WORK PLACEMENT REPORT

Automatic arrival time picking for seismic inversion

Ngo Nghi Truyen Huynh

August 2021

Tutors:

Dr. Roland Martin

Prof. Thomas Oberlin

Dr. Bastien Plazolles

Institut National des Sciences Appliquées de Toulouse

Département Génie Mathématique et Modélisation

Abstract

sfcsfsa

Acknowledgment

Contents

I Introduction **5**

II State-of-the-art of seismic imaging techniques **6**

II.1 Seismic inversion problem **6**

II.2 Approaching by arrival time picking method **6**
II.2.1 Introduction to body waves 6
II.2.2 Data collection 7
II.2.3 Automatic arrival time picking by deep learning 8

III Automatic arrival time picking based on PhaseNet **9**

III.1 U-net architecture **9**
III.1.1 Transpose convolutional layers: Deconvolution 9
III.1.2 Skip connections 10
III.1.3 Architectural model 12

III.2 Label generation **13**
III.2.1 Generating segmentation map from arrival time 13
III.2.2 Detecting arrival time from output segmentation map 14

III.3 Testing and scoring metrics **16**

IV Application to the simulated data **17**

IV.1 Adaptation of PhaseNet to our near surface context **17**

IV.2 Generating a simulated data set **17**
IV.2.1 Generating data based on SPECFEM2D 17
IV.2.1.1 Simple simulations 18
IV.2.1.2 Adding realistic topography 18
IV.2.2 Data augmentation 20
IV.2.3 Adding noises 20

IV.3 Transfer learning **21**
IV.3.1 Why do we use transfer learning? 21
IV.3.2 Different transfer learning strategies 22

IV.4 Results on simulated data **25**
IV.4.1 Training and validation loss 25
IV.4.2 Scoring metrics 27

V Application to the real data set **29**

V.1 Semi-supervised learning **29**

V.1.1 Unlabeled data issue and an approach by semi-supervised learning 29

V.1.2 Improving pseudo-labelling 29

V.1.2.1 Anomaly detection using Huber regression 29

V.1.2.2 Correcting pseudo labels using Support Vector Regression 30

V.2 Results on real data **32**

VI Conclusion and perspectives **34**

References **35**

Part I
Introduction

This work is related to a GitHub code available on:

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

Monitoring of basement structures and water-saturated ground .....

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

II.1 Seismic inversion problem

Research bib for inversion problem .....

II.2 Approaching by arrival time picking method

Explain how arrival time picking can help to solve inversion problem???????????

II.2.1 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 [18]. 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 its 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).

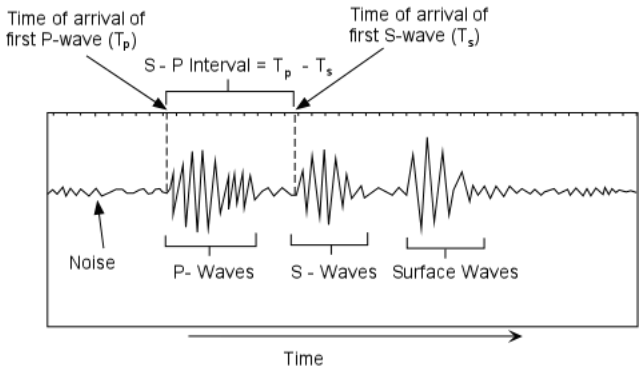

Figure 1: Body waves on a seismogram
Source: `http://earthsci.org/education/teacher/basicgeol/earthq/earthq.html`

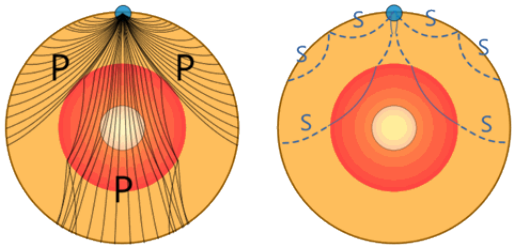

Figure 2: Motion of P-wave and S-wave
Source: `https://www.mathsisfun.com/physics/waves-seismic.html`

II.2.2 Data collection

To illustrate the experiments we 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, we survey a studied profile that traverses through the stream (Figure 3). Then, the sources and geophones (receivers) are lined up on the studied profile (Figure 4). The sources such as trucks 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, we consider 96 sources and 96 receivers, so our data contain 9,216 ($= 96 \times 96$) seismograms recorded by a single component.

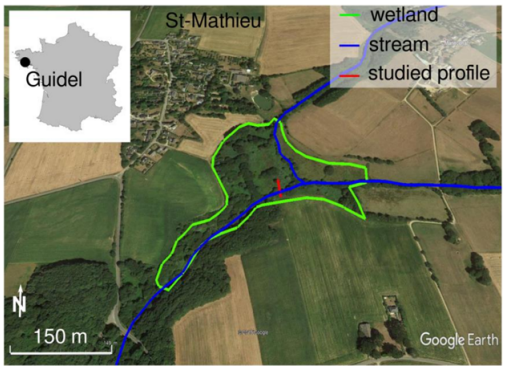

Figure 3: Studied zone for the experiment

The P-waves propagate in the same direction with the z-axis while the S-waves are slower and propagate in the direction perpendicular to the P-waves movement. Physically, the z-axis records so more information of the P-wave while most of S-wave information recorded by the x and y-axis. Consequently, the fact that our measurement based on a single component may result in a lack of information on the S-wave detection.

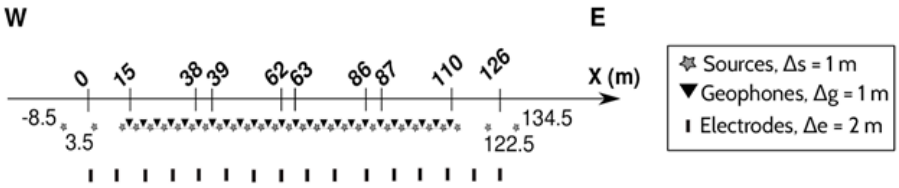

Figure 4: Measuring experiment method

II.2.3 Automatic arrival time picking by deep learning

Previously, we used the hand-picking methods to solve the inversion problem with time picking. For instance, we first identify the arrival time by hand-picking and then reconstruct the seismic velocity by using ray tracing method [11, 14, 15, 5, 8, 1] (Figure 5). Nevertheless, the hand-picking methods require too much effort and can be affected by inconsistencies. Therefore, we are attempting to develop automatic-picking methods like Deep Learning approaches to calculate these arrival times.

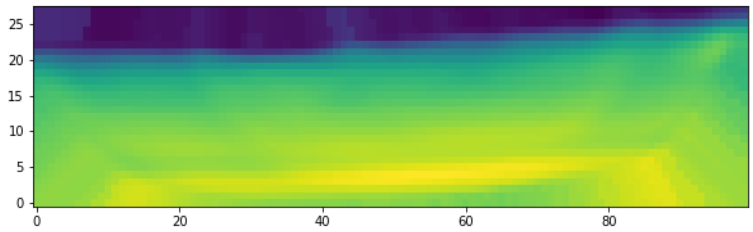

Figure 5: Seismic velocity model

Research bib for classical arrival time picking in order to compare to deep learning method.....
Research more about time picking methods by deep learning..... [19]

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 the models which do not utilize the deep neural network, particularly, a very higher precision of predicted time picking with S-waves in [20]. This model is an U-net that is used for biomedical image segmentation and particularly, has shown its efficiencies for segmentation of neuronal structures in electron microscopic stacks in [16]. We will see in III.1.1 and III.1.2 two essential techniques that are used in U-net and PhaseNet.

III.1 U-net architecture

III.1.1 Transpose convolutional layers: Deconvolution

Transposed convolution, or deconvolution, or upsampling is now appearing in many CNN architectures. This technique exists in symmetrical architectures (encoder-decoder) and can be used to reconstruct the original representation of a convolution filter map by using transposed convolution. The deep deconvolution network proposed in [13] 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). But how does this work exactly? Let us have a look at a normal convolution.

Considering a simple example that we want to apply a convolution with a 2x2 kernel on a 3x3 input image as below:

$$input: \begin{bmatrix} 2 & 1 & 4 \\ 1 & 3 & 1 \\ 5 & 4 & 2 \end{bmatrix} \text{*convolution operation*} \ kernel: \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}.$$

We assume a stride of 1, so the convolution process consists in that the kernel the kernel is first placed at the upper left corner of the input matrix. Subsequently, it performs a Hadamard product (element-wise multiplication) with the overlapping area of the input. We repeat this operation for the next position of the kernel (for example, one step shift to the right) and so on. Thus, we obtain the convolution below:

$$\begin{bmatrix} 2 & 1 & 4 \\ 1 & 3 & 1 \\ 5 & 4 & 2 \end{bmatrix} * \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} = \begin{bmatrix} 12 & 11 \\ 18 & 15 \end{bmatrix}.$$

Now, if we consider a flatten input matrix of the size 9x1 instead of 3x3, and a convolution matrix that demonstrates all positions of the kernel on the input (i.e. each row of this convolution matrix represents a position of the kernel on top of the input). Hence, the convolution operation is just like the matrix multiplication between the convolution matrix and the flatten input matrix:

$$\begin{bmatrix} 2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 & 0 \\ 0 & 2 & 1 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 & 1 & 0 & 1 & 2 \end{bmatrix} . \begin{bmatrix} 2 \\ 1 \\ 4 \\ 1 \\ 3 \\ 1 \\ 5 \\ 4 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 18 \\ 15 \end{bmatrix} \longrightarrow \begin{bmatrix} 12 & 11 \\ 18 & 15 \end{bmatrix}.$$

So, if we try to go backward from a summarized version of the original input to the original input, or at least something that hopefully looks like it, we just need to try to multiply the transposed convolution matrix with the flatten compressed input matrix. For instance, we want to perform an upsampling for a compressed matrix $\begin{bmatrix} 1 & 2 \\ 3 & 5 \end{bmatrix}$ with the same kernel in the previous example, so the decompression process is simply described as below:

$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 2 & 0 \\ 2 & 1 & 1 & 2 \\ 0 & 2 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 2 \end{bmatrix} . \begin{bmatrix} 1 \\ 2 \\ 3 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 2 \\ 7 \\ 17 \\ 9 \\ 3 \\ 11 \\ 10 \end{bmatrix} \longrightarrow \begin{bmatrix} 2 & 5 & 2 \\ 7 & 17 & 9 \\ 3 & 11 & 10 \end{bmatrix}.$$

III.1.2 Skip connections

Currently, skip connection is a standard module in many Convolutional Neural Network (CNN) architectures. It plays a very important role in the fact of building a "fully convolutional network" (FCN) [12]. Based on a CNN, we can build a typical FCN by:

- adding an expanding path (decoder) called "long skip connections" (Figure 6a) 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 6b) that allows us to increase the convergence speed of the loss function and build very deep neural networks that can contain hundreds of layers.

To understand how does the skip connection work, we first go back to understand a little bit of backpropagation algorithm. As we know, neural networks can learn their weights and biases using the gradient descent algorithm in order to optimize these parameters with respect to the loss function. The backpropagation algorithm provides us a way to calculate the gradient of the loss function by computing the partial derivatives. However, in some cases, we see that the loss function stops decreasing but it is still far from the desired result at a certain step of updating parameters during the training. The so-called "vanishing gradient problem" causes an uninterrupted gradient flow from the first layer to the last layer in our model.

Mathematically, if W is a parameter that we want to learn for during the training, then the updating step is:

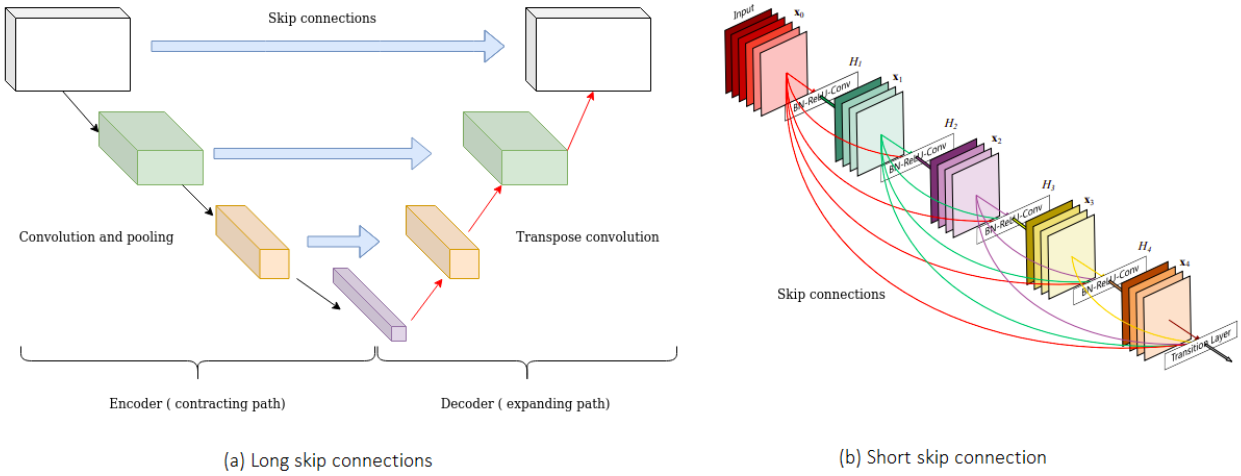

Figure 6: Two kinds of setup of skip connection

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

where λ is the learning rate and L is the loss function. Now we assume 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 7:

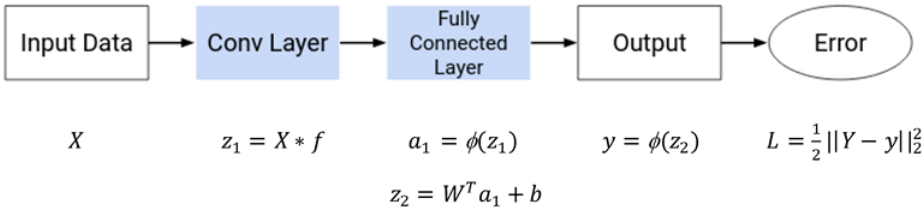

Figure 7: Simple architecture of CNN

Then, for calculating the quantity $\frac{\partial L}{\partial W}$, we need to go backward to calculate the partial derivatives:

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

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 the problem of vanishing gradient.

Thus, skip connections allow to avoid the vanishing gradient problem. 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 [6] or addictive skip connection back-propagates through the identity function by using a vector addition (Figure 8a) while DenseNet

[7] or concatenative skip connection allows to avoid the fact of having low-level information shared between the input and output, by concatenation of previous feature maps (Figure 8b).

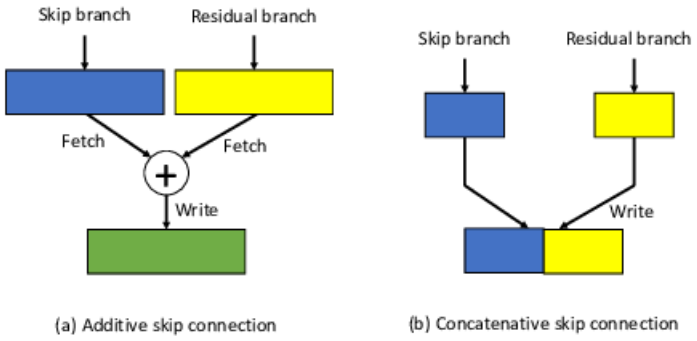

Figure 8: Additive connection and concatenative skip connection
Source: `https://www.researchgate.net/figure/Additive-skip-connections-vs-concatenative-skip-connections-Rectangles-represent-data_fig1_329115979`

Go back to the long skip connections that are used in symmetrical architecture (convolution-deconvolution or encoder-decoder) as U-net or PhaseNet. 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 [3] showed that adding long skip connections in the encoder-decoder architecture can achieve an incredibly effective work in medical image segmentation.

III.1.3 Architectural model

Figure 9 shows the architecture of PhaseNet with two symmetrical branches containing 4 downsampling stages and 4 upsampling stages. Each stage is a convolution followed by ReLU activation function. The downsampling stages compress the input signal and reduce the size of the data. Afterwards, the up-sampling stages expand and convert the useful information into probability distributions of the outputs for each time point. The skip connection used in each up-sampling stage allows to concatenate directly the left output to the right layer without going through the deep layers between them.

The inputs are three-component seismograms where the labels are the corresponding arrival times[1], and the outputs are probability distributions of P-waves, S-waves and noise. Let x be a time series of the input, $z(x)$ be the unscaled values of the last layer and $j = 1, 2, 3$ represent noise, P-wave and S-wave respectively. We note that, at time t, the probability that we can observe noise is equal to one minus the probability that we can observe the P-wave and minus the probability that we can observe the S-wave:

$$z_1(t) = 1 - z_2(t) - z_2(t).$$

[1]PhaseNet takes the arrival times as labels but what the network really takes is the probability distributions of the arrival times (see in III.2).

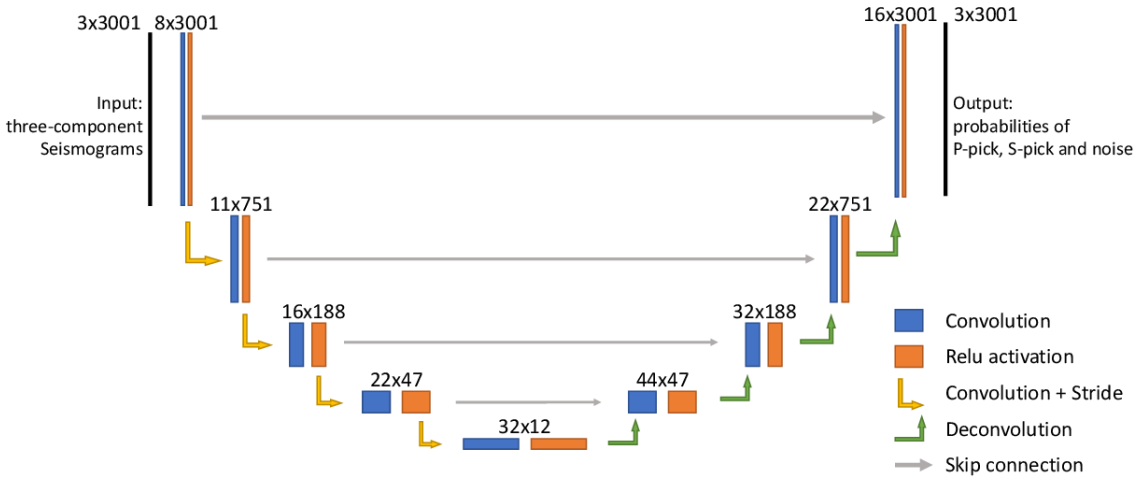

Figure 9: PhaseNet architecture
Taken from [20]

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

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

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

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

III.2 Label generation

PhaseNet takes the arrival times of P and S-wave as the label input and returns the time picking of these both waves as the output, while the network takes and returns the probability distributions of the arrival times and the loss function is based on cross entropy between the true probability distribution and predicted distribution. So we want to find out how to generate probability distributions (segmentation maps) from the labels (III.2.1) and pick arrival times from the output segmentation maps (III.2.2).

III.2.1 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.

Each symmetric "bell curve" is generated by using 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 noises and both phases are generated by the following formula:

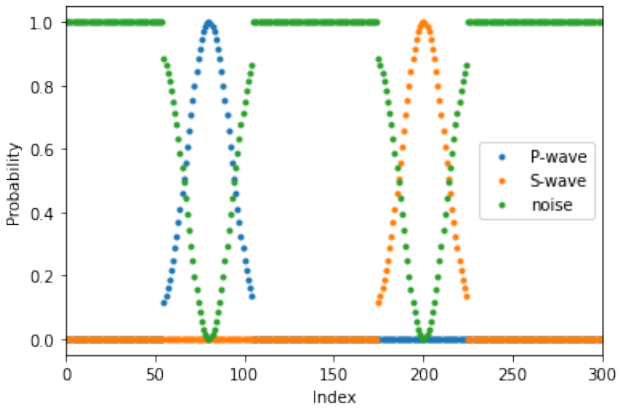

Figure 10: Example of generating segmentation map

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

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

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

where d is the standard deviation, or the Gaussian RMS width, that controls the width of the "bell".

III.2.2 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 11).

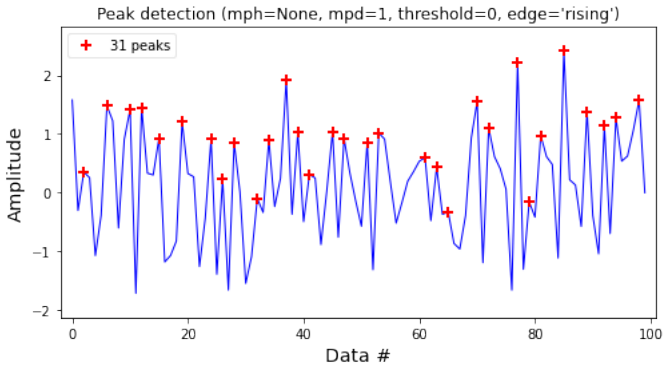

Figure 11: Detecting all peaks in the data

Besides, there are some calibratable parameters 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 criteria for a data as Figure 12, 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 13, 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 14, 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.

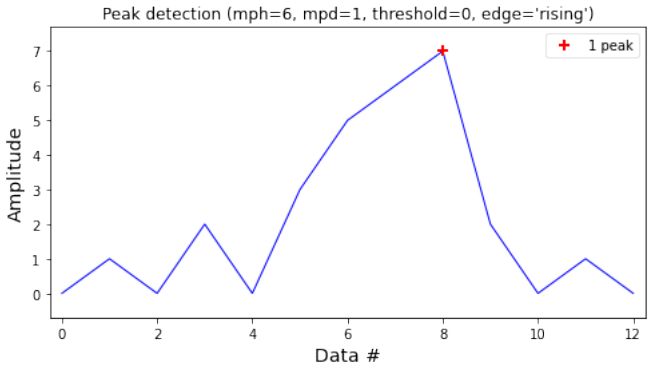

Figure 12: Detecting peaks in the data with mph criteria

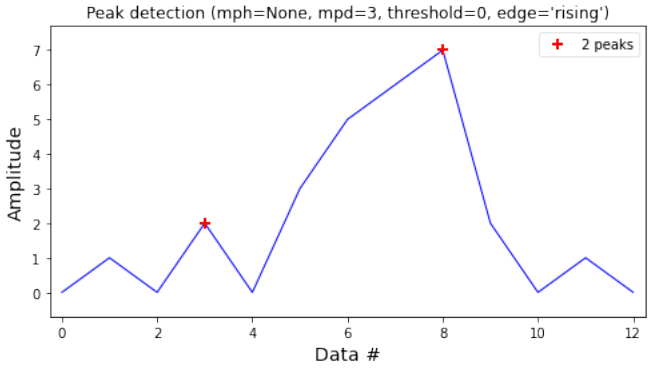

Figure 13: Detecting all peaks in the data with mpd criteria

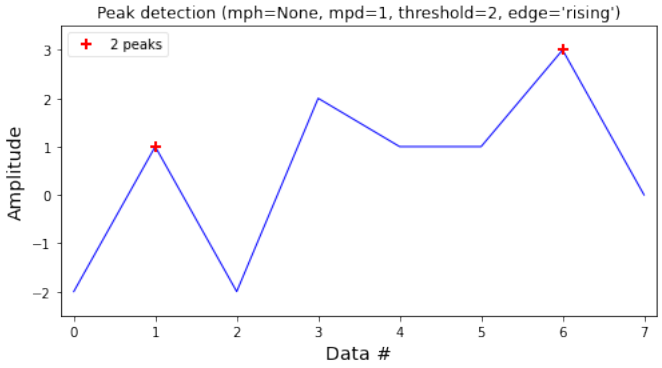

Figure 14: Detecting all peaks in the data with threshold criteria

III.3 Testing and scoring metrics

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

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

Namely, precision describes the precision of predictive values. On the other hand, 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 154 samples (154 seismograms). Among these 154 samples, we find that the P-waves for example appear 142 times in reality (so the number of relevant elements is 142). Then, the model predicts that the P-waves appear 144 times (i.e. there are 144 peak probabilities of P-waves that overtake 0.5), so the positive picks (the total number of predictive values) is 144. Now, we find that among these 144 positive picks, there are 134 predicted values whose residual is less than 0.1s, so the number of true positives is 134 while that of false positives is $144 - 134 = 10$. Thus, the number of false negatives is $142 - 134 = 8$. Hence, precision and recall are respectively:

$$P = \frac{134}{144}, R = \frac{134}{142}.$$

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

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 steps. Those are totally different from our real data (Table 1).

PhaseNet	**Our data**
Passive seismic: sources are earthquakes	Active seismic: sources are hammering, artificial explosion, etc.
Trained with over 0.7 million samples	9216 unlabeled samples
Each sample contains 3 seismograms	Each sample contains 1 seismogram
Each seismogram is recorded with 9000 time steps	Each seismogram is recorded with 4000 time steps

Table 1: Differences between PhaseNet and our data

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

IV.2 Generating a simulated data set

IV.2.1 Generating data based on SPECFEM2D

SPECFEM2D [10] 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 full waveform imaging (FWI) or adjoint tomography. By using this software, we attempt to simulate the seismic waveforms 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.

Some specific model parameters can be varied 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 elements, the number of layers and thickness between them, the phase speed at each layer as well as the nature of each layer (acoustic or elastic), etc..

IV.2.1.1 Simple simulations

We realize simple simulations by defining some flat interface surfaces, typically as Figure 15:

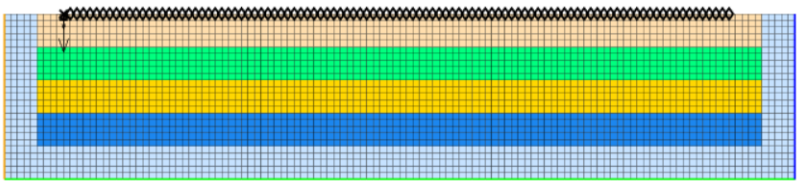

Figure 15: Simple interface model with 5 layers and 4 flat interfaces

The simulations can be confirmed to work well by verifying the spectrogram results, the forward wave field and also the seismogram output (Figure 16).

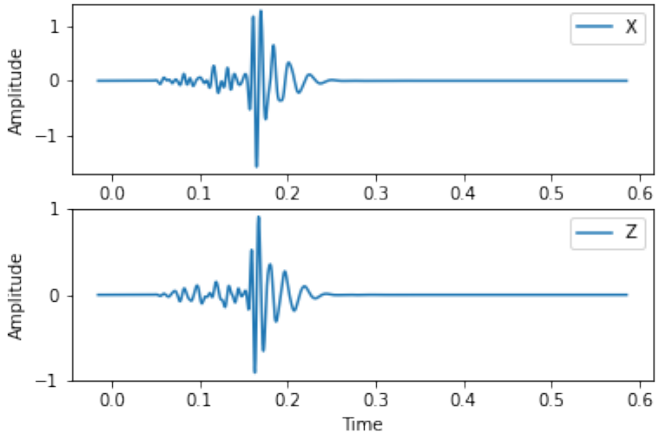

Figure 16: Simple seismic waveform simulated by SPECFEM2D

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.2.3). Then, we can build a complex velocity model with 30 layers and 29 interfaces. Figure 17 shows us the interface model in this case:

Figure 18 conveys a seismogram simulated by adding the topography, that has a waveform more complex than Figure 16 and is closer to reality:

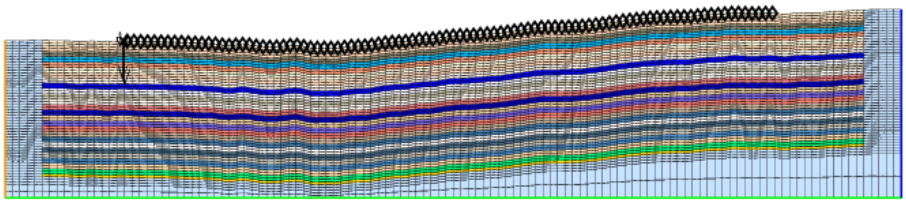

Figure 17: Interface model with 30 layers and 29 interfaces

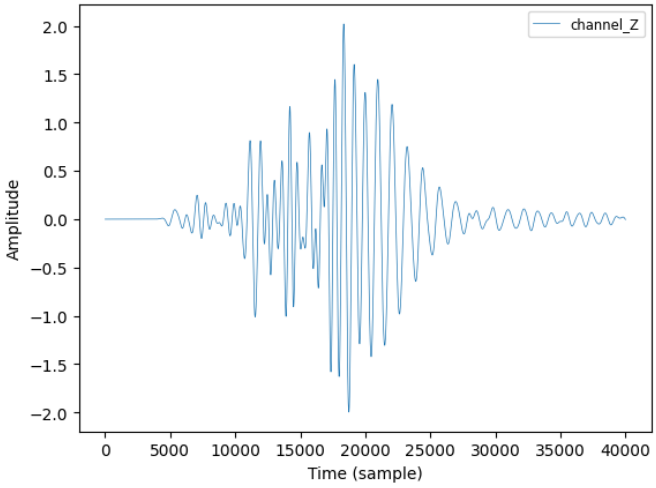

Figure 18: Example of a seismogram simulated by adding a topography

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. Typically, we run SPECFEM2D to simulate the seismograms with 24,001 time steps. Then, for each seismogram, we randomly extract some parts from the original sample. Each extracted seismogram however consists of 10,001 time steps. We can chose the length of the time series we want to extract (here is 10,001), the number of extracted version from the original sample (for example, if we want to randomly selected 10 seismograms from each original seismogram, so the data size will increase 10 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, we can reverse the waveform by multiplying the signal with -1. Figure 19 shows some extraction cases from a original seismogram.

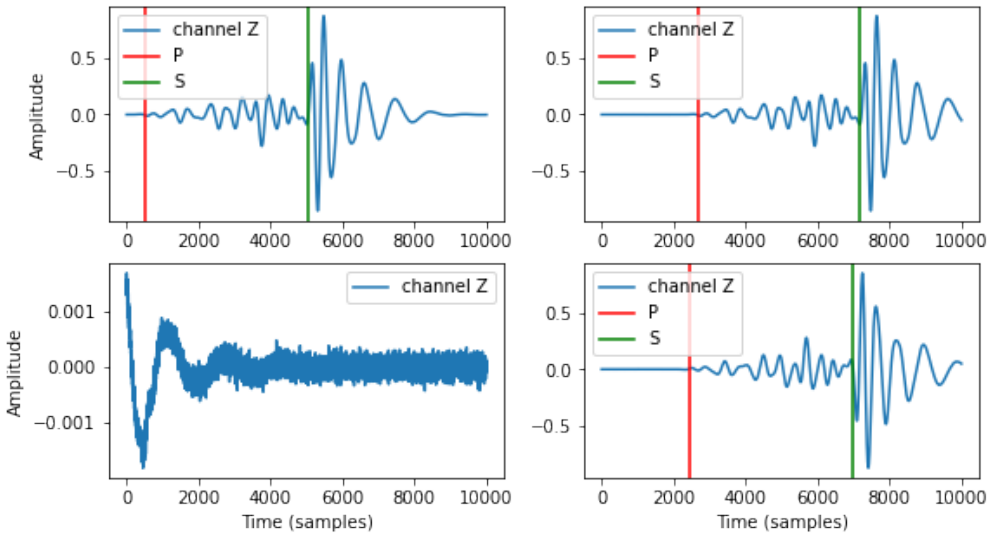

Figure 19: Different seismograms randomly selected from the original one. Both seismograms at the top are selected from the essential part, the one at the bottom left is selected from the part which does not contain the arrival times, and the one at the bottom right is selected from the essential part which is however reversed

IV.2.3 Adding noises

Once we have created a simulated database with data augmentation, we can further alter the original signal samples with noises of varying signal to noise ratios and evaluate the performance of the model under these noisy conditions. We focus on the way using "Additive White Gaussian Noise" (AWGN). This kind of noise is easier to generate and to model for analytical analyzes. In AWGN, the noises are 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:

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

IV.3 Transfer learning

IV.3.1 Why do we use transfer learning?

Nowadays, many deep learning algorithms work well only under a common assumption even if their models achieved a significant precision with their data set. If we want to apply these methods for a newly collected training 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 used the data based on *Northern California Earthquake Data Center Catalog* (NCEDC 2014) that has a lot of different assumptions with our data set 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 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 [2], 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 method 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, improve the learning performance. On the other hand, if the tasks are too dissimilar, the fact that we use transfer learning may hinder performance of the training. The experiments in [17] compared the training performances (Figure 20) 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).

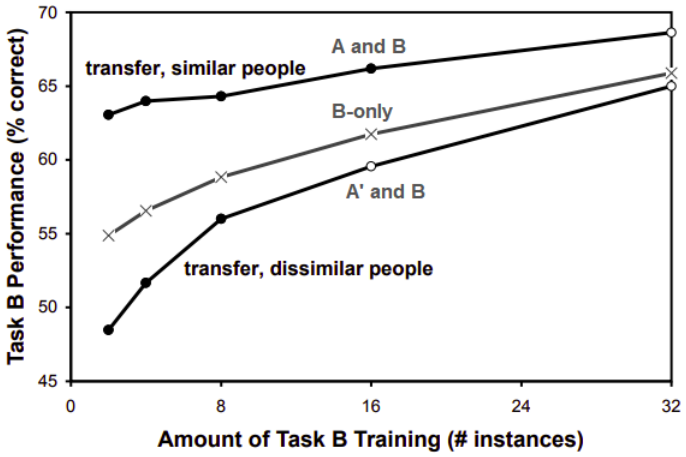

Figure 20: Effects of transfer learning in three cases
Taken from [17]

IV.3.2 Different transfer learning strategies

Primarily, we need to convert all of the PhaseNet code from tensorflow to keras that provides a convenient way in order to implement transfer learning. Once we train the model with a new data set using transfer learning, keras first loads all required model weights from a pre-trained model. Effectively, ... 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 criterion:

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

For example, in the scenario shown in Figure 22, we load weights from the whole pre-trained model into our new model and fine-tune all of these weights. In Figure 23, 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. While with the configuration shown in Figure 24, 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.

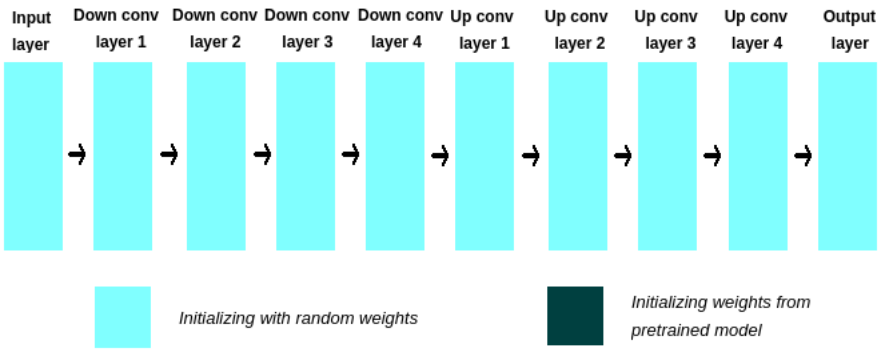

Figure 21: Training all layers from scratch (initializing random weights)

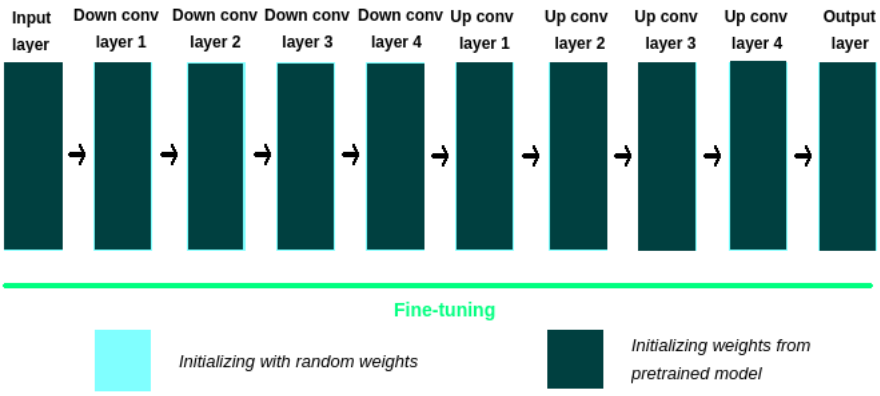

Figure 22: Initializing weights from all layers of pre-trained model and fine-tune all layers

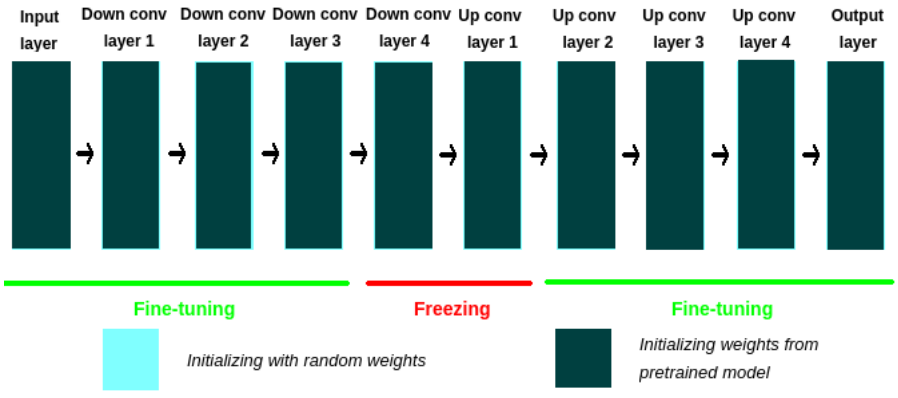

Figure 23: Initializing weights from all layers of pre-trained model and freezing two deepest layers

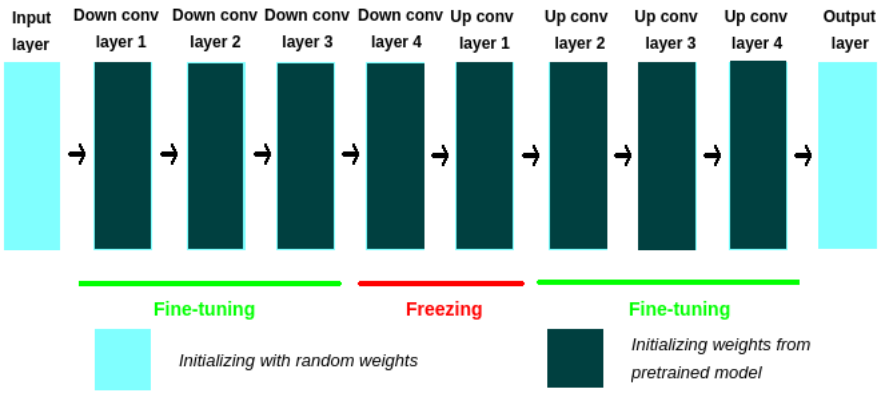

Figure 24: Initializing weights from only encoder-decoder layers of pre-trained model and freezing two deepest layers

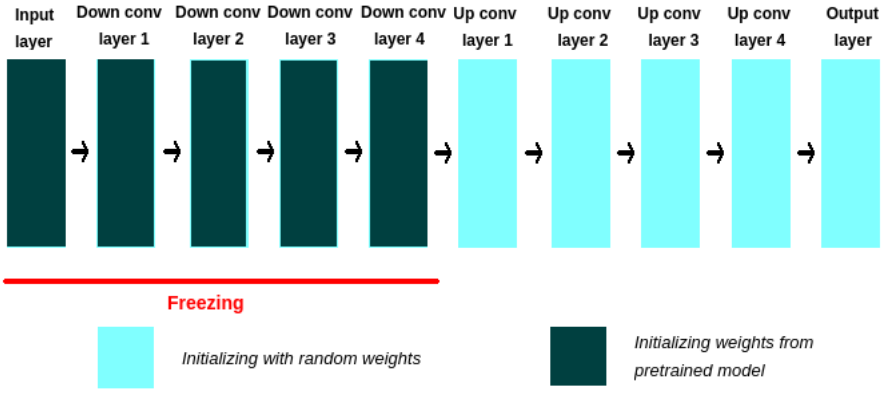

Figure 25: Initializing weights from only input and encoder layers of pre-trained model and freezing these layers

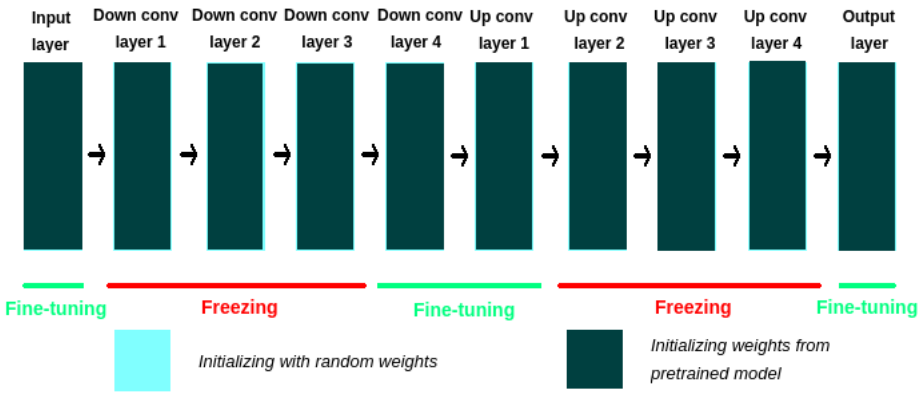

Figure 26: Initializing weights from the entire of pre-trained model and freezing some encoder and decoder layers

IV.4 Results on simulated data

In this section, we will see 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

Based on Figure 27, we clarify that the models implemented transfer learning (cases 2, 3, 4, 5, 6) learn the problem quickly and its loss function are converge faster than the loss in the case of training from scratch (case 1).

Figure 28 shows the training and validation loss in the different cases without realistic topography.....

Loss with topo 29:

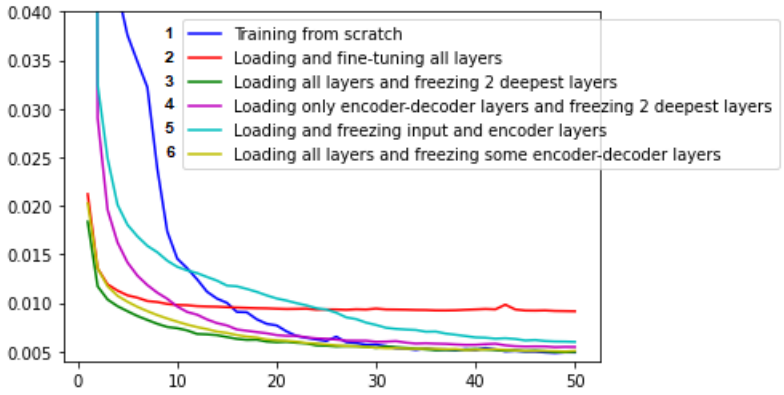

Figure 27: Comparison of training loss in different transfer learning ways without adding realistic topography to the simulated data set

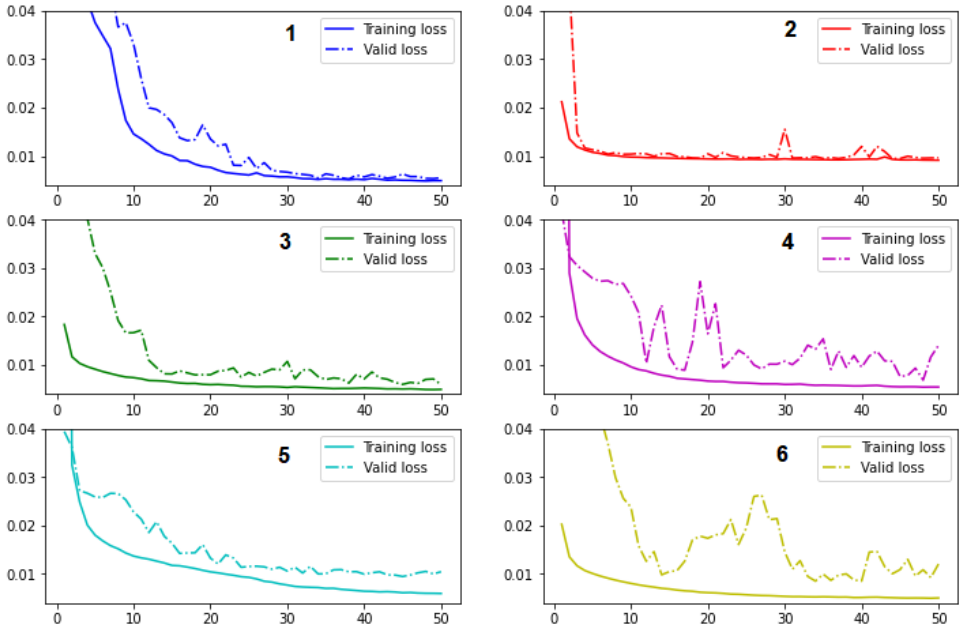

Figure 28: Training and validation loss in different transfer learning ways without adding realistic topography to the simulated data set

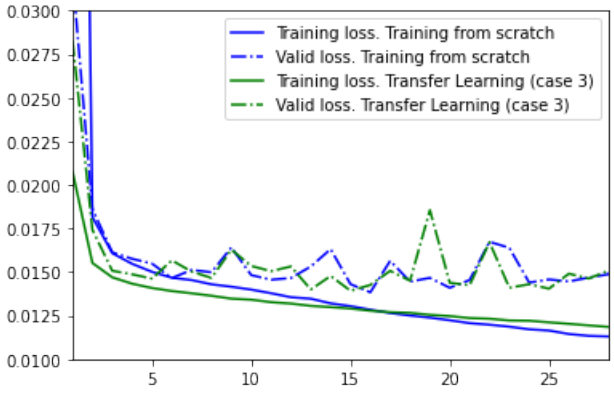

Figure 29: Training and validation loss with realistic topography

IV.4.2 Scoring metrics

As the outputs of the network are the probability distributions, so we need a "detecting peaks" functions that can pick some peak probabilities for each phase (P and S) from the segmentation map. This "detecting peaks" function allows us to pick the arrival time based on some criteria (see in III.2.2). For instance, we can consider "a picked peak" as the peak which is the highest probability and greater than 0.5. We recall that, for each phase, a prediction is:

- a true positive if there is a peak detected and its residual with the true label is less than a tolerance,
- a true negative if the true label does not exist and there is no peak detected (for example, in the case that we have a seismogram with the noises and the model predicts that there is no arrival time detected),
- a false positive if there is a peak detected and:
  - its residual with the true label is greater than a tolerance or,
  - the true label does not exist,
- a false negative if the true label exists but there is no peak detected.

Table 2 shows the scoring metrics in different transfer learning strategies .....

Interpret the score??? Explain why we chose "Loading all layers and freezing 2 deepest layers"???????????????????????????

add score with topo .....

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	0.640	0.557	0.596
	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

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 Semi-supervised learning

V.1.1 Unlabeled data issue 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 requires too much time and effort, it is impossible to label by hand-picking methods, that are 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 use 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 ideal is: using a pretrained 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 30). Once we get and correct all labels for our real data set, we can train a neural network model based on our own data.

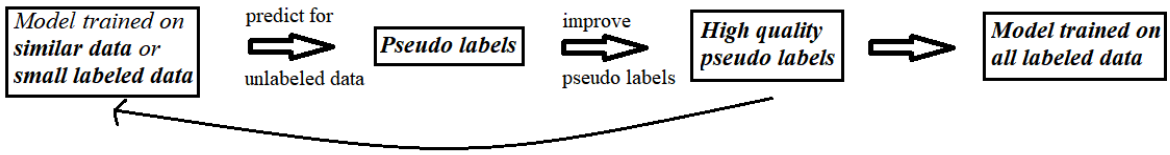

Figure 30: Applying pseudo-labeling to our case

V.1.2 Improving pseudo-labelling

Physically, the greater the distance between the source and the receiver, the greater 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 31), we can further improve the quality of pseudo labels by correcting these predictions using robust regression methods and SVR (Support Vector Regression).

V.1.2.1 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 [9] uses a different loss function rather than the traditional least square by solving:

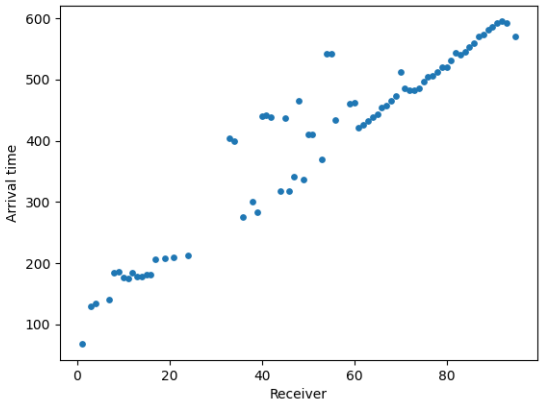

Figure 31: Example of S-phase prediction for unlabeled data at source 1

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

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 32, 33), we solve:

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

V.1.2.2 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 ideal 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 [4]. While many regression algorithms attempt

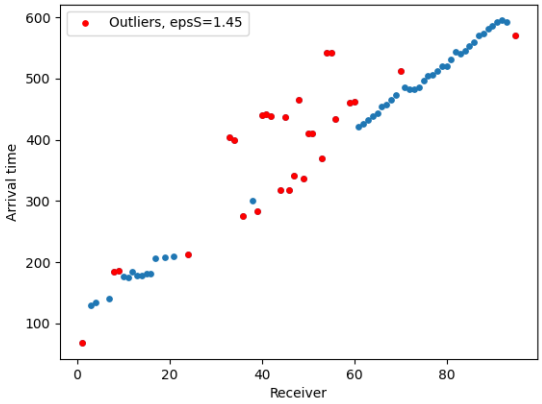

Figure 32: Outlier detection for S-phase using linear Huber regression

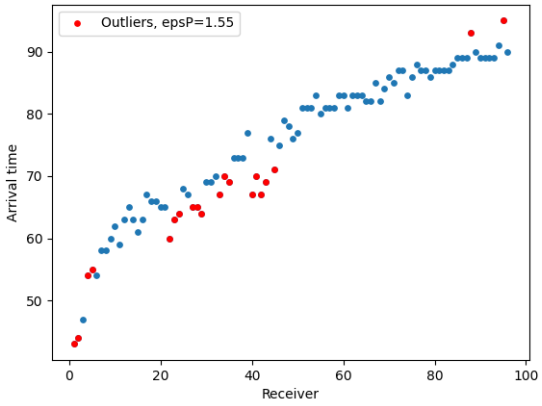

Figure 33: Outlier detection for P-phase using logarithmic Huber regression

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 34 and Figure 35 show the SVR models for P and S-phase based on the data without outliers (detected by using Huber regression):

V.2 Results on real data

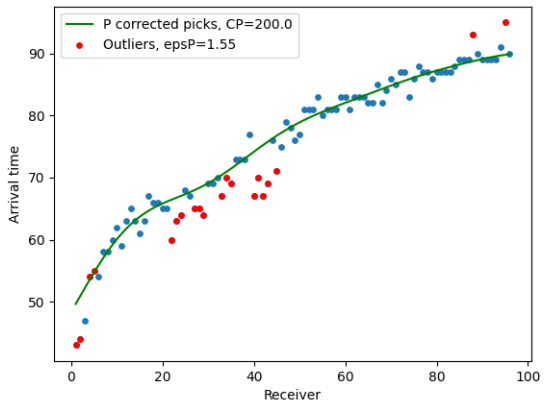

Figure 34: Correcting labels for P-phase using SVR and Huber regression

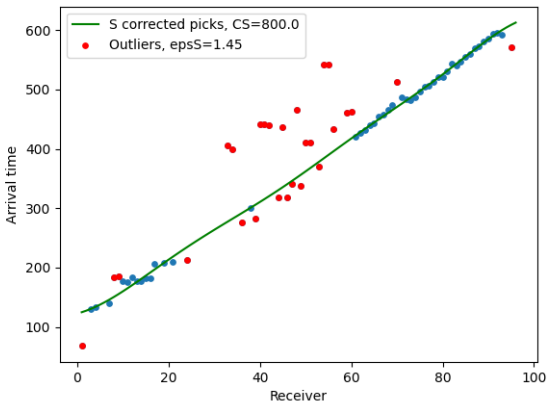

Figure 35: Correcting labels for S-phase using SVR and Huber regression

Part VI
Conclusion and perspectives

References

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

[2] 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. e2020GL088651 2020GL088651.

[3] 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.

[4] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, *Support vector regression machines*, in Proceedings of the 9th International Conference on Neural Information Processing Systems, NIPS'96, Cambridge, MA, USA, 1996, MIT Press, pp. 155–161.

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

[6] 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 (CVPR), June 2016.

[7] 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.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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 (CVPR), June 2015.

[13] H. Noh, S. Hong, and B. Han, *Learning deconvolution network for semantic segmentation*, in Proceedings of the IEEE International Conference on Computer Vision (ICCV), December 2015.

[14] 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.

[15] 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.

[16] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, CoRR, abs/1505.04597 (2015).

[17] M. T. Rosenstein, Z. Marx, L. P. Kaelbling, and T. G. Dietterich, *To transfer or not to transfer*, in In NIPS'05 Workshop, Inductive Transfer: 10 Years Later, 2005.

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

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

[20] 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.