

**INVERSION OF VERTICAL SEISMIC PROFILING  
(VSP) DATA VIA CONSECUTIVE NEURAL  
NETWORKS**

BY  
**HUSSAIN ABBAS ALMARZOOQ**

A Thesis Presented to the  
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the  
Requirements for the Degree of

**MASTER OF SCIENCE**

In  
**GEOPHYSICS**

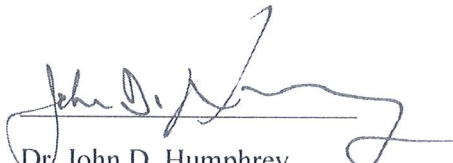
**SEPTEMBER 2023**

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN- 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

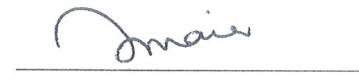
This thesis, written by **Hussain Almarzooq** under the direction of his thesis advisor and approved by his thesis committee, has been presented and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN GEOPHYSICS**.

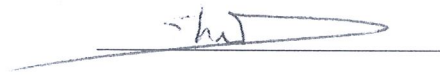
  
Dr. John D. Humphrey  
Department Chairman

  
Prof. Suliman Saleh Al-Homidan  
Dean of Graduate Studies

26/09/2023

Date

  
Dr. Umair bin Waheed  
(Advisor)

  
Dr. Sherif M. H. Mahmoud  
(Member)

  
Dr. Motaz A. Alfarraj  
(Member)



© Hussain Almarzooq

2023

*This thesis is dedicated to my family, whose love and support during this period of my life made this work possible.*

## ACKNOWLEDGMENTS

First, I would like to acknowledge the support of my family, friends, and colleagues for bearing with me during the three years of study at KFUPM which resulted in producing this thesis work. It was tough balancing between home, work, school, and sleep. This delicate balance wouldn't have been possible without the understanding of my wife, parents, and siblings whom I would have wished to have been able to dedicate more of myself to during these past three years.

Secondly, my educational journey wouldn't have been the same if I hadn't had the honor of learning with and from Dr. Umair and Dr. Sherif during GEOP599 which covered the basics of machine learning as it pertains to geoscientists, a field that I was very weak in prior to joining KFUPM. My education was further enhanced by the two courses GEOP603 & GEOP629 in which I had the delight of computationally solving several geophysical problems with Prof. Peter. I would also like to thank Dr. Abdullah for ensuring that we had solid knowledge of fundamental concepts through GEOP501, GEOP503, and GEOP599, in which he always emphasized the need for both social and geophysical skills.

Finally, I would like to thank my colleagues at the Smart Earth Exploration and Monitoring (SEEM) group for their interesting talks and thoughts during all of our meetings as well as Dr. Motaz Alfaraj for his comments and thoughts which helped enhance this work further. Finally, I would like to thank all KFUPM faculty for their time and the exceptional environment which they cultivated at KFUPM, it will stick with me for the rest of my academic and professional careers. Thank you, a thousand times, to my family, friends, colleagues, mentors, and KFUPM faculty.

# TABLE OF CONTENTS

|                                                          |      |
|----------------------------------------------------------|------|
| ACKNOWLEDGMENTS .....                                    | V    |
| TABLE OF CONTENTS.....                                   | VI   |
| LIST OF TABLES .....                                     | IX   |
| LIST OF EQUATIONS .....                                  | X    |
| LIST OF FIGURES .....                                    | XI   |
| LIST OF ABBREVIATIONS.....                               | XIV  |
| ABSTRACT.....                                            | XVI  |
| ملخص الرسالة .....                                       | XVII |
| CHAPTER 1 INTRODUCTION.....                              | 1    |
| 1.1 Opening Statement.....                               | 1    |
| 1.2 Motivation.....                                      | 2    |
| 1.3 Literature Review .....                              | 3    |
| 1.3.1 Vertical Seismic Profiling (VSP) .....             | 3    |
| 1.3.2 Deep Learning .....                                | 15   |
| 1.3.3 The Role of Machine Learning in VSP Inversion..... | 25   |
| 1.4 Aims and Objectives .....                            | 26   |
| CHAPTER 2 METHODOLOGY .....                              | 28   |
| 2.1 Synthetic Data Generation .....                      | 28   |
| 2.2 Data Pre-Processing.....                             | 34   |
| 2.2.1 Standard Pre-Processing Steps .....                | 34   |
| 2.2.2 Wavelet Transforms.....                            | 36   |

|                                   |                                                               |            |
|-----------------------------------|---------------------------------------------------------------|------------|
| 2.2.3                             | Handling of Real Datasets .....                               | 40         |
| 2.3                               | Neural Networks Architectures .....                           | 41         |
| 2.4                               | Consecutive Neural Networks.....                              | 42         |
| <b>CHAPTER 3 RESULTS .....</b>    |                                                               | <b>45</b>  |
| 3.1                               | Direct Inversion of The Seismic Trace .....                   | 45         |
| 3.1.1                             | Hyperparameter Optimization .....                             | 46         |
| 3.1.2                             | Further Refinements.....                                      | 58         |
| 3.1.3                             | Inversion for Other Properties .....                          | 66         |
| 3.1.4                             | Blind Testing on Finite Difference Dataset .....              | 73         |
| 3.1.5                             | Summary of Direct Inversion Results .....                     | 79         |
| 3.2                               | Inversion of The Seismic Trace via Wavelet Transforms .....   | 81         |
| 3.2.1                             | Stationary Wavelet Transforms (SWT) .....                     | 82         |
| 3.2.2                             | Continuous Wavelet Transforms (CWT).....                      | 91         |
| <b>CHAPTER 4 DISCUSSION .....</b> |                                                               | <b>99</b>  |
| 4.1                               | Direct Inversion versus Wavelet Transformation Inversion..... | 100        |
| 4.2                               | Denoising: When is it Appropriate? .....                      | 101        |
| 4.3                               | Encoders as a Computational Performance Boost.....            | 102        |
| 4.4                               | The Argument for Consecutive Networks .....                   | 104        |
| 4.5                               | Finite Difference Data and How it Relates to Real Data .....  | 107        |
| <b>CHAPTER 5 CONCLUSIONS.....</b> |                                                               | <b>108</b> |
| 5.1                               | Summary and Remarks.....                                      | 108        |
| 5.2                               | Future Work.....                                              | 109        |
| <b>APPENDIX .....</b>             |                                                               | <b>110</b> |
| A1.                               | Computational Environment .....                               | 110        |

|                                              |            |
|----------------------------------------------|------------|
| <b>A2. Code High-Level Walkthrough .....</b> | <b>111</b> |
| <b>REFERENCES .....</b>                      | <b>126</b> |
| <b>VITAE.....</b>                            | <b>134</b> |

## LIST OF TABLES

|                                                                                                    |     |
|----------------------------------------------------------------------------------------------------|-----|
| <b>Table 1:</b> <i>Density and Velocity Empirical Relationship for Different Lithologies</i> ..... | 30  |
| <b>Table 2:</b> <i>Wavelet Transformation Inversion Models Summary</i> .....                       | 100 |
| <b>Table 3:</b> <i>Denoising Effect on Inversion Models Summary</i> .....                          | 102 |
| <b>Table 4:</b> <i>Encoder Effect on Inversion Models Summary</i> .....                            | 104 |

## LIST OF EQUATIONS

|                                                                               |    |
|-------------------------------------------------------------------------------|----|
| <b>(2.1)</b> <i>1D Earth Reflectivity Equation</i> .....                      | 29 |
| <b>(2.2)</b> <i>Empirical Relationship between Density and Velocity</i> ..... | 30 |
| <b>(2.3)</b> <i>Reflectivity Coefficient Equation</i> .....                   | 34 |
| <b>(2.4)</b> <i>Min-Max Scaling Equation</i> .....                            | 35 |

## LIST OF FIGURES

|                                                                                                                                 |    |
|---------------------------------------------------------------------------------------------------------------------------------|----|
| <b>Figure 1:</b> Schematic of the most commonly used VSP Geometries -----                                                       | 7  |
| <b>Figure 2:</b> ZVSP Processed Upgoing Wavefield & Corridor Stack -----                                                        | 8  |
| <b>Figure 3:</b> ZVSP Processing Workflow -----                                                                                 | 10 |
| <b>Figure 4:</b> ZVSP Corridor Stack Lookahead Inversion -----                                                                  | 14 |
| <b>Figure 5:</b> Simple Neural Network Schematic -----                                                                          | 18 |
| <b>Figure 6:</b> Two Layer CNN Schematic -----                                                                                  | 21 |
| <b>Figure 7:</b> CNN-LSTM Schematic -----                                                                                       | 22 |
| <b>Figure 8:</b> TCN Schematic -----                                                                                            | 24 |
| <b>Figure 9:</b> Example of Generated Models -----                                                                              | 32 |
| <b>Figure 10:</b> Example of Generated Datasets -----                                                                           | 33 |
| <b>Figure 11:</b> Example of SWT on Generated Datasets -----                                                                    | 38 |
| <b>Figure 12:</b> Example of CWT on Generated Datasets -----                                                                    | 39 |
| <b>Figure 13:</b> Consecutive Neural Network Proposed Schematic -----                                                           | 44 |
| <b>Figure 14:</b> First Denoising HPO Optuna Study -----                                                                        | 49 |
| <b>Figure 15:</b> Second Denoising HPO Optuna Study -----                                                                       | 50 |
| <b>Figure 16:</b> First Inversion HPO Optuna Study -----                                                                        | 52 |
| <b>Figure 17:</b> Second Inversion HPO Optuna Study -----                                                                       | 53 |
| <b>Figure 18:</b> Blind Testing and Objective Function for Denoising Models -----                                               | 56 |
| <b>Figure 19:</b> Blind Testing and Objective Function for Inversion Models -----                                               | 57 |
| <b>Figure 20:</b> Blind Testing and Objective Function for Clean, Denoised, and Noisy Trace<br>Inputs to Inversion Models ----- | 60 |
| <b>Figure 21:</b> Encoder-Inversion HPO Optuna Study -----                                                                      | 63 |

|                                                                                                                                                   |         |
|---------------------------------------------------------------------------------------------------------------------------------------------------|---------|
| <b>Figure 22:</b> Blind Testing and Objective Function for Encoder-Inversion versus Standalone Inversion Model                                    | -----64 |
| <b>Figure 23:</b> Blind Testing and Objective Function for Encoder-Inversion versus Standalone Inversion Model                                    | -----65 |
| <b>Figure 24:</b> Blind Testing and Objective Function for Transfer Learning versus Training from Scratch for Acoustic Impedance Inversion Models | -----68 |
| <b>Figure 25:</b> Joint Inversion HPO Optuna Study                                                                                                | -----71 |
| <b>Figure 26:</b> Blind Testing and Objective Function for Joint Inversion Model                                                                  | -----72 |
| <b>Figure 27:</b> Raw Gathers of Finite Difference ZVSP Data                                                                                      | -----76 |
| <b>Figure 28:</b> Processed Gathers and Corridor Stack of Finite Difference ZVSP Data                                                             | -----77 |
| <b>Figure 29:</b> Prediction on Finite Difference Corridor Stack via Encoder-Inversion & Standalone Inversion Models                              | -----78 |
| <b>Figure 30:</b> SWT-CNN Autoencoders HPO Optuna Study                                                                                           | -----84 |
| <b>Figure 31:</b> SWT-TCN Autoencoders HPO Optuna Study                                                                                           | -----85 |
| <b>Figure 32:</b> Blind Testing and Objective Function for Shallow, Intermediate, and Deep SWT-CNN Autoencoder Models                             | -----86 |
| <b>Figure 33:</b> Blind Testing and Objective Function for Shallow SWT-TCN Autoencoder Model                                                      | -----87 |
| <b>Figure 34:</b> Blind Testing and Objective Function for SWT Encoder-Inversion Model                                                            | ---89   |
| <b>Figure 35:</b> Blind Testing and Objective Function for SWT Standalone Inversion Model                                                         | 90      |
| <b>Figure 36:</b> CWT-CNN Autoencoders HPO Optuna Study                                                                                           | -----93 |
| <b>Figure 37:</b> Blind Testing and Objective Function for Shallow CWT-CNN Autoencoder Model                                                      | -----94 |

|                                                                                                   |           |
|---------------------------------------------------------------------------------------------------|-----------|
| <b>Figure 38:</b> Blind Testing and Objective Function for Intermediate CWT-CNN Autoencoder Model | -----95   |
| <b>Figure 39:</b> Blind Testing and Objective Function for Deep CWT-CNN Autoencoder Model         | -----96   |
| <b>Figure 40:</b> Blind Testing and Objective Function for CWT Encoder-Inversion Model            | --98      |
| <b>Figure 41:</b> Final Revised Consecutive Neural Network Schematic                              | ----- 106 |

## LIST OF ABBREVIATIONS

|             |                                |
|-------------|--------------------------------|
| <b>3C</b>   | 3-Components                   |
| <b>CNN</b>  | Convolutional Neural Network   |
| <b>CSTK</b> | Corridor Stack                 |
| <b>CWT</b>  | Continuous Wavelet Transform   |
| <b>DNN</b>  | Deep Neural Network            |
| <b>DT</b>   | Sonic Log                      |
| <b>DWT</b>  | Discrete Wavelet Transform     |
| <b>HPO</b>  | Hyperparameter Optimization    |
| <b>HTI</b>  | Horizontal Transverse Isotropy |
| <b>LSTM</b> | Long-Short Term Memory         |
| <b>NLP</b>  | Natural Language Processing    |
| <b>OWT</b>  | One Way Time                   |
| <b>P-</b>   | Primary Waves                  |
| <b>Q-</b>   | Quality Factor                 |
| <b>RHOB</b> | Density Log                    |
| <b>RMSE</b> | Root Mean Square Error         |
| <b>RNN</b>  | Recurrent Neural Network       |
| <b>S-</b>   | Shear Waves                    |
| <b>SWT</b>  | Stationary Wavelet Transform   |
| <b>TCN</b>  | Temporal Convolutional Network |
| <b>TWT</b>  | Two Way Time                   |

|             |                                        |
|-------------|----------------------------------------|
| <b>VSP</b>  | Vertical Seismic Profiling             |
| <b>VTI</b>  | Vertical Transverse Isotropy           |
| <b>ZVSP</b> | Zero-offset Vertical Seismic Profiling |

# **ABSTRACT**

Full Name : Hussain Abbas Salman Almarzooq

Thesis Title : Inversion of Vertical Seismic Profiling (VSP) Data Via Consecutive Neural Networks

Major Field : Masters of Science in Geophysics

Date of Degree : September 2023

The inversion of zero-offset VSP corridor stacks for velocities and depth prediction is an important tool for hazard prediction ahead of the bit and for extending velocity information below the well's final depth. The conventional approach suffers from issues related to the non-linearity of the inversion especially in areas where data is limited as well as biases related to the choice of wavelet, noise level, low frequency model, and other associated parameters. Thus, a deep learning approach using a consecutive neural network (split-training), consisting of a denoising, feature extraction and compression, and velocity inversion sub-models, is proposed. The combined model can predict results ten times faster than all other tested models at a higher accuracy.

## ملخص الرسالة

الاسم الكامل: حسين بن عباس بن سلمان آل مرزوق.

عنوان الرسالة: انعكاس بيانات التتميط الزلزالي العمودي عبر الشبكات العصبية المتتالية.

التخصص: ماجستير العلوم في الجيوفيزياء.

تاريخ الدرجة العلمية: سبتمبر 2023.

يتم التنبؤ بسرعة الصخور وعمقها باستخدام انعكاس الموجات الزلزالية العمودية منعمة الحيد وهذا مهم للتنبؤ بالأخطار المتعلقة بالحفر وللتنبؤ بسرعات الصخور إلى عمق أكثر من عمق البئر النهائي. الطريقة التقليدية للتنبؤ تعاني من مشاكل متعلقة بعدم خطية عملية الانعكاس وبالأخص في المناطق التي تحتوي معلومات محدودة ، أوحين يوجد تحيز متعلق باختبار نوع الموجة ، أو مستوى الضوضاء ، أو نموذج التردد المنخفض ، أو العديد من العوامل المتعلقة الأخرى. لهذا تم اقتراح عملية التعلم العميق والتي تتم باستخدام شبكة عصبونية متتالية (تدريب متفرق) ، وتحتوي على النماذج الفرعية التالية: نموذج معالجة للضوضاء ، نموذج استخلاص وضغط الميزات ، ونموذج للتنبؤ بالسرعة. النموذج النهائي المشترك يستطيع التنبؤ بالنتائج أسرع بعشرة مرات من كل النماذج الأخرى وبدقة أعلى.



# CHAPTER 1

## INTRODUCTION

### 1.1 Opening Statement

Seismic data is mainly acquired with the purpose of obtaining information about the subsurface such as velocities or Earth properties like porosity or water saturation. Several methods in which these properties can be obtained rely on the process of seismic inversion. The first step of such process usually tries to recover the Earth's reflectivity series from seismic data. This involves estimating a wavelet based on the data and then applying it to obtain the reflectivity series. This thesis will look into a subset of seismic data called Vertical Seismic Profiling (VSP) data and attempt to extract a velocity profile automatically from the VSP corridor stack (essentially a zero-offset seismic trace with no multiples) without the need to explicitly extract the reflectivity series first. This will be accomplished via the use of neural networks trained using synthetically generated noisy pseudo- corridor stacks. To dive deeper into the rationale of why the use of machine learning is proposed, it is needed to first explain how seismic inversion works traditionally.

Conventional seismic inversion processes rely heavily on the quality of input data as well as the estimated wavelet being representative of the inverted data. There exist cases where multiple wavelets (or a "wavelet volume") are created to account for the spatial or temporal variation of the wavelet due to either geology or acquisition

geometry changes. This process however can become quite complex and harder to ensure the consistency of. Thus, this could negatively impact the final inversion results and in turn affect the certainty of any calculations or estimations based on these inversions.

In an effort to decouple the seismic inversion process from the wavelet estimation process, a deep learning approach is used to directly extract the velocity information from the noisy synthetic corridor stacks as this process is not dependent on a particular wavelet or a volume of wavelets, it should be able to dynamically adapt to spatial variations more robustly.

## **1.2 Motivation**

This work is inspired by the need to address industry and academic related challenges. In the industry, the process of inverting VSP data is usually used to “Lookahead” into undrilled formations to allow for a better depth and lithological prediction prior to reaching them. Automating and improving this process of Lookahead VSP will increase confidence in the procedure and improve the reliability of operations that need accurate formation tops. Additionally, and potentially, if this process is reliable then it could be used to “deepen” the velocity information at well locations beyond their final depth which will allow for the creation of better velocity models for seismic data. On the other end of the spectrum, the problem of inverting for velocity from VSP corridor stacks suffers from non-uniqueness so if it is solvable via a deep learning approach this might actually indicate a hidden link beyond this non-uniqueness between the corridor stack

and the velocity profile. If accomplished correctly, the solution of this 1D issue could pave the way for solutions of 3D seismic inversion based on the approaches established here. It could also offer insights into the analysis of seismic traces as time series using combined approaches of pre-processing and deep learning.

## **1.3 Literature Review**

Since this paper attempts to intertwine the process of VSP inversion with machine learning (and deep learning more specifically), it is important that the literature review covers both adequately to lay the ground for the upcoming work. Thus, it will be divided into a section for VSP and machine learning individually as well as a section exploring how the two might be connected for the purpose of VSP inversion.

### **1.3.1 Vertical Seismic Profiling (VSP)**

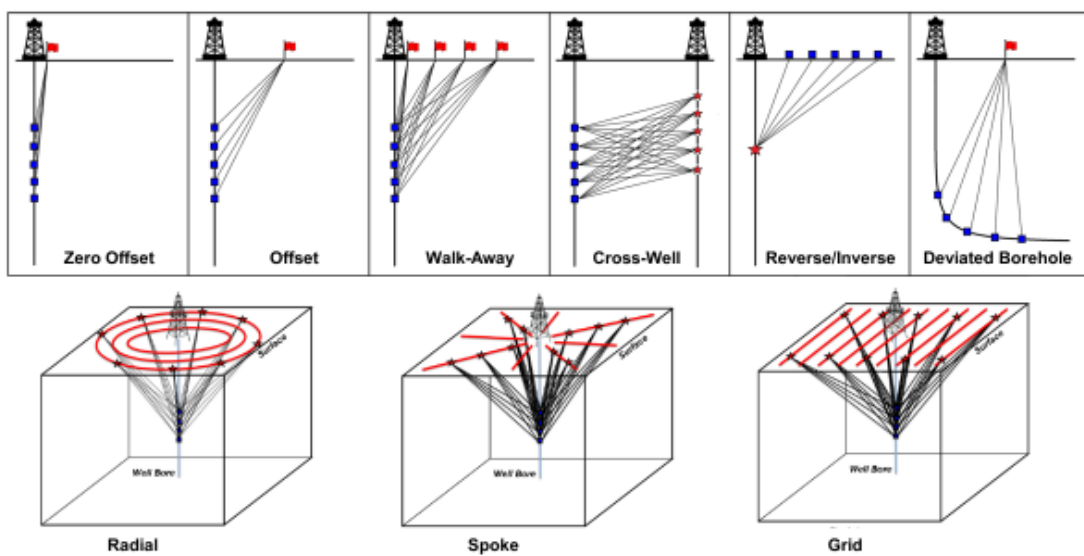
Vertical Seismic Profiling (VSP) is the process of acquiring seismic data inside boreholes in the subsurface as opposed to Horizontal Seismic Profiling commonly known as surface seismic. The first work related to VSP originated in the early 1930s through 1950s in the United States (Hardage, 2020). Early work focused on the application of VSP for direct measurements of velocity only (first break picking) (e.g., Dix, 1939). While other work explored the potential of exploring the full waveform in more details for information about the subsurface (e.g., McCollum & LaRue, 1931; Jolly, 1953; Riggs, 1955; and Levin & Lynn 1958). The enthusiasm of the global geophysical community especially within the

United States and Western Europe was limited (Hardage, 2020). In the Soviet Union however, it was a different story, Gal'perin a Soviet geophysicist in his book about VSP from 1974, shared significant examples of the advances that the Soviet geophysicists made when it comes to VSP acquisition, processing, and interpretation during the period from the 1950s till the 1970s. Only after Gal'perin was employed in the United States for the purpose of transferring VSP knowledge to American geophysicists, did interest in VSP increase (Hardage, 2020; Sheriff & Geldart 1995). During the 1980s, several books regarding VSP were written by American geophysicists such as the two vertical seismic profiling principles books (Hardage, 1984; Toksoz & Stewart 1984) as well as a book by Balch and Lee (1984). These books showed the increased commitment of Western geophysicists to VSP data acquisition, processing, and understanding. Most of the current fundamental concepts remain very similar to what was discussed in these early books.

Looking at VSP from an acquisition point of view, it's a broad term in which multiple acquisition types fall under. Classifying these VSP geometries by source and receiver positions or by source type, four survey types can be established. Firstly, the standard VSP in which the source is located at the surface with the receivers in the borehole (Hardage, 1984). Secondly, reverse VSP where the source and receiver positions are flipped with the source inside the borehole (proliferation gun or dynamite typically) and the receivers at the surface (Hardage, 1992). Third type is cross-well acquisition in which the source and receivers are located in two adjacent wells (usually less than 1 km apart) (Hardage, 1992). The final category that could be considered is passive VSP monitoring in which the source is usually either natural or induced seismicity with the receivers inside the borehole (e.g., Maxwell & Urbancic, 2001).

Focusing on the standard VSP configuration, it also has multiple subcategories based on the source geometry each designed to address a different challenge (see Figure 1). The simplest VSP geometry is called a Zero-Offset VSP (ZVSP) in which there is one source position located as close as possible to the well-head (Oristaglio, 1985). The primary results of such acquisition are an accurate time-depth relationship and a corridor stack. The time depth relationship is obtained up to the final depth of well from picking the direct arrivals. The corridor stack on the other hand, is the result of processing and stacking the upgoing reflections at the well location which is why it contains – mostly – the primary reflections and little to no multiples (see Figure 2 for an example of a corridor stack). Other applications for ZVSP exist such as the estimation of the attenuation with depth and the analysis of multiples, their generators, and travel-paths. Next, there are Offset and Walkaway VSPs in which a single source is offset from the well-head for the former, or multiple equally spaced shots are aligned along a line passing through the well for the latter (Dillon & Thomson, 1984). These two types are mainly used for their enhanced vertical resolution imaging compared to surface seismic. The Walkaway VSP could be further used for the estimation of vertical transverse isotropy (VTI) anisotropy (Bakulin et al., 2010). The fourth geometry type is called Walkaround VSP, and it is when keeping the sources at a constant offset from the well and going azimuthally at a fixed angle step (usually 3°-5°). Technically, imaging of Walkaround VSP data is possible (similar to offset VSP) but the primary motivation of conducting this type of acquisition is to either estimate horizontal transverse isotropy (HTI) anisotropy or to perform fracture characterization (Horne, 2003; Tsuji et al., 2011). The final two geometries that will be discussed here are 3D and 4D VSPs, the first as the name implies is the acquisition of a spiral or grid of shot points

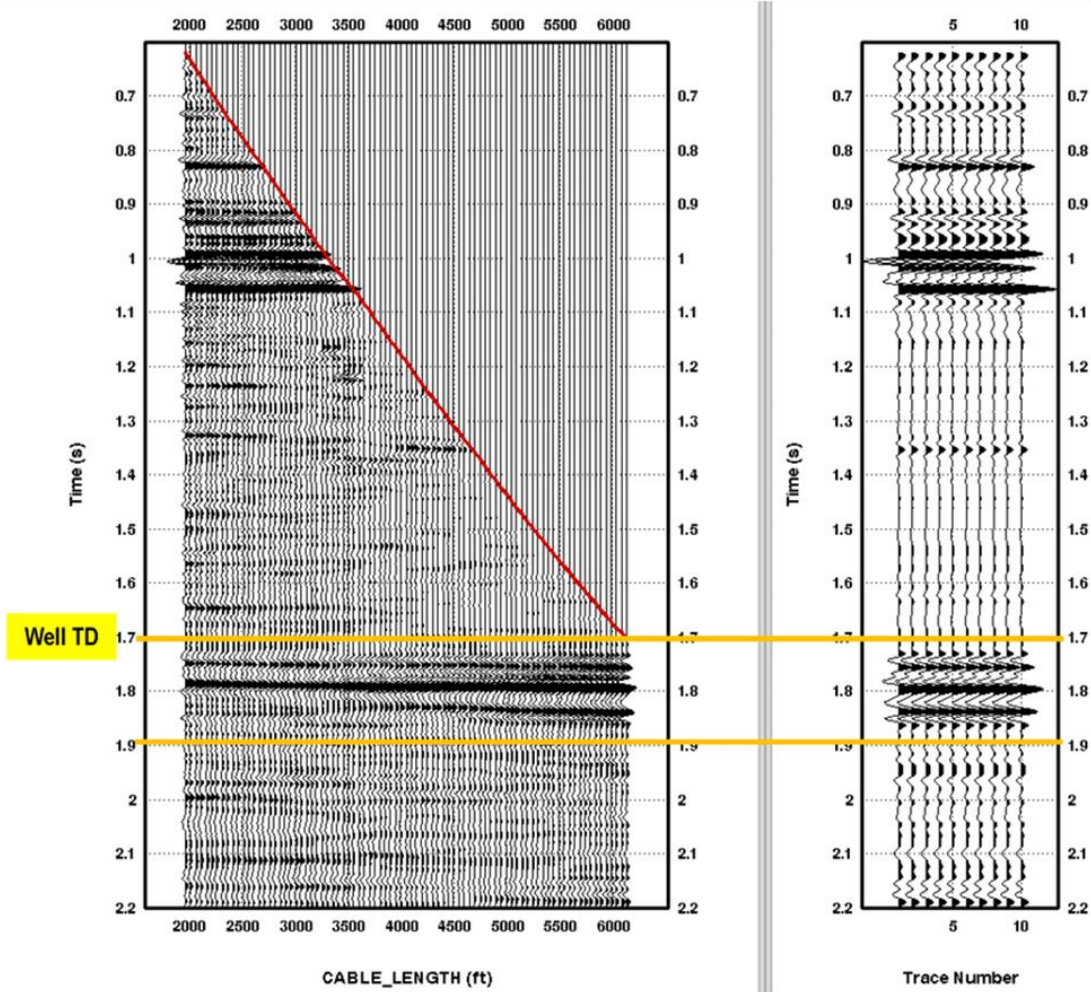
surrounding the well. This 3D dataset can be used for a multitude of purposes including everything covered above in addition to 3D high resolution imaging of the subsurface (Burch et al., 2010). 4D VSP adds the dimension of time into the equation in which multiple successive 3D VSPs are acquired usually separated by period of months to years in between surveys. These surveys are mainly acquired to perform timelapse analysis in order to monitor field performance (Mateeva et al., 2017), or the plume development in the case of CO<sub>2</sub> monitoring (Pevzner et al., 2021). While not explicitly mentioned above all of the above geometries can accommodate both P- and S- sources (converted waves as well) because typically it is standard to use 3-components (3C) receivers in VSP applications which allows for the recording of the entirety of the seismic wavefield.



**Figure 1:** Schematic of the most commonly used VSP geometries (BJV, 2022).

## Upgoing Wavefield After Deconvolution

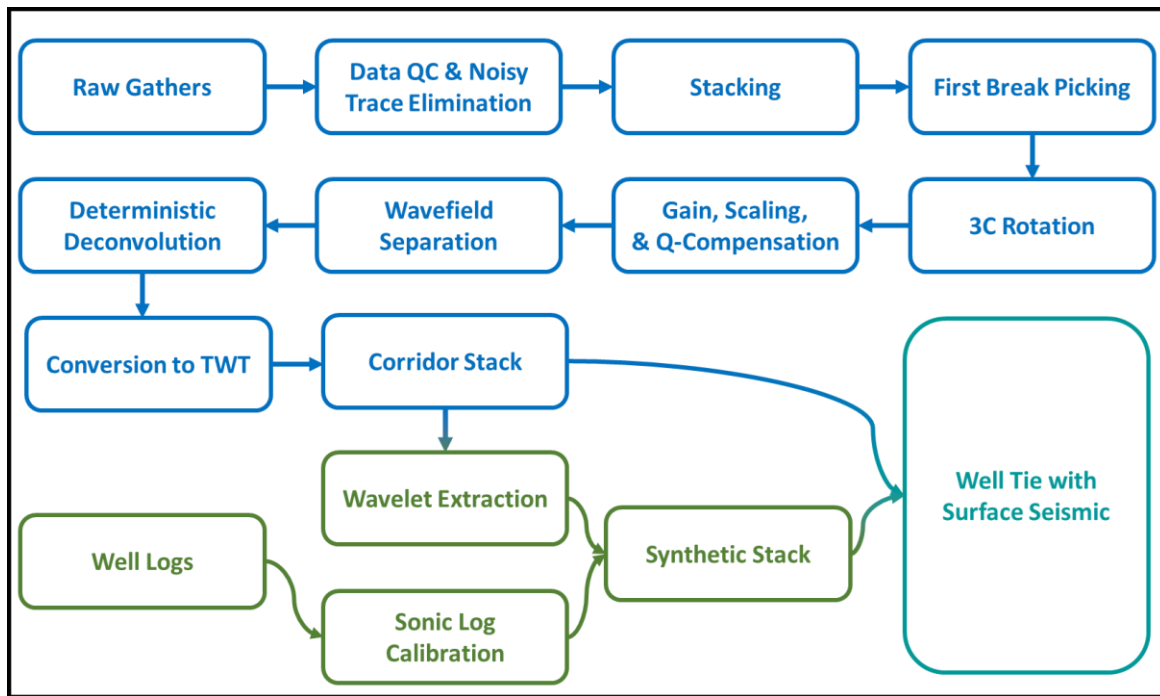
## Corridor Stack



**Figure 2:** A fully processed deconvolved upgoing wavefield from a zero-offset VSP on the left and the resulting corridor stack on the right. The section highlighted in yellow is below the final depth of the well which is why a VSP lookahead inversion is required to extract velocity information from that section (Daoud & Abd El Dayem, 2018).

As discussed in the opening statement, the purpose of this work is to invert for Earth properties (P-wave velocity in this case) from corridor stacks but first it is important to understand how these corridor stacks are created. Earlier it was explained that a corridor stack is generated from the upgoing field of a ZVSP dataset. A typical processing workflow of how to process and generate that corridor stack from raw ZVSP gathers is shown in Figure 3. In the next paragraphs, a high-level summary of each step is provided for further details one can refer to Hardage (1984) or Hinds et al. (1996) for a very detailed introduction of ZVSP processing steps (and more).

The processing workflow begins after the completion of the data acquisition, and the first step depends on the type of source used. If a Vibroseis source was used then similar to surface seismic surveys, a cross-correlation between the sweep and the uncorrelated gathers is needed. For impulsive sources (such as airguns), the only correction needed is applying the time-shifts from the hydrophone component that are related to the trigger delays and positioning of the source to the raw gathers. Optionally, one might apply a source de-signature approach for either of the datasets. Next, A quality check of the gathers is done to eliminate noisy traces and then stack the remaining traces at each receiver position. After stacking, only one trace per component should exist at a particular depth level. Next, the picking of direct P-wave arrivals is performed and based on that an interval P-velocity profile covering the entirety of the well can be generated.



**Figure 3:** A typical processing workflow for a zero-offset VSP dataset in blue with synthetic seismogram creation workflow in green. Usually, the primary use of both the corridor stack and the synthetic seismogram is to tie them with surface seismic data.

If one is interested in the analysis of S-waves, or if P-waves direct arrivals are not polarized towards the vertical, a 3C rotation approach can be used. In the second case, a polarization analysis to measure the direction of P-wave polarization using hodogram analysis is needed. Subsequently, 3C rotation based on the rotational matrix can be applied first on the two horizontal components with the vertical component fixed to maximize the P-wave arrivals in one component and minimizing it in the other, thus having a so called radial and transverse components. Finally, a rotation between the vertical and the radial can be applied with the transverse component fixed which will result in a direct component maximizing down-going P-wave energy, and a perpendicular component which should have down-going P-wave minimized.

After that, compensations related to spherical divergence (geometrical spreading) and optionally inelastic attenuation (Q-compensation) are applied on the data. Now, the vertical component (or direct) is isolated from the other two components and is processed separately for P-wavefields. A wavefield separation technique is used to separate these down-going and up-going P-wavefields from each other. Multiple techniques exist but the simplest approach is to flatten the down-going data first (using the picked first breaks) and then apply a median filter (width is usually between 5-13 traces). After isolating the down-going, it is then subtracted from the full wavefield, and the remanent field would be the up-going wavefield plus some noise. To get rid of that noise, another median filter is applied with the remanent data flattened using reverse first break picks to isolate the up-going wavefield.

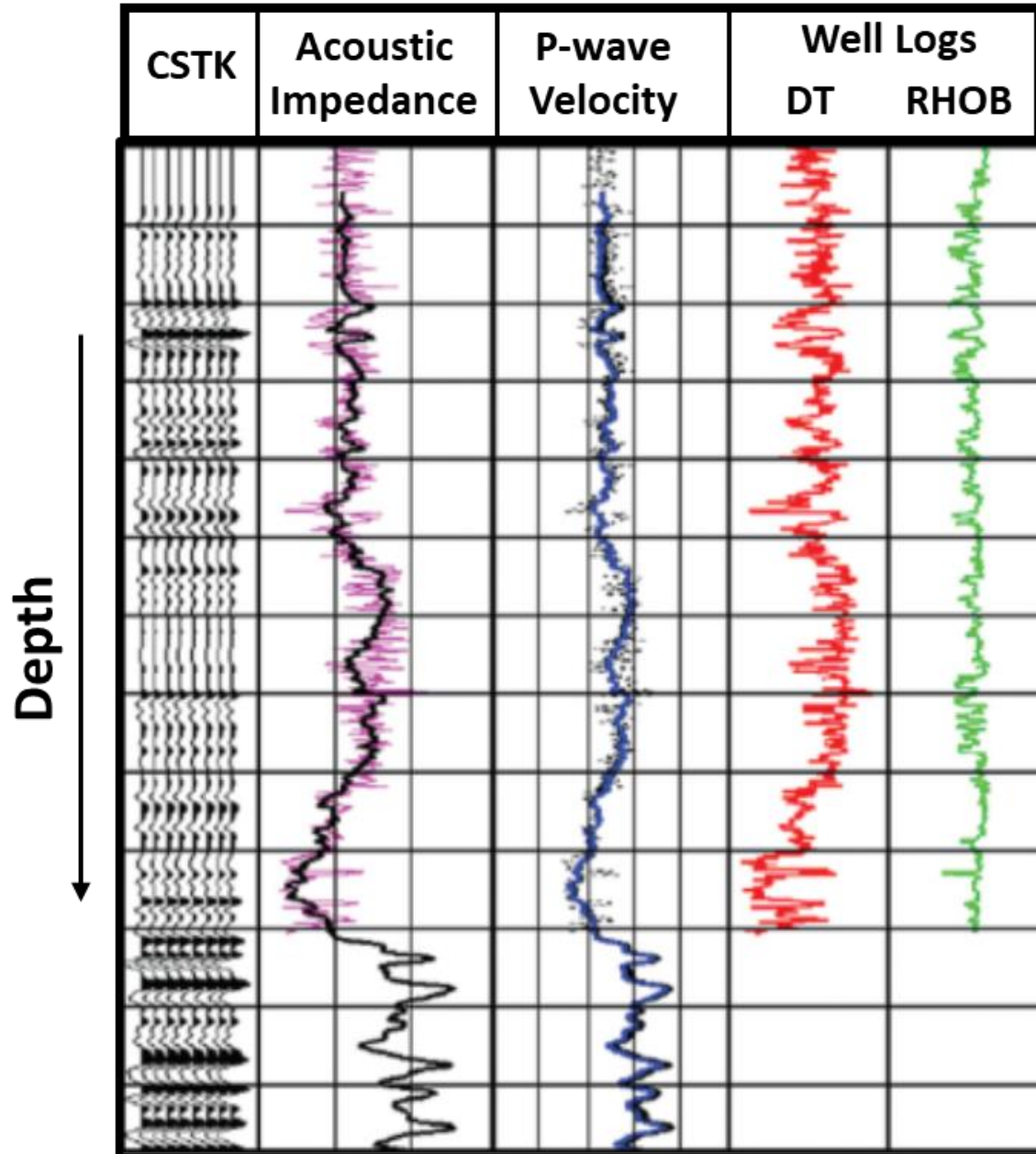
The next step is to create a deconvolutional operator in order to sharpen the wavelet, remove multiples, and correct the phase. In surface seismic, this is done via predictive or

statistical methods. However, for VSP data, a deterministic approach can be taken since the down-going wavefield is recorded and isolated. Thus, an operator can be created for each trace from this wavefield. Once created, a deterministic deconvolution can be applied to the down-going wavefield to check the stability of the operator and if stable applied to the up-going wavefield as well.

To match the travel-time of surface seismic, this deconvolved up-going wavefield is converted from one-way time (OWT) to two-way time (TWT) by shifting each trace by the value of the first break pick for that trace (assuming symmetric travel path). Finally, a corridor stack is generated by keeping and stacking a small corridor of the TWT up-going wavefield close to the first breaks and muting the rest (usually 60-120 ms of data below the first break picks is kept). This corridor stack is usually very clean and remnant noise after completing the processing sequence is usually random noise due. This noise can affect the fidelity of the reflections in the corridor stack and is usually tackled via relatively light post-processing approaches such as band-pass filtering to only keep the desired frequencies or more aggressively via rank-reduction methods (e.g. Chen & Sacchi 2013; Oboué et al., 2023).

With the creation of the corridor stack, the process of VSP inversion or Lookahead VSP as it is commonly known can begin. The process starts by either creating a synthetic wavelet or statistically extracting one from the corridor stack itself. After that, the corridor stack is deconvolved with the generated wavelet to obtain a reflectivity series. The reflectivity series is then inverted for relative acoustic impedance using a sparse spike inversion approach. A low frequency acoustic impedance model can be created from well logs (sonic and density) if available. Alternatively, the velocity profile from the VSP with a density

profile from a nearby well can be used to generate the model instead. For the section below the well, the density model can be extended by either using priori information or via a suitable Gardener's relation. The low frequency model is then added to the relative acoustic impedance to get the absolute acoustic impedance curve. Using the density profile and the absolute acoustic impedance, the velocity curve can be calculated for the entire well and to some depth below the well (depending on noise level below the well) (see Figure 4). This lookahead inversion process has been established in the late 1980s, and is still being refined to this day. This conventional approach requires a significant amount of intervention to obtain good results as priori information need to be integrated as mentioned earlier and determining the appropriate parameters for this inversion process is non-trivial and can affect the results significantly (See Daoud & Abd El Dayem, 2018, for an example of parameter optimization approach). One of the earliest papers containing the process of VSP inversion is by Stewart & Disiena (1989) in which they briefly explained the process and showcased an example. Other papers showcasing this process include but not limited to Payne (1994), Yuhong et al. (2010), Chandrasekhar et al. (2013), Yang et al. (2013), Daoud & Abd El Dayem (2018), and Almarzooq (2022). Further details of the Lookahead VSP process and its application can be obtained from Hardage (1994).



**Figure 4:** A VSP corridor stack (CSTK) and the resulting acoustic impedance inversion (black curve) overlaid over log acoustic impedance (purple) in addition to p-wave velocity inversion (blue) overlaid over log velocities (dashed black line). For reference the sonic log (red) and density (green) are shown on the right (Chandrasekhar et al., 2013).

### 1.3.2 Deep Learning

Now that the VSP has been introduced and explained, it is important to examine the other side of this research and provide a literature review of the important concepts that would be used to perform the VSP inversion.

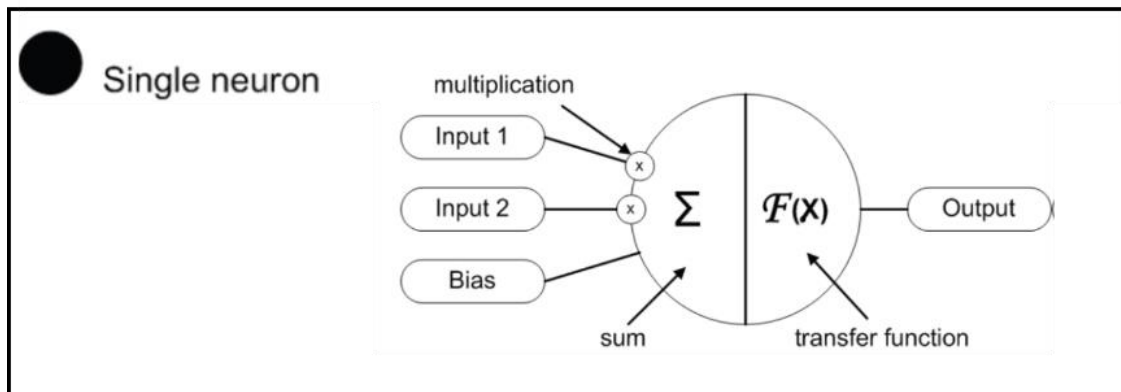
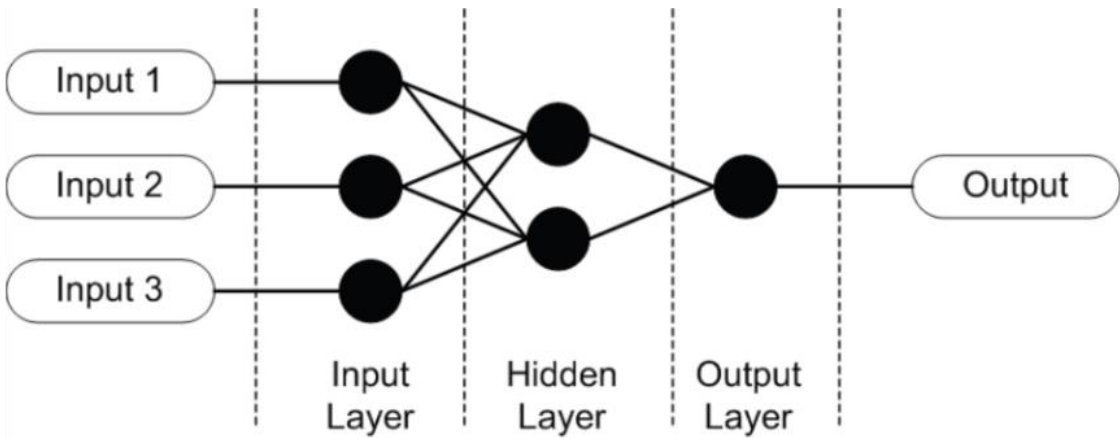
Before that however a brief overview of machine learning will be attempted. The publication most credited as the birthplace of machine learning as a concept is the paper by McCulloch & Pitts (1943). After them, Alan Turing in 1950 came up with the test to determine whether a machine is intelligent or not, the so called ‘Turing Test’. One of the earliest examples of ‘machine learning’, was in 1952 when Samuel created a program that played checkers and improved the more times it played (Samuel, 1967). Research and interest in machine learning went through several cycles of continued development followed by disillusionment. The most recent cycle of renewed interest started at around 2006 – 2012 and still in force today (in 2023) due to the success of deep neural networks (DNNs), spearheaded by the several publications and success of Geoffrey Hinton and others (e.g., Hinton et al., 2006a; Hinton & Salakhutdinov, 2006; Hinton et al., 2006b). DNNs are an example of supervised machine learning, there exists two other types of machine learning; unsupervised and reinforcement learning but the focus of this paper will be on the supervised approach and thus the rest of this literature review will only focus on it.

However, before diving deep into DNNs and other neural networks and to show the importance of the development of DNNs their building blocks need to be explained briefly. Each layer of a neural network (deep or shallow) comprises a number of neurons and

biases as well as the activation functions (see Figure 5). The neurons are responsible for storing the weights of the model while bias term as the name implies is responsible for the constant added to the product of the model weights and inputs. The activation function on the other hand is the function that defines what the output of each neuron would be given a certain input. Most commonly used activation functions are Linear, Sigmoid, Tanh, Relu, and SoftMax functions. What determines which function is to be used is the type of the problem the neural network is trying to solve. For example, a SoftMax function is well-suited for classification problems while a linear function can be used as the final activation function for a regression problem. Nonetheless, Sibi et al. (2013) have shown that while an appropriate activation function choice is important, eventually even sub-optimal functions will converge in most cases. A shallow network as the name implies might contain only one or two layers compared to a deep neural network which will contain significantly more layers. All layers between the input and output layers are usually called hidden layers. The process of calculating the gradients (derivatives) of the model biases and weights is called backpropagation in which the derivatives of objective function error is calculated from the output towards the input (hence the term backpropagation) (see Hecht-nielsen, 1992). Then, the process of updating the parameters based on these gradients is usually done through a gradient descent. Further details on the basics of machine learning and implementation can be found in Rivas (2020) introductory book on deep learning or Suzuki (2011) first three chapters.

Next, a brief discussion on relevant neural networks will be done starting with DNNs. This discussion will include commonly used networks such as long short-term memory networks (LSTMs) and convolutional neural networks (CNNs) but will also dive into

recently devised ones such as temporal neural networks (TCNs). It will also touch on CNN-LSTM architecture which as the name implies combines CNNs and LSTMs in one neural network. Finally, this section will finish on a certain configuration that can be applied to any neural network type called autoencoders.



**Figure 5:** A simple schematic of a neural network containing one hidden layer, each neuron within this network is characterized by the components at the bottom (Suzuki, 2011).

### **1.3.2A Deep Neural Networks (DNNs)**

Deep neural networks were one of the earliest types of layers available in a neural network. Simply put, a DNNs is comprised of any number of layers, each containing a number of neurons (often called dense layers). All of these neurons would be connected to every other neuron in the previous and next layers. These types of networks perform exceptionally well when the relationship (even if non-linear) is clearly present between the input (features) and the outputs. Usually, inputs being fed into DNNs require careful selection of features via feature extraction and enhancement methods to improve the performance of said networks. This type of networks has been applied to all kind of regression and classification problems. Examples from within the geoscientific community include Uncuoglu et al. (2008) for stress prediction as well as Weinzierl & Wiese (2021) for pressure and saturation prediction.

### **1.3.2B Long Short-Term Memory Networks (LSTMs)**

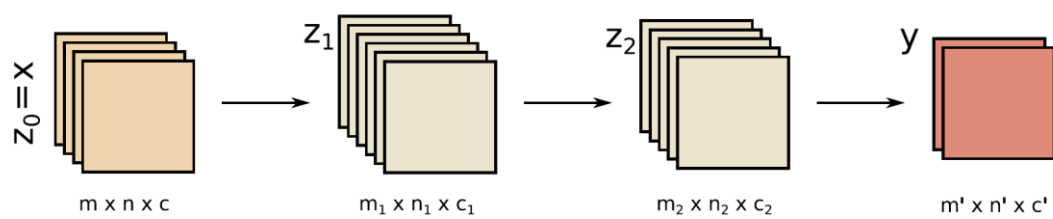
LSTMs are a type of recurrent neural networks (RNNs) which excel at tasks related to time series prediction due to their ability to store and use temporal information. Other types of RNNs usually suffer from difficulties in training due to vanishing or exploding gradient with the exception of LSTMs, which seems to be more resilient. For this reason, the primary type of RNNs used nowadays are LSTM networks. For more details on RNNs and LSTMs as well as a detailed comparison and benchmarking, one can read Graves (2012) book regarding these networks.

### **1.3.2C Convolutional Neural Networks (CNNs)**

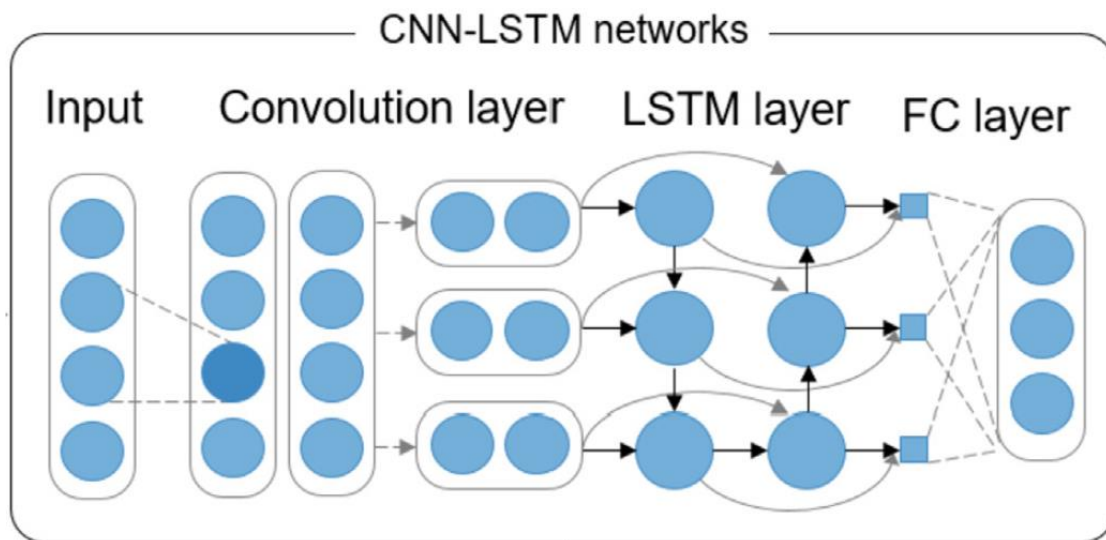
CNNs main advantage is the addition of filters which have their own weights to be updated (see Figure 6 for a simple schematic). The purpose of these filters is to automate the process of feature extraction and include it within the neural network architecture. Often, CNNs will be paired with either dense layers, pooling layers, or both. CNNs are probably the most popular choice especially for tasks regarding image analysis and classification such as can be seen in Pelt & Sethian (2017). It has also been used in natural language processing (NLP) applications like that performed by Collobert & Weston (2008) as well as for seismic data interpolation (e.g., Wang et al., 2019). The versatility and simplicity of application of this type of network is what attracted a significant amount of research towards it.

### **1.3.2D CNN-LSTM Networks**

As the name implies, CNN-LSTM networks are a combination of CNN and LSTM networks (see Figure 7). The primary motivation of this configuration is to attempt to combine the automated spatial feature extraction of CNNs with the temporal awareness of LSTMs. One of the earliest examples is by Sainath et al. (2015) in which they combined the two in order to do NLP related tasks. Further examples of CNN-LSTM include; Wang et al. (2016) and their sentiment analysis work, Kim & Cho (2019) in which they predicted residential energy consumption, and finally earthquake location prediction done by Kail et al. (2022).



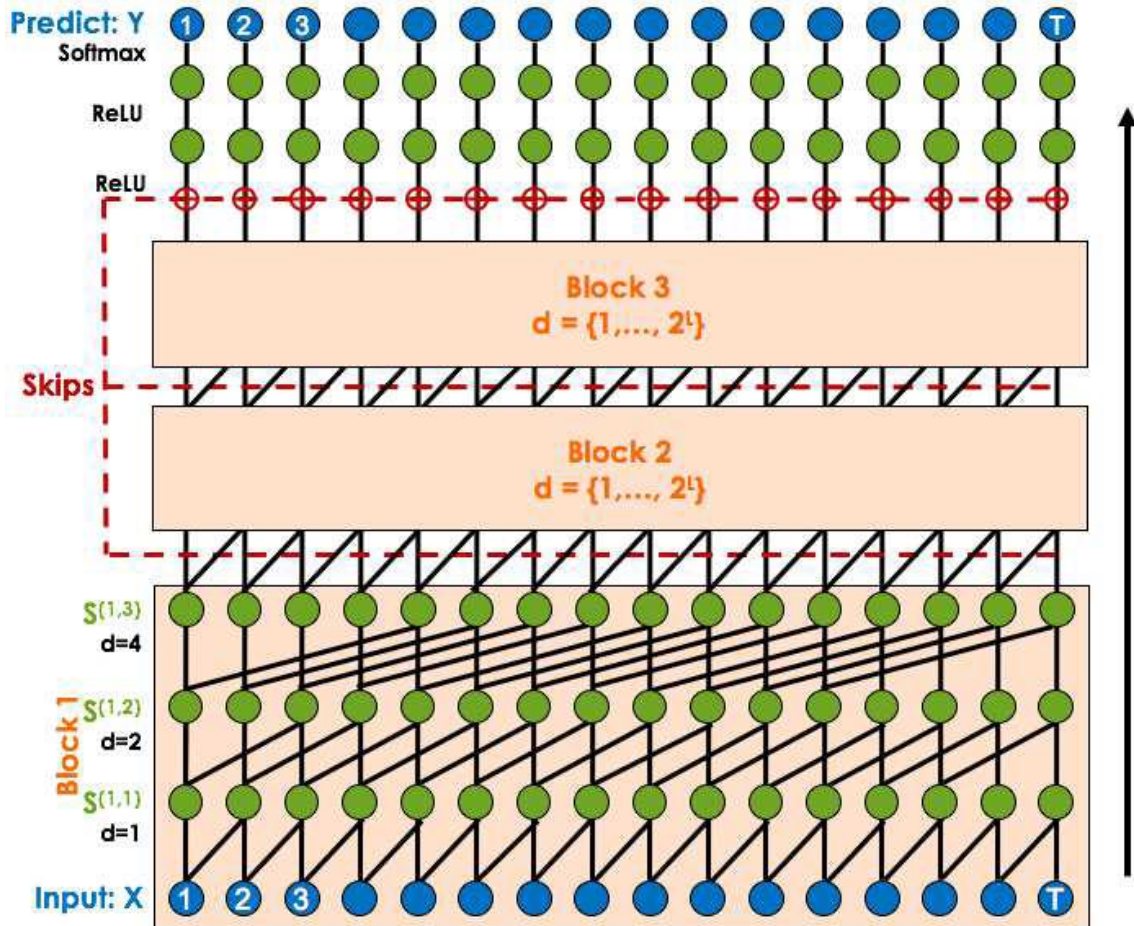
**Figure 6:** A schematic representation of a simple two-layer CNN (Pelt & Sethian, 2017).



**Figure 7:** A schematic showing a CNN-LSTM network. The layers are organized as follows; convolutional layers, LSTM layers, and finally dense (FC) layers (Kim & Cho, 2019).

### **1.3.2E Temporal Neural Networks (TCNs)**

The idea of combining spatial information from CNNs and temporal information from RNNs proved to be quite useful for many applications. Thus, Lea et al. (2016 & 2017) proposed a new neural network type they called temporal neural networks (TCNs) which uses spatial and temporal information like CNN-LSTM networks but does so simultaneously instead of iterative via a process called dilatational convolution which determines the steps between each filter in time (see Figure 8 for a schematic). This design as implied has the advantage of having both the convolutional and temporal parts working together instead of separately as in the case of CNN-LSTM architecture. Additionally, TCNs are formulated in such a way that only information from current and previous time steps contribute to the convolutional layer at that current time step with no information leakage from future time steps (Zhao et al., 2019). Rather than performing the computations on the entire time steps, the computations are performed on a dilated representation of the time steps, which reduces the computational load significantly. Additionally, these dilations allow the network to sample further time steps without sampling the entire timeframe thus allowing for a wider temporal perception field more efficiently. Besides Lea et al. (2016 & 2017) in which TCNs were used for action segmentation and detection. TCNs were also used by Zhao et al. (2019) for short-term traffic forecasting, He & Zhao (2019) for anomaly detection, and Hewage et al. (2020) for weather forecasting.



**Figure 8:** A schematic of a TCN network in which the convolutional block contains three layers of dilated convolutional layers (Lea et al., 2017).

### **1.3.2F Autoencoders**

Encoding and decoding data usually refer to the act of compressing data by discarding unnecessary details and retaining important ones while the act of decoding is using these retained features to reconstruct the encoded data as accurately as possible. The process of training a neural network to encode and decode is usually referred to as autoencoding and probably the earliest example is by Hinton & Salakhutdinov (2006). The only requirement for an autoencoder architecture is the asymmetry between the encoder part and the decoder part. An autoencoder architecture can be of any depth but is usually shallow and can consist of DNNs, CNNs, or TCNs as layers for the autoencoder depending on the problem at hand. Several examples exist related to seismic data including Saad & Chen (2020) which used it for denoising seismic data, and Wang et al. (2020) in which they interpolated seismic data via convolutional autoencoders.

### **1.3.3 The Role of Machine Learning in VSP Inversion**

This section will attempt to answer the question of whether it is appropriate to use machine learning and more specifically deep learning for VSP inversion and why it has been chosen. As discussed in the opening statement, the standard methodology makes certain assumptions at every step of the inversion process. Firstly, a fundamental assumption regarding the sparsity of reflectivity in the Earth is made which is essential for inverting the corridor stack into acoustic impedance. Another important assumption is that the wavelet is time-invariant, which assumes that the effect of attenuation is negligible or has

been taken care of during processing. Thirdly, the density below the well's final depth is often not known and to calculate one, usually one assumes that the lithology follows Gardener's relation between velocity and density, which is not always the case especially when halite is expected. These three assumptions have the most impact on the inversion process, but are not the only ones present. One could make adjustments to counteract each point to a satisfactory degree if data is abundant and clean. However, the combined effort of such task often makes it infeasible to address all assumptions and their effect on the data. The application of deep learning was considered because if the model architecture is appropriate and the model parameters are large enough, then it could – in theory – be able to learn and counteract these complications implicitly. It could also potentially deal with the non-linearity and non-uniqueness aspects of the problems better than the traditional approach. Furthermore, the issue of parameterizing the inversion process is often non-trivial and moving the parameterization to the deep learning model might offer a better prediction if the training and validation is representative of real datasets. This, however, will require a large enough training dataset with clean labels that is diverse enough to combat these two aspects.

## **1.4 Aims and Objectives**

As alluded to above, the primary aim of this thesis is to invert VSP corridor stack data (essentially a seismic trace) and obtain a velocity profile encompassing the entire length of this input data using a series of deep learning models. To achieve this, the following will be accomplished:

1. A realistic 1D Earth model synthetic data generation engine will be created. This engine should be capable of generating and outputting all relevant data such as velocity, density, and acoustic impedance profiles as well as reflectivity series, convolved reflectivity that is clean or noisy. Additionally, this engine should be also capable of generating models and their corresponding datasets rapidly and on-demand.
2. Pre-processing of this synthetic data (and real data) to prepare it for input in the deep learning models.
3. Create and optimize the deep learning architectures for each sub-task leading up to the inversion of VSP data for velocity.
4. Blind testing of the models on data generated via the synthetic data generation engine and on a finite difference model processed via a typical VSP processing workflow.

## **CHAPTER 2**

### **METHODOLOGY**

In this section, the primary steps needed to accomplish the VSP inversion process will be discussed in the following order; synthetic data generation, data pre-processing, and the role of neural networks in the inversion process.

#### **2.1 Synthetic Data Generation**

Traditional approaches for seismic inversion including VSP lookahead inversion rely on the quality of the estimated wavelet as well as the data for the inverted reflectivity series. Contrary to traditional approaches of seismic inversion, the most significant aspect of a successful deep learning model is the quality of the input and output training and validation datasets used to calibrate its weights and hyperparameters.

Ideally, real data should be used when training neural networks due to the fact that the primary use of the network post-training would be on real datasets. Unfortunately, the quantity of available data for any given problem is often orders of magnitude smaller than the amount needed to create an unbiased and generalized model. Secondly, checking the quality of the inputs and outputs for real data is often impractical unless if one is able to heavily automate it. Finally, there is uncertainty when it comes to the outputs (labels) of real data. It is typically the case that the model is predicting an unknown quantity or

property so the reliability of labels from real data is always difficult to determine. For these reasons, this paper will attempt to generate systemic and high-quality inputs and outputs synthetically rather than rely on uncertain real data for training.

The first step in the generation of any synthetic data is understanding what it is composed of and how to recreate it. A seismic trace (the input) can be thought of as a simple convolutional model. In other words, a seismic trace is a reflectivity series convolved with a source wavelet plus some noise as described by the Equation 2.1 below:

$$u(t) = e(t) * w(t) + n(t) \quad (2.1)$$

Where  $u$  is the seismic trace,  $e$  is the Earth's response (reflectivity),  $w$  is the source wavelet, and  $n$  is the random noise (Yilmaz, 2001).

So, in order to recreate this seismic trace, an engine is needed to create plausible models of the subsurface and another is needed to generate common source wavelets to convolve with. The noise can be thought of as random and thus can be generated by adding white noise to the final dataset. The Earth model building portion is accomplished by starting with generalized geological sequences; clastic, carbonate, or salt sequences, each assigned to be generated in proportion to how common each sequence is. Next, a random number of layers for each sequence is chosen, with variable lithology and thickness. For example, a clastic sequence is mainly composed of sands and shales, and thus each layer within that sequence will be assigned as one or the other (see Figure 9 for an example model). In order to create a reflectivity series out of this geological model, velocity and density values need to be assigned to each layer according to its lithology (see Table 1), and depth due to compaction. Once these values are assigned, small inter-layer variations are introduced to

simulate small geological variations within each layer. With the introduction of geological velocity and density layers, acoustic impedance can be calculated, thus a reflectivity series can be created.

**Table 1:** Shows the constants for the empirical relationship between density and velocity for different lithologies (Mavko et al., 2020) following the Equation:

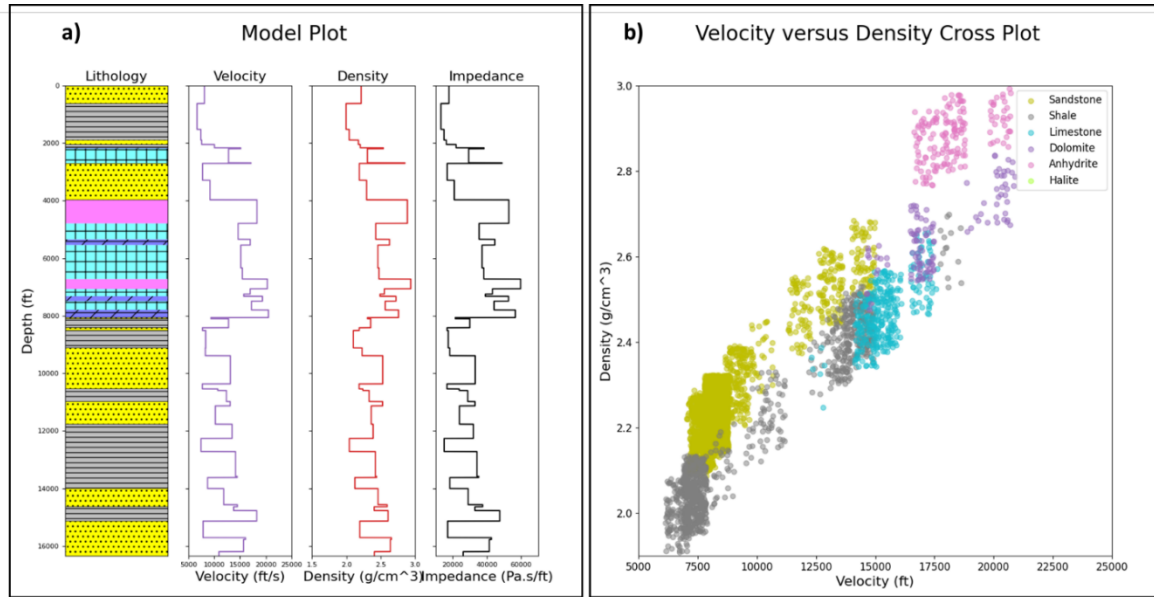
$$\rho = av^2 + bv + c \quad (2.2)$$

| Lithology | a       | b      | c      |
|-----------|---------|--------|--------|
| Shale     | -0.0261 | 0.3730 | 1.4580 |
| Sandstone | -0.0115 | 0.2610 | 1.5150 |
| Limestone | -0.0296 | 0.4610 | 0.9630 |
| Dolomite  | -0.0235 | 0.3900 | 1.2420 |
| Anhydrite | -0.0203 | 0.3210 | 1.7320 |
| Halite    | -0.0455 | 0.5892 | 0.4358 |

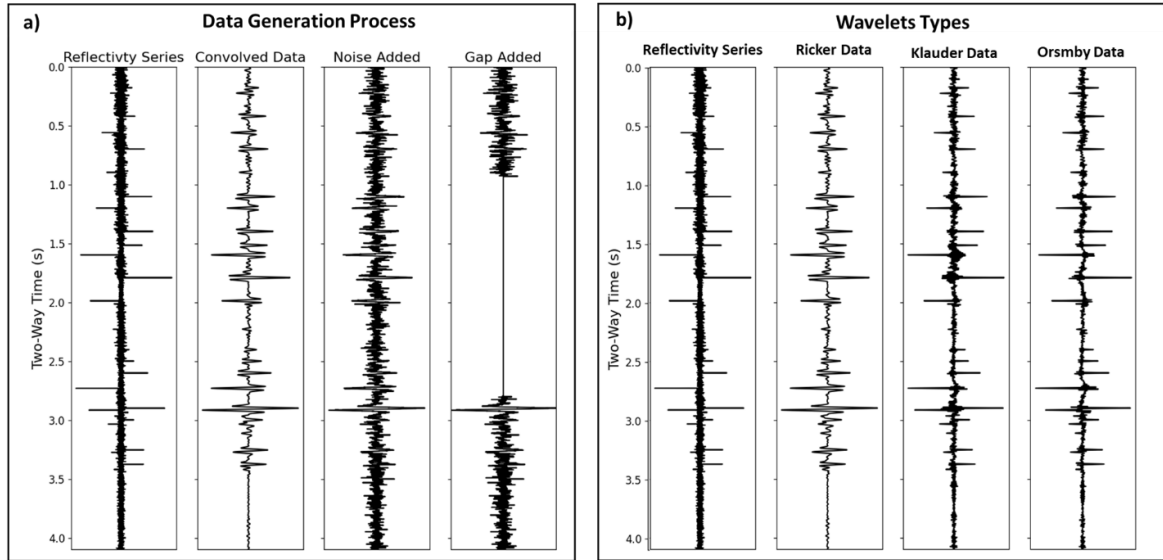
The second component is the source wavelet, and to ensure its diversity, three wavelets have been chosen to represent the variety of sources available as follows; Klauder, Ormsby, and Ricker wavelets. The range of frequencies for each wavelet is varied to have a diverse dataset in terms of available frequencies. Each reflectivity series is then convolved with a random wavelet and then white noise is added to the convolved data (see Figure 10) thus creating the inputs and outputs to be used in the neural network.

With all components necessary for the generation of synthetic traces ready, the creation of a custom generation engine to generate these traces and their corresponding models on the fly is completed. This will ensure that an unlimited number of datasets are available, which will reduce bias in the neural network compared to a limited number of datasets. All networks trained in the Results chapter are using this aforementioned engine to generate datasets. Each epoch for models used in grid search related HPO contains 1000 unique

datasets with 4096 samples each. For Optuna HPO (an optimization framework that will be discussed in Section 2.3 and the Results Chapter) related models, the number of datasets per epoch is reduced to 250 in order to have a larger number of tests in the Optuna study. The reason for this difference is that the tests for Optuna studies are selected automatically compared to manually for the grid search method. For the final models of each section, we revert back to 1000 datasets per epoch. For validation datasets, additional unique datasets equal to a third of the training datasets are used. Finally, 1000 datasets will be used for blind testing to establish which model performs better. If a definitive answer cannot be established with 1000 datasets and if the compared models have very similar root mean square errors (RMSE) values, then the number of datasets will be increased until a clearer picture of which model performs better can be established.



**Figure 9:** An example of a model generated using the data generation engine proposed here showing the lithology, velocity, density, and the acoustic impedance on the left. Additionally, a cross-plot of velocities versus densities is shown on the right and is colored by lithology.



**Figure 10:** a) The process of generating the different data based on the reflectivity series (created from model acoustic impedance). The data is convolved with a wavelet, noise is then added, and finally a random gap is added to both the data and velocity profile to stimulate missing data. b) Example of different wavelets that can be convolved with the data (Ricker, Klauder, and Ormsby).

## 2.2 Data Pre-Processing

With the synthetic data generated, this section aims to discuss the possible pre-processing steps that are required prior to deep learning and other steps that could be beneficial to the learning and prediction process. The first section will discuss the ‘required’ steps while the next two sections will discuss potential pre-processing steps that might improve a portion of the deep learning process.

### 2.2.1 Standard Pre-Processing Steps

Since the datasets are synthetically generated only minimal pre-processing is needed. Namely, the datasets should be normalized. It is not necessary to normalize the data to get a reasonable result out of a deep learning algorithm, but it is often recommended especially in the case of multiple inputs (Jayalakshmi & Santhakumaran, 2011) or significantly varying amplitudes for the inputs especially in the case of real data where special care is needed to balance the amplitudes prior to feeding it into the neural network as will be discussed in section 2.2.3. For this reason and since the range of values is known for both reflectivity-based input, including convolved and noisy data, as well as the velocity profile. For reflectivity-based inputs, the range is between -1 and 1 by the definition of reflectivity (see Equation 2.3 two below).

$$R_c = \frac{\rho_2 v_2 - \rho_1 v_1}{\rho_2 v_2 + \rho_1 v_1} \quad (2.3)$$

Where  $R_c$  is the reflection coefficient,  $\rho$  is density, and  $v$  is velocity (Mavko et al., 2020).

Outputs are generally not normalized but in the case of multiple outputs with significant difference in range, as is the case for the velocity and density, it seems reasonable to normalize them. For the velocity profile, the range is roughly between 5000 ft/s and 25000 ft/s based on observable ranges (Mavko et al., 2020). To cover all physical values, one might extend the range between 0 ft/s and 30000 ft/s for normalization purposes. Similarly for density, physical values encountered in the subsurface range between 1.5 g/cm<sup>3</sup> to 3.0 g/cm<sup>3</sup> (Mavko et al., 2020) and this can be extended to 1.0 g/cm<sup>3</sup> and 3.5 g/cm<sup>3</sup> for normalization in order to ensure that outliers fit within the range.

With the ranges known for both inputs and outputs, the simplest approach for normalizing the data is the min-max normalization described by Equation 2.4 for any arbitrary property  $x$  as follows:

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)} \quad (2.4)$$

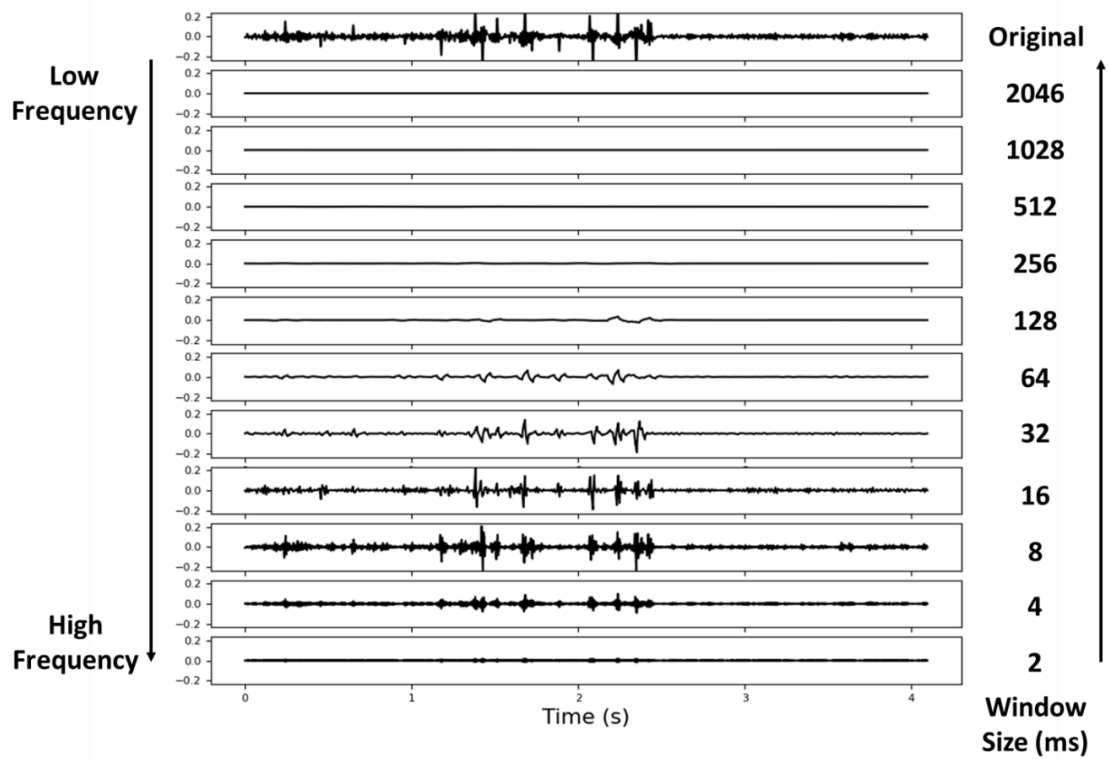
It is trivial to reverse the process once a prediction is made as the minimum and maximum ranges are known and described above. As for standardization, it most likely should not be performed on these datasets because it should not necessarily follow a standard distribution. If it is forced to follow such distribution via standardization, especially in the case of velocity and density, variations in lithology might be masked or obscured.

### 2.2.2 Wavelet Transforms

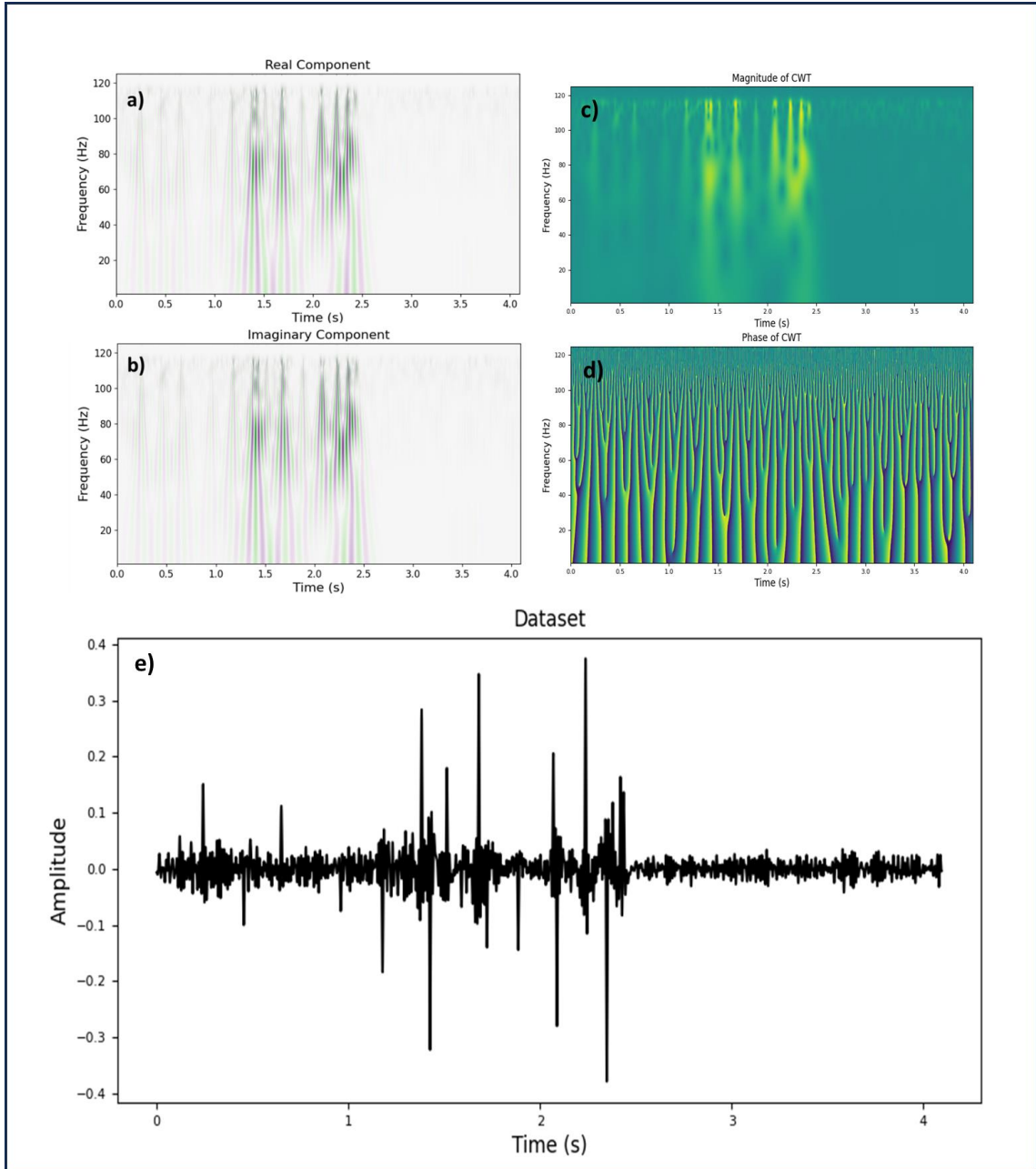
In this section, a brief explanation of what wavelet transforms are and how they might be beneficial to the inversion of VSP data via neural networks. A wavelet transform is a methodology to transform or analyse data (including seismic) that have features that span various wavelengths (Rioul & Duhamel, 1992). In principle, all wavelet transforms work similarly, first a wavelet is chosen. This wavelet is then applied on the original data through windows that cover the entire dataset. The wavelet is shifted, scaled, and then the process is repeated over a specified range of window sizes. The short windows and wavelength wavelets sample high frequencies while long ones sample the low frequencies (see Zhang (2019) for a more complete explanation). Three main wavelet transforms are of specific interest for our signal analysis purposes; the continuous wavelet transform (CWT), stationary wavelet transform (SWT), and discrete wavelet transform (DWT). The CWT as the name implies transforms the data into a continuous time and frequency plot while the DWT and SWT sample only specific pairs of window sizes and frequencies (approximately each subsequent trace doubles the window size and halves the frequency content). The primary difference between DWT and SWT is that the former down-samples the data for the larger windows while the SWT keeps the number of samples the same for all (Zhou et al., 2006). For easier application within neural networks, it is desirable to have the different decomposed signals to be of the same length. Thus, only CWT and SWT will be considered in this work.

Several examples from literature showcase the benefit of CWT as an analysis or processing step for traditional applications of seismic processing (e.g., Sinha et al. 2005; Diallo al.,

2006). As well as its use for seismic in combination with classification deep learning applications (e.g., Zhang et al. 2018). Due to this wide applicability in both the wider field of signal analysis and the field of seismic data analysis, the effect of applying CWT on the deep learning process proposed here should be quantified. Additionally, if there is an improvement with the use of CWT and since SWT is computationally less expensive to run, the same process should be applied with SWT to check whether similar results can be achieved more efficiently (see Figure 11 and Figure 12 for examples of SWT and CWT application on data generated with the process explained in Section 2.1).



**Figure 11:** A stationary wavelet transform (SWT) performed on a sample data (Original). It shows how different windows capture different frequency content. In this case, most are concentrated within 2-128 ms window size.



**Figure 12:** An example of continuous wavelet transform performed on a sample data (e). The CWT is composed of real (a) and imaginary (b) components. Alternatively, it can be transformed into magnitude (c) and phase (d) components.

### 2.2.3 Handling of Real Datasets

As discussed in section 2.1, the neural network will be trained on synthetically generated data and then some minor pre-processing steps will be applied before training the model. Ultimately however, it is necessary to formulate a methodology to adapt the network to be able to make predictions on real datasets.

First discrepancy and probably the most significant between real and synthetic data is the issue of the amplitude scale. Real data can be recorded via a variety of tools (e.g., geophones or accelerometers), each with their own measurement that will vary from each other and most importantly from synthetic data.

Additionally, appropriate gain control in the field in one region can be different in another region. To some extent, these amplitude and magnitude issues are taken care of during processing but not completely. Thus, an approach to scale the data is suggested as follows. Usually well logs (such as sonic and density) are acquired at least in a portion of the well in addition to VSP data. From these well logs as shown in Figure 3 in Section 1.3.1, a synthetic seismogram not too dissimilar to the synthetics used for training can be generated. Then, a scaling factor can be obtained by analyzing the corridor stack and the synthetic seismogram in the section where both exist.

Furthermore, a cross correlation between the synthetics and corridor stacks can be calculated and if correlation is high that will suggest that this scaling factor could be reliable. If that is the case, then this scaling factor can be applied to the corridor stack prior to feeding it into the neural network for prediction.

If that doesn't work, the neural network architecture could potentially be modified and retrained to allow for additional priori information as input to give more predicative power to the neural network and reduce the effect of non-uniqueness. This priori information would come in the form of the measured velocity profile in the drilled section (as performed traditionally for VSP lookahead inversion). Thus, the primary task of the neural network would shift from predicting the entire velocity profile to only the velocity profile of the undrilled (and unmeasured) section below the final depth of the well. To simulate real data more closely a section on predictions made on finite difference synthetics will be made to see how the network perform with data not generated through the synthetic data generation engine.

## **2.3 Neural Networks Architectures**

With all pre-requisites of a successful deep learning experiment discussed, it is time to explore one of the most critical aspects of neural networks, its architecture. The history of machine learning in a broad sense has been discussed during the literature review, so here a more focused and pragmatic approach focused on the advantages, disadvantages, and applicability of each architecture introduced in the introduction will be discussed.

A VSP corridor stack or any seismic reflectivity series is in essence a time-series and thus architectures that are suitable for time-series analysis or prediction should be suitable for the corridor stack. This means that the most likely candidates for a proper architecture would be ones that can deal with spatial and temporal aspects of the signal properly. Thus,

the architectures most likely to succeed optimally for the inversion of VSP data are CNN-LSTM and TCN type architectures. For other tasks such as denoising and data reconstruction, CNN, CNN-LSTM, and TCN autoencoders should perform similarly. One might make use of CWT or SWT transforms with autoencoders to perhaps extract features that wouldn't have been easily retrieved by a regular autoencoder. With all of this added complexity and array of tasks at hand, the need to establish a baseline via DNNs, LSTMs, and CNNs is probably wise in order to ensure that this added complexity is actually performing better than simpler models.

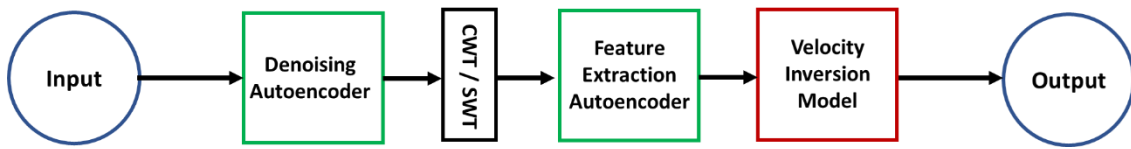
The issue of choosing appropriate network architecture and hyperparameters for each task is quite complex. An entire field of deep learning is dedicated to it and is referred to as hyperparameter optimization (HPO). For each attempted task an HPO analysis via an automated HPO framework called Optuna (Akiba et. al., 2019) will be attempted instead of manual methods of HPO as it allows for more quantitate approach of hyperparameter tuning compared to conventional methods.

## **2.4 Consecutive Neural Networks**

As shown in the previous section, usually each neural network and architecture excels at a specific range of tasks but perform poorly in others. Not to mention that some types of predictions require significantly shallower neural networks than other types. Thus, the idea of consecutive neural networks is formulated to address this (see Figure 13 for a schematic). For example, the problem of VSP inversion for velocity is a multi-faceted one

in which multiple issues need to be tackled. First, any remnant noise in the corridor stack needs to be tackled and as discussed previously – and as will be shown in the next section – autoencoders are robust when it comes to denoising. Even if the data is clean, the transformation from a corridor stack to some properties is more manageable compared to others. For example, the inversion for acoustic impedance is significantly less convoluted compared to velocity or density. All of these properties are related to the corridor stack but the latter two are one additional degree of separation away from the corridor stack compared to the acoustic impedance. Furthermore, with multiple consecutive networks, one can inject an intermediate processing step into the process that could potentially improve the predictions made (e.g., the wavelet transforms discussed earlier). Finally, segmenting or approaching the problem in a piece-wise approach will provide more transparency for both the training and the results compared to a single deep and complex network comprising all of the different necessary components.

To be able to quantify whether this proposal of a consecutive neural network is going to provide all of these added benefits, an array of neural networks will be tested and pitted against this consecutive neural network approach.



**Figure 13:** The proposed schematic for the consecutive neural network. It starts with denoising the corridor stack, then applies a wavelet transform which is fed into a second autoencoder (to train and then use the encoder to compress the data), and finally the velocity inversion is applied on the compressed clean signal for the final output.

## **CHAPTER 3**

### **RESULTS**

The results chapter is primarily divided into two sections. The first section deals with the results of the inversion for velocity with the input being the corridor stack trace directly without any transformations. The second section attempts to answer the question of whether applying any wavelet transformation to the input prior to the inversion for velocity will improve the inversion results or not.

#### **3.1 Direct Inversion of The Seismic Trace**

As mentioned earlier, this section will deal with the direct inversion of the corridor stack through the proposed consecutive neural network architecture. The first step of any successful deep learning model building and training is choosing the appropriate hyperparameters for said model. So, the first section will deal with the HPO process as discussed earlier for the denoising and inversion sub-models and the resultant HPO-based final models. Next, two additional refinements will be considered. Firstly, the effect of the denoising sub-model on the results compared against a model that directly inverts the noisy traces instead of denoising them would be considered. Secondly, the effect of a feature extraction and compression autoencoder on the results. This section will then briefly show the potential of training similar models for the inversion of other parameters such as

acoustic impedance or the joint inversion of two properties simultaneously (velocity and density). It will conclude by blind-testing the models on data not generated by the custom synthetic generation engine used for training and validating the model as a final and unbiased blind test.

### **3.1.1 Hyperparameter Optimization**

The proper approach to HPO is to attempt to automatically and efficiently explore the hyperparameter multi-dimensional space. A simplistic approach such as trial & error or 1D/2D grid search to optimize the hyperparameter has been considered initially. However, since, a more quantitatively defined approach can be better justified, the automated approach (Optuna framework) will be performed for fine-tuning the model hyperparameters.

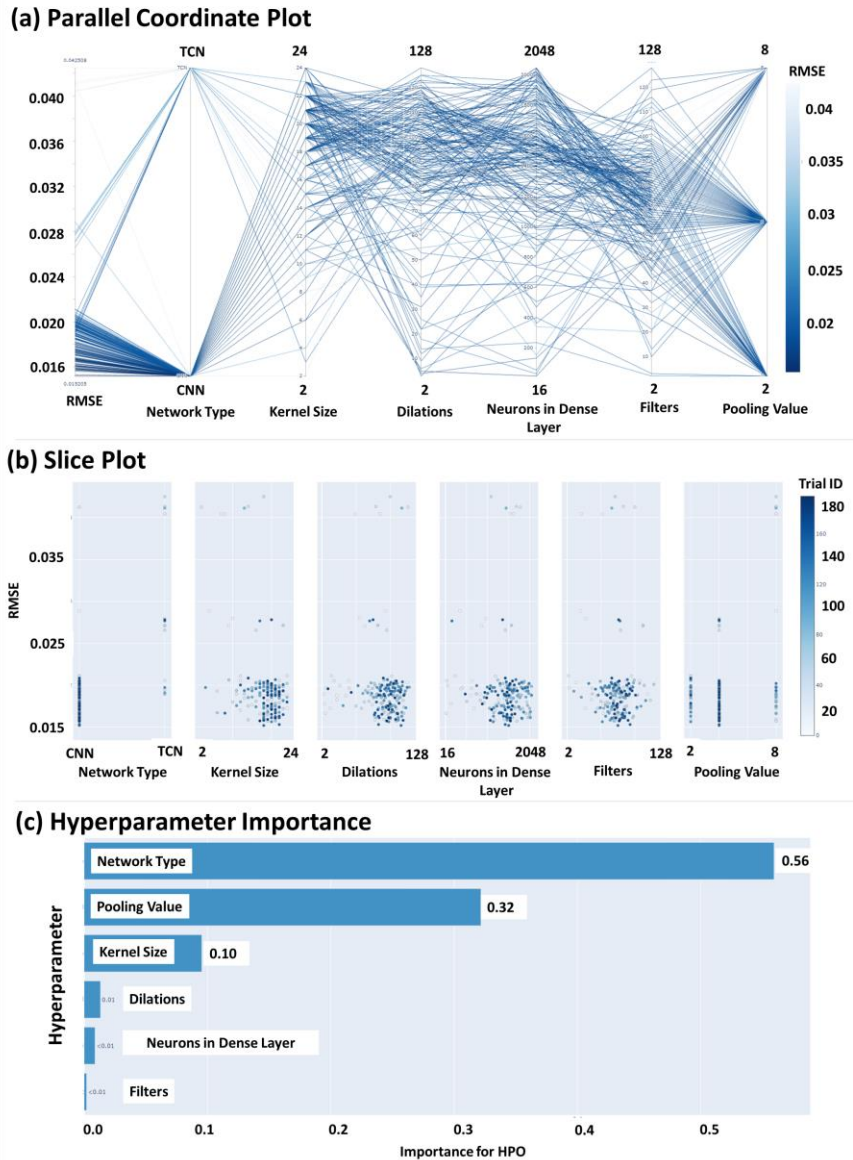
There is a third approach that was explored early in the thesis work related to the use of physics informed neural networks (PINNs) and it is a field that garnered real interest especially for geophysical applications of deep learning as it attempts to bridge the gap between the physics and the deep learning approaches which should yield greater explainability of said deep learning models. This approach however was deemed out-of-scope for this particular problem as establishing a PDE that connects the corridor stack and the velocity profile for the objective function was deemed infeasible. There is a wealth of literature regarding PINNs and the interested reader is referred to the following publications as examples for the use of PINNs in geophysical applications; Song et al. (2021), Waheed et al. (2021), and Alkhalifah et al. (2021).

### **3.1.1A Denoising Models**

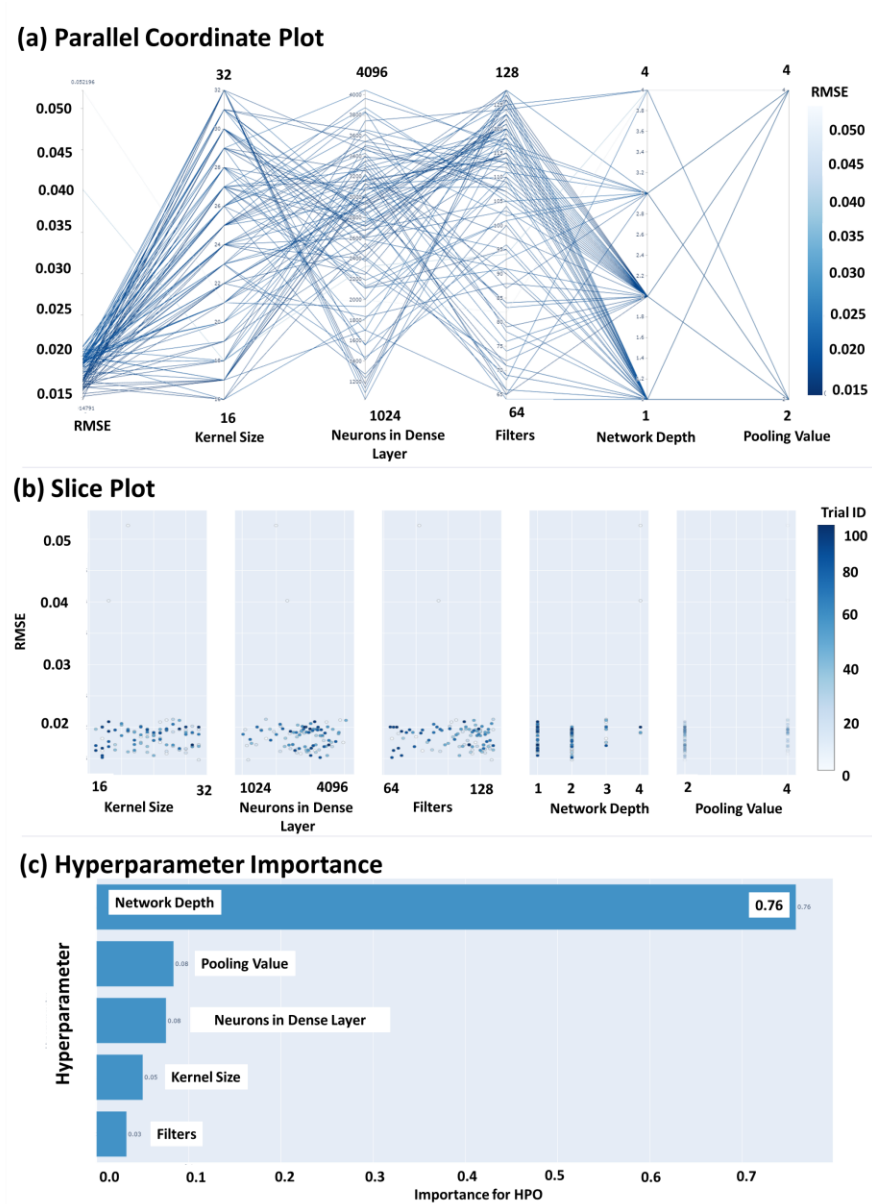
The first model to be optimized is the denoising model. Several architecture options were considered but the most promising seemed to be CNN and TCN architectures. So, two Optuna studies to choose between them and optimize their respective hyperparameters were conducted. The hyperparameters in question are network depth, kernel size, number of filters, number of neurons in the dense layer, and max pooling value for both network configurations and an additional hyperparameter related to dilations for TCN based networks only.

The first study was composed of 191 tests, and it excluded the network depth from optimization. This study was kept at the shallowest configuration as optimizing the network depth is included in the second study because of its expected dominant effect on other hyperparameters. Additionally, this will allow for a faster convergence of the optimal results for a single layer including the network type. The results of this first study showed that the appropriate network type for denoising is a CNN style network with a kernel size of around 18, 64 filters, 1500 neurons in the dense layer, and a max pooling value of 2 or 4 (see Figure 14 for a summary of the results). The second study consisted of 99 tests and was primarily designed to tackle the question of appropriate network depth while also refining the rest of the hyperparameters besides the network type. The results showed that a network depth of 1-2 is appropriate while the kernel size was determined to be optimal as long as it was within the 18-32 range, and similarly for the number of neurons in the dense layer which had a range of acceptable values within 2000-4000 neurons, 124 filters however were determined to be appropriate based on the second study. Finally, the max

pooling value remains unchanged between the first and second study in which either a value of 2 or 4 is acceptable (see Figure 15 for a summary of the second study results).



**Figure 14:** The results of the Optuna study summarized in three automatically generated plots. In (a) a ray plot starting with the objective function (RMSE) value and passing through each optimized hyperparameter value for a given trial is shown. The ray density is indicative of where the optimal values should be. Another view where each hyperparameter is plotted against RMSE is shown in (b). Finally, (c) shows the importance of each hyperparameter relative to improvement in the RMSE loss minimization.



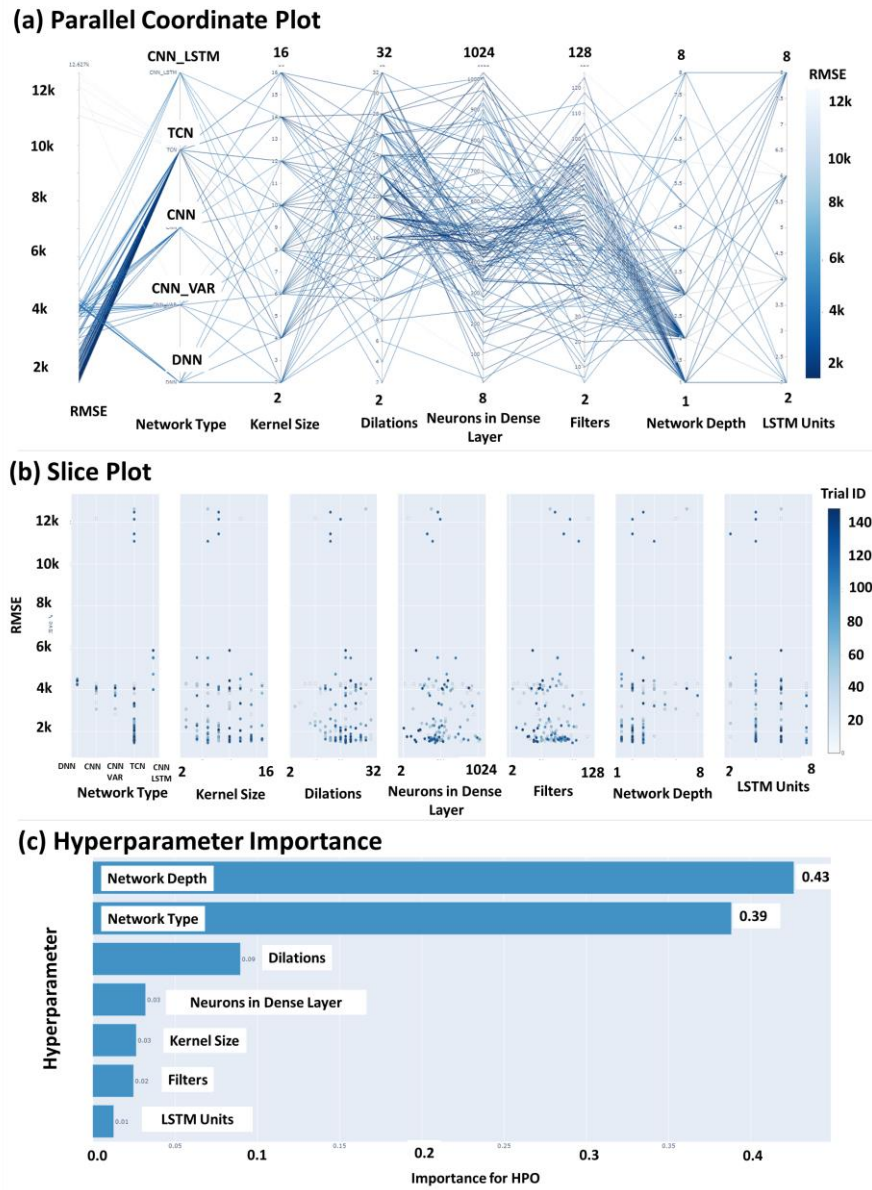
**Figure 15:** Similar to Figure 14, (a) shows the parallel coordinate plot, (b) the slice plot, and (c) the hyperparameter importance. Notice how the most important hyperparameter here is the network depth while the others combined have a marginal effect on the results. Thus, the need for separating it from the first study.

### **3.1.1B Inversion Models**

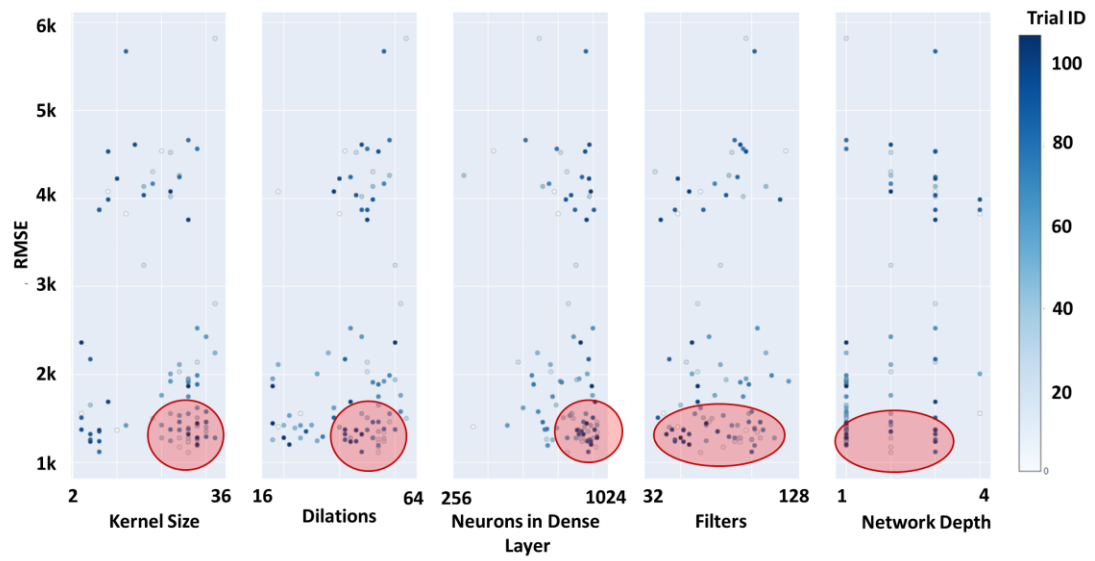
The second model to be optimized is the one responsible for the inversion of the clean seismic trace for the velocity profile. The approach followed here is similar to that in the previous section for the denoising model. Two Optuna studies were conducted here as well with the first focusing on obtaining the optimal network architecture while the second attempted to refine the rest of the hyperparameters.

The first study consisted of 151 tests that sampled several promising (CNN-LSTM & TCN) and baseline network architectures (DNN & CNN) as well as reasonable ranges for the network depth, dilations (TCN), LSTM units (CNN-LSTM), neurons in the dense layer, kernel size and number of filters (CNN, CNN-LSTM, and TCN). The results showed a significant lead (half the loss of other networks) for TCN type networks over all other types tested architectures. Additionally, and as expected, the three primary hyperparameters in terms of objective function reduction are the network type, network depth and the number of dilations. In terms of network depth, a value of 1-3 layers was found to be appropriate, while for dilations it was around 24 dilations. A summary of the results can be seen in Figure 16.

For the second study, 108 tests were conducted to further refine the rest of the hyperparameters. Based on these tests, it was found that the appropriate kernel size is around 24 while almost the entire range of tested number of filters were acceptable so a central value of 64 filters was chosen. The network depth remains unchanged from the first study with a slight lead for a network depth of 3 layers compared to 1 and 2 layers. The rest of the hyperparameters were further refined as well and all results can be seen in Figure 17.



**Figure 16:** Summary of the Optuna results for the first study of the inversion model following the same format as Figures 14 and 15 with the ray plot (a), slice plot (b), and hyperparameter importance (c).



**Figure 17:** The final HPO for TCN-based inversion model. Some hyperparameters are clearly defined while others are more spread as highlighted in red.

### 3.1.1C HPO-based Final Models

The two previous sections showed the process of HPO for the two primary models needed to create a consecutive neural network without any wavelet transformation. This section will show the final HPO-based models created and their training, validation and blind-testing results. The results will be shown quantitatively via objective function minimization for the training and validation datasets and the average RMSE for the predictions of 5,000 blind testing datasets with one example shown visually.

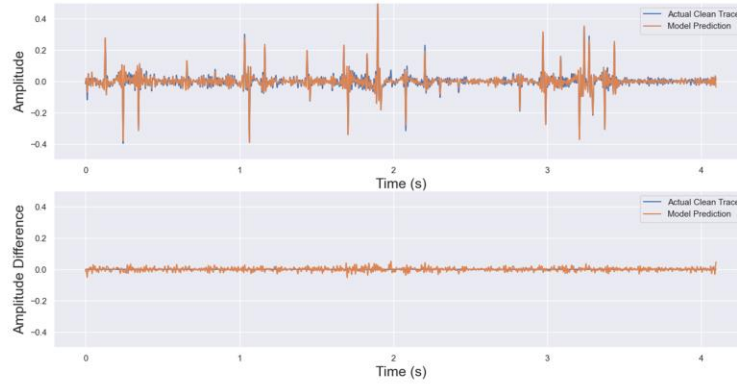
Firstly, for the denoising model, an average RMSE value of 0.0105 was achieved with the blind testing datasets which is very similar to the training and validation losses achieved while training the model in Figure 18. Additionally, Figure 18 shows a blind testing dataset with the actual dataset and the difference between them to show the accuracy of the prediction qualitatively. Secondly, the inversion model had an average RMSE value of 2748.79 for the blind testing datasets which is higher than the minimum validation loss by 1314.84 which is significant. Nonetheless the prediction results seem to be relatively good despite the higher RMSE as shown in Figure 19. In fact, the prediction error seems to be minimal compared to errors observed in conventional approaches (e.g., Daoud & Abd El Dayem, 2018; Almarzooq, 2022).

An interesting side note is that during the HPO process, the CNNs performed significantly better via autoencoders for denoising compared to TCNs, while the opposite was true for the inversion model in which TCNs outperformed CNNs. This could be related to the preceptive field of each architecture. For example, TCNs and their dilatational approach allows for both global and local sampling of the input dataset which in the case of the velocity inversion allows it to learn both the low frequency part as well as the high

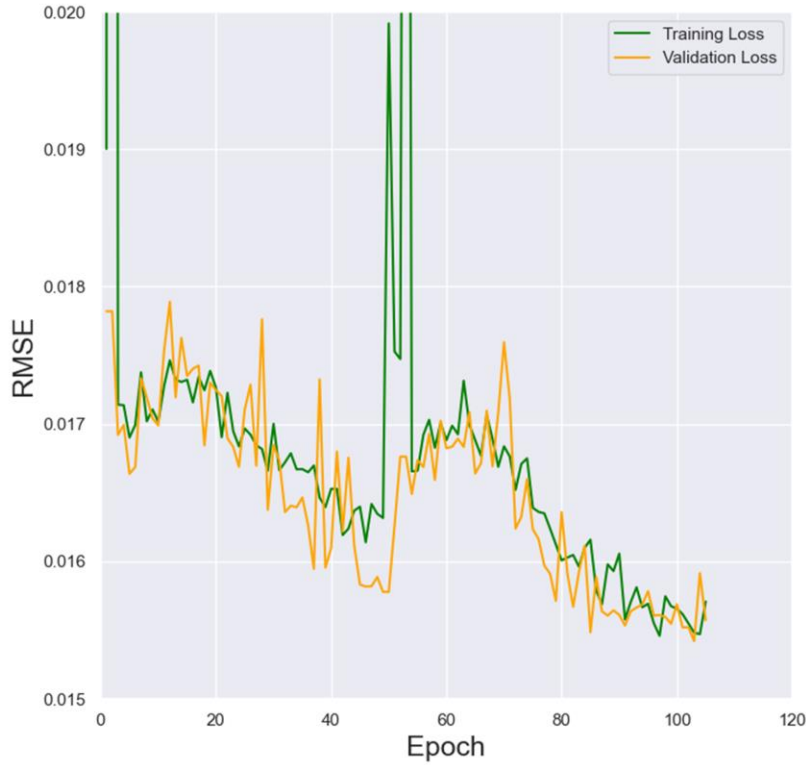
frequency fluctuations more reliably compared to CNNs which are more localized in the preceptive field and don't have this "variable" perception of TCNs. For autoencoding tasks such as seismic trace denoising on the other hand, this local preceptive field allows the network to focus on the noise and remove it more robustly compared to TCNs in which their dilatational aspect hinders their ability to focus on the local features causing it to underperform compared to CNNs.

With these two models in hand, the tools needed to run the consecutive neural network are almost ready. The next section will deal with few considerations that might further refine the generated models.

**(a) Blind Test Prediction Results**

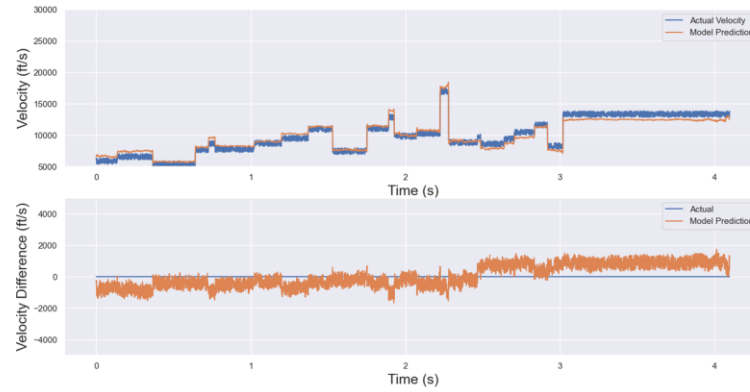


**(b) Training and Validation Loss**

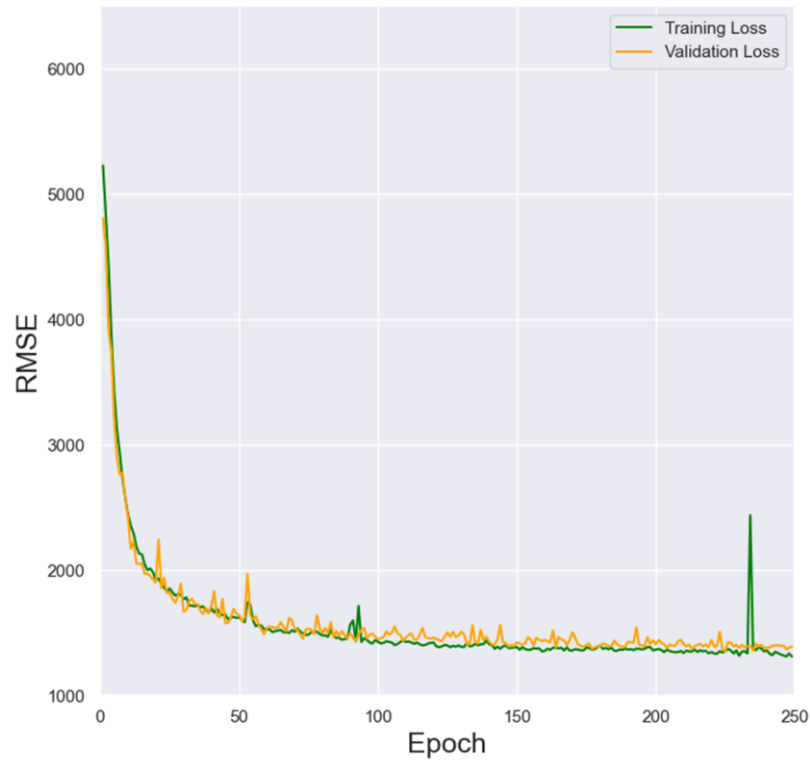


**Figure 18:** (a) Shows an example of a blind testing dataset prediction with the actual clean trace at the top and their differences at the bottom. (b) The training (green) and validation (orange) losses for the model used to generate the above prediction are shown here.

### (a) Blind Test Prediction Results



### (b) Training and Validation Loss



**Figure 19:** (a) a blind testing prediction for velocity at the top and the error at the bottom. (b) training and validation curves for the inversion model used in (a).

### 3.1.2 Further Refinements

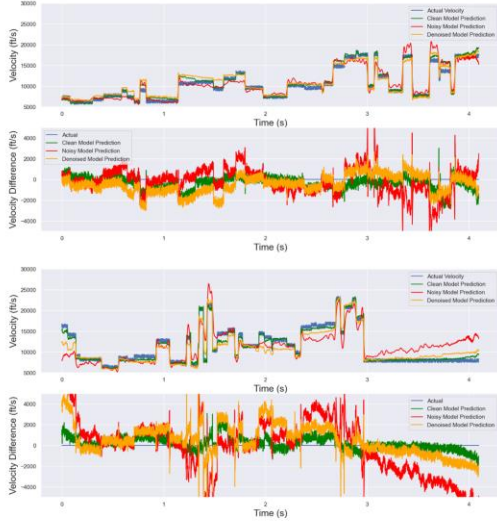
This section will cover additional options that were considered for the final models to understand the thought process of why the consecutive neural network was constructed. The first part will deal with the reason for including the denoising sub-model why it was needed. The second part will deal with autoencoders as a feature extraction model to enhance the computational performance of the inversion model by the way of data encoding or compression.

#### 3.1.2A Denoising Sub-Model Effect

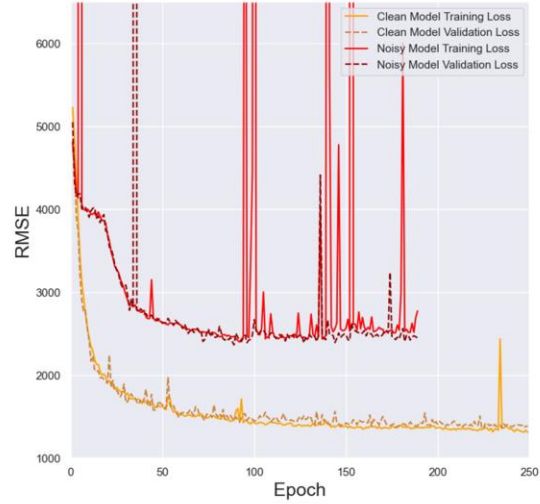
The problem of inverting for velocity can be either solved by directly inverting the noisy trace or after denoising it first. It seems intuitive to assume that the denoised trace would perform better but this section will explicitly show three predictions and compare them with the actual velocity quantitatively and qualitatively. The first prediction will be done by a model trained on noisy traces to invert for the velocity, the second prediction will use clean traces for the inversion and will act as a best-case scenario of what is possible in terms of predictions, and the final prediction will use denoised traces. The second and final predictions will be done via a model trained on clean traces for velocity inversion with the assumption that the denoised traces are going to be almost identical to the clean traces which seems to be a very fair assumption as can be seen in Figure 18 from the previous section. For randomly generated 1000 blind testing datasets, the results were as expected with the clean prediction performing the best at RMSE of 2875.79, denoised prediction in

the second place with 3018.58 RMSE, and the least performant being the noisy model prediction at 3404.15 RMSE (see Figure 20 for a representative example and the loss curves of the models used). If the clean model is set at the baseline, then the denoised model performs almost four times better than the noisy model (4.96% more RMSE from the baseline as opposed to 17.50% for the noisy prediction).

(a) Two Blind Test Prediction Results



(b) Training and Validation Loss

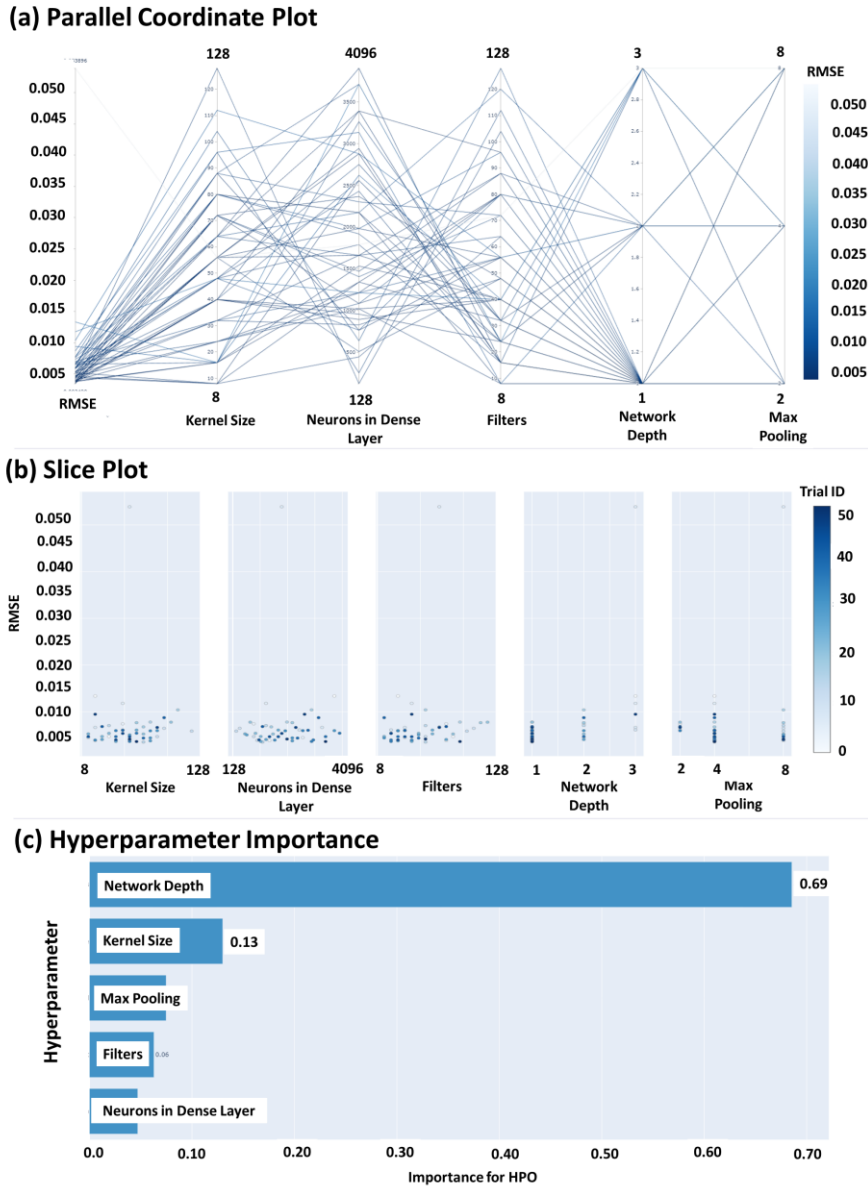


**Figure 20:** (a) Shows two representative blind testing predictions for velocity from clean (green), denoised (orange), and noisy (red) traces. As can be observed all perform relatively well but the noisy prediction is more susceptible to be influenced by noise despite the fact that the model it was trained on used noisy traces for training. The denoised prediction is much closer to that of the clean prediction. (b) Shows the training and validation losses for the two models responsible for the clean and denoised data (orange) and the noisy data (red) predictions.

### **3.1.2B Autoencoders for Feature Extraction and Encoding**

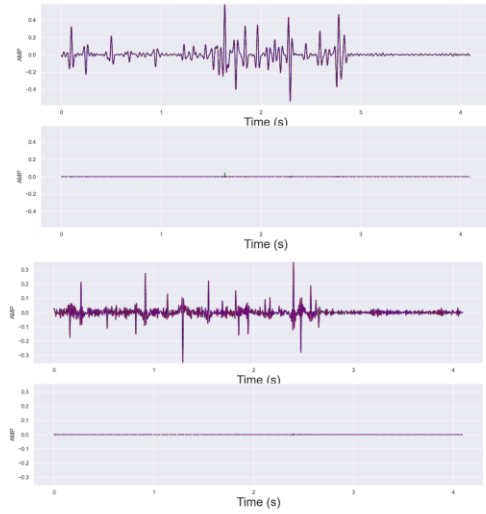
The final addition that might provide a boost in computational performance for both the training and prediction of the inversion model is the use of autoencoders for feature extraction and data encoding. It has already been shown in the denoising section that CNN architectures excel at autoencoders and thus an Optuna study similar to the one done in 3.12.1 for denoising will be used for feature extraction. This Optuna study contains 50 tests, and the results are shown in Figure 21. It shows that the most important hyperparameters are the network depth, kernel size, and max pooling value. The optimal hyperparameters for feature extraction are a network depth of 1-2 layers, kernel size of 48, max pooling value of 2 or 4, a filter size of 64, and 2048 neurons in the dense layer. While deeper networks perform relatively worse than shallower networks for autoencoding (which is to be expected), they offer a higher degree of compression (encoding) and subsequently performance improvements. Thus, three final networks with all the previous optimal hyperparameters fixed with the exception of network depth which will be varied to 1-, 2-, and 3-layers models. The results of these three autoencoders are shown in Figure 22 which shows that unexpectedly the model with a depth of two layers performed the best, followed by the two other models. The RMSE values for 2000 blind testing models for the three layers are 0.0014, 0.0005, and 0.0011 for the 1-, 2-, and 3- layers models respectively. Next, the encoder part of the 2-layer model will be attached to the same inversion model and will be pitted against the inversion model without any encoders attached. The results are shown in Figure 23 and seem to indicate that surprisingly again the model with the encoder attached outperforms the model without the encoder. 4000 blind testing datasets were used to ensure that the previous observations are reliable, and they seem to agree with

the RMSE values of 2781.62, and 2146.56 for the standalone inversion model, and 2-layers encoder-inversion model respectively. Two important points need to be addressed regarding the use of the encoder model with the inversion model. Firstly, there is a significant uplift in computational performance. Namely, it offers 10 times faster predictions with the 2-layers model (compression factor of 16) compared to the model without the encoder. There is a downside to this encoding, however, which might be subtle in Figure 23, the encoder model seems to fail at picking very thin layers and the exact point of transition between layers. This can be observed by noticing the spikes in the velocity difference plot compared to the no encoder model which seems to be able to handle these thin layers more reliably.

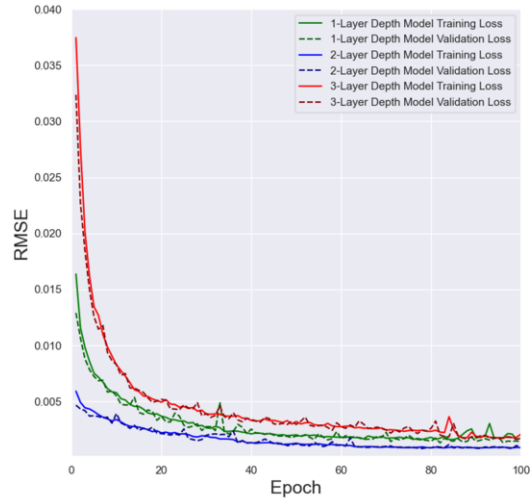


**Figure 21:** The Optuna study used to design a feature extraction autoencoder to pair with the inversion model. (a) A parallel coordinate plot where the density of the rays indicates the more optimal models. (b) Slice plot with the hyperparameters plotted against the RMSE value of the model. (c) A list of the most important hyperparameters for the objective function.

(a) Blind Test Prediction Results

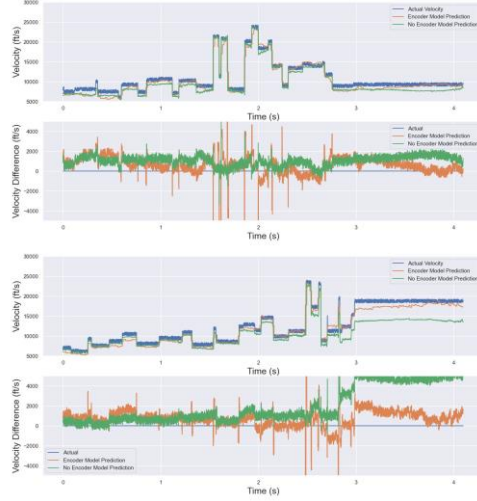


(b) Training and Validation Loss

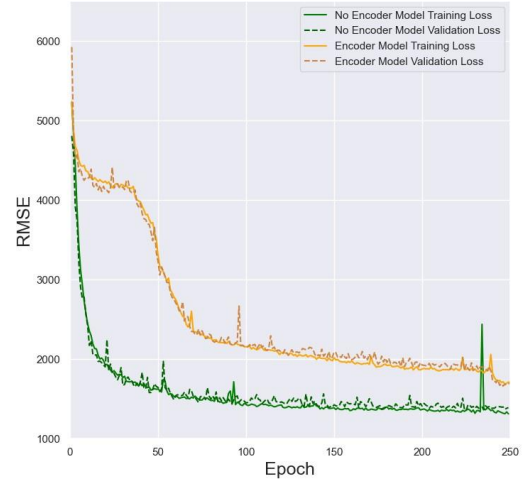


**Figure 22:** (a) Two blind testing results for the three 1- (green), 2- (blue), and 3-layers (red) feature extraction autoencoders. The results show that all three more or less converge to about the same training and validation losses (b). Surprisingly, the 2-layers model outperforms the other two.

(a) Two Blind Test Prediction Results



(b) Training and Validation Loss



**Figure 23:** (a) Two blind testing results for encoder-inversion (orange) and standalone inversion model (green). The results show that both are performing nicely with a slight better performance for the encoder-inversion model. (b) This seems to be counter to the training and validation losses for the two models which indicate that the standalone should perform better than the encoder-inversion model. To confirm this counterintuitive result, the blind testing was performed on 4000 datasets and they showed that the encoder-inversion model outperformed the standalone model with the former having an RMSE value of 2146.56 compared to 2781.62.

### **3.1.3 Inversion for Other Properties**

The seismic trace or the corridor stack in this case contains a wealth of information hidden behind the convolution the reflectivity series, wavelet, and noise. In the previous sections, the process of inversion for velocity information was primarily attempted as it is the main aim of this thesis work. Nonetheless, this section here briefly shows the potential of inverting for other properties. Some of these properties are potentially easier to invert for such as acoustic impedance while others are more challenging such as density. The easier task of inverting for acoustic impedance will be attempted first with the joint inversion for velocity and density in the subsequent section.

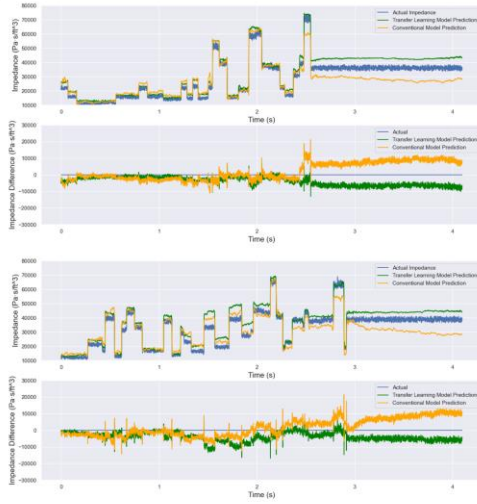
#### **3.1.3A Inversion for Acoustic Impedance**

The inversion for acoustic impedance will be attempted via two approaches. The first approach will follow exactly the same methodology performed in the previous section and will use the same network architecture and hyperparameters, but the training will be done from scratch. The second approach will attempt transfer learning in which the final model used for velocity inversion in the previous section will be used as a basis to fine tune the model for acoustic impedance inversion rather than velocity. Once done, the two approaches will be compared and evaluated based on the same criteria as before, the training and validation losses as well as blind testing datasets.

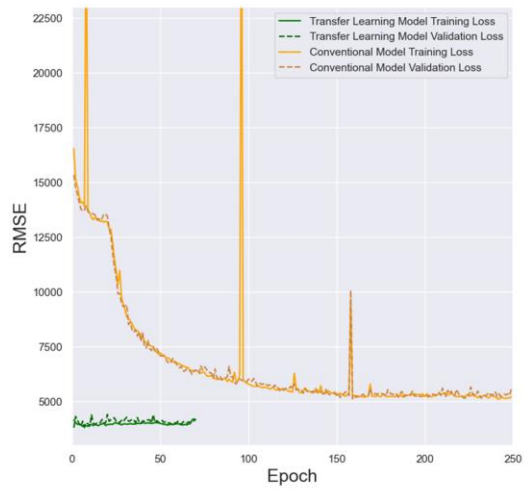
The results show a significant advantage in terms of both prediction and training speed performance when utilizing a transfer learning approach from an already trained velocity

inversion model as an opposed to a model trained from scratch. The minimum validation loss achieved for the model trained from scratch is an RMSE value of 5136.34 at epoch 159. In comparison, the transfer learning model achieved a minimum RMSE value of 3814.96 after just 3 epochs. So, the transfer learning approach is 98.11% faster and 25.73% more accurate in terms of validation loss than the conventional approach. To drive this point further, Figure 24 shows a comparison of the objective functions for two models as well as examples of blind testing performed. The average RMSE for 1000 blind testing datasets is 6310.89 for the transfer learning model and 9298.51 for the conventional model closely matching the difference in accuracy between the two models for the validation loss.

(a) Two Blind Test Prediction Results



(b) Training and Validation Loss



**Figure 24:** (a) shows two representative examples of blind testing datasets with the transfer learning prediction (green) and conventional model prediction (orange). (b) The training and validation losses for the transfer learning and conventional models. Note that while the transfer learning model was allowed to run for 70 epochs, it achieved the minimum validation RMSE value at just 3 epochs of training.

### **3.1.3B Joint Inversion for Velocity and Density**

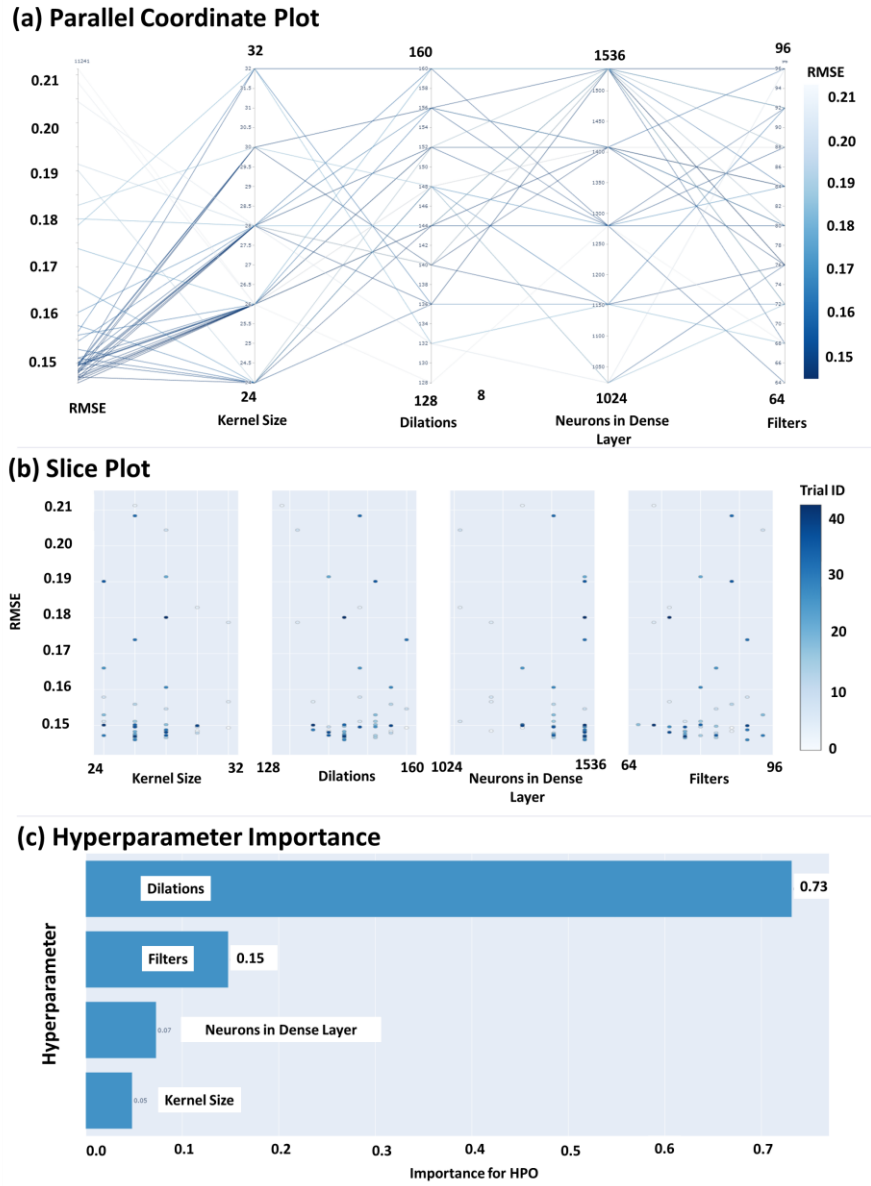
With the success of transfer learning in the previous section, one would imagine that the same technique would be used here. However, because two outputs are expected here rather than one, the network architecture is slightly different. Thus, a transfer learning technique might not be suitable for this joint inversion of velocity and density. Traditionally, one does not scale the output of a network or expect the scaling to have an impact on the results. However, it is probably better to scale the density and velocity since their difference in magnitude is quite significant. In other words, if left unscaled most of the loss will come from the velocity label with minimal updates from the density label. For these reasons, the labels will be scaled, and a model will be trained from scratch for the inversion of velocity and density jointly.

Initially, the same network architecture and number of parameters as the velocity inversion model were used. However, the results were unsatisfactory, and this could potentially be due to one of two reasons. Firstly, it could be that the problem of inverting for both density and velocity is simply infeasible with the current input (the corridor stack) and additional inputs may be needed to successfully perform this inversion. The second option could be that this joint inversion is orders of magnitude more complex than an inversion for acoustic impedance or velocity alone. To eliminate the second option, a hyperparameter optimization via Optuna for this added complexity will be performed first. Based on this, a final model will be trained and blind tested to understand whether this joint inversion is feasible with the corridor stack as its sole input or if possibly additional inputs are needed.

The results of HPO showed that better performance would be achieved by roughly doubling the number of parameters used in the model as can be seen in Figure 27. The most important

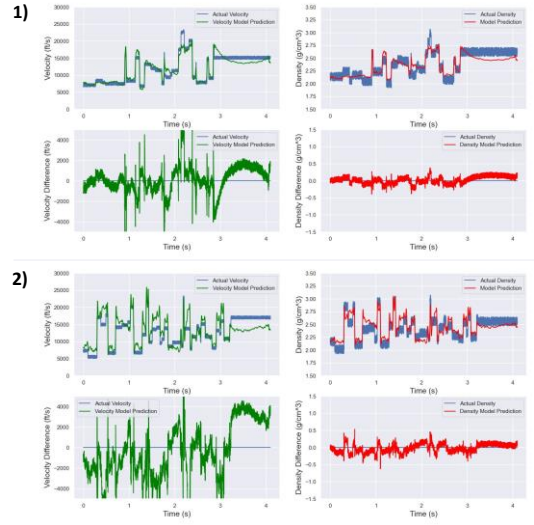
amongst the analysed hyperparameters is the number of dilations within TCN layers in which an increase from 128 to 156 dilations was found to be more optimal. Other hyperparameters were increased slightly from their original values but their combined total importance was found to be only 27% compared to the number of dilations with 73%. The number of filters was increased from 64 to 80 filters, number of neurons in dense layer from 1024 to 1536 neurons, and the kernel size was increased from 24 to 28. Other hyperparameters were kept the same as before.

The final model produced from the above HPO shows a reasonable ability at predicting both velocities and densities as can be seen in Figure 28. It shows two representative examples of blind testing results for both velocities and densities predicted by the model. Due to memory constraints only 200 blind testing datasets were tested and their average RMSE is 2907.27 and 0.165 for velocity and density respectively. Thus, it seems at least for the synthetic datasets and models used in this thesis, the prediction of velocities and densities simultaneously is feasible with the refined model based on the HPO analysis in Figure 27. Interestingly and surprisingly, it seems that the density prediction performance in this model is better than that for velocity despite it being usually the opposite. It remains to be seen whether a similar performance could be achieved with real datasets.

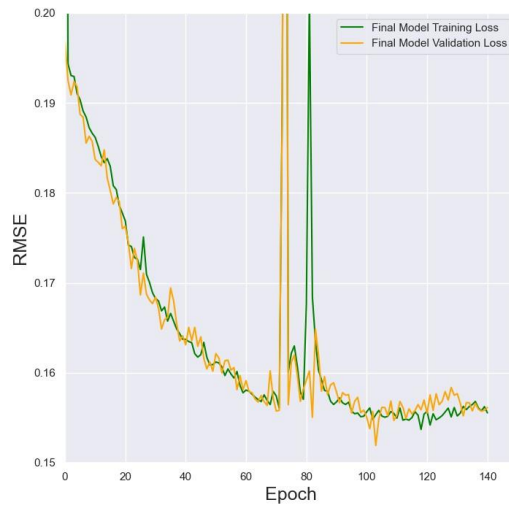


**Figure 25:** HPO for refining the joint inversion model. The results show a slight increase in all parameters from their base values could potentially improve the results of the joint inversion.

(a) Two Blind Test Prediction Results



(b) Training and Validation Loss



**Figure 26:** (a) Two blind testing prediction results for velocities (green) and densities (red) with the actual velocities and densities (blue). (b) the training (green) and validation (orange) losses used for the joint inversion predictions.

### 3.1.4 Blind Testing on Finite Difference Dataset

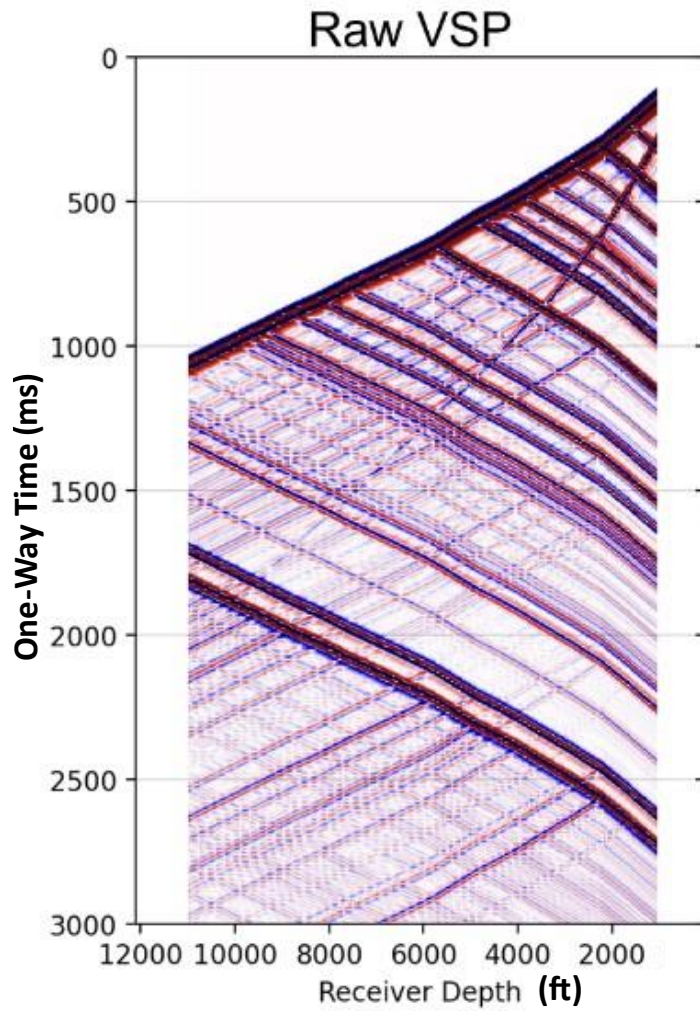
In this section, an attempt at a truly blind testing will be done. This will be accomplished by using a finite difference ZVSP dataset that was generated without any prior knowledge of the original model that was used to generate it. The dataset is obtained from Campbell (2023) and was processed via the same code. The dataset can be seen in Figure 27 and while the parameters of the modelling scheme are not known, one can glean some insight by examining the dataset. For example, three wavefields can be observed in this dataset, a downgoing P-wavefield, an upgoing P-wavefield, and a less apparent downgoing S-wavefield. This indicates that the modelling scheme is elastic rather than acoustic. Furthermore, the presence of  $\sim 23$  upgoing primary P-reflections indicate that the model is relatively detailed and that there exists at least what appears to be three primary reflections below the final receiver which makes this dataset optimal for testing a VSP inversion approach.

So, an approach not too dissimilar from Figure 3 was carried out to process this dataset and obtain the final corridor stack as can be seen in Figure 28. The major processing steps include separating the downgoing (Figure 28A) and upgoing (Figure 28C) P-wavefields and then deterministically deconvolving them (Figure 28D). Finally, a corridor mute below the first breaks up to the final few receivers (the data below them is kept to capture deeper reflections) is performed to ensure that mostly only the primaries remain in the data (Figure 28E). This muted deconvolved upgoing wavefield is then stacked to obtain the final corridor stack (Figure 28F).

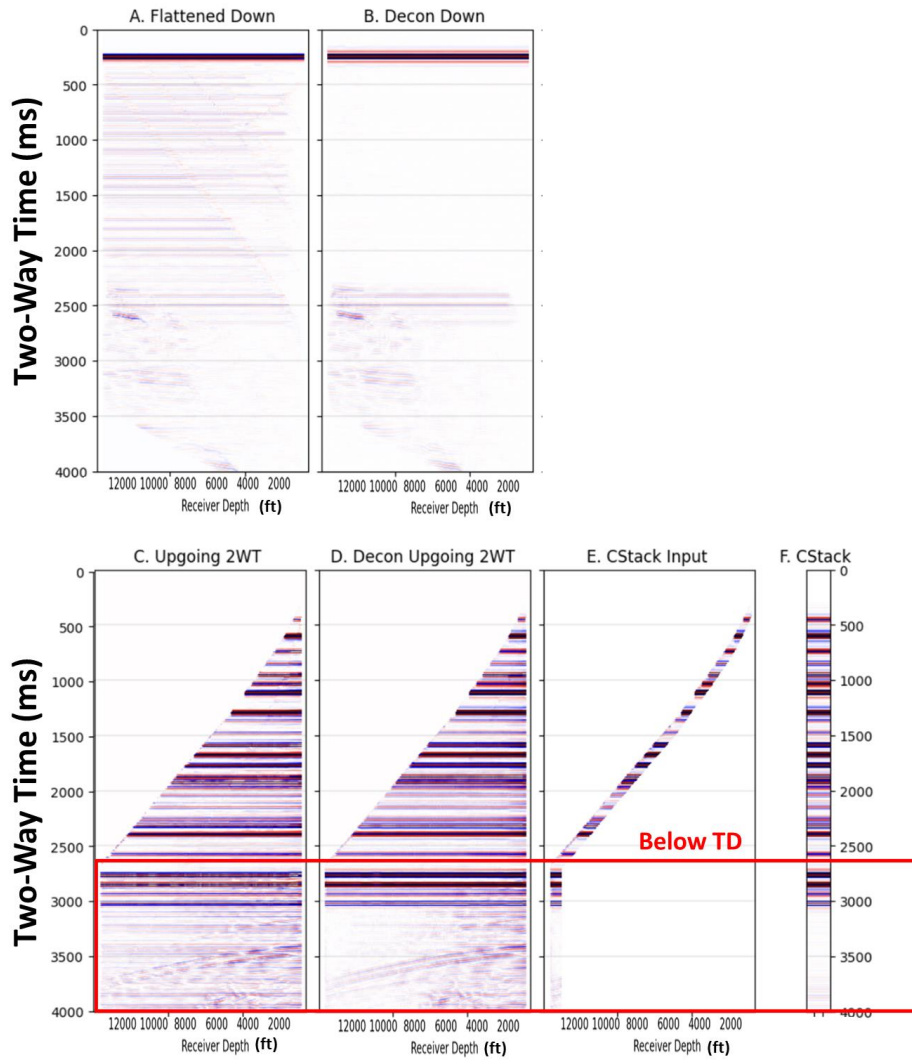
The final step is to use this corridor stack to predict the velocity profile for the entire well. Prior to that, and as discussed in 2.2.3, the corridor stack needs to be normalized prior to inputting it into the trained model. Normally, well logs will be used to generate a synthetic corridor stack that will then be used to calculate a normalization factor for the corridor stack. Since this dataset is synthetic, no well logs exist and one cannot generate pseudo-logs because the original model is unknown, an alternative is needed. An alternative can be done to find this normalization factor which is searching for the optimal factor. In other words, since the velocity in a portion of the well is known, one can use different normalization factors and then predicting the velocity for all of them. The corridor stack with the least error as compared to the measured section will be assumed to have the correct normalization factor. This method will be tested on the encoder-inversion model and the standalone inversion model without the denoising autoencoder as this dataset doesn't have any noise. This final comparison will attempt to answer if indeed the encoder-inversion model is performing overall better than the standalone inversion model.

As can be seen in Figure 29a, for all normalization factors, the encoder-inversion model outperformed the standalone inversion model. Nonetheless, best factors for each were estimated as -2.61 and -2.85 for the encoder-inversion and the standalone inversion models respectively. Note that the scaler is negative which indicates that this dataset's polarity convention is opposite of the one the model was trained on. Furthermore, even with incorrect scaling of the data, the model was able to predict a somewhat reasonable (if incorrect velocity) which is probably due to the inversion being driven in part by the relative differences in reflectivity rather than the absolute values. Still, there is no question that finding the correct scaling factor is essential to a successful inversion especially for

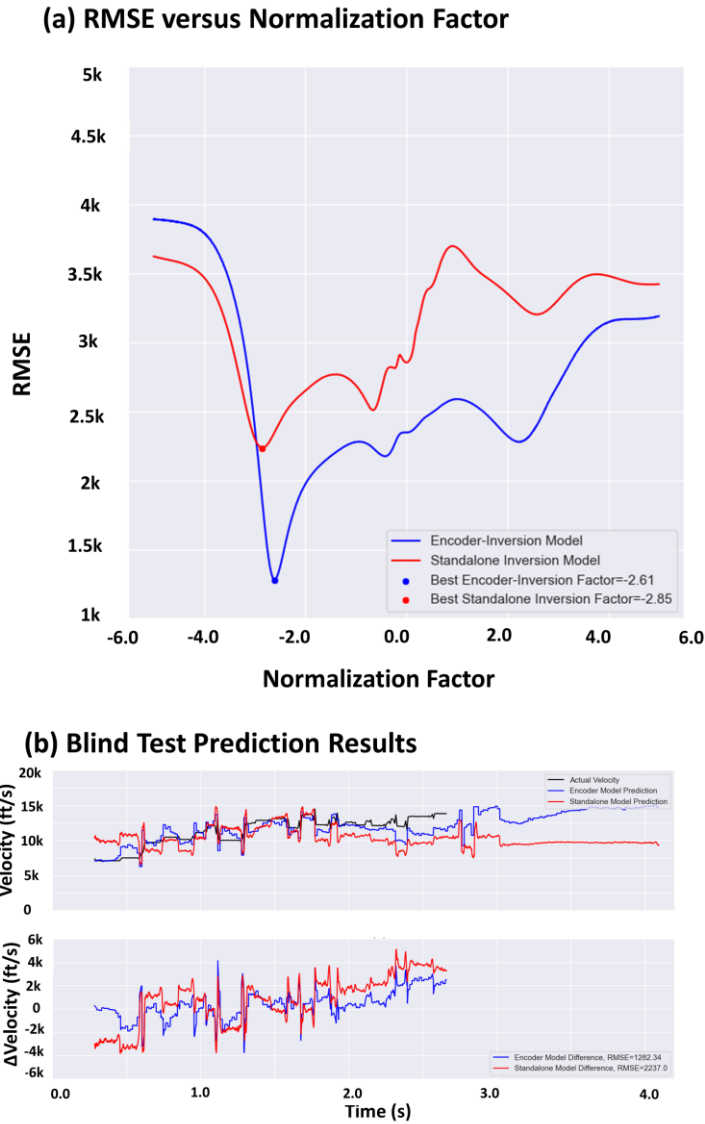
datasets that weren't part of the training or validation regime like this finite difference dataset. Looking at the inversion results in Figure 29b, it again confirms that even with the appropriate normalization, the encoder-inversion model outperforms the standalone model with the RMSE values being 1282.34 and 2237.00 respectively. While one cannot be sure which one of the models perform in the "undrilled" section as the actual velocity model is unavailable as is often the case with real datasets, it seems to be fair to assume that the model performing the best at predicting the section with measured velocities to be also performing better at the undrilled section. Finally, it seems that the standalone model low frequency trend is more or less flat at around  $\sim 10$  k ft/s while the encoder-inversion model seems to be capturing a more appropriate trend going from  $\sim 7.5$  k ft/s at the shallow section to  $\sim 15$  k ft/s in the deeper section.



**Figure 27:** Generated finite difference raw z-component of a ZVSP dataset. The data have dominant downgoing and upgoing P-wavefields and less apparent downgoing S-wavefield which indicates that the model is elastic. The dataset is obtained from Campbell (2023) so the true model is not known besides the measured section via first break picking.



**Figure 28:** Shows the primary results processing steps of ZVSP on the dataset from Figure 27 starting with flattened separated downgoing P-wavefield (A) and its corresponding deterministic deconvolution (B). Then, the separated upgoing wavefield in two-way time is shown prior to deconvolution (C) and after deconvolution (D). The final step involves muting everything but a corridor below the first breaks (E) and then stacking them to get the corridor stack (F) (Campbell, 2023).



**Figure 29:** (a) A 1D grid search over normalization factors between -5.0 and 5.0 was run in order to determine the most appropriate scaler for the finite difference corridor stack. The RMSE was calculated over the section with the measured velocity (picked section) with the lowest values for the encoder-inversion (blue) and the standalone inversion (red) being -2.61 and -2.85 respectively. (b) The results of the predictions for the two models after scaling the results by the appropriate scaling factor for each at the top, and the difference between the actual and predicted velocity at the bottom.

### 3.1.5 Summary of Direct Inversion Results

This sub-section will attempt to summarize all of the different results achieved in Section 3.1 starting with the results achieved for all the different inversion models with a seismic trace as the direct input. This will include all models inverting for velocity, as well as inverting for acoustic impedance, and jointly inverting for velocity and density. Based on these results, preliminary conclusions will be made in order to prepare for the comparison with the results in the upcoming wavelet transforms section.

The first test that was performed was done to determine the validity of denoising the seismic trace prior to inputting it into the velocity inversion model. This test included three results, the inversion of clean (baseline), denoised, and noisy seismic traces. The first two used the same model while the latter one used a model trained on noisy traces (to attempt to allow it to understand the noise). The results showed that clean and denoised model predictions performed better than the results based on the noisy model. Thus, confirming the validity of the approach of denoising prior to the inversion for velocity.

The second test was completed in order to examine the effect of using encoders for feature extraction and encoding. The training and validation results showed that the model with no encoder performing better than that of the model with the encoder attached. On the other hand, the blind testing datasets used show that the encoder model performed significantly better than all other models that were considered. This could be due to the encoder feeding more relevant information to the trained model compared to the model with no encoder attached. These two tests guided the construction of the final consecutive model used for

the inversion of velocity directly from traces which will be pitted against the final model from the wavelet transform section to produce a final model for this thesis.

Two side tests were performed to understand the feasibility of inverting the same synthetic corridor stacks for other properties. Firstly, the inversion for acoustic impedance was investigated and was found to be feasible both via training from scratch and transfer learning. The latter was found to be significantly faster and performed better in all blind tests that were performed. The other exercise was for the task of jointly inverting for the velocity and density simultaneously and for this task a higher-powered model was needed and this model performed reasonably well in both inverting for velocities and densities with density errors being surprisingly less than velocity errors (in relative terms).

This chapter concluded by performing a final blind testing exercise using a finite difference dataset with an unknown model that was processed with a typical ZVSP workflow to get the corridor stack. Then, this dataset was normalized and the encoder-inversion and the standalone inversion models were used to predict the velocities. No denoising model was needed because the dataset was noise-free and due to the modularity of the consecutive network, excluding it from the test was a relatively easy endeavour. The results confirmed earlier blind testing results that the encoder-inversion model was the best performing model for inverting for velocities.

## 3.2 Inversion of The Seismic Trace via Wavelet Transforms

The previous section showed that direct inversion of the seismic trace for velocity or other properties is not only feasible but also quite accurate. So, one would question the need for the added complexity of a wavelet transformation as a preprocessing step. The primary reason for this addition is that, as shown in section 2.2.2, there have been several cases where wavelet transformation improved the results of both traditional processing and deep learning applications for seismic data. These earlier applications are not related to the issue of seismic inversion. Nonetheless, as these applications and seismic inversion use seismic data as a starting point, it is possible that wavelet transformation will squeeze additional accuracy for the velocity inversion problem.

This section will deal with two wavelet transforms (SWT and CWT) and the results of using them as a part of the consecutive neural network structure. Both will follow a similar workflow, starting with an attempt at encoding and compressing the input data size via autoencoders. Then the encoder part will be separated and used as a part of the inversion for velocity model. Finally, this section will conclude by a comparison between SWT and CWT results in terms of accuracy and computational performance.

### **3.2.1 Stationary Wavelet Transforms (SWT)**

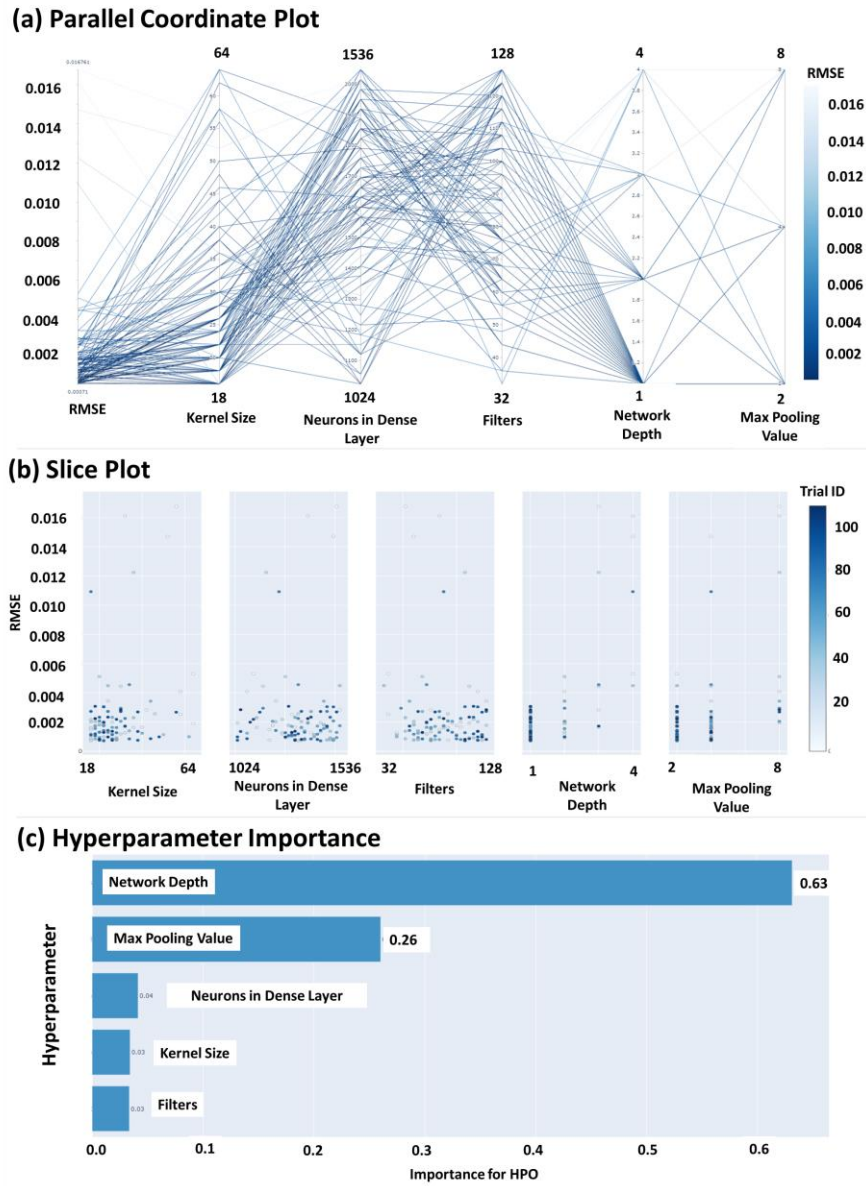
#### **3.2.1A Feature Extraction via Autoencoders**

As discussed earlier, the first step is to create and optimize an autoencoder to minimize the cost associated with the wavelet transformation. To establish this two Optuna studies, one focused on CNNs networks with 109 tests while the other focused on TCNs with 40 tests were performed to obtain the autoencoder with the lowest loss and with a reasonable compression. The reason for conducting the two studies is that the CNNs significantly outperformed the TCNs in initial tests thus dominating the hyperparameter space search (as can be seen in Figures 30 and 31). Several best models for both the CNN and TCN autoencoders are desired in order to later pair them with the inversion models and to see how they would perform combined.

The results of the first study for CNNs seem to indicate that a relatively shallow network (1-2 network depth), with a max pooling value of 2-4 is appropriate. Other hyperparameters such as the kernel size, number of filters, and number of neurons in the dense layer have a minor effect on the results accuracy. For the final CNN-SWT autoencoder models, three models were created with varying network depths of 1, 2, and 3 layers (symmetrical), max pooling value of 4, kernel size of 24, number of filters of 64, and a number of neurons in dense layers of 1536. The compression for the shallowest model has a factor of 4 only while the other two models have a significantly higher compression factors of 16 and 64 compared to the original data. The results for these models are shown in Figure 32 with the first model being able to reconstruct the data almost perfectly while the second and third

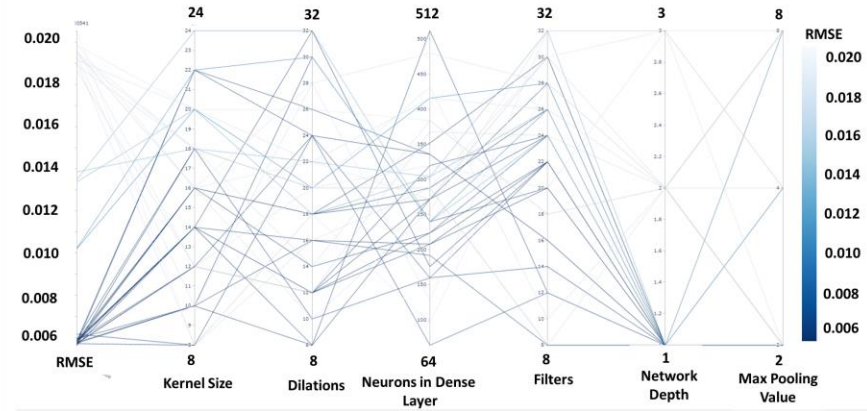
deeper models are able to reasonably reconstruct the data but not to the same quality as the first model. Compression artifacts are localized, barely noticeable, and the most significant one is less than 5% of the maximum amplitude.

For the TCN based autoencoders, the picture is completely different as can be seen in Figure 31. The network performance is heavily dependent on the network depth and anything but the shallowest of networks (i.e. networks with depth of 1), severely underperforms in terms of RMSE. To compare, a TCN network with a depth of 1 layer was created. It has 24 dilations, 24 filters, kernel size of 16, 256 neurons in the dense layer, and a max pooling value of 4. The max pooling value is chosen because the main objective behind autoencoding the data is to compress it to attempt to create a better performant model in terms of accuracy and speed via wavelet transforms and encoder feature extraction. The results of this model clearly indicate (as shown in Figure 33) that the TCN network are significantly worse than CNN based networks for autoencoding. Thus, for the next part only CNN autoencoders will be considered.

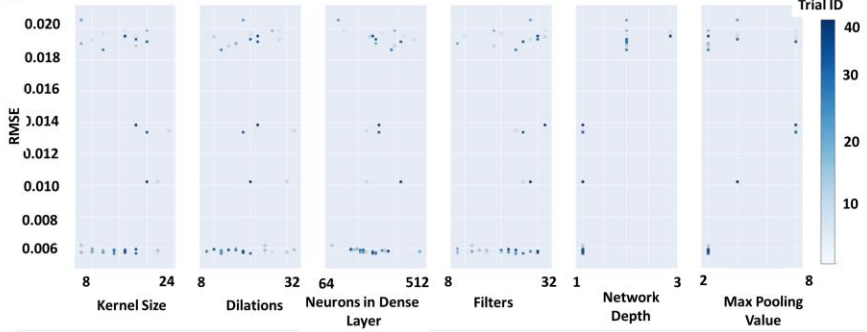


**Figure 30:** The results of HPO for CNN-based autoencoders. (a) The denser the rays in the parallel coordinate plot indicate which values are more optimal. (b) Slice plots of each hyperparameter against the RMSE. (c) The most important hyperparameters, with the most impact on RMSE, are shown here.

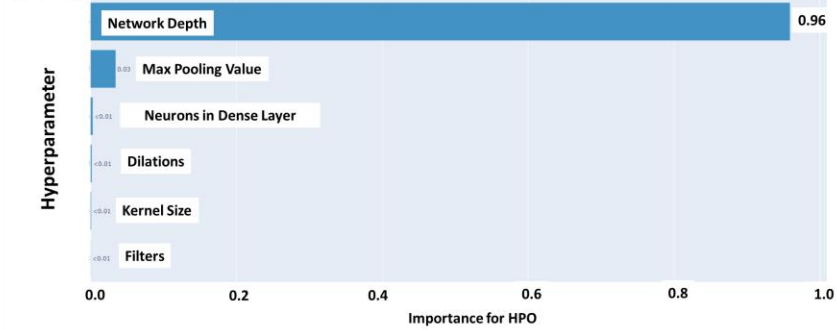
(a) Parallel Coordinate Plot



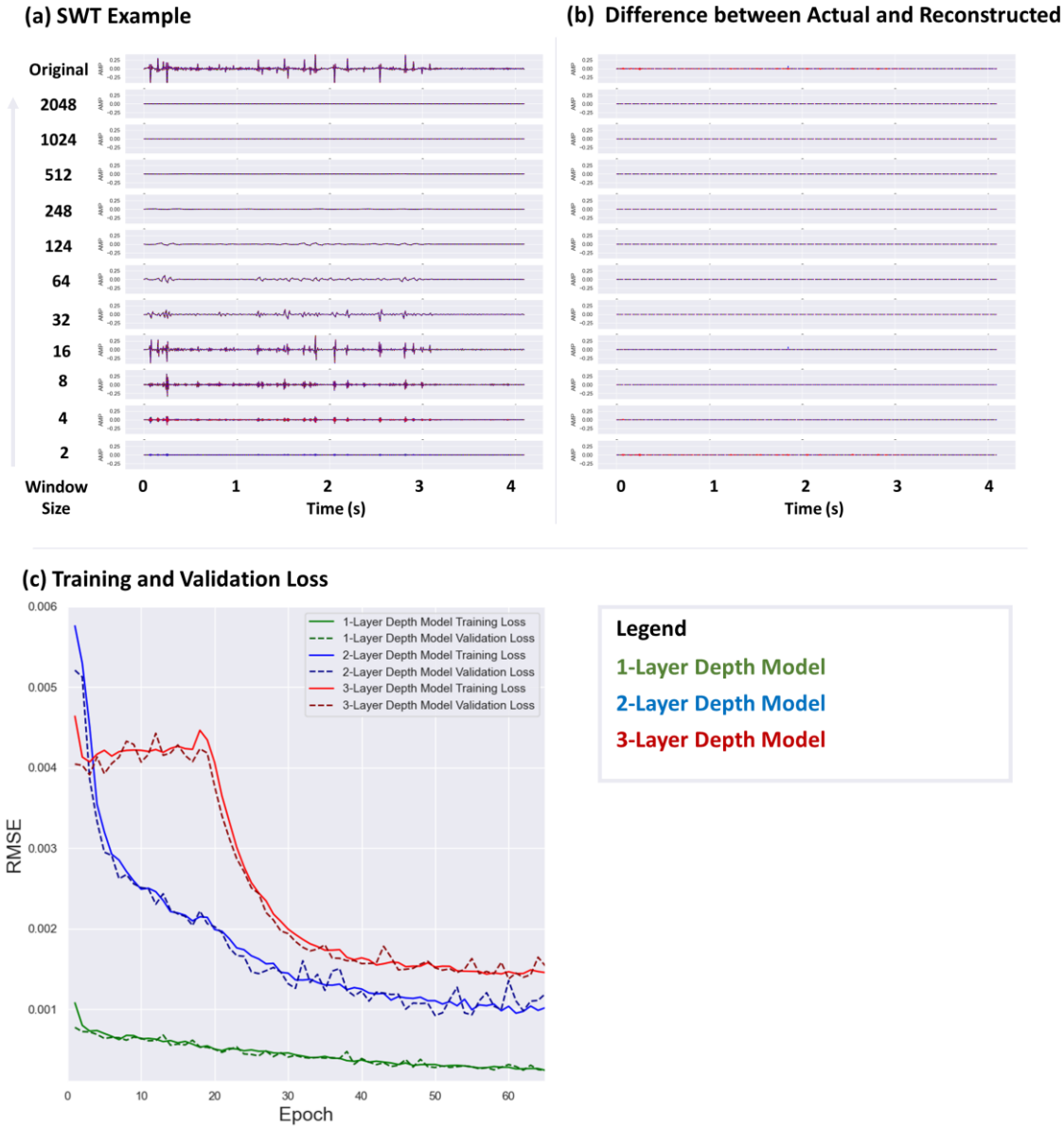
(b) Slice Plot



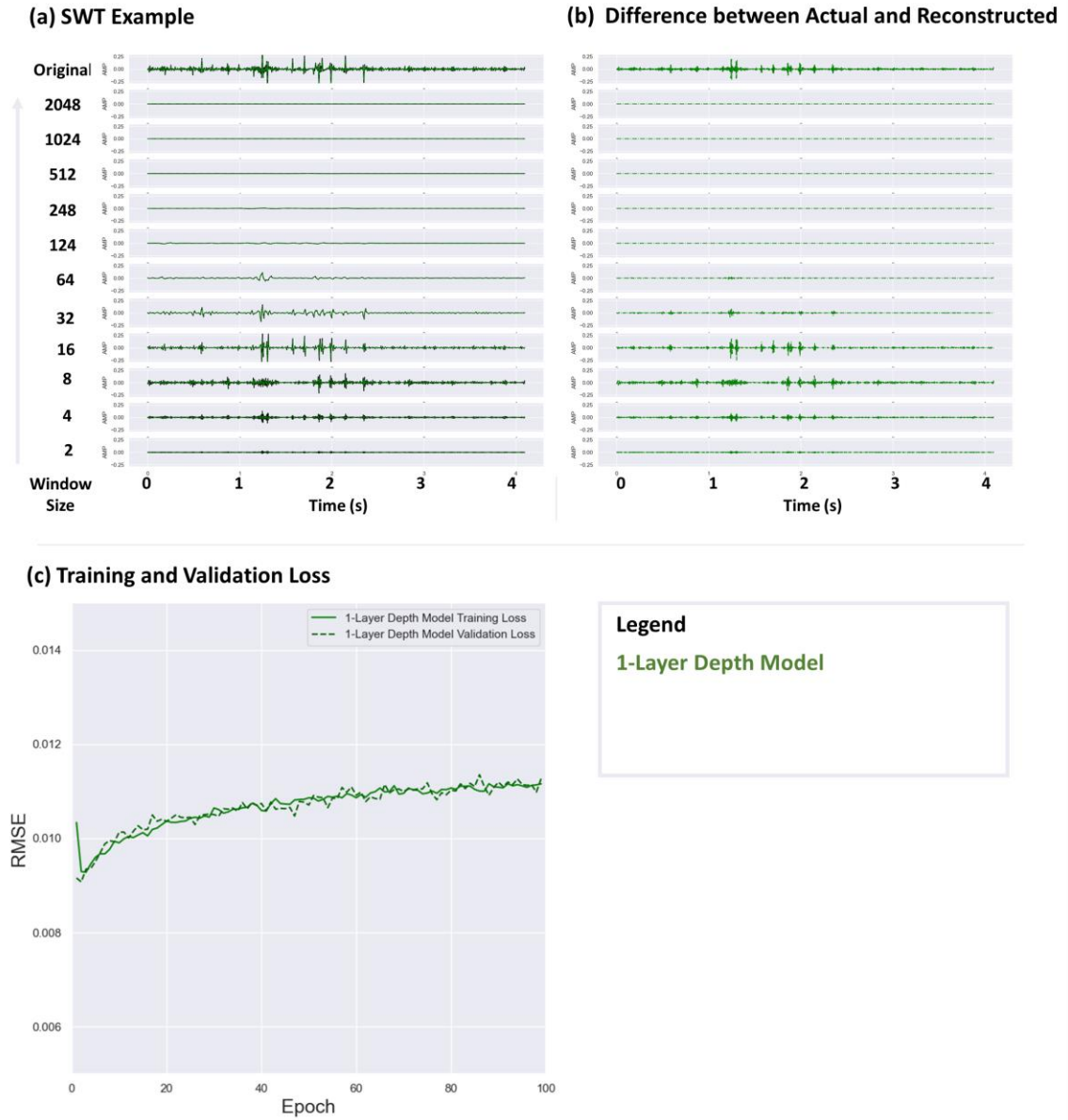
(c) Hyperparameter Importance



**Figure 31:** Similar to Figure 30 but for TCN based autoencoders. In all three plots the dominance of network depth over all other hyperparameters for autoencoding is clear. Note that the ranges are smaller for TCNs because the added dilations parameters increase the overall number of parameters significantly compared to CNN networks.



**Figure 32:** (a) Synthetic Trace Example with its SWT decomposition (black) as well as the reconstructed SWT results for three autoencoder models with network depths of 1, 2, and 3 layers (green, blue, and red respectively). (b) The difference between the actual and the three models, note that the compression artifacts are minimal in all three models (obviously deeper models contain a slightly larger error). (c) The training and validation RMSE losses for the three models.



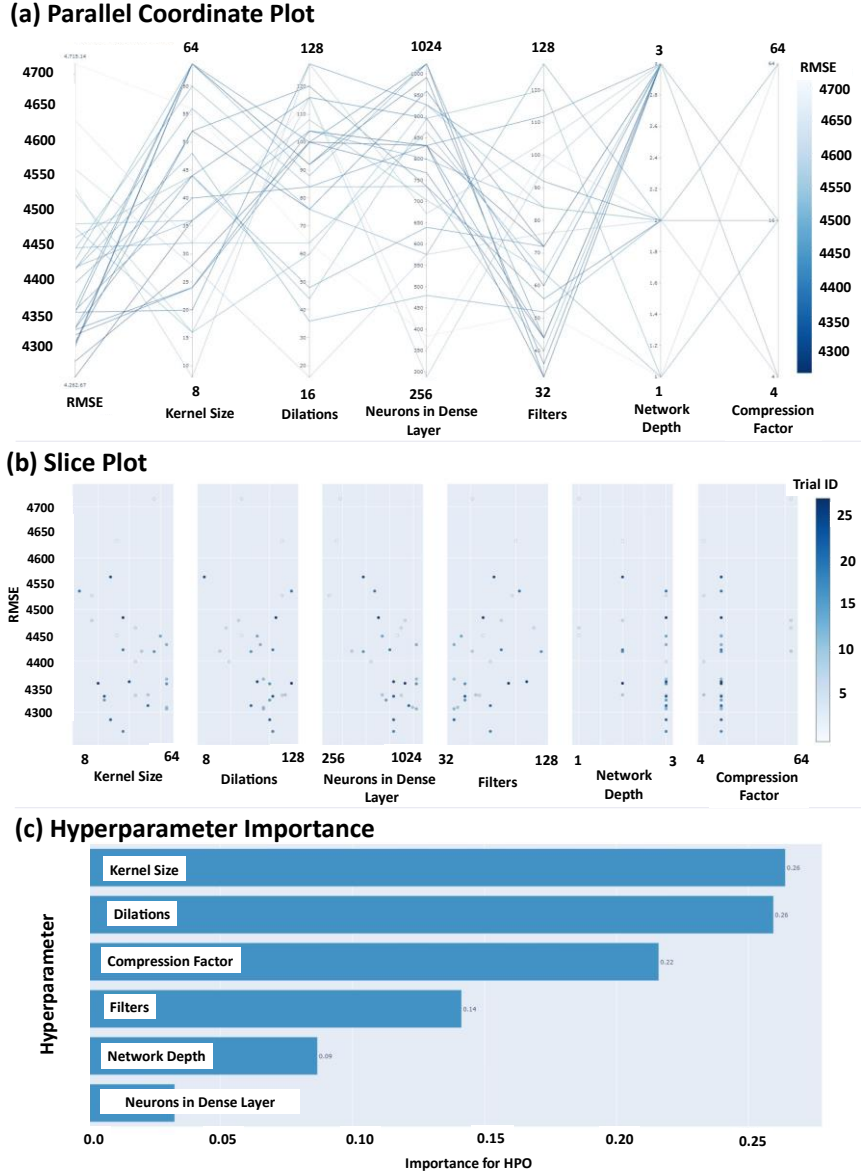
**Figure 33:** Even a shallow TCN autoencoder is unable to properly learn the synthetic seismic trace as can be seen from the prediction results (a), their differences (b), and the training and validation losses in (c).

### **3.2.1B Encoder-Inversion SWT Model**

To be able to determine which encoder to use with the inversion model, an Optuna study was performed with 28 tests in which all three CNN encoders from earlier were tested in combination with variations in the inversion model. The test results shown in Figure 34 indicate that there is an inability for the combined encoder-inversion model (with the inversion model being based on TCN architecture) to perform to similarly to results from the direct inversion results in the previous chapter. The best models achieved a value of around 4000 RMSE which is more than double the direct inversion results as can be seen in Figure 34.

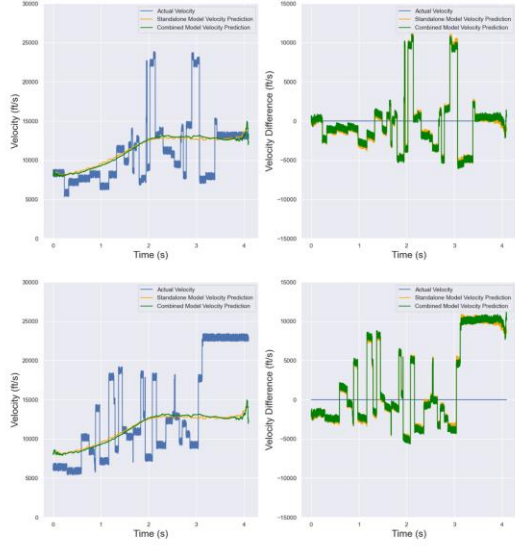
### **3.2.1C Standalone Inversion SWT Model**

To determine whether the issue with the inversion model is due to the SWT or the encoder portion, a standalone model without the encoder (compression) portion is trained and validated and then compared with the combined encoder-inversion model (see Figure 35). The standalone model is much more expensive to run with overall lower number of parameters due to both computational time and memory requirements. Nonetheless, this comparison shows that clearly, the SWT portion of the model is causing issues with the training of the model by only allowing it to learn a smooth trend of the model. It seems that the encoder portion however, is working properly as the combined encoder-SWT model is giving similar RMSE values to the standalone SWT inversion model.

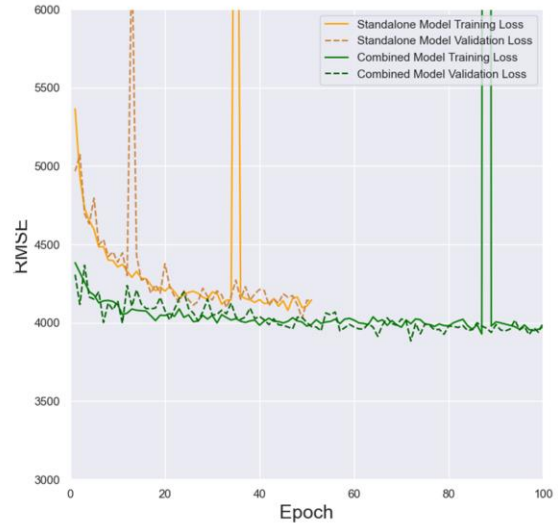


**Figure 34:** The results of the encoder-inversion model (with the encoder being based on CNN architecture while the inversion portion was based on TCN architecture). (a) and (b) clearly show that no matter the compression factor for the encoder or the rest of the hyperparameters for the inversion, the results are significantly worse than what can be achieved via direct inversion. (c) Shows that there is not a single hyperparameter dominating the performance of the model (i.e., no matter the hyperparameter selection, the performance remains weak).

(a) Blind Test Prediction Results



(b) Training and Validation Loss



**Figure 35:** (a) Two blind testing prediction results show that both the standalone SWT inversion model (orange) and the combined Encoder-SWT model (green) are only able to learn a very low frequency representation of the data. (b) The training and validation losses showing a similar story for both models were eventually the loss flattens and the models are stuck at a minimum. It is noteworthy however that the encoder didn't cause any loss of performance in terms of RMSE loss but was 3 times faster in terms of training.

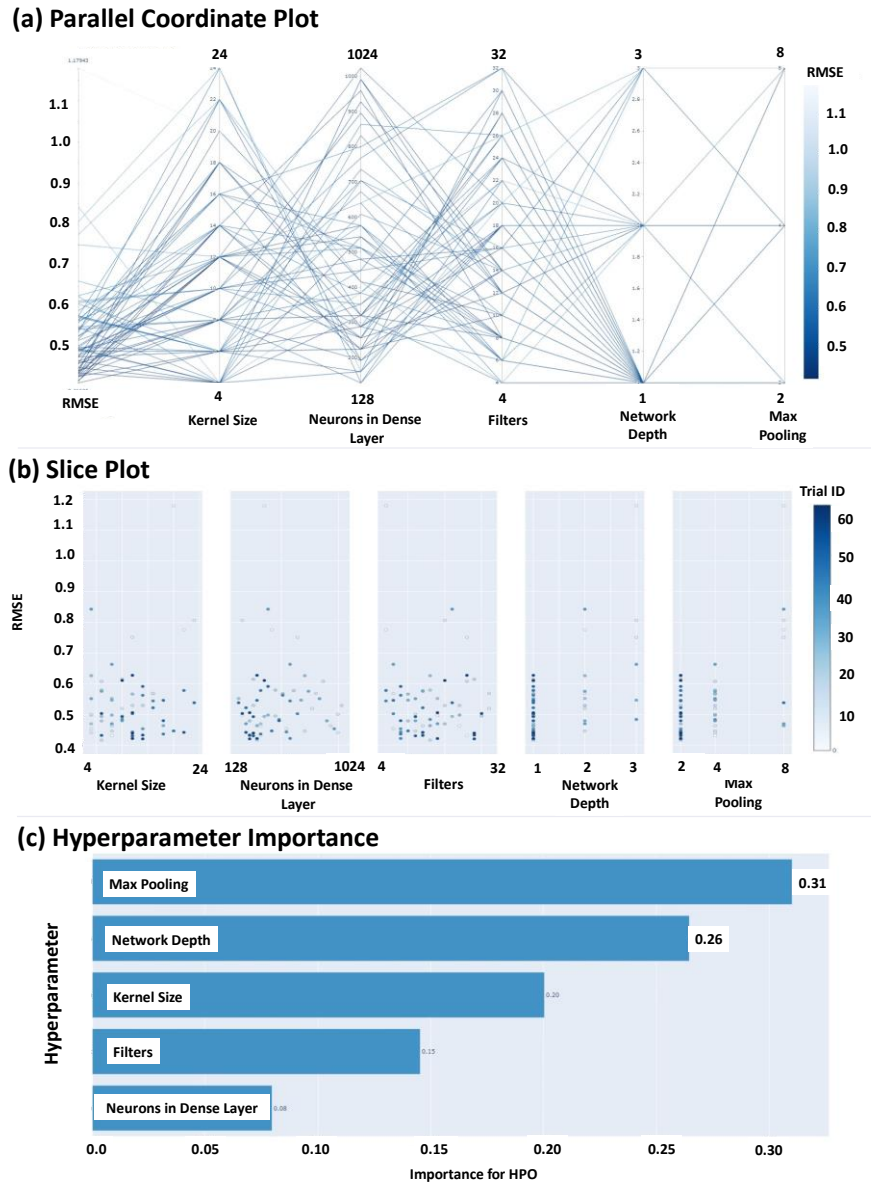
### **3.2.2 Continuous Wavelet Transforms (CWT)**

#### **3.2.2A Feature Extraction via Autoencoders**

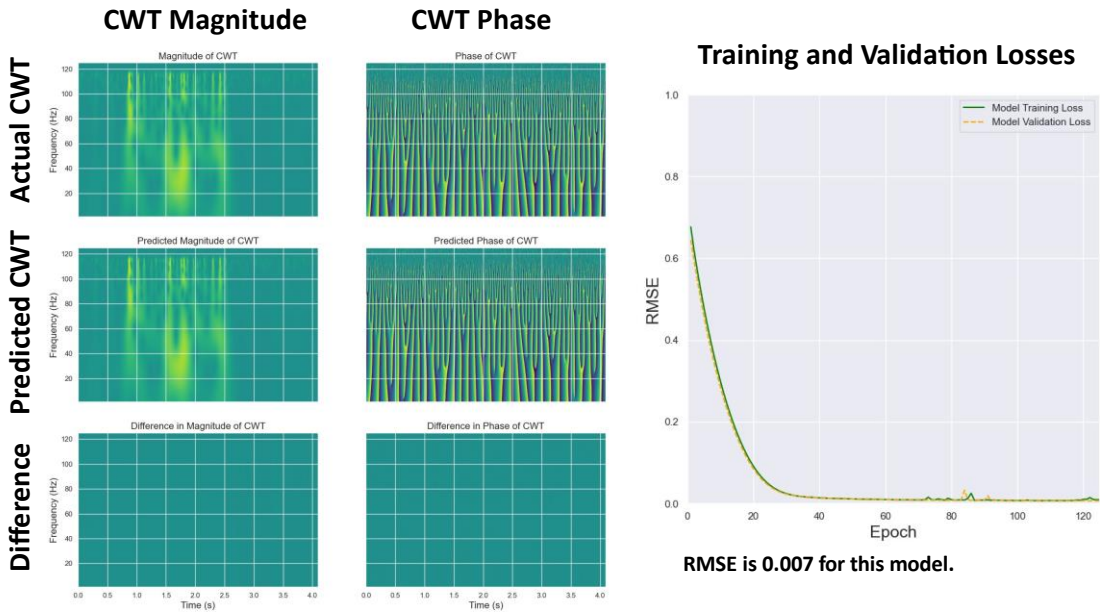
As evident in the previous section, the attempt to use SWT as an additional optimization for the velocity inversion model was unsuccessful for anything but a very low frequency trend of the velocity prediction regardless of whether encoders were used for compression. For completeness, the same procedure will be attempted with CWT which is more computationally expensive as it contains two components: magnitude and phase (as shown in Figure 12). Each component typically contains about 10-12 times more samples compared to SWT and more than 100 times the samples of a single trace. Thus, encoders for CWT are a necessity for a reasonable computational performance rather than being optional as with SWT. Since there was virtually no difference between the results obtained via an encoder-inversion model or a standalone inversion model for SWT, then using the encoder for CWT should not cause any degradation in performance.

Thus, the first step is to find the optimal autoencoder hyperparameters via Optuna and then train a final autoencoder for use with the inversion model. The results of the Optuna study with 65 tests can be seen in Figure 36. Note that the ranges are smaller than previously explored due to the added memory requirements for this larger data size. Next, three final autoencoder model were created and trained as can be seen in Figures 37, 38, and 39 for network depths of 1, 2 and 3 layers respectively. The hyperparameters of the models were chosen based on the Optuna study as follows; 256 neurons in dense layer, 12 filters, and a kernel size of 8 except for the max pooling value which was set at 4. This is to attempt to gain the maximum possible compression even at the cost of some fidelity. All three model are able to reconstruct both of the CWT components reasonably well with the shallowest

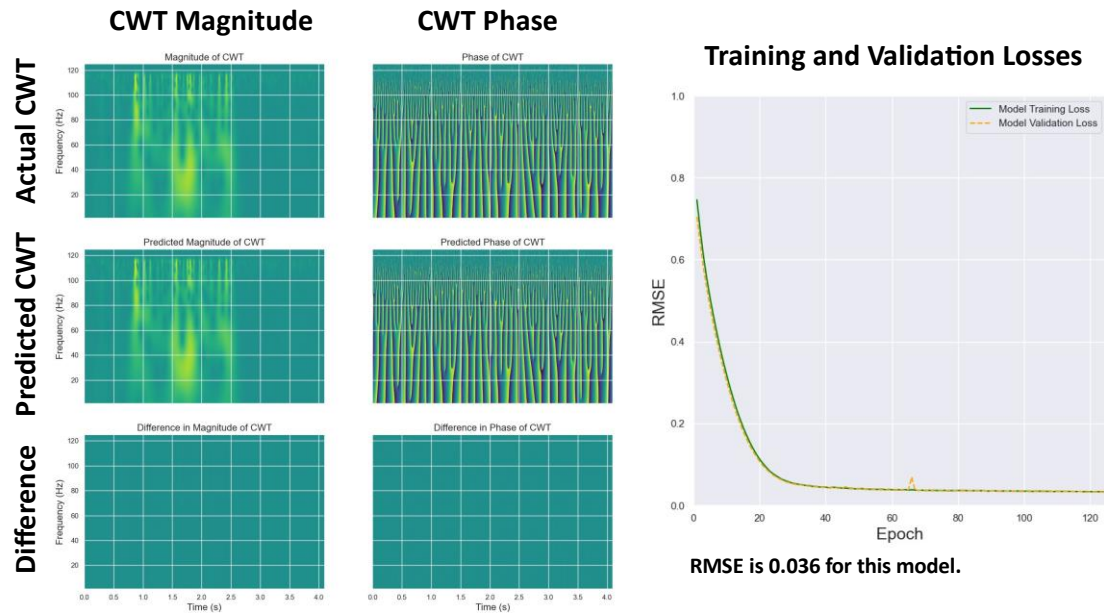
performing the best and the deepest the worst as expected. The RMSE values for the blind testing data were 0.007, 0.036, and 0.166 for depths of 1, 2 and 3 layers respectively. To strike a balance for the inversion model between computational performance and reconstruction fidelity, only the two layers model will be used as an encoder to speed up the performance at a minimal cost to data quality.



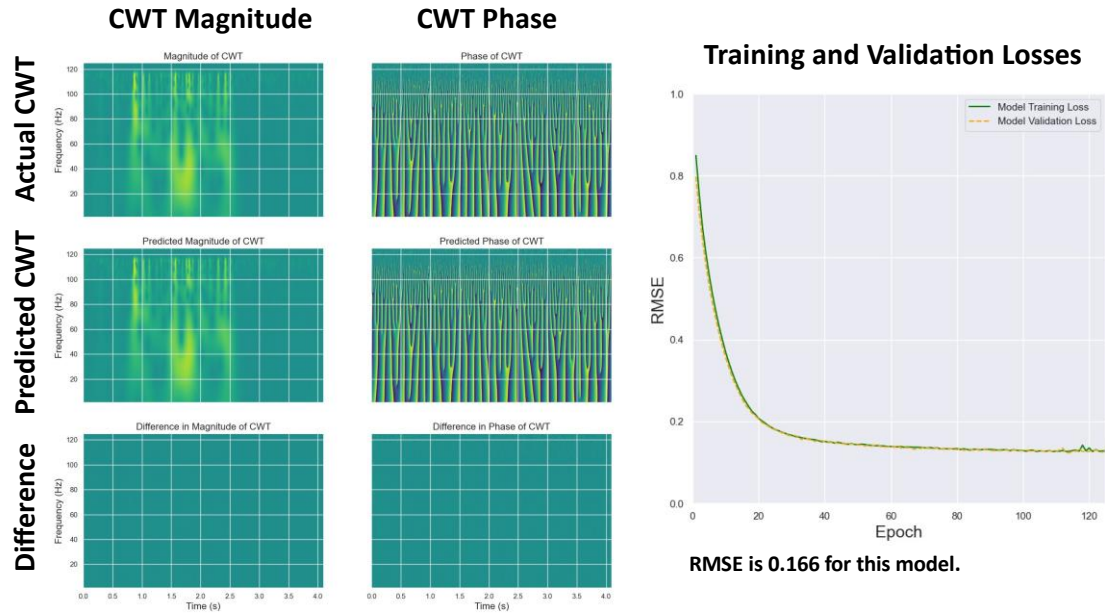
**Figure 36:** The Optuna study used to create the CNN autoencoder for the CWT inversion model. (a) and (b) show that generally a moderately powered network is optimal for an autoencoder with the most important hyperparameters being the max pooling value, network depth, and kernel size as seen in (c). Note that, despite higher values for network depth and max pooling being less optimal, they are generally more preferable as they have a higher compression factor.



**Figure 37:** On the left, the actual CWT transformation (magnitude and phase) at the top versus predicted (reconstructed) CWT via the one-layer autoencoder model at the middle and the difference between them at the bottom. On the right, the training (green) and validation (orange) losses for the model used for the prediction. Note how the differences in both magnitude and phase are very minimal and the RMSE loss is minimal.



**Figure 38:** On the left, the actual CWT transformation (magnitude and phase) at the top versus predicted (reconstructed) CWT via the two-layer autoencoder model at the middle and the difference between them at the bottom. On the right, the training (green) and validation (orange) losses for the model used for the prediction. Note how the difference in and phase is also minimal. Still, the RMSE loss is more than five times higher than the one-layer model.

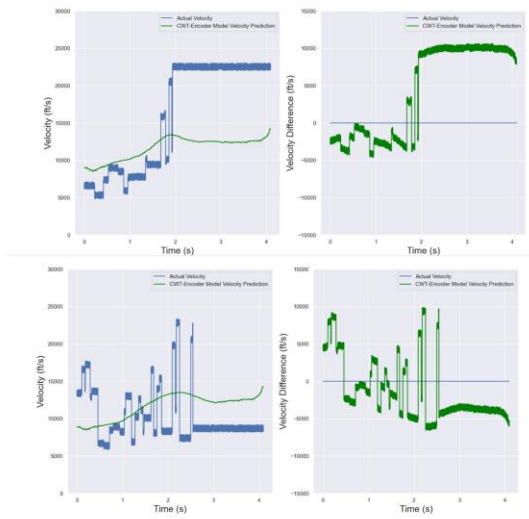


**Figure 39:** On the left, the actual CWT transformation (magnitude and phase) at the top versus predicted (reconstructed) CWT via the three-layer autoencoder model at the middle and the difference between them at the bottom. On the right, the training (green) and validation (orange) losses for the model used for the prediction. Visually the difference is minimal, but it is more than 4 times the two-layer model and 24 times the one-layer model. Thus, it might cause some minor discrepancies in reconstruction.

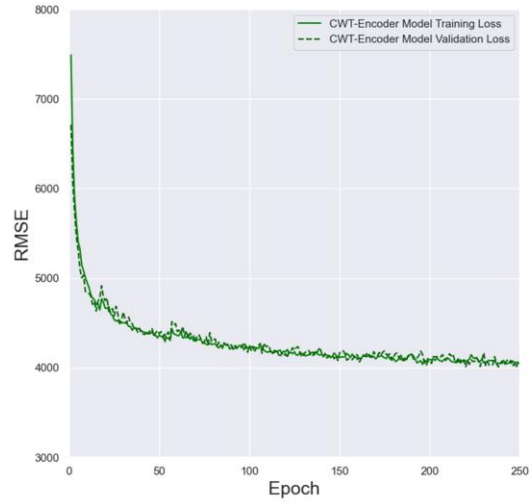
### **3.2.2B Encoder-Inversion CWT Model**

Unlike the previous SWT encoder-inversion model shown in the previous section, this section will directly create and train the model without starting with an Optuna study. This is because the structure of both SWT and CWT data is similar with the latter having an additional component and frequencies within it. Thus, the Optuna study performed for SWT should be enough to get a suitable model to evaluate whether the CWT encoder-inversion model will be able to match the direct inversion models or fail to perform like the SWT encoder-inversion model. Subsequently, a model with the same architecture as in the SWT model was trained and the results seem to suggest that both the SWT and CWT fail to match the performance of the direct inversion method as can be seen in Figure 40.

(a) Blind Test Prediction Results



(b) Training and Validation Loss



**Figure 40:** (a) two blind testing results show that the CWT encoder-inversion model fails at capturing anything but the very low frequency of the model. (b) The training and validation losses show that it also unable to match the performance of the direct inversion models with RMSE values more than double those of the direct models.

## **CHAPTER 4**

### **DISCUSSION**

The previous chapter showed the results of all experiments performed in detail. This chapter will attempt to first summarize key results, and then analyze them to find the best approach as well as the advantages and disadvantages of each addition to the overall goal of inverting corridor stacks for velocity information.

So, the chapter will first start by comparing results of inversion using the direct trace, SWT, and CWT as input. Next, a discussion of when it is appropriate to use the denoising in deep learning models. After that, a discussion on the use of autoencoders for feature extraction and data compression will be done. This chapter will then move to a discussion of the benefits of a consecutive network (split-training) over a traditional network (end-to-end trained). This discussion will pave the way for combining the denoising, feature extraction, and inversion models for one single final model. The chapter will conclude with a discussion regarding the validity of using finite difference data as a blind testing substitute to real datasets.

## 4.1 Direct Inversion versus Wavelet Transformation Inversion

In section 2.2.2, an explanation of wavelet transforms and how they might play a role in improving the inversion process because previous research showed improvements in results when combining wavelet transforms with deep learning models. Thus, the options of inverting for velocity from a trace that underwent SWT and CWT data were explored in section 3.2. The results from both sections showed that the added transformations actually performed substantially worse than the direct inversion attempted in section 3.1. The wavelet transform based models were only able to predict a very low frequency trend of the velocity profile while the direct approach was able to predict the entire velocity profile with all its details to a reasonable degree. Additionally, the increased data size for SWT (11 times) and CWT (250 times) significantly slowed the computational performance of both the training and the predictions compared to the direct approach. Thus, it seems that in this application at least, it is preferable to directly invert for velocity rather than attempt a wavelet transformation first. Table 2 below shows a summary of the relevant results to make the comparison clearer.

**Table 2:** Contains the minimum training and validation losses for all three input types.

| <b>Transformation Type</b>                | <b>Minimum Training Loss</b> | <b>Minimum Validation Loss</b> |
|-------------------------------------------|------------------------------|--------------------------------|
| <b>No Transformation</b>                  | 1492.94                      | 1433.95                        |
| <b>Stationary Wavelet Transform (SWT)</b> | 3913.56                      | 3883.90                        |
| <b>Curvelet Wavelet Transform (CWT)</b>   | 4036.22                      | 4001.87                        |

The reason behind why the wavelet transformations failed to improve the results is not immediately obvious. The link between the actual trace and the velocity profile could be more obvious to the deep learning algorithm as opposed to a frequency decomposed representation of it as is the case for SWT and CWT. While separating the frequency content might be useful in other applications such as filtering or denoising even within deep learning context, it doesn't seem to be useful in the case of inverting for velocity. Therefore, for the final network only the direct inversion approach is considered and recommended, as it is the best in terms of results and computational performance.

## **4.2 Denoising: When is it Appropriate?**

With the inversion model established, it is important to also discuss the additional pre-processing step that were considered prior to inputting the seismic trace. Namely, the step of trace denoising was considered in sections 3.1.2A. For the denoising problem considered, it seems clear that denoising improves the inversion model results across all aspects from training, validation, and blind testing as compared to leaving the issue of denoising to the primary inversion network regardless of if the exact network shape and parameters were used as was done in section 3.1.2C or if only the inversion model was used with the input for the training being the noisy trace as was tested in section 3.1.2A. A summary of all denoising tests can be found in Table 3 below, which shows that, as expected, the inversion of a clean trace performs the best in training and validation losses as well as in blind testing. This is closely followed by the consecutive model approach in which two sub-models, denoising and inversion, are trained individually (split-training)

and then combined for predictions. This is testament to the quality of the denoising done by the dedicated sub-model.

**Table 3:** All relevant testing done for the purpose of determining the best approach for denoising the data.

| <b>Model Name</b>                                                                                                                                                                            | <b>Training Input</b>  | <b>Prediction Input</b> | <b>Training Loss</b> | <b>Validation Loss</b> | <b>Blind Testing</b> |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------|-------------------------|----------------------|------------------------|----------------------|
| <b>Clean Inversion Model</b>                                                                                                                                                                 | Clean Trace            | Clean Trace             | 1310.40              | 1353.68                | 2875.79              |
| <b>Noisy Inversion Model</b>                                                                                                                                                                 | Noisy Trace            | Noisy Trace             | 2399.84              | 2364.69                | 3404.15              |
| <b>Consecutive Inversion Model*</b>                                                                                                                                                          | Noisy +<br>Clean Trace | Noisy Trace             | -                    | -                      | 2950.23              |
| * Contains two sub-models a denoising model that takes a noisy trace, denoise it, and then feeds it into the clean inversion model. Thus, it has no separate training and validation losses. |                        |                         |                      |                        |                      |

### 4.3 Encoders as a Computational Performance Boost

Having reliable and robust inversion results is the most important aspect of this thesis work and its primary aim. At some point, however, one needs to consider the computational performance of said model. For this work, because the data used here is synthetic and relatively small (a seismic trace and its underlying velocity profile), an unoptimized approach can work in a reasonable timeframe. For example, an inversion model with no encoder attached takes one epoch, containing 1,000 datasets, 131 seconds to train and 42 seconds to predict which is acceptable for small scale training and prediction exercises.

However, to scale up the inversion model for larger scale applicability a faster turnaround is required both for training and prediction as the amount of data is significantly larger. Thus, it was demonstrated in 3.1.2D, that an autoencoder is not only useful for denoising as mentioned earlier, but it can also be useful as a tool for feature extraction, data compression, and performance enhancements. An inversion model with an encoder was used to train one epoch as well in 59 seconds (2.2 times faster) and predict in 4 seconds (more than 10 times faster). Such an increase in computational performance is highly desirable especially if it doesn't come at a significant accuracy cost. When training the encoder-inversion model, it was expected that it would perform worse than the standalone inversion model. As can be seen in Table 4, for training and validation losses, this expectation came true. However, for blind testing, despite using 4000 datasets and repeating the test multiple times, the encoder-inversion model always outperformed the standalone inversion model. Despite the  $\sim 200$  RMSE difference in training and validation in favor of the standalone model, the encoder-inversion model was able to outperform the former. A rationale for these results is that the encoder-inversion model feature extraction capabilities are able to extract important features only from the seismic trace which allowed it to feed more relevant information to the inversion portion of the model. Obviously, one cannot rule out the stochastic nature of deep learning, but these results seem to be promising both for enhancing the accuracy of the predictions as well as the computational performance of the model. Thus, it will also be included in the final inversion model alongside the denoising sub model.

**Table 4:** A comparison between the standalone inversion and the encoder-inversion models.

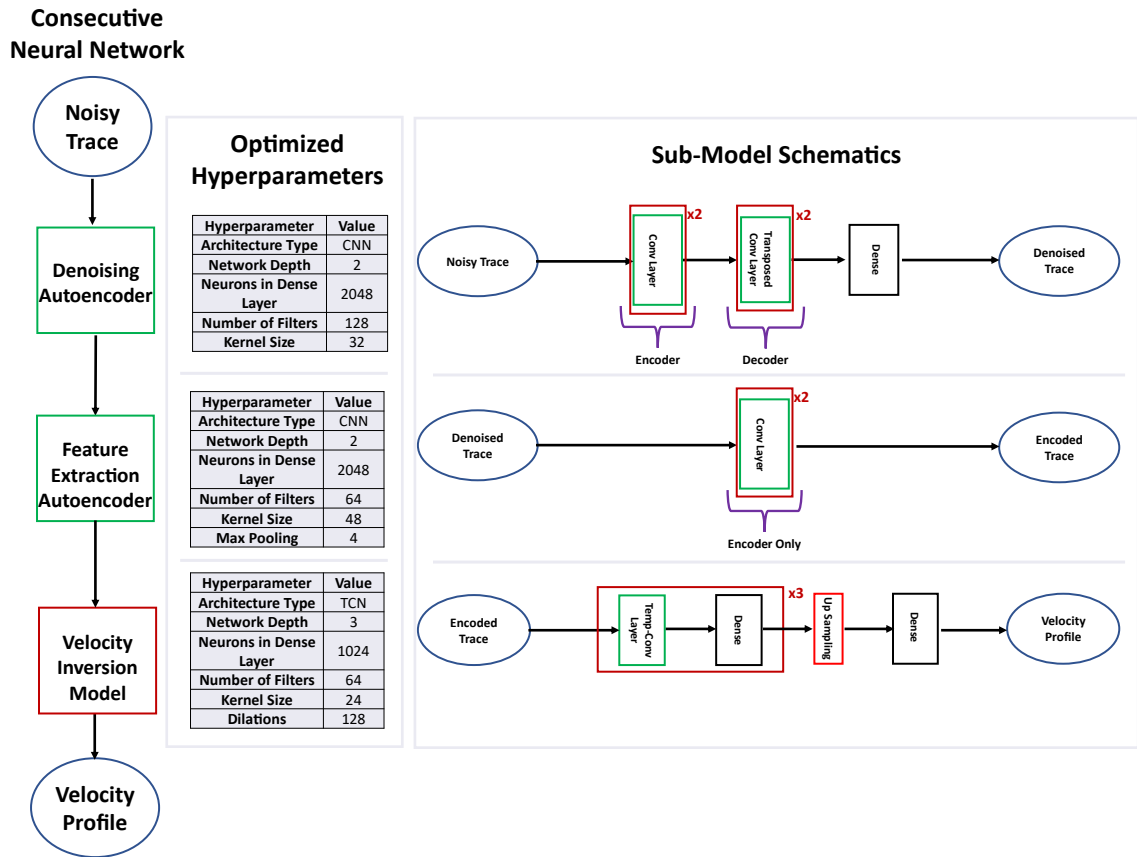
| <b>Model</b>                      | <b>Compression Factor</b> | <b>Minimum Training Loss</b> | <b>Minimum Validation Loss</b> | <b>Blind Testing</b> |
|-----------------------------------|---------------------------|------------------------------|--------------------------------|----------------------|
| <b>Standalone Inversion Model</b> | N/A                       | 1492.94                      | 1433.95                        | 2781.62              |
| <b>Encoder-Inversion Model</b>    | 16                        | 1682.70                      | 1659.27                        | 2146.56              |

#### 4.4 The Argument for Consecutive Networks

With all pieces in place for the final inversion model to take shape, it is important to emphasize why this consecutive network approach (split-training) was taken to begin with and how it may serve as a schematic for future seismic inversion work. The original schematic shown in section 2.4 (Figure 13) was composed of four components; a denoising sub-model, wavelet transformation pre-processing, feature extraction sub-model, and an inversion sub-model. The revised schematic shown in Figure 41 keeps the structure mostly the same with the exception of the wavelet transformation pre-processing step, which was removed due to the poor performance compared to the direct approach.

This approach of split-training was chosen for several reasons such as modularity and explain-ability. The first aspect is the modularity of this approach, current understanding leans towards autoencoders being excellent for denoising and feature extraction applications as shown earlier. However, if a better architecture becomes available in the future, the fact that the components are separate allows for replacing the denoising sub-

model and retaining the rest of the architecture without any additional changes to the rest of the network. Additionally, because the models are separate, one is able to use them for other applications if needed easily. The third and final point is regarding the explain-ability of this approach compared to an end-to-end trained approach. Deep learning models are often criticized as being quite opaque or a black-box and this is mainly because of the difficulty of relating to how a model made the jump from the initial input to the prediction. When it fails, it is often harder to debug, especially for larger models that attempt to do a significant leap from input to prediction. The modular approach allows for optimizing each step of the process, with its own optimal hyperparameters. It also allows for the effect of the model to be clearly observed and corrected if it is performing sub-optimally. For example, one can observe how accurately the denoising model is able to denoise the trace, or how the inversion model is able to perform with a clean trace versus a denoised trace as an input. The benefits are clear with the primary drawback being the additional time needed to train the separate models rather than one training regime for a combined model. This however can also be argued against because in the long-term having modular models allows for a wider application and more reusability.



**Figure 41:** The final revised schematic for the proposed consecutive (split-trained) neural network. It is composed of three main component a denoising, feature extraction & encoding (compression), and velocity inversion sub-models. The input is a noisy seismic trace, and the output is the velocity profile at the same location.

## 4.5 Finite Difference Data and How it Relates to Real Data

The ultimate test to any new approach or methodology is to test it on what it is eventually going to be used for. In the case of this consecutive network and the proposed inversion approach, the final test would have been ideally performed on a real dataset. Unfortunately, open-access to real VSP datasets is often very difficult to get considering that they're usually classified as well logs which are highly confidential. In the future and if the necessary permissions obtained, this model will be tested on real datasets and the results of that will be published to give a definitive answer on whether it is going to be useful for real-world applications.

Nonetheless, a robust approximation for simulating real seismic data exists which is finite difference modelling. It's often used as precursor to any field acquisition in order to understand and optimize the survey design. It allows for simulating the wave propagation, setting up the survey geometry, and then processing the resulting synthetic datasets in a way that closely mimics real datasets. This approximation was used in section 3.1.5 in order to perform a final blind testing for the consecutive network, one in which the synthetic data generation engine which was used through the entire process prior was not used here. This allowed for a more accurate testing of the network capabilities and the results showed that the network was able to predict the velocities reasonably well. Thus, proving that its performance is not limited to datasets generated by the training engine and that its performance encompasses datasets from more advanced generation algorithms.

## CHAPTER 5

### CONCLUSIONS

This final chapter will first attempt to summarize the most important aspects of this thesis as well as provide final remarks and conclusions. Then, it will end with few potential avenues for future work that might further improve or expand the work completed here.

#### 5.1 Summary and Remarks

To summarize, one needs to go back to the primary aim of this work which was to invert for a velocity profile from a VSP corridor stack (seismic trace). To reach that aim, an efficient and realistic data generation engine was created, pre-processing workflow that includes denoising, scaling, and wavelet transformation was explored, and finally several potential deep learning architectures were tested.

The most promising approach was that of a consecutive network (split-trained) which consisted of three sub-models performing the following tasks; denoising (1-layer encoder CNN), feature extraction and compression (2-layers encoder CNN), and finally velocity inversion (3-layers TCN). This approach performed better than all other alternatives explored such as using a simpler model for the entire process, wavelet transformation (SWT or CWT) as an intermediate pre-processing step, as well as all other different combinations and hyperparameter optimizations that were tested throughout this thesis.

Other side tests were performed to further understand the capabilities of this model such as inverting for acoustic impedance and training the model via transfer learning or making an initial attempt at a joint inversion of velocity and density.

## **5.2 Future Work**

While the results of this thesis primary aim were excellent, there are several avenues for both expanding and improving this work. Firstly, and most importantly, the first area that needs to be looked at, is attempting to replicate the success of this work on real field data. This could encompass either some modifications to the proposed schematic or data generation and augmentation used in training and validating the models. Secondly, the issue of joint inversion of several properties, such as density and velocity, was briefly attempted here, the results showed some promising results but there is still a significant room for improvement. For example, one might need to look at weighing the different properties differently or changing the objective function in order to allow for better prediction of both properties. Finally, while the results of the wavelet transformation were disappointing for inversion, one could attempt additional pre-processing steps to see if it would improve the results. For example, one could perform the transformation on the target (velocity) in addition to the input (seismic trace) which might allow the model to be able to connect each frequency with its corresponding trend of velocity. With these potential aspects to tackle and the previous conclusions, this work is concluded.

# Appendix

## A1. Computational Environment

This section of the appendix will briefly cover the computational environment that was used to write and run the code. It will be split into hardware and software sections.

- **Hardware Environment**

All the codes, plots, and results were created on an Asus Zephyrus G14 GA401QM (2021) laptop with the below specifications.

| Item             | Specification                        |
|------------------|--------------------------------------|
| Processor        | AMD Ryzen™ 9 5900HS Mobile Processor |
| Graphics         | NVIDIA® GeForce RTX™ 3060 Laptop GPU |
| GPU Memory       | 6GB GDDR6                            |
| RAM Memory       | 32GB DDR4                            |
| Storage Capacity | 4TB                                  |

- **Software Environment**

The operating system used was Windows 11 (kept up-to-date) with the main integrated development environment (IDE) in use throughout being Pycharm 2021-2022. The entirety of the code was written for Python 3.9 and was accelerated using CUDA V11.7.64. The

following libraries with their respective versions necessary for the environment setup are shown below.

| Library Purpose   |         |                  |         |               |         |          |         |
|-------------------|---------|------------------|---------|---------------|---------|----------|---------|
| Computational     |         | Graphical        |         | Deep Learning |         | Other    |         |
| Library           | Version | Library          | Version | Library       | Version | Library  | Version |
| bruges            | 0.5.4   | matplotlib       | 3.7.0   | keras         | 2.10.0  | ipython  | 8.10.0  |
| numpy             | 1.24.2  | optuna-dashboard | 0.9.0   | keras-tcn     | 3.5.0   | nptyping | 2.4.1   |
| pandas            | 1.5.3   | seaborn          | 0.12.2  | optuna        | 3.1.0   |          |         |
| PyWavelets (pywt) | 1.4.1   |                  |         | scikit-learn  | 1.2.1   |          |         |
| scipy             | 1.10.0  |                  |         | tensorflow    | 2.10.0  |          |         |

## A2. Code High-Level Walkthrough

In this section, a high-level summary of the different files and scripts composing the code that was written for this thesis will be attempted. The code is divided into five primary categories with the first four being primarily composed of classes, methods, and functions to support the last category. The first four categories are codes written for data generation, pre-processing, plotting, and deep learning while the last category deals with calling and using the previous four in order to create, run, update, save, and plot the deep learning models and their results.

The entire code is available at the private GitHub repository (will be made public at a later date): <https://github.com/Hussain747715/pythonLookaheadDeepLearningVSP>

- **Data Generation**

All the synthetic data generation and augmentation for this thesis is done inside the dataGeneration.py file. The most important portion of this file is the DataGenerationClass class which is responsible for creating random lithological sequences, adding layers based on appropriate lithologies for each sequence. These layers are then filled with appropriate velocity and density values based on empirical relationships for each lithology. Next, the acoustic impedance and subsequently the reflectivity series are calculated. Finally, a wavelet is convolved with the reflectivity series in addition to some noise to produce the synthetic corridor stack (seismic trace). Below is the header of that class with the initialization parameters that can be modified.

```
class DataGenerationClass:
    """
    A class that deals with data generation for the lookahead project. It generates synthetic reflectivity series that
    can be convolved with certain wavelets, noise and gaps can be added. It can also return the velocity and density of
    the model that was used to create the datasets.
    """

    # Hussain *
    def __init__(self, variable_dt: int = 0, variable_data_length: int = 0, constant_data_length: int = 4096,
                  density_empirical: int = 1, convolve_with_wavelets: int = 1, add_noise: int = 1, add_gaps: int = 1,
                  save_data_to_file: int = 1, save_data_as_txt: int = 0, plotting_option: int = 0,
                  path_to_save: str = 'Data/', run_number: int = 1, t0: float = 0.0):
        """
        Initializer for the dataset generator class.
        Arguments:
            variable_dt (int): Set to either variable (1) or constant (0) sampling rate.
            variable_data_length (int): Set to either variable (1) or constant (0) data length .
            constant_data_length (int): If variable_data_length=0, sets the value for the constant data length.
            density_empirical (int): Whether to use random (0) or empirical (1) relationship for calculating density.
            convolve_with_wavelets (int): Whether to convolve reflectivity with wavelets (1) or keep it as is (0).
            add_noise (int): Whether to add gaussian noise to the data (1) or keep it as is (0).
            add_gaps (int): Whether to add gaps to the data (1) or keep it as is (0).
            save_data_to_file (int): Whether to save the data (1) or not (0).
            save_data_as_txt (int): Whether to save the data as a .txt file (1) or .npy file (0).
            plotting_option (int): Whether to plot (1) the generated model & convolved datasets or not (0).
            path_to_save (str): Path to save the generated data if save_data_to_file=1.
            run_number (int): Number of datasets to generate.
            t0 (float): Start time of the file, default is 0.0.
        """
```

Other functions that are contained in this file are wavelet\_convolution, data\_noise, data\_gaps, data\_average, save\_data, and scale\_range. These functions are mainly designed

to support the DataGenerationClass. For example, the wavelet\_convolution function takes a reflectivity series and convolve it with a given wavelet. The data\_noise, data\_gaps, and data\_average functions also perform supporting calculations for the primary class. The data\_noise simply adds gaussian noise to any given time series. The data\_gaps and data\_average functions complement each other in which the first simulates a missing section of the data by creating a gap in the time series (corridor stack). On the other hand, the second function averages the velocity and densities in this gap interval as the only known velocity in this missing interval is the average velocity. The save\_data function simply saves the synthetic model properties as well as all generated seismic data in either a .txt or .npy formats if one desires to load them from storage later. The final function scale\_range performs Min-Max scaling on any given input. See below code snippets of all six supporting functions.

```
def wavelet_convolution(wavelet_type: str, data_to_conv: NDArray, dt_wavelet: float):  
    """  
    Convolve dataset with a wavelet and returns the convolved dataset.  
    Arguments:  
        wavelet_type (str): Determines the wavelet type. It can be either Ricker, Klauder, Ormsby, or All if all three  
        types are desired.  
        data_to_conv (NDArray): the dataset that needs to be convolved. Expects an NDArray.  
        dt_wavelet (float): Sampling rate of wavelet. Should be the same as data.  
    Returns:  
        A single convolved dataset if either wavelet_type == Ricker, Klauder, or Ormsby.  
        Otherwise, returns the data convolved with all three wavelets if wavelet_type == All.  
    """
```

```
def data_noise(data_to_noise: NDArray) -> NDArray:  
    """  
    Adds random noise to dataset.  
    Arguments:  
        data_to_noise (NDArray): the dataset that noise will be added to. Expects an NDArray.  
    Returns:  
        A dataset with the noise added.  
    """
```

```
def data_gaps(data_to_gap: NDArray, top_mute: int = 0, bottom_mute: int = 0) -> {NDArray, int, int}:
    """
    Creates a random gap in the dataset.
    Arguments:
        data_to_gap (NDArray): the dataset that gap will be added to.
        top_mute (int): the upper limit of the mute.
        bottom_mute (int): the lower limit of the mute.
    Returns:
        A dataset with the gap added.
    """
```

```
def data_average(data_to_average: NDArray, top_mute: int, bottom_mute: int) -> NDArray:
    """
    Averages the dataset in the gap section of the dataset.
    Arguments:
        data_to_average (NDArray): the dataset that the gap will be averaged.
        top_mute (int): the upper limit of the mute.
        bottom_mute (int): the lower limit of the mute.
    Returns:
        A dataset with the gap averaged.
    """
```

```
def save_data(save_folder_path: str, save_file_name: str, data_to_save: NDArray, precision: Union[str, Iterable, None],
              save_as_txt: int):
    """
    Saves dataset to the specified location as either txt or npy file.
    Arguments:
        save_folder_path (str): Folder path for the file to be saved.
        save_file_name (str): The file name for the file to be saved.
        data_to_save (NDArray): The data to be saved.
        precision: The desired precision of the dat to be saved.
        save_as_txt (int): Saves the dataset as either txt (1) or npy (0).
    Returns:
        No returns, saves files to desired location.
    """
```

```
def scale_range(data_to_scale: NDArray, min_data_value: float, max_data_value: float) -> NDArray:
    """
    Scales dataset using min-max scaling.
    Arguments:
        data_to_scale (NDArray): The data to be scaled.
        min_data_value (float): min value to be used in min-max scaling.
        max_data_value (float): max value to be used in min-max scaling.
    Returns:
        Scaled dataset.
    """
```

- **Data Pre-Processing**

Similar to the way the data generation functionality was handled, one file is primarily responsible for the data pre-processing prior to input to deep learning models which is dataPreProcessing.py file. This file is composed of two classes, DataPreProcessingClass and CustomDataGen. The DataPreProcessingClass is responsible for pre-processing the

data if it was saved to storage in the data generation step. This file is less developed compared to the CustomDataGen class as for the thesis work, the data was generated on the fly rather than saved to storage. As implied earlier, CustomDataGen class is responsible for using the DataGenerationClass to create datasets and feeding them directly to the deep learning models without saving them to disk first. Snippets of the two classes are shown below.

```
class DataPreProcessingClass:
    """
    A class that handles the data loading and pre-processing steps to the generated data in preparation for ML applications.
    """

    # Hussain *
    def __init__(self, load_txt: int = 0, load_dens: int = 0, exclude_base_refl: int = 1, folder_path: str = 'Data/'):
        """
        Initializer for the dataset generator class.
        Arguments:
            load_txt (int): Set to either load .txt format (1) or .npz format (0).
            load_dens (int): Set to either load density (1) or not (0).
            exclude_base_refl (int): Set to either load reflectivity series (1) or not (0).
            folder_path (str): Path to load data from.
        """
```

```
class CustomDataGen(Sequence):
    """
    Custom data generator for machine learning applications. Generates data on the fly.
    """

    # Hussain *
    def __init__(self, batch_size: int = 1, n: int = 10, gen_type: str = 'AE', transform_type: str = 'NA',
                  noisy_date: bool = True, scaling_flag: bool = False):
        """
        Initializer for the dataset generator class.
        Arguments:
            batch_size (int): The number of datasets to use in each batch for training, default is 1.
            n (int): total number of datasets to generate for each epoch of training.
            gen_type (str): Type of generator to be used which depends on the application type, can be 'ML_VEL',
                           'ML_IMP', 'AE', 'Spiking', or 'VEL_DENS_JOINT'.
            transform_type (str): Type of transformation to be applied on the data, can be 'NA', 'SWT', or 'CWT'.
            noisy_date (bool): Whether to add noise to the generated data or not.
            scaling_flag (bool): Whether to apply scaling to the generated data or not.
        """
```

These two classes are supported by 12 other functions available in this file. They can be classified into three categories; data organization (2 functions), data manipulation (6 functions), and custom loss functions (4 functions). The two data organizations functions

are `create_refl_dataframe` and `directory_create`. The first is responsible for organizing input and output data into a Pandas dataframe structure while the second simply creates a directory if that directory does not exist. Code snippets are available below.

```
def create_refl_dataframe(time: NDArray, refl_data: NDArray, vel_data: NDArray, dens_data: NDArray = np.zeros(1),
                        data_id: int = 0) -> pd.DataFrame:
    """
    Function that creates a pandas dataframe containing id, time, reflectivity, velocity, and density (optional).
    Arguments:
        time (NDArray): Array of time values.
        refl_data (NDArray): Reflectivity dataset.
        vel_data (NDArray): Velocity dataset.
        dens_data (NDArray) (opt): Velocity dataset.
        data_id (int): ID.
    Returns:
        A dataframe containing id, time, reflectivity, velocity, and density (optional).
    """
```

```
def directory_create(dir_path: str):
    """
    If folder doesn't exist, then create it.
    """
```

The data manipulation section primarily deals with data normalization and the use of various wavelet transformations. The `normalize_data` function applies Min-Max scaling on the input data. The `mra_data` and `cwt_data` functions apply discrete / stationary and continuous wavelet transformations respectively. The `decompose_complex_cwt`, `magn_complex_cwt`, and `phase_complex_cwt` functions further manipulate CWT data via decomposing the complex CWT into magnitude and phase components. See code snippets for all 6 functions below.

```
def normalize_data(data: NDArray, min_data: float, max_data: float, min_range: float = 0, max_range: float = 1) -> \
    {NDArray, float}:
    """
    Applies data normalization on input data.
    Arguments:
        data (NDArray): Dataset where normalization will be applied.
        min_data (float): Min value to use in normalization.
        max_data (float): Max value to use in normalization.
        min_range (float): Min value to normalize data into.
        max_range (float): Max value to normalize data into.
    Returns:
        Returns the normalized data and the scaler.
    """
```

```
def mra_data(data: NDArray, wavelet_type: str = 'haar', trans_type: str = 'dwt') -> NDArray:
    """
    Computes the MRA for a given dataset.
    Arguments:
        data (NDArray): Dataset where MRA will be applied.
        wavelet_type (str): Type of wavelet to use in MRA.
        trans_type (str): Transformation type can be either 'dwt' or 'swt'.
    Returns:
        Returns the coefficients of the MRA.
    """
```

```
def cwt_data(time: NDArray, data: NDArray, w_min: int = 1, w_max: int = 125, returned_output: str = 'cwt',
            plot: bool = False):
    """
    Applies CWT transform to data.
    Arguments:
        time (NDArray): Array of time values.
        data (NDArray): Dataset where CWT will be applied.
        w_min (int): Minimum width.
        w_max (int): Maximum width.
        returned_output (str): Will define returns can be 'cwt', 'complex', or 'magnitude'.
        plot (bool): Determines whether to plot CWT results or not
    Returns:
        Returns CWT results based on returned_output parameter.
    """
```

```
def decompose_complex_cwt(cwt_trans) -> {NDArray, NDArray}:
    """
    Decomposes complex CWT.
    Arguments:
        cwt_trans: Complex CWT.
    Returns:
        Returns the real and imaginary parts of CWT.
    """
```

```
def magn_complex_cwt(cwt_trans) -> NDArray:
    """
    Retrieves the absolute magnitude of complex CWT.
    Arguments:
        cwt_trans: Complex CWT.
    Returns:
        Returns the absolute magnitude of CWT.
    """
```

```
def phase_complex_cwt(cwt_trans) -> NDArray:
    """
    Retrieves the absolute magnitude of complex CWT.
    Arguments:
        cwt_trans: Complex CWT.
    Returns:
        Returns the phase of CWT.
    """
```

The final category with the custom loss functions was explored in order to improve the performance of the model. Four custom loss functions were tested, RMSE

(root\_mean\_squared\_error), Log(RMSE) (log\_root\_mean\_squared\_error), correlation (correlation\_loss), and a combination of RMSE and correlation (mse\_correlation\_loss). These functions were not fully developed except for root\_mean\_squared\_error which was used as the primary metric for this thesis work.

- **Data Plotting**

The next file dataPlotting.py has many of the underlying basic plotting functions designed for this thesis. Some of the plots were generated using secondary files such as HPO\_Plotting.py and ThesisPlotting.py. These two secondary files are unstructured, provided with minimal commenting, and are hardcoded to load models and their objective functions. The dataPlotting.py contains 14 functions below is a brief description of which plot is generated by which function and code snippets of the functions.

| Function Name                | Purpose                                                                                  |
|------------------------------|------------------------------------------------------------------------------------------|
| <b>font_sizes_defaults</b>   | Loads default font sizes for plots (used in other plots).                                |
| <b>max_axis_value</b>        | Determines appropriate max axis value.                                                   |
| <b>plot_model</b>            | Plots lithology, velocity, density, and acoustic impedance in depth.                     |
| <b>plot_xplot</b>            | Plots a crossplot of velocity versus density colored by lithology or acoustic impedance. |
| <b>plot_refl</b>             | Plots a reflectivity series, convolved data, noisy data, and data with gap.              |
| <b>plot_refl_compare</b>     | Plots a comparison between data convolved with the three wavelet options.                |
| <b>plot_cwt_real_imag</b>    | Plots the real and imaginary components of a CWT dataset.                                |
| <b>plot_cwt_magn</b>         | Plots the magnitude of a CWT dataset.                                                    |
| <b>plot_cwt_magn_phase</b>   | Plots the magnitude and phase of a CWT dataset.                                          |
| <b>plot_mra</b>              | Plots a DWT/SWT dataset.                                                                 |
| <b>plot_ae_predict_raw</b>   | Plots the actual and reconstructed trace.                                                |
| <b>plot_ae_mra_predict</b>   | Plots the actual and reconstructed DWT/SWT dataset.                                      |
| <b>plot_cwt_magn_predict</b> | Plots the actual and reconstructed CWT dataset.                                          |
| <b>PlotLearning</b>          | Plots the objective function and learning rate at epoch end.                             |

```
def font_sizes_defaults(small_size: int, medium_size: int, bigger_size: int):
    """
    Function that handles font sizes for plots.
    Arguments:
        small_size (int): value for small font size.
        medium_size (int): value for medium font size.
        bigger_size (int): value for large font size.
    Returns:
        None.
    """
```

```
def max_axis_value(data) -> float:
    """
    Function that retrieves the absolute maximum value of a dataset.
    Arguments:
        data: dataset to retireve max value.
    Returns:
        Absolute maximum value of the dataset.
    """
```

```
def plot_model(lithology_type: list[int], depth_layer: list[int], vel_layer: list[int], dens_layer: list[int],
               imp_layer: list[int]):
    """
    Function that plots lithology, velocity, density, and impedance in depth.
    Arguments:
        lithology_type (list[int]): List of lithologies.
        depth_layer (list[int]): List of depths.
        vel_layer (list[int]): List of velocities.
        dens_layer (list[int]): List of densities.
        imp_layer (list[int]): List of impedance values.
    Returns:
        None, plots the model.
    """
```

```
def plot_xplot(lith_time: NDArray, vel_time: NDArray, dens_time: NDArray, imp_time: NDArray = (-999 * np.ones(1))):
    """
    Function that crossplots the velocity with the density colored by lithology or impedance if supplied.
    Arguments:
        lith_time (NDArray): Lithology array in time.
        vel_time (NDArray): Velocity array in time.
        dens_time (NDArray): Density array in time.
        imp_time (NDArray) (opt): Impedance array in time, optional.
    Returns:
        None, crossplots the model.
    """
```

```
def plot_refl(t: NDArray, refl_time: NDArray, refl_time_wavelet: NDArray, refl_time_wavelet_noise: NDArray,
              refl_time_wavelet_noise_gaps: NDArray, title_comment: str = ''):
    """
    Function that plots the different stages of reflectivity series creation in time.
    Arguments:
        t (NDArray): Array of time values.
        refl_time (NDArray): Raw reflectivity series.
        refl_time_wavelet (NDArray): Convolved reflectivity series.
        refl_time_wavelet_noise (NDArray): Reflectivity series with noise added.
        refl_time_wavelet_noise_gaps (NDArray): Reflectivity series with noise and gaps added.
        title_comment (str): String to add to the title of the plot.
    Returns:
        None, plots the different stages of reflectivity series.
    """
```

```
def plot_refl_compare(t, refl_time, refl_time_ricker, refl_time_klauder, refl_time_ormsby):
    """
    Function that plots the different the reflectivity series with different wavelets convolved.
    Arguments:
        t (NDArray): Array of time values.
        refl_time (NDArray): Raw reflectivity series.
        refl_time_ricker (NDArray): Convolved reflectivity series with ricker wavelet.
        refl_time_klauder (NDArray): Convolved reflectivity series with klauder wavelet.
        refl_time_ormsby (NDArray): Convolved reflectivity series with ormsby wavelet.
    Returns:
        None, plots the reflectivity with different wavelets.
    """
```

```
def plot_cwt_real_imag(t, real, imag, w_min=1, w_max=125):
    """
    Function that plots the real and imaginary components of CWT.
    Arguments:
        t: Array of time values.
        real: Real part of CWT.
        imag: Imaginary part of CWT.
        w_min: Minimum width.
        w_max: Maximum width.
    Returns:
        None, plots the real and imaginary parts of CWT.
    """
```

```
def plot_cwt_magn(t, data, magn, w_min=1, w_max=125, title: str = '', max_amp: float = 0):
    """
    Function that plots the magnitude of the CWT.
    Arguments:
        t: Array of time values.
        data: Dataset.
        magn: Absolute magnitude of CWT.
        w_min: Minimum width.
        w_max: Maximum width.
        title: Plot title.
        max_amp: Maximum amplitude for plotting default is calculated from data if none is provided.
    Returns:
        None, plots the magnitude of the CWT.
    """
```

```
def plot_cwt_magn_phase(t, data, magn, phase, w_min=1, w_max=125, title: str = '', max_amp: float = 0):
    """
    Function that plots the magnitude of the CWT.
    Arguments:
        t: Array of time values.
        data: Dataset.
        magn: Absolute magnitude of CWT.
        phase: Phase of CWT.
        w_min: Minimum width.
        w_max: Maximum width.
        title: Plot title.
        max_amp: Maximum amplitude for plotting default is calculated from data if none is provided.
    Returns:
        None, plots the magnitude of the CWT.
    """
```

```
def plot_mra(t, data, all_coeffs_mra, title: str = '', max_amp: float = 0):
    """
    Function that plots all MRA coefficients.
    Arguments:
        t: Array of time values.
        data: Dataset.
        all_coeffs_mra: Coefficients of the MRA.
        title: Plot title.
        max_amp: Maximum amplitude for plotting default is calculated from data if none is provided.
    Returns:
        None, plots the coefficients of the MRA.
    """
```

```
def plot_ae_predict_raw(x: NDArray, x_hat: NDArray, save_path: str = ''):
    """
    Function that plots actual and predicted (reconstructed) AE result.
    Arguments:
        x: Actual series.
        x_hat: Predicted series.
        save_path: path to save file, leave empty if file saving is not desired.
    Returns:
        None, plots actual and predicted (reconstructed) AE result.
    """
```

```
def plot_ae_mra_predict(t, data, mra_actual, mra_pred):
    """
    Function that plots all MRA coefficients.
    Arguments:
        t: Array of time values.
        data: Dataset.
        mra_actual: Actual coefficients of the MRA.
        mra_pred: Predicted coefficients of MRA.
    Returns:
        None, plots the coefficients of the MRA.
    """
```

```
def plot_cwt_magn_predict(t, data, magn_actual, magn_pred, w_min=1, w_max=125):
    """
    Function that plots the magnitude of the CWT.
    Arguments:
        t: Array of time values.
        data: Dataset.
        magn_actual: Actual Absolute magnitude of CWT.
        magn_pred: Predicted Absolute magnitude of CWT.
        w_min: Minimum width.
        w_max: Maximum width.
    Returns:
        None, plots the magnitude of the CWT.
    """
```

```
class PlotLearning(keras.callbacks.Callback):
    """
     Callback to plot the objective function and learning curves of the model during training.
    """
```

- **Base Deep Learning Model**

The baseML.py is composed of 3 classes and 4 functions. The classes are BaseMLClass, Dwell\_CB, and CLR\_CB. The first one is the one responsible for constructing, training, and predicting deep learning models while the latter two are custom callbacks related to implementing a dynamic learning rate while training. The four available functions are create\_model, run\_model, init\_results\_file, and append\_results\_to\_file. The first two functions use the BaseMLClass to create and train deep learning models while the latter two are used within BaseMLClass to create .csv files to store training history data. Below are code snippets of the 3 classes and 4 functions.

```
class BaseMLClass:
    """
    A class that deals with data machine learning models for the lookahead project.
    """

    def __init__(self, trainingDataGen: Sequence, validationDataGen: Sequence, num_ml_layers: int = 2,
                  dnn_units: int = 1000, lstm_units: int = 10, num_filters: int = 64, kernel_size: int = 3,
                  max_dilation: int = 32, pooling_value: int = 1, dropout_rate: float = 0, activation_func: str = 'swish',
                  optimizer_func: optimizers = optimizers.Adam(learning_rate=0.003, amsgrad=True),
                  regularizer_func: regularizers = regularizers.L1L2(l1=1e-5, l2=1e-4), monitor_metric: str = 'val_loss',
                  loss_func='MSE', num_epochs: int = 10, num_workers: int = 1, verbose_val: int = 1,
                  cbs: callbacks = None, use_advanced_lr: bool = True, use_encoder: bool = False, encoder_path: str = '',
                  load_custom_objects=None, disp_loss: bool = True, model_name: str = 'Model_',
                  path_to_save: str = 'Base_ML_Model\\'):

        """
        Initializer for the dataset generator class.
        Arguments:
            trainingDataGen (generator): Data generator for training.
            validationDataGen (generator): Data generator for validation.
            num_ml_layers (int): How many layers in encoder/decoder (ons side).
            num_filters (int): Number of filters in Conv1D/TCN layers.
            dnn_units (int): number of neurons in a dense layer.
            lstm_units (int): number of neurons in a LSTM layer.
            kernel_size (int): Kernel size for Conv1D/TCN layers.
            max_dilation (int): Maximum dilation number.
            activation_func (str): Activation function for Conv1D layers.
            pooling_value (int): Determine value of MaxPooling1D and UpSampling1D.
            dropout_rate (float): Determine rate of dropout in dropout layer.
            optimizer_func (keras.optimizer): Optimizer function for the CNN network.
            monitor_metric (str): Monitor metric.
            loss_func (str): Loss function for the ML network.
            num_epochs (int): Epoch number for training/validation.
            num_workers (int): number of workers for training if set to 1 swish to not use multiprocessing.
            verbose_val (int): Whether to be verbose (1) or not (0).
            cbs (callbacks): For non-default callbacks.
            use_advanced_lr (bool): Whether to use dwell callback.
            use_encoder (bool): Whether to add an encoder to the beginning of a model.
            encoder_path (str): Path to encoder.
            load_custom_objects: Loads custom objects for loaded models.
            disp_loss (bool): whether to display loss or not.
            model_name (str): Name of model used for saving.
            path_to_save (str): Path to save all model related files to.

        Returns:
            None.
        """
```

```
# Dwell_CB & CLR_CB are custom keras callbacks to try to make the learning rate more dynamic and improve the training.

> class Dwell_CB(keras.callbacks.Callback):...

> class CLR_CB(keras.callbacks.Callback):...
```

```
def create_model(path_to_save: str, trainingDataGen, validationDataGen, num_epochs: int, arch_type: str, num_lyr: int,
                 num_dnn: int, num_dil: int, num_filt: int, krnl_size: int, pool_val: int, num_lstm: int = 10,
                 learning_rate: float = 0.001, new_model: bool = True, network_type: str = 'SEQ',
                 model_update_path: str = '', custom_objects=None, use_encoder: bool = False, encoder_path: str = ''):
    """
    Function that handles the creation of deep learning models based on the BaseML Class.
    Arguments:
        path_to_save (str): the path to the directory to save the created model.
        trainingDataGen (customKerasGenerator): a custom Keras generator that is used to generate the training datasets.
        validationDataGen (customKerasGenerator): a custom Keras generator that is used to generate the validation datasets.
        num_epochs (int): number of epochs that will be used for training
        arch_type (str): the architecture of the model to be created, can be DNN, CNN, CNN_VAR, CNN_LSTM, or TCN.
        num_lyr (int): number of layers for the network, for SEQ networks it is the same as the input, for AE networks, the actual value is double.
        num_dnn (int): number of neurons in the dense layers of the network.
        num_dil (int): Value of max dilational value for TCNs.
        num_filt (int): Number of filters for convolutional layers.
        krnl_size (int): Kernel size for convolutional layers.
        pool_val (int): For AE networks, max pooling value for TCNs and stride for CNNs.
        num_lstm (int): Number of units in LSTM or CNN_LSTM networks.
        learning_rate (float): The initial learning rate for the network training.
        new_model (bool): Check as true if this is a model that is trained from scratch or False if it is a model that is loaded from a directory.
        network_type (str): The type of network being used either SEQ for sequential or AE for autoencoders.
        model_update_path (str): The path of model to be loaded if network_type is equal to False.
        custom_objects (customKerasObjects): CustomObjects to be loaded alongside the loaded model if network_type is equal to False.
        use_encoder (bool): Whether to attach an encoder at the beginning of a SEQ network. Only works if arch_type is SEQ_ENCODER.
        encoder_path (str): The path to the encoder to be loaded if use_encoder is set to True.
    Returns:
        model: The created model.
        model_label: The model name.
    """
```

```
def run_model(path_to_save: str, trainingDataGen, validationDataGen, x, y, num_epochs: int, arch_type: str,
              num_lyr: int,
              num_dnn: int, num_dil: int, num_filt: int, krnl_size: int, pool_val: int, num_lstm: int = 10,
              learning_rate: float = 0.001, plot_fig: bool = True, new_model: bool = True, network_type: str = 'SEQ',
              model_update_path: str = '', custom_objects=None, save_encoder: bool = False, use_encoder: bool = False,
              encoder_path: str = ''):
    """
    Function that handles the creation of deep learning models based on the BaseML Class.
    Arguments:
        path_to_save (str): the path to the directory to save the created model.
        trainingDataGen (customKerasGenerator): a custom Keras generator that is used to generate the training datasets.
        validationDataGen (customKerasGenerator): a custom Keras generator that is used to generate the validation datasets.
        x: A single blind testing input dataset that can be plotted if plot_fig is set to True.
        y: A single blind testing output dataset that can be plotted if plot_fig is set to True.
        num_epochs (int): number of epochs that will be used for training
        arch_type (str): the architecture of the model to be created, can be DNN, CNN, CNN_VAR, CNN_LSTM, or TCN.
        num_lyr (int): number of layers for the network, for SEQ networks it is the same as the input, for AE networks, the actual value is double.
        num_dnn (int): number of neurons in the dense layers of the network.
        num_dil (int): Value of max dilational value for TCNs.
        num_filt (int): Number of filters for convolutional layers.
        krnl_size (int): Kernel size for convolutional layers.
        pool_val (int): For AE networks, max pooling value for TCNs and stride for CNNs.
        num_lstm (int): Number of units in LSTM or CNN_LSTM networks.
        learning_rate (float): The initial learning rate for the network training.
        new_model (bool): Check as true if this is a model that is trained from scratch or False if it is a model that is loaded from a directory.
        plot_fig (bool): Plots a single blind testing dataset results (actual versus predicted).
        network_type (str): The type of network being used either SEQ for sequential or AE for autoencoders.
        model_update_path (str): The path of model to be loaded if network_type is equal to False.
        custom_objects (customKerasObjects): CustomObjects to be loaded alongside the loaded model if network_type is equal to False.
        save_encoder (bool): Set to True to save the encoder portion of an autoencoder (only use when network_type is AE).
        use_encoder (bool): Whether to attach an encoder at the beginning of a SEQ network. Only works if arch_type is SEQ_ENCODER.
        encoder_path (str): The path to the encoder to be loaded if use_encoder is set to True.
    Returns:
        min_val_loss: Minimum validation loss encountered
    """
```

```
def init_results_file(path_to_save):
    """
    Function that handles the initialization of csv file for saving model summaries.
    Arguments:
        path_to_save (str): the path to the directory to save the summary csv file.
    """
```

```
def append_results_to_file(path_to_save, csv_data):
    """
    Function that appends summary results to the csv file with model summaries.
    Arguments:
        path_to_save (str): the path to the directory to with the summary csv file.
        csv_data (str): The data to be saved to the file.
    """
```

- **Creating, Running, and Updating Models**

The final group of code files is responsible for using all aforementioned codes in order to create the specific deep learning models shown in this thesis. Below is a table with the file name and the models that were created from said file.

| File                               | Input Type    | Purpose                                            |
|------------------------------------|---------------|----------------------------------------------------|
| <b>run_DenoisingML</b>             | Seismic Trace | Denoising seismic traces.                          |
| <b>run_DIR_Autoencoder</b>         | Seismic Trace | Feature extraction and data encoding.              |
| <b>run_SWT_Autoencoder</b>         | SWT Data      |                                                    |
| <b>run_CWT_Autoencoder</b>         | CWT Data      |                                                    |
| <b>run_InversionML</b>             | Seismic Trace | Inverting for velocity.                            |
| <b>run_InversionML_SWT</b>         | SWT Data      |                                                    |
| <b>run_InversionML_CWT</b>         | CWT Data      |                                                    |
| <b>run_InversionML_DIR_Encoder</b> | Seismic Trace | Inverting for velocity with encoded seismic trace. |
| <b>run_Inversion_Acoustic_ML</b>   | Seismic Trace | Inverting for acoustic impedance.                  |
| <b>run_combined_models</b>         | Seismic Trace | Inverting for velocity using end-to-end model.     |
| <b>run_Joint_InversionML</b>       | Seismic Trace | Jointly inverting for velocity and density.        |



## References

- [1] Akiba, T., Sano, S., Yanase, T., Ohta, T. and Koyama, M. 2019. Optuna: A Next-generation Hyperparameter Optimization Framework *In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* [Online]. KDD '19. New York, NY, USA: Association for Computing Machinery, pp.2623–2631. [Accessed 30 March 2023]. Available from: <https://doi.org/10.1145/3292500.3330701>.
- [2] Alkhalifah, Tariq, Chao Song, Umair bin Waheed, and Qi Hao. “Wavefield Solutions from Machine Learned Functions Constrained by the Helmholtz Equation.” *Artificial Intelligence in Geosciences* 2 (December 1, 2021): 11–19. <https://doi.org/10.1016/j.aiig.2021.08.002>.
- [3] Almarzooq, H. 2022. Enhancing conventional VSP data to broadband and its quantitative improvement on lookahead inversion *In: OnePetro*. [Accessed 11 November 2022]. Available from: <https://onepetro.org/SEGAM/proceedings/IMAGE22/1-IMAGE22/D011S192R001/512669>.
- [4] Bakulin, A., Woodward, M., Nichols, D., Osypov, K. and Zdraveva, O. 2010. Localized anisotropic tomography with well information in VTI media. *GEOPHYSICS*. **75**(5), pp.D37–D45.
- [5] Balch, A.H. and Lee, M.W. 1984. Vertical seismic profiling: Technique, applications, and case histories.
- [6] BJV n.d. Vertical Seismic Profiles. [Accessed 11 November 2022]. Available from: <https://www.bjv-3ddesign.com/seismic-acquisition-geometries/vsp/>.
- [7] Burch, T., Hornby, B., Sugianto, H. and Nolte, B. 2010. Subsalt 3D VSP imaging at Deimos Field in the deepwater Gulf of Mexico. *The Leading Edge*. **29**(6), pp.680–685.

- [8] Campbell, Allan. "Ajcampbell25/VSP\_Basics." Jupyter Notebook, January 21, 2023. GitHub Repository. [Accessed 22 September 2023]. [https://github.com/ajcampbell25/VSP\\_Basics](https://github.com/ajcampbell25/VSP_Basics).
- [9] Chandrasekhar, S., Dotiwala, F., Lim, T.K., Khaitan, M.L. and Kumar, R. 2013. Reducing Target Uncertainties and Guiding Drilling Using Seismic While Drilling Technology, a Novel Approach in Andaman Sea Deepwater *In: OnePetro*. [Accessed 11 November 2022]. Available from: <https://onepetro.org/SPEAPOG/proceedings/13APOG/All-13APOG/SPE-165834-MS/173897>.
- [10] Chen, Ke, and Mauricio D. Sacchi. "Robust Reduced-Rank Seismic Denoising." In *SEG Technical Program Expanded Abstracts 2013*, 4272–77. SEG Technical Program Expanded Abstracts. Society of Exploration Geophysicists, 2013. <https://doi.org/10.1190/segam2013-0822.1>.
- [11] Collobert, R. and Weston, J. 2008. A unified architecture for natural language processing: deep neural networks with multitask learning *In: Proceedings of the 25th international conference on Machine learning* [Online]. ICML '08. New York, NY, USA: Association for Computing Machinery, pp.160–167. [Accessed 12 November 2022]. Available from: <https://doi.org/10.1145/1390156.1390177>.
- [12] Daoud, A.M. and Abd El Dayem, M.A. 2018. Look-Ahead Vertical Seismic Profiling VSP Inversion Approach for Density and Velocity in Bayesian Framework *In: OnePetro*. [Accessed 11 November 2022]. Available from: <https://onepetro.org/SPEATCE/proceedings/18ATCE/2-18ATCE/D021S024R006/213773>.
- [13] Diallo, M.S., Kulesh, M., Holschneider, M., Scherbaum, F. and Adler, F. 2006. Characterization of polarization attributes of seismic waves using continuous wavelet transforms. *GEOPHYSICS*. **71**(3), pp.V67–V77.
- [14] Dillon, P.B. and Thomson, R.C. 1984. Offset Source Vsp Surveys and Their Image Reconstruction\*. *Geophysical Prospecting*. **32**(5), pp.790–811.
- [15] Dix, C.H. 1939. Interpretation of well-shot data. *GEOPHYSICS*. **4**(1), pp.24–32.

- [16] Graves, A. 2012. *Supervised Sequence Labelling with Recurrent Neural Networks* [Online]. Berlin, Heidelberg: Springer. [Accessed 12 November 2022]. Available from: <http://link.springer.com/10.1007/978-3-642-24797-2>.
- [17] Hardage, B. 2020. Vertical Seismic Profiling: history, science, and geopolitics [Part 1]. *Vertical Seismic Profiling – history, science and geopolitics*. [Online]. [Accessed 11 November 2022]. Available from: <https://www.geoinsights.com/vertical-seismic-profiling/>.
- [18] Hardage, B.A. 1994. Chapter 8 Seismic Prediction of Overpressure Conditions Ahead of the Bit in Real Drill Time In: W. H. Fertl†, R. E. Chapman and R. F. Hotz, eds. *Developments in Petroleum Science* [Online]. Studies in Abnormal Pressures. Elsevier, pp.241–250. [Accessed 11 November 2022]. Available from: <https://www.sciencedirect.com/science/article/pii/S0376736109702328>.
- [19] Hardage, B.A. 1992. *Crosswell Seismology & Reverse VSP*. Geophysical Press.
- [20] Hardage, B.A. 1984. *Vertical Seismic Profiling Part A: Principles* (M. N. Toksoz, ed.). London : Amsterdam, Netherlands: Geophysical Press.
- [21] He, Y. and Zhao, J. 2019. Temporal Convolutional Networks for Anomaly Detection in Time Series. *Journal of Physics: Conference Series*. **1213**(4), p.042050.
- [22] Hecht-nielsen, R. 1992. III.3 - Theory of the Backpropagation Neural Network\*\*Based on “nonindent” by Robert Hecht-Nielsen, which appeared in Proceedings of the International Joint Conference on Neural Networks 1, 593–611, June 1989. © 1989 IEEE. In: H. Wechsler, ed. *Neural Networks for Perception* [Online]. Academic Press, pp.65–93. [Accessed 12 November 2022]. Available from: <https://www.sciencedirect.com/science/article/pii/B9780127412528500108>.
- [23] Hewage, P., Behera, A., Trovati, M., Pereira, E., Ghahremani, M., Palmieri, F. and Liu, Y. 2020. Temporal convolutional neural (TCN) network for an effective weather forecasting using time-series data from the local weather station. *Soft Computing*. **24**(21), pp.16453–16482.
- [24] Hinds, R.C., Anderson, N.L., Kuzmiski, R.D. and Hardage, B.A. 1996. *VSP Interpretive Processing: Theory and Practice*.

- [25] Hinton, A., Kwiatkowska, M., Norman, G. and Parker, D. 2006. PRISM: A Tool for Automatic Verification of Probabilistic Systems *In: H. Hermanns and J. Palsberg, eds. Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, pp.441–444.*
- [26] Hinton, G.E., Osindero, S. and Teh, Y.-W. 2006. A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*. **18**(7), pp.1527–1554.
- [27] Hinton, G.E. and Salakhutdinov, R.R. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science*. **313**(5786), pp.504–507.
- [28] Horne, S. 2003. Fracture characterization from walkaround VSPs. *Geophysical Prospecting*. **51**(6), pp.493–499.
- [29] Jayalakshmi, T. and Santhakumaran, A. 2011. Statistical Normalization and Back Propagation for Classification. *International Journal of Computer Theory and Engineering*., pp.89–93.
- [30] Jolly, R.N. 1953. Deep-hole geophone study in garvin county, oklahoma. *GEOPHYSICS*. **18**(3), pp.662–670.
- [31] Kail, R., Burnaev, E. and Zaytsev, A. 2022. Recurrent Convolutional Neural Networks Help to Predict Location of Earthquakes. *IEEE Geoscience and Remote Sensing Letters*. **19**, pp.1–5.
- [32] Kim, T.-Y. and Cho, S.-B. 2019. Predicting residential energy consumption using CNN-LSTM neural networks. *Energy*. **182**, pp.72–81.
- [33] Lea, C., Flynn, M.D., Vidal, R., Reiter, A. and Hager, G.D. 2017. Temporal convolutional networks for action segmentation and detection *In: Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017* [Online]. Institute of Electrical and Electronics Engineers Inc., pp.1003–1012. [Accessed 11 November 2022]. Available from: <https://jhu.pure.elsevier.com/en/publications/temporal-convolutional-networks-for-action-segmentation-and-detec>.
- [34] Lea, C., Vidal, R., Reiter, A. and Hager, G.D. 2016. Temporal Convolutional Networks: A Unified Approach to Action Segmentation *In: G. Hua and H. Jégou, eds. Computer Vision – ECCV 2016 Workshops. Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.47–54.*

- [35] Levin, F.K. and Lynn, R.D. 1958. Deep-hole geophone studies. *GEOPHYSICS*. **23**(4), pp.639–664.
- [36] Mateeva, A., Lopez, J., Chalenski, D., Tatanova, M., Zwartjes, P., Yang, Z., Bakku, S., Vos, K. de and Potters, H. 2017. 4D DAS VSP as a tool for frequent seismic monitoring in deep water. *The Leading Edge*. **36**(12), pp.995–1000.
- [37] Mavko, G., Mukerji, T. and Dvorkin, J. 2020. *The Rock Physics Handbook: Tools for Seismic Analysis of Porous Media* [Online] 3rd ed. Cambridge: Cambridge University Press. [Accessed 11 November 2022]. Available from: <https://www.cambridge.org/sa/universitypress/subjects/earth-and-environmental-science/solid-earth-geophysics/rock-physics-handbook-3rd-edition>
- [38] Maxwell, S.C. and Urbancic, T.I. 2001. The role of passive microseismic monitoring in the instrumented oil field. *The Leading Edge*. **20**(6), pp.636–639.
- [39] McCollum, B. and Larue, W.W. 1931. Utilization of Existing Wells in Seismograph Work1. *AAPG Bulletin*. **15**(12), pp.1409–1417.
- [40] McCulloch, W.S. and Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*. **5**(4), pp.115–133.
- [41] Oboué, Yapo A. S. I., Wei Chen, Omar M. Saad, and Yangkang Chen. “Adaptive Damped Rank-Reduction Method for Random Noise Attenuation of Three-Dimensional Seismic Data.” *Surveys in Geophysics* 44, no. 3 (June 1, 2023): 847–75. <https://doi.org/10.1007/s10712-022-09756-7>.
- [42] Oristaglio, M.L. 1985. A guide to current uses of vertical seismic profiles. *GEOPHYSICS*. **50**(12), pp.2473–2479.
- [43] Payne, M.A. 1994. Looking ahead with vertical seismic profiles. *GEOPHYSICS*. **59**(8), pp.1182–1191.
- [44] Pelt, D.M. and Sethian, J.A. 2018. A mixed-scale dense convolutional neural network for image analysis. *Proceedings of the National Academy of Sciences*. **115**(2), pp.254–259.
- [45] Pevzner, R., Isaenkov, R., Yavuz, S., Yurikov, A., Tertyshnikov, K., Shashkin, P., Gurevich, B., Correa, J., Glubokovskikh, S., Wood, T., Freifeld, B. and Barraclough, P. 2021. Seismic monitoring of a small CO<sub>2</sub> injection using a multi-

- well DAS array: Operations and initial results of Stage 3 of the CO2CRC Otway project. *International Journal of Greenhouse Gas Control*. **110**, p.103437.
- [46] Riggs, E.D. 1955. Seismic wave types in a borehole. *GEOPHYSICS*. **20**(1), pp.53–67.
- [47] Rioul, O. and Duhamel, P. 1992. Fast algorithms for discrete and continuous wavelet transforms. *IEEE Transactions on Information Theory*. **38**(2), pp.569–586.
- [48] Rivas, D.P. and Montoya, L. 2020. *Deep Learning for Beginners: A beginner's guide to getting up and running with deep learning from scratch using Python*. Packt Publishing.
- [49] Saad, O.M. and Chen, Y. 2020. Deep denoising autoencoder for seismic random noise attenuation. *GEOPHYSICS*. **85**(4), pp.V367–V376.
- [50] Sainath, T.N., Vinyals, O., Senior, A. and Sak, H. 2015. Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*., pp.4580–4584.
- [51] Samuel, A.L. 1967. Some Studies in Machine Learning Using the Game of Checkers. II—Recent Progress. *IBM Journal of Research and Development*. **11**(6), pp.601–617.
- [52] Sheriff, R.E. and Geldart, L.P. 1995. *Exploration Seismology* [Online] 2nd ed. Cambridge: Cambridge University Press. [Accessed 11 November 2022]. Available from: <https://www.cambridge.org/core/books/exploration-seismology/CC00727A219943E62F6FF01426DBA9D2>.
- [53] Sibi, P., Jones, S.A. and Siddarth, P. 2013. Analysis of different activation functions using back propagation neural networks. *Journal of theoretical and applied information technology*. **47**(3), pp.1264–1268.
- [54] Sinha, S., Routh, P.S., Anno, P.D. and Castagna, J.P. 2005. Spectral decomposition of seismic data with continuous-wavelet transform. *GEOPHYSICS*. **70**(6), pp.P19–P25.
- [55] Song, Chao, Tariq Alkhalifah, and Umair Bin Waheed. “Solving the Frequency-Domain Acoustic VTI Wave Equation Using Physics-Informed Neural Networks.” *Geophysical Journal International* 225, no. 2 (April 1, 2021): 846–59. <https://doi.org/10.1093/gji/ggab010>.

- [56] Stewart, R.R. and Disiena, J.P. 1989. The values of VSP in interpretation. *The Leading Edge*. **8**(12), pp.16–23.
- [57] Suzuki, K. and Suzuki, K. 2011. *Artificial Neural Networks - Methodological Advances and Biomedical Applications* [Online]. [Accessed 11 November 2022]. Available from: <https://www.intechopen.com/books/117>.
- [58] Toksoz, N. and Stewart, R. 1984. *Vertical Seismic Profiling, Part B: Advanced Concepts*. London : Amsterdam, Netherlands: London: Geophysical Press 1984.
- [59] Tsuji, T., Hino, R., Sanada, Y., Yamamoto, K., Park, J.-O., No, T., Araki, E., Bangs, N., von Huene, R., Moore, G. and Kinoshita, M. 2011. In situ stress state from walkaround VSP anisotropy in the Kumano basin southeast of the Kii Peninsula, Japan. *Geochemistry, Geophysics, Geosystems*. **12**(9).
- [60] TURING, A.M. 1950. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*. **LIX**(236), pp.433–460.
- [61] Uncuoglu, E., Laman, M., Saglamer, A. and Kara, H.B. 2008. Prediction of Lateral Effective Stresses in Sand Using Artificial Neural Network. *Soils and Foundations*. **48**(2), pp.141–153.
- [62] Waheed, Umair bin, Ehsan Haghighat, Tariq Alkhalifah, Chao Song, and Qi Hao. “PINNeik: Eikonal Solution Using Physics-Informed Neural Networks.” *Computers & Geosciences* 155 (October 1, 2021): 104833. <https://doi.org/10.1016/j.cageo.2021.104833>.
- [63] Wang, B., Zhang, N., Lu, W. and Wang, J. 2019. Deep-learning-based seismic data interpolation: A preliminary result. *GEOPHYSICS*. **84**(1), pp.V11–V20.
- [64] Wang, J., Yu, L.-C., Lai, K.R. and Zhang, X. 2016. Dimensional Sentiment Analysis Using a Regional CNN-LSTM Model *In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* [Online]. Berlin, Germany: Association for Computational Linguistics, pp.225–230. [Accessed 12 November 2022]. Available from: <https://aclanthology.org/P16-2037>.
- [65] Wang, Y., Wang, B., Tu, N. and Geng, J. 2020. Seismic trace interpolation for irregularly spatial sampled data using convolutional autoencoder. *GEOPHYSICS*. **85**(2), pp.V119–V130.

- [66] Weinzierl, W. and Wiese, B. 2021. Deep learning a poroelastic rock-physics model for pressure and saturation discrimination. *Geophysics*. **86**(1), pp.MR53–MR66.
- [67] Yang, H., Cai, J., Guo, S., Gao, Y., Yanhua, Y.L. and Lim, T.K. 2013. Predicting Drilling Target Ahead of Bit Using VSP Lookahead Technique that Help Mitigate Drilling Risk in Over Pressure Zone, a Case Example from South China Sea *In: OnePetro*. [Accessed 11 November 2022]. Available from: <https://onepetro.org/SPEAPOG/proceedings/13APOG/All-13APOG/SPE-165838-MS/173941>.
- [68] Yilmaz, Ö. 2001. *Seismic Data Analysis* [Online]. Society of Exploration Geophysicists. [Accessed 11 November 2022]. Available from: <https://library.seg.org/doi/10.1190/1.9781560801580>.
- [69] Yuhong, X., Jun, C., Lingxia, Z., Hong, T., Yanhua, L., Voskamp, A. and Bailin, W. 2010. Ahead of Bit Pore Pressure Prediction using VSP - A Case Study in South China Sea *In: OnePetro*. [Accessed 11 November 2022]. Available from: <https://onepetro.org/SPEIOGCEC/proceedings/10IOGCEC/All-10IOGCEC/SPE-130551-MS/106859>.
- [70] Zhao, Wentian, Yanyun Gao, Tingxiang Ji, Xili Wan, Feng Ye, and Guangwei Bai. “Deep Temporal Convolutional Networks for Short-Term Traffic Flow Forecasting.” *IEEE Access* 7 (2019): 114496–507. <https://doi.org/10.1109/ACCESS.2019.2935504>.
- [71] Zhang, D. 2019. Wavelet Transform *In: D. Zhang, ed. Fundamentals of Image Data Mining: Analysis, Features, Classification and Retrieval* [Online]. Texts in Computer Science. Cham: Springer International Publishing, pp.35–44. [Accessed 11 November 2022]. Available from: [https://doi.org/10.1007/978-3-030-17989-2\\_3](https://doi.org/10.1007/978-3-030-17989-2_3).
- [72] Zhang, G., Wang, Z. and Chen, Y. 2018. Deep learning for seismic lithology prediction. *Geophysical Journal International*. **215**(2), pp.1368–1387.

## Vitae

Name :Hussain Abbas Salman Almarzooq

Nationality :Saudi

Date of Birth :3/18/1992

Email :Hussain747715@gmail.com

Address :P.O box 9094, Al-Mujama, Saudi Aramco, Dhahran,  
31311, Saudi Arabia

Academic Background :This thesis is part of the requirements to fulfil a masters of science in geophysics degree at KFUPM. My journey with KFUPM started in January of 2021 and will soon conclude before the end of 2023. Before that, in 2015, I complete a bachelor degree in geophysics from University of Leeds. Between my pursuit of the masters and bachelor degrees, I joined Saudi Aramco where I learned how to be a well-rounded geoscientist. Then, I decided to pursue a career as a borehole seismic geophysicist which involved dealing with all manners of vertical seismic profiling data (VSPs). I distinctly choose this specialization because it allowed for seeing the seismic data in a 3D sense which is very rare elsewhere and it allowed for examining different waveforms and types in a more intimate manner. Below is a list of publications, conference abstracts, and workshop abstracts that I was involved in:

- Yousef, B. M., et al. "Fracture parameter inversion from passive seismic shear-wave splitting: A validation study using full-waveform numerical synthetics." *Tectonophysics* 712 (2017): 736-746.
- Almarzooq, Hussain. "Enhancing conventional VSP data to broadband and its quantitative improvement on lookahead inversion." *SEG International Exposition and Annual Meeting*. SEG, 2022.
- Almubarak, M. S., et al. " Utilizing Harmonics to Achieve Wider Bandwidth for Zero-Offset VSP Data." *SEG Advances in Quantitative Interpretation Workshop*. SEG, 2023.
- Almubarak, M. S., et al. " Seismic Source-Generated Harmonic Distortions: Use Them or Lose Them?." *EAGE Annual Conference & Exhibition 2023*. EAGE, 2023.
- Almarzooq, Hussain, and Waheed, Umair bin. " Inverting for Acoustic Impedance from Noisy Synthetic ZVSP Corridor Stack using Machine Learning." *Seventh EAGE Borehole Geophysics Workshop*. EAGE, 2023.
- **Professional & Academia Accounts:**
  - **Linkedin:** <https://www.linkedin.com/in/hussain-almarzooq-9359684b/>
  - **Google Scholar:** [https://scholar.google.com/citations?user=\\_NQe8gEAAAAJ&hl=en](https://scholar.google.com/citations?user=_NQe8gEAAAAJ&hl=en)
  - **ResearchGate:** <https://www.researchgate.net/profile/Hussain-Almarzooq>