
Chair of Applied Geophysics

Master's Thesis

# Machine Learning-based approach for the calculation of the most energetic traveltimes for Kirchhoff prestack depth migration

Guido Manzi

September 2024

MONTANUNIVERSITÄT LEOBEN
www.unileoben.ac.at

**AFFIDAVIT**

I declare on oath that I wrote this thesis independently, did not use any sources and aids other than those specified, have fully and truthfully reported the use of generative methods and models of artificial intelligence, and did not otherwise use any other unauthorized aids.

I declare that I have read, understood and complied with the "Good Scientific Practice" of the Montanuniversität Leoben.

Furthermore, I declare that the electronic and printed versions of the submitted thesis are identical in form and content.

Date 05.09.2024

Signature Author
Guido Manzi

# Acknowledgements

I write only a few words here to express the deep gratitude I hold for my supervisors Professor Tognarelli and Professor Aleardi and co-supervisors Dr. Caporal, Professor Bienati and Professor Bleibinhaus. I sincerely thank them for their guidance, insightful critiques and delightful discussions. But what I will remember the most, what will stay with me forever, is that in this short period, they taught me what it truly means to do science and the real work of a scientist.

A warm thank you goes to all my old and new friends I have met during this journey, who have made me experience extraordinary moments. I would also like to extend my thanks to the PhD students from the Faculty of Geophysics in Pisa and to the entire AESI group at the Eni headquarters in San Donato Milanese, who allowed me to experience the thrill of being part of a close-knit and cohesive team. A special thanks goes to my friend Diego Eustachio Farchione, who has patiently and consistently offered me advice and topics for discussion. Finally, my heartfelt thanks go to my dear family, who have always, unconditionally, supported me.

# Abstract

There is an extensive body of literature demonstrating that Prestack Kirchhoff Depth Migration (PSDM) using first-arrival traveltimes often produces suboptimal migrated images in regions with complex geology, which refers to areas with significant variations in the propagation velocity of seismic waves. To address this limitation, researchers since the 1990s have explored the use of the most energetic traveltimes for Kirchhoff migration instead of relying solely on the first arrivals. It has been shown that this approach yields more accurate and reliable subsalt images and offset image gathers. However, the primary drawback is the significant increase in computation time, as calculating the most energetic arrivals often requires solving the wave equation, which is more computationally intensive than the eikonal equation used for first arrivals. Focusing on the 2D acoustic case, this study aims to reduce the computational burden of generating max-energy arrival times by leveraging machine learning techniques. Specifically, U-Net-like architectures were employed in a supervised learning framework to predict the most energetic traveltimes, using velocity models and first-arrival traveltimes as inputs. The choice of U-Net is motivated by its robustness and versatility across various applications. Additionally, diffusion models were applied to the U-Net outputs to further enhance the quality of the migrated images. To validate the proposed approach, the Marmousi velocity model was migrated using the predicted traveltimes. The resulting migrated images were compared against those obtained using traditional Kirchhoff migration with first-arrival and most energetic arrival traveltimes. This thesis demonstrates that the U-Net-based approach substantially improves computational efficiency, reducing processing time by approximately two orders of magnitude. However, in particularly complex geological scenarios, the resolution of the U-Net outputs is sometimes lower than desired. Applying diffusion models improved image quality, but at the cost of increased computational time.

# Contents

# Chapter 1

# Introduction

Most Kirchhoff migration algorithms in the industry rely on first-arrival traveltimes as an approximation of the full Green's function (Nichols, 1996) [1]. However, since these traveltimes are based on the high-frequency approximation and the first arrivals often carry little energy in complex environments, the resulting images in such areas tend to be of poor quality. As early as the late 20th century, studies began to highlight the advantages of using the most energetic traveltimes in migration algorithms. These traveltimes provide the best single-event approximation of the full Green's function (Nichols, 1996) [1]. In Nichols, 1996 [1], a method was developed to calculate the most energetic traveltimes in the seismic frequency band, not by using the high-frequency approximation but by solving the Helmholtz equation for a few frequencies and then, with a parametric fit to the wavefield, estimating traveltime, amplitude and phase. Despite these advancements, first-arrival traveltimes are still preferred due to their lower computational cost compared to the most energetic arrivals, which require solving the wave equation rather than the simpler eikonal equation or using ray tracing.

In recent years, several papers have explored the potential of using Kirchhoff migrations with the most energetic traveltimes, calculated by solving the wave equation, for quality control in Full Waveform Inversion (FWI) (Pu et al., 2021) [2], (Wang et al., 2023) [3], (Jin et al., 2020) [4], (Jin et al., 2023) [5]. This method, known as Wave Equation Kirchhoff (WEK), produces more reliable offset gathers and image stacks than first-arrival Kirchhoff migrations, without needing to smooth the high-resolution velocity, which required significant computational effort to be estimated, as they were produced by high-frequency FWI. Moreover, WEK has an advantage over Reverse Time Migration (RTM) because it requires significantly less computational resources to generate stacks and gathers.

This thesis introduces a new method for calculating the most energetic traveltimes using a supervised machine learning approach. It aims to overcome the main drawback of Kirchhoff migration with the most energetic traveltimes: the higher computational cost. A Convolutional Neural Network (CNN), specifically based on a U-Net architecture, was chosen as the foundation for this work.

The thesis is organized as follows:

Chapter 2: Offers an overview of the 2D acoustic wave equation, introducing the Finite Difference (FD) method. The application of the FD method to solve the wave equation is discussed, with emphasis on critical factors like stability, dispersion, and boundary conditions. The chapter wraps up with a brief introduction to the Python library Devito, used for implementing the FD method efficiently.

Chapter 3: Opens with a theoretical exploration of Kirchhoff migration, outlining its core concepts and principles. The focus then shifts to the specific Kirchhoff migration approach developed in this thesis, which relies on directly solving the wave equation instead of using high frequency approximations based on the eikonal equation. Implementation details of the migration algorithm are also provided, offering insights into its design and functionality.

Chapter 4: Delivers a description of the neural network architectures used in this thesis. It includes an in-depth exploration of U-Net, along with its variants, Attention U-Net and ResAUnet. The chapter also delves into Neural Style Transfer (NST), covering both its theoretical and mathematical aspects, and introduces the ensembling technique. Finally, Diffusion Models are examined, focusing on their theoretical and mathematical foundations.

Chapter 5: Summarizes the entire workflow, outlining the logical sequence of steps that led to the objectives of this thesis. Key experiments and processes are briefly mentioned, setting the stage for a more detailed discussion in the following chapter.

Chapter 6: Stresses the importance of a high-quality training dataset for the success of machine learning projects. It details the process of selecting and preparing datasets, using five different velocity models that represent various geological scenario. The well-known Marmousi synthetic velocity model is chosen for the test dataset, specifically designed for evaluating imaging algorithms.

Chapter 7: Examines the practical applications of the methods discussed in previous chapters. It showcases the progression from the creation of the initial dataset to the generation of migrated images and their subsequent comparison. Throughout, both final and intermediate results are presented, with an in-depth analysis of the various parameters fine-tuned during the process.

Chapter 8: Concludes with a discussion and analysis of the results, evaluating the performance of the proposed method and considering its implications for future research and applications.

# Chapter 2

# Acoustic Forward Operator

## 2.1 Acoustic Wave Equation

The Acoustic Wave Equation is a second order linear partial differencial equation that describes the propagation of acoustic waves (Pierce 1991) [6]. The solution $u(t, \mathbf{x})$ is a time-dependent pressure field, with $\mathbf{x} \in \Omega$ and $t > 0$. Where $\Omega$, in this work two-dimensional, depicts the collection of points within the simulated environment. For each point $x$ in the environment and for each $t$, the function $u$ describes the sound pressure. The Acoustic Wave equation relates the second time derivative of pressure to its spatial Laplacian, along with a source term $f$.

$$\frac{\partial^2 u}{\partial t^2} = c^2 \Delta u + f \tag{2.1}$$

where considering two dimensions

$$\Delta = \nabla \cdot \nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial z^2}. \tag{2.2}$$

The 2D Acoustic wave equation:

$$\frac{\partial^2 u(x,z,t)}{\partial t^2} = c(x,z)^2 \left( \frac{\partial^2 u(x,z,t)}{\partial x^2} + \frac{\partial^2 u(x,z,t)}{\partial z^2} \right) + f(x,z,t) \tag{2.3}$$

describes the propagation of sound waves in a medium. In the context of this thesis, the medium is represented by a velocity model. Here, $u$ represents the pressure field, $c$ is the speed of sound, and $f(x, z, t)$ is a source term that typically represents an initial perturbation. The equation is a partial differential equation (PDE) because it involves derivatives of $u$ with respect to more than one variable. To solve PDEs, boundary conditions and initial conditions must be specified.
Boundary conditions (BCs) define how the boundaries of the environment reflect sound waves. These can be:

- Dirichlet Boundary Conditions: Specify the values of the solution on the boundaries.
- Neumann Boundary Conditions: Specify the values of the normal derivatives of the solution on the boundaries.
- Absorbing Boundaries: Designed to absorb incoming waves to simulate open boundaries.

Initial conditions specify the initial pressure distribution $u(0, \mathbf{x})$ and the initial velocity distribution $u_t(0, \mathbf{x})$ at $t = 0$.

## 2.2 Finite Difference Method

Analytical solutions to the wave equation are only feasible for very simple velocity models, such as constant velocity, linear gradients, homogeneous layered media with parallel interfaces and so on. For more complex velocity models, numerical methods are necessary.

Numerical Analysis is a branch of mathematics focused on approximating solutions to continuous problems using discrete methods. It involves discretizing the solution domain in both time and space, and solving the resulting discrete equations.

To approximate derivatives of a continuous function $f(x)$ defined on the interval $[0, 1]$, discretization is used. This involves dividing this interval into $N$ equally spaced points $x_i$ where $i = 1, \ldots, N$, with a distance between consecutive points denoted by $h$.

The Taylor series expansion provides a useful tool for deriving these approximations. Expanding the function $f(x)$ around a point $x_0$ is obtained:

$$f(x) = f(x_0) + (x - x_0)f'(x_0) + \frac{(x - x_0)^2}{2!}f''(x_0) + \frac{(x - x_0)^3}{3!}f'''(x_0) + \cdots \tag{2.4}$$

By truncating this series and rearranging terms, different finite difference formulas can be derived to approximate the derivative of $f(x)$ at specific points using values of the function at neighboring points.

The forward difference method approximates the derivative at $x_0$ by using the value of the function at $x_0$ and its next point $x_0 + h$. This gives:

$$\frac{\partial f}{\partial x}(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h} + O(h), \tag{2.5}$$

This method is a first-order approximation, meaning the error decreases linearly with $h$

Similarly, the backward difference method uses the value of the function at $x_0$ and its previous point $x_0 - h$, giving the approximation:

$$\frac{\partial f}{\partial x}(x_0) \approx \frac{f(x_0) - f(x_0 - h)}{h} + O(h), \tag{2.6}$$

Like the forward difference, this is also a first-order approximation.

A more accurate approach is the central difference method, which uses the values of the function at both $x_0 + h$ and $x_0 - h$ to approximate the derivative. This gives:

$$\frac{\partial f}{\partial x}(x_0) \approx \frac{f(x_0 + h) - f(x_0 - h)}{2h} + O(h^2). \tag{2.7}$$

The central difference is a second-order approximation, meaning its error decreases proportionally to $h^2$, making it more accurate than both the forward and backward methods for small $h$.

Figure 2.1 compares the first-order and second-order difference approximations of the function $\sin(1/x)$ over the interval $[-0.2, -0.1]$. The figure illustrates that higher-order approximations, the central difference scheme (2.7), provide better accuracy by more closely aligning with the true derivative.

The second-order approximation for the second derivative at $x_0$ is:

$$\frac{\partial^2 f}{\partial x^2}(x_0) \approx \frac{f(x_0 - h) - 2f(x_0) + f(x_0 + h)}{h^2} + O(h^2), \tag{2.8}$$

where $h$ is the step size. This formula provides an error proportional to $h^2$.

For an even more accurate approximation, the fourth-order formula is:

$$\frac{\partial^2 f}{\partial x^2}(x_0) \approx \frac{-f(x_0 + 2h) + 16f(x_0 + h) - 30f(x_0) + 16f(x_0 - h) - f(x_0 - 2h)}{12h^2} + O(h^4). \tag{2.9}$$

The fourth-order approximation improves accuracy with an error proportional to $h^4$, making it more precise than the second-order method for small $h$.

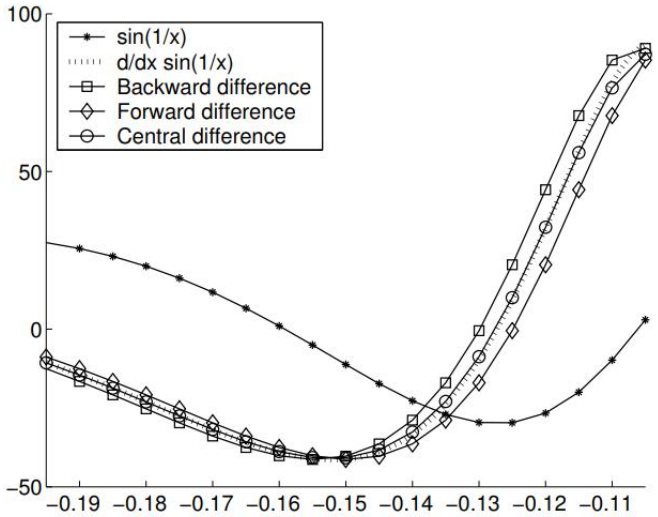


Figure 2.1: Comparison of the first-order and second-order difference approximations of the function $\sin(1/x)$ on the interval $[-0.2, -0.1]$. Image taken from (Lehtinen 2003) [7].

## 2.3 Finite Difference Solution applied on Acoustic Wave Equation

To solve the 2D acoustic equation, it is first necessary to discretize the dimensions of the field, from continuous (2.10) to discrete (2.11) , (2.12) , (2.13).

$$\frac{\partial^2 u(x,z,t)}{\partial t^2} = c(x,z)^2 \left( \frac{\partial^2 u(x,z,t)}{\partial x^2} + \frac{\partial^2 u(x,z,t)}{\partial z^2} \right) + f(x,z,t) \tag{2.10}$$

The functions then become:

$$u(x,z,t) \rightarrow u^n_{j,k} = u(j\,dx, k\,dz, n\,dt) \tag{2.11}$$

$$c(x,z) \rightarrow c_{j,k} = c(j\,dx, k\,dz) \tag{2.12}$$

$$f(x,z,t) \rightarrow f^n_{j,k} = f(j\,dx, k\,dz, n\,dt) \tag{2.13}$$

Here, $u(x,z,t)$ represents the acoustic pressure field as a function of spatial coordinates $x$ and $z$, and time $t$, while $j$, $k$, and $n$ are integers representing the indices of the spatial and temporal discretization. When discretized, it becomes $u^n_{j,k}$, which denotes the pressure at the discrete spatial grid points $j\,dx$ and $k\,dz$, and at the discrete time step $n\,dt$.
The function $c(x,z)$ represents the speed of sound at any point in the continuous domain. Its discretized form, $c_{j,k}$, represents the speed of sound at the grid point $(j\,dx, k\,dz)$.
The source term $f(x,z,t)$ represents the source term in the acoustic field. Its discretized version, $f^n_{j,k}$, denotes the source term at the spatial grid points $(j\,dx, k\,dz)$ and the time step $n\,dt$.

By substituting the Finite Difference approximation for the second time and space derivatives:

$$\frac{\partial^2 u(x,z,t)}{\partial t^2} \approx \frac{u^{n+1}_{j,k} - 2u^n_{j,k} + u^{n-1}_{j,k}}{dt^2} \tag{2.14}$$

$$\frac{\partial^2 u(x,z,t)}{\partial x^2} \approx \frac{u^n_{j+1,k} - 2u^n_{j,k} + u^n_{j-1,k}}{dx^2} \tag{2.15}$$

$$\frac{\partial^2 u(x,z,t)}{\partial z^2} \approx \frac{u^n_{j,k+1} - 2u^n_{j,k} + u^n_{j,k-1}}{dz^2} \tag{2.16}$$

Substituting (2.14) , (2.15) , (2.16) into the 2D discrete Acoustic Wave Equation results in (2.17):

$$\frac{u^{n+1}_{j,k} - 2u^n_{j,k} + u^{n-1}_{j,k}}{(dt)^2} = c^2_{j,k}\left(\frac{u^n_{j+1,k} - 2u^n_{j,k} + u^n_{j-1,k}}{(dx)^2} + \frac{u^n_{j,k+1} - 2u^n_{j,k} + u^n_{j,k-1}}{(dz)^2}\right) + f^n_{j,k} \tag{2.17}$$

Assuming that in the grid $dx = dz$ and solving for $u^{n+1}_{j,k}$ yields:

$$u^{n+1}_{j,k} = 2u^n_{j,k} - u^{n-1}_{j,k} + (dt)^2 f^n_{j,k} + \frac{(dt)^2}{(dx)^2}\left[c^2_{j,k}\left(u^n_{j+1,k} - 2u^n_{j,k} + u^n_{j-1,k} + u^n_{j,k+1} - 2u^n_{j,k} + u^n_{j,k-1}\right)\right] \tag{2.18}$$

Equation (2.18) provides a method to compute the future value of the acoustic field $u$ at a given grid point $(j,k)$ for the next time step $n+1$, based on its current value $u^n_{j,k}$, the previous value $u^{n-1}_{j,k}$, the source term $f^n_{j,k}$, and the values of $u$ at neighboring grid points.

## 2.4 Dispersion and Stability Conditions

Finite Difference computation of the Acoustic Wave Equation necessitates the establishment of criteria for spatial and temporal discretization. A poor choice of temporal and spatial interval can lead to dispersion and instability problems.

Dispersion error refers to the phenomenon where waves propagate at incorrect speeds due to inadequate discretization. This effect results in the distortion of waveforms and can severely impact the accuracy of simulations. Higher-order discretizations help mitigate this issue by providing a more accurate representation of the wave equation, reducing the dispersion error. Furthermore, selecting appropriate grid spacing and analyzing the numerical method's dispersion characteristics can enhance accuracy, ensuring that wave propagation is modeled effectively (Kieri, 2016) [8].

$$\Delta x < \frac{V_{\text{min}}}{n \cdot f_{\text{max}}} \tag{2.19}$$

(2.19) specifies that the spatial grid interval $\Delta x$ should be smaller than $\frac{V_{\text{min}}}{n \cdot f_{\text{max}}}$ to minimize numerical dispersion in two-dimensional wave propagation. In this context, $n$ represents the number of grid points per wavelength. For a good approximation in the case of a second order approximation of the spatial derivatives, values of n equal to 10 are considered accurate (Alford et al., 1994) [9]. The parameter $V_{\text{min}}$ is the minimum velocity, and $f_{\text{max}}$ is the maximum frequency of the wave.

Instability arises when the energy of the discretized solution grows over time, rather than remaining constant. The time spacing to keep the solution stable must satisfy the Courant-Friedrichs-Levy (CFL) stability criterion, which in the 2D case with a second order approximation is equivalent to (2.20) (Lines et al., 1999) [10]:

$$\frac{V_{\text{max}} \cdot dt}{dx} < \frac{1}{\sqrt{2}} \tag{2.20}$$

Therefore, appropriate choices for both spatial and temporal discretization are crucial. Initially, spatial sampling is selected to minimize grid dispersion, ensuring an accurate representation of the wave field. Subsequently, temporal sampling is chosen to avoid instability, adhering to the CFL criterion to ensure that the numerical solution remains stable and reliable.

## 2.5 Boundary Conditions in practice

A significant challenge in computational simulations of acoustic waves is addressing the infinite propagation of waves in all directions, which is not feasible to model directly. To approximate this infinite domain, techniques like Absorbing Boundary Conditions (ABC) (Clayton and Engquist, 1977) [11] or Perfectly Matched Layers (PML) (Bèrenger, 1994) [12] are employed. These techniques help simulate an infinite medium by dampening and absorbing waves at the domain boundaries, thereby preventing reflections.

The method applied in this work is an absorbing damping mask, since it was the least computationally expensive. This method involves extending the computational domain and introducing a Sponge layer at the boundary. The Sponge layer acts to absorb incident waves. The Acoustic Wave Equation incorporating this damping mask is represented as:

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} - c^2 \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial z^2} \right) + \eta \frac{\partial u}{\partial t} = f \\ u(x, z, 0) = 0 \\ \frac{\partial u}{\partial t}(x, z, 0) = 0 \end{cases} \tag{2.21}$$

In this context (2.21) , $\eta$ represents the damping mask. Within the physical domain, $\eta$ is set to 0, and it increases progressively within the sponge layer in order to attenuate the signal and minimize reflections at the boundary, effectively simulating an open or infinite domain.

## 2.6 Devito

Devito is a domain-specific language (DSL) created to develop high-performance solvers for partial differential equations (PDEs) using finite difference methods (Louboutin et al., 2018) [13]. It was specifically designed to address the challenges of exploration seismology, where techniques like Full-Waveform Inversion (FWI) and Reverse-Time Migration (RTM) are used to process seismic data. These techniques are extremely computationally demanding, often taking weeks to complete even on modern supercomputers, largely due to the intensive calculations required to solve wave equations and their adjoints. Devito was developed with a core focus on optimizing stencil computations, a series of calculations performed on a grid of values, which are essential for solving differential equations and other spatially dependent computations. This core capability allows Devito to significantly improve performance across different computer architectures, reducing computation time.

In this Thesis work, Devito was chosen for solving the isotropic acoustic wave equation due to its ability to express the solution in just a few lines of code, thanks to its high-level abstraction using SymPy for symbolic computation. Additionally, Devito is a Python package that is easily downloadable and straightforward to integrate into existing workflows. Once the solution was obtained in the form of temporal snapshots of wave propagation, it became possible to create traveltime maps of the most energetic arrivals for each velocity model. Moreover, Devito generated also the synthetic seismograms used in the migration part.

The parameters used for simulations with Devito included:

- $V_p$ (P-wave velocity)
- Source wavelet
- Boundary thickness
- Source and receiver locations
- Sampling frequency
- Number of receivers
- Grid spacing
- Total time of simulation

Below is an example of Python code utilizing the Devito library, where the essential parameters mentioned above are defined to perform the solution of the wave equation.

```
# velocity model creation
gridspacing = 10
damspace = 300
velocity = np.ones((400, 400)) * 1.5
shape = (velocity.shape[1], velocity.shape[0])
spacing = (gridspacing, gridspacing)
origin = (0., 0.)
Tampmax = np.zeros((np.shape(velocity)[0], np.shape(velocity)[1]),
                   dtype="float32")
v = np.transpose(velocity[:, :])
model = Model(vp=v, origin=origin, shape=shape, spacing=spacing,
              spaceorder=2, nbl=damspace, bcs="damp")

# total time and dt
tn = 4000
t0 = 0.
dt = model.criticaldt
timerange = TimeAxis(start=t0, stop=tn, step=dt)

# source characteristics
f0 = 0.010
src = RickerSource(name='src', grid=model.grid, f0=f0, npoint=1,
```

```
                    timerange=timerange)
src.coordinates.data[0, :] = np.array(model.domainsize) * .5
src.coordinates.data[0, -1] = 1.

# receivers characteristics
xextent,  = model.domainsize
xlocs = np.linspace(0, xextent, shape[0])
reccoords = [(x, 1) for x in xlocs]
rec = Receiver(name='rec', npoint=shape[0], grid=model.grid,
               coordinates=reccoords, timerange=timerange)
rec.coordinates.data[:, 1] = 1.
```

# Chapter 3

# Kirchhoff migration

## 3.1 Kirchhoff migration in general

Kirchhoff migration is a well known technique in seismic imaging, it is used to reconstruct subsurface geological structures from recorded seismic data. It operates by summing seismic data along predicted diffraction paths to produce an accurate subsurface image. This method was introduced by (Schneider, 1971) [14] and has been widely adopted due to its efficiency and precision in seismic processing.

Kirchhoff migration can be performed in several domains, including space-time, space-frequency, wavenumber-time, and wavenumber-frequency (Zhu and Lines, 1998) [15]. Prestack Kirchhoff migration relies on the solution of the acoustic wave equation. A common approach to this solution involves the use of the WKBJ approximation, which provides a high-frequency asymptotic form of the Green's function (Aki and Richards, 1980) [16]. To calculate the traveltimes used in the migration process, the eikonal equation is often employed (Cerveny, 2001) [17]. The migration integral of a single shot in space-time domain can be expressed by the following equation (Zhu and Lines, 1998) [15]:

$$R(x;x_s) = \int_{\sum} n \cdot \nabla\tau_r(x_r;x) A(x_r;x;x_s) u^m(x_r, \tau_s(x;x_s) + \tau_r(x_r;x);x_s)\, dx_r \tag{3.1}$$

where $\sum$ represents the recording surface; $\tau_s$ and $\tau_r$ are the traveltimes from the source point $x_s$ to the subsurface position $x$, and from $x$ to the receiver at $x_r$, respectively; and $n$ is the outward normal of the surface $\sum$. The term $u^m$ denotes the time derivative of the recorded traces, with $m = \frac{1}{2}$ for the 2-D case. The term $A(x_r; x; x_s)$ represents the geometrical spreading, which functions as an amplitude modulator for the recorded traces.

Migration using equation (3.1) is basically a weighted summation of the derivative traces along the presumed diffraction trajectory t = $\tau_s$+$\tau_r$ . The approximation of the weights is done based on a constant velocity model. There, the weights, are expressed as a function of velocity, traveled distance, and the obliquity of the emergence ray at the recording surface (Bleistein et al., 1987) [18].

To summarize, the steps involved in performing Kirchhoff migration for each input trace are:

1. Compute the source and receiver traveltimes using a velocity-depth model.
2. Calculate the two-way traveltime by summing the traveltime from the source to the subsurface point and the traveltime from the subsurface point to the receiver.
3. For each point, compute the amplitude factor .
4. Position each amplitude value of the seismic trace at the corresponding point with the same traveltime, multiplying it by its amplitude factor.

Then Impulse Response of all traces (Fig.3.1) are stacked, creating seismic reflectors where energy accumulates (Fresnel zone) and cancel each other out through stacking outside.

One significant drawback of Kirchhoff migration when the computed traveltimes are first arrivals traveltimes is its accuracy in complex geological structures. The method may fail to capture the full energy of the seismic wavefield recorded in the data, leading to poor imaging results, especially if the spatial variation of propagation velocity in the subsurface is very complex (i.e., sharp velocity contrasts along coplex interface like, for instance, those associated to the presence of salt diapirs). In fact first arrival traveltimes in complex geological areas may contain little energy and Imaging using these traveltimes does not coherently stack the most important parts of the wavefield (Fig.3.2).

Another issue arises from the high frequency approximations used in calculating the Green's function traveltimes. These traveltimes are not a good approximation to the traveltimes of the seismic wavefield if the medium is dispersive (Kieri, 2016) [8].

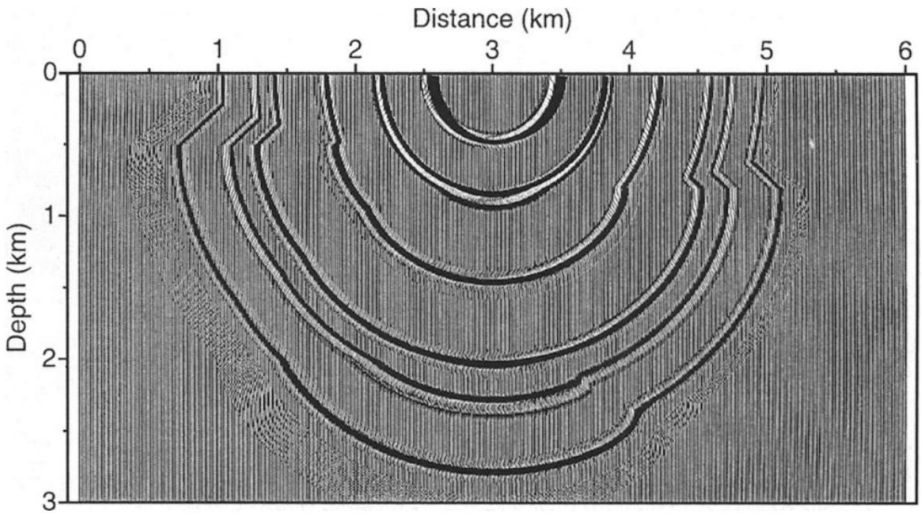


Figure 3.1: Kirchhoff migration impulse, image taken from (Zhu, 1998) [15]

In an effort to overcome the limitations of traditional Kirchhoff migration, several authors proposed replacing ray-based traveltime calculations with wave-equation-based approaches. Nichols (1996) suggested estimating the wavefield by solving the Helmholtz equation at a few selected frequencies within the seismic bandwidth, rather than solving the eikonal equation for traveltime. He then applied a parametric fit to the computed wavefield to estimate traveltime, amplitude, and phase. It is noted that, according to (Nichols, 1996)) [1] and (Nguyen and McMechan, 2013) [20], the maximum energy traveltime is the best single-event approximation to the full Green's function Fig(3.2). (Ehinger et al. 1996) [21] and (Etgen, 2012) [22] proposed Kirchhoff migration using Green's function computed with wavefield extrapolation techniques based on a finite difference implementation of the wave equation. (Andrade et al. 2015) [23] showed a method to calculate maximum amplitude traveltimes with the Chebyshev polynomial recursion. (Jin and Etgen, 2020) [4] directly generated maximum-amplitude traveltimes using finite-difference solutions to the full wave equation. (Pu et al., 2021) [2] presented a Kirchhoff migration using both maximum energetic traveltimes and amplitudes derived from the wavefield computed by full wavefield propagation, calling this scheme Wave Equation Kirchhoff (WEK) (Fig.3.3).

On the migration side of this Thesis work, an attempt was made to implement a Kirchhoff migration based on the solution of Wave Equation, using as traveltimes the most energetic ones. The Wave Equation has been solved via a Finite Difference method.

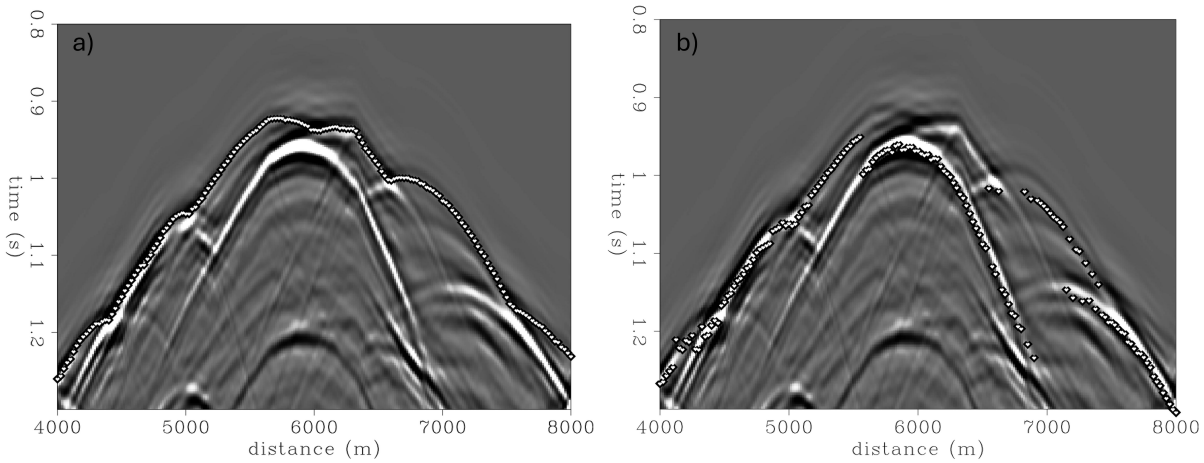


Figure 3.2: a) First arrival traveltimes in dotted data superimposed upon the full diffraction wavefield, b) Most energetic arrival traveltimes in dotted data superimposed upon the full diffraction. Image taken from (Audebert et al., 1997) [19]

## 3.2 Wave equation Kirchhoff

In this Thesis, a Wave Equation Kirchhoff prestack depth migration was implemented. The use of the Devito software not only allowed the resolution of the Acoustic Wave Equation, but also the calculation of the most energetic traveltimes and the obtaining of shot gathers. The migration was performed on zero offset data.
The formulation used for performing the migration is the following:

$$\text{migrated\_area}(x,y) = \sum_{s=0}^{N_{\text{traces}}-1} \sum_{t=0}^{T_{\text{recorded}}-1} \sum_{\tau(x',y',s)=t\times\Delta t\}} (\text{trace\_sample}(t,s) \times W(x',y',t,s)) \tag{3.2}$$

Where in (3.2) :

- $N_{\text{traces}}$: Total number of seismic traces.
- $T_{\text{recorded}}$: Total number of recorded time samples.
- $\tau(x', y', s)$: Travel time from the source to the subsurface point $(x', y')$ for trace $s$.

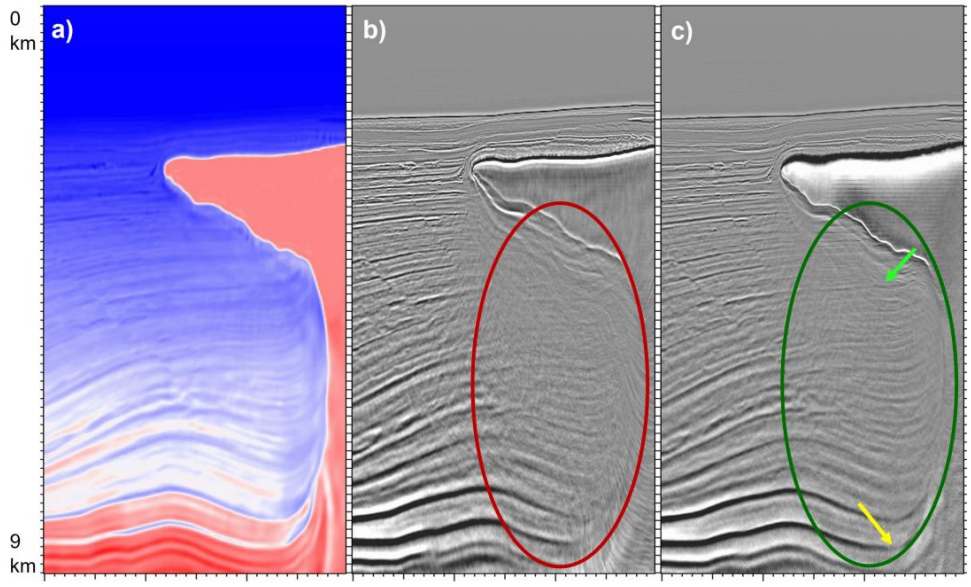


Figure 3.3: a) Velocity model, b) Conventional Kirchhoff stack image, c) WEK stack image. Image taken from (Jin, 2023) [5]

- $t \times \Delta t$: Corresponding time sample $t$ in the recorded seismic data.
- trace_sample$(t, s)$: Amplitude of the seismic trace $s$ at time $t$.
- $W(x', y', t, s)$: Amplitude correction weight.

As a weight, it was decided to implement an approximation of Bleistein'weight, (Bleistein et al., 1998) [24], taken for the common 2D offset case by (Zhang et al., 2000) [25], equals to:

$$\frac{4z}{v^2 t} \tag{3.3}$$

In (3.3) $z$ stands for depth, $v$ for velocity and $t$ for two-way travel time. This weight, which is valid in constant velocity cases, has the advantage that it can be directly calculated during the data spreading on the isochrone, since $z$ and the two-way travel time $t$ can be easily computed. This allows for a significant reduction in the cost of Kirchhoff migration. In (Fig.3.4) a schematic summary of the Kirchhoff algorithm implemented.

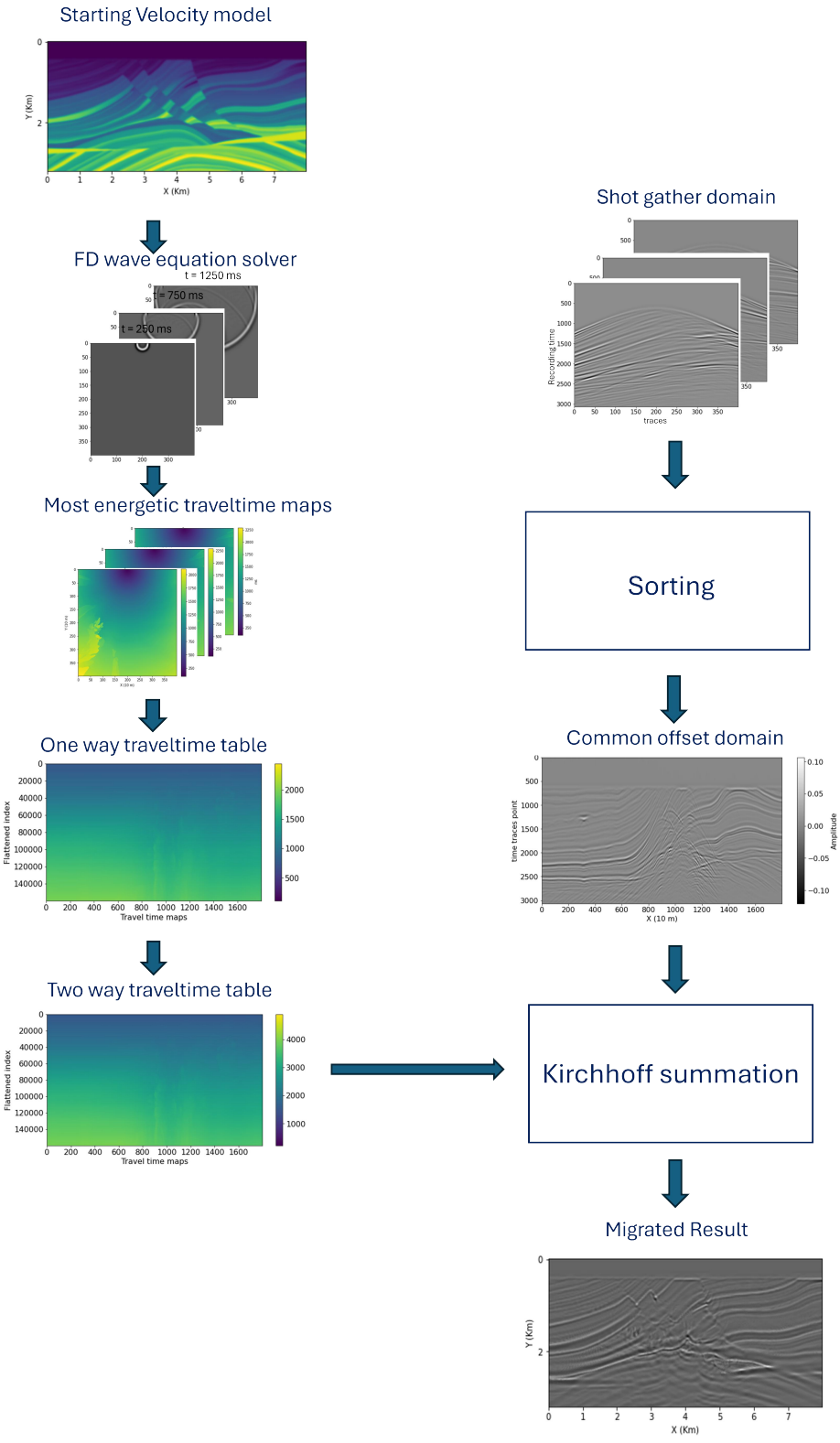


Figure 3.4: Workflow of Wave Equation Kirchhoff Migration implemented in this thesis work

## 3.3 Implementation

This section outlines the implementation of the pre-stack migration process used in this study, based on a ”spraying” approach.

The velocity model, used for migration, was divided into several smaller sub-models. Within each sub-model, the wave equation was solved via Devito using the finite difference method, assuming the source was located at the central, near-surface position of the sub-model. From the wave equation solution, synthetic data were generated in the form of shot gathers, along with the most energetic arrival traveltime map. This traveltime map shows the time it takes for the most energetic arrival to reach each point within the sub-model, starting from the source location.

Before starting the migration, the shot gather data were filtered using a $\sqrt{i2\pi f}$ filter. This filter was necessary due to the mathematical formulation of the algorithm, which is derived from the wave equation. After filtering, the data were sorted to transition from the shot gather domain to the common offset domain. In this thesis, however, only zero-offset data (0 m offset) were used.

Before migration, for each trace in the common offset domain, it was necessary to calculate the corresponding most energetic two-way traveltime map. This map represents the time taken for the most energetic arrival to travel from the source to any subsurface point and then back to the receiver. In the case of this thesis, where zero-offset migration (0 m offset) is employed, the calculation of the most energetic two-way traveltime map is greatly simplified, as the source and receiver are located at the same position. This traveltime map is used to ”spray” the data along corresponding trace across surfaces known as isochrones. By repeating this operation for all traces and summing the results, the final migrated section is obtained.

The algorithm employed for migrating these traces is described as follows:

```
1  for s = 0 to number_of_traces do
2      migrated_section = np.zeros((rows, columns)) ;
3      for t = 0 to data_points_recorded - 1 do
4          position = np.where(two_way_travel_time[:, s] == t · dt)[0] ;
5          if position.size >0 then
6              for element in position do
7                  row = element // len(migrated_section[0, :]) ;
8                  column = element % len(migrated_section[0, :]) ;
9                  result = function(column) ;
10                 migrated_section[row, column] +=
11                 current_offset[t, s] ·Weight_function[t, z]
12     migrated_section = migrated_section · migration_aperture ;
13     migrated_area += migrated_section ;
```

The Kirchhoff migration process begins by iterating through each trace that composes the common offset data, starting from the leftmost trace to the rightmost one. For each cycle, an initially empty matrix `migrated_section` is created to store the migrated amplitudes for that trace.

Then the algorithm, for every trace, iterates through all recorded time points $t$ and calculates the corresponding positions where the two-way travel time equals $t \times dt$. These positions represent the isochrones—curves or surfaces in the subsurface where energy arrives simultaneously.

For each computed isochrone, the seismic amplitudes corresponding to the current time $t$ are distributed across these positions in the `migrated_section` matrix. This is done by adding the weighted amplitude values from the seismic traces, `current_offset`$[t, s] \cdot$ `Weight_function`$(t, z)$, to the `migrated_section`. The weights are determined by a `Weight_function`(t,z), which adjusts the contribution of each trace to account for factors such as geometric spreading, the velocity model, and other migration parameters (see equation 3.3).

After summing the amplitudes across all isochrones for a given shot, a depth-variant migration

aperture mask is applied to the `migrated_section`. This mask limits the migrated area to a specific region of interest, effectively controlling the spatial extent of the migration process. Finally, the migrated section of the current shot is added to the cumulative `migrated_area` matrix. This matrix represents the entire migrated image, and by summing over successive shots, the algorithm progressively builds the final migrated result.

# Chapter 4

# Neural Networks

## 4.1 U-Net-like Architectures

### 4.1.1 Introduction to U-Net and Its Variants

**U-Net: Background and Significance**

Introduced by Ronneberger et al. 2015 [26], U-Net is a powerful and widely-used neural network architecture, originally developed for biomedical image segmentation. Its unique architecture allows it to effectively segment images even with limited training data, which is particularly beneficial in fields such as medical imaging.

The U-Net architecture is renowned for its distinctive encoder-decoder structure, which is crucial for its ability to capture both high-level semantic information and local spatial features. This characteristic makes U-Net not only important for segmentation but also versatile for other tasks like image regression, where precise spatial information is essential.

The U-Net model is composed of three main components:

- Encoder: The encoder path captures the context of the image through successive convolutional layers and down-sampling operations. It consists of a series of convolutional blocks followed by max-pooling layers. Each block typically includes a convolutional layer, batch normalization, and ReLU activation function (for more information, see the Glossary of Terms later in this chapter). The convolutional layers generate feature maps at each level, which are intermediate representations of the image that capture essential attributes such as edges, textures, and patterns. These feature maps are progressively refined through each convolutional block and are crucial for the encoder to effectively

learn and represent the input image.

- Decoder: The decoder path reconstructs the image from the feature maps produced by the encoder. It uses up-sampling operations to reconstruct the spatial dimensions of the feature maps. Each up-sampling step is followed by a concatenation with the corresponding feature map from the encoder path (skip connections) and additional convolutional layers.

- Skip Connections: These connections between the encoder and decoder paths allow the network to retain high-resolution features from earlier layers. They help preserve spatial information that might be lost during down-sampling, thus improving the accuracy of the reconstructed image (Fig.4.1).

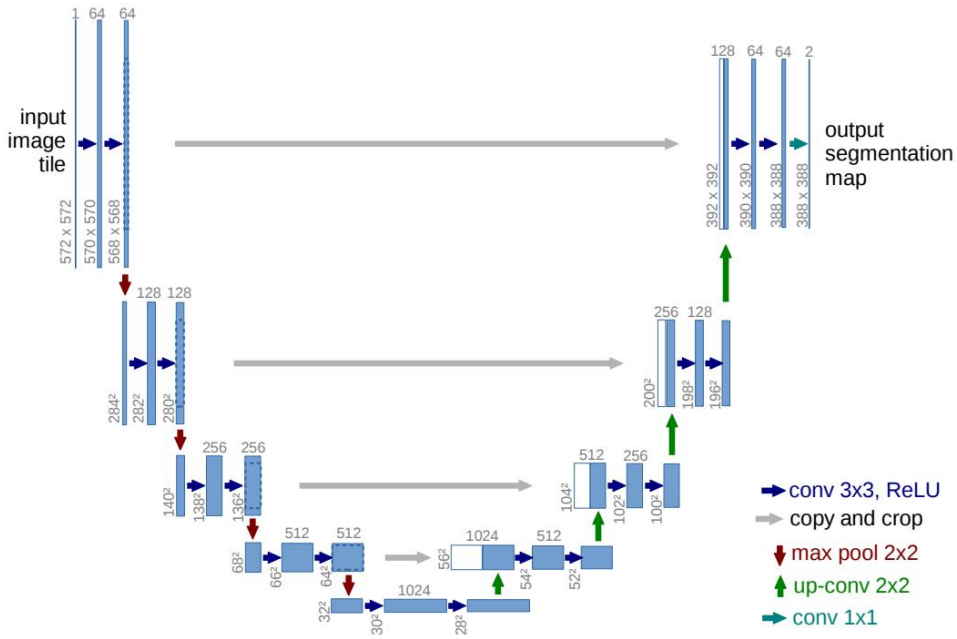


Figure 4.1: The U-Net architecture features an encoder on the left that reduces spatial dimensions while extracting key features through convolutional and downsampling layers. On the right, the decoder reconstructs the spatial dimensions using upsampling layers to produce a segmented output. Skip connections (in gray) between corresponding encoder and decoder layers preserve spatial information and improve segmentation accuracy. Image from Ronneberger et al. (2015).

In this work, U-Net and its variants have been adapted for regression tasks, specifically to calculate the most energetic travel time. The regression problem involves predicting contin-

uous values rather than class labels, and U-Net's ability to retain detailed spatial information makes it suitable for such tasks.
The standard U-Net architecture was extended and tested with two notable variants: the Attention U-Net (AUnet) and the Residual Attention U-Net (ResAUnet).

- Attention U-Net (AUnet): This variant (Oktay et al. 2018) [27] incorporates attention mechanisms to improve the model's ability to focus on relevant features while ignoring irrelevant ones. Attention blocks adjust the contribution of different features dynamically, which can enhance performance in complex scenarios where certain regions of the image are more significant than others.

- Residual Attention U-Net (ResAUnet): This model (Ni et al. 2019) [28] combines residual connections with attention mechanisms. Residual blocks help in training deeper networks by mitigating the vanishing gradient problem (see chapter 4.1.4), while attention blocks enhance feature selection. This combination aims to improve both feature learning and focus on important regions, potentially leading to better results.

In summary, U-Net and its variants offer powerful solutions for image segmentation and regression tasks, with attention and residual mechanisms providing additional improvements in performance and accuracy.

### 4.1.2 Algorithm: U-Net training

1. **Initialize:** Set $\theta$ for $\mathcal{M}(\theta)$.

2. **For epoch** $e = 1$ **to** $E$**:**

   - **For each batch** $\{(x_j, y_j)\}_{j=1}^{B}$**:**

     (a) **Forward Pass:**

$$
\begin{aligned}
&\text{for } i = 1 \text{ to } L-1: \\
&\quad c_{i+1} = \sigma(\text{BN}(\text{Conv2d}(p_i, \theta_{2i-1}))) \\
&\quad c_{i+1} = \sigma(\text{BN}(\text{Conv2d}(c_{i+1}, \theta_{2i}))) \\
&\quad p_{i+1} = \text{MaxPool}(c_{i+1}) \\
&\text{end for} \\
&c_L = \sigma(\text{BN}(\text{Conv2d}(p_{L-1}, \theta_{2L-1}))) \\
&c_L = \sigma(\text{BN}(\text{Conv2d}(c_L, \theta_{2L}))) \\
&\text{for } i = L \text{ downto } 1: \\
&\quad u_i = \text{UpSampling}(c_{i+1}) \\
&\quad u_i = \text{Concat}(u_i, c_i) \\
&\quad u_i = \sigma(\text{BN}(\text{Conv2d}(u_i, \theta_{2i+1}))) \\
&\quad u_i = \sigma(\text{BN}(\text{Conv2d}(u_i, \theta_{2i+2}))) \\
&\text{end for} \\
&\hat{y}_j = \text{Conv2d}(u_1, \theta_f)
\end{aligned}
$$

     (b) **Loss Computation:**

$$\mathcal{L}(\theta) = \frac{1}{B} \sum_{j=1}^{B} \|\hat{y}_j - y_j\|_1$$

     (c) **Backward Pass & Parameter Update:**

$$
\begin{aligned}
\nabla_\theta &\leftarrow \frac{\partial \mathcal{L}}{\partial \theta} \\
\theta &\leftarrow \mathcal{O}(\theta, \nabla_\theta)
\end{aligned}
$$

**Glossary of Terms:**

- $\mathcal{D} = (x_i, y_i)_{i=1}^{N}$: Dataset. The collection of $N$ samples, where each sample consists of an input image $x_i$ and its corresponding ground truth $y_i$.
- $x_i \in \mathbb{R}^{C \times H \times W}$: Input Image. A tensor representing an image with $C$ channels, height $H$, and width $W$.
- $y_i \in \mathbb{R}^{H \times W}$: Output Image. A tensor representing the ground truth or target image with height $H$ and width $W$.
- $E$: Number of Epochs. The total number of complete passes through the entire dataset during training.
- $B$: Batch Size. The number of samples processed together in one forward and backward pass during training.
- $\mathcal{M}(\theta)$: U-Net Model. The U-Net architecture parameterized by $\theta$, where $\theta$ represents the model parameters.
- $\mathcal{L}$: Loss Function. A function used to measure the discrepancy between the predicted output and the ground truth, guiding the optimization process.
- $\mathcal{O}$: Optimizer. An algorithm used to update the model parameters $\theta$ based on the gradients computed during training.
- $N$: Number of Samples. The total number of samples in the dataset $\mathcal{D}$.
- $C$: Number of Channels. The number of channels in the input image $x_i$. For example, $C = 3$ for RGB images.
- $H$: Height. The height of the input image $x_i$ and the output image $y_i$.
- $W$: Width. The width of the input image $x_i$ and the output image $y_i$.
- $\sigma$: Activation Function. Typically, $\sigma$ represents the activation function applied after each convolution. Common choices include the Rectified Linear Unit (ReLU) function, which is defined as $\sigma(x) = \max(0, x)$.

- BN: Batch Normalization. A technique used to normalize the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. This helps to stabilize and accelerate training.
- Conv2d: 2D Convolution. A convolutional layer that applies 2D filters (kernels) to the input to extract features. It is defined by the convolution operation Conv2d$(x, \theta)$ where $x$ is the input and $\theta$ represents the filter weights.
- MaxPool: Max Pooling. A down-sampling operation that reduces the spatial dimensions of the input by taking the maximum value within a specified window. It is used to reduce the computational load and capture dominant features.
- UpSampling: Up-Sampling. An operation that increases the spatial dimensions of the input feature map, often using methods such as nearest-neighbor interpolation or transposed convolution.
- Concat: Concatenation. An operation that concatenates two feature maps along a specified dimension, typically the channel dimension, to combine features from different layers.
- $\theta_f$: Final Convolution Weights. The weights used in the final convolutional layer to produce the output predictions from the last feature map $u_1$.

### 4.1.3 Forward Pass: Attention U-Net (A-Unet)

For the Attention U-Net (AUnet) variant, the overall architecture follows the same structure as the standard U-Net, with the exception of the forward pass step. The forward pass for A-Unet incorporates attention gates, which refine the feature maps to focus on important regions.

$$
\begin{aligned}
&\text{for } i = 1 \text{ to } L-1: \\
&\quad c_{i+1} = \sigma\left(\text{BN}\left(\text{Conv2d}(p_i, \theta_{2i-1})\right)\right) \\
&\quad c_{i+1} = \sigma\left(\text{BN}\left(\text{Conv2d}(c_{i+1}, \theta_{2i})\right)\right) \\
&\quad p_{i+1} = \text{MaxPool}(c_{i+1}) \\
&\text{end for}
\end{aligned}
$$

$$
\begin{aligned}
c_L &= \sigma\left(\text{BN}\left(\text{Conv2d}(p_{L-1}, \theta_{2L-1})\right)\right) \\
c_L &= \sigma\left(\text{BN}\left(\text{Conv2d}(c_L, \theta_{2L})\right)\right)
\end{aligned}
$$

$$
\begin{aligned}
&\text{for } i = L \text{ downto } 1: \\
&\quad u_i = \text{UpSampling}(c_{i+1}) \\
&\quad u_i = \text{Concat}(u_i, c_i) \\
&\quad u_i = \text{AttentionGate}(g = u_i, x = c_i) \\
&\quad u_i = \sigma\left(\text{BN}\left(\text{Conv2d}(u_i, \theta_{2i+1})\right)\right) \\
&\quad u_i = \sigma\left(\text{BN}\left(\text{Conv2d}(u_i, \theta_{2i+2})\right)\right) \\
&\text{end for}
\end{aligned}
$$

$$
\hat{y}_j = \text{Conv2d}(u_1, \theta_f)
$$

The Attention Gate (AG) is a mechanism designed to enhance the performance of convolutional neural networks, particularly in tasks like image segmentation, by dynamically focusing on the most relevant regions of the input. In the architecture described by Oktay et al. 2018 [27], the Attention Gate is applied within a U-Net framework to refine the feature maps passed from the encoder to the decoder.

**Feature Map** $x$:

- $x$ represents the input feature map coming from the encoder path of the U-Net. This feature map contains detailed spatial information but may include both relevant and irrelevant features for the segmentation task.

**Gating Signal** $g$:

- $g$ is the gating signal, a feature map derived from a higher-level, coarser layer of the decoder path. This signal carries contextual information and helps guide the attention mechanism to focus on the relevant parts of $x$.

**Operation of the Attention Gate**:

- The Attention Gate takes both $x$ and $g$ as inputs. Through a series of linear transformations (convolutions with 1×1×1 kernels), activation functions and element-wise operations, the gate computes the attention coefficients $\alpha$. These coefficients indicate the importance of each spatial region in $x$ based on the contextual information provided by $g$.

- The coefficients $\alpha$ are then applied to the feature map $x$, effectively scaling it so that more relevant features are emphasized while less relevant ones are suppressed.

**Output**:

- The output of the Attention Gate is the modulated feature map, where important features are enhanced, and irrelevant ones are attenuated. This output is then passed to the subsequent layers of the U-Net decoder for further processing.

By integrating this mechanism (Fig.4.2), the network becomes more effective at focusing on the parts of the image that are most important for accurate segmentation, improving the overall performance of the model.

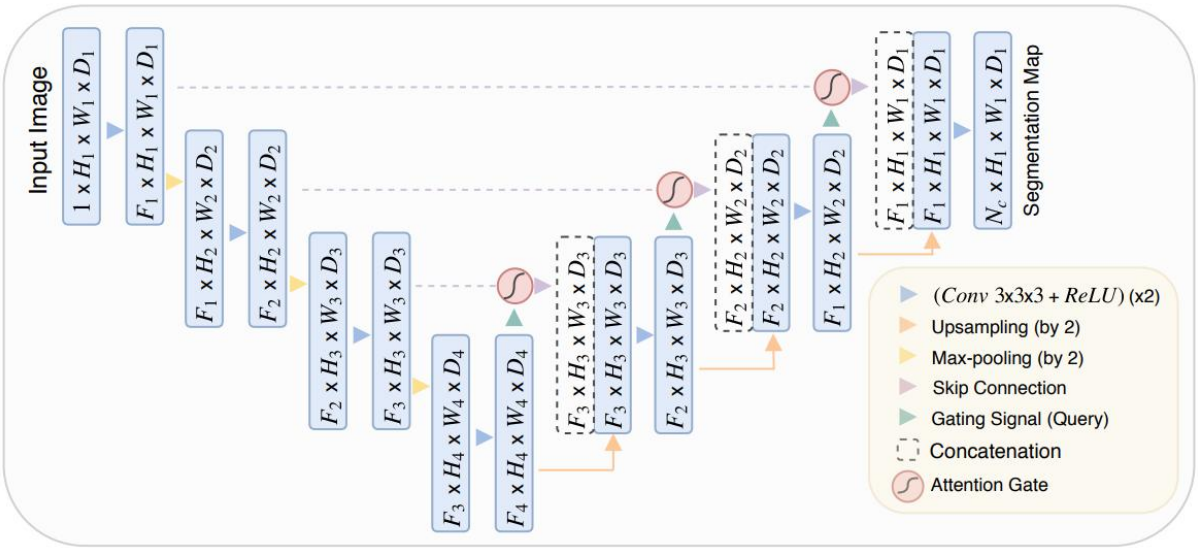


Figure 4.2: A schematic representation of the implemented Attention U-Net. Attention Gates filter the features propagated through the skip connections. Image taken from (Oktay et al. 2018) [27]

### 4.1.4 Forward Pass: Residual Attention U-Net (Res-A-Unet)

For the Residual Attention U-Net (ResAUnet) variant, the overall architecture follows the same structure as the standard U-Net, with the exception of the forward pass step. The forward pass for Res-A-Unet introduces additional components, including residual and attention mechanisms.

$$
\begin{aligned}
&\text{for } i = 1 \text{ to } L: \\
&\quad c_{i+1} = \text{ResidualBlock}(p_i, \theta_{2i-1}) \\
&\quad p_{i+1} = \text{MaxPool}(c_{i+1}) \\
&\text{end for} \\
&\\
&\text{center} = \text{ResidualBlock}(p_{L-1}, \theta_{2L}) \\
&\\
&\text{for } i = L \text{ downto } 1: \\
&\quad u_i = \text{UpSampling}(c_{i+1}) \\
&\quad u_i = \text{Concat}(u_i, c_i) \\
&\quad u_i = \text{AttentionBlock}(g = u_i, x = c_i) \\
&\quad u_i = \text{ResidualBlock}(u_i, \theta_{2i+1}) \\
&\text{end for} \\
&\\
&\hat{y} = \text{Conv2d}(u_1, \theta_f)
\end{aligned}
$$

The ResidualBlock (Fig.4.3) is a type of block designed to improve the training of deep neural networks and to tackle the vanishing gradient problem, which is a common issue in training deep neural networks. The vanishing gradient problem occurs when gradients of the loss function with respect to the network's weights become exceedingly small as they are propagated backward through the network during training. This issue is particularly pronounced in deep networks with many layers, where gradients can diminish exponentially through each layer. Consequently, weight updates become minimal, leading to very slow or halted learning in the earlier layers of the network.

ResidualBlocks address this challenge by following the principles of residual learning. Specifically, they pass the input through a series of convolutional layers and then add the original input back to the output of these layers. This shortcut connection helps to preserve the gradient flow and mitigate the vanishing gradient problem, thereby improving the network's training efficiency and performance.

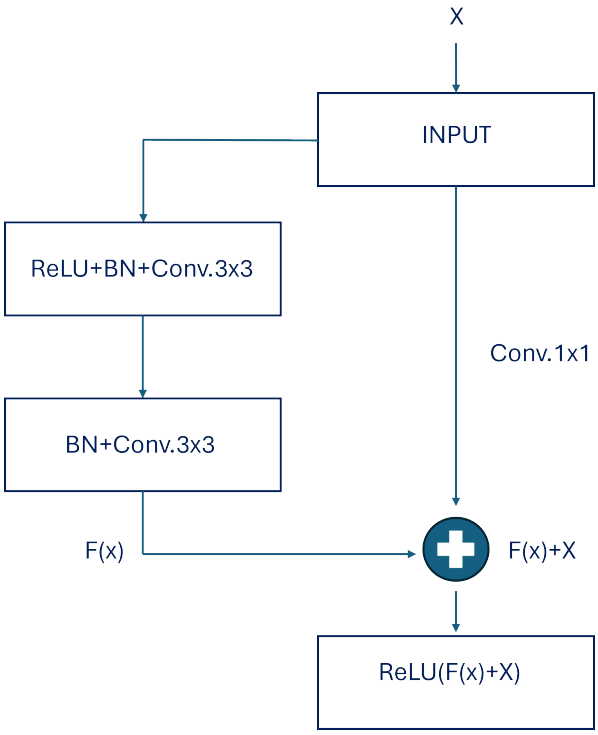


Figure 4.3: Simple graph explaining the composition of the residual block.

Specifically, the Residual Block operates as follows:

- The input feature map is first passed through a convolutional layer with a $3 \times 3$ kernel, followed by a batch normalization and a ReLU activation function.
- The result is then processed through another convolutional layer with a $3 \times 3$ kernel and batch normalization, if batch normalization is applied. This step refines the features extracted by the first convolutional layer.
- Simultaneously, the original input is processed through a $1 \times 1$ convolutional layer to match the dimensions of the output from the second convolutional layer.
- The output of the second convolutional layer is added to the output of the $1 \times 1$ convolutional layer.

- Finally, a ReLU activation function is applied to the summed result, producing the final output of the Residual Block.

## 4.2 Neural Style Transfer (NST)

Neural Style Transfer (NST) is an application of deep learning that enables the synthesis of images by combining the content of one image with the style of another (Fig.4.4). This is achieved by leveraging the powerful feature extraction capabilities of Convolutional Neural Networks (CNNs). Introduced by (Gatys et al. 2015) [29], NST has opened new avenues in both artistic image creation and the study of visual perception. NST has a wide range of applications in various fields; specifically in machine learning, it can be used to generate augmented datasets by creating stylized versions of existing images, which can improve the robustness of models. In this thesis work, it was employed to create a new training dataset, starting from the original training set and using style images from the test dataset, which proved to be highly useful for achieving the objectives of this study.

### 4.2.1 How Neural Style Transfer Works

The core idea behind NST is to manipulate the neural representations of images in such a way that the content from one image is preserved while adopting the stylistic features from another. This is achieved by using a pre-trained deep neural network, typically a VGG network, to extract high-level features from both the content and style images. In this Thesis a pre-trained VGG-19 has been used.

#### Content Representation

The content of an image is represented by the activations of the higher layers within a CNN when the image is processed. These activations are the output of the neurons in those layers and capture the high-level structure and objects, focusing on the arrangement and identity of elements within the image while abstracting away finer details like textures and colors. This makes the higher-layer activations ideal for content representation NST, as they reflect the semantic content of the image, as shown in the bottom images of Fig.4.5.

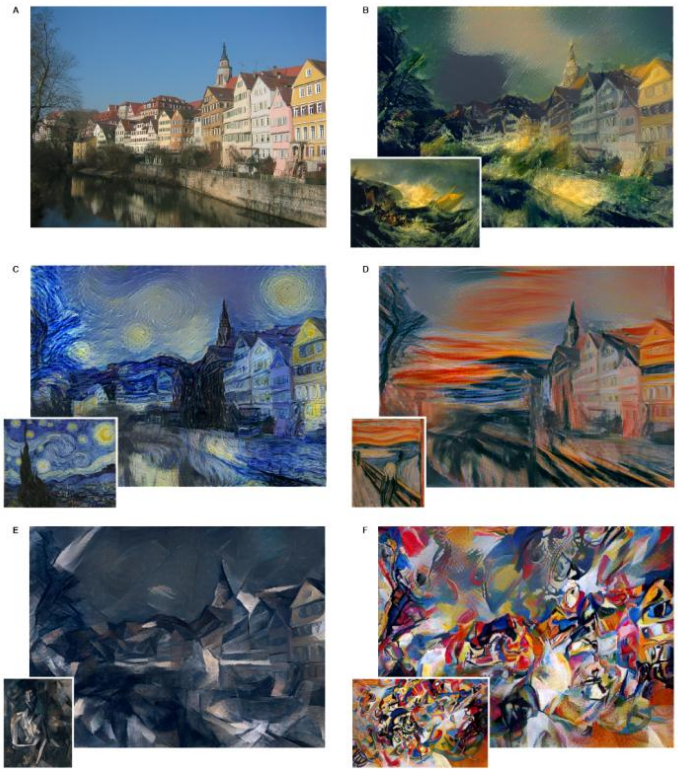

Figure 4.4: Image at the top left: content image. Subscripts images: style content. The others images are a combination of the content of the photograph with the style of several well-known artworks using NST. Image taken from (Gatys et al. 2015) [29]

### Style Representation

Style, on the other hand, is captured by the correlations between the different feature maps within the network. These correlations are computed using the Gram matrix (Gatys et al. 2015) [29], which measures the similarity between different feature maps. By using multiple layers of the CNN, NST captures style at different scales (images at the top of Fig.4.5), from fine textures to more abstract, global patterns.

### The Loss Function

The NST algorithm operates by defining a loss function that measures the deviation of the generated image from the desired content and style representations. This loss function consists of two components:

- Content Loss: This measures the difference between the feature activations of the content image and the generated image at highest layer of the CNN.

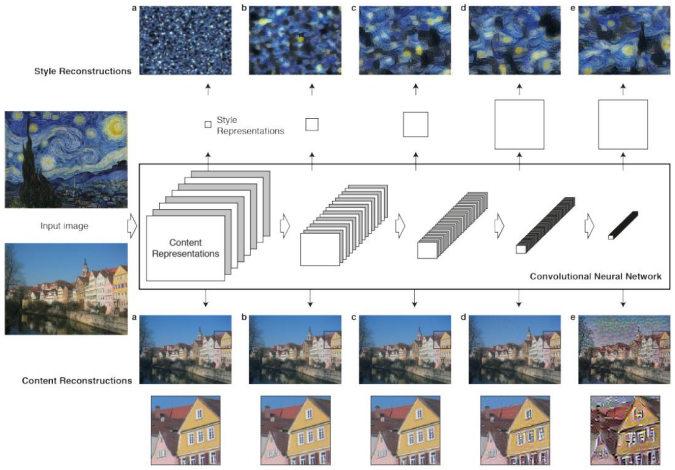


Figure 4.5: Styles and content can be computed at different layers of the Convolutional Neural Network (CNN). Image taken from (Gatys et al. 2015) [29]

- Style Loss: This quantifies the difference between the Gram matrices of the style image and the generated image across multiple layers of the CNN.

### 4.2.2 Mathematical Foundations of Neural Style Transfer

The mathematical foundation of NST lies in the feature extraction capabilities of CNNs for content representation and the use of Gram matrices for style representation.

#### Feature Extraction Using CNNs

Given an input image $\mathbf{x}$, a CNN processes it through a series of convolutional layers, producing a set of feature maps at each layer. For a layer $l$, the feature map is denoted by $\mathbf{F}^l \in \mathbb{R}^{N_l \times M_l}$, where $N_l$ is the number of filters and $M_l$ is the spatial dimension (height $\times$ width) of the feature map.

#### Content Loss

The content loss measures the difference between the feature representations of the content image $\mathbf{p}$ and the generated image $\mathbf{x}$ at a specific layer $l$ of the CNN. Each layer $l$ of the CNN produces feature maps, which are flattened into one-dimensional vectors. Let $P^l$ and $F^l$ denote the feature representations of the content image and the generated image, respectively, at layer $l$. The content loss is defined as:

$$L_{\text{content}}(\mathbf{p}, \mathbf{x}, l) = \frac{1}{2} \sum_{i,j} \left( F_{ij}^l - P_{ij}^l \right)^2 \tag{4.1}$$

Here, $F_{ij}^l$ and $P_{ij}^l$ represent the activations of the $i$-th filter at position $j$ in layer $l$ for the generated and content images, respectively. The process of generating the image $\mathbf{x}$, is initialized with an image of random noise. This random noise image is iteratively refined using gradient descent to minimize the content loss, gradually transforming it into an image that closely matches the content representation of the original image $\mathbf{p}$ at the chosen layer $l$.
The derivative of the content loss with respect to the activations in layer $l$ is:

$$\frac{\partial L_{\text{content}}}{\partial F_{ij}^l} = \begin{cases} F_{ij}^l - P_{ij}^l & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l \leq 0 \end{cases} \tag{4.2}$$

**Style Loss**

The style representation is captured through the Gram matrix $\mathbf{G}^l \in \mathbb{R}^{N_l \times N_l}$, which encodes the correlations between feature maps at layer $l$. The element $G_{ij}^l$ of the Gram matrix represents the inner product between the vectorized feature maps $i$ and $j$:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \tag{4.3}$$

To match the style of the generated image $\mathbf{x}$ to that of the style image $\mathbf{a}$, the mean-squared distance between the Gram matrices of the style image and the generated image is minimized. The generated image $\mathbf{x}$ is initially created as a random noise image. During the optimization process, this noise image is iteratively adjusted to reduce the style loss, bringing its Gram matrix closer to that of the style image. Let $A_{ij}^l$ and $G_{ij}^l$ represent the Gram matrices of the style image and the generated image at layer $l$. The contribution of layer $l$ to the total style loss is defined as:

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} \left( A_{ij}^l - G_{ij}^l \right)^2 \tag{4.4}$$

The total style loss is computed by summing the contributions from each layer:

$$L_{\text{style}}(\mathbf{a}, \mathbf{x}) = \sum_l w_l E_l \tag{4.5}$$

where $w_l$ are the weighting factors for the contribution of each layer to the total style loss. The derivative of the style loss with respect to the activations $F_{ij}^l$ is computed as follows:

$$\frac{\partial E_l}{\partial F_{ij}^l} = \begin{cases} \frac{1}{N_l^2 M_l^2} \left( \left(F_{ij}^l\right)^T \left(A_{ij}^l - G_{ij}^l\right) \right) & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l \leq 0 \end{cases} \tag{4.6}$$

**Total Loss**

The total loss function is a weighted sum of the content and style losses:

$$L_{\text{total}}(\mathbf{p}, \mathbf{a}, \mathbf{x}) = \alpha L_{\text{content}}(\mathbf{p}, \mathbf{x}) + \beta L_{\text{style}}(\mathbf{a}, \mathbf{x}) \tag{4.7}$$

where $\alpha$ and $\beta$ are appropriately chosen scalar weights.

### 4.2.3 Hyperparameters in Neural Style Transfer

NST involves several hyperparameters that can significantly affect the quality of the generated images (Fig.4.6):

Figure 4.6: The column shows different relative weightings between the content and style reconstruction. On the left, the content is emphasized, while on the right, the style takes precedence. This image, taken from (Gatys et al. 2015) [29], refers to the same case study as in Fig.4.5, using the same style and content images.

- **Content Weight ($\alpha$)**: A scalar that controls the importance of content preservation in the generated image.
- **Style Weight ($\beta$)**: A scalar that determines the strength of the style transfer.
- **Layer Selection**: The choice of layers from which to extract content and style representations can greatly influence the output. Typically, higher layers are used for content, and a combination of lower to higher layers is used for style.

- **Optimization Method**: Gradient descent is commonly used, with options like L-BFGS (Liu and Nocedal, 1989) [30] or Adam optimizers (Kingma, 2017) [31]. The choice of optimizer and its parameters, such as learning rate, can affect convergence and image quality.

Tuning these hyperparameters is often a process of trial and error, guided by the specific artistic or practical goals.

## 4.3 Ensembling Network

Ensemble learning combines predictions from multiple models to improve overall performance (Zhou, 2012) [32]. The technique involves optimizing weights to determine the ideal contribution of each model's predictions (Fig.4.7).

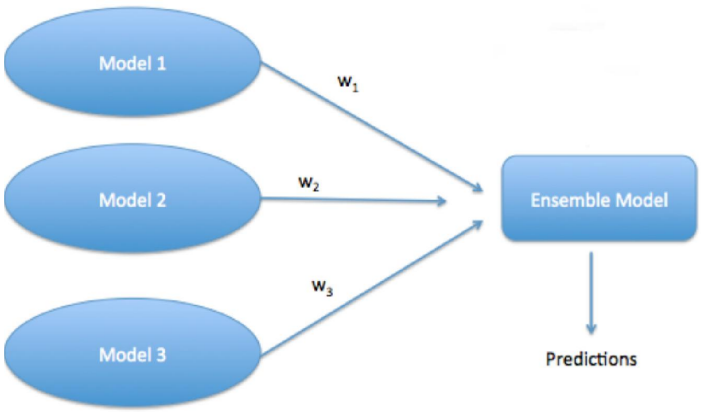


Figure 4.7: The image shows an ensemble network that combines the outputs of three neural networks. Each output is multiplied by a learned weight w, and the weighted outputs are summed to produce a final prediction that is closer to the true label.

The process is structured as follows:

1. **Model Training**: Multiple models are trained independently. Each model provides predictions for the same task but with potentially different characteristics due to varying hyperparameters, or architectures.

2. **Prediction Aggregation**: Predictions from all models are aggregated. This is typically done by calculating a weighted sum of the predictions, where each model's output is scaled by a corresponding weight.

3. **Weight Optimization**: The weights used for aggregating predictions are optimized to minimize the error of the combined prediction. This is achieved by defining a loss function that measures the discrepancy between the combined predictions and the true labels.

4. **Evaluation**: After optimizing the weights, the performance of the ensemble is evaluated. The final model's accuracy or error is assessed to ensure that the optimized weights improve prediction performance compared to individual models.

This structure allows for the systematic improvement of prediction accuracy by effectively combining multiple models and fine-tuning their contributions.

## 4.4 Diffusion models

Deep generative models have revolutionized various fields within machine learning, particularly in tasks involving image synthesis, editing, interpolation, colorization, denoising, and super-resolution. Diffusion probabilistic models, often simply referred to as Diffusion models (Sohl-Dickstein, 2015; Ho, 2020) [33, 34], have recently gained prominence due to their exceptional performance, frequently surpassing traditional algorithms like Generative Adversarial Networks (GANs; Goodfellow, 2014) [35] and Variational Autoencoders (VAEs; Kingma, 2013) [36]. Currently, they represent the state of the art in generative models and have only been used in the field of geophysics for a few years.
Generative models aim to learn the underlying distribution of the observable data, such as images. They focus on understanding and replicating how these data points are distributed in their feature space. In diffusion models, this is achieved by adding controlled amounts of noise to the data, gradually corrupting the data points. The model is then trained to reverse this process by learning to denoise the data and recover the original distribution, effectively capturing the underlying structure of the feature space (Fig.4.8).
There are various methods for estimating probability distribution $p(x)$, including autoregressive models, latent variable models, flow-based models, and energy-based models. Each of these approaches has its own way of addressing the problem of modeling data distributions. Here the focus is on latent variable models, as they form the foundation for diffusion models.

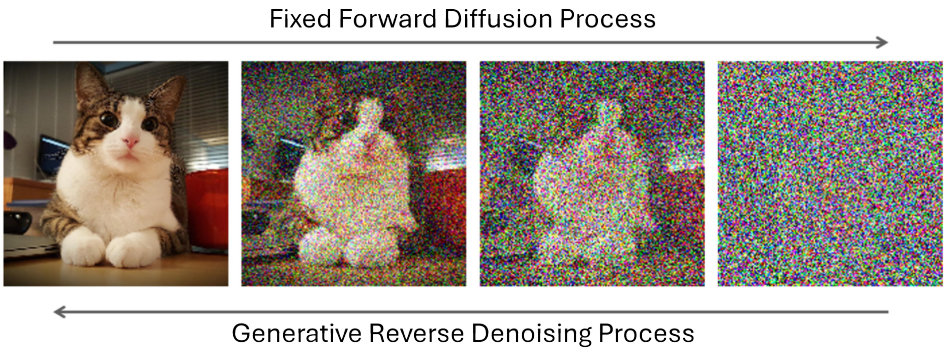


Figure 4.8: In diffusion models, the probability distribution $p(x)$ of data is obtained by progressively adding noise to the images and then learning how to reverse this process, effectively reconstructing the original image.

### 4.4.1 Latent Variable Models

Latent variable models introduce hidden or latent variables z that help explain the observed data x. In the context of diffusion models, these latent variables represent unobserved factors that capture the underlying structure of the input, which is essential for generating or reconstructing the observable information. They serve as an intermediary representation that aids in modeling complex distributions by encoding the information necessary to recover the original inputs after progressive noise has been added. The joint distribution $p(x, z)$ represents the combined distribution of both observable and latent variables. The marginal distribution $p(x)$, is obtained by integrating out the latent variables:

$$p_\theta(x) = \int_z p_\theta(x, z) \; = \int_z p_\theta(x|z)\, p_\theta(z) \tag{4.8}$$

where:

- $p_\theta(x|z)$ is the likelihood of $x$ given $z$, indicating how likely $x$ is for a specific latent variable $z$.
- $p_\theta(z)$ is the prior distribution over the latent variables, representing the beliefs about $z$ before seeing any data.

Since the true distribution $p(x)$ is often unknown, approximated methods are used. Variational inference is a common approach, which optimizes the Evidence Lower Bound (ELBO):

$$\log p(x) \geq \mathbb{E}_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] - \mathrm{KL}[q_\phi(z|x)\|p(z)], \tag{4.9}$$

where:

- $q_\phi(z|x)$ is the variational posterior, an approximation of the true posterior distribution of $z$ given $x$.
- KL denotes the Kullback-Leibler divergence, a measure of how one probability distribution diverges from another (Kullback and Leibler, 1951) [37].

### 4.4.2 Hierarchical Latent Variable Models

Hierarchical models use multiple layers of latent variables to explain complex data structures. For instance, with two latent variables $z_1$ and $z_2$, the joint distribution is:

$$p_\theta(x) = \int_{z_1} \int_{z_2} p_\theta(x, z_1, z_2) \; = \int_{z_1} \int_{z_2} p_\theta(x|z_1)\, p_\theta(z_1|z_2)\, p_\theta(z_2) \tag{4.10}$$

where:

- $p_\theta(x|z_1)$ is the likelihood of $x$ given $z_1$.
- $p_\theta(z_1|z_2)$ is the conditional distribution of $z_1$ given $z_2$.
- $p_\theta(z_2)$ is the prior distribution of $z_2$.

The ELBO for hierarchical models is:

$$\log p(x) \geq \mathbb{E}_{z_1 \sim q_\phi(z_1|z_2)}[\log p_\theta(x|z_1)] - \mathrm{KL}[q_\phi(z_1|x) \| p_\theta(z_1|x)] - \mathrm{KL}[q_\phi(z_2|z_1) \| p(z_2)]. \tag{4.11}$$

### 4.4.3 Derivation of Diffusion Models

Diffusion models can be viewed as a particular instance of hierarchical latent variable models, where the inference process is defined without relying on learnable parameters. In Fig(4.9) can be seen the three mentioned models.

In practice, diffusion models incrementally add noise to the data following a Markov chain process and then learn to reverse this process to generate data samples. In Fig(4.10) an example from Durall (2022) [38] illustrates seismic data being progressively contaminated by noise and then recovered from noise back to its original form. The idea of applying the diffusion method to this thesis work was inspired by Durall's approach. These models are characterized by a hierarchical structure where the final latent distribution $q(x_T)$ approaches a standard Gaussian distribution $\mathcal{N}(0, I)$ as $T$ (the number of steps) increases.

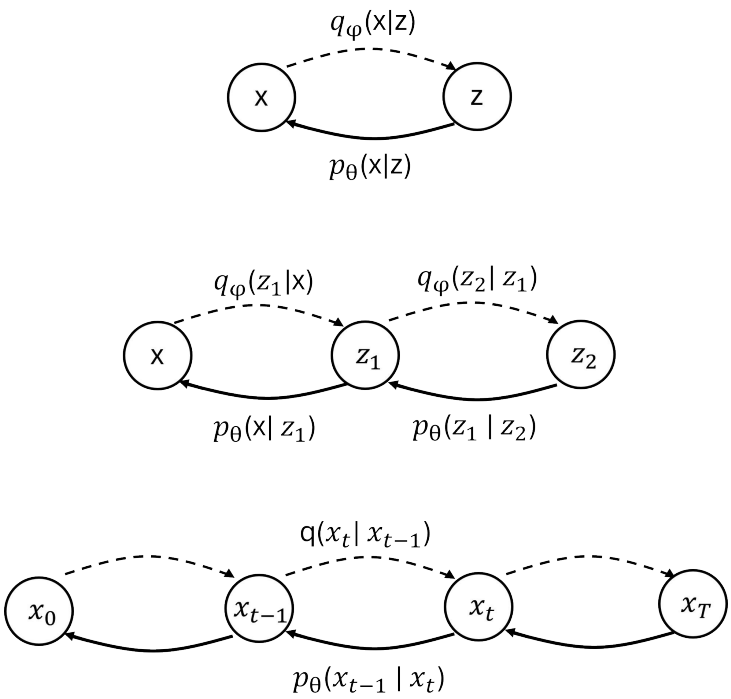


Figure 4.9: Scheme of the different latent variable models. (Top) Single latent variable model. (Center) Hierarchical latent variable model. (Bottom) Diffusion model. Image taken from (Durall, 2022) [38]

**Forward Diffusion Process** In the forward diffusion process, Gaussian noise $\epsilon$ is added to the input image $x_0$ over $T$ steps, resulting in progressively noisier images:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}) = \prod_{t=1}^{T} \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I), \tag{4.12}$$

where:

- $\mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$ denotes the Gaussian distribution of $x_t$ given $x_{t-1}$, with $\sqrt{1-\beta_t}x_{t-1}$ as the mean and $\beta_t I$ as the covariance matrix.
- $\beta_t$ is a variance schedule that determines how the noise level changes at each step.

**Parameterized Reverse Process** The reverse process learns to reconstruct the original data by denoising from $x_T$ back to $x_0$.

The model parameterizes the reverse transitions as:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t) = p(x_T) \prod_{t=1}^{T} \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)), \tag{4.13}$$

where:

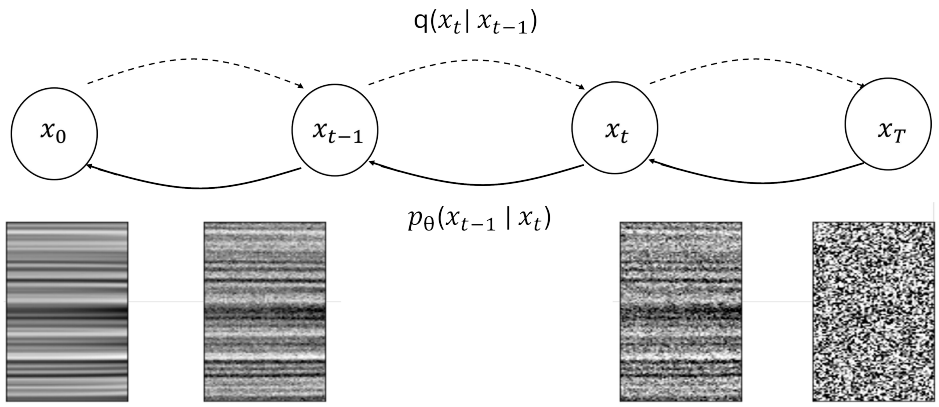


Figure 4.10: Denoising diffusion process. While the Markov chain of the forward diffusion gradually adds noise to the input (dash arrows), the reverse process removes it stepwise (solid arrows). Image taken from (Durall, 2022) [38]

- $\mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t))$ denotes the Gaussian distribution of $x_{t-1}$ given $x_t$, with $\mu_\theta$ and $\Sigma_\theta$ representing the learned mean and covariance, respectively.

**Training Objective** The training objective is to maximize the lower bound on the log-likelihood of the data. This is achieved by minimizing the discrepancy between predicted and actual noise components:

$$\log p(x) \approx \sum_{t=2}^{T} \left\| \epsilon - \epsilon_\theta \left( \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} \epsilon, t \right) \right\|^2, \tag{4.14}$$

where:

- $\epsilon_\theta$ is the model's prediction of the noise component $\epsilon$ at timestep $t$.
- $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{t} \alpha_i$ are accumulated noise schedules used in training.

### 4.4.4 Reverse Process Architecture

The diffusion model used in this thesis employs a time-conditional U-Net (Ho, 2020) [34] as the backbone for the reverse process. This architecture consists of:

- A stack of residual layers and downsampling convolutions, followed by a stack of residual layers with upsampling convolutions. This structure helps to capture multi-scale features.

- Skip connections that link layers with the same spatial dimensions to maintain information throughout the network.
- A global attention layer with a single head that integrates timestep embeddings into each residual block, allowing the model to effectively use temporal information.

The U-Net ensures that the input and output dimensions match, which is crucial for accurately learning the noise prediction across the diffusion steps.

### 4.4.5 Concluding remarks

Diffusion models offer a compelling approach in generative modeling by focusing on learning the data distribution $p(x)$, i.e., understanding how data points are distributed in their space. Unlike discriminative models that learn to classify or predict data, diffusion models aim to capture the underlying data distribution itself, enabling them to generate high-quality samples from noise. Their success in tasks like image denoising and data synthesis highlights their robustness and versatility, making them a powerful tool for various applications in data processing.

# Chapter 5

# Workflow

This chapter presents the workflow followed in this research, detailing the key steps from parameter tuning to final data migration. The workflow bridges the theoretical concepts and the empirical results, providing a clear narrative of the methodologies employed.

.

- The process began by creating the initial dataset from synthetic velocity models and tuning the parameters of Devito for forward modeling. Key configurations were established, including maximum recording time, the use of a Ricker wavelet, finite difference (FD) orders for temporal and spatial dimensions, and the specification of the absorbing boundary thickness. These settings were chosen to optimize the forward modeling process
- Subsequently A method was developed to extract the most energetic arrival time from the solution of the wave equation. This step involved modeling wave propagation and identifying, for every grid position, the moment in time of maximum energy, a crucial aspect for subsequent stages of the research.
- With the method in place, an initial dataset was created. The dataset consisted of velocity model segments, randomly selected from synthetic models, serving as the input data. The corresponding traveltime maps, representing the times of maximum energy arrival, were used as labels. This required replicating the maximum energy arrival time extraction process thousands of times, requiring significant computational resources. To handle this, parallelization techniques were employed, which greatly reduced the

computational burden.

- Attention then shifted to the tuning of the U-Net neural network. This involved adjusting several parameters, including the number of inputs, the type of loss function, optimizers, learning rates, the number of filters, and the size of the dataset. It was found that the network's performance improved significantly when the input included both the velocity model and the first-arrival travel time, the latter calculated using an Eikonal equation solver script. This insight led to more accurate and reliable results from the network.

- However, the initial dataset had a limitation. The ultimate goal was to migrate the most energetic travel times predicted by the neural network on a complex system like Marmousi. In complex velocity models like Marmousi, the most energetic travel time maps can exhibit phenomena such as triplications (for a detailed explanation, refer to paragraph 7.3), which were almost absent in the initial training dataset. To address this issue, the dataset was augmented using Neural Style Transfer (NST; for further details on the technique see chapter 4). In this process, the Marmousi model was used as the style image, while the initial velocity models served as the content images. This approach allowed the dataset to better reflect the complexities of real-world scenarios.

- To further understand the triplication phenomena, a sensitivity study was conducted. Velocity models were gradually smoothed to observe the persistence of triplications. This study provided valuable insights into the critical parameters that influence these phenomena, helping to refine the models and improve the network's predictive capabilities.

- Despite these advancements, a challenge persisted. Although the network trained on the modified NST dataset could predict where triplications should occur, the predictions were low-resolution and appeared as smoothed versions of the originals. To address this, ensemble learning was employed to combine multiple models in order to overcome the challenge.

- An alternative to ensemble learning was proposed with the application of diffusion models, which were used to enhance the resolution of the U-Net's outputs. These models

are in general effective in improving the resolution but at the cost of increased computational time.

- Parallel to these efforts, an algorithm for Kirchhoff migration was developed, with specific focus on tuning parameters such as migration aperture and depth. This step was critical for the accurate migration of the data, ensuring that the final images were of high quality and suitable for detailed analysis.
- Finally, the migration results were compared across different scenarios to evaluate the effectiveness of the approaches used throughout the research. The scenarios included the Marmousi model migrated using first-arrival travel times, the most energetic travel times, the traveltimes obtained from the U-Net trained using the simplest dataset, traveltimes obtained by the U-Net trained using the NST modified dataset, and the traveltimes calculated by the diffusion model. These comparisons revealed valuable insights into the strengths and limitations of each approach, providing a clear understanding of the most effective methodologies for the task at hand.

In conclusion, the workflow outlined in this chapter demonstrates the step-by-step approach taken to address the research problem. Each step was crucial in refining the methods and achieving the final results, which are discussed in detail in the subsequent chapters.

# Chapter 6

# Synthetic velocity models

The neural networks were trained on five synthetic 3D velocity models, which are openly accessible to the public. These datasets encompass a range of geological scenarios, designed to maximize the generalizability of the network's potential applications. Fig.6.1 illustrates slices of these five velocity models: (a) a velocity model with complex salt formations, (b) a velocity model representing a carbonate platform, (c) a velocity model highlighting bending geometries, (d) a velocity model with a salt structure, and (e) a velocity model featuring an overthrust area. From these comprehensive velocity models, smaller 2D sections with a fixed size of 4x4 km were randomly selected, smoothed, and subsequently employed in the training process.

To test the trained neural networks, the Marmousi dataset (Versteeg, 1994) [39] was used as the test dataset. The Marmousi velocity model was extended by appending 200 traces to both its left and right sides. This extension was achieved by replicating the first and last traces of the original model 200 times, respectively. The purpose of this extension was to ensure accurate migration near the lateral boundaries of the velocity model.

As for the training models, the Marmousi model was systematically subdivided into a series of smaller, overlapping velocity sub-models. In total, 1800 sub-models were extracted, each measuring 4 km by 4 km. These sub-models were generated by extracting portions of the original velocity model. The center of the first sub-model was positioned at 2 km along the x-axis of the original model, resulting in its boundaries extending from 0 km to 4 km. Subsequent sub-models were created by shifting the center of each new sub-model by 10 m (equivalent to one column) to the right, relative to the previous sub-model. This incremental shifting process was repeated until the entire model was fully covered by these overlapping 4x4 km

sub-models.

Fig.6.2 shows the Marmousi velocity model, with examples of 4x4 km sub-models extracted from the larger model. The network was then tested on each of these velocity models by predicting the corresponding maximum energy traveltime map. The use of computed traveltime maps enables the migration of seismic data obtained through Devito.

A visual representation of this subdivision process can be found in Fig.6.3, where the velocity sub-models have been replaced with their respective most energetic traveltime maps.

Alongside the traveltime maps, the seismic traces were also calculated using Devito. In Fig.6.4, it is illustrated that from each velocity sub-model, both the corresponding traveltime map and shot gather were obtained.

A second training dataset was created using Neural Style Transfer (NST), which was introduced in paragraph 4.2. This technique was employed to enhance the dataset because the original set was not general enough to fully represent the complexity of the Marmousi model. Here, velocity models from the original dataset were used as content images, while sections of the Marmousi dataset were used as style images (Fig. 6.5). The result are new, more complex velocity models. Specifically, 6,000 images representing velocity models were randomly selected from the original dataset of 12,000 elements, which had been created from the synthetic 3D velocity models. These images were used as content images and combined with style images, all of which were derived from randomly selected sections of the Marmousi model. In total, five style images were extracted from the Marmousi model, and each was flipped to create a total of 10 style images, as shown in Fig. 6.6. To increase the influence of the style over the content during the NST process, the style (denoted by $\beta$) was assigned a weight of $10^7$, while the content loss (denoted by $\alpha$) was set to 1

Due to the high computational cost, NST was applied to only half of the initial 12,000 content images. To maintain the dataset size, data augmentation techniques such as flipping were employed, resulting in a final dataset of 12,000 images. The velocity values in the output images, which originally ranged from 0 to 1, were adjusted to match the velocity range present in the style images.

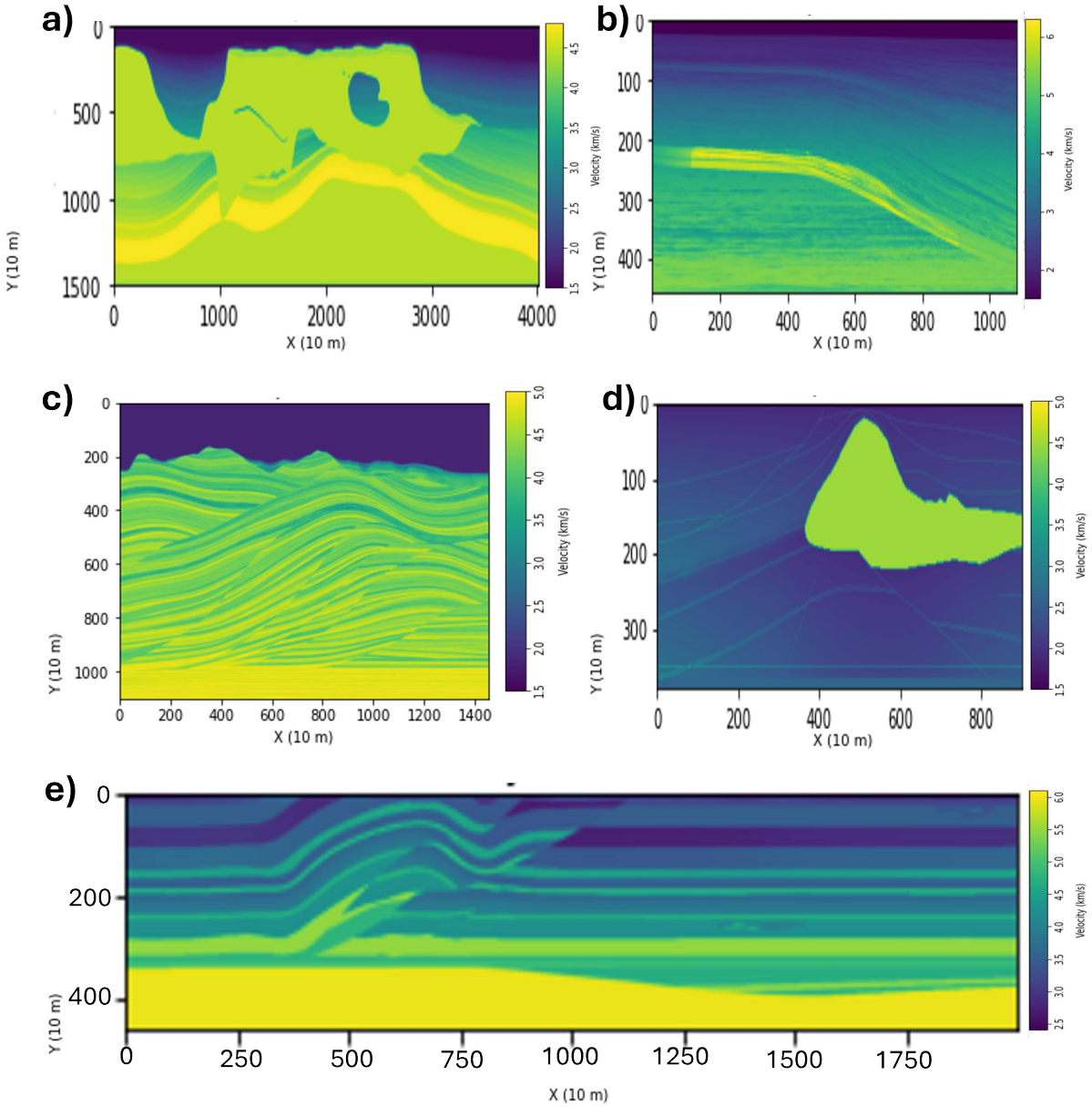


Figure 6.1: Different examples of velocity model used in the training: (a) a velocity model with complex salt formations, (b) a velocity model representing a carbonate platform, (c) a velocity model highlighting bending geometries, (d) a velocity model in which a salt structure is present and (e) a velocity model featuring an overthrust area.

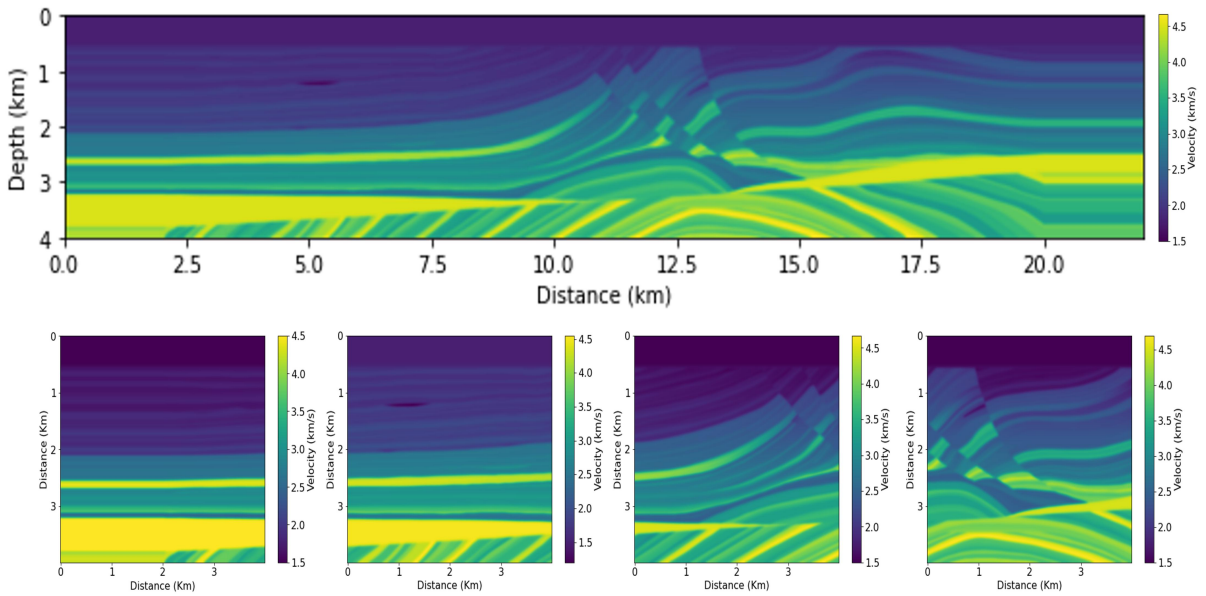


Figure 6.2: Marmosui velocity model (top) and 4 smaller velocity models extracted from it (bottom)

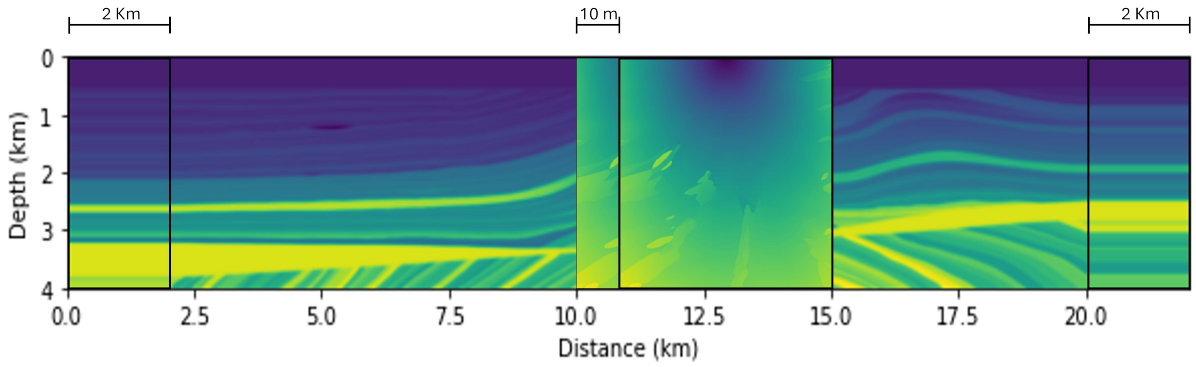


Figure 6.3: This image represents the Marmousi velocity model extended by 200 traces on both the left and right sides. The first and last traces have been replicated. The traveltimes, with dimensions of 4x4 km, were calculated at intervals of 10 meters.

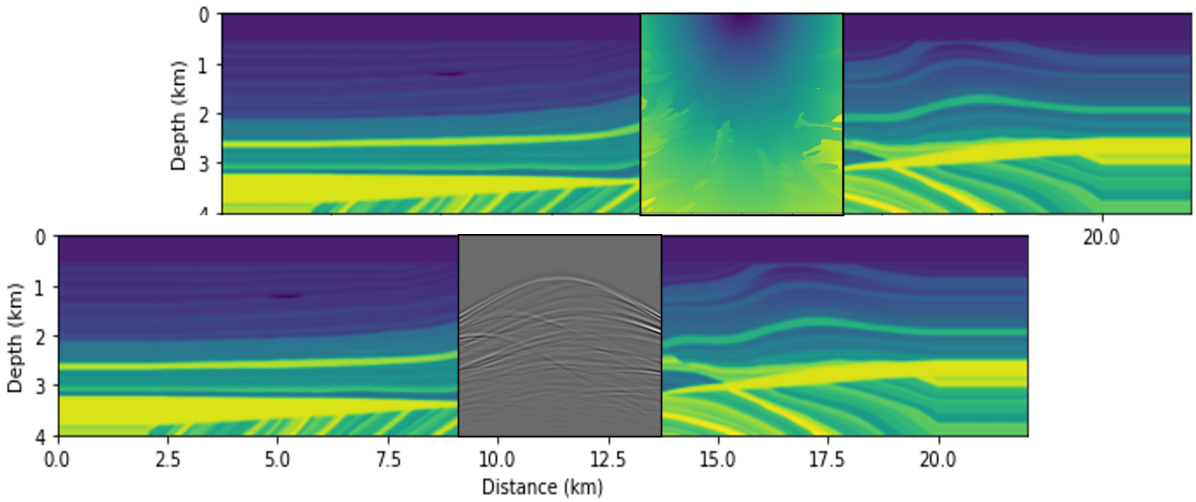


Figure 6.4: In addition to calculating the traveltime map using Devito, the seismic traces were also computed through the software.

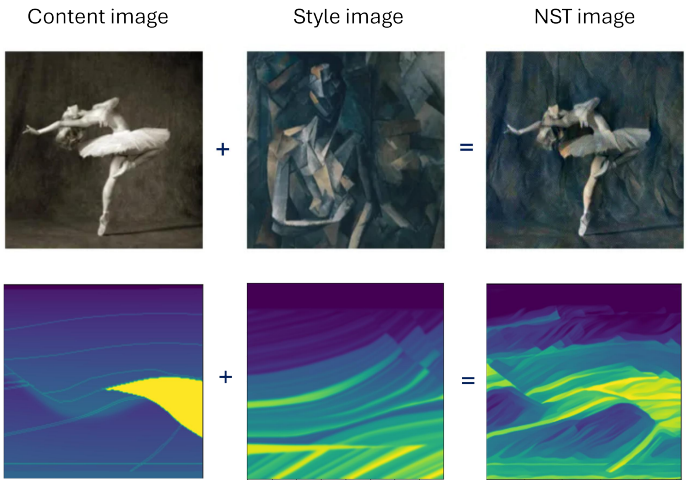


Figure 6.5: Example of Neural Style Transfer (NST) application in art (first row) and in this thesis (second row).

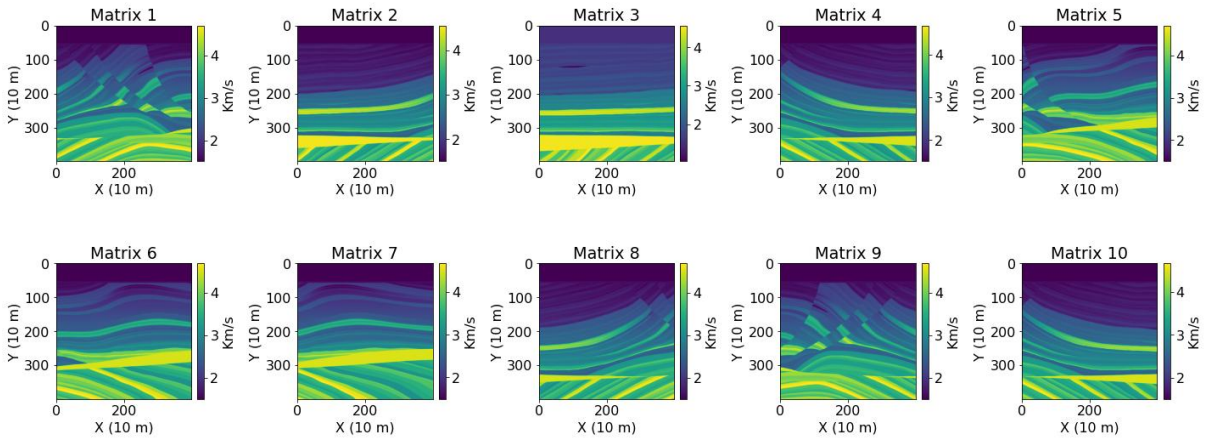


Figure 6.6: These matrices represent all the style images that were used to enhance the complexity of the dataset. In total, 10 style images from the Marmousi dataset were utilized.

# Chapter 7

# Applications

As mentioned in Chapter 6, for this thesis five synthetic 3D velocity models have been used, each representing distinct geological conditions. These models were used to train a neural network, with the intention of enhancing its versatility and adaptability. From each velocity model, 2000 two-dimensional models (samples), each covering an area of 4 km x 4 km, were randomly extracted. Figure 7.1 illustrates the extraction process of these 2D samples and additionally highlights the considerable size of the velocity model containing the salt formations. The larger size of this salt model, as seen in image a of 7.1 compared to the carbonate platform model in image b of 7.1, allowed for the extraction of 4000 samples, in contrast to the 2000 samples extracted from the other models. In total, 12,000 samples were collected, on which a Gaussian filter was used for smoothing. Fig.7.2 shows a representation of some of these 12,000 models. A size of 4x4 km was chosen to match the depths typically encountered during seismic data acquisition campaigns. These samples served as the initial inputs for the neural network, which was tasked with predicting the respective traveltime maps for the most energetic arrivals.

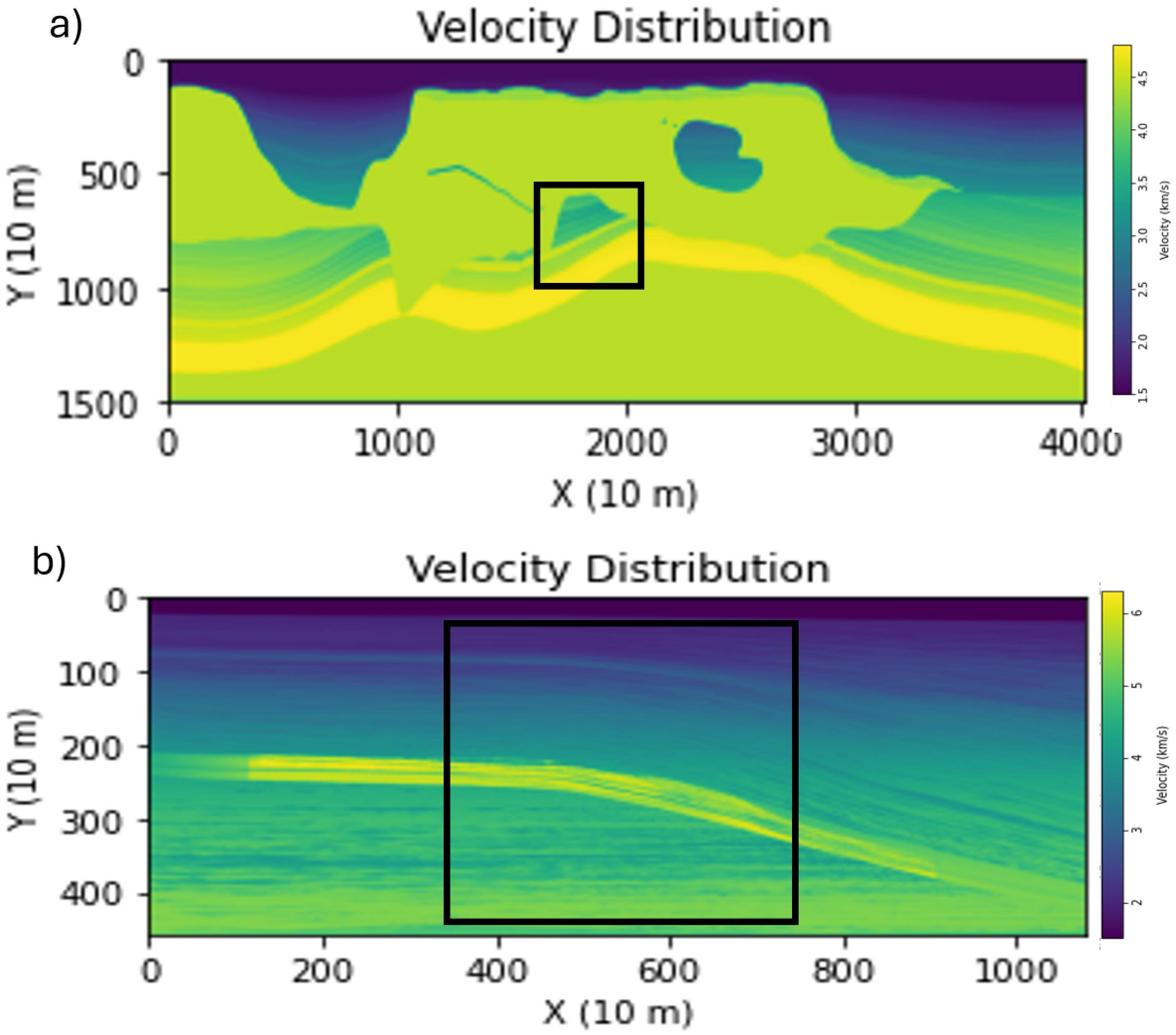


Figure 7.1: Examples of 4x4 km models extracted from the synthetic velocity models. The black square represents a potential model of this area randomly extracted. Figure a represents a velocity model with complex salt formations, while Figure b illustrates a velocity model representing a carbonate platform. The size difference between these two velocity models is evident, with the velocity model represented in a being significantly larger.

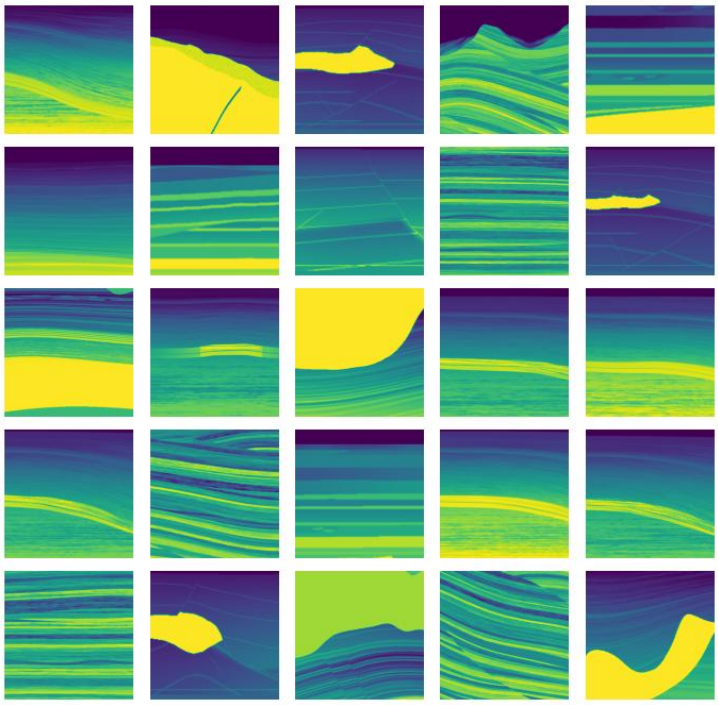

Figure 7.2: Displayed in this image is a set of 25 two-dimensional 4x4 km models, selected from the 12,000 subsections extracted from the three-dimensional synthetic velocity models introduced in the previous chapter.

## 7.1 Forward Modeling

As mentioned in chapter 2, a Python script was developed utilizing the Devito package to simulate accurate wave propagation through each of the extracted subsections.
The following parameters were employed in the forward modeling:

- Source Type: 10 Hz Ricker wavelet (Fig.7.3).
- Source Position: Top center of the model, at a depth of 0 meters, and 2 km from the lateral boundaries (Fig.7.4).
- Grid Spacing: 10 meters
- Receiver Spacing: 10 meters, positioned uniformly throughout the top of the model (Fig.7.4).
- Boundary Conditions: Absorbing boundary layers to minimize reflections.
- Boundary Layers Thickness: 300 grid points.

- Recording Total Time $t_0$ : 4000 ms.
- Time Step ($\Delta t$): Selected to satisfy the Courant-Friedrichs-Lewy (CFL) condition for stability (see equation 2.19).
- Space Order: 2 (for finite difference discretization).
- Time Order: 2 (for finite difference discretization).

These parameters were selected to balance the computational efficiency with the accuracy of the simulated wavefields. The grid spacing was calculated to satisfy the dispersion condition (see equation 2.19), ensuring that the wave propagation is accurately captured without numerical artifacts. The time step was determined using Devito's `model.critical_dt` function, which computes the largest ($\Delta t$) that satisfies the Courant-Friedrichs-Lewy (CFL) stability condition (see equation 2.20). Notably, ($\Delta t$) varied from model to model, as it is dependent on the maximum velocity present in each velocity model. In cases of real data, a hypothetical maximum velocity would need to be assumed to ensure a correct time step, safeguarding against instability in the simulation. The space order and time order were both set to 2 to reduce computational costs while maintaining sufficient accuracy. Absorbing boundary layers, 300 grid points thick, were applied on all sides to simulate an infinite domain and minimize reflections at the boundaries. A total recording time of 4000 ms was selected to ensure that all relevant wave phenomena were captured, allowing for accurate reconstruction of the most energetic traveltimes from the wavefield solution.

The following Python code illustrates the main setup used to perform the wave propagation simulation:

- ```
  u = TimeFunction(name="u", grid=model.grid,
                   timeorder=2, spaceorder=2,
                   save=timerange.num)
  ```

  This line of code defines a `TimeFunction` named `u` on the computational grid `model.grid`, that takes as input the velocity profile, grid shape, spacing, and boundary conditions. This function represents the wavefield over both time and space. The parameters `time_order=2` and `space_order=2` specify the accuracy of the finite difference scheme used for time and spatial derivatives. The `save=time_range.num` argument ensures that the wavefield is stored at each time step throughout the simulation.
  ```

- `pde = model.m * u.dt2 - u.laplace + model.damp * u.dt`

  This code formulates the 2D acoustic wave equation as a partial differential equation (PDE). The term `model.m * u.dt2` represents the second time derivative of the wavefield, scaled point-wise by the squared inverse of the velocity model (`model.m`). The `u.laplace` term calculates the Laplacian of the wavefield, representing the spatial second derivatives. The `model.damp * u.dt` term includes a damping factor to absorb outgoing waves and minimize reflections from the boundaries.

- `stencil = Eq(u.forward, solve(pde, u.forward))`

  This line sets up the finite difference stencil, which calculates the wavefield at the next time step (`u.forward`) by solving the PDE. The `solve` function automatically rearranges the PDE to isolate the future wavefield.

- ```
  srcterm = src.inject(field=u.forward,
                       expr=src * dt**2 / model.m)
  ```

  This line of code injects the source wavelet (`src`) into the subsurface at the source location. The expression `src * dt**2 / model.m` scales the source term according to the time step and velocity model to ensure proper amplitude and energy distribution in the simulation.

- `recterm = rec.interpolate(u)`

  Here, the code sets up the receiver interpolation, which records the wavefield at the receiver locations (`rec`) as the simulation progresses (Fig.7.5).

- ```
  op = Operator([stencil] + srcterm + recterm,
                subs=model.spacingmap)
  op.apply(time=timerange.num-2, dt=dt)
  ```

  These lines create and execute the finite difference `Operator`, which combines the wave equation stencil, source injection, and receiver interpolation (Fig.7.6). The operator is applied over the entire simulation time (`time=time_range.num-2`) using the specified time step (`dt=dt`).

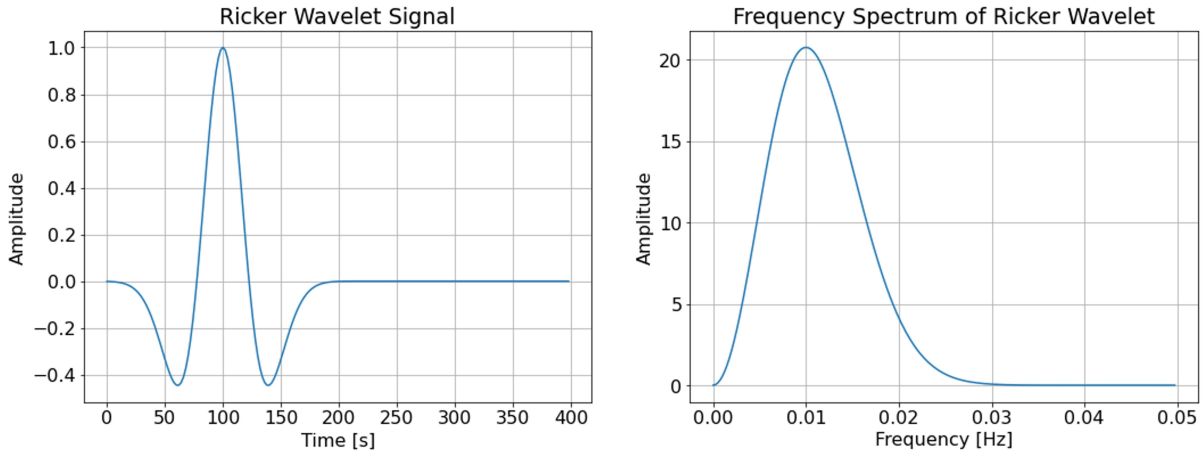


Figure 7.3: The figure shows two side-by-side plots. On the left is the time-domain waveform of a 10 Hz Ricker wavelet, with time on the x-axis and amplitude on the y-axis. On the right is the frequency spectrum of the same wavelet, with frequency (Hz) on the x-axis and amplitude on the y-axis, highlighting a peak at 10 Hz.

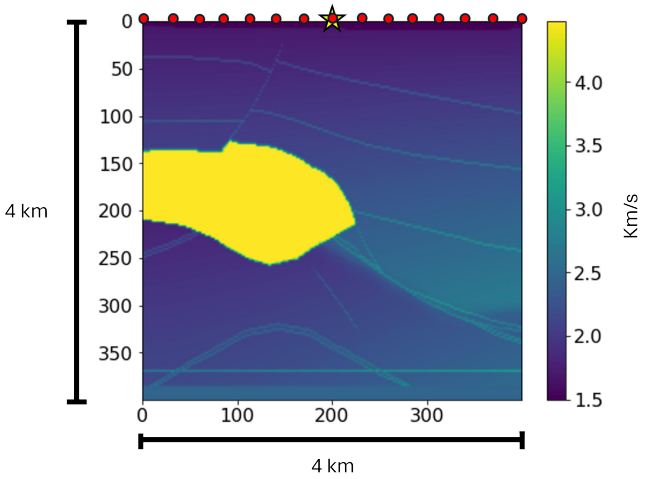


Figure 7.4: This image shows an example of a velocity model. The yellow star indicates the position of the source, while the red dots represent the locations of the receivers. The red dots are spaced 10 meters apart in the actual setup, though in this illustration, the spacing appears more relaxed for practical reasons.

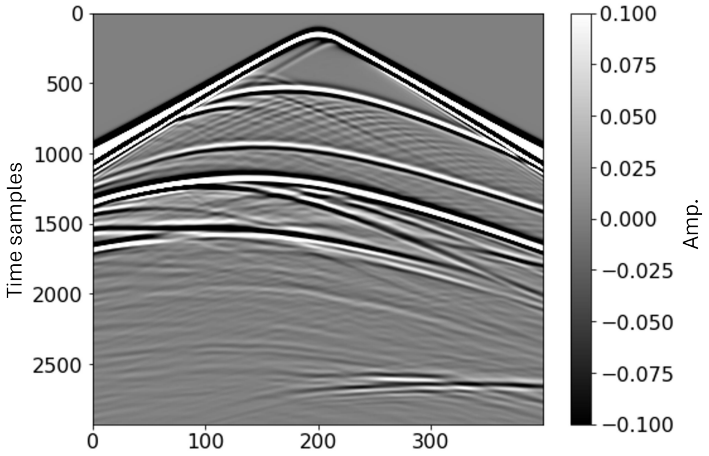


Figure 7.5: This image displays the shot gather traces recorded by the receivers after solving the wave equation over the same example velocity model (Fig.7.4). Each trace represents the recorded wavefield at a particular receiver location over time.

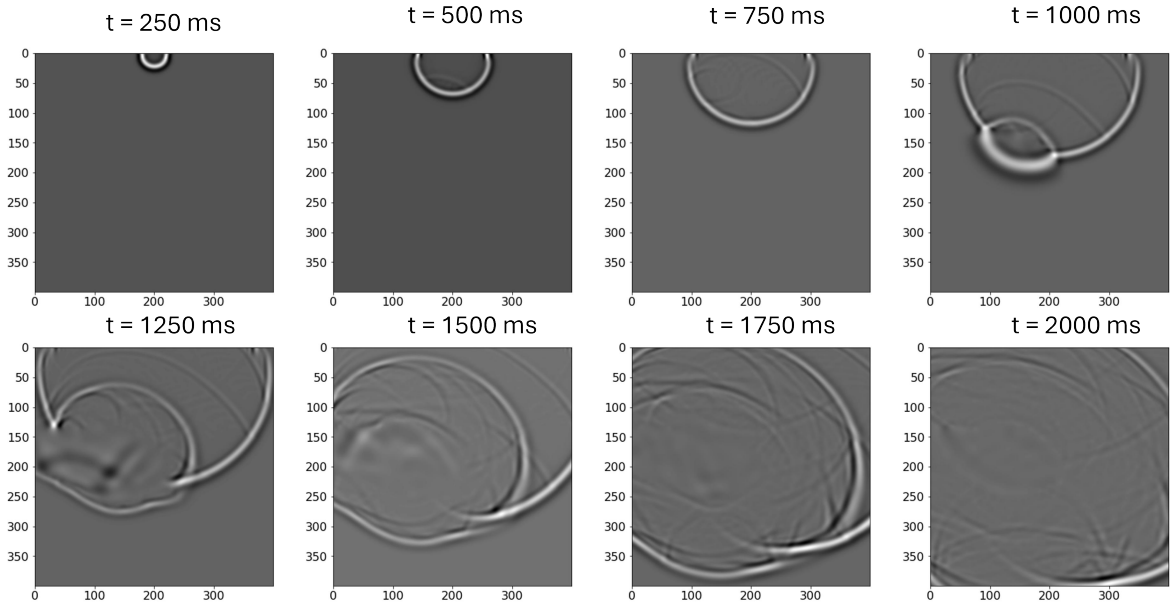


Figure 7.6: The image depicts several time snapshots of wave propagation within the velocity model in (Fig.7.4), with time intervals ranging from 250 ms to 2000 ms. At 750 ms, the reflected wave from the salt body located at the center of the model is prominently visible. This reflection is indicative of the wave interacting with the contrast in the medium caused by the presence of the salt.

## 7.2 Max amplitude traveltimes

In order to compute the max amplitude traveltimes, the finite difference wave equation was solved using a specific time step, $\Delta t$ (see chapter 7.1), over a total simulation duration $t_0$. The solution includes a series of snapshots capturing the wave's propagation, where amplitudes are recorded at intervals of $\Delta t$ until the total time $t_0$ is reached. The total number of snapshots, denoted as $N$, is equal to $t_0/\Delta t$. The values of $\Delta t$ could oscillate between 1 ms and 2 ms, resulting in corresponding numbers of snapshots ranging from approximately 4000 to approximately 2000. Each snapshot results in a matrix of size $400 \times 400$, with each element spaced 10 meters apart, representing grid points in the model.

For each grid point, the time series of amplitude values was extracted, and the square of these amplitudes was calculated to determine the energy at each point. From this series, the maximum value was identified, and the corresponding index $n$ was recorded. By multiplying this index by $\Delta t$, we obtained the time at which the most energetic wave arrival occurred at that specific point in the velocity model. This process was repeated for every grid point in the matrix, resulting in a traveltime map showing the time of the most energetic wave arrival (Fig. 7.7). Fig. 7.8 displays on the left an example of different snapshots obtained by solving the wave equation with a Finite Difference method at different times. In each snapshot, the same point in the domain is considered, and by collecting the amplitudes at that point across all times, a temporal signal is obtained, which can be seen at the bottom of the image. This signal is then squared, and the index $n$ corresponding to the maximum value represents the most energetic arrival. Multiplying this index $n$ by $\Delta t$ provides the arrival time of the most energetic wave at that specific point. This procedure, applied to every point in the matrix, results in the traveltime map of the most energetic arrival, which can be seen on the right of the image.

This method for calculating the traveltime map of the most energetic arrivals was then applied across all 12,000 velocity models, producing a total of 12,000 traveltime maps (Fig. 7.9). The entire process took approximately 17 hours of machine time, around 5 seconds for each model. For details on the specifications of the machine used for the calculations, see Appendix A. The use of thick absorbing boundaries was critical for accurately calculating the travel times of the most energetic arrivals. Fig. 7.10 shows a comparison of maximum arrival traveltime maps generated using different boundary layer thicknesses. On the left, the boundaries are only 10 grid points thick, which leads to reflections within the model and visible artifacts, such as

spikes at 3500 ms. On the right, the boundaries are 300 grid points thick, and the map is free of such artifacts, indicating more accurate traveltime results. This demonstrates the effect of boundary layer thickness on the quality of the traveltime maps.

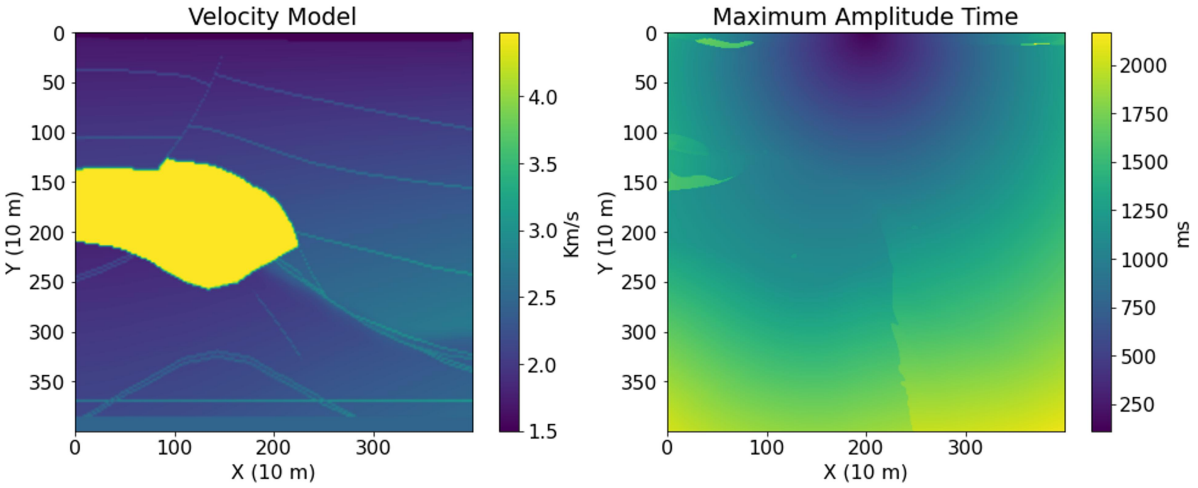


Figure 7.7: On the left an example of a velocity model, on the right its correspective maximum arrival traveltime map

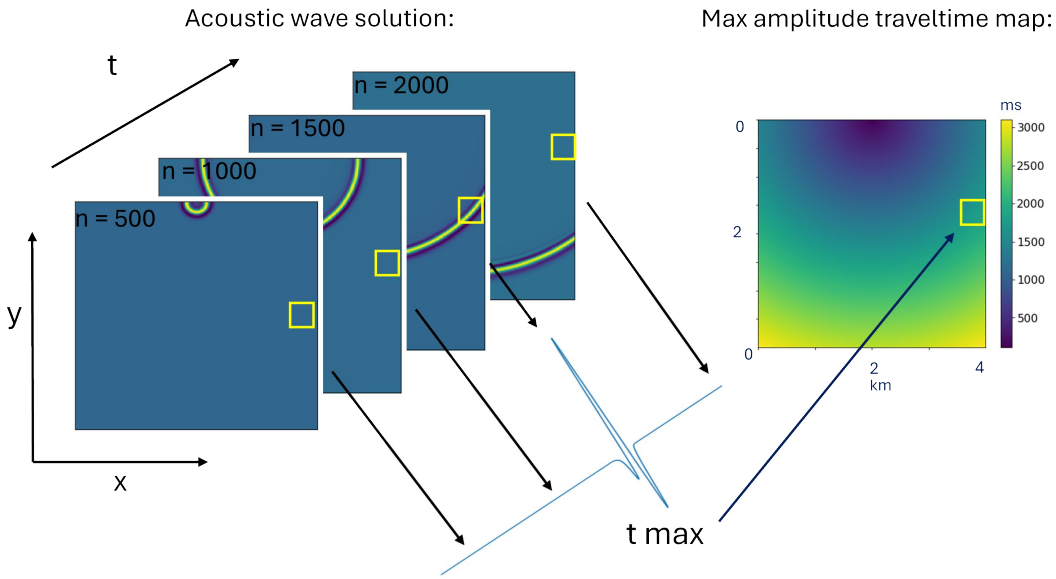


Figure 7.8: Example of how to obtain the maximum energy traveltime map by solving the wave equation using the finite difference method

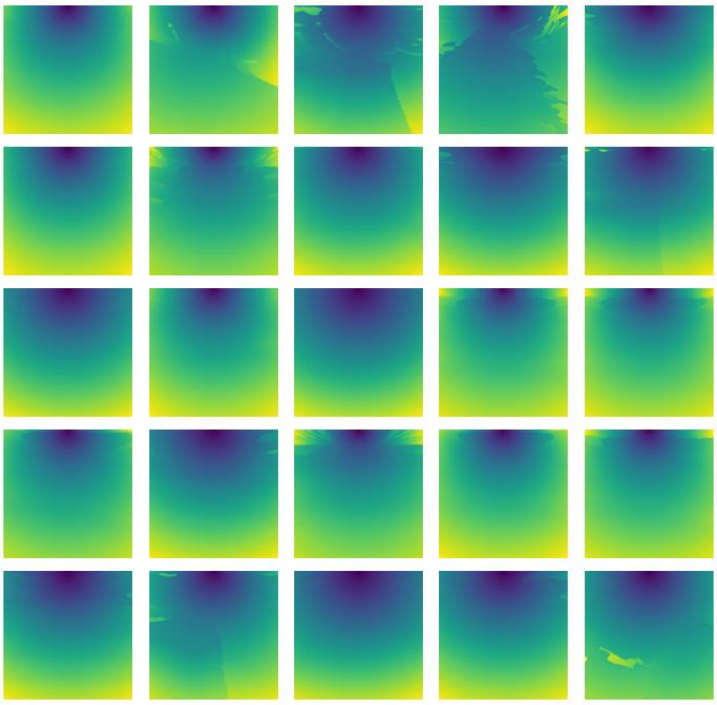

Figure 7.9: Displayed in this image is a sample of 25 maximum arrival traveltime maps calculated from (Fig.,7.2), selected from the 12,000 forming the labels of the dataset

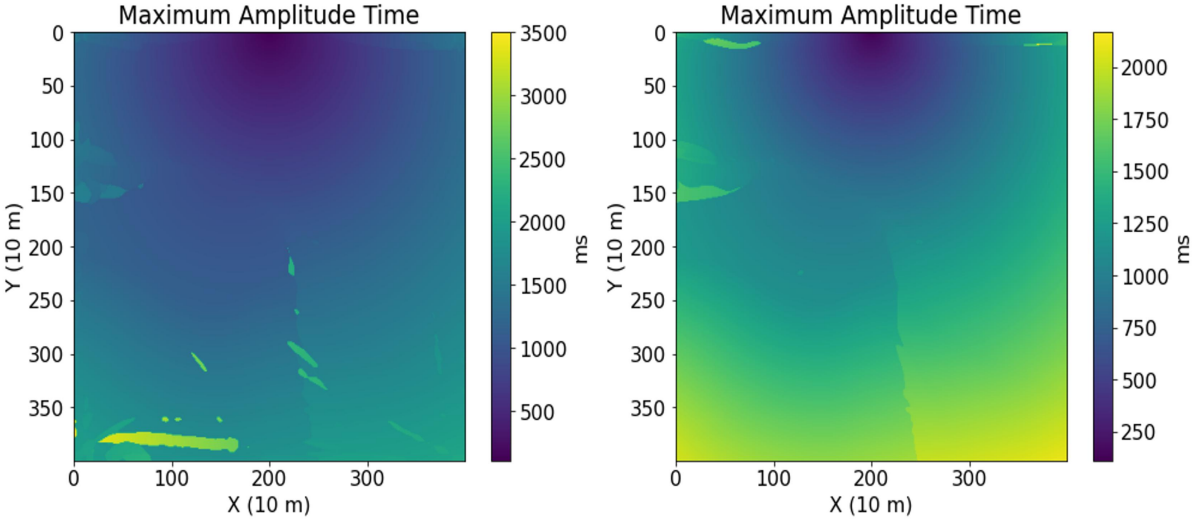


Figure 7.10: Comparison of maximum arrival traveltime maps using different thicknesses for boundary layers. On the left, boundaries are only 10 points thick, while on the right, boundaries are 300 points thick.

## 7.3 Sensibility velocity analysis

In geologically complex environments, it is common to observe the formation of regions where the most energetic arrivals are significantly delayed compared to the surrounding areas. These delayed arrival zones are a result of various wave propagation phenomena, which are influenced by the complexity of the velocity model. This phenomena are commonly referred to as triplications (Aki  Richards, 1980) [16]. In this study, the phenomenon of triplications was investigated further by analyzing the arrival times of the most energetic waves within a velocity model. For each point in the model, the arrival time of the wave with the highest energy was calculated.

To explore the behavior of these delayed arrival zones, progressive smoothing was applied to the velocity model using Gaussian filters from the `scipy.ndimage` library, with sigma values set to 3, 5, and 7. In Fig. 7.11, the velocity models are displayed in the left column, progressively smoothed from top to bottom. In the right column, the corresponding travel time maps of the most energetic wave arrivals are shown. It is evident that, as the velocity models become increasingly smoothed, the regions of delayed energy arrivals characterized by higher arrival times compared to adjacent areas gradually diminish and eventually disappear.

This demonstrates the strong correlation between the complexity of the velocity model and the presence of zones of triplication: as smoothing reduces the complexity of the model, these zones also diminish.

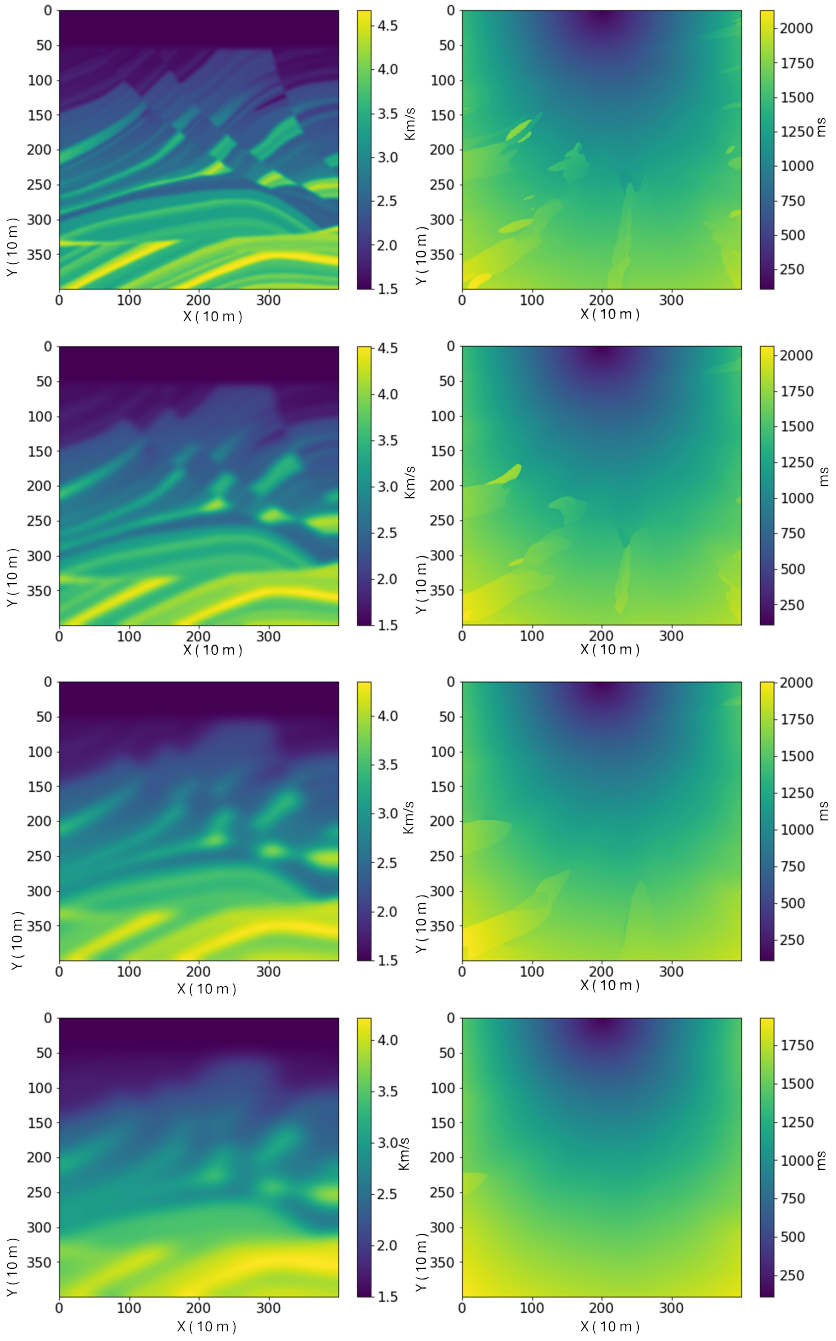


Figure 7.11: The left column shows the progressively smoothed velocity models, while the right column displays the corresponding travel time maps of the most energetic wave arrivals. As the smoothing increases, the regions of delayed energy arrivals (triplications) gradually diminish and disappear.

## 7.4 Comprehensive Description of Experimental Methods

The initial set of 12,000 samples, representing 12,000 different velocity models, was split randomly into two datasets for training the U-Net network: the training dataset and the validation dataset. Since the dataset was quite large, 90% of the models (10,800 models) were allocated to the training dataset, while the remaining 10% (1,200 models) were used for the validation dataset. The training dataset was used to optimize the model's parameters, while the validation dataset was employed to monitor the model's performance during training and to adjust hyperparameters. For all tests conducted, the loss function utilized was always the L1 loss, which is particularly advantageous for regression tasks due to its robustness (Hodge, 2004) [40].

In this thesis, a predefined number of epochs was not set for training. Instead, an early stopping mechanism based on the validation loss was utilized; if the validation loss did not improve within 100 epochs, the training process was halted. This early stopping technique was employed to prevent the model from overfitting on the training dataset, which could otherwise lead to progressively worse results on the validation dataset and, consequently, in general performance. For testing purposes, and as the primary focus of this thesis, velocity models derived from the Marmousi model were used in the testing set (see Chapter 6).

**experiment 1: number of inputs**

The first experiment aimed to assess whether U-Net would perform better with a different number of inputs. In both cases, the label used for training was the same: the traveltime maps of the most energetic arrivals. One network was trained using a single input, while a second network was trained using two inputs. Namely, both NN were trained using the velocity models as input, while only the second also relied on the first arrival travel time maps as additional input (Fig. 7.12). The script for generating the fisrt arrival traveltime map from the velocity model, based on the solution of the eikonal equation, was provided by Eni.

The decision to include the first arrival traveltime map as an additional input was motivated by two key reasons: first, its quick and efficient computation, requiring only about 0.1 seconds per velocity model, which adds minimal computational overhead, in contrast, solving the wave equation and extracting traveltime maps for the most energetic arrivals typically takes about 5 seconds per model; and second, the overall similarity between first arrival and most energetic traveltimes. This similarity suggests that incorporating the traveltime map could

provide a head start in the training process.

The tests were conducted using two distinct configurations. In the first configuration, both the training and validation sets consisted solely of velocity models, with a training set of 10,000 samples and a validation set of 1,000 samples drawn randomly respectively from the initial 10,800 and 1200 datasets. In the second configuration, the training set also consisted of 10,000 samples and the validation set of 1,000 samples, but this time, the input matrices had dimensions 400x400x2. Here, the first channel contained the velocity model, while the second channel contained the traveltime map of first arrivals.

These dual-input matrices were provided to the network using a technique analogous to how RGB images are typically processed, with the difference being that instead of three channels, as in RGB images, the network was fed with two channels. The experimental results demonstrated that the network trained with the dual-input dataset exhibited significantly better performance. This outcome aligns with the intuition that providing additional, relevant information to the network should enhance its performance. Fig. 7.13 depicts the validation losses of the U-Net network trained using two different input configurations. The blue line corresponds to the network trained with only the velocity model as input, while the orange line represents the network trained with two inputs: the velocity model and the first-arrival traveltime. It is evident that the loss of the network trained with two inputs is significantly lower than that of the network trained with only the velocity model. Fig. 7.14 shows on the left, the performance of the U-Net network on the central part of Marmousi model is evaluated using Root Mean Square Error (RMSE) as the metric, while on the right, Mean Absolute Error (MAE) is used. The red line is the U-Net with 2 inputs while the blue one the network using only one input. It is evident that the U-Net network trained with two inputs instead of one shows a significant improvement in performance.

Given the superior performance observed with two inputs, subsequent efforts focused on fine-tuning the network by optimizing several parameters. The parameters adjusted in this study included the learning rate, the size of the dataset, and the progressive number of filters within the U-Net architecture.

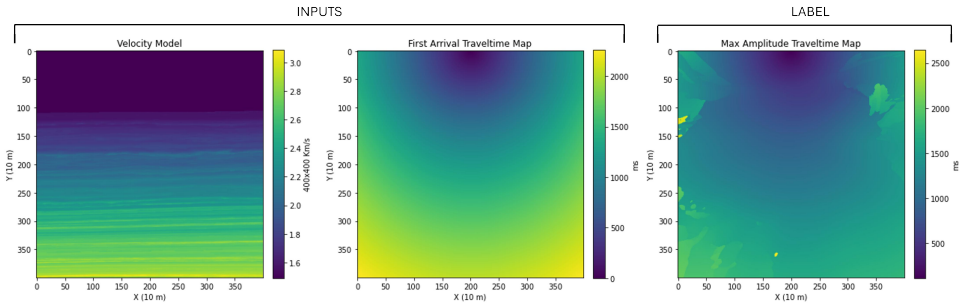


Figure 7.12: The first two matrices on the left represent the inputs: the velocity model and the first-arrival traveltime map, respectively. The image on the right represents the label, which is the traveltime map of the most energetic arrivals.

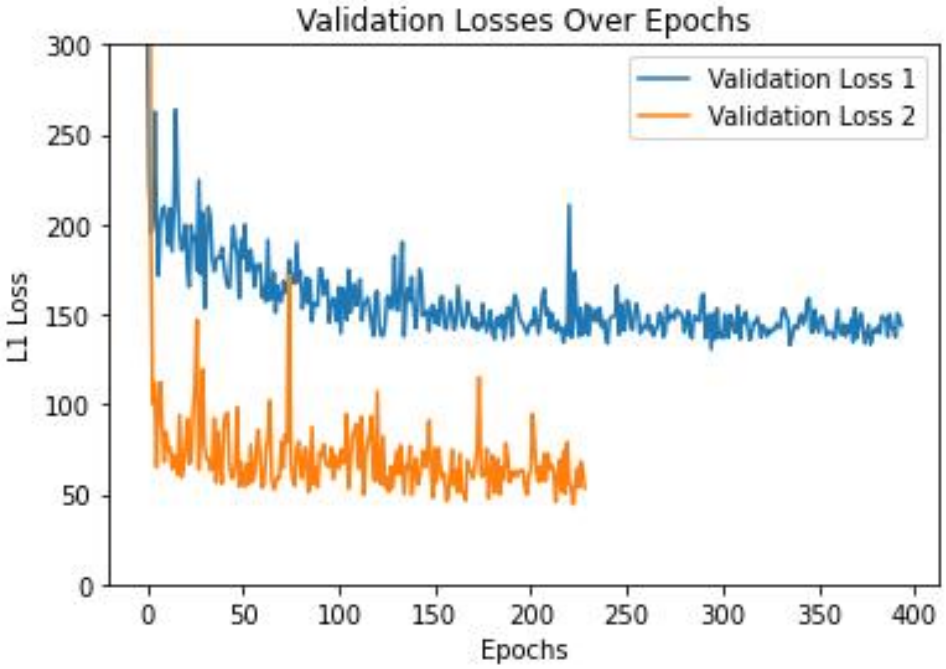


Figure 7.13: The image shows the validation losses of the U-Net network for two input configurations: the blue line represents the network trained with only the velocity model, while the orange line indicates the network trained with both the velocity model and the first-arrival traveltime.

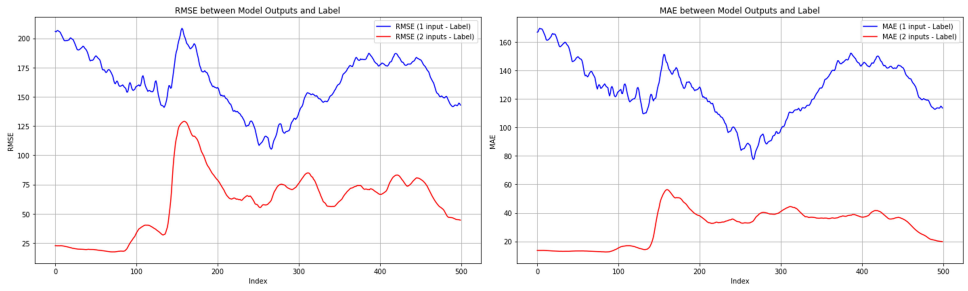


Figure 7.14: Performance evaluation of the U-Net network on the central part of the Marmousi model. The left side uses RMSE (Root Mean Square Error) as the metric, while the right side uses MAE (Mean Absolute Error). The red line represents the U-Net with two inputs, and the blue line represents the network with a single input.

### experiment 2: learning rate

In this experiment, different learning rates were tested: 0.1, 0.01, 0.001, and 0.0001. The learning rate of 0.1 proved to be the most unstable, exhibiting numerous spikes. Similar results could be achieved with either 0.01, 0.001 or 0.0001. The value 0.01 was chosen at the end since it reached similar results to the others in fewer epochs. The graph showing the various loss curves for different learning rates can be seen in (Fig. 7.15)

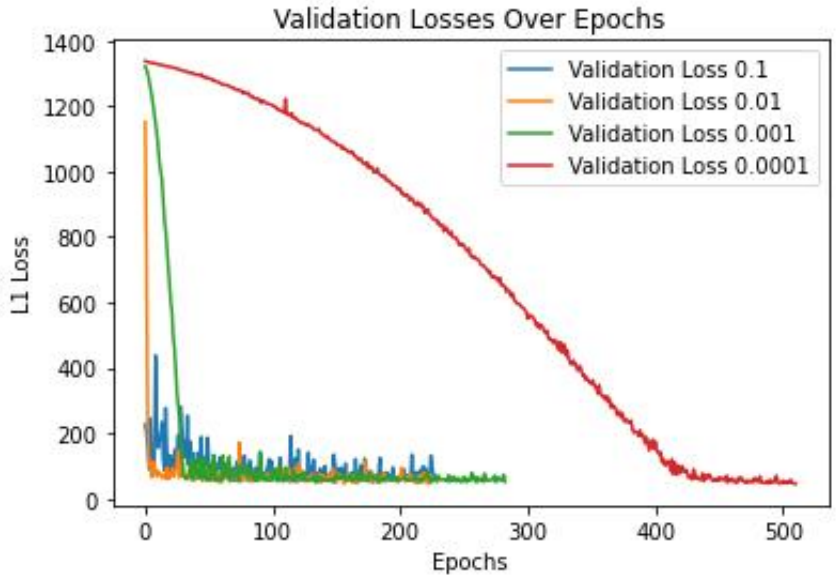


Figure 7.15: The image illustrates various validation losses calculated using different learning rates.

**experiment 3: optimizers**

For this analysis, different optimizers were compared: RMSprop, AdamW, and Adam. RMSprop was ultimately chosen due to its lower validation loss values, although AdamW exhibited a more stable learning curve. The graph showing the various loss curves for different optimizers can be seen in (Fig. 7.16).

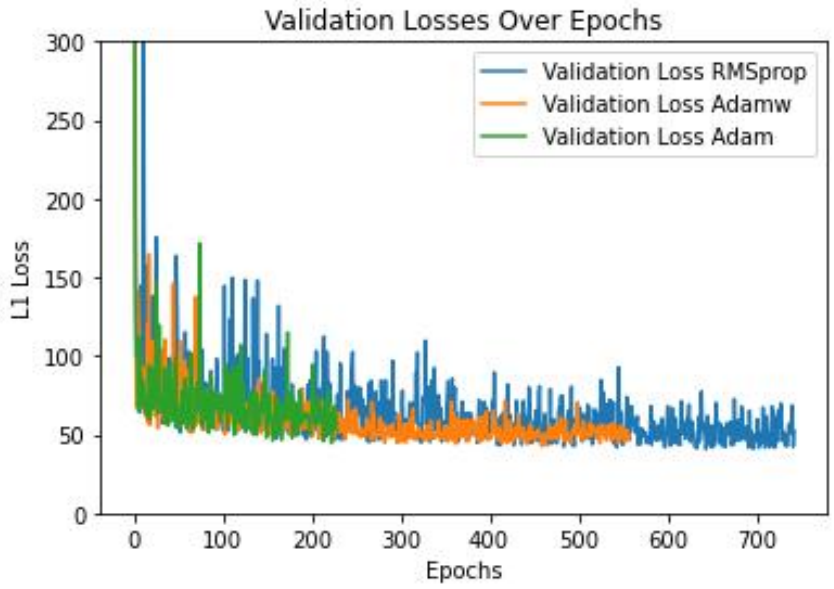


Figure 7.16: The graph shows the validation losses of the U-Net network calculated using three different optimizers: RMSprop, AdamW, and Adam.

**experiment 4: number of filters**

Different filter configurations within the U-Net were evaluated to determine their impact on performance. The first configuration used filters of [64, 128, 256, 512, 1024], while the second utilized filters of [256, 512, 1024, 2048, 4096]. Although the second U-Net required greater computational effort due to its increased number of parameters, it did not lead to a significant reduction in loss. Therefore, the lighter version with filters [64, 128, 256, 512, 1024] was preferred for its efficiency and comparable results. The graph illustrating the various validation losses for these configurations can be seen in (Fig. 7.17)

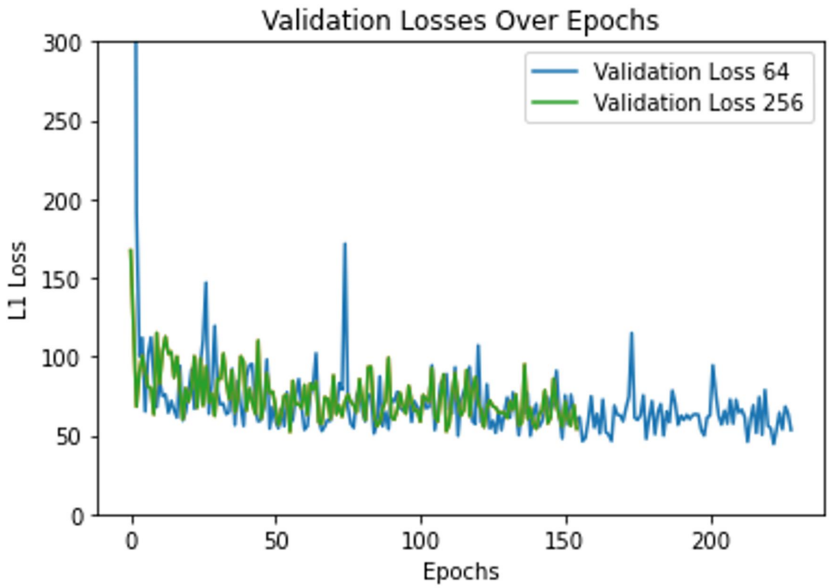


Figure 7.17: The image shows various validation losses calculated using different filter configurations within the U-Net.

**experiment 5: the size of the dataset**

The effect of training dataset size on model performance was evaluated by comparing two training dataset sizes: 2500 and 10000. Fig. 7.18 indicates that increasing the number of samples in the training set leads to a decrease in validation loss. Although the larger dataset requires a linear increase in training time, the decision was made to utilize 10000 samples due to the substantially lower loss achieved, which is significantly better than that obtained with the smaller dataset.

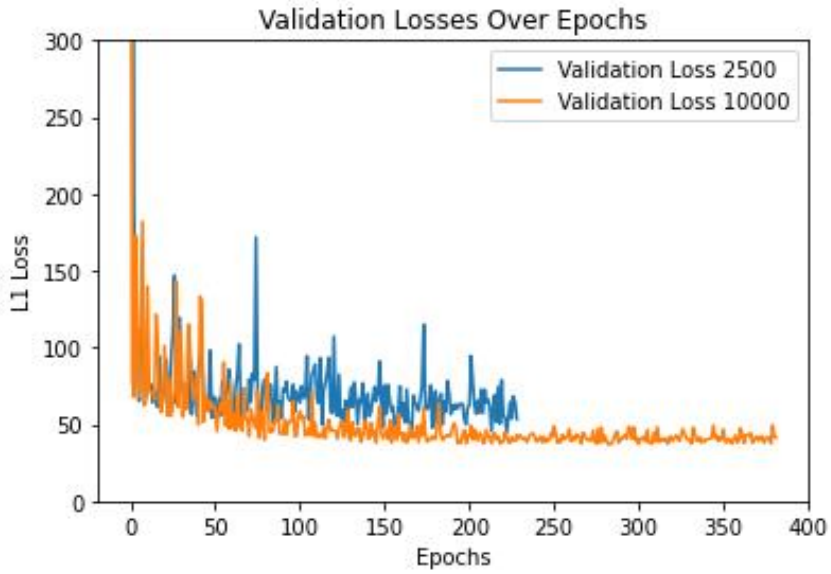


Figure 7.18: The image shows the validation losses of the Unet trained with a dataset composed of 2500 models and 10000 models

**experiment 6: NST augmentation**

The effects of NST augmentation were evaluated based on the hypothesis that creating a more complex training dataset would better represent the complexities of the Marmousi model. To address this, velocity models from the original dataset were used as content images, while sections of the Marmousi dataset served as style images (see chapters 4 and 6).

The computational time for creating a more complex dataset using NST was approximately 25 seconds per image, leading to a total processing time of nearly 42 hours for the 6,000 images. For details on the specifications of the machine used for these calculations, please refer to Appendix A.

The results indicated that the complexity of the travel time patterns in the NST-enhanced dataset was more aligned with those expected in the test dataset, particularly in the central region of the Marmousi model.

In Fig.7.19 the enhancement achieved through NST is illustrated. The top left shows a simple velocity model, along with its corresponding more energetic traveltime map, which lacks the complexity necessary to capture triplications. In contrast, the bottom left presents the NST-enhanced model, which reveals more pronounced areas of triplication in the traveltime map displayed on the right. This improved alignment with the test dataset led to a significant improvement in the model's performance, as shown in Fig.7.20 and Fig.7.21. The compari-

son in Fig.7.20 clearly demonstrates that the network trained with the NST-enhanced dataset achieved an average reduction in MAE by 10 ms in the central region of the Marmousi model, represented by the blue line, compared to the network trained without NST (red line).
Fig.7.21 presents further performance comparisons on segments of the Marmousi dataset between the U-Net trained with the base dataset and the U-Net trained with the dataset enhanced through NST. The velocity models are displayed on the left. Each example, labeled as a and b, represents a different velocity model. In the first row of each example, the images are arranged from left to right as follows: the more energetic traveltime map produced by the U-Net trained with the NST-enhanced dataset, the corresponding ground truth traveltime map (the label), and the absolute difference between the two. The second row of each example showcases the U-Net trained on the non-NST dataset, with the leftmost image displaying the traveltime calculated by this U-Net, the middle image presenting the label and the rightmost image depicts the absolute difference between the output from the non-NST U-Net. It is evident that the U-Net trained with the less complex dataset fails to predict triplications, whereas the U-Net trained with the NST-enhanced dataset successfully captures these features, albeit with lower resolution compared to the label.
These findings suggest that applying NST selectively to augment the training dataset can effectively increase its complexity, thereby improving the model's ability to handle complex scenarios.

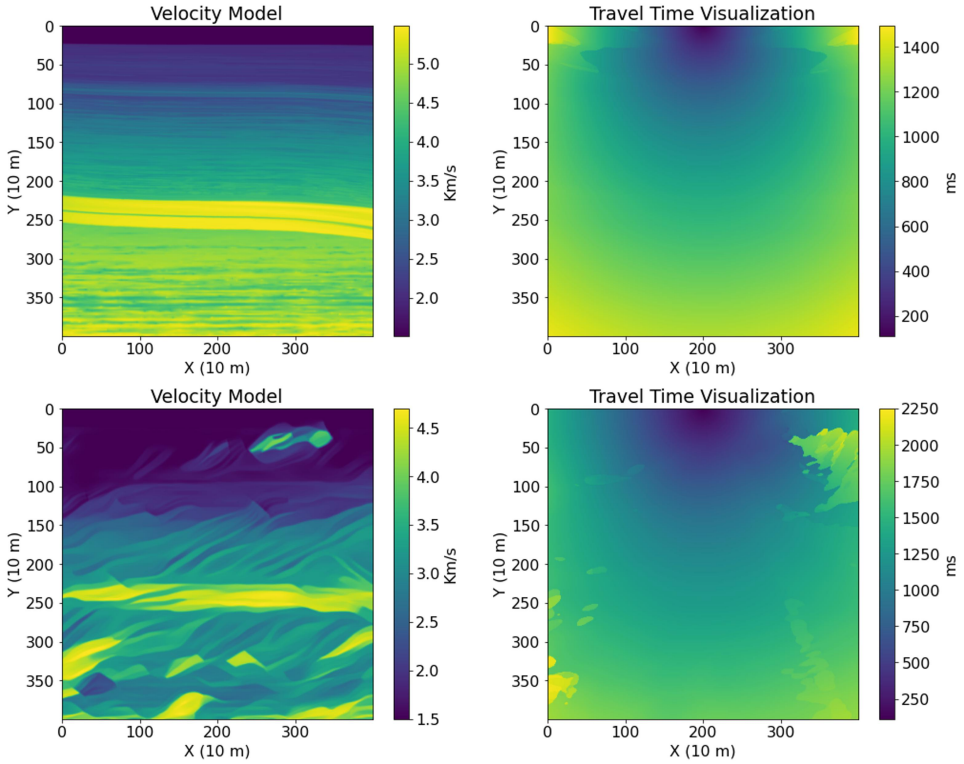


Figure 7.19: Example of velocity model enhancement. At the top left is a simple velocity model, with its corresponding more energetic traveltime map to its right. At the bottom left is the enhanced model obtained through NST, with its corresponding more energetic traveltime map to its right

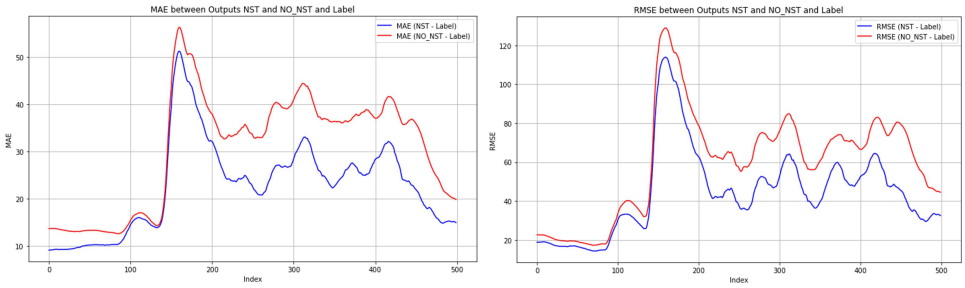


Figure 7.20: Comparison of MAE and RMSE errors in the central region of the Marmousi between the network trained with the NST-enhanced dataset and the one trained without NST. with the RMSE plot on the left and the MAE plot on the right.

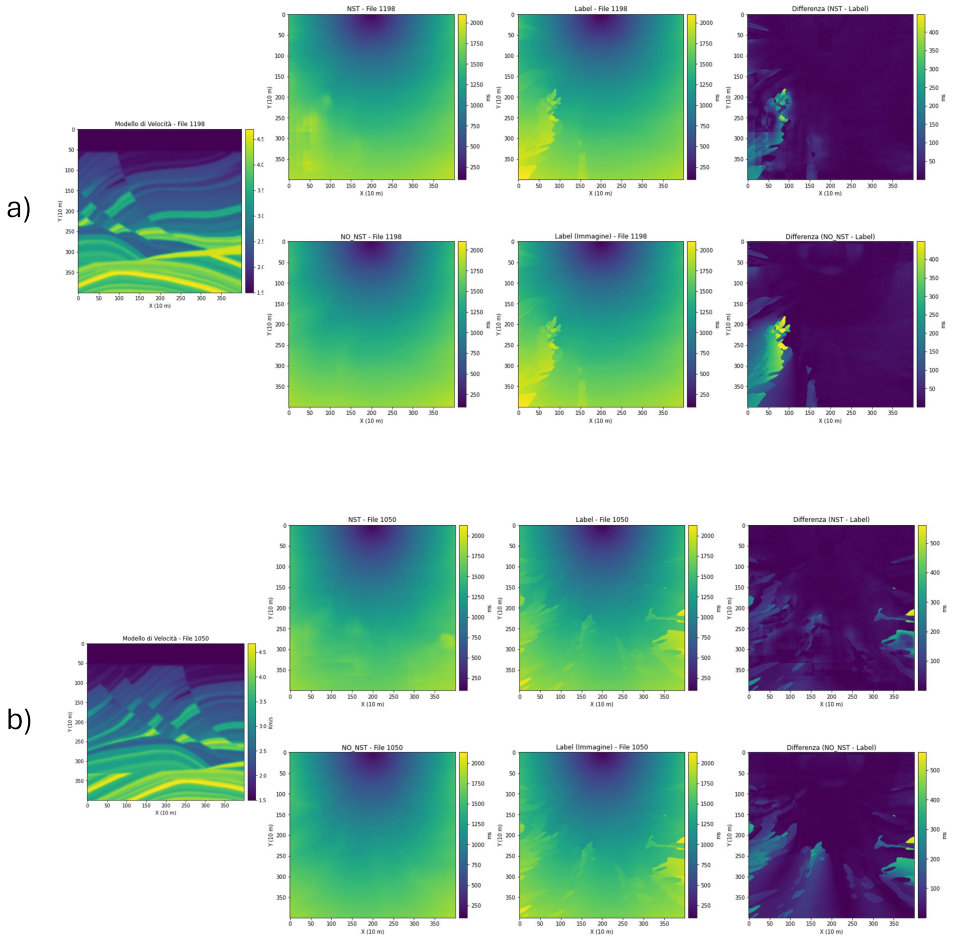


Figure 7.21: Comparison of performance on Marmousi segments between the U-Net trained with the base dataset and the U-Net trained with the dataset enhanced through NST. The velocity models are displayed on the left. Each example, labeled as a and b, represents a different velocity model. In the first row of each example, the images are arranged from left to right as follows: the more energetic traveltime map produced by the U-Net trained with the NST-enhanced dataset, the corresponding ground truth traveltime map (the label), and the absolute difference between the two. The second row of each example showcases the U-Net trained on the non-NST dataset, with the leftmost image displaying the traveltime calculated by this U-Net, the middle image presenting the label and the rightmost image depicts the absolute difference between the output from the non-NST U-Net.

**experiment 7: AUnet, ResAUnet and Ensemble**

The performance of the U-Net architecture can be considered highly satisfactory in regions where geological complexity is not predominant. However, in areas with high complexity, the U-Net tends to predict the background traveltime data quite accurately but struggles to

achieve precise predictions in regions affected by triplication phenomena, resulting in low-resolution images.

Fig. 7.22 illustrates this behaviour by displaying five different velocity models in the first column. The second column shows the first arrival traveltimes used as the second input to the network. In the third column, the network's output is presented, while the fourth column provides the ground truth labels. Finally, the fifth column highlights the absolute difference between the output and the label. As can be readily observed, the network performs well in less complex environments, but it exhibits a lack of resolution in areas of triplication within more complex settings.

An attempt was to enhance performance in complex areas by increasing the complexity of the CNN architecture. This was achieved by incorporating Attention Gates to better focus on the edges of the triplication zones. These modifications led to the development of the AUnet architecture. Additionally, residual blocks were added with the expectation that a more intricate network would improve results. These modifications led to the development of the ResAUnet architecture.

The U-Net architecture used had 31 million parameters, while the Attention U-Net (AUnet) had 31 million and 400 thousand parameters, indicating that the addition of Attention Gates introduces a relatively minor increase in complexity. On the other hand, the ResAUnet, which integrates both residual blocks and Attention Gates, had a total of 32 million and 800 thousand parameters. This significant increase in the number of parameters reflects the added complexity of the network.

The hyperparameters used were those obtained from tuning the original U-Net architecture, based on the experiments introduced before: a learning rate of 0.01, a dataset of 10,000 images, the RMSprop optimizer, and filter sizes [64, 128, 256, 512, 1024]. With these parameters and a training batch size of 20, the U-Net network required approximately 1.02 minutes to complete an epoch. This training time was practically identical to that of the AUNet 1.09 minutes and slightly less than that of the ResAUnet 1.20 minutes.

Given the varying characteristics of each architecture, it was hypothesized that combining the outputs of the U-Net, AUnet, and ResAUnet using ensemble techniques could yield improved results. This approach aimed to leverage the specialized capabilities of each model to enhance overall predictions in both simple and complex geological environments.

To implement this strategy, the results from these three neural networks were combined using

ensemble techniques. In this specific case, the weights for the ensemble network were determined through an iterative process, where numerous combinations of weights were tested, and the resulting loss was calculated for each. The combination that minimized the loss on the validation dataset was selected (see chapter 4.3). The final weights were [0.3580, 0.3626, 0.2809], applied respectively to the U-Net, AUNet, and ResAUnet, with the expectation that this approach would enhance the final predictions.

However, during the training process, it became apparent that the results from the three networks were highly comparable in this regression task, with no significant differences observed. Fig. 7.23 present plots of RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) for the 4x4 km models from the central region of the Marmousi dataset. Clearly, the performance of the U-Net, Attention U-Net, and ResAUnet networks with two inputs is very similar. Furthermore, the application of the ensemble technique did not yield any notable improvement in performance. For example, the error plots of the networks on the test dataset clearly show that all models exhibit the maximum error at the same index. In Fig. 7.24 examples of travel times maps computed using the aforementioned network architectures and derived from the same velocity model are presented. Row (a) corresponds to U-Net, row (b) to Attention U-Net, row (c) to ResAUnet, and row (d) to the ensemble output. The first column represents the outputs of the networks, the second shows the labels, and the third illustrates the absolute differences between the network outputs and the labels. As shown, the results tend to be very similar across all models and exhibit a lack of resolution necessary for accurately representing areas with triplication phenomena..

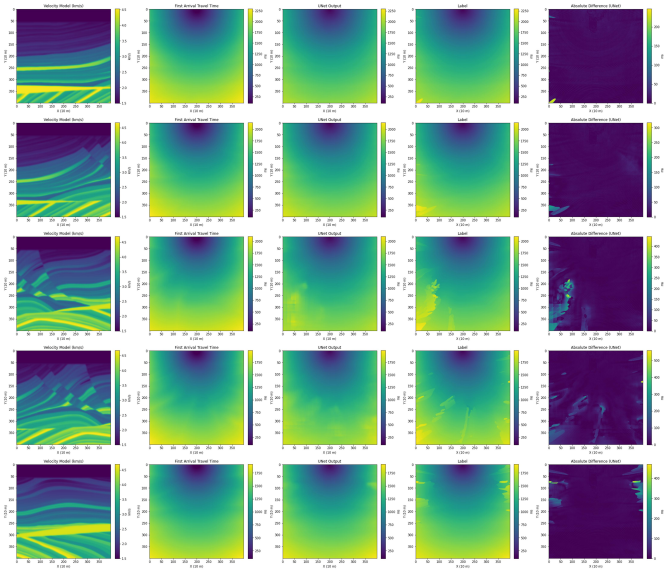

Figure 7.22: Behaviour of U-Net in different geological scenarios. In the first column 5 different velocity models are presented. The second column shows the first arrival traveltimes used as the second input to the network. In the third column, the network's output is presented, while the fourth column provides the ground truth labels. Finally, the fifth column highlights the absolute difference between the output and the label.

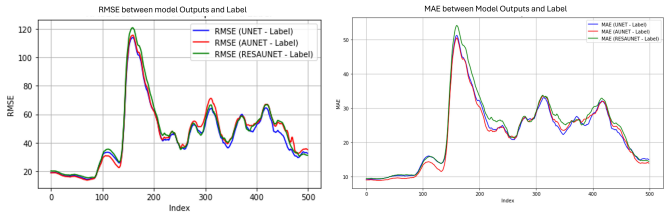


Figure 7.23: Plots of RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error) are presented for 4x4 km models from the central region of the Marmousi dataset, with the RMSE plot on the left and the MAE plot on the right

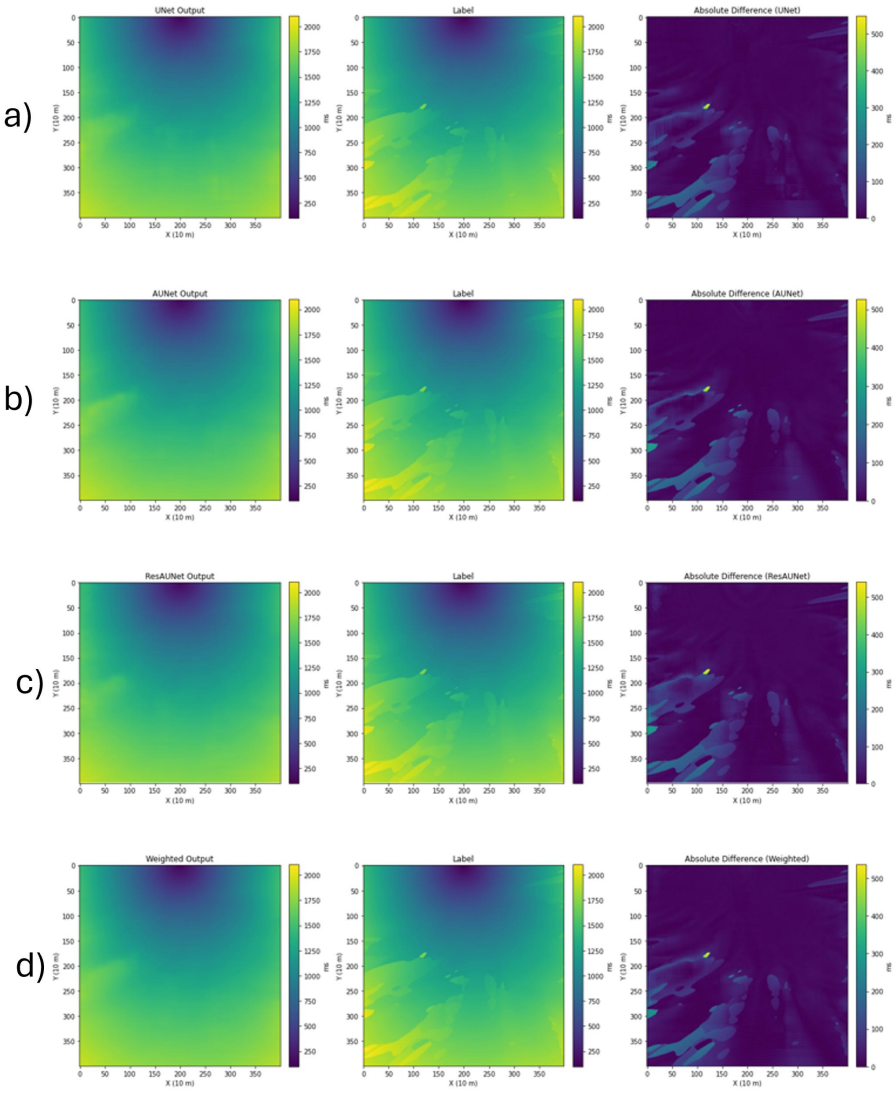


Figure 7.24: Examples of travel time maps computed using different networks. Row (a) corresponds to U-Net, row (b) to Attention U-Net, row (c) to ResAUnet, and row (d) to the ensemble output. The first column represents the outputs of the networks, the second shows the labels, and the third illustrates the absolute differences between the network outputs and the labels. Although different networks were used and ensemble techniques were tested, the outputs tend to be very similar.

**experiment 8: Diffusion model**

The problem with the U-Net model appears to be its difficulty in accurately defining the boundaries of triplication areas. Adjusting the hyperparameters, increasing the complexity of the architecture did not result in sensible improvement, nor did combining various architectures through ensembling methods. To address this issue, generative models, specifically Diffusion Probabilistic Models, were employed. These models were designed to enhance the resolution of the U-Net output. The architecture of the network was adapted, with minor modifications, from Durall, 2020 [38], who utilized diffusion models to apply demultiple to data, converting multiple-infested seismic data into multiple-free data. The forward process involves a series of operations that add noise to the image, while the reverse process entails denoising the image to recover the original clean version.

In this thesis, the reverse process begins with an image composed entirely of noise and the most energetic traveltime map calculated using the U-Net trained on the NST-enhanced dataset. The diffusion model first processes this completely noisy image along with the U-Net output. Through this process, the model gradually reduces the noise, progressively revealing an image that, over several iterations, approaches a resolution very similar to that of the original label image.

This refined image is then fed back into the model, along with the U-Net output, allowing the network to enhance the image further. This procedure is repeated for a fixed number of iterations. As a result, the final output exhibits reduced noise and improved resolution, making it more comparable to the label image than the initial U-Net output.

In all experiments, the diffusion model was trained for 450,000 iterations with a batch size of 5. The weighting parameter $\beta$ (see Chapter 4) was set to increase linearly at each step during the diffusion process, and a depth of 2000 timesteps was used for both the forward process and the reverse denoising process.

The results are presented in Fig. 7.25, which compares the U-Net and Diffusion Model on test data from the central region of the Marmousi dataset. The Diffusion Model clearly outperforms the U-Net, particularly in regions with potential wave triplications, and it significantly reduces the peak error seen around index 160 in the U-Net's output. Fig. 7.26 and Fig. 7.27 provide further detailed comparisons. In Fig. 7.26, the first row shows the U-Net's output (a), the corresponding label (b), and the absolute difference between them (c). The second row displays the Diffusion Model's output (d), the label (e), and their absolute difference (f). Fig. 7.27

follows the same structure, with the first row showing the U-Net's results and the second row showing the Diffusion Model's results, along with the corresponding labels and differences. Overall, the results are highly satisfactory, demonstrating a notable improvement in image quality and sharper edges in regions of triplication using the Diffusion Model. However, a drawback of this method is the high computational time required for generating a single image, approximately 160 seconds, which is four orders of magnitude greater than the prediction generation time of the U-Net.

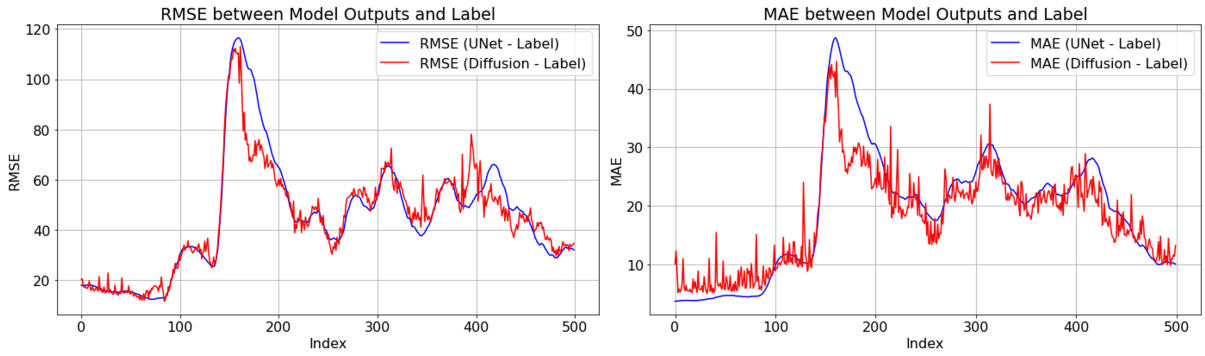


Figure 7.25: Comparison of the results between the U-Net and Diffusion Model on test data from the central region of the Marmousi dataset, , with the RMSE plot on the left and the MAE plot on the right

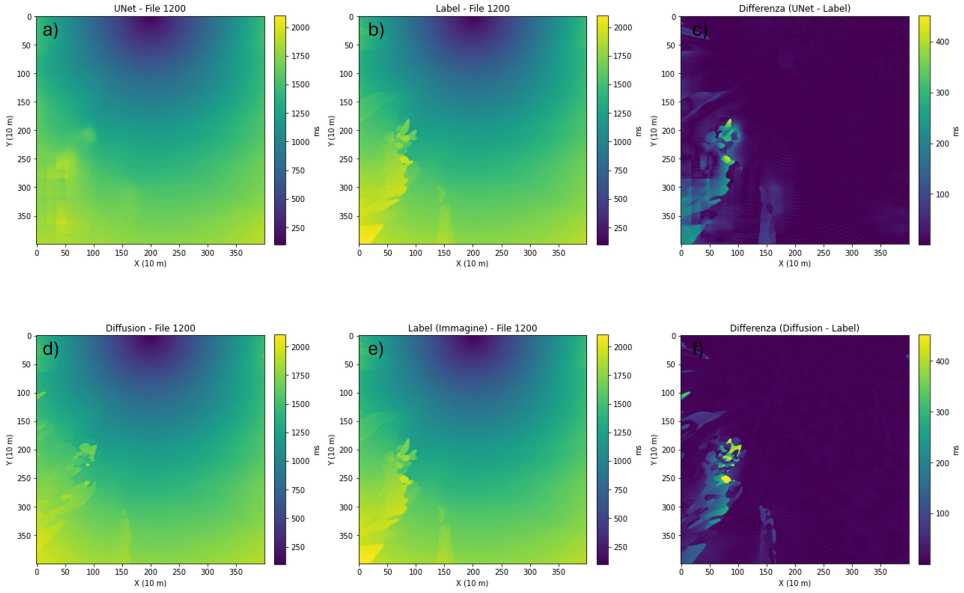


Figure 7.26: Visual comparison of Unet and Diffusion model outputs compared to their corrispective labels. The first row shows the U-Net's output (a), the corresponding label (b), and the absolute difference between them (c). The second row displays the Diffusion Model's output (d), the label (e), and their absolute difference (f).

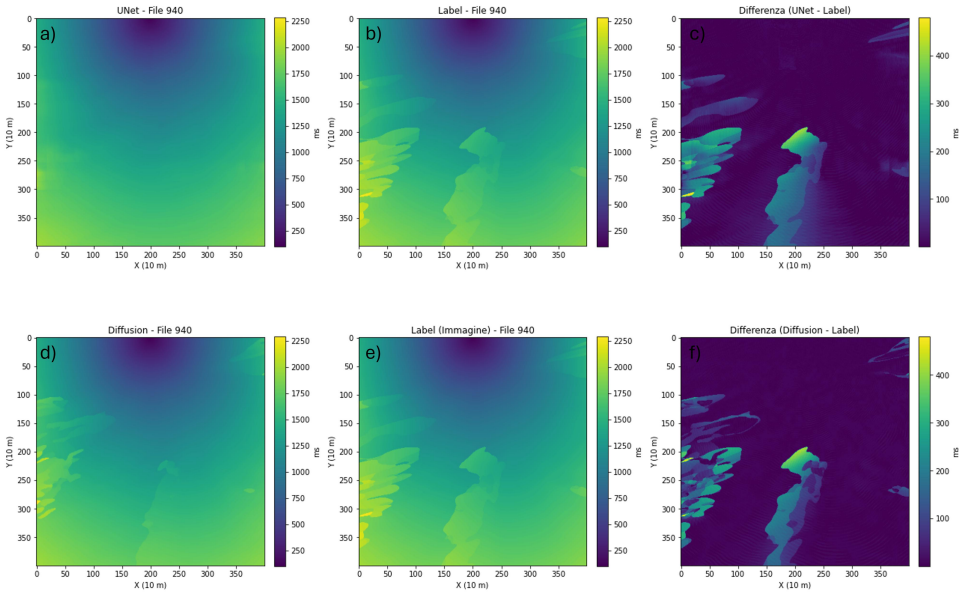


Figure 7.27: Another visual comparison of Unet and Diffusion model outputs compared to their corrispective labels. The first row shows the U-Net's output (a), the corresponding label (b), and the absolute difference between them (c). The second row displays the Diffusion Model's output (d), the label (e), and their absolute difference (f).

## 7.5 Kirchhoff migration

In this study, synthetic data in the form of shot gathers were generated by solving the wave equation using the finite difference method with Devito, applied to each sub-model derived from the Marmousi velocity model (see chapter 6). These shot gathers were then multiplied in the frequency domain by the filter $\sqrt{i2\pi f}$ . Fig.7.30 shows an unfiltered shot gather from the Marmousi model on the left and its corresponding filtered version on the right. In the image below a comparison of the spectra of the filtered and unfiltered traces is presented. Before migrating the data, it was essential to convert the synthetic data from the shot gather domain to the common offset domain. In this thesis, the focus was on migrating traces with an offset of 0 m. This was accomplished by extracting the trace corresponding to the source position from each shot gather. Fig.7.31 illustrates this process: the left side shows the shot gather domain, while the right side displays the common offset domain, using data from the Marmousi model.

By calculating the most energetic traveltime maps from all the velocity sub-models extracted from the Marmousi velocity model (see chapter 6), the traveltime table was constructed. Each column of this table represents the traveltime map calculated at the corresponding position, organized in a columnized format. Fig.6.4 compares the Marmousi traveltime table calculated with first arrivals on the left and the table calculated with the most energetic arrivals on the right. It is evident that the traveltimes in the most energetic arrivals table are generally overestimated.

Creating the two-way traveltime table was straightforward. Since this thesis works with a common offset of 0 m, the one-way traveltime tables for the source and receiver are identical, making the two-way traveltime table simply the one-way table with doubled values. Fig.7.29 illustrates the formation of the two-way traveltime table.

To achieve optimal results from the migration algorithm described in Section 3.3, parameters such as migration aperture and the depth at which the maximum migration aperture is reached must be carefully tuned. Fig.7.32 presents the Marmousi velocity model on the left, while the right side showcases various migration results characterized by the same migration aperture but different depths at which the maximum migtation aperture is reached. Images (a), (b), (c), and (d) correspond to depths of 1 km, 2 km, 3 km, and 4 km, respectively, with the best resolution observed in image (a).

In Fig.7.33, the depth is fixed at 1 km while varying the migration aperture: image (a) shows

an aperture of 1 km, image (b) an aperture of 2 km, and so on, with the selected migration aperture set to 3 km. Fig.7.34 illustrates the chosen migration parameters on the left, where the maximum migration aperture of 3 km is achieved at a depth of 1 km. A cosine taper function is applied at the edges, affecting the outermost 20% of the aperture. The right side presents the impulse response of the trace migrated using these illustrated parameters.

Once the optimal migration parameters were determined, migration was performed. The top image of Fig.7.35 represents the Marmousi velocity model, while the bottom image displays the resulting migration from zero-offset data and the two-way traveltime table derived from the most energetic traveltime maps calculated using Devito. The parameters used for the migration are those discussed previously. Notably, the migrated image shows no artifacts at the lateral edges, as areas where traces were replicated during the extension of the model (see chapter 6) have been excluded from the final migration results. The black boxes identify the central region of the Marmousi model, representing the geologically most complex part of the area. Fig.7.36 represents a close-up of the previous figure. At the top is the velocity model taken from the central part of the Marmousi model, and below is the migrated model, with a maximum migration aperture of 3 km and a depth at which it is reached of 1 km.

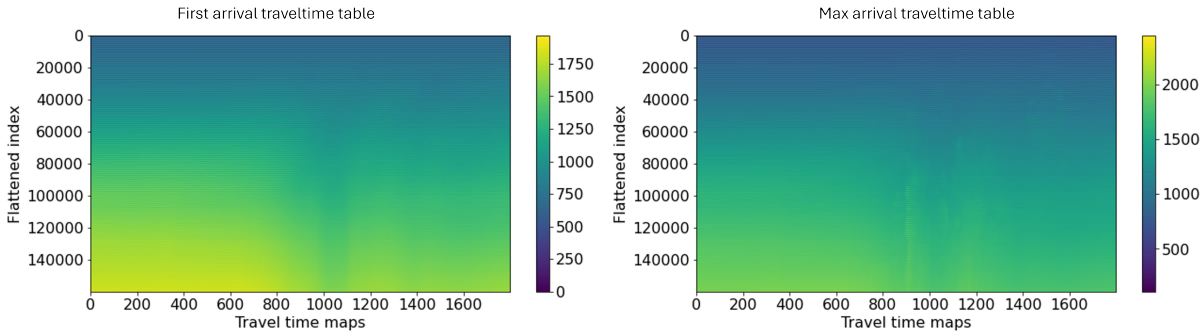


Figure 7.28: On the left is the traveltime table of the first arrivals, while on the right is the traveltime table of the most energetic arrivals.

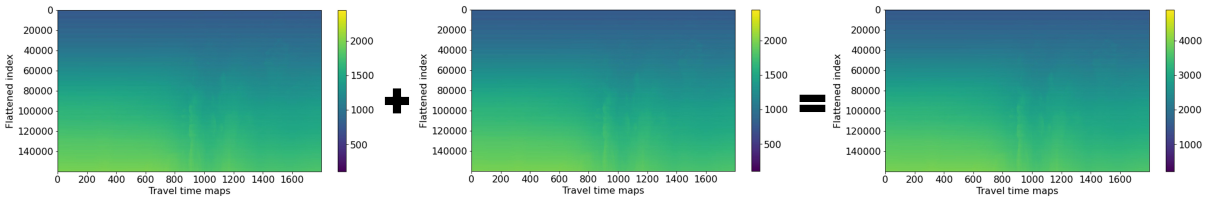


Figure 7.29: An example of the formation of the two-way traveltime table, which enables the migration of seismic traces. It is obtained by summing the traveltime table of the source and the traveltime table of the receivers.

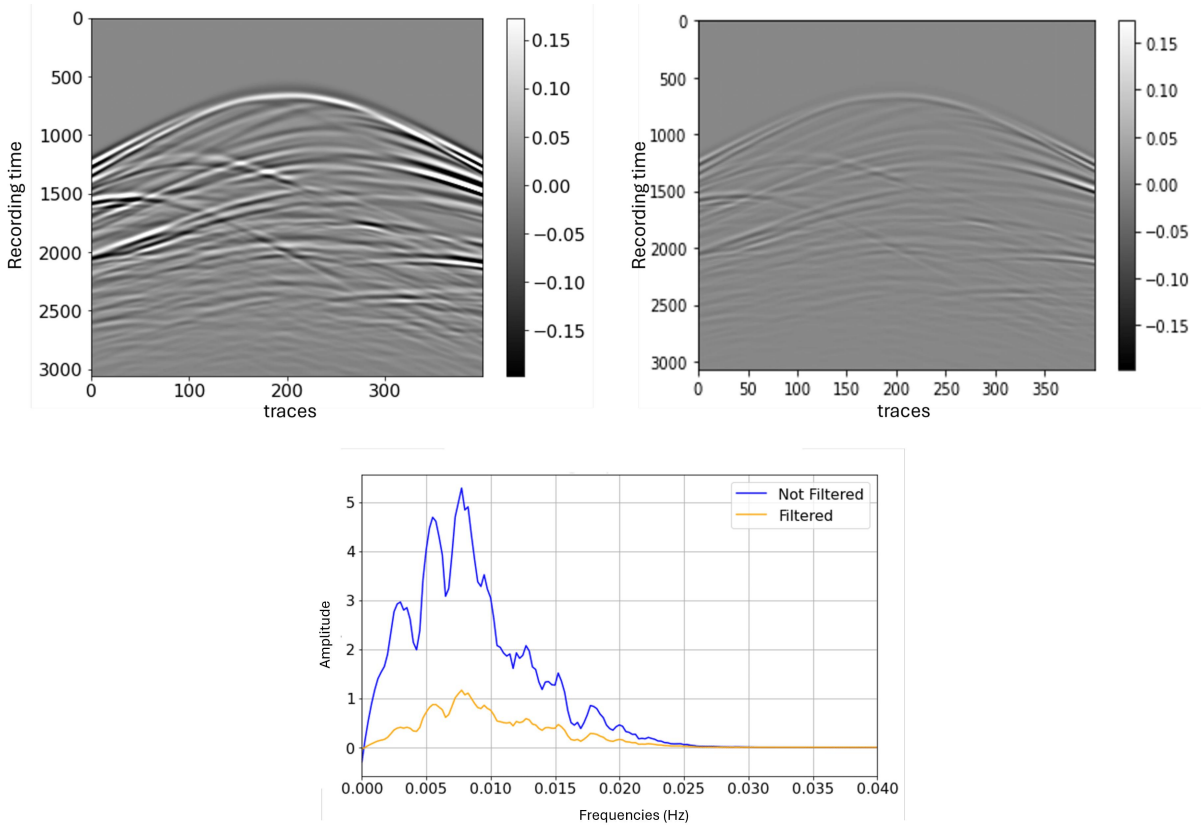


Figure 7.30: On the left unfiltered shot gather, on the right filtered one and below a comparison of their respective spectra.

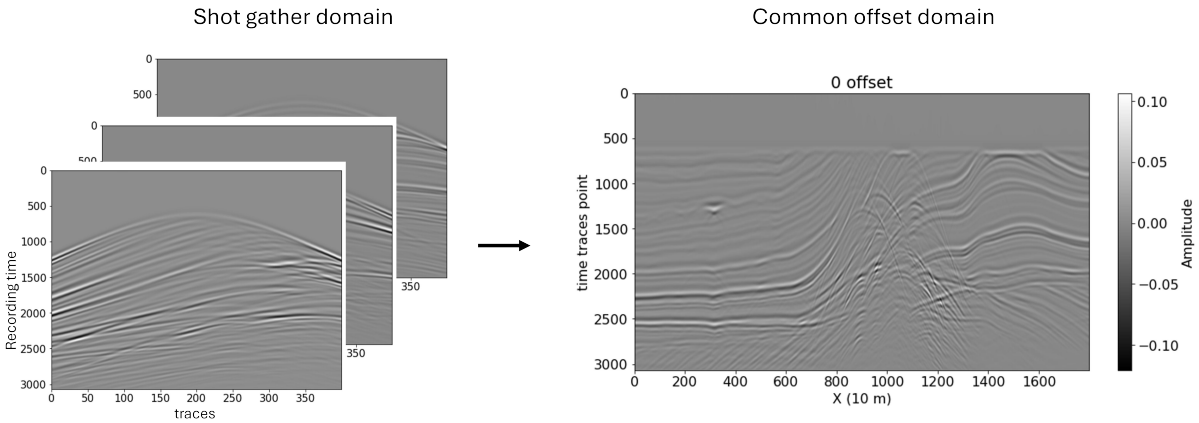


Figure 7.31: Transition from the shot gather domain to the common offset domain, a process carried out using a sorting algorithm

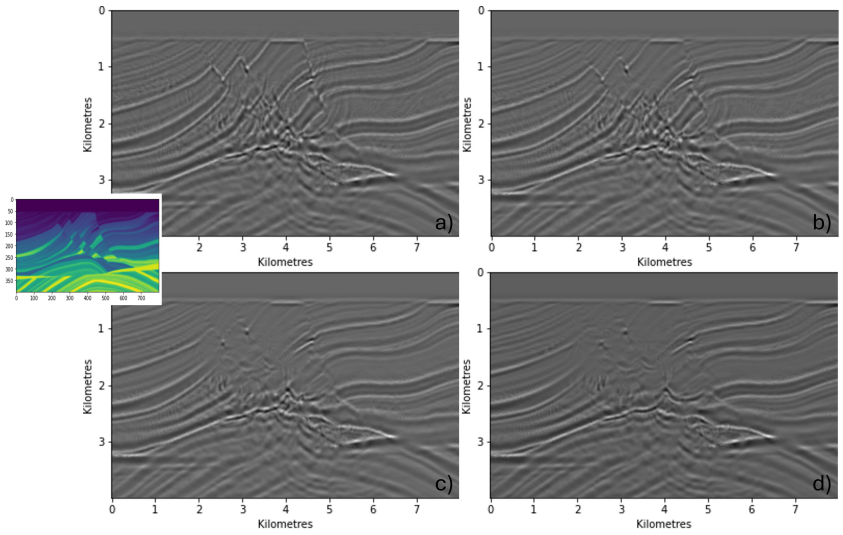


Figure 7.32: These images represent the tuning of the depth at which the maximum migration aperture is reached. The maximum migration aperture was kept constant.

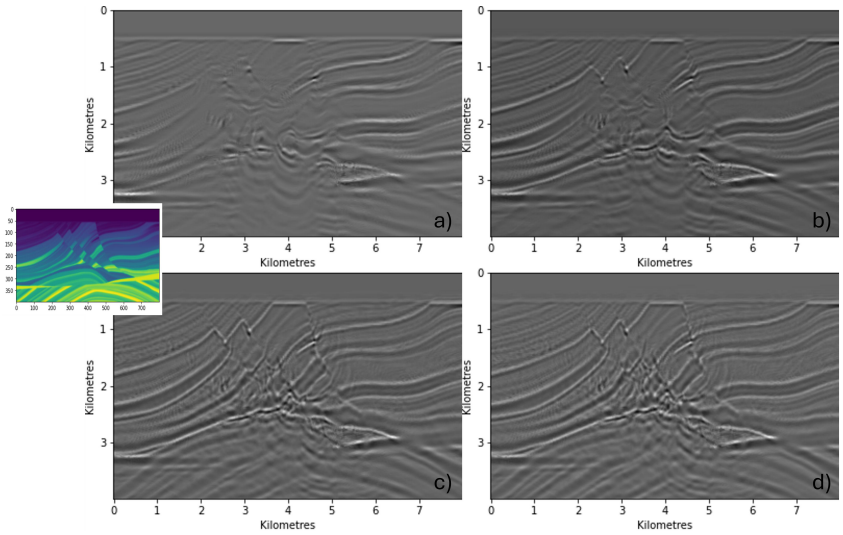


Figure 7.33: These images represent the tuning of the maximum migration aperture on the central part of Marmousi velocity model. The depth at which the maximum migration aperture was reached was kept constant.

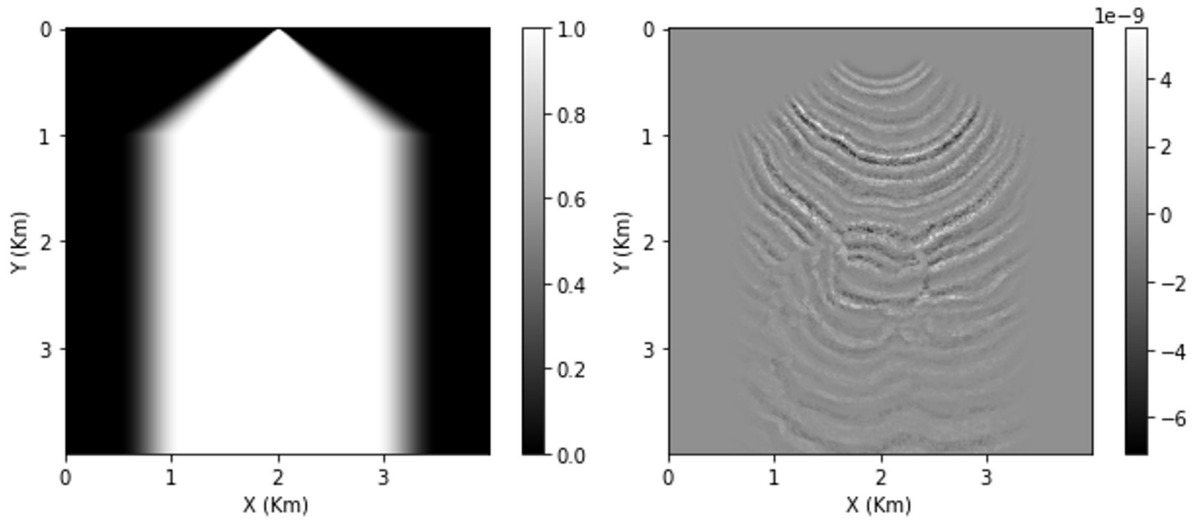


Figure 7.34: The image on the left depicts the migration parameters. On the right, the impulse response of the trace is presented, migrated using the migration parameters illustrated on the left.

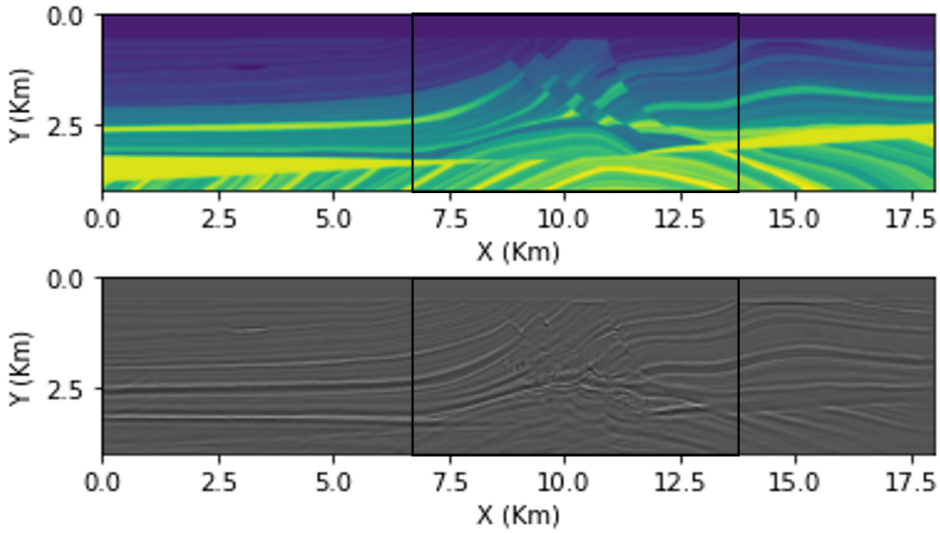


Figure 7.35: The image at the top represents the Marmousi velocity model, while the image at the bottom displays the resulting migration. The parameters used for the migration are the ones discussed above. The black boxes identify the central region of the Marmousi model, representing the geologically most complex part of the area.

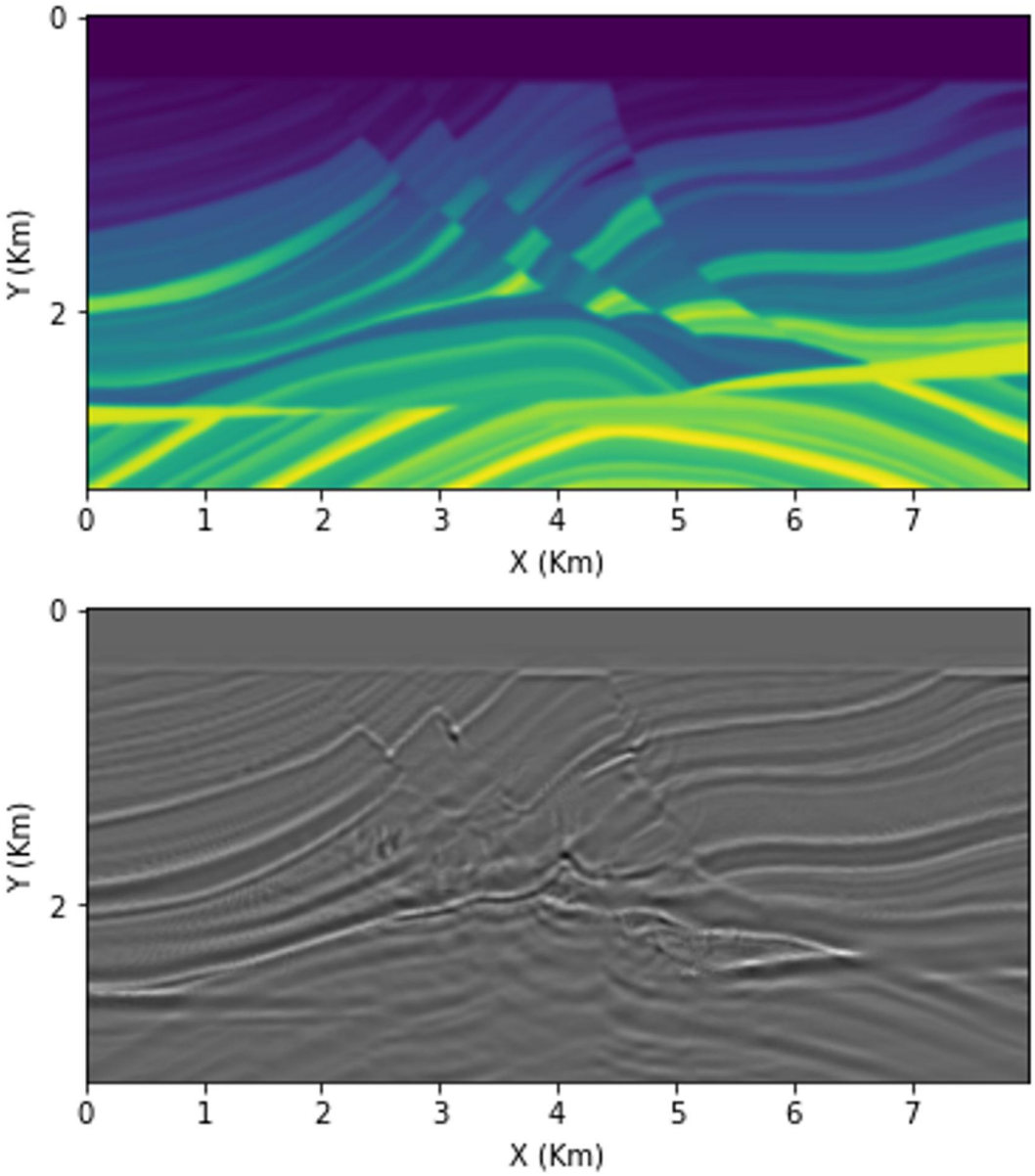


Figure 7.36: Close-up of the previous figure. At the top is the velocity model taken from the central part of the Marmousi model, and below is the migrated model.

## 7.6 Final results

To align with the objectives of this thesis, Figure 7.43 is used to reiterate that migration is more accurate with the traveltime of the most energetic arrivals (subfigure c), which serves as the benchmark in this study. This benchmark is computed from the most energetic traveltimes calculated using the Devito software. in comparison to the migration performed with first-arrival traveltime (subfigure b). As discussed in the introduction and in chapter 3, using first-arrival traveltime can lead to inaccuracies in areas with high geological complexity. This is evident in the red-circled areas of the model, located in its most internal section, which is characterized by faults and tilted strata. In these regions, the image migrated with first-arrival traveltime fails to provide a sufficiently accurate image, whereas the image using the most energetic arrivals yields a much clearer result. It is important to note that the most energetic traveltime represents the best approximation when considering the use of a single traveltime for the entire wavefield.

In figure Fig.7.38, three traveltime tables are plotted: from left to right, the first was created using traveltime maps obtained from Devito based on the Marmousi velocity model (benchmark); the second was derived from the same Marmousi model but calculated using a U-Net network; and the third was generated using the diffusion model. As observed, the traveltime table from the diffusion model closely matches the reference table calculated with Devito. The table in Fig.7.39 presents the time required for different models to compute the most energetic arrival traveltime maps from their respective inputs. For the Devito model, this computation is made directly from the velocity model to the energetic arrival traveltime map, taking 5 seconds. In contrast, for both the trained U-Net and the trained diffusion model, the computation involves generating the most energetic arrival traveltime map from both the velocity model and the first arrival traveltime map. The U-Net network, completes this process approximately $10^2$ times faster than Devito. The diffusion model, however, is significantly slower due to its 2000 iterations within the time U-Net framework, requiring about 30 times more time than the Devito model.

The final figure (Fig.7.40) presents a comparison of the migration results using the traveltimes obtained from Devito, those obtained through the U-Net, and those from the diffusion model derived from the U-Net outputs. The U-Net tends to yield results that are nearly identical to the label (Figure a of Fig.7.40), where geological complexity is lower, but it struggles more in the central part, achieving only a moderate level of accuracy. Conversely, the diffusion model

achieves excellent results even in the central part (Figure b of Fig.7.40). The areas of significant importance for the comparision are marked in red.
In conclusion, it is evident that the diffusion model outperforms the other methods. However, it is crucial to stress that generating each traveltime map for constructing the traveltime table used in the diffusion model required significantly more time than that required by the U-Net.

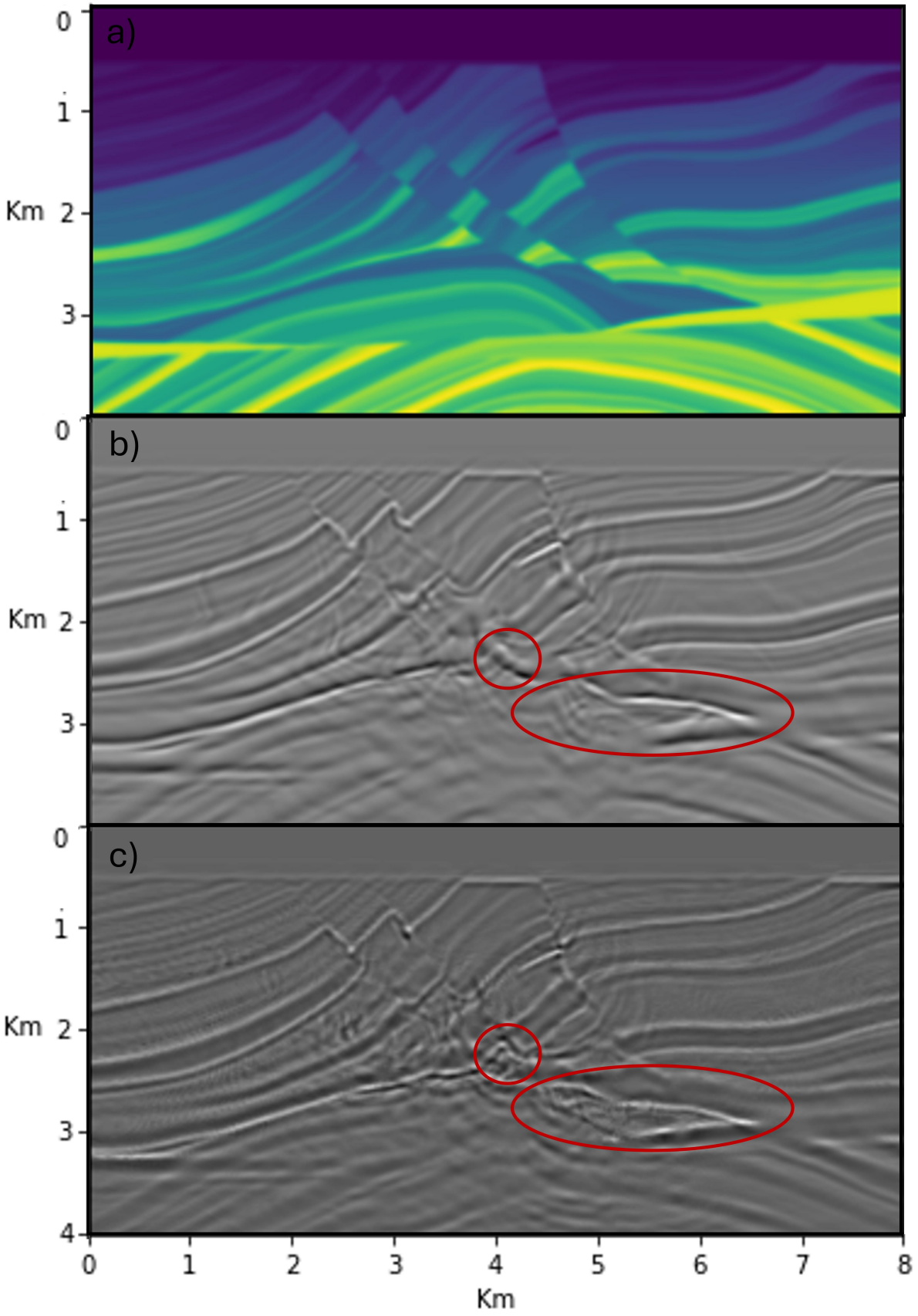


Figure 7.37: Comparison: a) velocity model, b) Kirchhoff migrated area using first arrival traveltimes, c) Kirchhoff migrated area using max energetic arrival traveltimes

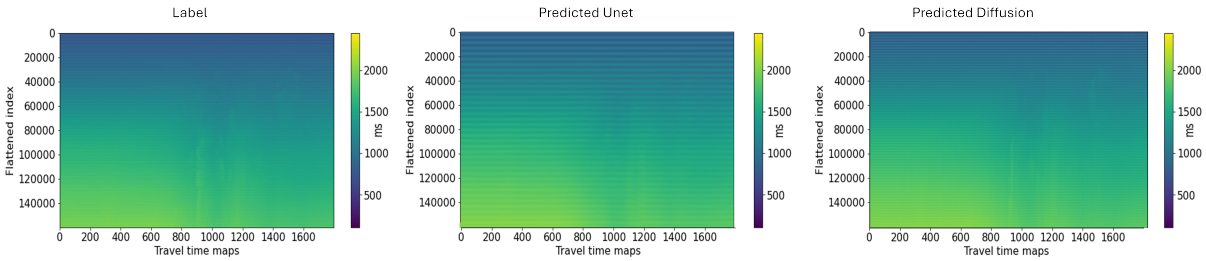


Figure 7.38: On the left traveltime table calculated using the max amplitude traveltime maps calculated with Devito, in the center traveltime table calculated using max amplitude traveltime maps predicted with U-Net, on the right traveltime table calculated using max amplitude traveltime maps predicted with diffusion model

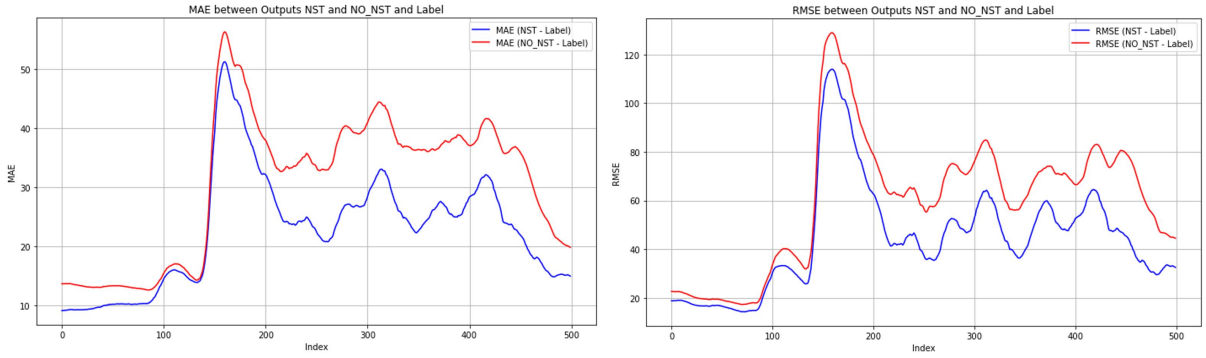


Figure 7.39: The table above presents the time required for different models to transition from their respective inputs to the most energetic arrival traveltime maps.

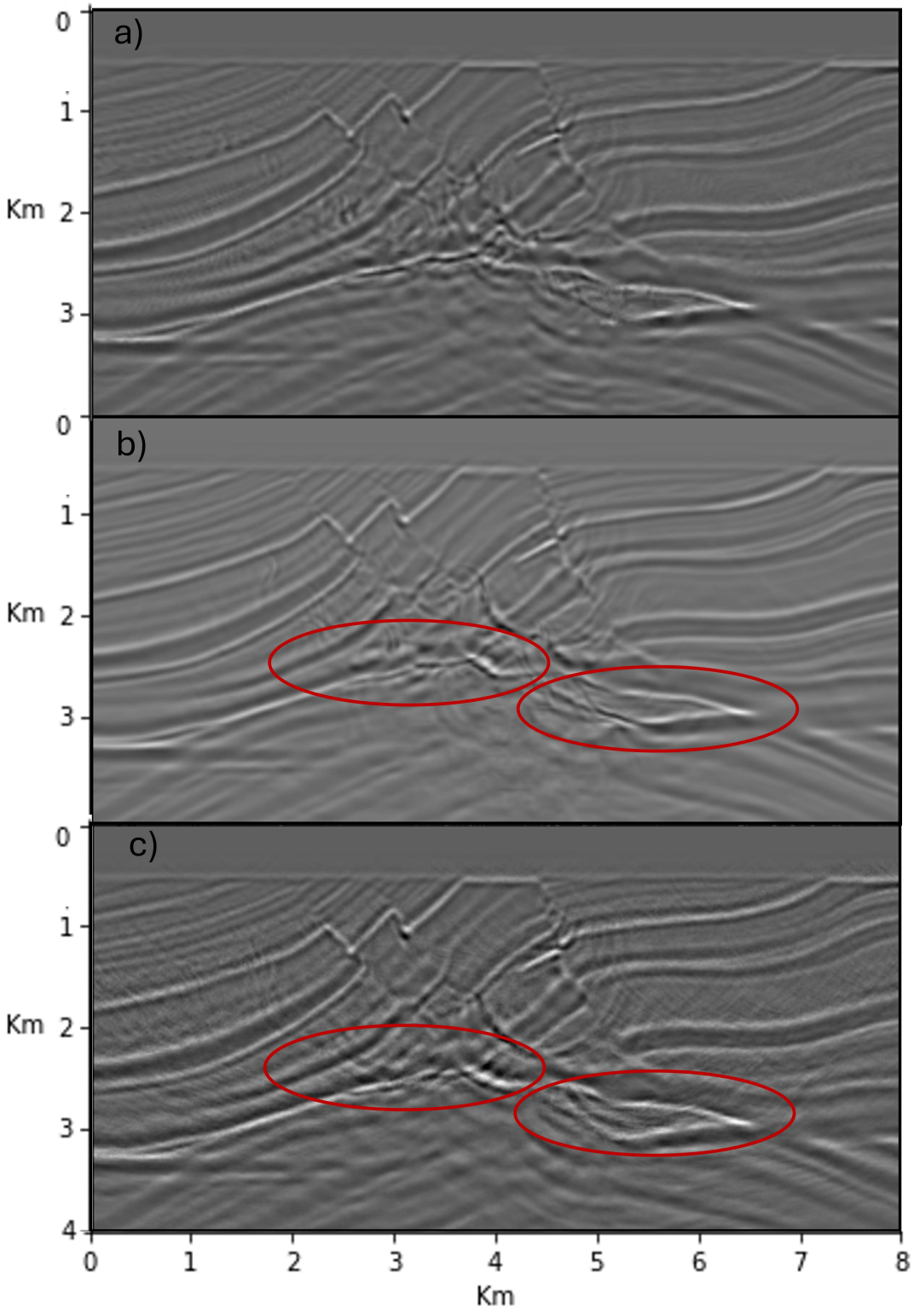


Figure 7.40: Comparison: a) Kirchhoff migrated area using max energetic arrival traveltimes, b) Kirchhoff migrated area using max energetic arrival traveltimes calculated using U-Net, c) Kirchhoff migrated area using max energetic arrival traveltimes calculated using Diffusion model

# Chapter 8

# Discussion and Conclusions

In recent years, with the widespread adoption of Full Waveform Inversion (FWI) for building high-resolution velocity models, Kirchhoff migration using the most energetic arrivals has regained interest as a quality control tool for FWI, offering the advantage over first arrival Kirchhoff migration of generating accurate migrated images even in complex geological settings. Within this line of research this thesis represents a novel contribution to the field, addressing a gap in the existing literature. By applying advanced techniques from other domains, such as machine learning and deep learning, to concepts that were initially proposed decades ago, this work explores the potential of using traveltimes of the most energetic arrivals for Kirchhoff migration, as opposed to the more commonly used first arrivals. This thesis aims to apply this technique while minimizing the computational cost of calculating the most energetic arrivals through the integration of U-Net architectures and diffusion models.
To provide a structured overview of the research, the thesis can be divided into three main steps:

- **Wave Equation Solution and Traveltime Extraction:** The first step involved solving the 2D acoustic wave equation and extracting the traveltimes of the most energetic arrivals based on velocity models, which was essential for dataset creation. This was achieved through the development of a custom code that employs the finite difference method to solve the wave equation. Particular attention was given to reducing computational time by optimizing the grid spacing, time step ($\Delta t$), and the size of absorbing boundaries. These parameters were carefully balanced to avoid issues such as dispersion and instability, ensuring that the results were both accurate and computationally efficient.

- **Architecture Selection, Training, and Hyperparameter Tuning:** The second area focused on selecting the appropriate machine learning architecture and optimizing its performance. The U-Net architecture was trained, and ensembling techniques, such as AUnet and ResAUnet, were explored to enhance the robustness of the models. A key innovation that emerged during the course of this research was the decision to provide the U-Net with two inputs, the velocity model and the first arrivals, instead of just one. This approach, which evolved as the work progressed, provided a substantial boost to the model's performance. Subsequently, the dataset was further enhanced using Neural Style Transfer (NST) techniques, which contributed to better model generalization. Hyperparameter tuning also played a crucial role in refining the network's effectiveness, highlighting the importance of careful parameter optimization in machine learning workflows.

- **Migration Algorithm Development:** Finally, the third area involved writing the migration algorithm itself, where careful tuning of migration parameters was crucial for achieving the desired results.

This thesis should be regarded as a precursor in this area, demonstrating the feasibility of integrating machine learning techniques to expedite the migration process while simultaneously improving accuracy compared to traditional methods that rely solely on first arrival traveltimes.
The final results are promising, indicating the validity of the proposed approach. The key findings can be summarized as follows:

- Increasing **the number of samples** in the training dataset has led to improved network performance.

- The application of **data augmentation techniques such as NST**, which introduced greater complexity to the training dataset, proved to be a winning strategy.

- **The U-Net model** is capable of delivering excellent results in geologically simple scenarios.

- The exploration of various **U-Net variants and ensembling techniques** did not yield any significant changes in performance.

- In more **complex geological scenarios**, the U-Net model struggles, requiring the support of a **diffusion model** to achieve satisfactory results, albeit at the cost of increased computational time.
- There is **room for improvement**, particularly in reducing the time required for the diffusion models to compute the most energetic traveltimes.This could be achieved, for example, by reducing the steps in the forward and reverse processes or by simplifying the temporal U-Net architecture used within the diffusion model.
- It would also be interesting to apply the developed methods to **real-world data** to further validate and refine the approach.

In conclusion, while the primary objective of reducing computational time has been partially achieved, thanks to the U-Net's rapid processing capabilities, the challenge of maintaining high resolution in geologically complex areas remains significant. The application of diffusion models in this field, although promising, requires further investigation and refinement. These architectures are highly dynamic, and their potential in geophysical applications is still largely unexplored. Continued research and development in this area could unlock new possibilities, ultimately leading to more efficient and accurate solutions in seismic imaging.

# Appendix A

# Computational resources

All computations for this thesis were carried out on the Galileo100 cluster, which is co-funded by the European ICEI (Interactive Computing e-Infrastructure) project and engineered by DELL. Access to this high-performance computing environment was provided through the Eni bastion host known as Cometa.

The computing nodes in the cluster are equipped with Intel Xeon Platinum 8260 CPUs, each node having 48 CPUs and 2 GPUs. The Intel Xeon Platinum 8260 CPUs are capable of operating at a base clock speed of 2.40 GHz, with a maximum clock speed of 3.90 GHz. The GPUs installed are NVIDIA Tesla V100 models, which offer 32 GB of memory per GPU.

The computational tasks were managed using Spyder IDE with Python version 3.11.8. To enhance performance, the solution of the wave equation for velocity models was efficiently parallelized. This was achieved by utilizing multiple shell scripts; specifically, 20 shell files were executed in parallel, each initiating 20 separate processes that solved the wave equation simultaneously. This setup allowed for the concurrent use of multiple CPU cores.

For accelerating neural network training, CUDA was employed to leverage the GPU capabilities. This approach significantly improved the speed of computations by utilizing the parallel processing power of the NVIDIA Tesla V100 GPUs.

# Bibliography


[1] D. Nichols. Maximum energy traveltimes calculated in the seismic frequency band. *Stanford Exploration Project*, 80:1–18, 2001.

[2] Y. Pu, G. Liu, D. Wang, H. Huang, and P. Wang. Wave-equation traveltime and amplitude for kirchhoff migration. *First International Meeting for Applied Geoscience Energy*, 1:2684–2687, 2021.

[3] Y. Wang, Y. He, A. Yeh, F. Liu, B. Wang, and C. Calderón. Improve kirchhoff depth imaging using full wave equation traveltimes. *Third International Meeting for Applied Geoscience Energy*, 1:1598–1601, 2023.

[4] J. Jin and J. Etgen. Evaluating kirchhoff migration using wave-equation generated maximum amplitude traveltimes. *SEG International Exposition and 90th Annual Meeting*, 90:2968–2971, 2020.

[5] H. Jin, V. Bashkardin, P. Jilek, C. Kumar, H. Liu, J. Etgen, M. Vyas, and G. Xia. Wave equation traveltime kirchhoff with real data applications. *Third International Meeting for Applied Geoscience Energy*, 3:1638–1641, 2023.

[6] A. Pierce. Acoustics – an introduction to its physical principles and applications. *Acoustical Society of America*, 1991.

[7] Lehtinen J. Time-domain numerical solution of the wave equation. 2003.

[8] E. Kieri. Numerical methods for wave propagation. *Uppsala University*, 2016.

[9] R. M. Alford, K. R. Kelly, and D.M. Boore. Accuracy of finite difference modeling of the acoustic wave equation. *Geophysics*, 39:834–842, 1974.

[10] L. Lines, R. Slawinski, and R. Bonding. A recipe for stability of finite-difference wave-equation computations. *Geophysics*, 64:967–969, 1999.

[11] R. Clayton and B. Engquist. Seismic ray theory. *Cambridge University Press*, 2001.

[12] J.-P. Bèrenger. A perfectly matched layer for the absorption of electromagnetic waves. *J. Comput. Phys., vol. 114,*, pages 185–200, 1994.

[13] M. Louboutin, M. Lange, F. Luporini, N. Kukreja, P. A. Witte, F. J. Herrmann, P. Velesko, and G. J. Gorman. Devito: an embedded domain-specific language for finite differences and geophysical exploration. *Devito introduction*, 2018.

[14] W. Schneider. Developments in seismic data processing and analysis (1968-1970). *Geophysics*, 36:1043–1073, 1971.

[15] J. Zhu and L. Lines. Comparison of kirchhoff and reverse-time migration methods with applications to prestack depth imaging of complex structures. *Geophysics*, 63:1166–1176, 1998.

[16] K. Aki and P. Richards. Quantitative seismology, theory and methods. *Quantitative Seismology, Theory and Methods*, 1980.

[17] V. Cerveny. Absorbing boundary conditions for acoustic and elastic wave equations. *Bulletin of The Seismological Society of America*, 1977.

[18] N. Bleistein, J. Cohen, and F. Hagin. Two and one-half dimensional born inversion with an arbitrary reference. *Geophysics*, 57:26–36, 1987.

[19] F. Audebert, D. Nichols, T. Rekdal, B. Biondi, D. Lumley, and H. Urdaneta. Imaging complex geologic structure with single-arrival kirchhoff prestack depth migration. *Geophysics*, 62:1533–1543, 1997.

[20] B. Nguyen and G. McMechan. Excitation amplitude imaging condition for prestack reverse-time migration. *Geophysics*, 78:37–46, 2013.

[21] A. Ehinger, P. Lailly, and K. Marfur. Green's function implementation of common-offset wave equation migration. *Geophysics*, 61:1813–1821, 1996.

[22] J. Etgen. 3d wave equation kirchhoff migration: 82nd annual international meeting. *SEG, Expanded Abstracts*, 2012.

[23] P. Andrade, R. Pestana, and A. dos Santos. Kirchhoff depth migration using maximum amplitude traveltimes computed by the chebyshev polynomial recursion. *14th International Congress of the Brazilian Geophysical Society and EXPOGEF, Expanded Abstracts*, pages 1109–1113, 2015.

[24] N. Bleistein, J. Cohen, and J. Stockwell. Mathematics of multidimensional seismic inversion. *Springer*, 1998.

[25] Y. Zhang, S. Gray, and J. Young. Exact and approximate weights for kirchhoff migration. *SEG 2000, Expanded Abstracts*, 2000.

[26] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv*, 2015.

[27] O. Oktay, J. Schlemper, L. Le Folgoc, M. Heinrich M. Lee, K. Misawa, K. Mori, S. McDonagh, N. Y. Hammerla, B. Kainz, B. Glocker, and D. Rueckert. Attention u-net: Learning where to look for the pancreas. *arXiv*, 2018.

[28] Z. Ni, G. Bian, X. Zhou, Z. Hou, X. Xie, C. Wang, Y. Zhou, R. Li, and Z. Li. Raunet: Residual attention u-net for semantic segmentation of cataract surgical instruments. *arXiv*, 2019.

[29] L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. *arXiv*, 2015.

[30] D. Liu and J. Nocedal. On the limited memory method for large scale optimization. *Mathematical Programming B.*, 1989.

[31] D. P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv*, 2017.

[32] Z. Zhou. Ensemble methods: Foundations and algorithms. *Chapman Hall/CRC*, 2012.

[33] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *International Conference on Machine Learning*, pages 2256–2265, 2015.

[34] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, pages 6840–6852, 2020.

[35] I. Goodfellow, J. Pouget-Abadi, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks. *arXiv*, 2014.

[36] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv*, 2022.

[37] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 1951.

[38] R. Durall, A. Ghanim, M. Fernandez, N. Ettrich, and J. Keuper. Deep diffusion models for seismic processing. *AirXiv*, 2020.

[39] R. Versteeg. The marmousi experience: Velocity model determination on a synthetic complex data set. *The Leading Edge*, pages 927–936, 1994.

[40] V. Hodge and J. Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, pages 85–126, 2004.