed into a standard transformer encoder. This encoder has a residual structure starting with a layer normalisation, followed by a multi-head self-attention, another layer normalisation, and a multi-layer perceptron (MLP) (Figure 2.6). Layer normalisation helps stabilise the training process and improve the model's convergence by normalising the inputs across features (J. Xu, Sun, Zhang, Zhao, & Lin, 2019). The self-attention mechanism enables the model to weigh the importance of each patch relative to others, focusing on important features in the image. After the second layer normalisation, MLP is applied, which consists of hidden layers to further process the features extracted by the self-attention mechanism, enhances the model's capacity to learn complex patterns and relationships. Note that the MLP layers retain only local information while the self-attention layers compute global dependencies (Dosovitskiy et al., 2020). The transformer encoder is repeated multiple times depending on the depth of the architecture.

- **Output:**
  In Dosovitskiy et al.'s model, designed for image classification, an additional learnable classification token is calculated to output classes. For our image translation task, we apply patch un-embedding to convert the flattened positional encoded vector back to the desired output image size.

Swin Transformer

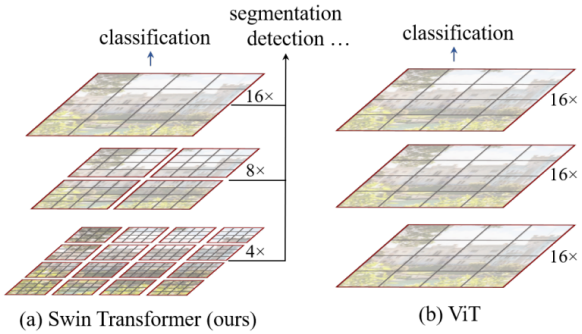


Figure 2.7: *(a) Swin transformer patching with hierarchical feature maps which perform self-attention only within local windows (red boxes); (b) ViT patching (taken from (Z. Liu et al., 2021)).*

Swin transformer, proposed by Z. Liu et al. (2021), applies a hierarchical structure on the ViT by starting with small patches and gradually increasing the patch size. In contrast to ViT that applies global self-attention, Swin transformer utilises a shift window-based self attention. Window-based self-attention limits self-attention within

a local window at each stage (red boxes in Figure 2.7), reducing the computational complexity. The shifted windows enable the model to capture both local and global pixel dependencies (Gao et al., 2024). Swin Transformer is designed to be more efficient and scalable, making it better suited for high-resolution images, perfect for our large seismic images.

2.2.4 Model 2: Swin Transformer Convolutional Residual Network (SCRN)

SCRN Architecture

The SCRN architecture proposed by Gao et al. (2024) consists of multiple feature fusion blocks (FFBs). Each FBB has a CNN module and a ViT module, in which the CNN block extracts local features and the ViT block extract global features. The CNN block excels at extracting local features, capturing the fine details and textures within a small receptive field. Meanwhile, the ViT block is designed to extract global features, allowing the model to consider long-range dependencies across the entire image (Dosovitskiy et al., 2020). The combination of these modules enables the network to learn both local patterns and long-range feature correlations, thereby preserving the continuity of events in seismic data while maintaining high sensitivity to weak signals.

In the original design, SCRN starts and ends with a 3 x 3 convolution layer, and has 5 FFBs in between. The initial convolutional layer increases the input channel of 1 (for a greyscale seismic image) to 64 channels for later processing. For an input with height and width of 256, the dimension will be modified from [256, 256, 1] to [256, 256, 64]. Conversely, the final convolutional layer reduces 64 channels back to 1 to match the output size [256, 256, 1]. In our experiment, we modified the architecture to 7 and 9 FFBs to cope with more complex real industrial seismic datasets. The detailed architecture is depicted in Figure 2.8.

Feature fusion blocks (FFBs)

FBBs are connected in three different manners in our experiment. The original design (Figure 2.9 right) is a mirrored residual network (ResNet) where shallow layers have skip connections to deeper layers, allowing the network to learn residual mappings. This architecture takes the advantages of ResNet, which include easier optimisation and avoidance of vanishing or exploding gradients (He, Zhang, Ren, & Sun, 2016). The bottleneck design, starting and ending with convolutional layers, facilitates the construction of deeper models without a proportional increase in computational complexity (Srinivas et al., 2021). The other two connections are ResNet and simple forward pass, which will be discussed in the subsection 3.4.3.

Each FBB begins with a 1x1 convolutional layer to learn residual mapping. Its 64 channels are then split into two groups of 32 channels, i.e. [256, 256, 32]. One group is passed to a CNN block for small-scale correlation learning, while the other group is processed by a ViT block to extract long-range pixel relevance. After these parallel processes, the channels are concatenated, combining information extracted at different scales into a single, richer representation. The block concludes with a 3x3 convolutional layer to capture spatial details, followed by another 1x1 convolution layer to restore the original dimensionality (Gao et al., 2024).

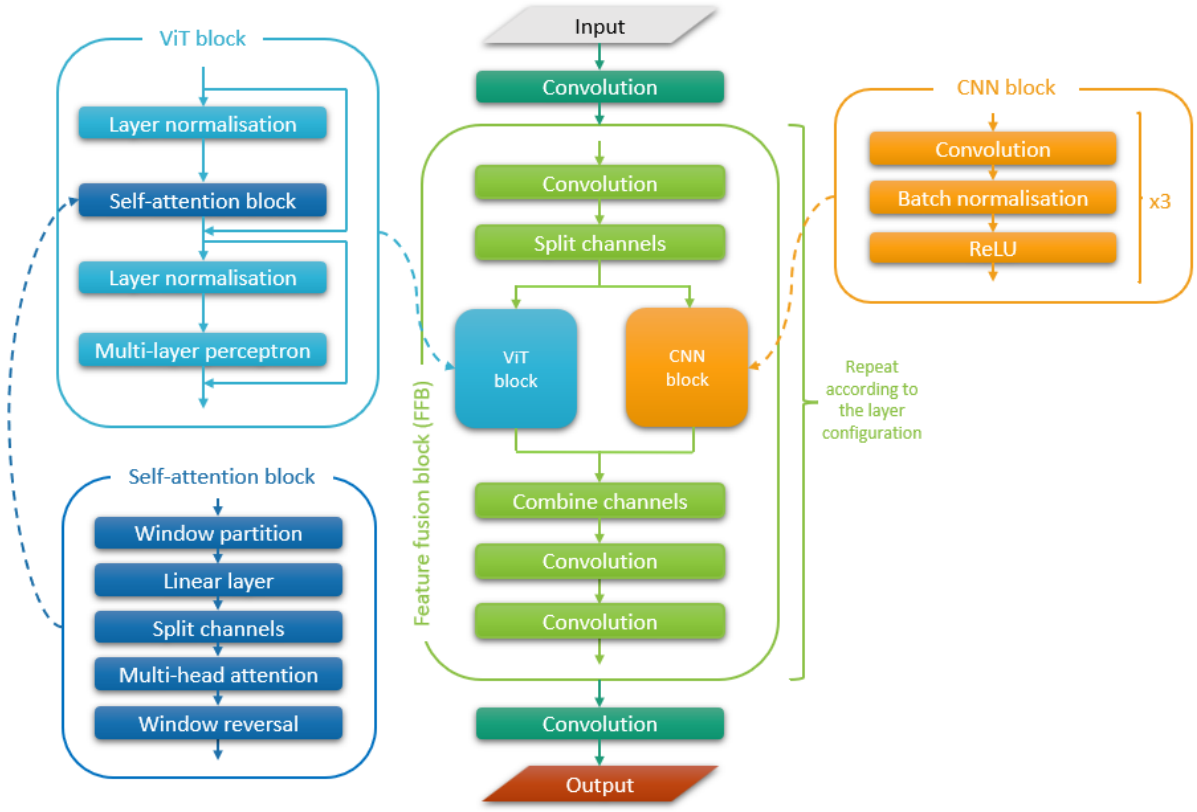


Figure 2.8: *SCRN architecture (modified from Gao et al. (2024)).*

ViT block: global feature extraction

The ViT block in each FBB shares the same structure as mentioned in Section 2.2.3, which starts with a layer normalisation, followed by a self-attention block, another layer normalisation, and a multi-layer perceptron (MLP). The self-attention mechanism is the core of the ViT block and we will explain in details next. The MLP consists of a hidden layer of dimension [256, 256, 128] to capture more complex patterns.

The self-attention block we deployed is a Window-based Multi-Head Self-Attention (W-MSA). Traditional self-attention mechanisms, though powerful, suffer from quadratic complexity with respect to the input size, making them computationally expensive and memory-intensive for large images. W-MSA addresses this challenge by partitioning the input image into smaller, non-overlapping windows and applying self-attention within each window independently, significantly reducing computational needs compared to performing self-attention on the entire feature map (J. Li, Yan, Liao, Yang, & Shao, 2021).

The first step in W-MSA is window partition, by dividing the input image of dimension [256, 256, 32] into smaller fixed-size windows of size 8×8, the resulting dimension is $[(\frac{256}{8} \times \frac{256}{8}), 8 \times 8, 32]$. This partitioning reduces computational complexity by focusing attention computations within localised regions, effectively capturing local features. The fully connected layer then projects these input features into a higher-dimensional space, 3-folding our input channels from 32 to 96, generating Query (Q), Key (K), and Value (V) representations

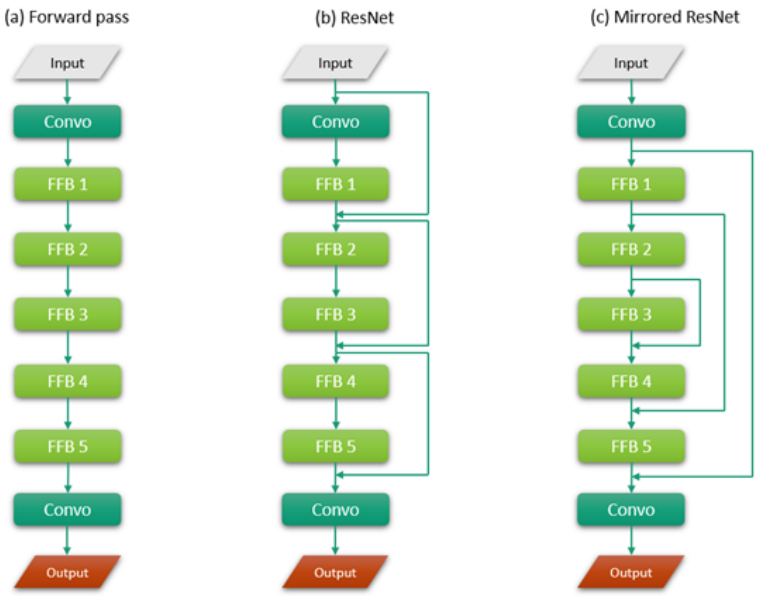


Figure 2.9: *FFBs configuration and passing.*

essential for calculating attention scores, such that the dimension is $[(\frac{256}{8} \times \frac{256}{8}), 8 \times 8, 32 \times 3]$. Within each window, self-attention scores are computed by measuring the compatibility between queries and keys, determining the importance of each patch relative to others. To incorporate spatial information, relative positional encodings are used, capturing the relationships between patches within a window. Then we split the channels in 3 matrices and changed the dimension to $[3, (\frac{256}{8} \times \frac{256}{8}), 3, 8 \times 8, 32]$, to match our 3-head attention mechanism. This mechanism operates in parallel, with each head processing different subsets of features to capture diverse relationships. The outputs from all heads are concatenated and linearly transformed. Depending on the W-MSA type, a mask is generated to restrict attention to certain areas, crucial for shifted windows to focus correctly. For this Swin Transformer model, windows are shifted before applying self-attention to capture cross-window interactions, improving feature extraction (Z. Liu et al., 2021). Finally, window reversal is done by applying a linear projection which transforms the concatenated multi-head attention output into the desired dimension of [256, 256, 32] for further processing (S, 2023).

CNN block: local feature extraction

The CNN block in this architecture is composed of a sequence of three repeated layers, each containing a 3 x 3 convolution layer, a batch normalisation layer, and a ReLU activation function. Each sequential layer learns increasingly abstract representations, capturing intricate patterns and details from the input data.

The convolutional layer in the sequence applies filters to input images, detecting local patterns like edges and textures. Using 3x3 kernels with a stride of 1 and padding of 1 maintains spatial dimensions, preserving details. Batch normalisation stabilises training by normalising outputs per mini-batch to zero mean and unit variance, enhancing efficiency and stability. It includes learnable parameters for scaling and shifting outputs. ReLU activation follows, introducing non-linearity by outputting positive values directly and zero for negatives to avoid

vanishing gradient.

By integrating the CNN block into the SCRN architecture, the model is equipped to effectively handle the intricate and large-scale features present in seismic images, facilitating the low-to-high-quality image translation task.

2.2.5 Model 3: Swin Transformer Convolutional Residual Generative Adversarial Network (SCR-GAN)

Torbunov et al. (2023) developed a general-purpose architecture, UVCGAN, to perform image-to-image translation task, leveraging the power of both U-Net GAN and ViT. Similarly, SCR-GAN is a GAN model that incorporates SCRN as the generator (Figure 2.10), while preserving the discriminator architecture described in subsection 2.2.2. This approach leverages the strengths of SCRN to enhance the generator's performance, particularly boosting the overall seismic features' continuity. The generator architecture is explained previously in subsection 2.2.4. With the aid of the ViT block capturing large-scale features and the CNN block learning regional details, the generator's capability of generating realistic pseudo-TDRI is improved through these advanced feature fusion techniques, while the presence of the discriminator is increasing the robustness of the model.

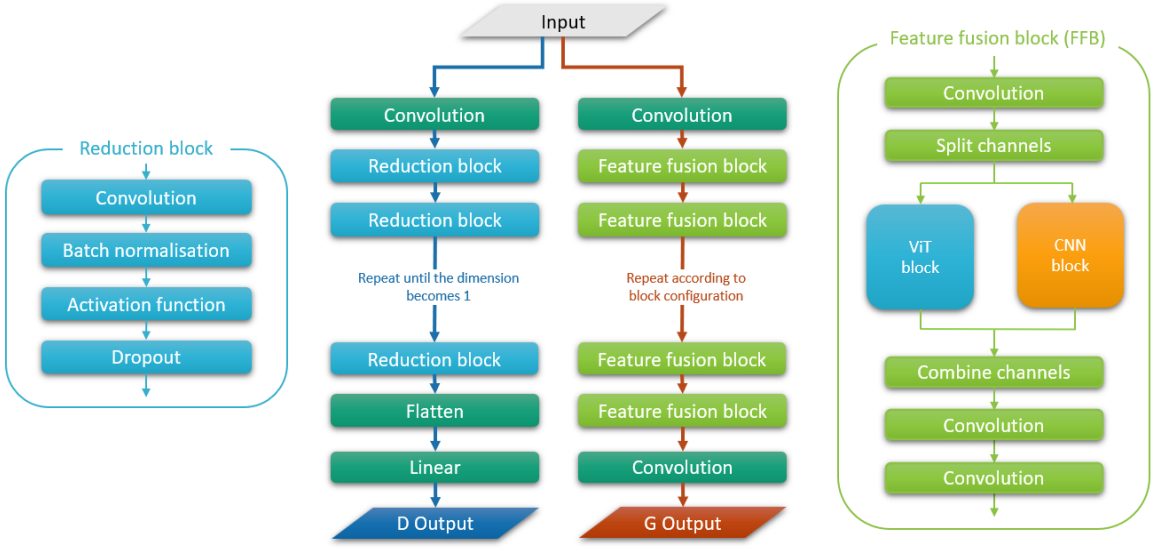


Figure 2.10: *SCR-GAN architecture.*

CHAPTER 3

Methodology

3.1 Data preparation

In contrast to traditional image translation tasks, where entire images are processed at once, seismic images cover an areal extent of kilometers and contain significantly more pixels and details than a high definition (HD) picture. To handle this complexity, we first visually interpreted the raw seismic volume, extracted slices according to the desired gather, and then created smaller patches of 256 x 256 pixels for our deep learning model to process and learn from. Data preparation is vital for the model to manage and learn from the vast amount of information contained in seismic images.

3.1.1 Seismic datasets

Acquired in the Gulf of Mexico, our raw 5D seismic volume is collected from a seismic survey conducted using deep-water ocean bottom nodes (OBN) at a depth ranging from 500-1700 meters. 5D refers to time (t), source's x-coordinate (S_x), source's y-coordinate (S_y), receiver's x-coordinate (R_x), receiver's y-coordinate (R_y). The source grid composes of 601 inline points with 50 m spacing and 826 crossline points with 100 m spacing, covering an area of 10300 km^2. The receiver nodes are deployed in a staggered grid of 1000 m by 1000 m over an an area of 3700 km^2. The time sampling interval is 12 ms, and we have a total of 500 time samples, which is an equivalent of 6 seconds of data.

The raw seismic volume was processed using Omega, a geophysical processing software, transforming the raw data into two seismic volumes from a 50x100m grid to a 12.5x12.5m grid using MPFI and TDRI respectively. From these, we extracted two common receiver gathers, resulting in two 3D volumes represented by t, S_x, and S_y.

3.1.2 Data slicing

From each 3D seismic volume, we extracted a total of 601 2D seismic crossline slices (Figure 3.1, left). Crossline slices are chosen as they have larger spacing in the source grid and require higher interpolation rate. Every 2D seismic slice was further divided into patches of size 256 x 256 with a 50% overlap (Figure 3.1, right). For edge patches, zero padding was applied to maintain the patch size of 256 x 256. In our subsequent experiments, we will exclude the edge patches for training, such that our model can focus on the high energy parts that captured the major events. 10 patches were created from each 2D slice, summing up to a total of 6010 patches for broadband input, or 30050 patches for frequency augmented input. Every MPFI patch was paired up with the TDRI patch extracted at the same location, and they are normalised with respect to the maximum amplitude of the MPFI patch. We randomised the order of these paired patches using a subset random sampler in the data-loader and separated 80% of them as the training dataset and the remaining 20% as the validation counterpart. For the testing dataset, we will use patch pairs, including the edge patches, created from a separate seismic volume.

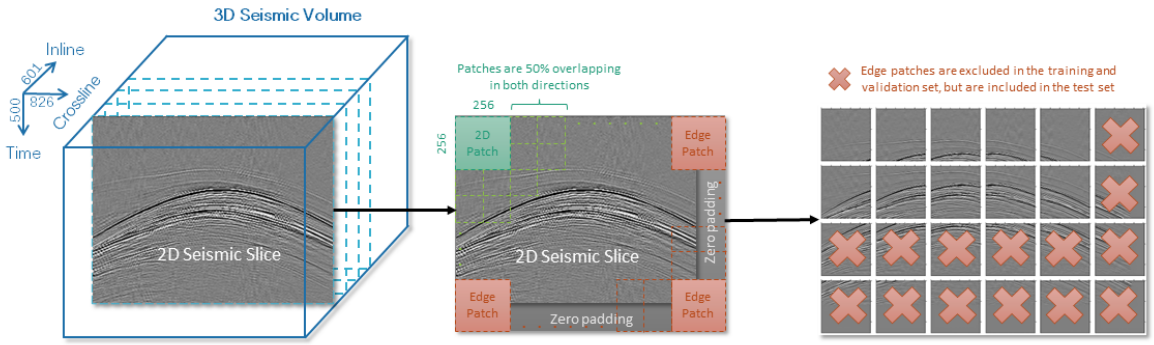


Figure 3.1: *Creating patches from a 3D seismic volume.*

3.2 Model training and validation

All our experiments were performed using NVIDIA A100 80GB Tensor Core GPUs. With up to 1.3 TB of unified memory per node, the A100 80GB GPU was ideal for training large models involving areal seismic volume input (NVIDIA, 2020). We used PyTorch v2.3 as our deep learning framework. More information on the version control can be found in Appendix A.1.

Algorithm 1 GAN and SCRN-GAN training and validation loop

Input: Epoch e, Generator G, Discriminator D, Criterion *Crit*, SNR function *snr_func*, Optimisers *optim_G* *optim_D*, Training dataloader L_{train}, Validation dataloader L_{val}, Learning rate schedulers *sche_G* *sche_D* (optional)

for $epoch \in \{1, \ldots, e\}$ **do**
  for $x, y \in L_{train}$ **do**
    ∇*optim_G* $\leftarrow 0$
    ∇*optim_D* $\leftarrow 0$
    $\hat{y} \leftarrow \mathrm{G}(x)$
    fake_score $\leftarrow \mathrm{D}(\hat{y})$
    real_score $\leftarrow \mathrm{D}(y)$
    train_loss_G $\leftarrow$ *Crit*(fake_score,0) + 100·MAE($\hat{y}, y$) + 100·MSE($\hat{y}, y$) + 100·MAE(fft2($\hat{y}$),fft2(y))
    train_loss_D $\leftarrow$ (*Crit*(fake_score,0)+*Crit*(real_score,1)) $\div$ 2
    train_snr $\leftarrow$ *snr_func*($\hat{y}, y$)
    train_loss_G.backward()
    train_loss_D.backward()
    optim_G.step()
    optim_D.step()
  end
  sche_G.step() ◁ If present
  sche_D.step() ◁ If present
  for $x, y \in L_{val}$ **do**
    $\hat{y} \leftarrow \mathrm{G}(x)$
    fake_score $\leftarrow \mathrm{D}(\hat{y})$
    real_score $\leftarrow \mathrm{D}(y)$
    val_loss_G $\leftarrow$ *Crit*(fake_score,0) + 100·MAE($\hat{y}, y$) + 100·MSE($\hat{y}, y$) + 100·MAE(fft2($\hat{y}$),fft2(y))
    val_loss_D $\leftarrow$ (*Crit*(fake_score,0)+*Crit*(real_score,1)) $\div$ 2
    val_snr $\leftarrow$ *snr_func*($\hat{y}, y$)
  end
end

We followed Algorithm 1 for training both GAN and SCRN-GAN models, and Algorithm 2 specifically for SCRN. Training GAN-based models using Algorithm 1 was relatively similar to training SCRN models, the main difference was that GAN-based training involved updating the extra weights for the discriminator, resulting in two separate optimisers, losses and schedulers.

Algorithm 2 SCRN training and validation loop

Input: Epoch e, SCRN Model M, SNR function *snr_func*, Optimiser *optim*, Training dataloader L_{train}, Validation dataloader L_{val}, Learning rate schedulers *sche* (optional)

for $epoch \in \{1, \ldots, e\}$ **do**
  for $x, y \in L_{train}$ **do**
    $\nabla optim \leftarrow 0$
    $\hat{y} \leftarrow \mathrm{M}(x)$
    train_loss $\leftarrow$ 100·MAE($\hat{y}, y$) + 100·MSE($\hat{y}, y$) + 100·MAE(fft2($\hat{y}$),fft2(y))
    train_snr $\leftarrow$ *snr_func*($\hat{y}, y$)
    train_loss.backward()
    optim.step()
  end
  sche.step() ◁ If present
  for $x, y \in L_{val}$ **do**
    $\hat{y} \leftarrow \mathrm{M}(x)$
    val_loss $\leftarrow$ 100·MAE($\hat{y}, y$) + 100·MSE($\hat{y}, y$) + 100·MAE(fft2($\hat{y}$),fft2(y))
    val_snr $\leftarrow$ *snr_func*($\hat{y}, y$)
  end
end

Our training script involved one outer loop (epoch) and two inner loops (training and validation). Within each epoch, the script iterated over batches of data from the training data-loader, where each batch contained pairs of MPFI (x) and TDRI (y) patches. The number of pairs determined by the batch size. For each forward pass, it first reset our optimiser(s) gradient to zero. The model then generated pseudo TDRI ($\hat{y}$) patches based on x. In case of a GAN-based model, the discriminator evaluated the real y and the generated $\hat{y}$ patches, calculating their resemblance score respectively. Afterwards, it calculated the losses and the SNR, logging these metrics for later analysis. Details of the error quantification metric will be discussed in the next section. The losses were back-propagated and optimisers were updated within the training inner loop. Once all training data was iterated, the learning rate schedulers (if used) were stepped to adjust the learning rates. The validation loop mirrored the training loop but with the model and discriminator set to evaluation mode, deactivating dropout and ensuring all neurons are active. It iterated over batches from the validation data loader, calculating and accumulating validation losses and SNRs in the same manner as in the training loop.

Models were saved during training whenever a new highest validation SNR was achieved, as well as at the end of the final epoch. Saved models are used later for the image translation task or as a base model for transfer learning on other seismic applications. Hyper-parameters such as batch sizes, optimisers, and learning rates were carefully selected, with details provided in section 3.4.

3.3 Model evaluation metrics

The goal of our model is to create realistic pseudo TDRI images from any MPFI input. Achieving satisfactory performance on both the training dataset and unseen validation dataset is crucial to ensure the robustness and generalisability of the model. Throughout the training process, we monitored four primary metrics within each epoch: training loss, validation loss, training Signal-to-Noise Ratio (SNR), and validation SNR. For GAN-based models, we also tracked two additional metrics to evaluate the discriminator's performance: discriminator training loss and discriminator validation loss. These validation metrics were given particular emphasis as they reflected how well the model generalised to cope with unseen data.

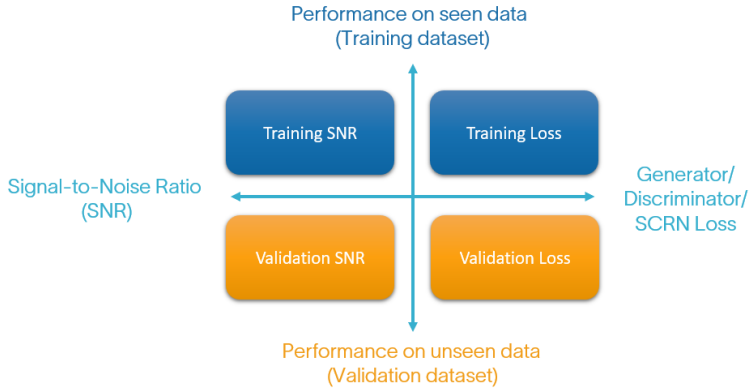


Figure 3.2: *Four model performance metrics: Training loss, Validation loss, Training SNR, Validation SNR.*

Our discriminator loss, denoted as Loss_D, is defined in Equation 2.21. To optimise the generator loss Loss_G in Equation 2.20, we experimented with different weighting coefficients λ. Specifically, we tested fixed weight, automatic-weight, and normalised weight. Through these experiments, we determined that the optimal values for the weighting coefficients were $\lambda_1 = \lambda_2 = \lambda_3 = 100$. Consequently, the generator loss function was formulated as follows:

$$\text{Loss}_\text{G} = \underbrace{\text{Criterion}(\text{D}(\hat{y}), 0)}_{G \text{ performance on fooling } D} + \underbrace{100 \cdot \text{MSE}(\hat{y}, y) + 100 \cdot \text{MAE}(\hat{y}, y)}_{\hat{y} \text{ resemblance to } y \text{ in TX domain}} + \underbrace{100 \cdot \text{MAE}(\text{fft2}(\hat{y}), \text{fft2}(y))}_{\hat{y} \text{ resemblance to } y \text{ in FK domain}} \tag{3.1}$$

Our SCRN loss function operates similarly to the generator loss, but without the criterion term as it does not rely on the discriminator. The SCRN loss function is expressed as:

$$\text{Loss}_\text{M} = \underbrace{100 \cdot \text{MSE}(\hat{y}, y) + 100 \cdot \text{MAE}(\hat{y}, y)}_{\hat{y} \text{ resemblance to } y \text{ in TX domain}} + \underbrace{100 \cdot \text{MAE}(\text{fft2}(\hat{y}), \text{fft2}(y))}_{\hat{y} \text{ resemblance to } y \text{ in FK domain}} \tag{3.2}$$

The lower Loss_G or Loss_M reached, the better the model performance. This also applies to the training Loss_D. However, for the validation Loss_D, convergence to a fixed value signifies a balanced state where both the discriminator and generator are learning and challenging each other effectively. Substituting 0.5 for $D(y)$ and $D(\hat{y})$ in Equation 2.21, validation Loss_D will reach 0.5 for the least square criterion or 0.7 for the BCE criterion, indicating a well-trained generator in GAN. In addition to the loss metrics, we use SNR to quantitatively evaluate the image translation results. The SNR is calculated using the following equation:

$$\text{SNR} = -20 \cdot \log_{10} \left(\frac{\|y - \hat{y}\|_F}{\|y\|_F} \right), \tag{3.3}$$

Here, F denotes the Frobenius norm. The SNR metric provides a measure of the resemblance of the generated image ($\hat{y}$) compared to the real image (y), with higher SNR values indicating better performance. By utilising these metrics, we can effectively optimise our model and select the best hyper-parameters, ensuring that the model performs well on both the training and validation datasets.

3.4 Model optimisation

3.4.1 GAN Hyper-parameters optimisation

To optimise our GAN performance, we focused on selecting the best hyper-parameter values that yield the highest SNR and the lowest generator loss for our final model. By systematically changing one hyper-parameter at a time, we evaluated the generator performance using the test values in Table 3.1 and picked the best option according to the result shown in Figure 3.4:

Hyper-parameter	**Base value**	**Test values**	**Best value**
Activation function	ELU(α=1)	ReLU, leakyReLU(α=0.2)	ELU(α=1)
Batch size	1	2, 4	2
Feature size	8	16, 32, 64, 128, 256	64
Optimiser	Adam	RMSprop	Adam
Learning rate	0.0001	0.00001, 0.00003, MultiStepLR, CyclicLR	MultiStepLR
Criterion	LSE loss	BCE loss	LSE loss

Table 3.1: *GAN hyper-parameter test values.*

1. **Activation functions** ReLU, leaky ReLU and ELU were tested to determine which one best supports the learning process of our GAN, influencing how the network handles non-linearities. Employing ELU as the activation function could reduce validation loss by 4.2% and enhance 0.8 dB in SNR compared to using ReLU and Leaky ReLU. Note that the occasional instability in the validation metrics is normal due to the stochastic nature of small batch training. Our optimiser adjusts the model weights based on each batch in the training set, not on the validation set. Therefore, fluctuations in validation metrics from epoch to epoch are acceptable as long as the overall trend shows convergence.

2. **Batch size:** Batch sizes of 1, 2 and 4 were experimented with to determine the best size that ensures stable training and efficient gradient estimation. A batch size of 2 brought improvements compared to a batch size of 1, reducing validation loss by 3.4% and enhancing SNR by 0.2 dB. Batch size of 2 also had more stability than batch size of 4 towards the final epochs. Notably, the choice of a batch size of 2 significantly reduced the training time by 50% compared to a batch size of 1 (Figure 3.3).

3. **Feature size:** We tested various numbers of features (8, 16, 32, 64, 128, 256) to assess their impact on the performance of our GAN in generating realistic outputs. Notably, feature sizes of 128 and 256 exhibited high fluctuations and instabilities in losses and SNR, particularly on the training set. They also took significantly longer time for model training (Figure 3.3) compared to smaller feature sizes. Conversely, smaller sizes (8,

16, 32) resulted in much lower SNR and higher losses. In the final epoch, feature size of 8 reached a validation loss of 232 and SNR of 13 dB, while feature size of 64 resulted a validation loss of 182 and SNR of 15 dB. Therefore, the optimal feature size for stable performance and convergence is determined to be 64.

4. **Optimiser:** We experimented with Adam and RMSprop to find the one that most effectively updates the model's weights, improving convergence speed and stability. They both exhibited similar trend with Adam very slightly outperforming RMSprop in terms of the validation loss (-1%) and SNR (+0.15 dB).

5. **Learning rate:** Various learning rates were applied to identify the optimal rate at which the model learns, balancing between too fast (risking instability) and too slow (causing slow convergence). We tested a fixed learning rate of 0.0001, 0.00003, and 0.00001. Additionally, we experimented with a multi-step learning rate that reduces by 20% every 10 epochs, starting from 0.0001. Finally, we applied a cyclical learning rate which increases from base line learning rate 0.00003 to maximum 0.0001 and then decreases back to base rate over the 50 epochs, completing one cycle. With triangular2 mode, the range for the next cycle is half of the previous one, progressively narrowing the oscillation range. While all five learning rates showed similar convergence trend, the multi-step learning rate had the highest stability and the best validation metrics values.

6. **Criterion:** We evaluated LSE loss and BCE loss to find the criterion that best measures the discrepancy between discriminator prediction and the real state, guiding the generator to produce more accurate images. Both loss functions showed similar convergence in terms of SNR and losses. However, in terms of the training stability, LSE loss outperformed BCE loss.

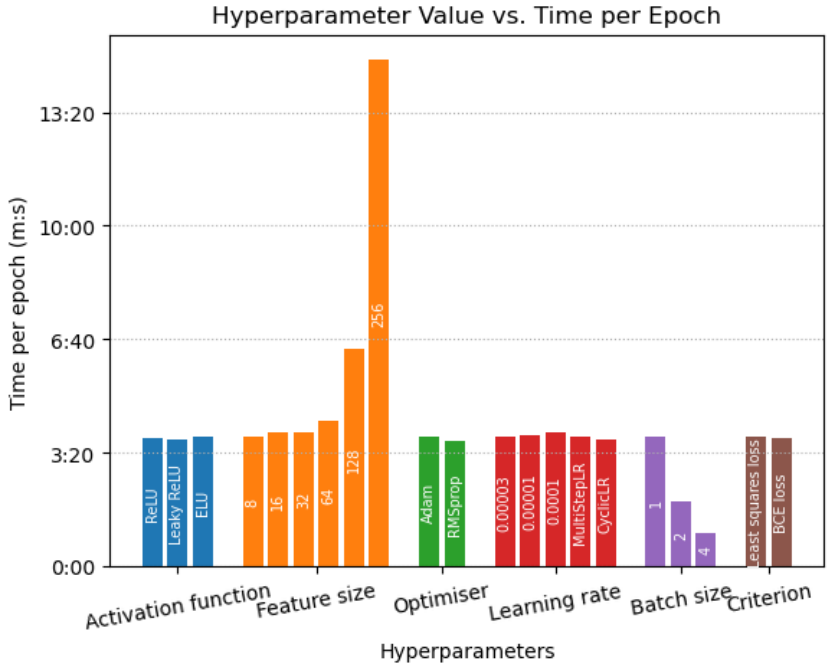


Figure 3.3: *Measurement of the training time taken per epoch for GAN configured with specified hyper-parameters.*

After adjusting and testing these hyper-parameters, we identified the optimal configuration for our GAN as follows: using ELU activation function, feature size 64, Adam optimiser, multi-step learning rate scheduling, batch size 2, and LSE loss as the criterion.

3.4.2 SCRN hyper-parameters optimisation

By applying the same methods as GAN hyper-parameter values optimisation, we tested and selected the best optimiser, learning rate and batch size for our SCRN (see Table 3.2).

Hyper-parameter	Base value	Test values	Best value
Optimiser	Adam	RMSprop	RMSprop
Learning rate	0.0002	0.00005, 0.0001, MultiStepLR, CyclicLR	MultiStepLR
Batch size	4	1, 2	2

Table 3.2: *SCRN hyper-parameter test values.*

1. **Optimiser:** Adam and RMSprop were utilised as learning algorithms in our experiment. Both algorithms demonstrated similar convergence patterns; however, Adam exhibited greater stability throughout the training process. Specifically, RMSprop had a marginal advantage in terms of validation loss and SNR, with a lower validation loss of 0.1% and an increase in SNR of 0.1 dB over Adam.

2. **Learning rate:** We tested five different learning rates: 0.0003, 0.0001, 0.00005, a multi-step scheduler (starting at 0.0002 and decreasing by 20% every 10 epochs), and a cyclic learning rate (fluctuating from 0.00005 to 0.0002 over 50 epochs, with the oscillation window narrowing according to the triangular2 mode). The fixed rate of 0.00005 was too small and it resulted higher losses and lower SNR compared to other rates. All learning rates showed good convergence. Notably, the multi-step learning rate achieved the lowest validation loss of 180 and an SNR exceeding 15.5 dB, indicating its effectiveness.

3. **Batch size:** Similar to our approach with GANs, we tested batch sizes of 1, 2, and 4. Batch sizes of 2 and 4 both demonstrated stable convergence. Using a batch size of 4 resulted in the lowest training loss and highest training SNR. However, in the validation metrics, batch size of 2 outperformed, with approximately 0.2 dB better in the validation SNR, suggesting better generalisation and model robustness.

After conducting the SCRN hyper-parameter test, we concluded the best model setup is with RMSprop optimiser, multi-step scheduler, and a batch size of 2.

3.4.3 SCRN Block configuration

The default SCRN model by Gao et al. (2024) has 5 Feature Fusion Blocks (FFBs) connected with Mirrored ResNet. To optimise the SCRN model for our low-to-high resolution seismic image translation purpose, we experimented with adjusting the number of FFBs and block passing strategies to determine the configuration that delivers the most effective performance.

1. **Number of FFBs:** We experimented using 5, 7, 9, and 11 FFBs in our SCRN model. Figure 3.7 (first row) shows the more the FFBs, the higher SNR and the lower losses the model can reach, valid for both training

Block configuration	Base value	Test values	Best values
Number of FFBs	5	7, 9, 11	11
Block passing	Mirrored ResNet	Forward pass, ResNet	Mirrored ResNet

Table 3.3: *SCRN block configuration test values.*

and validation metrics. They all showed stability and convergence. In particular, 11 FFBs model reached a high validation SNR of 17 dB, a 1.3 dB improvement compared to the original 5 FFBs model. Indicating our image translation task is more complex and required a deeper network to learn all the necessary features.

2. **Block passing:** We tested simple forward pass, ResNet and Mirrored ResNet for connecting our FFBs. Each passing strategy is detailed in Figure 2.9. In Figure 3.7 (second row), we can see the simple forward pass performed the best in the validation set for a 5-FFBs model. However, applying on an 11-FFBs model, Mirrored ResNet and ResNet performed better in all metrics. This is because the deeper the network, the smaller the gradients back-propagate, leading to slower learning. ResNet creates shortcut connections and allow gradient flow through the network. Mirrored ResNet marginally outperformed ResNet in our 11-FFBs scenario.

After conducting the SCRN block configuration analysis, we concluded the best model setup is with 11 FFBs connected with a Mirrored ResNet. Conducting detailed hyper-parameters analysis for our three models, we have summarised the best set up in Table 3.4, which will be employed in our model training.

	Architecture-based hyper-parameters			
Model	**Activation function**	**Feature size**	**No. of FFBs**	**Block passing**
GAN	ELU($\alpha = 1$)	64	/	/
SCRN	/	/	11	Mirrored ResNet
SCR-GAN	ELU($\alpha = 1$)	64	11	Mirrored ResNet
	Training-based hyper-parameters			
Model	**Batch size**	**Optimiser**	**Learning rate**	**Criterion**
GAN	2	Adam	MultiStepLR*	LSE
SCRN	2	RMSprop	MultiStepLR'	/
SCR-GAN	2	RMSprop	MultiStepLR'	LSE

MultiStepLR*: Learning rate reduced by 20% every 10 epochs, starting from 0.0001

MultiStepLR': Learning rate reduced by 20% every 10 epochs, starting from 0.0002

Table 3.4: *Final model parameters for GAN, SCRN and SCR-GAN.*

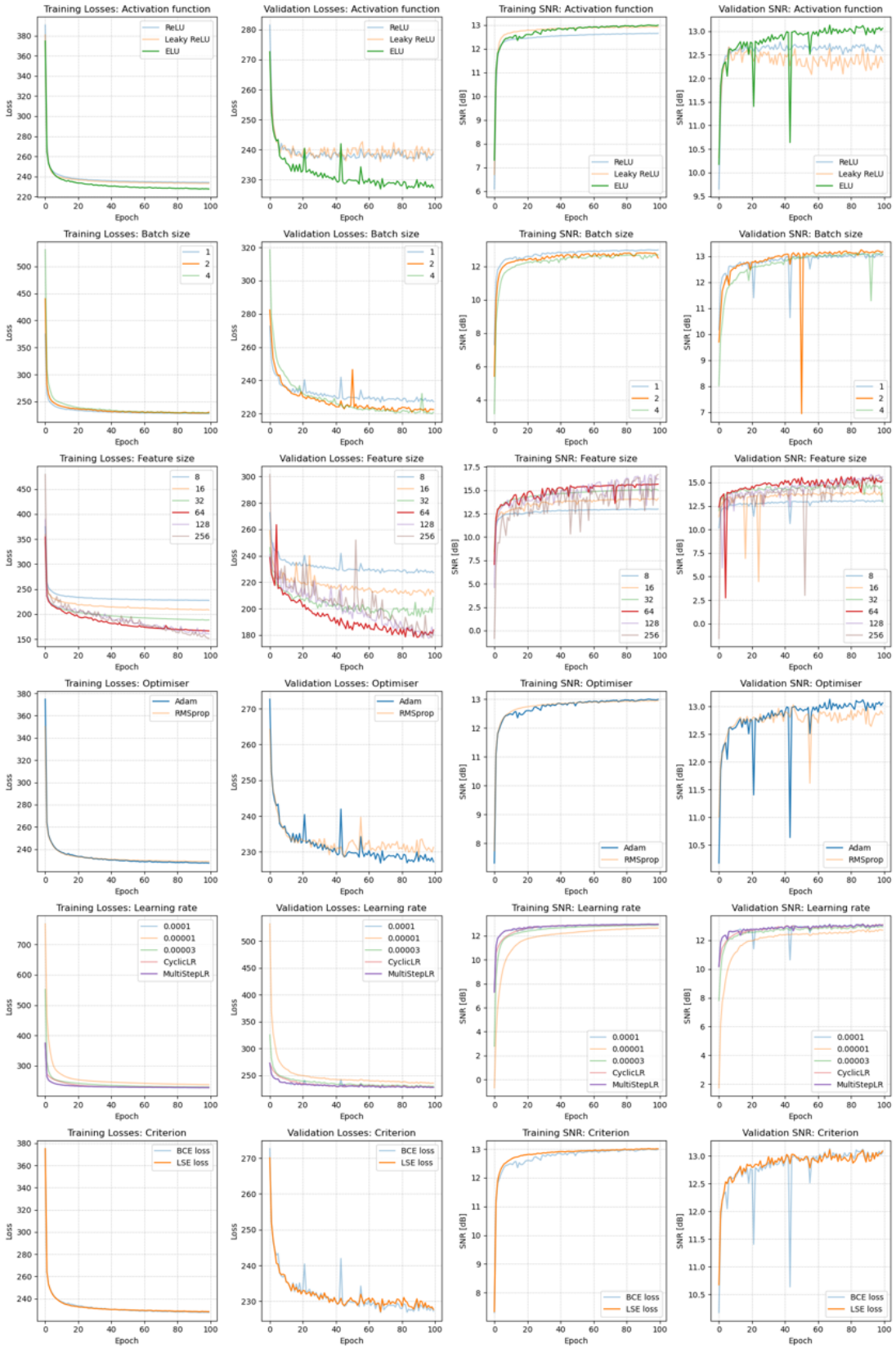


Figure 3.4: *The effects of hyper-parameters on GAN training and validation. The parameters examined include activation functions (ReLU, Leaky ReLU, and ELU), batch sizes (1, 2, 4), feature sizes (8, 16, 32, 64, 128, 256), optimisers (Adam, RMSprop), learning rates (0.0001, 0.00001, 0.00003, MultiStepLR, CyclicLR), and loss criteria (LSE loss, BCE loss). Each row corresponds to a different hyper-parameter; and each column corresponds to a validation metrics (training losses, validation losses, training SNR, and validation SNR) over 100 epochs.*

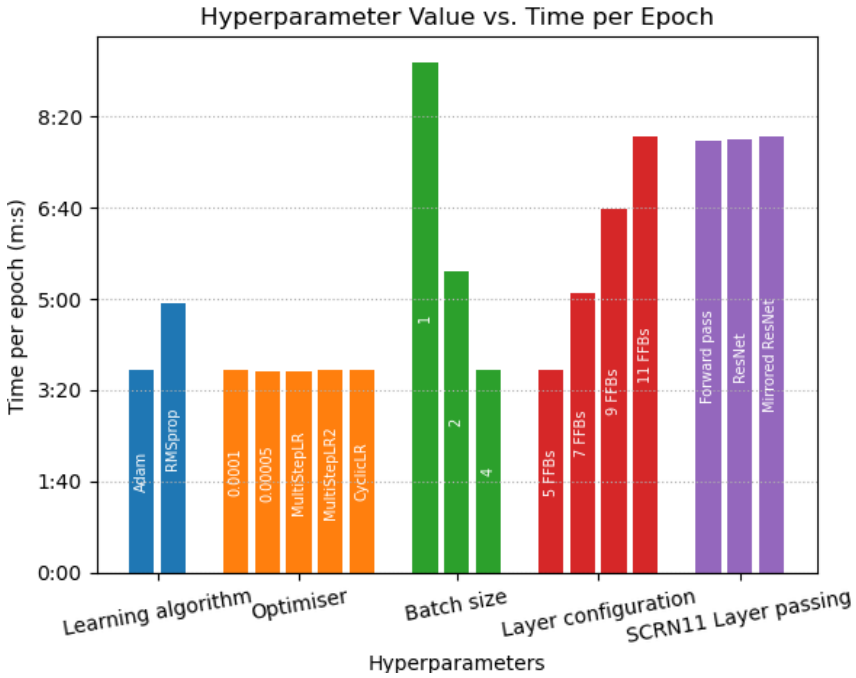


Figure 3.5: *Measurement of the training time taken per epoch for SCRN configured with specified hyper-parameters.*

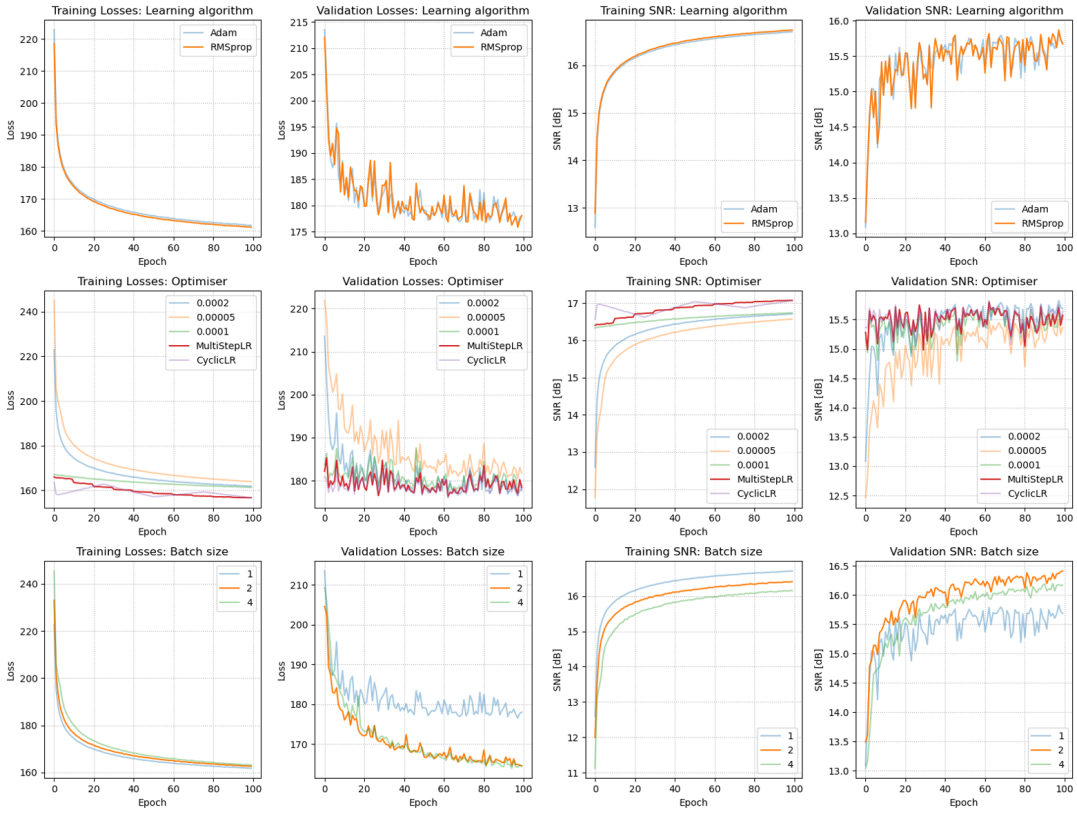


Figure 3.6: *The effects of hyper-parameters on SCRN training and validation. The parameters examined include optimisers (Adam, RMSprop), learning rates (0.0001, 0.0002, 0.00005, MultiStepLR, CyclicLR), and batch sizes (1, 2, 4). Each row corresponds to a different hyper-parameter; and each column corresponds to a validation metrics (training losses, validation losses, training SNR, and validation SNR) over 100 epochs.*

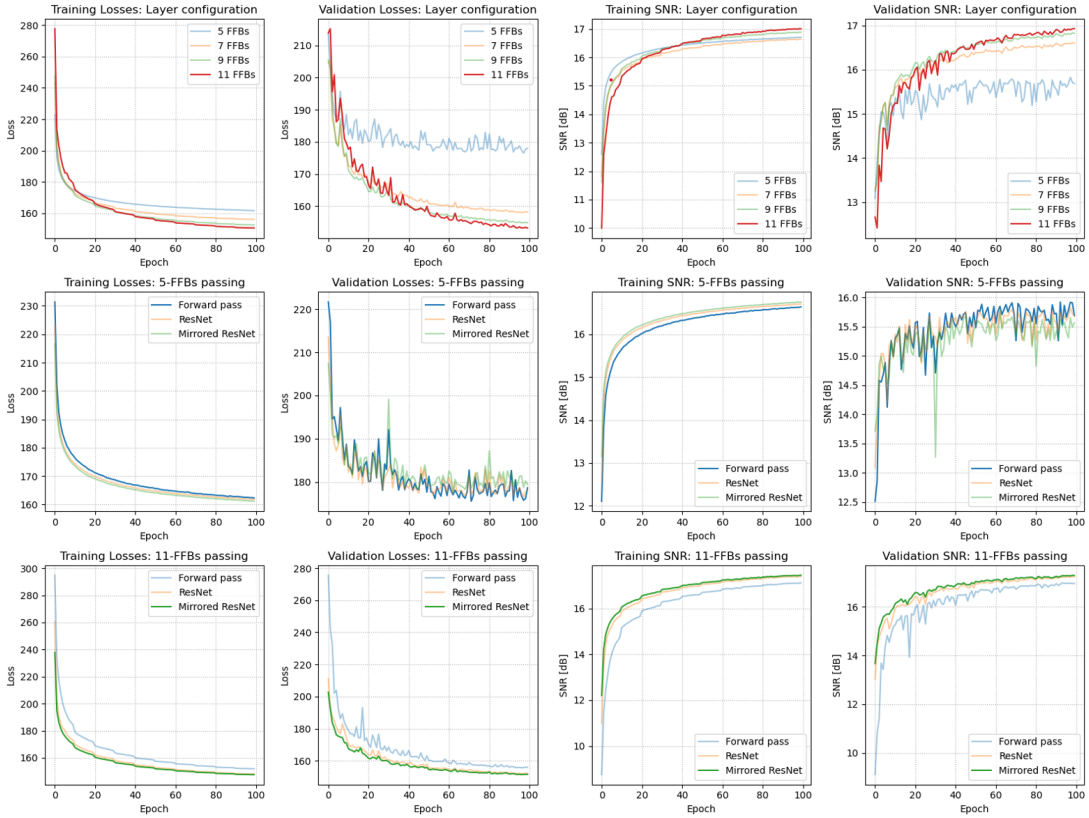


Figure 3.7: *The effects of block configuration on SCRN training and validation. The configurations examined include the number of FFBs (5, 7, 9, 11), block passing on 5-FFBs SCRN (Forward pass, ResNet, Mirrored ResNet), and block passing on 11-FFBs SCRN (Forward pass, ResNet, Mirrored ResNet). Each row corresponds to a different configuration; and each column corresponds to a validation metrics (training losses, validation losses, training SNR, and validation SNR) over 100 epochs.*

CHAPTER 4

Results

In this chapter, we present the MPFI-to-Pseudo-TDRI image translation results using our GAN, SCRN, and SCR-GAN models. The hyper-parameters selected for each model are summarised in Table 3.4. Ideally, the pseudo TDRI image generated should closely match its ground truth TDRI image. However, achieving perfect reconstruction is challenging due to finite computational resources. Even with infinite training epochs, the model might overfit the training set and fail to generalise to unseen validation and test data. Consequently, we limited the number of epochs for each model to 100, at which point the evaluation metrics (SNR and loss in training and validation dataset) showed stagnation in improvement. The figures presented in the model comparison Section 4.2 are based on the test set, which was another common receiver gather extracted from the same seismic volume.

4.1 Model performance in terms of evaluation metrics

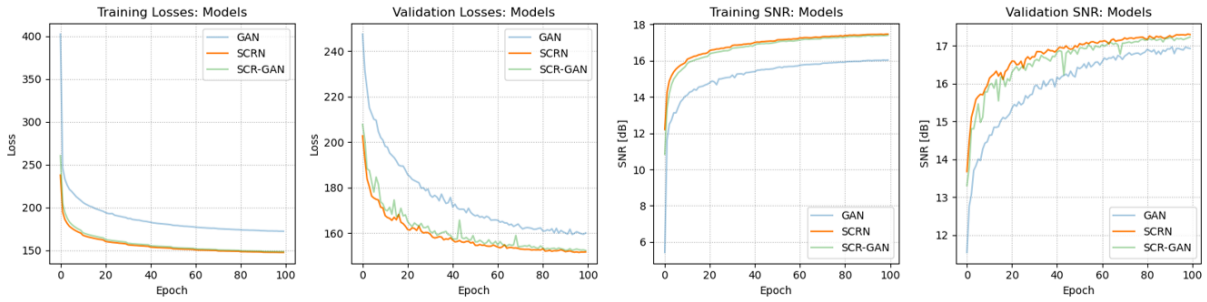


Figure 4.1: *Training loss, validation loss, training SNR and validation SNR over 100 epochs with GAN, SCRN, and SCR-GAN model.*

In Figure 4.1, we can clearly see the performance differentiates between the GAN and SCRN-based models. SCRN-based models exhibit a faster convergence trend, particularly in the validation metrics, with the slope of metrics levelling out around the 70th epoch. Table 4.1 shows that SCRN-based models have 14% lower training loss, 5%

lower validation loss, and 8% lower test loss compared to GAN. In terms of SNR, SCRN-based models outperform GAN by more than 1 dB in SNR for both the training and test sets. When comparing SCRN and SCR-GAN, SCRN very slightly leads by an average of 0.5% across all metrics. Additionally, Figure 4.1, indicates that SCR-GAN has less stability in the validation metrics compared to SCRN.

Based on both quantitative metrics and qualitative analysis from the plots, SCRN is the best model. It demonstrates the lowest losses and highest SNR across training, validation, and test sets. SCRN-based models exhibit superior performance, faster convergence, and greater stability compared to GAN.

	Training set		**Validation set**		**Test set**	
Model	**Loss**	**SNR [dB]**	**Loss**	**SNR [dB]**	**Loss**	**SNR [dB]**
GAN	172.35	16.02	159.99	16.92	180.33	15.19
SCRN	147.69	17.45	151.65	17.29	165.33	16.32
SCR-GAN	148.47	17.39	152.26	17.23	168.45	16.25

Table 4.1: *Model performance on training, validation, and test sets in terms of normalised loss and SNR. For the GAN-based model, the loss here refers to the generator loss. The training and validation metrics are based on the final epoch, while the test metrics are evaluated on the fully trained model.*

4.2 Image translation results

While the losses and SNR metrics indicate the image translation ability on individual patches, it is important to assess the overall 2D slice translation results. In seismic interpretation, the focus is on the signals across the entire 2D slice rather than on small 256 x 256 patches. This holistic view of seismic slices offer a comprehensive representation of large-scale subsurface structures and stratigraphy, enabling geophysicists to interpret geological features, fault lines, and other critical subsurface anomalies. Therefore, we group the patches processed by the models back into slices, ensuring 50% overlap in both x and y directions (Figure 4.2). Overlapping patches can maintains signal continuity and reduces edge effects, resulting in a smoother transition between patches and a more coherent reconstruction of the 2D slice. In this section, we will analyse the image translation results for unseen MPFI slices taken from another common receiver gather, evaluating their pseudo-TDRI reconstruction by comparing them to the target TDRI values globally and regionally.

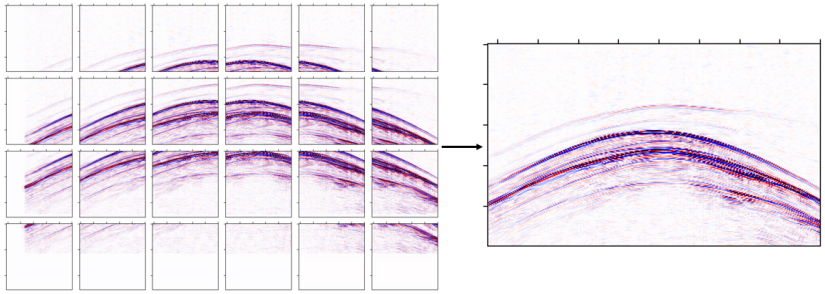


Figure 4.2: *Regrouping the translated 256 x 256 patches into a 2D slice with 50% overlapping.*

4.2.1 Global result evaluation

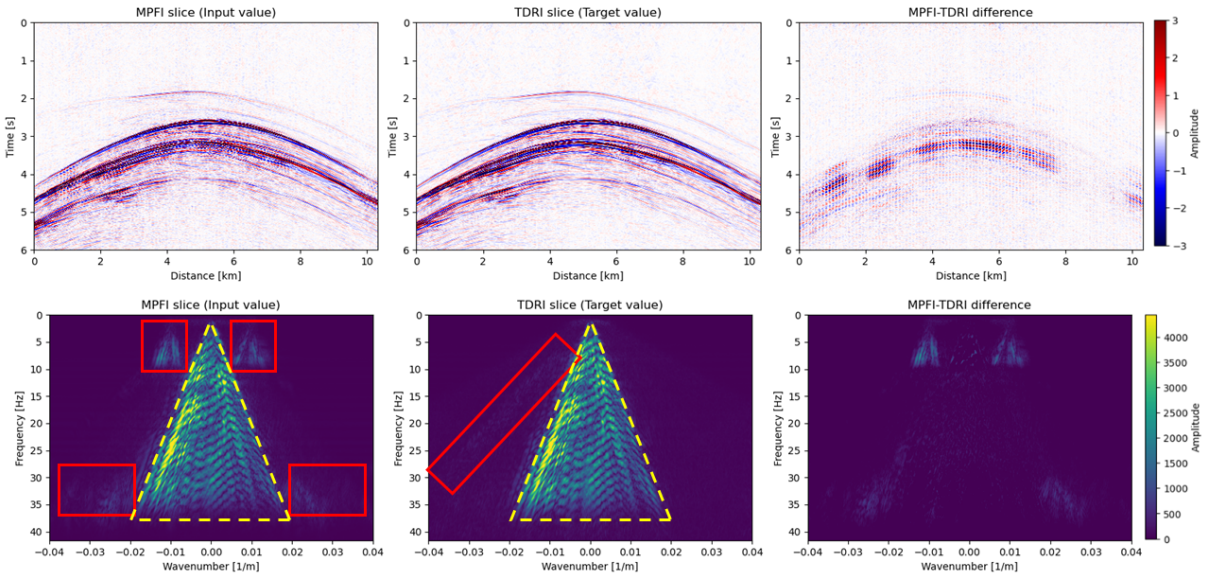


Figure 4.3: *2D seismic slices in the TX domain: The MPFI slice (top left) serves as the input, the TDRI slice (top middle) is the reference, and the difference between MPFI and TDRI slices (top right). 2D seismic slices in the FK domain: The MPFI slice (bottom left), the TDRI slice (bottom middle), and the difference between MPFI and TDRI slices (bottom right). The yellow cone represents the region where our signal exists, while the red boxes indicate areas where noises can appear in the spectrum.*

Before evaluating the quality of the reconstructed images, it is essential to understand the input MPFI slice and the target TDRI slice. As an industry-standard interpolator, MPFI produces results that, while not achieving the same quality as the high-cost TDRI solution, are nevertheless close (see Figure 4.3). In the FK domain, the two curved events in the TX domain are represented by a range of dips, appearing as large dispersive cones in both the MPFI and TDRI slices, highlighted in yellow in Figure 4.3. Even at the large scale shown in Figure 4.3, we can appreciate that the MPFI slice contains significant inherent noises (see red boxes), such as symmetrical cones at lower frequencies due to aliasing and high-frequency artifacts on both sides of the main signal, are not suppressed from the interpolation process. Conversely, TDRI proves to be a superior interpolation method, effectively eliminating these noises and leaving only minimal traces at negative wavenumbers. Ideally, anything outside the main signal cone should have zero amplitude, constrained by the compressional wave speed in water at 1450 m/s. However, these noises outside the main signal cone are not a primary concern as they can be easily removed using dip-dependent filters in seismic processing. In the following discussion, we will focus on residuals within the cone region.

Figure 4.4 displays the image translation results in both domains. All three models reconstructed high resemblance TDRI image. However, the GAN model introduced a slight artifact, resulting as a bluish hue between distances of 4-6 km shown in the TX domain. In contrast, both SCRN-based models produced a background that was paler than the TDRI reference in TX domain (see Figure 4.3(top)), suggesting that these models not only retained the reflection signals but also effectively performed de-noising.

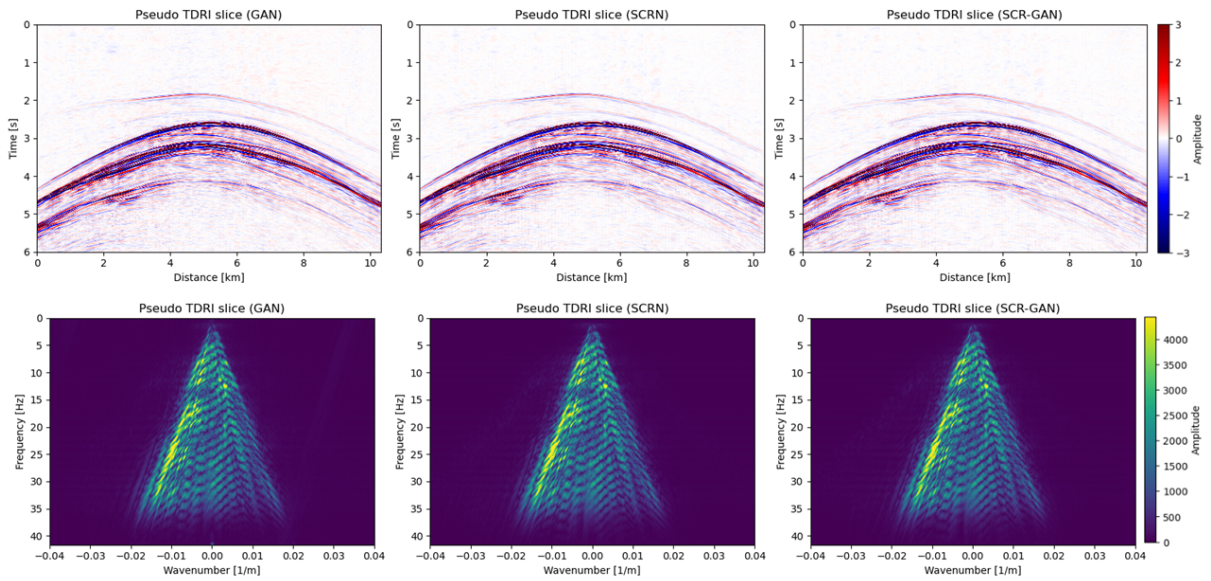


Figure 4.4: *Reconstructed slices in the TX domain: Pseudo TDRI slices produced by GAN (top left), by SCRN (top middle), and by SCR-GAN (top right). Reconstructed slices in the FK domain: Pseudo TDRI slices produced by GAN (bottom left), by SCRN (bottom middle), and by SCR-GAN (bottom right).*

Since all reconstructions display clear texture and event continuity, we next evaluate model performance based on residuals. Figure 4.5 illustrates the differences between the pseudo-TDRI slices and the ground truth. Note that the amplitude range is smaller than that of the previous plot to emphasise the residuals. In the TX domain, the GAN model exhibits the highest residuals, concentrated around the two main events. SCRN and SCR-GAN also show residuals at these locations, but with lower amplitude. The presence of residuals could be attributed to the limited coverage in our training data (see Figure 3.1), which may have prevented the models from learning events beyond t=4.6s. However, most residuals are observed before this time, suggesting that our models effectively captured the essence of seismic traces and generalised well to unseen parts of the image. The blue hue shown in the GAN residual background is an ultra-low frequency artifact (indicated with an arrow in Figure 4.5). Moving on to the FK domain, we see that the pseudo-TDRI slice generated by GAN shows significant residuals (highlighted in red boxes in Figure 4.5), which match the dip of the primary energy signal (yellow band in Figure 4.4), indicating the presence of spatial aliasing. This artifact partially overlaps with our signal cone (wavenumber -0.02 to 0.02 [1/m]), which might cause distortion. On the other hand, the SCRN-based models removed a noise band (orange boxes in Figure 4.5) that is present in the original TDRI slice (red box in Figure 4.3(bottom middle)), once again proving their noise reduction capabilities.

4.2.2 Regional result evaluation

A more detailed analysis of the results in smaller patches allows a better understanding of the image translation performance. To further differentiate the quality between SCRN and SCR-GAN reconstructions, we analyse the residuals in patches that capture the strongest events. Figure 4.6 presents the residuals and SNR for each patch, the first row of displays patches produced by GAN. Patch (a) is an edge patch with noticeable smearing noises

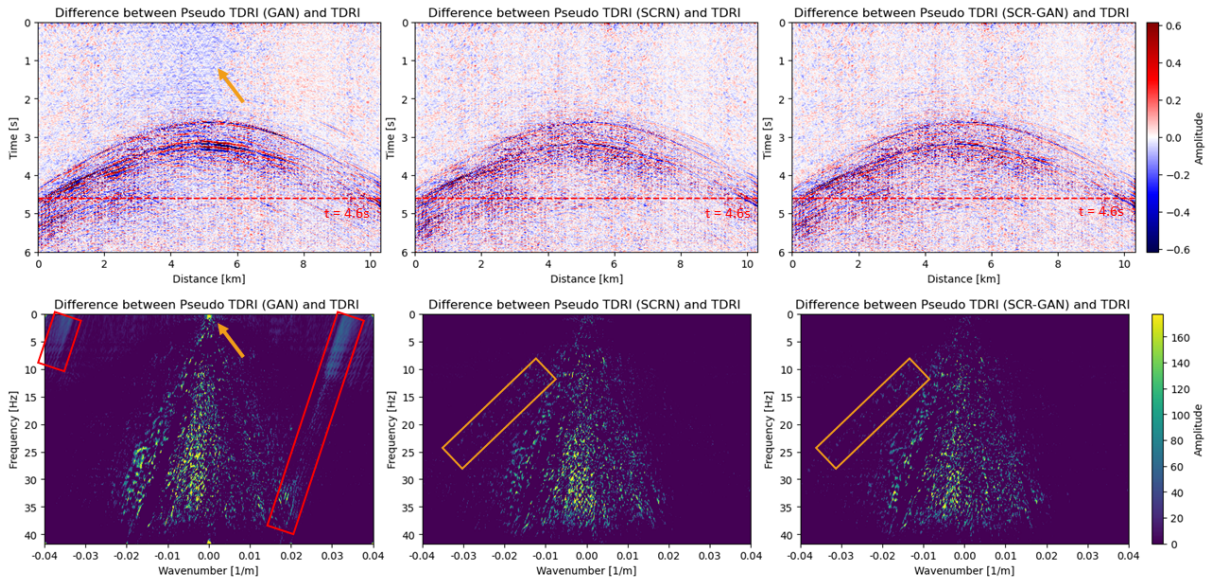


Figure 4.5: *Residual in the TX domain: Difference between the ground truth TDRI slice and the pseudo TDRI slices produced by GAN (top left), by SCRN (top middle), and by SCR-GAN (top right). Residual in the FK domain: Difference between the ground truth TDRI slice and the pseudo TDRI slices produced by GAN (bottom left), by SCRN (bottom middle), and by SCR-GAN (bottom right). Note that the amplitude range is reduced by 80% to better visualise the low amplitude residuals.*

in pink, while center patches, particularly (d), exhibit a red hue, indicating that GAN introduced a substantial noise in that area during reconstruction. These patches show an SNR that is 2 units lower than those produced by the SCRN-based models. In contrast, SCRN and SCR-GAN generated nearly identical images (second and third row in Figure 4.6), with SCRN performing slightly better, averaging +0.1 dB SNR. Their residual is less coherent than GAN. The key differences are observed in patches (c) and (d), where SCR-GAN has higher residuals from the second event, with SNRs of -0.4 dB and -0.2 dB compared to SCRN. This suggests that SCR-GAN shares the same weakness as GAN in reconstructing regions with the highest amplitude.

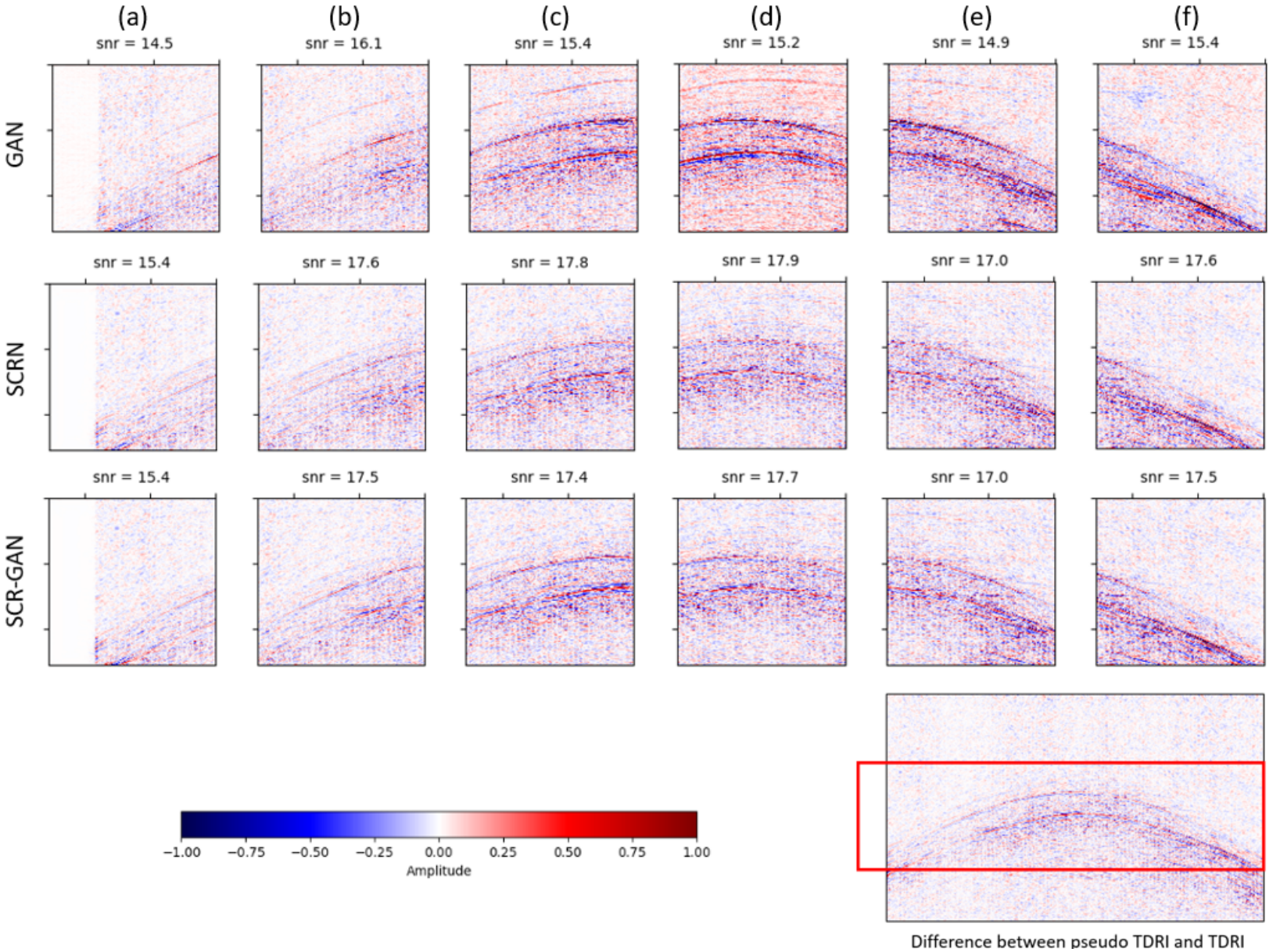
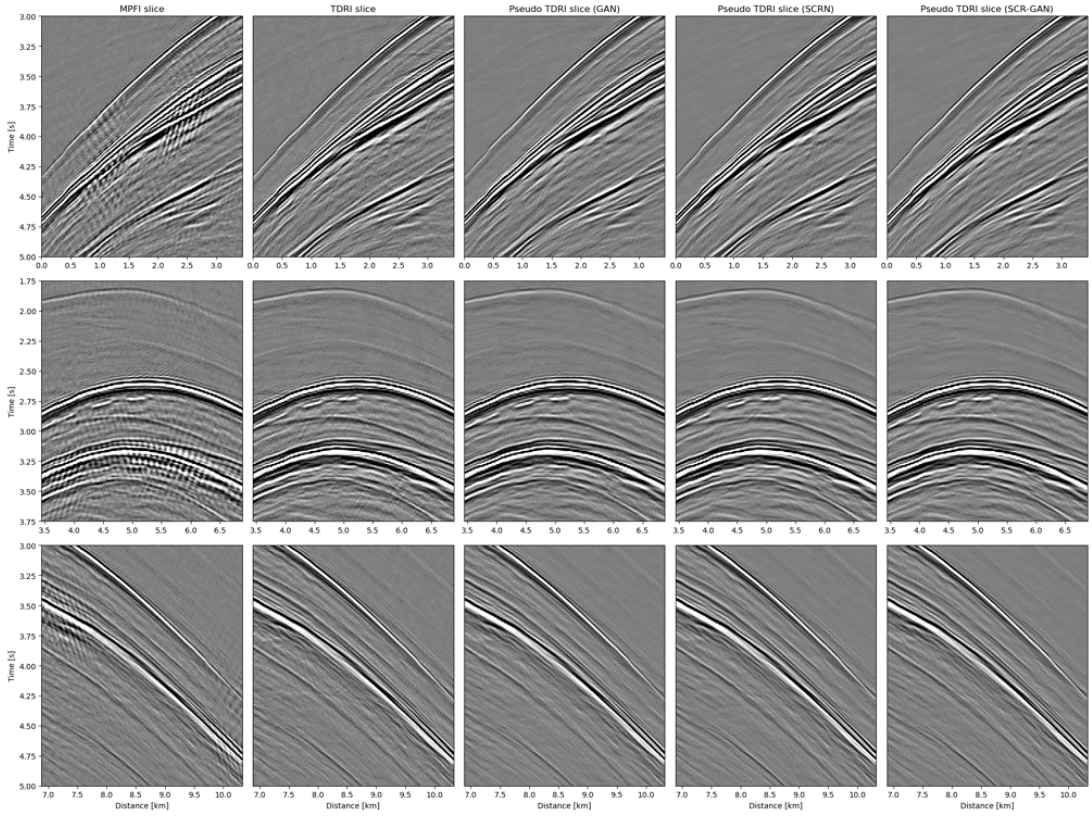


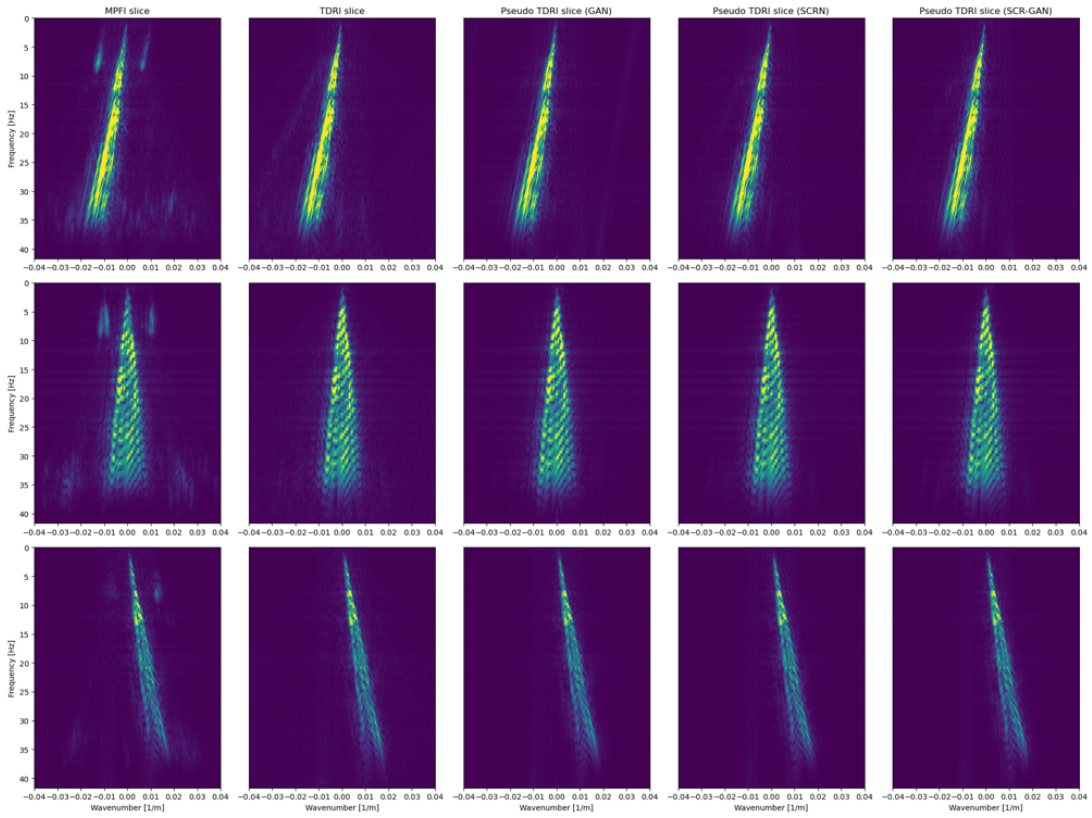
Figure 4.6: *Difference between the TDRI patches (ground truth) and the pseudo-TDRI patches generated by GAN (top row), SCRN (middle row), and SCR-GAN (bottom row). The six patches in each row are extracted from t=1.9-3.8s and x=0-10.33km, as depicted by the red box in the bottom right slice. The unit of SNR here is in dB. The colour limits (amplitude range) are set from -1 to 1 to highlight the residual.*

Next, we perform a regional analysis along the main signal in both domains. In Figure 4.7, the two major events are divided into three segments of equal distance (3.44 km), each featuring 2 seconds of data. On the MPFI slices, noticeable ripple-like noise appears in the TX domain, particularly in the near offset, leading to additional noise emerging within the signal cone area in the FK domain. The TDRI slices exhibit coherence in the signal but include more texture below the main reflection. These low-amplitude cross-mark texture in the TX domain may indicate either noise or signal. The pseudo-TDRI patches generated by all three models show a high degree of similarity to the target TDRI values in both the TX and FK domains. In the TX domain, these reconstructions are smoother, particularly below the main signal, where some texture is lost compared to TDRI. In the FK domain, the de-noising effect is evident as the areas outside the main signal band approach zero, resulting in less noise compared to the ground truth TDRI.

The residual plot in Figure 4.8 shows that the MPFI-TDRI difference in the TX domain contains ripple-like noise, which is effectively eliminated in all pseudo-TDRI residuals. Recovering the very evident coherent seismic events that was lost in MPFI in the Figure 4.8(first column) is the biggest achievement of our deep learning models. At the first location (Figure 4.8(top row)), cross marks are evident in the bottom right corners of the reconstructed TDRI residual patches. Among the models, GAN exhibits the highest residual amplitude, a pattern consistent across other locations. GAN TDRI residuals are also less coherent than those of the other models. In the TX domain, residuals from all three models still display remnants of the signal curve, indicating that some signal has been removed along with the noise. In the FK domain, residuals within the signal cone (wavenumber -0.02 to 0.02 [1/m]) suggest partial signal removal. GAN shows the strongest residual amplitude in this domain as well. Additionally, artifacts from aliasing, highlighted in red boxes in Figure 4.5, are further localised in the near offset plot. Each model exhibits slight differences in performance based on residual amount at various locations. GAN has the highest residual at the first location but the lowest at the third. SCRN and SCR-GAN perform similarly, with nearly identical residuals. Overall, despite small level of signal removal, we appreciate the model performance in terms of the reduction of the alias and noises within the MPFI the signal cone.

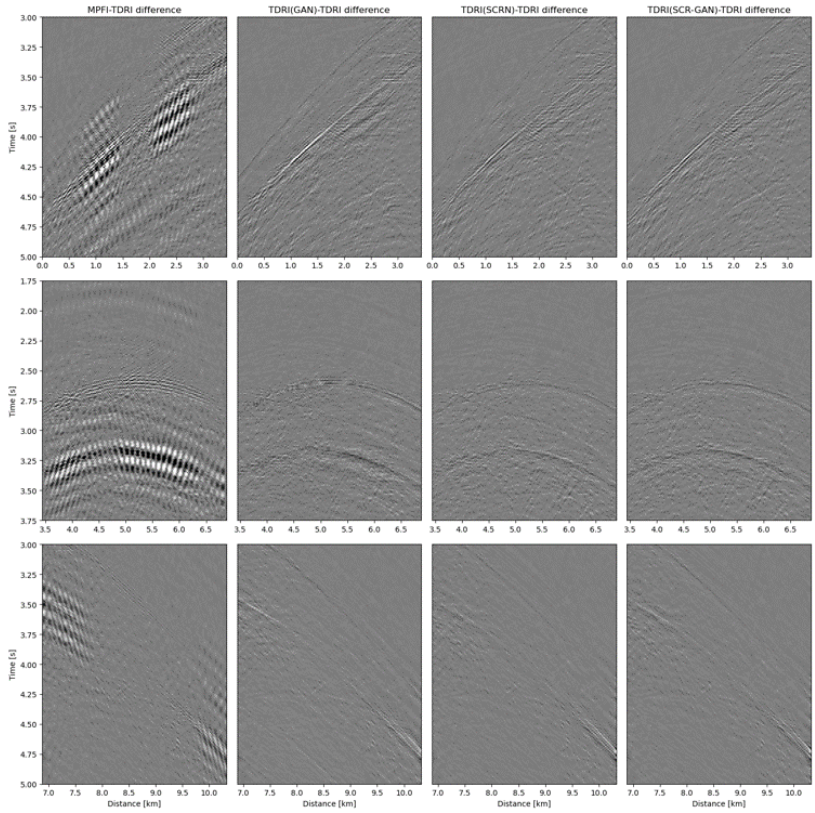


(a) Slices in the TX domain

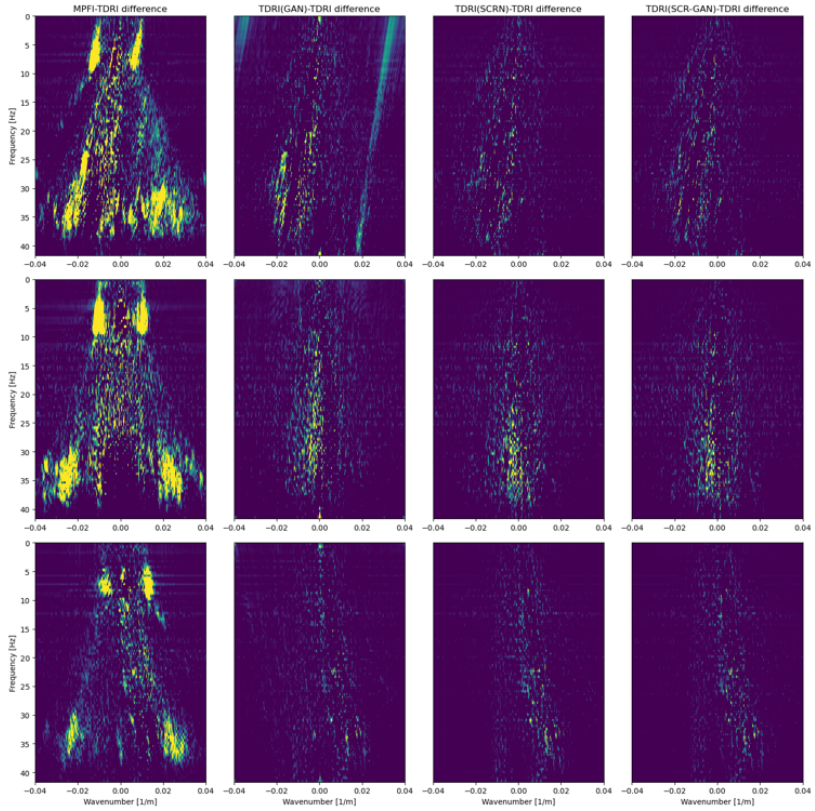


(b) Slices in the FK domain

Figure 4.7: MPFI patches, TDRI patches, Pseudo TDRI patches generate by GAN, SCRN and SCR-GAN (left to right) featuring different time-distance ranges in TX domain and the FK domain. The ranges from top to bottom are: t=3-5s, x=0-3.44km; t=1.75-3.75s, x=3.44-6.88km; t=3-5s, x=6.88-10.33km.



(a) Residual in the TX domain



(b) Residual in the FK domain

Figure 4.8: MPFI-TDRI difference, Pseudo TDRI-TDRI difference reconstructed by GAN, SCRN and SCR-GAN (left to right) featuring different time-distance ranges in TX domain and the FK domain. The ranges from top to bottom are: t=3-5s, x=0-3.44km; t=1.75-3.75s, x=3.44-6.88km; t=3-5s, x=6.88-10.33km.

CHAPTER 5

Discussion

5.1 Comparison of GAN, SCRN and SCR-GAN

Based on our evaluation metrics and slice reconstruction performance, SCRN stands out as the most effective model. SCRN-based models deliver a higher SNR than GANs because they excel in signal retention while performing de-noising. This advantage is evident in Figure 4.3, where SCRN-based pseudo TDRI slices exhibit less background noise compared to the ground truth, which is unexpected since our model aims to replicate the ground truth. This suggests that SCRN's de-noising capability might even surpass that of the TDRI.

Conversely, GAN introduce significant low-frequency noise (<1Hz) into the reconstructed images, as illustrated in Figure 4.8 and the histogram in Figure 5.1. All three models show greater discrepancies from the ground truth TDRI at higher frequencies compared to lower frequencies (Figure 5.1), leading to some residuals at key seismic event positions. We plan to address this issue by employing transfer learning—loading the trained model and continuing training with a dataset that emphasises high frequencies (see Section 5.4.1 for more details). Nonetheless, SCRN consistently translates images with the least residuals across all frequencies, establishing it as the most robust and accurate model among the three.

5.2 Computational time improvement

Our best model, SCRN, required 5.4 times more computational resources for training compared to GAN, and 5 times more for model application. This is because SCRN is a deeper model that involves 3 times the total multiply-accumulate operations (MACs) compared to GAN (see Figure A.1 and A.2 in the Appendix). Note that our model is trained on two common receiver gathers (MPFI-TDRI pair), and the image translation is applied on thousands of gathers with the trained model. Since the model training is a one-time process, this cost is incurred only once. For subsequent image translation, we consider only the 3D MPFI computational cost (13 hours) plus the application time. As the image translation time for thousands of 3D common receiver gather with dimensions

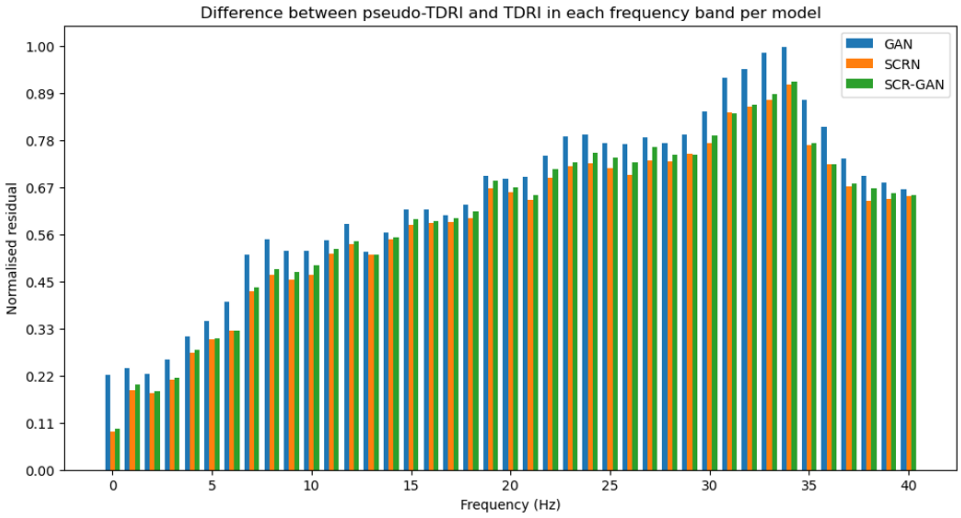


Figure 5.1: *Image reconstruction residual per frequency bins.*

Model	Model training time	Image translation time	Total time (excluding MPFI)
GAN	4h19m	2m	4h21m
SCRN	23h25m	11m	23h36m
SCR-GAN	35h30m	11m	35h41m

Table 5.1: *GPU time to train and apply the model on a 3D common receiver gather of the same dimension [t,x,y]=[500,601,826].*

[t,x,y]=[500,601,826] is less than one-fifth of an hour on a GPU, the total computational time equals to 13 CPU hours plus 0.2 GPU hour. Note that MPFI and TDRI are measured in CPU hours, while our method is measured in GPU hours. There is no direct conversion between CPU and GPU hours, as it depends on the computer model. Nevertheless, it is clear that our method is significantly faster, by nearly four orders of magnitude, compared to the 127,294 CPU hours required for 3D TDRI on thousands of common receiver gathers (Figure 5.2).

5.3 Limitations

Despite SCRN's high-quality image translation capabilities and relatively low computational requirements, the method has limitations due to inherent noise in the dataset and the limited amount of training data.

5.3.1 Garbage in, garbage out

Deep learning relies heavily on accurate datasets to produce satisfactory results. In our study, the raw dataset D fed into Omega (a seismic processing software) contains n traces. MPFI and TDRI performed 3D interpolation from a 50x100m grid to a 12.5x12.5m grid, effectively adding 7 extra traces between the measurement traces in our crossline slices. However, this interpolation can introduce artifacts into the data. Ideally, the difference $\text{MPFI}(D) - \text{TDRI}(D)$ should be zero for every 8th trace (column) as they are at the positions where data were actually measured. Figure 5.3 reveals inherent noise in our TDRI and MPFI datasets, as the difference every

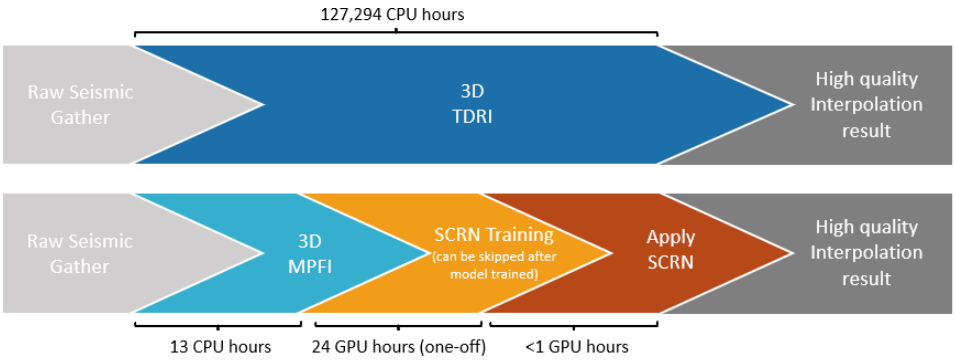


Figure 5.2: *Computational time of TDRI and low-to-high-quality seismic image translation.*

8th trace contains residuals. While all traces exhibit some level of noise, those at these specific positions have minimal noise and are closest to the actual measurements. Given that our training dataset contains noise, the quality of the generated images is influenced by the quality of the dataset, causing the model to learn both the underlying signals and noises.

5.3.2 Limited data, limited generalisation

Performing 3D TDRI for all common receiver gathers over an area of 3700 km^2 requires 127,294 CPU hours, and only 13 CPU hours for 3D MPFI. Despite running these interpolations in parallel, generating TDRI-MPFI pairs for training is extremely computationally expensive. While MPFI data is abundant and covers various subsurface types, our supervised training method requires both TDRI and MPFI datasets. Currently, our model is trained solely on data from the Gulf of Mexico. To enhance the robustness of our model, we need datasets from diverse locations to handle a broader range of seismic reflection patterns. However, the difficulty in obtaining these datasets, especially for TDRI data, poses a significant challenge to achieving model generalisation.

One potential solution is to re-train the network for each project. This approach would involve selecting a representative subset of data, as small as possible, to perform both MPFI and TDRI. While this would increase costs, the use of a smaller training and validation subset could still result in significant savings.

5.4 Future research

5.4.1 Improvement on the SCRN model

Weighting key traces during model training

When training our SCRN model, we initially assigned equal weighting to all traces (columns) in our patches. However, as discussed in Section 5.3.1, the positions of the originally measured traces interpolated by MPFI (and TDRI) contains less noise. Assigning higher weighting to these less noisy traces could improve image translation results by guiding the model to focus and learn from the most accurate parts of the image.

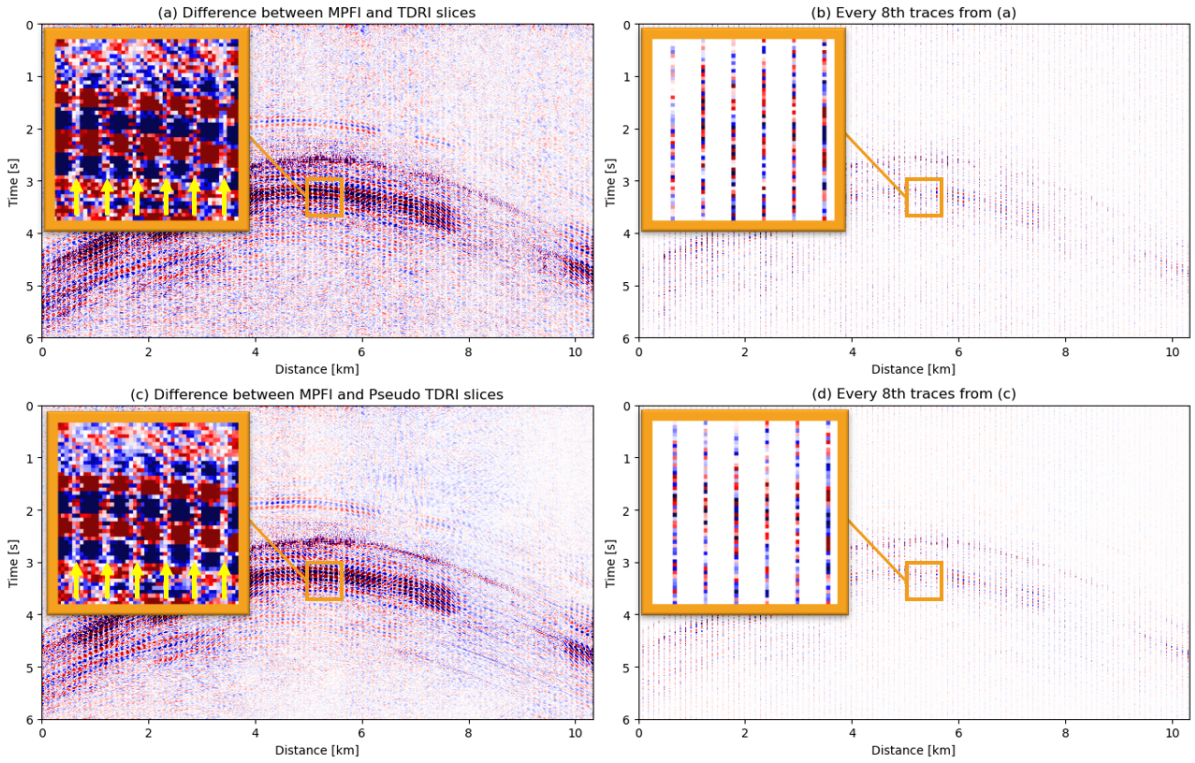


Figure 5.3: *Inherent noise in training datasets can lead to artifacts in the generated images. (a) Displays the difference between MPFI and TDRI slices; (b) Shows every 8th trace from the MPFI-TDRI difference; (c) Illustrates the difference between MPFI and Pseudo-TDRI(SCRN) slices; (d) Shows every 8th trace from the MPFI-Pseudo-TDRI(SCRN) difference. The orange boxes zoom into the area between* $t = 240 - 290$ *and* $x = 400 - 450$*, with yellow arrows highlighting residuals at every 8th traces caused by noise in the datasets.*

Expanding the training data with common receiver gathers

Despite our 5D dataset being collected from a single location, we can generate thousands of 3D seismic volumes by selecting different common receiver locations. Initially, we trained our SCRN model using only two common receiver gathers (one MPFI and one TDRI). To enhance model generalisation, we can resume training by loading the pre-trained SCRN model and feeding it patch pairs extracted from the remaining common receiver gathers.

Increasing dataset variety through frequency augmentation

Another effective way to expand the variety of our existing dataset is through image augmentation. By applying frequency filters to our broadband data—such as 0-5 Hz, 5-10 Hz, 10-25 Hz, and >25 Hz (see Figure 5.4)—we can increase the dataset size fivefold, as each frequency band produces an additional seismic volume. It is crucial to adjust the normalisation weighting for each frequency band to ensure consistent scaling across the data. This prevents issues where certain frequency components might dominate due to their larger magnitude, thereby maintaining balanced input for the model.

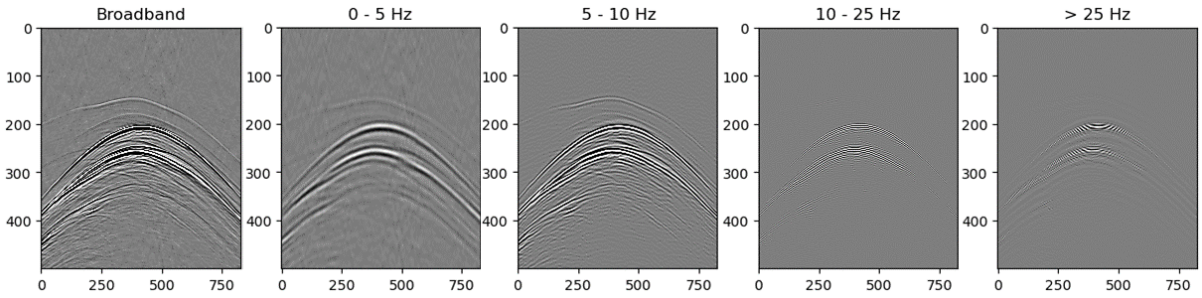


Figure 5.4: *Using frequency filters to create 5-times the dataset.*

5.4.2 Low-to-high-quality seismic image translation on other applications

In this thesis, we have demonstrated the feasibility of low-to-high-quality seismic image translation using the MPFI-TDRI datasets. The ultimate objective is to apply this two-step workflow to create various high-quality seismic solutions at a lower cost, as illustrated in Figure 1.2 in Chapter 1.

Extending beyond the interpolation applications, we explored this workflow on seismic deconvolution. Deconvolution of upgoing and downgoing wavefields is a technique used to remove the water column effect from ocean-bottom node (OBN) seismic data, thereby attenuating free-surface multiples (Y. Wang, Grion, & Bale, 2009) and providing an estimate of the Green's function (subsurface reflectivity). A technology called Up-down deconvolution (UDD) effectively produces optimal Green's function for flat seabeds and mildly varying subsurface structures. However, UDD becomes unstable in the presence of seabed tilts with sharp lateral variations, such as faults (Amundsen, 2001; Y. Wang et al., 2009; Boiero & Bagaini, 2021). A more general approach, multi-dimensional deconvolution (MDD), can overcome these limitations for any geological scenario, stabilising the estimation of Green's function (Wapenaar et al., 2011). Unfortunately, MDD involves severely higher computational cost and typically requires more complex flows.

In OBN scenarios, data is often acquired with sparser receiver spacing to reduce costs. To enable integration along receivers, interpolation is necessary. MDD requires stable interpolation across sparsely sampled receivers, significantly increasing the computational burden and complexity compared to UDD, which operates on common receiver gathers without such interpolation (Wapenaar et al., 2011; Ravasi, Meles, Curtis, Rawlinson, & Yikuo, 2015). Therefore, if we can use deep learning to translate UDD results to MDD outputs, it would substantially reduce the associated costs.

We experimented training the SCRN model (Figure 2.8) with selected parameters (Table 3.4) on the UDD-MDD dataset. By applying the same workflow but with a different dataset (see Figure 5.5), we translated UDD images to their MDD counterparts.

Our preliminary results, shown in Figure 5.6, indicate that while the resemblance of the reconstructed MDD slice is not exceptionally high due to non-optimised model parameters for the UDD-MDD purpose, the approach shows potential.

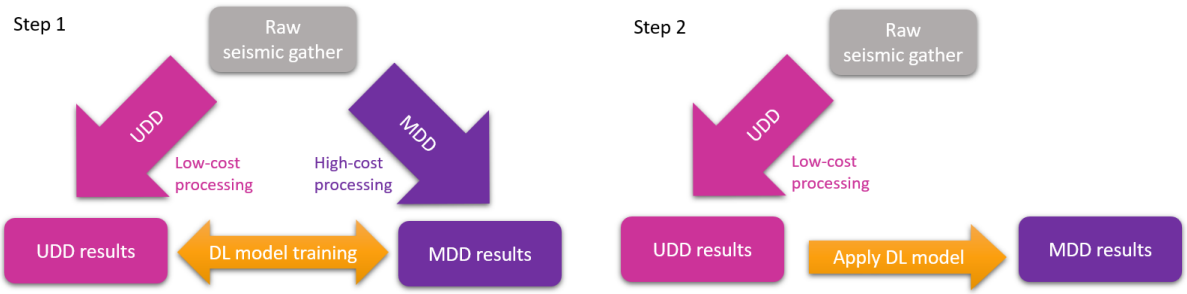


Figure 5.5: *2-Step deep learning workflow for UDD-MDD seismic image translation.*

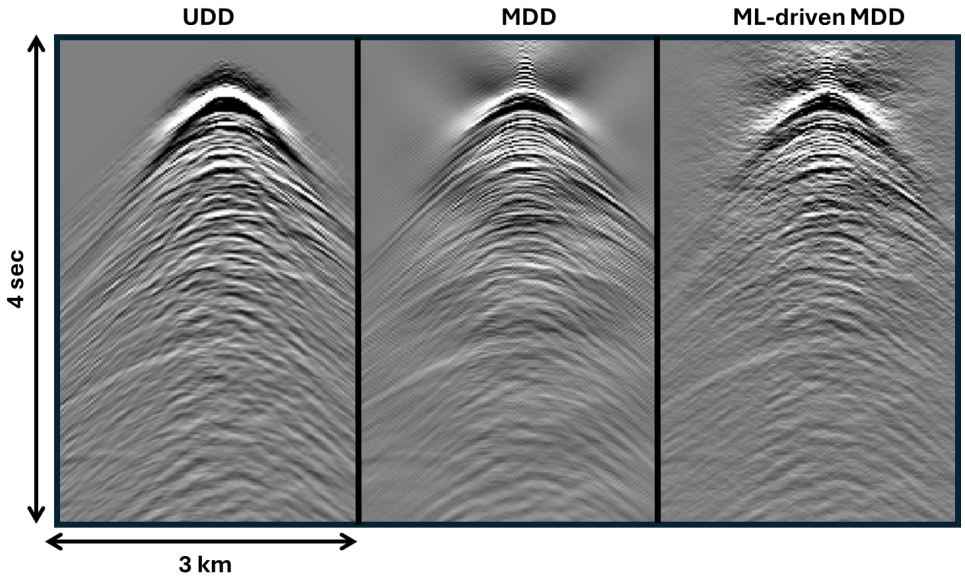


Figure 5.6: *UDD slice (input), MDD slice (reference) and Pseudo MDD slice (reconstructed using deep learning).*

From both the UDD-MDD and particularly MPFI-TDRI image translation results, this approach can be extended to other seismic applications upon further research, offering a efficient solution for enhancing seismic image quality. With optimisation and refinement of model parameters, this workflow could revolutionise the processing seismic data, providing more accurate and cost-effective solutions in the field of geophysics.

CHAPTER 6

Conclusion

Seismic processing technologies offer a range of options for addressing a given problem, typically presenting a choice between low-cost, low-resolution solutions and high-cost, high-resolution solutions. With advancements in deep learning, users now have a third option: mapping low-quality data to high-quality results. In this thesis, we have explored and optimised three models, namely GAN, SCRN and SCR-GAN, for the MPFI-to-TDRI seismic image translation purpose. These models effectively produced high-quality TDRI images. Based on our evaluation metrics and slice reconstruction performance, SCRN stands out as the most effective model and its de-noising capabilities, potentially surpassing TDRI itself. Once trained, the SCRN model could produce TDRI quality result through the low-to-high quality translation workflow, with computational costs comparable to MPFI processing.

Is there a way do high-quality seismic interpolation fast and cheap? Yes, there is.

While the performance is satisfactory, our current model is trained exclusively on Gulf of Mexico data, limiting its generalisation. To enhance robustness, future work should involve expanding the training dataset with diverse locations and common receiver gathers, as well as augmenting data through frequency filtering. This low-to-high quality image translation workflow can be extended to other seismic applications, such as deconvolution, by using pairs of low-cost and high-cost processing results.

APPENDIX A

Appendix

A.1 Data management plan

The MPFI and TDRI datasets used in this study were provided by SLB and are confidential. The datasets are obtained using Omega, a SLB in-house processing software, on a standard 20 core 125GB memory CPU node. Both the datasets and code are managed under version control using Azure DevOps. Deep learning was performed using PyTorch (v2.3.1) and Python (v3.11.7) on hardware equipped with an NVIDIA A100-SXM4-80GB GPU, with CUDA version 12.2 and driver version 535.161.08.

A.2 Model architecture

References

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., … Farhan, L. (2021). Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, *8*, 1–74.

Amundsen, L. (2001). Elimination of free-surface related multiples without need of the source wavelet. *Geophysics*, *66*(1), 327–341.

Bagchi, S., & Mitra, S. K. (1996). The nonuniform discrete fourier transform and its applications in filter design. i. 1-d. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *43*(6), 422–433.

Beylkin, G. (1987). Discrete radon transform. *IEEE transactions on acoustics, speech, and signal processing*, *35*(2), 162–172.

Bilsby, P., Kumar, R., Vassallo, M., & Zarkhidze, A. (2023). Multistage matching pursuit fourier interpolation with physics-based priors. In *84th eage annual conference & exhibition* (Vol. 2023, pp. 1–5).

Boiero, D., & Bagaini, C. (2021). Up and down deconvolution in complex geological scenarios. *Geophysics*, *86*(5), WC55–WC65.

Chen, X., Yu, R., Ullah, S., Wu, D., Li, Z., Li, Q., … Zhang, Y. (2022). A novel loss function of deep learning in wind speed forecasting. *Energy*, *238*, 121808.

Claerbout, J. F. (1976). *Fundamentals of geophysical data processing* (Vol. 274). Citeseer.

Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2020). A survey of deep learning and its applications: a new paradigm to machine learning. *Archives of Computational Methods in Engineering*, *27*, 1071–1092.

Diebold, J. B., & Stoffa, P. L. (1981). The traveltime equation, tau-p mapping, and inversion of common midpoint data. *Geophysics*, *46*(3), 238–254.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., … others (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Gan, S., Wang, S., Chen, Y., Zhang, Y., & Jin, Z. (2015). Dealiased seismic data interpolation using seislet transform with low-frequency constraint. *IEEE Geoscience and remote sensing letters*, *12*(10), 2150–2154.

Gao, L., Shen, H., & Min, F. (2024). Swin transformer for simultaneous denoising and interpolation of seismic data. *Computers & Geosciences*, *183*, 105510.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative adversarial nets. *Advances in neural information processing systems*, *27*.

Gu, Y. J., & Sacchi, M. (2009). Radon transform methods and their applications in mapping mantle reflectivity structure. *Surveys in geophysics*, *30*, 327–354.

Guo, Y., Fu, L., & Li, H. (2023). Seismic data interpolation based on multi-scale transformer. *IEEE Geoscience and Remote Sensing Letters*.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 770–778).

Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 1125–1134).

Kaur, H., Pham, N., & Fomel, S. (2019). Seismic data interpolation using cyclegan. In *Seg technical program expanded abstracts 2019* (pp. 2202–2206). Society of Exploration Geophysicists.

Kaur, H., Pham, N., & Fomel, S. (2021). Seismic data interpolation using deep learning with generative adversarial networks. *Geophysical Prospecting*, *69*(2), 307–326.

Khosro Anjom, F., Vaccarino, F., & Socco, L. V. (2023). Machine learning for seismic exploration: Where are we and how far are we from the holy grail? *Geophysics*, *89*(1), WA157–WA178.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

Kumar, A., Dancer, K., Rayment, T., Hampson, G., & Burgess, T. (2022). Deep learning swell noise estimation. *First Break*, *40*(9), 31–36.

Lau, Y. T. R. (2024, February 1). *Image reconstruction using generative adversarial networks* (Research Module Report). Zürich, Switzerland: ETH Zürich. (Supervised by Dr. Rajiv Kumar, Dr. Massimiliano Vassallo, and Dr. Dirk-Jan van Manen)

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., … others (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4681–4690).

Li, J., Yan, Y., Liao, S., Yang, X., & Shao, L. (2021). Local-to-global self-attention in vision transformers. *arXiv preprint arXiv:2107.04735*.

Li, S., Liu, B., Ren, Y., Chen, Y., Yang, S., Wang, Y., & Jiang, P. (2019). Deep-learning inversion of seismic data. *arXiv preprint arXiv:1901.07733*.

Liu, B., & Sacchi, M. D. (2004). Minimum weighted norm interpolation of seismic records. *Geophysics*, *69*(6), 1560–1568.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., … Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the ieee/cvf international conference on computer vision* (pp. 10012–10022).

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3431–3440).

Nabil, M. (2021). *Unpacking query, key, and value in transformers: An analogy with databases.* https://www.linkedin.com/pulse/unpacking-query-key-value-transformers-analogy-database-mohamed-nabil. (Accessed: 2024-07-23)

Naghizadeh, M. (2012). Seismic data interpolation and denoising in the frequency-wavenumber domain. *Geophysics*, *77*(2), V71–V80.

NVIDIA. (2020). *Nvidia ampere architecture in-depth.* Retrieved from https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/ (Accessed: 2024-07-18)

Oliveira, D. A., Ferreira, R. S., Silva, R., & Brazil, E. V. (2018). Interpolating seismic data with conditional generative adversarial networks. *IEEE Geoscience and Remote Sensing Letters*, *15*(12), 1952–1956.

Oppenheim, A. V. (1999). *Discrete-time signal processing*. Pearson Education India.

O'shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Ravasi, M., Meles, G., Curtis, A., Rawlinson, Z., & Yikuo, L. (2015). Seismic interferometry by multidimensional deconvolution without wavefield separation. *Geophysical Journal International*, *202*(1), 1–16.

Ronen, J. (1987). Wave-equation trace interpolation. *Geophysics*, *52*(7), 973–984.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–miccai 2015: 18th international conference, munich, germany, october 5-9, 2015, proceedings, part iii 18* (pp. 234–241).

S, J. (2023). *Scrn: Swin transformer for simultaneous denoising and interpolation of seismic data.* https://github.com/javashs/SCRN. (GitHub repository)

Saxena, D., & Cao, J. (2021). Generative adversarial networks (gans) challenges, solutions, and future directions. *ACM Computing Surveys (CSUR)*, *54*(3), 1–42.

Schlumberger. (2016, April). 3d interpolation using irregular_interp_rdn: Best practice for 3d time domain radon interpolation (tdri) [Computer software manual]. (Issue Date: 15-April-2016, Revision: 1, Content ID: 6828843)

Schonewille, M., Dishberger, D., & Kapadia, D. (2014). Comparison of 3d time-domain radon and matching-pursuit fourier interpolation. In *76th eage conference and exhibition 2014* (Vol. 2014, pp. 1–5).

Schonewille, M., Yan, Z., Bayly, M., & Bisley, R. (2013). Matching pursuit fourier interpolation using priors derived from a second data set. In *Seg technical program expanded abstracts 2013* (pp. 3651–3655). Society of Exploration Geophysicists.

Shannon, C. E. (1949). Communication in the presence of noise. *Proceedings of the IRE*, *37*(1), 10–21.

Shrestha, A., & Mahmood, A. (2019). Review of deep learning algorithms and architectures. *IEEE access*, *7*, 53040–53065.

Siahkoohi, A., Kumar, R., & Herrmann, F. (2018). Seismic data reconstruction with generative adversarial networks. In *80th eage conference and exhibition 2018* (Vol. 2018, pp. 1–5).

Soubaras, R. (1997). Spatial interpolation of aliased seismic data. In *67th annual international meeting, seg, expanded abstracts* (pp. 1167–1170).

Spitz, S. (1991). Seismic trace interpolation in the fx domain. *Geophysics*, *56*(6), 785–794.

Srinivas, A., Lin, T.-Y., Parmar, N., Shlens, J., Abbeel, P., & Vaswani, A. (2021). Bottleneck transformers for visual recognition. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 16519–16529).

Stolt, R. H. (2002). Seismic data mapping and reconstruction. *Geophysics*, *67*(3), 890–908.

Torbunov, D., Huang, Y., Yu, H., Huang, J., Yoo, S., Lin, M., … Ren, Y. (2023). Uvcgan: Unet vision transformer cycle-consistent gan for unpaired image-to-image translation. In *Proceedings of the ieee/cvf winter conference on applications of computer vision* (pp. 702–712).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wang, J., Ng, M., & Perz, M. (2010). Seismic data interpolation by greedy local radon transform. *Geophysics*, *75*(6), WB225–WB234.

Wang, Y., Grion, S., & Bale, R. (2009). What comes up must have gone down: The principle and application of up-down deconvolution for multiple attenuation of ocean bottom data. *CSEG Recorder*, *34*(10), 10–16.

Wapenaar, K., Van Der Neut, J., Ruigrok, E., Draganov, D., Hunziker, J., Slob, E., … Snieder, R. (2011). Seismic interferometry by crosscorrelation and by multidimensional deconvolution: A systematic comparison. *Geophysical Journal International*, *185*(3), 1335–1364.

Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, *5*(23), 495.

Xu, J., Sun, X., Zhang, Z., Zhao, G., & Lin, J. (2019). Understanding and improving layer normalization. *Advances in neural information processing systems*, *32*.

Xu, S., Zhang, Y., & Lambaré, G. (2010). Antileakage fourier transform for seismic data regularization in higher dimensions. *Geophysics*, *75*(6), WB113–WB120.

Xu, S., Zhang, Y., Pham, D., & Lambaré, G. (2005). Antileakage fourier transform for seismic data regularization. *Geophysics*, *70*(4), V87–V95.

Zou, F., Shen, L., Jie, Z., Zhang, W., & Liu, W. (2019). A sufficient condition for convergences of adam and rmsprop. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 11127–11135).

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
UNet256                                  [2, 1, 256, 256]          513
├─Sequential: 1-1                        [2, 64, 128, 128]         --
│    └─Conv2d: 2-1                       [2, 64, 128, 128]         1,024
│    └─ELU: 2-2                          [2, 64, 128, 128]         --
├─Sequential: 1-2                        [2, 128, 64, 64]          --
│    └─Conv2d: 2-3                       [2, 128, 64, 64]          131,072
│    └─BatchNorm2d: 2-4                  [2, 128, 64, 64]          256
│    └─ELU: 2-5                          [2, 128, 64, 64]          --
│    └─Dropout2d: 2-6                    [2, 128, 64, 64]          --
├─Sequential: 1-3                        [2, 256, 32, 32]          --
│    └─Conv2d: 2-7                       [2, 256, 32, 32]          524,288
│    └─BatchNorm2d: 2-8                  [2, 256, 32, 32]          512
│    └─ELU: 2-9                          [2, 256, 32, 32]          --
│    └─Dropout2d: 2-10                   [2, 256, 32, 32]          --
├─Sequential: 1-4                        [2, 512, 16, 16]          --
│    └─Conv2d: 2-11                      [2, 512, 16, 16]          2,097,152
│    └─BatchNorm2d: 2-12                 [2, 512, 16, 16]          1,024
│    └─ELU: 2-13                         [2, 512, 16, 16]          --
│    └─Dropout2d: 2-14                   [2, 512, 16, 16]          --
├─Sequential: 1-5                        [2, 512, 8, 8]            --
│    └─Conv2d: 2-15                      [2, 512, 8, 8]            4,194,304
│    └─BatchNorm2d: 2-16                 [2, 512, 8, 8]            1,024
│    └─ELU: 2-17                         [2, 512, 8, 8]            --
│    └─Dropout2d: 2-18                   [2, 512, 8, 8]            --
├─Sequential: 1-6                        [2, 512, 4, 4]            --
│    └─Conv2d: 2-19                      [2, 512, 4, 4]            4,194,304
│    └─BatchNorm2d: 2-20                 [2, 512, 4, 4]            1,024
│    └─ELU: 2-21                         [2, 512, 4, 4]            --
│    └─Dropout2d: 2-22                   [2, 512, 4, 4]            --
├─Sequential: 1-7                        [2, 512, 2, 2]            --
│    └─Conv2d: 2-23                      [2, 512, 2, 2]            4,194,304
│    └─BatchNorm2d: 2-24                 [2, 512, 2, 2]            1,024
│    └─ELU: 2-25                         [2, 512, 2, 2]            --
│    └─Dropout2d: 2-26                   [2, 512, 2, 2]            --
├─Sequential: 1-8                        [2, 512, 1, 1]            --
│    └─Conv2d: 2-27                      [2, 512, 1, 1]            4,194,304
│    └─ELU: 2-28                         [2, 512, 1, 1]            --
│    └─Dropout2d: 2-29                   [2, 512, 1, 1]            --
├─Sequential: 1-9                        [2, 512, 2, 2]            --
│    └─ConvTranspose2d: 2-30             [2, 512, 2, 2]
4,194,304
│    └─BatchNorm2d: 2-31                 [2, 512, 2, 2]            1,024
│    └─ELU: 2-32                         [2, 512, 2, 2]            --
│    └─Dropout2d: 2-33                   [2, 512, 2, 2]            --
├─Sequential: 1-10                       [2, 512, 4, 4]            --
│    └─ConvTranspose2d: 2-34             [2, 512, 4, 4]
8,388,608
│    └─BatchNorm2d: 2-35                 [2, 512, 4, 4]            1,024
│    └─ELU: 2-36                         [2, 512, 4, 4]            --
│    └─Dropout2d: 2-37                   [2, 512, 4, 4]            --
├─Sequential: 1-11                       [2, 512, 8, 8]            --
│    └─ConvTranspose2d: 2-38             [2, 512, 8, 8]
8,388,608
│    └─BatchNorm2d: 2-39                 [2, 512, 8, 8]            1,024
│    └─ELU: 2-40                         [2, 512, 8, 8]            --
│    └─Dropout2d: 2-41                   [2, 512, 8, 8]            --
├─Sequential: 1-12                       [2, 512, 16, 16]          --
│    └─ConvTranspose2d: 2-42             [2, 512, 16, 16]
8,388,608
│    └─BatchNorm2d: 2-43                 [2, 512, 16, 16]          1,024
│    └─ELU: 2-44                         [2, 512, 16, 16]          --
│    └─Dropout2d: 2-45                   [2, 512, 16, 16]          --
├─Sequential: 1-13                       [2, 256, 32, 32]          --
│    └─ConvTranspose2d: 2-46             [2, 256, 32, 32]
4,194,304
│    └─BatchNorm2d: 2-47                 [2, 256, 32, 32]          512
│    └─ELU: 2-48                         [2, 256, 32, 32]          --
│    └─Dropout2d: 2-49                   [2, 256, 32, 32]          --
├─Sequential: 1-14                       [2, 128, 64, 64]          --
│    └─ConvTranspose2d: 2-50             [2, 128, 64, 64]
1,048,576
│    └─BatchNorm2d: 2-51                 [2, 128, 64, 64]          256
│    └─ELU: 2-52                         [2, 128, 64, 64]          --
│    └─Dropout2d: 2-53                   [2, 128, 64, 64]          --
├─Sequential: 1-15                       [2, 64, 128, 128]         --
│    └─ConvTranspose2d: 2-54             [2, 64, 128, 128]
262,144
│    └─BatchNorm2d: 2-55                 [2, 64, 128, 128]         128
│    └─ELU: 2-56                         [2, 64, 128, 128]         --
│    └─Dropout2d: 2-57                   [2, 64, 128, 128]         --
├─Sequential: 1-16                       [2, 1, 256, 256]          --
│    └─ConvTranspose2d: 2-58             [2, 1, 256, 256]          2,048
│    └─Tanh: 2-59                        [2, 1, 256, 256]          --
==========================================================================================
Total params: 54,408,321
Trainable params: 54,408,321
Non-trainable params: 0
Total mult-adds (Units.GIGABYTES): 35.68
==========================================================================================
Input size (MB): 0.52
Forward/backward pass size (MB): 112.86
Params size (MB): 217.63
Estimated Total Size (MB): 331.02
==========================================================================================
```

Figure A.1: *Generator architecture and parameters for an input size of [2,1,256,256], where 2 is the batch size, 1 is the number of channel (greyscale), 256 is the image width and height.*

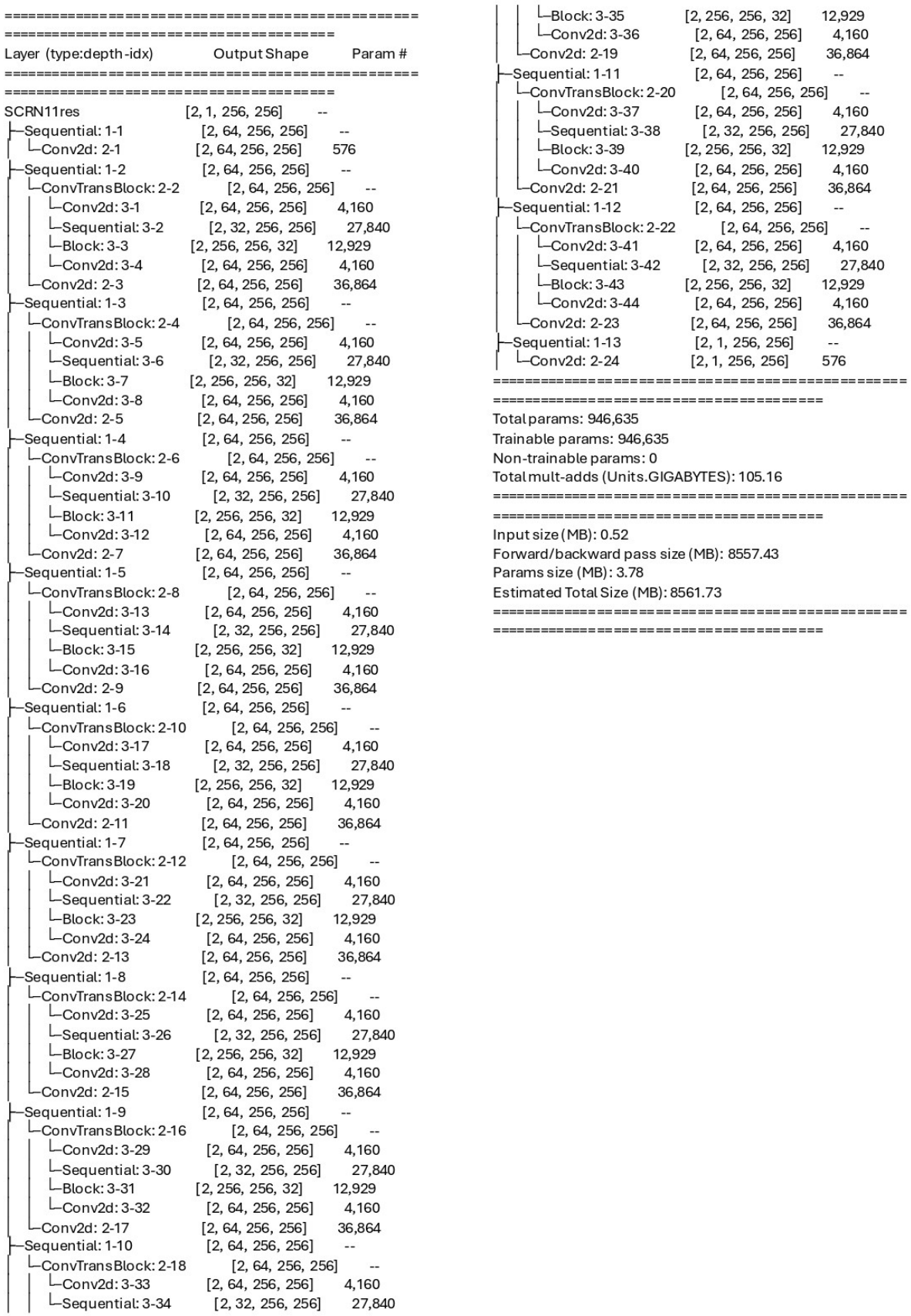


Figure A.2: *SCRN (11-FFBs) architecture and parameters for an input size of [2,1,256,256], where 2 is the batch size, 1 is the number of channel (greyscale), 256 is the image width and height.*

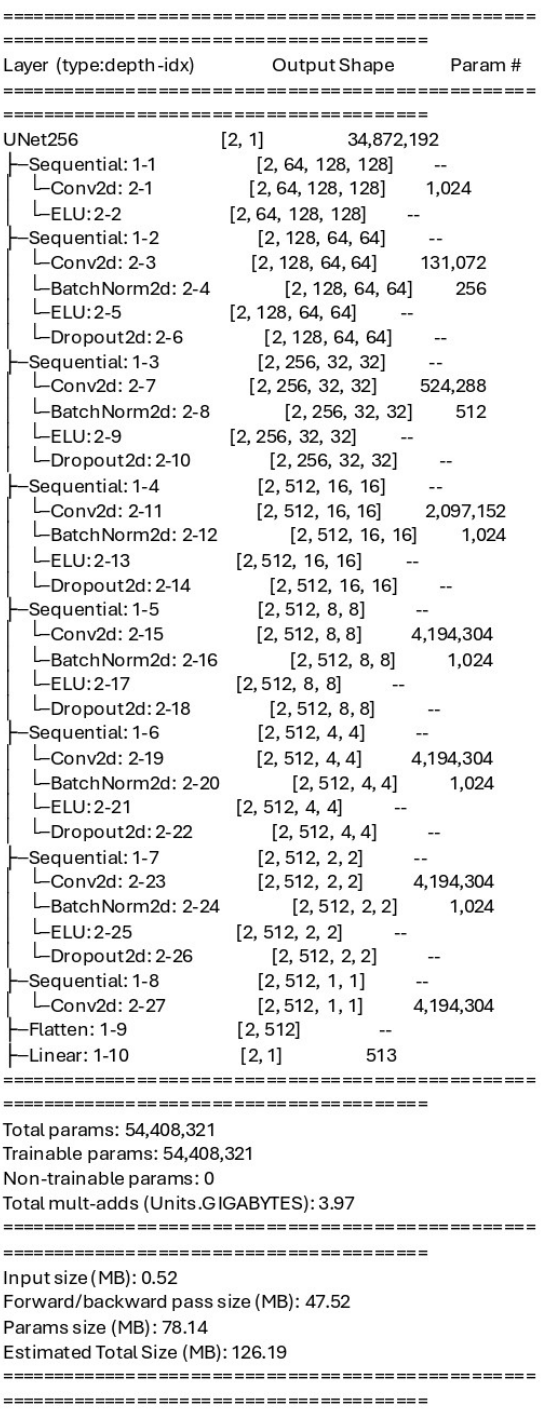


Figure A.3: *Discriminator architecture and parameters for an input size of [2,1,256,256], where 2 is the batch size, 1 is the number of channel (greyscale), 256 is the image width and height.*