WORK PLACEMENT REPORT

Automatic arrival time picking for seismic inversion with unlabeled data

Ngo Nghi Truyen Huynh

August 2021

Supervisors:

Roland Martin

Thomas Oberlin

Bastien Plazolles

Institut National des Sciences Appliquées de Toulouse

Département Génie Mathématique et Modélisation

Acknowledgment

I would like to express my gratitude and appreciation for the support of my supervisors, Roland Martin, Thomas Oberlin and Bastien Plazolles. Without your help, this project would not have been possible. Many thanks to Thomas for sharing your advice and your extensive knowledge of machine learning. Special thanks to Roland for your patience, guidance and for all the time we spent together throughout the project. I greatly appreciate your time for sharing your deep understanding of geophysics, and for reading my numerous revisions as well as helping make some sense of the confusion. Also thanks to Etienne Gondet, who helped me have the opportunity to attend the AI conferences, where I gave a presentation and met many peoples working in this field.

Lastly, I would like to thank the French National Centre for Scientific Research (CNRS) for providing me with the financial means to complete this project.

Abstract

Nowadays, the understanding of subsurface information on the Earth is crucial in numerous fields such as economics of oil and gas, archaeology and hydrology, particularly in a context of climate change. The methodology is to reconstruct the velocity model, that contains information about the basement structure, by resolving complex nonlinear systems with the data collected from seismic experiments and measurements. In the last few years, many deep neural networks have been proposed to simplify the seismic inversion problem based on adjoint state and arrival time picking methods. However, the labeling process of seismic data requires too much time and effort to train a neural network with supervised learning. We present an approach with unlabeled data by using transfer learning and semi-supervised learning based on a deep neural network for arrival time picking. Huber regression and Support Vector Machine are used to enhance the pseudo-labeling, that allows to train a neural network model with high-quality labeled data. The model showcased a very high score when evaluating and testing on the synthetic as well as real data. This can demonstrate the efficiency of automatic arrival time picking method for resolving seismic inversion problem.

Keywords: seismic survey, seismic inversion, seismic wave, velocity model, adjoint state, arrival time picking, deep neural network, unlabeled data, pseudo-labeling, transfer learning, semi-supervised learning, Huber regression, Support Vector Machine

Contents

Part I
Introduction

My internship at Toulouse Environmental Geosciences (GET) takes place from March 8 to September 7, 2021. This is part of the ENV'IA project from 2021 to 2023. The objective of ENV'IA is to strengthen the links between the communities of Environmental Sciences, Space, Universe (SEEU) and computing, in particular in AI and data science. The internship is a mix between geophysics, under the supervision of GET, and data science, under the supervision of ISAE-SUPAERO (National Higher French Institute of Aeronautics and Space) and ANITI (Artificial and Natural Intelligence Toulouse Institute).

GET is a fundamental and applied research laboratory with more than 165 permanent staffs (around 250 peoples with students, PhD students and postdocs, etc.). The laboratory is a member of the Midi-Pyrénées Observatory (OMP) and affiliated to 5 different research organisms (CNRS, IRD, Toulouse III University, CNAP and CNES). Having an analytical and instrumental park along with an experimentation platform, GET is recognized for its excellence in many fields (geochemistry, metallogeny, petrology, mineralogy, gravimetry, geodesy, geophysics, etc.) as well as for its skill in the fields of observation and spatial remote sensing as in numerical analysis (modeling). In fact, the laboratory has both national and international scope, allowing it to be part of the largest scientific think tanks.

The internship was proposed for the monitoring of basement structures and water-saturated ground purpose in the context of climate change, where discovering, producing and managing new resources such as underground water is a major challenge. In this perspective, it is important to develop and optimize digital 3D underground imaging tools in high resolution to detect and estimate groundwater content at regional scales. This involves, in particular, the inversion of massive seismic data. To limit computational times, it is crucial to identify and select the relevant portions of the recorded seismic signals, so as to reduce the size and complexity of the data when it is inverted. The main goal of this internship is to make use of recent AI techniques in order to simplify and help solve the seismic inversion problem.

My internship is under the supervision of Dr. Roland Martin (GET-OMP), Dr. Thomas Oberlin (ISAE-SUPAERO) and Dr. Bastien Plazolles (GET-OMP). I was impressed by the context of the internship and convinced by my supervisors at the first time we met for the interview. That is why I decided to choose this for my end-of-studies internship.

The report is organized as follows. Primarily, I introduce some state-of-the-art techniques for seismic imaging in Part II. Secondly, Part III describes in details a deep neural network developed in 2018, dedicated to arrival time picking for seismic phases. Subsequently, in Part IV and V, I propose an approach to train the neural network model with our synthetic as well as real data via transfer learning and semi-supervised learning. The robust linear regression methods and Support Vector Machine are employed in the pseudo-labeling to improve the quality of pseudo labels. Ultimately, I give a conclusion to this work as well as this internship in Part VI. A GitHub code related to my work is also available at:

`https://github.com/nghitruyen/PhaseNet_keras_version`

Part II
State-of-the-art techniques of seismic imaging

II.1 Introduction to body waves

Waves can be defined as a disturbance in materials that carries energy through a solid or acoustic medium where energy waves are generated by an earthquake or an artificial explosion. In general, an elastic material through which the wave propagates does not move with the wave. The movement of the material is considered as small motion as the wave passes. In other words, after the wave has passed, the material usually is not changed at all and looks just like it was before the wave. When an earthquake takes place or when an explosion or mechanical device is used to initiate a seismic disturbance artificially, a complex field of seismic waves is generated. Waves that travel through the interior of the Earth are called body waves [35]. They follow ray paths bent by varying density and modulus (stiffness) of the Earth's interior that are affected by composition, phase and temperature. There are two main kinds of body waves that are detected by a seismogram:

- compression waves (P-waves) that are polarized and moving in the direction the wave is traveling,
- shear waves (S-waves) that are slower than the P-waves and are moving in the direction the wave is traveling but with a polarization orthogonal to this direction.

P-waves travel faster and they are the first waves to arrive from the earthquake, closely followed by their reflection from the surface and S-waves arrive next (Figure 1). Moreover, the P-wave can travel through liquids and solids while the S-wave can only travel through solids (Figure 2).

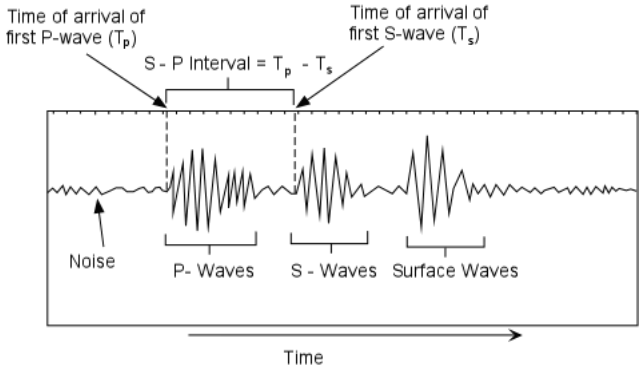

Figure 1: Body waves on a seismogram
Source: `http://earthsci.org/education/teacher/basicgeol/earthq/earthq.html`

II.2 Seismic inversion problem

Seismic exploration [37] is the use of seismic imaging to investigate the surface of the Earth. It plays a crucial role in the delineation of near surface geology for economic deposits of oil, gas,

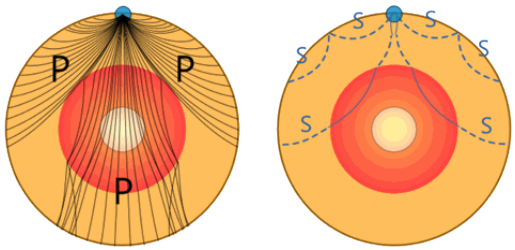

Figure 2: Motion of P-wave and S-wave
Source: `https://www.mathsisfun.com/physics/waves-seismic.html`

or minerals, but also for engineering purposes, archaeological, and scientific studies. Among the parameters used for seismic exploration purpose, seismic velocity is one of the most important ones. This can be defined as the speed with which an elastic wave propagates through a medium, and thus considered to be seismic properties. Seismic inversion is the fact of reconstructing the subsurface velocity model by employing the data collected from seismic experiments (Figure 3).

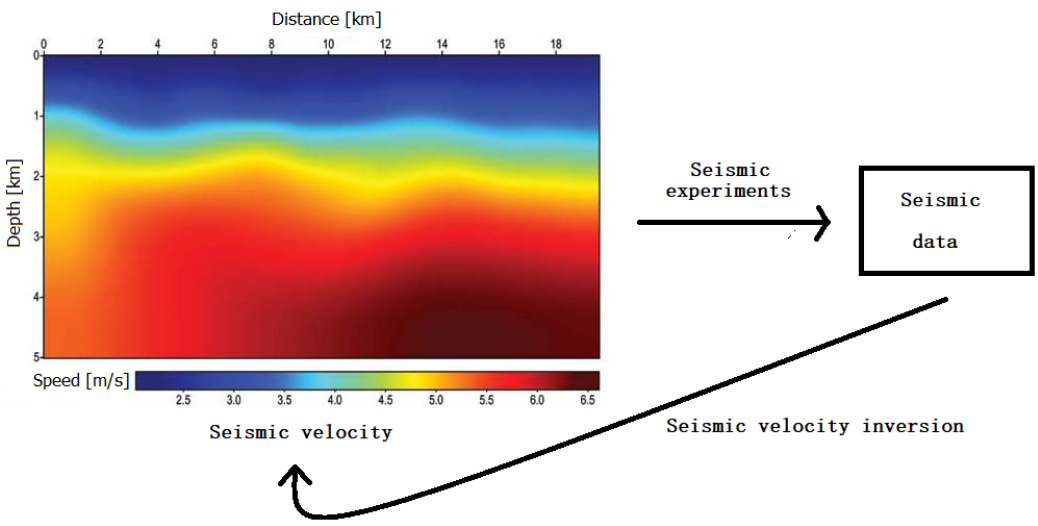

Figure 3: Illustration of the seismic inversion problem

Basically, such velocity information can be derived by travel-time tomography [15, 14, 5] or full-waveform inversion (FWI) [38, 40], that have been proven to be effective for acoustic model building in many geological scenarios. However, these techniques still have limitations affected by human interventions and are sophisticated in resolving highly nonlinear processes [21, 19]. Currently, most seismic inversion problems are approached by:

- the physically-driven seismic inversion based on the adjoint theory that is commonly used in the geophysical community. This classical method attempts to iteratively minimize a cost function based on the difference between the observed and calculated data by computing its gradient. This gradient is based on some model parameters such as the density ρ, the speed of seismic phases V_P, V_S, and calculated by the adjoint state method, that permits the

computation of the first derivative of a physical observable or an associated objective function with respect to model parameters [11].

- the data-driven seismic inversion based on deep learning [23] techniques, that uses supervised learning method to reconstruct the velocity model directly from recorded seismic data via deep convolutional neural networks (CNN) [1, 41, 42, 24].

Furthermore, using tools from deep learning such as automatic differentiation to improve the adjoint state method is now one of the most appealing approaches to solve seismic inversion problem with fast algorithm and high accuracy [6, 32, 44, 3].

Collecting data for a such fully end-to-end deep learning approach however costs a lot of time and money. On the contrary, resolving a part of the inversion problem by deep learning is less expensive and requires less data. Recently, geophysicists and data scientists have made many improvements in helping reconstruct the seismic velocity model by deep learning in frequency-domain, for extracting dispersion curves [8], and in time-domain, for arrival time picking [39, 43].

II.3 Automatic arrival time picking by deep learning

Arrival time picking is one of the less expensive solution for approaching the inversion problem by deep learning (Figure 4). This is based on the fact of detecting the first arrival time of body waves or surface wave and thus helps reconstruct the seismic velocity model thanks to the speed of P and S phases (Figure 5). Namely, these arrival times allow us to access to the time windows and thus compute the gradient of the cost function by using adjoint-based inversion techniques (tomography or FWI) or travel-time inversion techniques based on ray tracing method [22, 30, 31, 12, 17, 4].

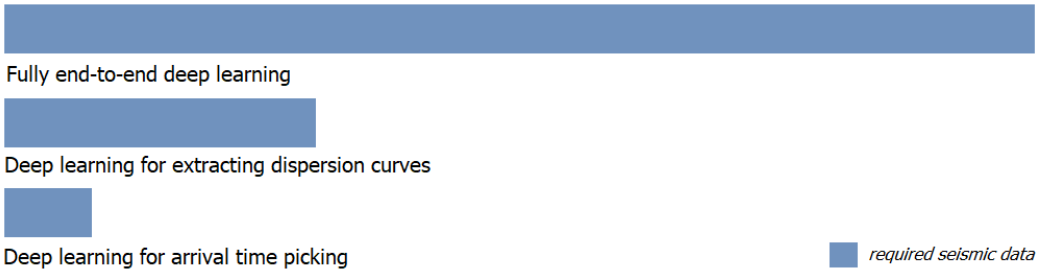

Figure 4: Illustration of required seismic data for different deep learning approaches

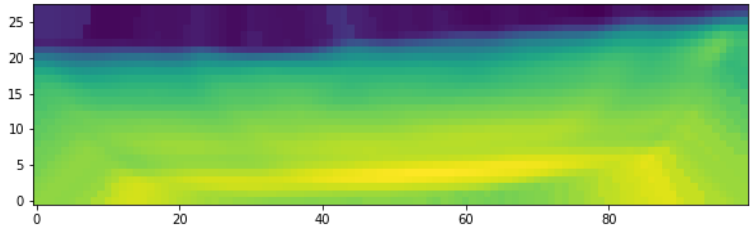

Figure 5: Seismic velocity model of P-phase

Before employing deep learning techniques, we used to utilize hand-picking methods to help solve seismic inversion problem. Nevertheless, the hand-picking methods require too much effort and can be affected by inconsistencies. Some classical automatic time picking methods, for example AR-picker [2], have helped dramatically reduce time and costs but are still less accurate. Therefore, we are attempting to develop automatic-picking methods based on deep learning approaches to calculate these arrival times. PickNet [39] and PhaseNet [43] are the state-of-the-art deep neural networks for arrival time picking method. PickNet is constructed from VGG-16 [36], a very deep CNN for large-scale image recognition. PhaseNet on the other hand is based on U-Net [33], that was introduced for biomedical image segmentation and particularly, has shown its efficiencies for segmentation of neuronal structures in electron microscopic stacks. The experiments in [27] showed that PhaseNet has a better score and standard deviation of error than PickNet (even though both models were tested on two different data sets). In this paper, we will thus focus on an automatic arrival time picking method based on PhaseNet.

Part III

Automatic arrival time picking based on PhaseNet

PhaseNet is a deep-neural-network-based arrival-time picking method that uses three-component seismic waveform as input and returns probability distributions of arrival times of both P-wave and S-wave along with the noise as output. We note that picking S arrivals is more challenging for automatic method because the S-waves are not the first arriving waves and thus, they emerge from the scattered waves of the P coda. PhaseNet model has shown a significant improvement compared to classical time picking methods, particularly, a very higher precision of predicted time picking with S-waves.

III.1 Simple CNN and chain rule

In the first place, let us consider a simple neural network with training data set (X, Y), kernel f, weights W, biases b, associated activation function ϕ and loss function L as Figure 6. Then, if W is a parameter that we want to learn for during the training, then the updating step is:

$$W_{new} = W_{old} - \lambda \frac{\partial L}{\partial W}$$

where λ is the learning rate and L is the loss function.

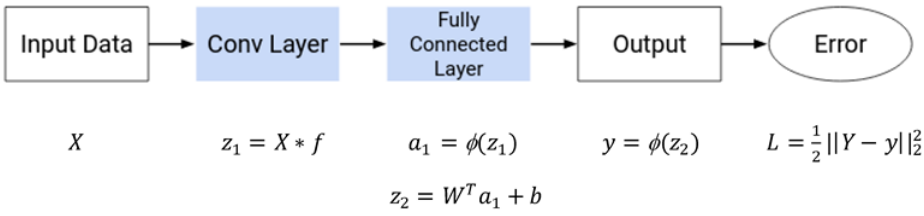

Figure 6: Simple architecture of CNN

The purpose of any CNN is to optimize learnable parameters (weights and biases) with respect to the loss function. This loss value can be minimized through a so-called "backpropagation algorithm" that provides us a way to calculate the gradient of the loss function by computing the partial derivatives. Mathematically, we need to go backward to compute the partial derivative $\frac{\partial L}{\partial W}$:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z_2} \cdot \frac{\partial z_2}{\partial W}.$$

In some cases, if we have a large number of layers when we go backward through the network, then the gradient of the network becomes smaller and tends to 0. This causes an uninterrupted gradient flow from the first layer to the last layer in the model, that is called the problem of vanishing gradient. In this scenario, the loss function stops decreasing but it is still far from the desired result at a certain step of updating parameters during the training. We will explain in III.2.3 how we can circumvent this problem for a complex CNN such as PhaseNet by using skip connections.

III.2 Architectural model

III.2.1 A deep neural network based on U-Net architecture

The PhaseNet model is built on an U-Net, a deep CNN, that was developed for biomedical image segmentation in 2015. It consists of two symmetrical branches containing 4 downsampling stages and 4 upsampling stages (Figure 7). Each stage is a convolution followed by ReLU activation function. The downsampling stages compress the input signal and reduce the size of the data while the upsampling stages expand and convert the useful information into probability distributions of the outputs for each discrete time series. The skip connection used in each upsampling stage allows to concatenate directly the left output to the right layer without going through the deep layers between them.

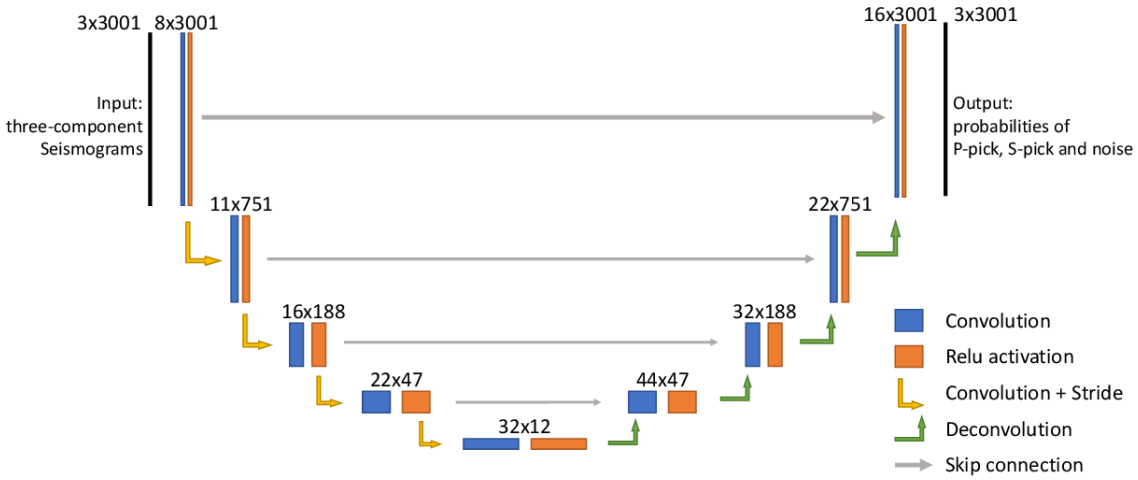

Figure 7: PhaseNet architecture
Taken from [43]

The inputs are three-component seismograms while the outputs are the probability distributions of P-wave, S-wave and noise. Let x be a time series of the input, $j = 1, 2, 3$ represent noise, P-wave and S-wave respectively, $z_j(x)$ be the unscaled values of the last layer such that:

$$\sum_{j=1}^{3} z_j(t) = 1, \forall t \in x.$$

So we utilize the soft-max normalized exponential function to model the probability of each category (noise, P-wave, S-wave) at each discrete time series:

$$q_j(x) = \frac{e^{z_j(x)}}{\sum_{i=1}^{3} e^{z_i(x)}}.$$

Finally, we define the loss function by using cross entropy between the true probability distribution $p(x)$ and predicted distribution $q(x)$:

$$L(p, q) = -\sum_{i=1}^{3} \sum_{x} p_i(x) \log q_i(x).$$

Basically, an U-net is stood on a CNN by adding upsampling stages (III.2.2) and skip connections (III.2.3). The following subsections detail these two essential techniques, that are used to construct a fully convolutional network (FCN) from a CNN.

III.2.2 Transpose convolutional layers

Upsampling is now appearing in many CNN architectures. This technique exists in symmetrical architectures (encoder-decoder or convolution-deconvolution) and can be used to reconstruct the original representation of a convolution filter map by using transposed convolution (deconvolution). The deep deconvolution network proposed in [28] has been used to develop a novel semantic segmentation algorithm. Firstly, the kernels in this architecture can be learnt over time thanks to regular convolutional layers (compressing input data) and then can be used to define the transposed convolution in order to find the original data (decompression). Mathematically, let X be an input image. Then, the convolution is:

$$SH * X$$

and the deconvolution is:

$$H^T S^T * X$$

where S is a sub-sampling operator and H is Toeplitz matrix related to the kernel.

.

III.2.3 Skip connections

Currently, skip connection is a standard module in many CNN architectures. It plays a very important role in the fact of building a FCN [25]. Based on a CNN, we can build a typical FCN by:

- adding an expanding path (decoder) called "long skip connections" (Figure 8a) that recovers spatial information (updated parameters: weights, biases) by merging features skipped from the various resolution levels on the contracting path (encoder),
- adding a "short skip connection" (Figure 8b) that allows us to increase the convergence speed of the loss function and build very deep neural networks that can contain hundreds of layers.

Skip connections allow to circumvent the vanishing gradient problem that is mentioned in III.1. They provide an alternative path for the gradient (with backpropagation) by skipping some layer in the deep architecture and feeding the output of one layer as the input to the next layers (instead of only the next one). Besides, another advantage of skip connections is that it can capture some information in the initial layers. This allows the later layers to also learn from them. In general, there are two main kinds of skip connections we usually use in deep learning:

- ResNet [13] (additive skip connection) backpropagates through the identity function by using a vector addition (Figure 9a),
- DenseNet [16] (concatenative skip connection) concatenates previous feature maps (Figure 9b) to avoid the fact of having low-level information shared between the input and output.

PhaseNet is built on a symmetrical architecture by adding long skip connections (with concatenative connection for example). These skip connections allow to recover fine-grained details and the full spatial resolution at the expanding path in order to improve significantly prediction results. Effectively, the experiments in [9] showed that adding long skip connections in the encoder-decoder architecture can achieve an incredibly effective work in medical image segmentation.

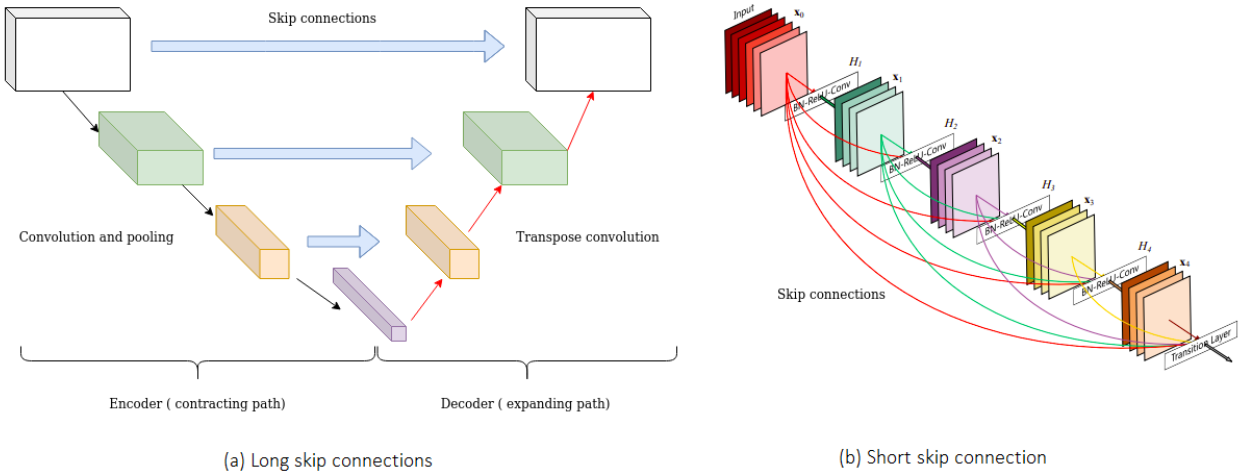

Figure 8: Two kinds of setup of skip connection
Source: `https://theaisummer.com/skip-connections/`

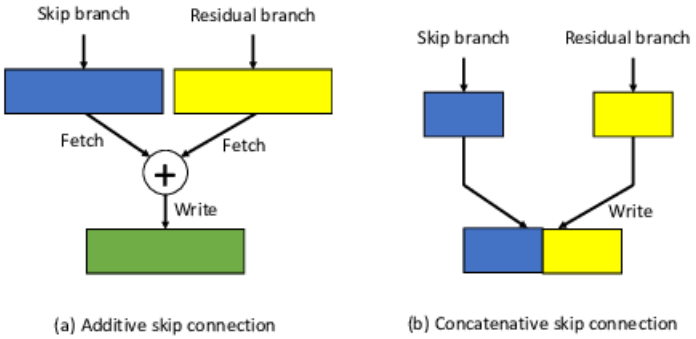

Figure 9: Additive connection and concatenative skip connection
Source: `https://www.researchgate.net/figure/Additive-skip-connections-vs-concatenative-skip-connections-Rectangles-represent-data_fig1_329115979`

III.3 Label generation

Since the data labels are the arrival times of the two phases while PhaseNet returns the probability distributions of arrival times for each class, we will find out in this section how to generate probability distributions (segmentation maps) from the labels (III.3.1) and transform our detection problem into a segmentation problem (III.3.2).

III.3.1 Generating segmentation map from arrival time

The goal of generating a segmentation map instead of taking solely the arrival time is to change the characteristics of the label into more meaningful ones, thus facilitating interpretation and classification. Figure 10 shows an example for generating segmentation map from the arrival times of P and S-wave.

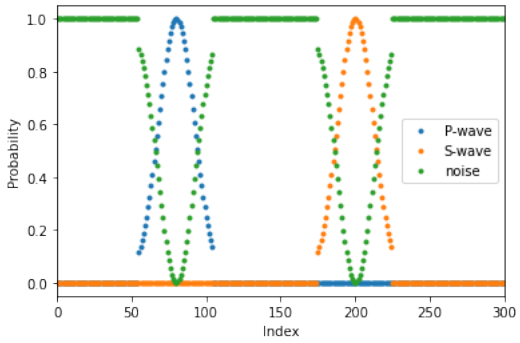

Figure 10: Example of generating segmentation map

Each symmetric "bell curve" is generated by using the Gaussian function. Assuming that t_P is an arrival time of phase P, t_S is an arrival time of phase S, n is the length of the time series. Then, the segmentation maps of noise and both phases are generated by the following formula:

$$\forall t \in [1, n], segment_P(t) = \begin{cases} e^{-\frac{1}{2}\frac{(t-t_P)^2}{d^2}}, \text{ if } t \in [t_P - 2d, t_P + 2d] \\ 0, \text{ otherwise} \end{cases},$$

$$segment_S(t) = \begin{cases} e^{-\frac{1}{2}\frac{(t-t_S)^2}{d^2}}, \text{ if } t \in [t_S - 2d, t_S + 2d] \\ 0, \text{ otherwise} \end{cases},$$

$$segment_{noises}(t) = 1 - segment_P(t) - segment_S(t)$$

where d is the standard deviation, or the Gaussian RMS width, that controls the width of the "bell". Ideally, we can choose $d = \frac{n}{100}$, so the bell is neither too narrow nor too wide. Figure 11 showcases an example with a suitable value of the standard deviation.

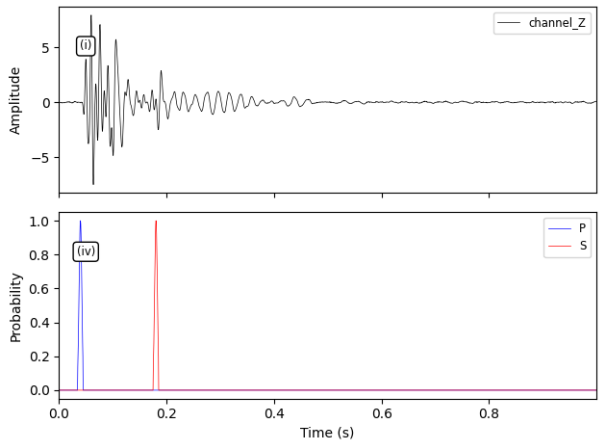

Figure 11: The upper figure is the seismogram. The lower figure is the probability distribution of P-wave and S-wave generated from the arrival times using the Gaussian function with $d = 20$ and $n = 2000$.

III.3.2 Detecting arrival time from output segmentation map

Now, we want to pick up the arrival times from the output segmentation maps returned by the network. The `detect_peaks()` function allows to detect the peaks in the data (the output of the network) based on their amplitude and other features. By default, the function will detect all peaks in the data (Figure 12).

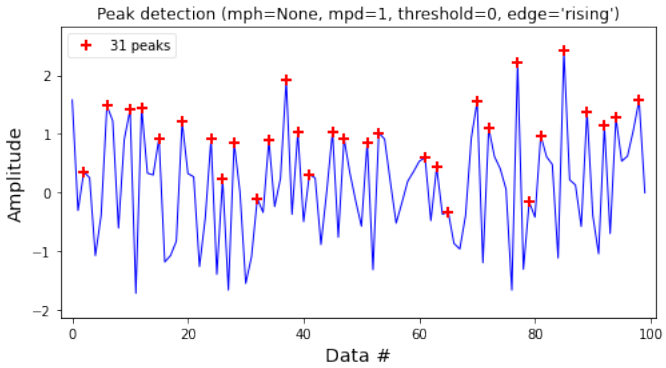

Figure 12: Detecting all peaks in the data

Besides, there are some parameters we need to tune based on different criteria such as:

- detect peaks that are greater than minimum peak height (mph). For instance, if we set a value of 6 for the parameter based on this criterion for a data as Figure 13, so the function only detects the peak which are not less than 6 (peak at index 8) but do not detect the three peaks at index 1, 3 and 11 which are less than 6.
- detect peaks that are at least separated by minimum peak distance in number of data (mpd). For instance, if we consider Figure 14, we can see that the biggest peak is 7 (at index 8) and

this peak is picked up at first. If we set the value of the peak distance is 3, so the peak at index 11 is not picked up as the distance from it to the peak at index 8 is 3 (there are 2 points between them) that is not greater than 3. On the contrary, the peak at index 3 is picked up and the peak at index 1 is not by the same way.

- detect peaks that are greater than a threshold in relation to their immediate neighbors. In Figure 15, we chose the value of threshold is 2, so the peak at index 3 is not picked up because the gap between its amplitude and the amplitude of its neighbor (here is the neighbor at index 4) is 1 ($= 2 - 1$) that is less than 2.

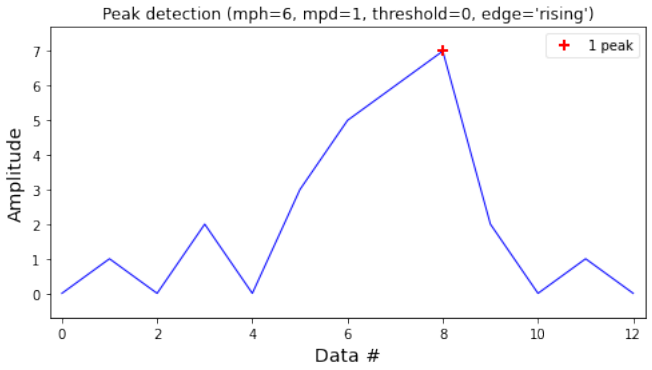

Figure 13: Detecting peaks in the data with mph criterion

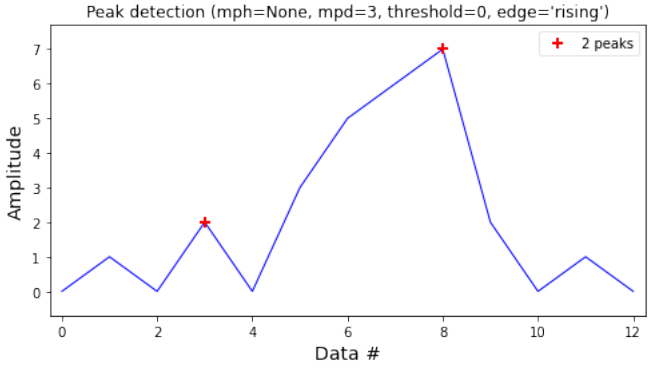

Figure 14: Detecting all peaks in the data with mpd criterion

III.4 Testing and scoring metrics

For evaluating the performance of the model with a test data set, we use the following evaluation metrics: precision, recall and F1-score. We assume that predicted peaks above 0.5 are positive picks.

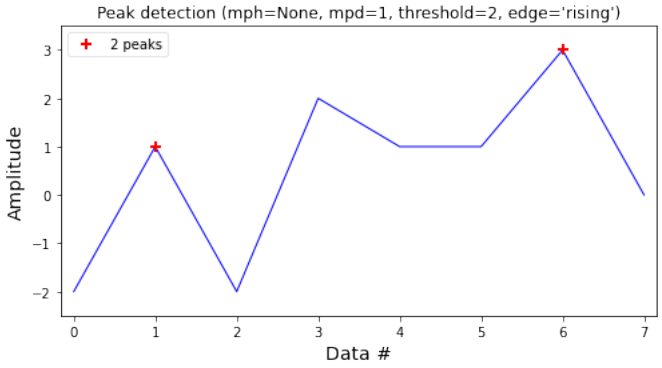

Figure 15: Detecting all peaks in the data with threshold criterion

Predicted peaks that have large arrival-time residuals comparing to the true label are counted as false positives (false predictive values). Those that have small residuals are counted as true positives (true predictive values). Those that are relevant elements but are not among the predicted values are called false negatives. Precision is so the fraction between the number of true positives and the total number of predicted values:

$$P = \frac{T_p}{T_p+F_p}.$$

While precision describes the precision of predictive values, recall refers to the percentage of total relevant results correctly classified:

$$R = \frac{T_p}{T_p+F_n}.$$

For instance, if we work for a test data that contains 1540 samples. Among these 1540 samples, we find that the P-waves for example appear 1420 times in reality (so the number of relevant elements is 1420). Then, the model predicts that the P-waves appear 1440 times (i.e. there are 1440 peak probabilities of P-waves that overtake 0.5), so the positive picks (the total number of predictive values) is 1440. Now, we find that among these 1440 positive picks, there are 1340 predicted values whose residual is less than 0.1s, so the number of true positives is 1340 while that of false positives is $1440 - 1340 = 100$. Thus, the number of false negatives is $1420 - 1340 = 80$. Hence, precision and recall are respectively:

$$P = \frac{1340}{1440}, R = \frac{1340}{1420}.$$

In some cases, we can get a very high precision but a very low recall. For example, if we set a very high threshold for positive picks, only the best picks are reported positive which is only a small portion of total true picks. So F1 score gives us a balanced criterion between precision and recall:

$$F1 = 2\frac{PR}{P+R}.$$

The pre-trained model in PhaseNet achieved a F1-score of 0.896 for P-wave and 0.801 for S-wave on their own test data set (with 78,592 samples). This showed a glorious improvement comparing to the results obtained by classical methods.

Part IV
Application to the simulated data set

IV.1 Adaptation of PhaseNet to our near surface context

The model in PhaseNet is trained on the data set that contains more than 0.7 million samples from natural earthquakes. Each sample consists of three seismograms corresponding to the signals recorded by three components. Each seismogram is recorded with 9000 time samples. Those are totally different from our real data (Table 1. Also, section V.1 describes the process of collecting the real data).

	PhaseNet data set	**Our real data set**
Type	Passive imaging: sources are earthquakes	Active imaging: sources are hammering, artificial explosion, etc.
Number of samples	Over 0.7 million labeled samples	9216 unlabeled samples
Input channels	3-component seismogram	1-component seismogram
Input length	9000 time samples	4000 time samples

Table 1: Differences between PhaseNet and our real data set

Thus, if we want the model to be adapted to our real data set, we have to retrain it, either from scratch, or from a pre-trained model (see IV.3.2). In this case, an approach based on simulated data was considered. The advantage of simulated data is that:

- we can build a physical model to simulate the data that are close to our real data,
- we can label for the simulated data by computing analytically the arrival times thanks to the given physical model.

IV.2 Generating a simulated data set

IV.2.1 Generating data based on SPECFEM2D

SPECFEM2D [20] is a computational software for 2D and 2.5D (i.e., axisymmetric) simulations of acoustic, elastic, viscoelastic, and poroelastic seismic wave propagation as well as FWI or adjoint tomography based on the spectral-element method (SEM). The SEM was originally developed in computational fluid dynamics and has been successfully adapted to address seismic wave propagation problems [29, 26]. SPECFEM was first developed by Dimitri Komatitsch and Jean-Pierre Vilotte at Institut de Physique du Globe (IPGP) in Paris, France from 1995 to 1997 and then, many progresses have been made over time by different teams and authors.

SPECFEM2D is written in Fortran2003 and performs a parallel programming based upon the Message Passing Interface (MPI). The latest release of the code includes support for GPU graphics card acceleration, that permits to remarkably speed up the simulation process. By using this software, we attempt to simulate the seismograms with 2 channels (2 components, here is X and Z) that are hopefully similar enough to our real data and thus, we can create a training data set for our model.

IV.2.1.1 Simple simulations

Some specific model parameters can be adjusted in order to achieve a diverse data set for our database. For instance, we can simulate with a range of natural states of the material depending on their densities, their absorbing boundary elements, the number of layers and the thickness between them, the phase speed at each layer as well as the nature of each layer (acoustic or elastic), etc.. We realize simple simulations by defining some flat interface surfaces, typically as Figure 16:

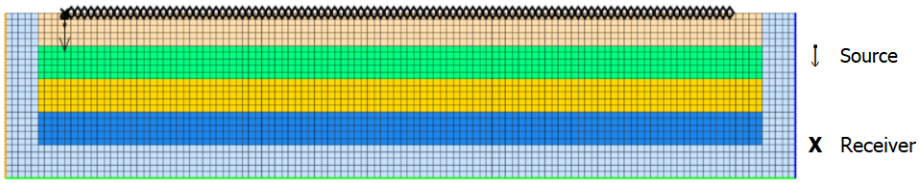

Figure 16: Simple interface model consisting of 5 layers and 6 flat interfaces with a length of 120m and a depth of 25m. We have 101 receivers in total and 1 source. Every two consecutive receivers are 1m far from each other. The dominant source frequency is 80Hz and the source type is Gaussian. The total number of time steps is 24000 and the time step of recording is 0.000025s. The element size is around 1m. The speed of P and S are from 900 to 4000m/s and from 500 to 2300m/s respectively.

The input parameters must respect to dispersion relations that describe the effect of dispersion on the properties of waves in a medium, as well as the convergence condition by Courant–Friedrichs–Lewy (CFL. In SPECFEM2D, $C_{CFL} \leq 0.68$) while solving certain partial differential equations numerically. To confirm the simulation results, we verify the spectrogram, the forward wave field and also the generated waveform (Figure 17).

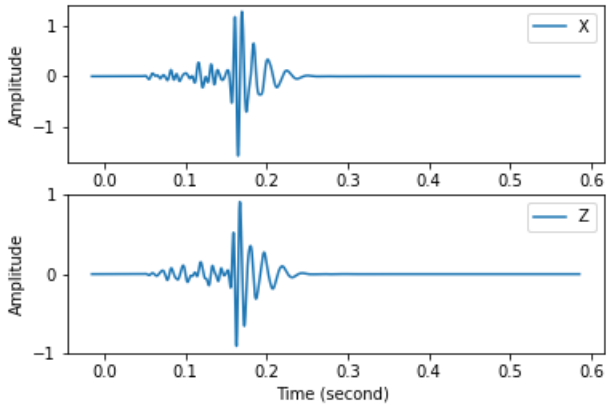

Figure 17: Simple seismic waveform simulated by SPECFEM2D

IV.2.1.2 Adding realistic topography

Now by adding a topography to the simulation, we can increase the complexity of the basement model to gain a nearest possible surface. Namely, based on the arrival times obtained by hand-picking methods, we can solve the seismic inversion problem by using ray tracing method (see II.3). Then, we can build a complex velocity model with 28 layers and 29 interfaces. Figure 18 shows the interface model in this case.

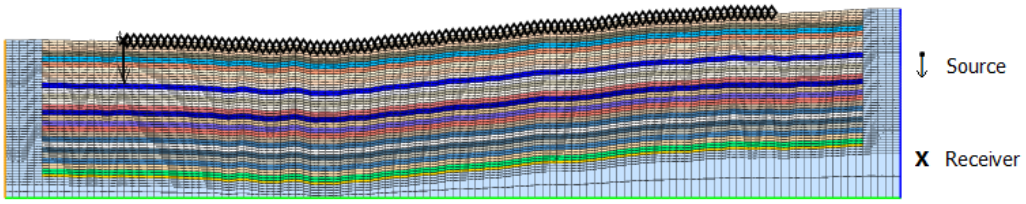

Figure 18: Interface model consisting of 28 layers and 29 interfaces with a length of 126m and a depth of 23.5 to 31m. We have 96 receivers in total and 1 source. Every two consecutive receivers are 1m far from each other. The dominant source frequency is 80Hz and the source type is Gaussian. The total number of time steps is 60000 and the time step of recording is 0.0000125s. The element size is around 1m. The speed of P and S are from 640 to 3900m/s and from 370 to 2200m/s respectively.

Figure 19 conveys a seismogram simulated by adding the topography, that has a waveform more complex than Figure 17 and is closer to reality.

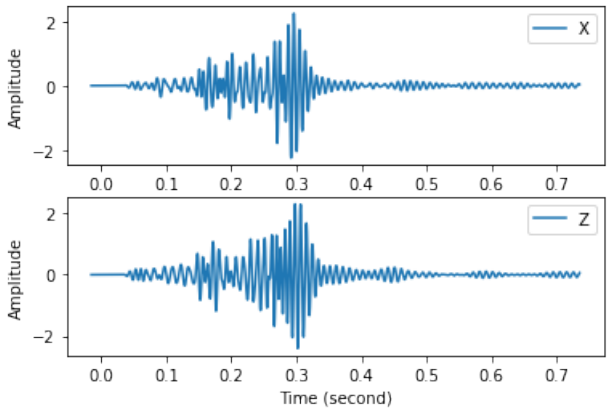

Figure 19: Example of a seismogram simulated by adding a topography

IV.2.2 Data augmentation

To increase the size of our data set from simulated database, we first use time shifting method that has been commonly used as data augmentation approach in seismic waveform treatment. For example, we run SPECFEM2D to simulate the seismograms with 2,000 time samples. Then, for each

seismogram, we randomly extract some parts from the original sample. Each extracted seismogram however consists of 1,000 time samples. We can choose the length of the time series we want to extract (here is 1,000), the number of extracted version from the original sample (for example, if we want to randomly selected 3 seismograms from each original seismogram, so the data size will increase 3 times) and a rate $\alpha \in [0, 1]$. With this rate, we have α chance of extracting the essential part containing the arrival times and $1 - \alpha$ chance of extracting the part which does not contain these arrival times. In addition to time shifting method, we reverse the waveform by multiplying the signal with -1 and further alter the original signal samples with noise of varying signal to noise ratios and evaluate the performance of the model under these noisy conditions. We focus on the way of using "Additive White Gaussian Noise" (AWGN) that is easy to generate. In AWGN, the noise is randomly sampled from a Gaussian distribution with mean value of zero and its standard deviation can vary depending on a so-called "Signal to Noise Ratio" (SNR). Considering a signal $S = (s_1, s_2, ...s_n)$, we recall that the root mean square (RMS) and the standard deviation (STD) of S are:

$$RMS_S = \sqrt{\frac{\sum_{i=1}^{n} s_i^2}{n}},$$

$$STD_S = \sqrt{\frac{\sum_{i=1}^{n} (s_i - \mu_S)^2}{n}},$$

where $\mu_S = \frac{1}{n} \sum_{i=1}^{n} s_i$ is the mean value of signal S. Now we define SNR as follows:

$$SNR = 10 \log \left(\frac{RMS_S^2}{RMS_{noise}^2} \right).$$

Hence, the required RMS of the noise that we generate is:

$$RMS_{noise} = \sqrt{\frac{RMS_S^2}{10^{\frac{SNR}{10}}}}.$$

As the mean value of noise is zero, so we must have $STD_{noise} = RMS_{noise}$. Finally, the noise that we generate by AWGN method is:

$$noise = (noise_1, noise_2, ...nosie_n) \sim \mathcal{N}_n(0, \sqrt{\frac{RMS_S^2}{10^{\frac{SNR}{10}}}} I_n).$$

Figure 20 shows some extraction cases using data augmentation from a original seismogram.

IV.3 Transfer learning

IV.3.1 Why do we use transfer learning?

Nowadays, many deep learning algorithms work well only under a data set even if their models achieved a significant precision with their data set. If we want to apply these methods for a newly collected data set that is drawn from another feature space and another distribution, then the models need to be rebuilt from scratch to learn about all features of the new data. For instance, PhaseNet employed the data based on *Northern California Earthquake Data Center Catalog* (NCEDC 2014) that has a lot of different characteristics compared to our real data set such as collected locations, collected methods and measurement conditions, etc.. Thus we can not just apply the model pre-trained with the NCEDC data to our real data. Either we retrain the model on our own data, or

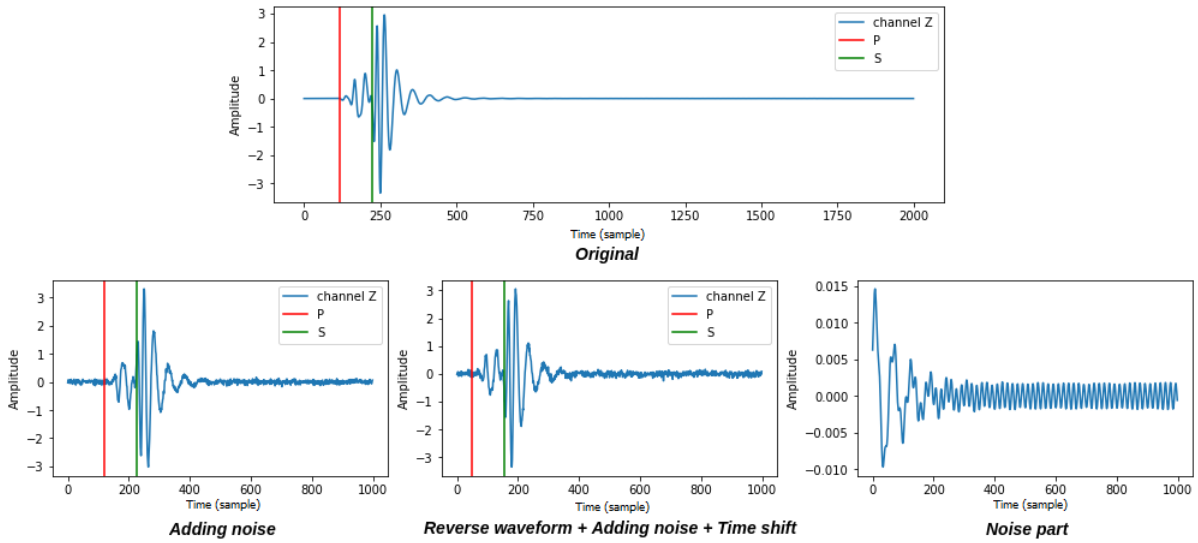

Figure 20: Different seismograms randomly selected from the original one with a random value of SNR from 15 to 30

we have to look for a technique which allows us to reuse the features the model has learnt in the past. In reality, it is more expensive to recollect the needed training data in order to rebuild the model. In [7], the original deep neural network model is retrained on local seismic data successfully with only 3,500 seismograms (0.45% of the data used to train the original model in PhaseNet). The so-called "transfer learning" is a methodology where a model trained on one task is repurposed on a second related task in order to improve performance on this second task.

In some cases, the transfer learning helps us train the model with better initial weights and so, enhance the learning performance. On the other hand, if the tasks are too dissimilar, the fact of implementing transfer learning may hinder performance of the training because of non-convexity initialization matters. The experiments in [34] compared the training performances (Figure 21) in the following three cases:

- no transfer training: train the model only on the data set B,
- similar transfer training: train the model on the data set B using transfer learning with the pre-trained model on the data set A (similar with B),
- dissimilar transfer training: train the model on the data set B using transfer learning with the pre-trained model on the data set A' (dissimilar with B).

IV.3.2 Different transfer learning strategies

The original code of PhaseNet was written in Tensorflow, a Python open-source library, that provides both high-level and low-level APIs. Tensorflow particularly showcases its efficiency when working with large data sets requiring an excellent functionality and a high performance. Keras on the other hand is a high-level neural network library that runs on top of Tensorflow. Normally, Keras is more user-friendly and easier to perform a neural network with small and medium data

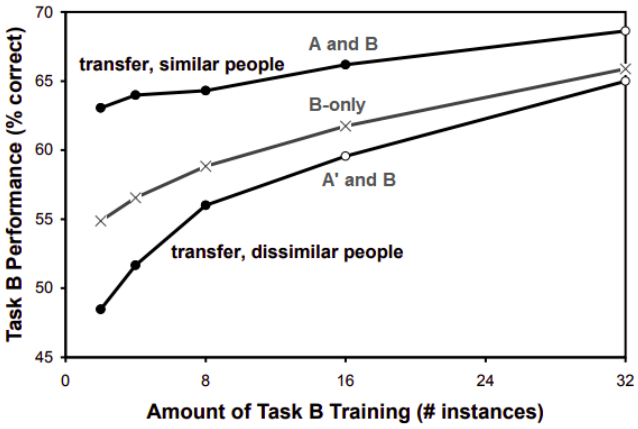

Figure 21: Effects of transfer learning in three cases
Taken from [34]

sets. Especially, it provides a convenient way to implement transfer learning with fine-tuning and more flexible choices.

Primarily, we need to convert all of PhaseNet code from Tensorflow to Keras. Then, once we train the model with a new data set using transfer learning, all required model weights are loaded from a pre-trained model. Subsequently, the network tries to learn new features of the new data using the knowledge of pre-trained model instead of initializing the model with random weights. Finally, the new model can be improved and adapted to the new data set using the knowledge in the past. In our case, the neural network takes only one channel, that means we apply only one filter for a single channel in the input layer instead of three for three channels as PhaseNet. Therefore, when loading weights from the pre-trained model into the input layer of our new model, we have to reshape the size of filter and also remove the weights based on the two redundant channels. We tried many transfer learning strategies based on two criteria:

- which layers are selected to load weights into,
- which layers are selected to freeze weights.

For example, in the scenario shown in Figure 23, we load weights from the whole pre-trained model into our new model and fine-tune all of these weights. In Figure 24, we do the same thing but the weights of two deepest layers are frozen. Namely, the weights in frozen layers are not updated during the training. In the configuration shown in Figure 25, we load weights only from downsampling and upsampling layers (i.e., the weights of input and output layers will be randomly initialized) and freeze weights of two deepest layers.

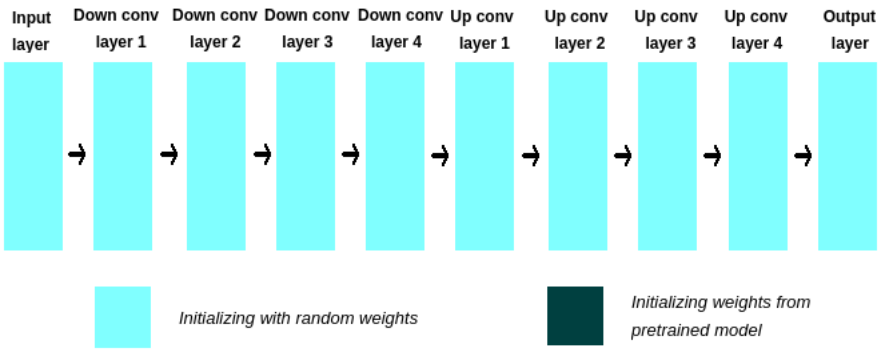

Figure 22: Training all layers from scratch (initializing random weights) - case 1

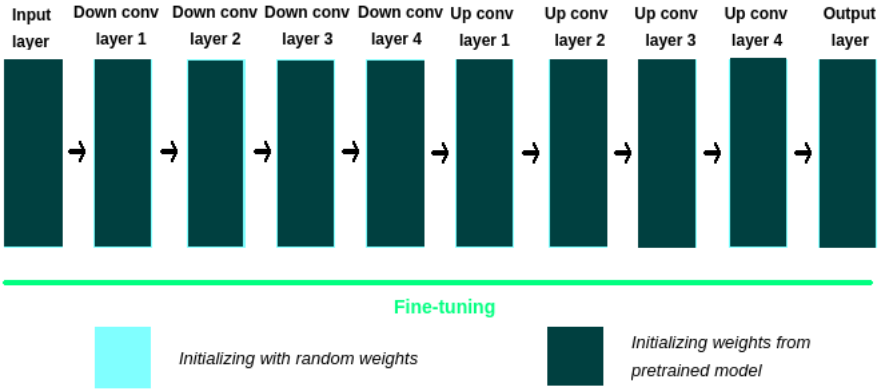

Figure 23: Initializing weights from all layers of pre-trained model and fine-tune all layers - case 2

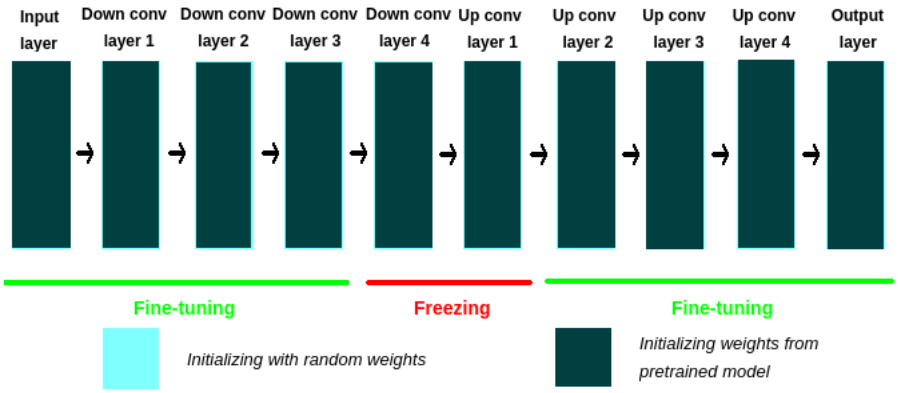

Figure 24: Initializing weights from all layers of pre-trained model and freezing two deepest layers - case 3

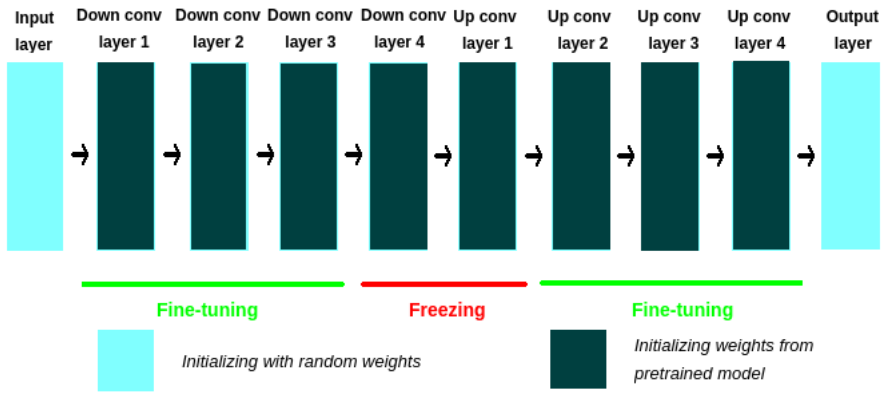

Figure 25: Initializing weights from only encoder-decoder layers of pre-trained model and freezing two deepest layers - case 4

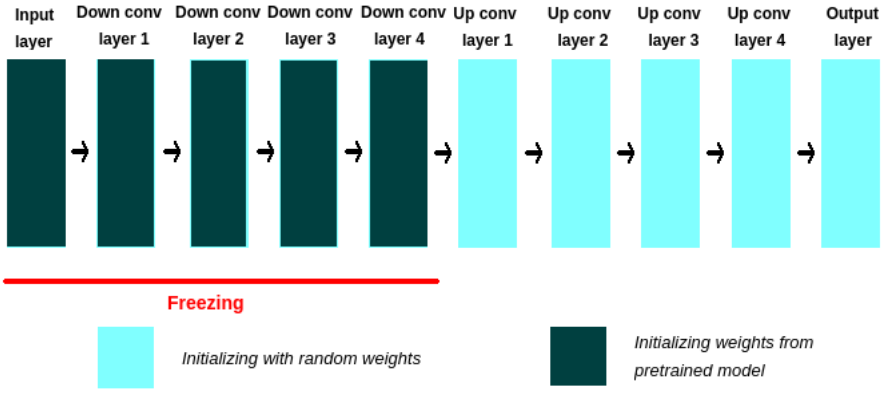

Figure 26: Initializing weights from only input and encoder layers of pre-trained model and freezing these layers - case 5

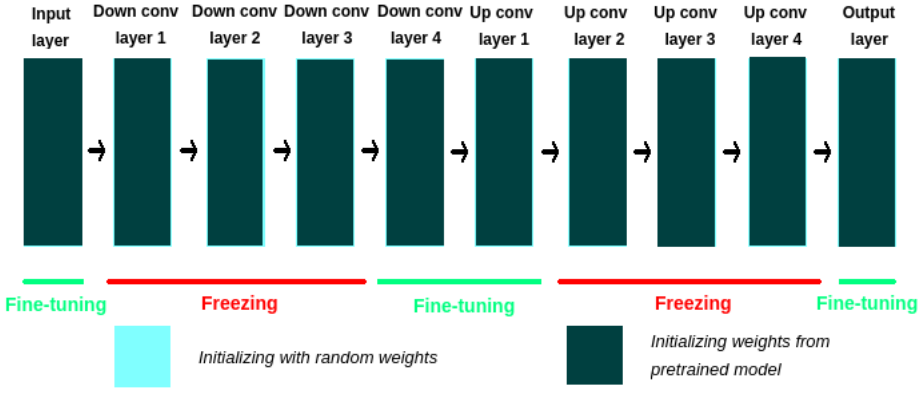

Figure 27: Initializing weights from the entire of pre-trained model and freezing some encoder and decoder layers - case 6

IV.4 Results on simulated data

In this section, we will present the performance of the network (with and without using transfer learning) on two data sets. The first data set is the one simulated without adding realistic topography, that is divided into a training-validation set and a test set with 3,890 and 480 samples respectively (after implementing data augmentation). The second one is the data simulated by adding realistic topography that is divided into a training-validation set and test set with 2,880 and 288 samples respectively (also after implementing data augmentation).

IV.4.1 Training and validation loss

Let us consider the model trained on the synthetic data without realistic topography. Based on Figure 28, we clarify that the models implemented using transfer learning (cases 2, 3, 4, 5, 6) learn the problem quickly and converge early compared to the case of training from scratch (case 1). The best curve is the one of case 3 while the one of case 2 seems to converge to a local minimum.

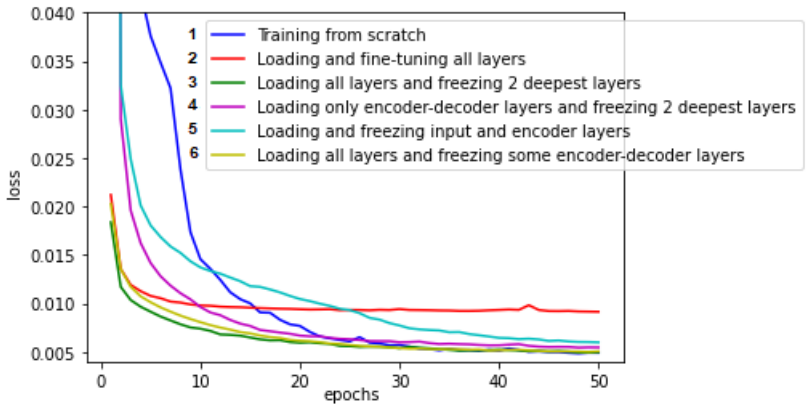

Figure 28: Comparison of training loss in different transfer learning ways without adding realistic topography to the simulated data set

Figure 29 showcases the training and validation loss in these different cases. We don't nearly have the problem of over-fitting or under-fitting in cases 1, 2, 3 while the three remaining cases may run into the problem of over-fitting in which the model "memorizes" the training samples and does not generalize well to samples from the test set. To avoid this phenomena, we can, for instance, employ some regularization parameters in order to simplify the complexity of the model.

IV.4.2 Scoring metrics

Table 2 shows the scoring metrics in different transfer learning strategies. We can interpret that the score of case 3 is the best among transfer learning strategies. Perhaps, the hidden features of deepest layers the neural network has learnt from the original data is much useful for our new data. By freezing these two deepest layers, the network just need to learn features of surface layers to adapt to the new data. Overall, picking for P-phase is not succeed as that for S-phase. This may come to the fact that P-wave is so much weaker than S-wave in our simulations. Consequently, the network may struggle to learn to detect the first arrival time for P-phase.

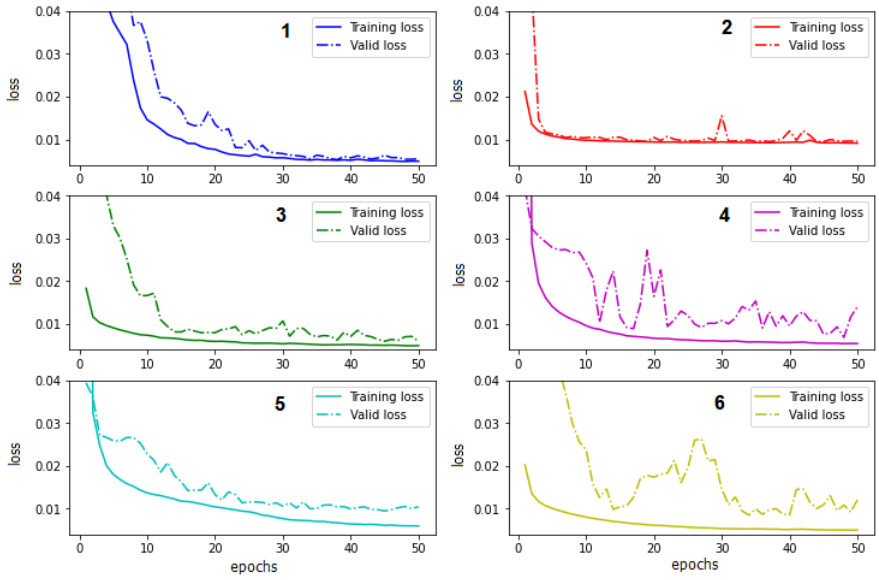

Figure 29: Training and validation loss in different transfer learning ways without adding realistic topography to the simulated data set

Simulated data without adding topography	**Phase**	**Precision**	**Recall**	**F1-score**
Training from scratch	P	0.694	0.612	**0.650**
	S	0.965	0.98	0.973
Loading and fine-tuning all layers	P	0.595	0.535	0.564
	S	0.983	0.987	0.985
Loading all layers and freezing 2 deepest layers	P	0.636	0.548	0.589
	S	0.983	0.999	**0.991**
Loading only down-upsampling layers and freezing 2 deepest layers	P	0.562	0.518	0.539
	S	0.967	0.963	0.965
Loading and freezing input and encoder layers	P	0.540	0.414	0.469
	S	0.980	0.985	0.982
Loading all layers and freezing some encoder-decoder layers	P	0.630	0.547	0.586
	S	0.998	0.980	0.989

Table 2: Scoring metrics for different transfer learning strategies without adding realistic topography to the simulated data set

Now, by training and testing on the synthetic data with realistic topography, we can see the impressive scores for both picking P and S phases in Table 3.

Simulated data with realistic topography	**Phase**	**Precision**	**Recall**	**F1-score**
Loading all layers and freezing 2 deepest layers	P	0.972	0.968	**0.970**
	S	0.954	0.954	**0.954**

Table 3: Scoring metrics by adding realistic topography to the simulated data set

Part V
Application to the real data set

V.1 Data collection

We describe here the real experimental setting done by the members of ER3 team of GET-OMP. To illustrate the experiments they realized for collecting the data, let us consider a wetland area in Guidel (a commune in the Morbihan department of Brittany in north-western France) for example. To monitor the water-saturated ground of this region, they survey a studied profile that crosses through the stream (Figure 30). Then, the sources and geophones (receivers) are lined up on the studied profile (Figure 31). The sources such as trucks or hammers are utilized to generate artificial explosions. Once an explosion takes place at a source, the waves will travel through the Earth's interior and be reflected to the receivers. Each receiver will then record the signals by only one component z into a seismogram. In the experiment, they consider 96 sources and 96 receivers, so the data consists of 9,216 ($= 96 \times 96$) seismograms recorded by a single component.

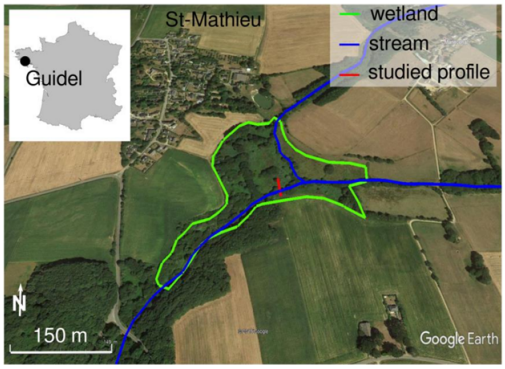

Figure 30: Studied zone for the experiment

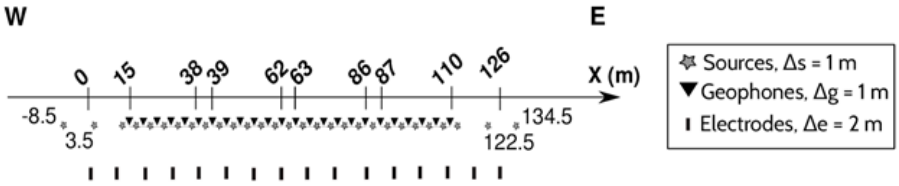

Figure 31: Measuring experiment method

V.2 Semi-supervised learning

V.2.1 Unlabeled data and an approach by semi-supervised learning

To train a neural network model based on our real data set with supervised learning, the data set has to be labeled. Since labeling for thousands of seismograms by hand-picking methods requires

too much time and effort, it is certainly affected by inconsistencies and blunders. But this does not imply that unlabeled data are useless for supervised tasks. We can even train our model with semi-supervised learning combining both unlabeled and labeled data.

The goal of semi-supervised learning is to employ both unlabeled and labeled samples to learn the underlying structure of the data step by step using pseudo labels. These pseudo labels are generated in pseudo-labeling, a process of using the labeled data model to predict labels for unlabeled data. In our case, the main idea is that: using a pre-trained model on a data set that is close to our real data, or a model trained on our small labeled data set, we can predict labels for our unlabeled and then make an effort to improve the quality of pseudo labels. This process can be repeated many times in order to increase the size of labeled data step by step (Figure 32). Once we get and correct all labels for our real data set, we can train a neural network model based on our own data.

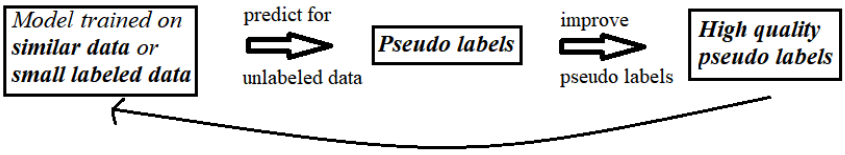

Figure 32: Applying pseudo-labeling to our case

V.2.2 Enhancing pseudo-labelling

Physically, the larger the distance between the source and the receiver, the larger the arrival time. For each source, let us consider a set of seismograms recorded by all of the receivers. After using labeled data model to predict for each unlabeled set (Figure 33), we can further improve the quality of pseudo labels by correcting these predictions using robust regression methods and SVR (Support Vector Regression).

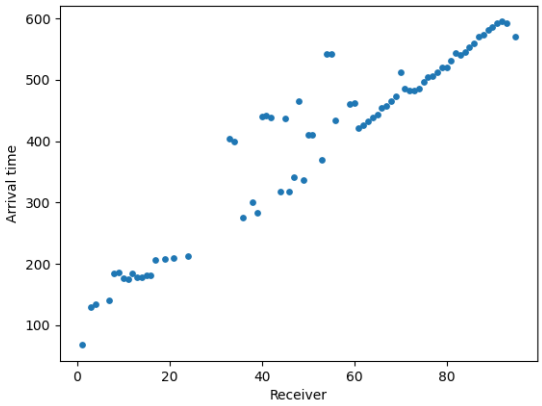

Figure 33: Example of S-phase prediction for unlabeled data at source 1

V.2.2.1 Anomaly detection using Huber regression

Robust regression methods provide an alternative to least squares regression by requiring less restrictive assumptions. These methods attempt to dampen outlier influences in the model to provide a better fit to the majority of the data. For example, Huber regression [18] uses a different loss function rather than the traditional least square by solving:

$$\min_{\beta\in\mathbb{R}^n} \sum_{i=1}^{m} H(y_i - x_i^T\beta) \text{ with the Huber function } H(r) = \begin{cases} r^2, \text{ if } |r| \le \epsilon \\ 2\epsilon r - \epsilon^2, \text{ if } |r| > \epsilon \end{cases}$$

where:

- y_i is the target,
- x_i is the predictor,
- β is the coefficient,
- $\epsilon > 0$ is a threshold that controls the number of samples that should be classified as outliers. For small residuals, the function is identical to the least squares penalty, but on large residuals, its penalty is lower and increases linearly rather than quadratically. It is thus more forgiving of outliers.

Now assuming that the trend of the data is a curve (non-linear), so we can use the logarithmic regression model instead (Figure 34, 35), we solve:

$$\min_{\beta\in\mathbb{R}^n} \sum_{i=1}^{m} H(y_i - (\log x_i^T)\beta).$$

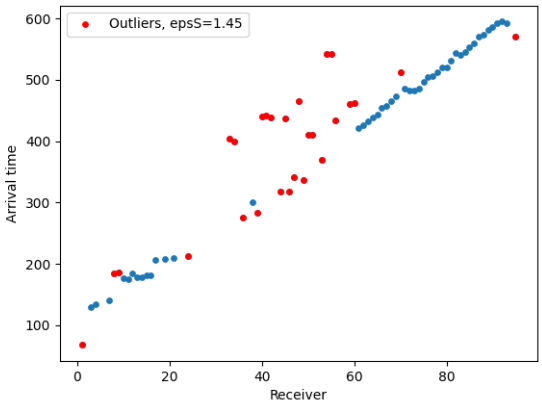

Figure 34: Outlier detection for S-phase using linear Huber regression

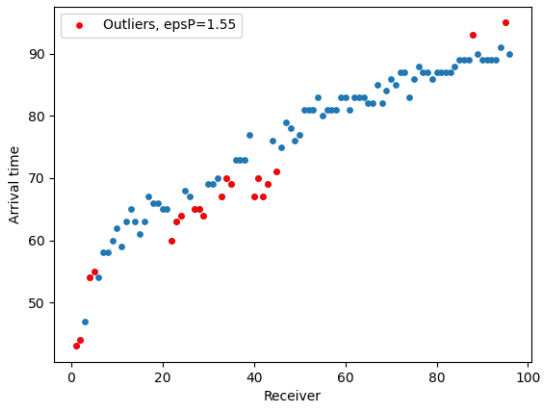

Figure 35: Outlier detection for P-phase using logarithmic Huber regression

V.2.2.2 Correcting pseudo labels using Support Vector Regression

Although we observed that the data almost take the shape of a line (or a curve), we have no reason to assume that the linear regression model (or logarithmic regression model) is the best fit to the data. But at least, robust regression methods helped us detect the anomalies that are unusually far from other observations. Now we apply SVR to the data with standard observations.

The idea of SVR is based on SVM (Support Vector Machine), that originates from the VC theory developed by Vladimir Vapnik and Alexey Chervonenkis during 1960-1990 and was first proposed for utilization in the regression tasks in [10]. While many regression algorithms attempt to minimize the sum squared errors with and without additional penalty parameters that aims to reduce the number of features used in the final model, SVR gives us the flexibility to define how much error is acceptable in finding an appropriate line or hyperplane to fit the data. The objective is to minimize the $L2$-norm of the coefficient vector (not the squared error) and additional deviation based on ξ-margin. This is handled in the constraints, where the absolute error is set less than or equal to a maximum error ϵ:

$$\min_{\beta,b,\xi,\xi^*} \frac{1}{2}\beta^T\beta + C\sum_{i=1}^{m}(\xi_i + \xi_i^*)$$

subject to:

$$y_i - \beta^T\phi(x_i) - b \leq \epsilon + \xi_i,$$

$$\beta^T\phi(x_i) + b - y_i \leq \epsilon + \xi_i^*,$$

$$\xi_i, \xi_i^* \geq 0, i = 1, ..., m$$

where:

- y_i is the target,
- x_i is the predictor,
- β is the coefficient,
- b is the intercept,

- ϵ is the maximum error,
- $\phi(.)^T\phi(.) = K(.,.)$ is the kernel,
- ξ, ξ^* are the margins that penalize samples whose prediction is at least ϵ away from their true target,
- C is the regularization parameter, that is inversely proportional to the strength of the regularization.

Figure 36 and Figure 37 show the improvements of SVR models for P and S-phase based on the data without outliers (detected by using Huber regression) compared to least squares methods.

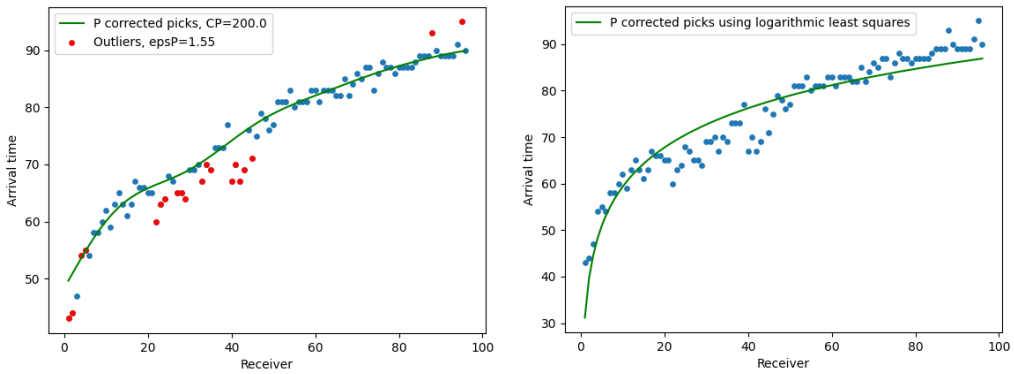

Figure 36: Correcting labels for P-phase using SVR and Huber regression (left) and using logarithmic least squares (right)

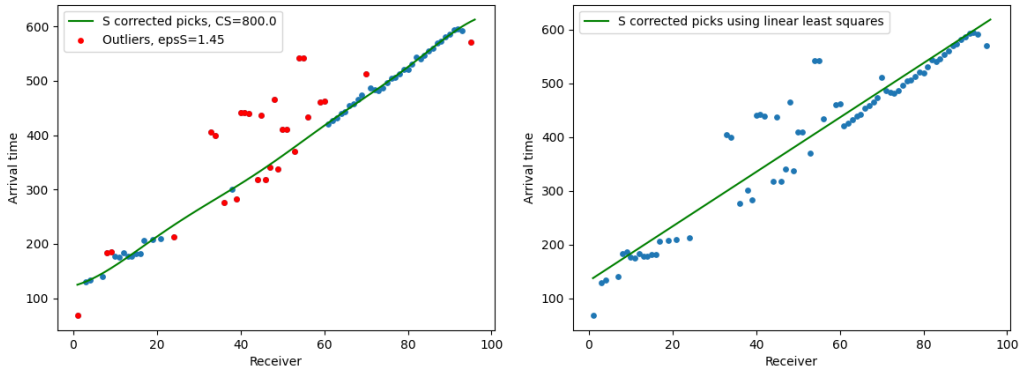

Figure 37: Correcting labels for S-phase using SVR and Huber regression (left) and using linear least squares (right)

V.3 Results on real data

Figure 38 compares the training and validation loss on the real data with and without transfer learning. The loss function of the model using transfer learning starts converging after 25 epochs while that of the model trained from scratch still shows no signs of convergence until epoch 25 and may run into the over-fitting situation. These results can demonstrate the efficiency of using transfer learning in our case, especially when comparing the scores obtained by the two models in Table 4.

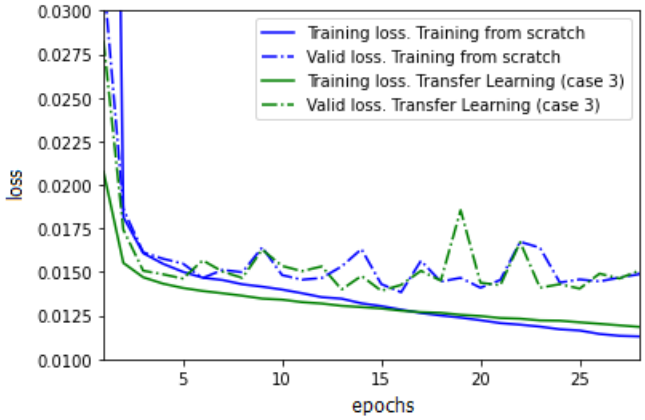

Figure 38: Training and validation loss on real data

Real data	**Phase**	**Precision**	**Recall**	**F1-score**
Training from scratch	P	0.9993	0.9995	0.9994
	S	0.8302	0.8259	0.8280
Loading all layers and freezing 2 deepest layers	P	0.9992	0.9998	**0.9995**
	S	0.8319	0.8283	**0.8301**

Table 4: Scoring metrics for models trained on real data

Figure 39 shows some prediction examples. The sub-figures (i), (ii), (iv), (v) and (vi) are the examples with both P and S-waves while the sub-figure (iii) represents an example without P and S-waves (that is the noise part). The detection of P-wave is much easier than S-wave because the first signal of P-wave is more energetic and clearly distinguished from the noise. Picking S-wave is harder when the source and the receiver are too close to each other, the S-wave is thus difficult to detect because it emerges from the noise and P-wave (sub-figures (i) and (iv)). This is however easier when the source is far enough away from the receiver and thus, the signal of S-wave is distinguished from the noise and P-wave (sub-figures (ii), (v) and (vi)).

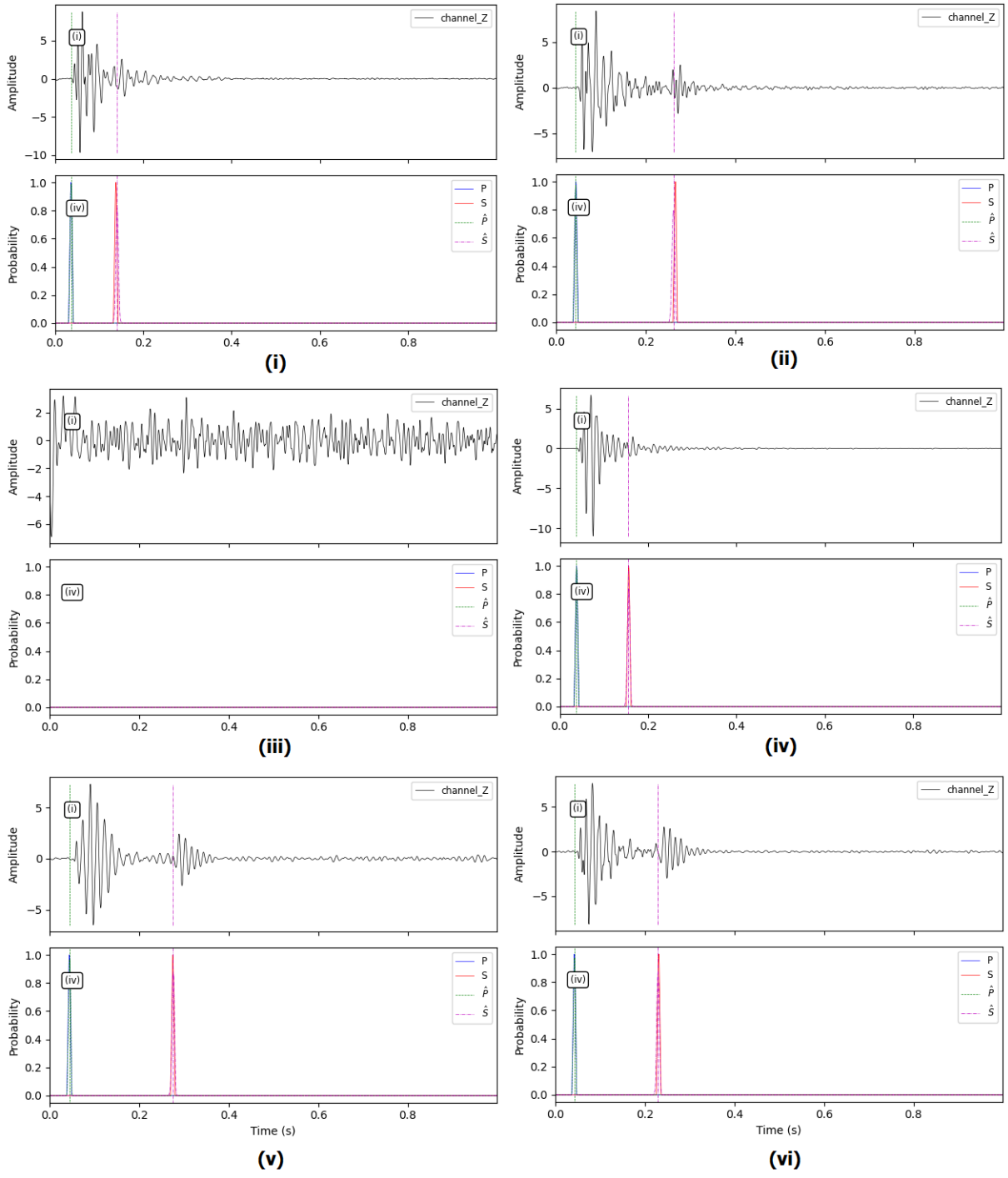

Figure 39: Some predictions while testing on real data. The upper sub-figures are the seismograms. The lower sub-figures are the predicted probability distributions of P-wave ($\hat{P}$) and S-wave ($\hat{S}$). The blue and red vertical lines are the true P and S arrival times.

Part VI
Conclusion

Deep learning methods for seismic inversion problem are being improved rapidly. An end-to-end deep learning is however unachievable because of the complexity and the lack of labeled data set. Automatic arrival time picking is one of the less expensive method in helping resolve the inversion problem. In spite of that, labeling of thousands of seismograms to train a deep neural network is still complicated. We proposed an approach for automatic arrival time picking with unlabeled data based on PhaseNet. By generating a data set that is close to our real data with SPECFEM2D, we tested many transfer learning strategies on the simulated data set and selected the best one to help the labeling of our real data. Moreover, the robust regression methods and SVM are employed in order to enhance the quality of pseudo-labels. The models trained on the simulated and real data finally showcased the very high scores for the detection of P and S wave. This can demonstrate the efficiency of automatic time picking by deep learning in helping resolve the seismic inversion problem.

In the future, if we collect more diverse data (for instance, collecting data from different regions or laboratory experiments, etc.), we can improve the neural network with these diverse data and adjust some parameters such as the depth, regularization, optimizer type, etc. of the network. Furthermore, we can try to make use of other approaches to resolve or help to resolve the seismic inversion problem by deep learning, that are mentioned in II.2 and II.3.

To conclude on this internship, I highly appreciate the professionalism in the workplace where I definitely enjoy my job and get along with my coworkers. Through the project, I improved my hard skills such as programming, machine learning and deep learning, but also my soft skills such as communication and teamwork. Furthermore, I had the chance to find out some related problems in physics and geophysics, as well as the link between AI and geology. For the most part, I could better understand some current challenges in applying data science to resolve a realistic problem in different fields.

References

[1] A. Adler, M. Araya-Polo, and T. Poggio, *Deep learning for seismic inverse problems: Toward the acceleration of geophysical analysis workflows*, IEEE Signal Processing Magazine, 38 (2021), pp. 89–119.

[2] T. Akazawa, *A technique for automatic detection of onset time of p-and s-phases in strong motion records*, in Proc. of the 13th World Conf. on Earthquake Engineering, Vancouver, Canada, 2004.

[3] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *Automatic differentiation in machine learning: a survey*, Journal of machine learning research, 18 (2018).

[4] F. Billette and G. Lambare, *Velocity macro-model estimation from seismic reflection data by stereotomography*, Geophysical Journal International, 135 (1998), pp. 671–690.

[5] R. P. Bording, A. Gersztenkorn, L. R. Lines, J. A. Scales, and S. Treitel, *Applications of seismic travel-time tomography*, Geophysical Journal International, 90 (1987), pp. 285–303.

[6] D. Cao and W. Liao, *A computational method for full waveform inversion of crosswell seismic data using automatic differentiation*, Computer Physics Communications, 188 (2015), pp. 47–58.

[7] C. Chai, M. Maceira, H. J. Santos-Villalobos, S. V. Venkatakrishnan, M. Schoenball, W. Zhu, G. C. Beroza, C. Thurber, and E. C. Team, *Using a deep neural network and transfer learning to bridge scales for seismic phase picking*, Geophysical Research Letters, 47 (2020), p. e2020GL088651.

[8] T. Dai, J. Xia, L. Ning, X. Chaoqiang, Y. Liu, and H. Xing, *Deep learning for extracting dispersion curves*, Surveys in Geophysics, 42 (2021), pp. 1–27.

[9] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, *The importance of skip connections in biomedical image segmentation*, in Deep Learning and Data Labeling for Medical Applications, G. Carneiro, D. Mateus, L. Peter, A. Bradley, J. M. R. S. Tavares, V. Belagiannis, J. P. Papa, J. C. Nascimento, M. Loog, Z. Lu, J. S. Cardoso, and J. Cornebise, eds., Cham, 2016, Springer International Publishing, pp. 179–187.

[10] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, V. Vapnik, et al., *Support vector regression machines*, Advances in neural information processing systems, 9 (1997), pp. 155–161.

[11] A. Fichtner, H.-p. Bunge, and H. Igel, *The adjoint method in seismology: I. theory*, Physics of the Earth and Planetary Interiors, 157 (2006), pp. 86–104.

[12] S. Fomel, S. Luo, and H. Zhao, *Fast sweeping method for the factored eikonal equation*, Journal of Computational Physics, 228 (2009), pp. 6440–6455.

[13] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 770–778.

[14] J. W. D. Hobro, S. C. Singh, and T. A. Minshull, *Three-dimensional tomographic inversion of combined reflection and refraction seismic traveltime data*, Geophysical Journal International, 152 (2003), pp. 79–93.

[15] J. A. Hole, *Nonlinear high-resolution three-dimensional seismic travel time tomography*, Journal of Geophysical Research: Solid Earth, 97 (1992), pp. 6553–6562.

[16] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, *Densely connected convolutional networks*, in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[17] G. Huang, S. Luo, T. Ari, H. Li, and D. C. Nobes, *First-arrival tomography with fast sweeping method solving the factored eikonal equation*, Exploration Geophysics, 50 (2019), pp. 144–158.

[18] P. J. Huber, *Robust Estimation of a Location Parameter*, The Annals of Mathematical Statistics, 35 (1964), pp. 73–101.

[19] I. F. Jones, *Tutorial: Velocity estimation via ray-based tomography*, first break, 28 (2010).

[20] D. Komatitsch, J.-P. Vilotte, P. Cristini, J. Labarta, N. Le Goff, P. Le Loher, Q. Liu, R. Martin, R. Matzen, C. Morency, D. Peter, C. Tape, J. Tromp, and Z. Xie, *Specfem2d v7.0.0 [software]*, 2012.

[21] D. Kosloff, J. Sherwood, Z. Koren, E. Machet, and Y. Falkovitz, *Velocity and interface depth determination by tomography of depth migrated gathers*, GEOPHYSICS, 61 (1996), pp. 1511–1523.

[22] I. Lecomte, P. Lubrano-Lavadera, I. Anell, S. Buckley, D. W. Schmid, and M. Heeremans, *Ray-based seismic modeling of geologic models: Understanding and analyzing seismic images efficiently*, Interpretation, 3 (2015), pp. SAC71–SAC89.

[23] Y. LeCun, Y. Bengio, and G. Hinton, *Deep learning*, Nature, 521 (2015), pp. 436–444.

[24] S. Li, B. Liu, Y. Ren, Y. Chen, S. Yang, Y. Wang, and P. Jiang, *Deep-learning inversion of seismic data*, IEEE Transactions on Geoscience and Remote Sensing, 58 (2020), pp. 2135–2149.

[25] J. Long, E. Shelhamer, and T. Darrell, *Fully convolutional networks for semantic segmentation*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 3431–3440.

[26] Y. Maday and A. T. Patera, *Spectral element methods for the incompressible Navier-Stokes equations*, in IN: State-of-the-art surveys on computational mechanics (A90-47176 21-64). New York, Jan. 1989, pp. 71–143.

[27] S. M. Mousavi, W. L. Ellsworth, W. Zhu, L. Y. Chuang, and G. C. Beroza, *Earthquake transformer: an attentive deep-learning model for simultaneous earthquake detection and phase picking*, Nature communications, 11 (2020), pp. 1–12.

[28] H. Noh, S. Hong, and B. Han, *Learning deconvolution network for semantic segmentation*, in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1520–1528.

[29] A. T. Patera, *A spectral element method for fluid dynamics: Laminar flow in a channel expansion*, Journal of Computational Physics, 54 (1984), pp. 468–488.

[30] P. Podvin and I. Lecomte, *Finite difference computation of traveltimes in very contrasted velocity models: a massively parallel approach and its associated tools*, Geophysical Journal International, 105 (1991), pp. 271–284.

[31] J. Qian, Y.-T. Zhang, and H.-K. Zhao, *Fast sweeping methods for eikonal equations on triangular meshes*, SIAM J. Numerical Analysis, 45 (2007), pp. 83–107.

[32] A. Richardson, *Seismic full-waveform inversion using deep learning tools and techniques*, (2018).

[33] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, in International Conference on Medical image computing and computer-assisted intervention, Springer, 2015, pp. 234–241.

[34] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, *To transfer or not to transfer*, in NIPS 2005 workshop on transfer learning, vol. 898, 2005, pp. 1–4.

[35] M. M. Selim Saleh, *Body waves*, in Encyclopedia of Solid Earth Geophysics, H. K. Gupta, ed., Dordrecht, 2011, Springer Netherlands, pp. 29–35.

[36] K. Simonyan and A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, arXiv preprint arXiv:1409.1556, (2014).

[37] M. Talwani and W. Kessinger, *Exploration geophysics*, in Encyclopedia of Physical Science and Technology (Third Edition), R. A. Meyers, ed., Academic Press, New York, third edition ed., 2003, pp. 709–726.

[38] J. Virieux, A. Asnaashari, R. Brossier, L. Métivier, A. Ribodetti, and W. Zhou, *An introduction to full waveform inversion*, in Encyclopedia of exploration geophysics, Society of Exploration Geophysicists, 2017, pp. R1–R40.

[39] J. Wang, Z. Xiao, C. Liu, D. Zhao, and Z. Yao, *Deep learning for picking seismic arrival times*, Journal of Geophysical Research: Solid Earth, 124 (2019), pp. 6612–6624.

[40] S. Xu, D. Wang, F. Chen, Y. Zhang, and G. Lambare, *Full waveform inversion for reflected seismic data*, in 74th EAGE Conference and Exhibition incorporating EUROPEC 2012, European Association of Geoscientists & Engineers, 2012, pp. cp–293.

[41] F. Yang and J. Ma, *Deep-learning inversion: A next-generation seismic velocity model building method*, GEOPHYSICS, 84 (2019), pp. R583–R599.

[42] Y. Zheng, Q. Zhang, A. Yusifov, and Y. Shi, *Applications of supervised deep learning for seismic interpretation and inversion*, The Leading Edge, 38 (2019), pp. 526–533.

[43] W. Zhu and G. C. Beroza, *PhaseNet: a deep-neural-network-based seismic arrival-time picking method*, Geophysical Journal International, 216 (2018), pp. 261–273.

[44] W. Zhu, K. Xu, E. Darve, and G. C. Beroza, *A general approach to seismic inversion with automatic differentiation*, Computers & Geosciences, 151 (2021), p. 104751.