——— 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

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 [11]. 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).

II.2.2 Data collection

II.2.3 Automatic arrival time picking by deep learning

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

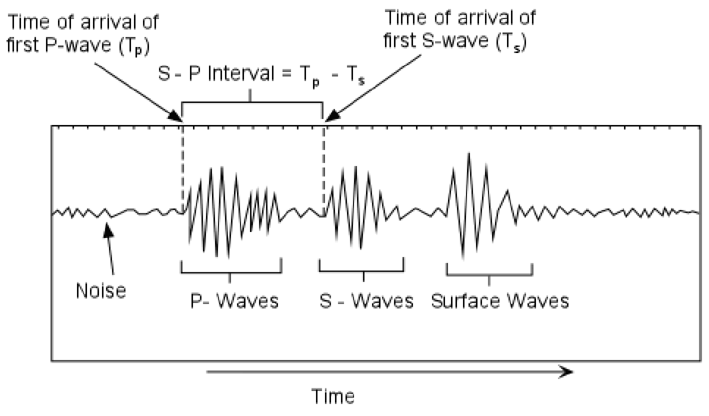

Time of arrival of first P-wave (T_p)
Time of arrival of first S-wave (T_s)
S - P Interval = T_p - T_s
Noise
P- Waves
S - Waves
Surface Waves
Time

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

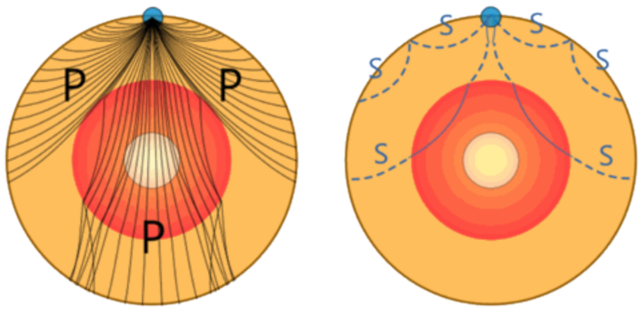


Figure 2: Motion of P-wave and S-wave
Source: `https://www.mathsisfun.com/physics/waves-seismic.html`

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 and also 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 showed 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 [14]. This model is an U-net that is used for biomedical image segmentation and particularly, showed its efficiencies for segmentation of neuronal structures in electron microscopic stacks in [9]. 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. 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) [12]. 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:

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

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:

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

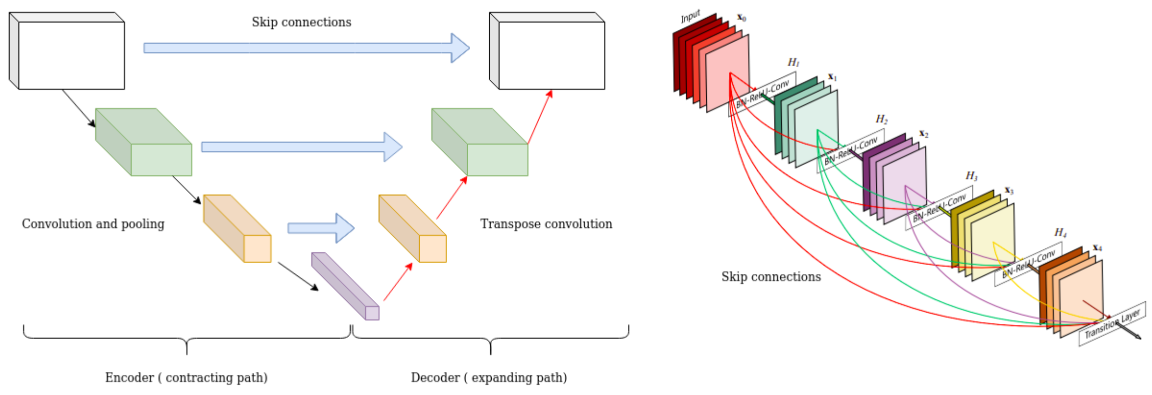
III.1.2 Skip connections

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) [8]. Based on a CNN, we can build a typical FCN by:

- Adding an expanding path (decoder) called “long skip connections” (Figure 3a) 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 3b) that allows us to increase the convergence speed of the loss function and build very deep neural networks that can contain hundreds of layers[4].

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 [2].

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



(a) Long skip connections (b) Short skip connection

Figure 3: 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 4:

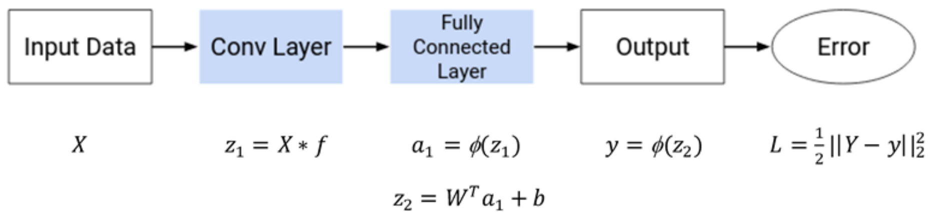


Figure 4: Simple architecture of CNN

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 [5] or addictive skip connection back-propagates through the identity function by using a vector addition (Figure 5a) while DenseNet [6] 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 5b).

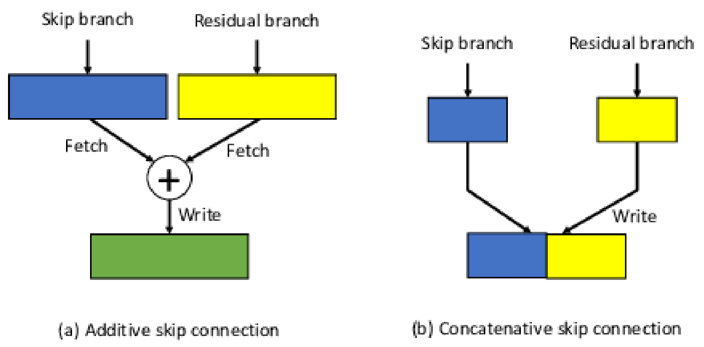


Figure 5: Additive connection and concatenative skip connection
Source: `https://www.researchgate.net/figure/Additive-skip-connections-vs-concatenative-skip-connections-Rectangles-represent-data_fig1_329115979`

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 [1] show 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 6 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:

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

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:

[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.2.2).

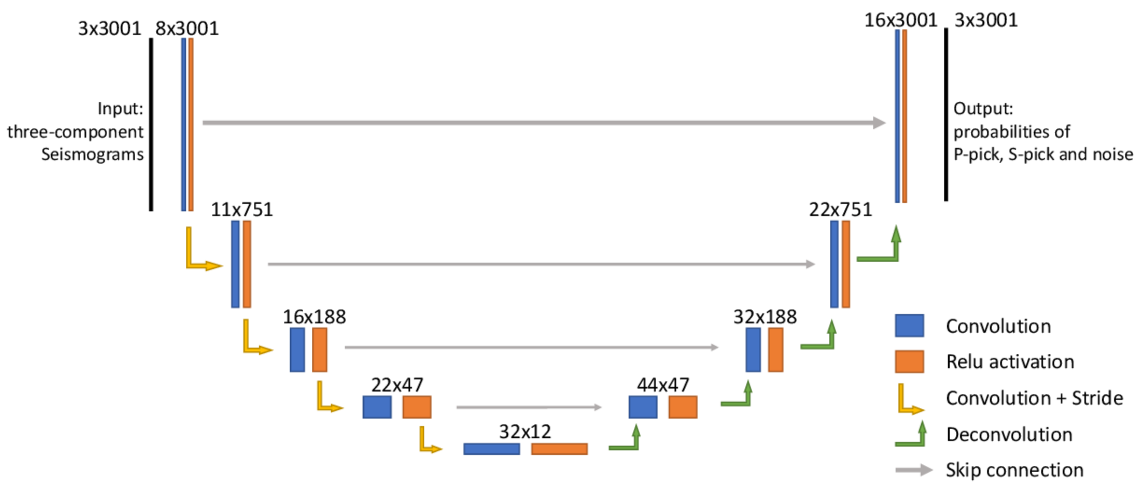


Figure 6: PhaseNet architecture
Taken from [14]

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

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

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

III.2 Data pre-processing and post-processing

III.2.1 Data extraction

The raw data used for training in PhaseNet are the seismograms with 9000 time steps but after pre-processing, the network takes the inputs with only 3000 time steps. It comes to the fact that they want the network to learn the essential information of their data and also learn to detect noises. Concretely, each seismogram entering the network is an extraction of 3000 points from the original one with a chance of 0.95 (this threshold can be calibrated of course). In this case, the labels (the arrival times of P and S-wave) have to be included in these 3000 time steps. On the contrary, with a chance of 0.05, the random extraction part contains only the noises (a part in which the arrival times are not observed) and then its labels are considered as an empty list.

For example, we consider an orginal raw sample with 9000 time steps, its label are ?? and ??? for P and S wave respectively. So, ...

add fig................

III.2.2 Label generation

As we know, PhaseNet takes the arrival times of P and S-wave as the label input for the training and returns the time picking of these both waves as the output for the prediction, while the network takes and returns the probability distributions of the arrival times (because the loss function is based on cross entropy between the true probability distribution and predicted distribution). So we want to find out how PhaseNet converts from the point times to the probability distributions before the input of the network (in III.2.2.1) and from the probability distributions to the time picking after the output of the network (in III.2.2.2).

III.2.2.1 Generating segmentation map from arrival time

Seeing that the network takes the probability distribution of P, S-waves and noise as labels, so we need to generate them from the arrival times. As we saw previously in III.2.1, the extracted data is either the noise part (with a chance of 5%), or the essential part (with a chance of 95%). In the first case, we realize a label as the zero probability distributions for P and S-wave, and the one probability distribution for the noise (Figure 7).

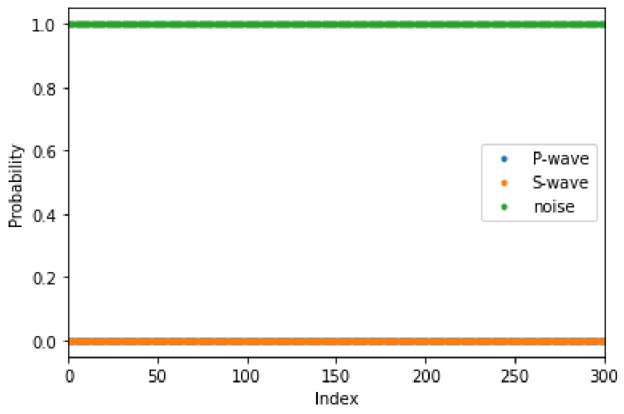


Figure 7: Output probability distributions when the extracted data is the noise part

In the second case, we realize the label as the symmetrical distribution. The mean of this distribution is the arrival time of P-wave (or S-wave) and we compute the probability for a calibratable number of points around the mean (Figure 8).

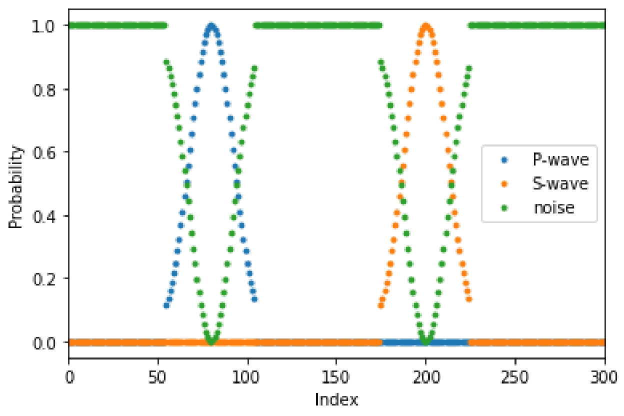


Figure 8: Output probability distributions realized with 50 points around the mean when the extracted data is the essential part

III.2.2.2 Detecting arrival time from output segmentation map

Now, we want to pick up the arrival times from the probability distributions 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 9).

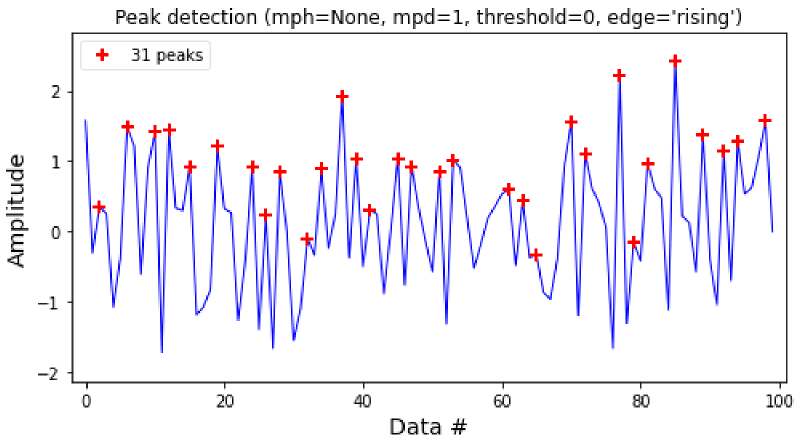


Figure 9: Detecting all peaks in the data

Beside, 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 10, so the function only detect 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 11, 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 12, 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.

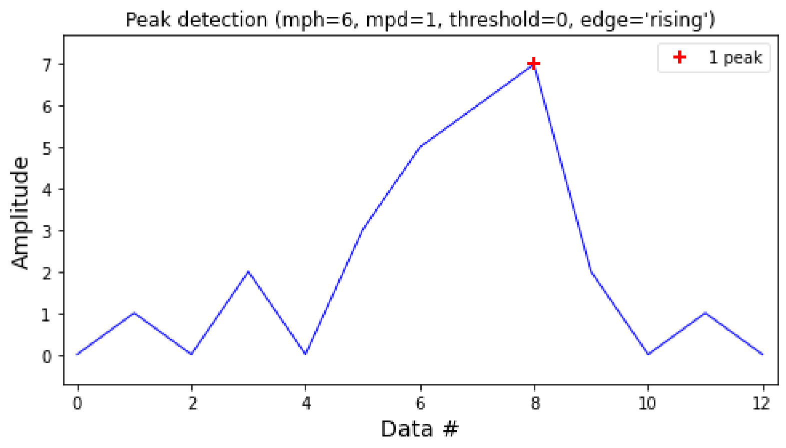


Figure 10: Detecting peaks in the data with mph criteria

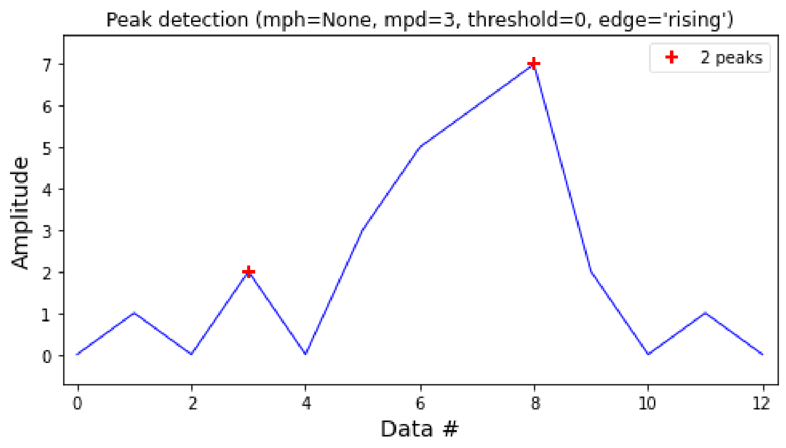


Figure 11: Detecting all peaks in the data with mpd criteria

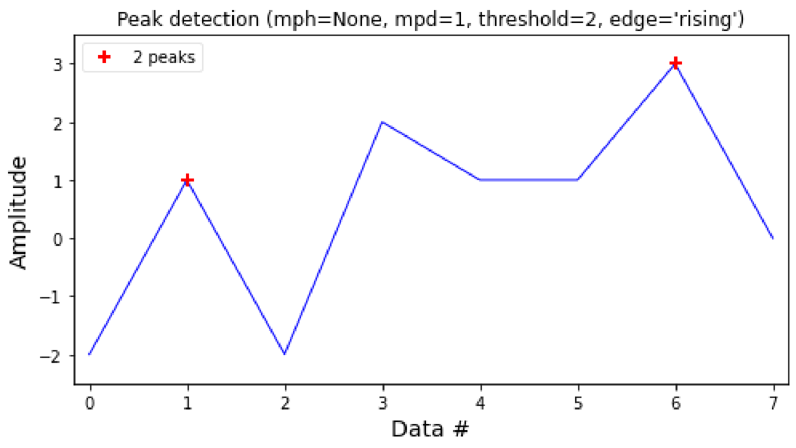


Figure 12: Detecting all peaks in the data with threshold criteria

III.3 Testing and scoring metrics

For evaluating the fitting of model with the data set, the method chosen is the evaluation metrics: precision, recall and F1 score that measure the performance of the model on the test data set. We assume that peaks probabilities above 0.5 are called as positive picks. Moreover, those which have large arrival-time residuals (greater than 0.1s) are counted as false positives (false predictive values) and those which have residuals are counted as true positives (true predictive values). Arrival-times which 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:

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

Namely, precision describes the precision of predictive values. On the other hand, recall refers to the percentage of total relevant results correctly classified:

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

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:

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

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:

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

Actually, F1 scores evaluating the performance of PhaseNet model on their own test data (with 78,592 samples) achieved 0.896 for P-waves and 0.801 for S-waves, that shows a glorious improvement comparing with 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 available data set that contains more than 0.7 million seismograms from natural earthquake. 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). Each sample of the data set consists of three seismograms corresponding to three-component signals (Figure 13).

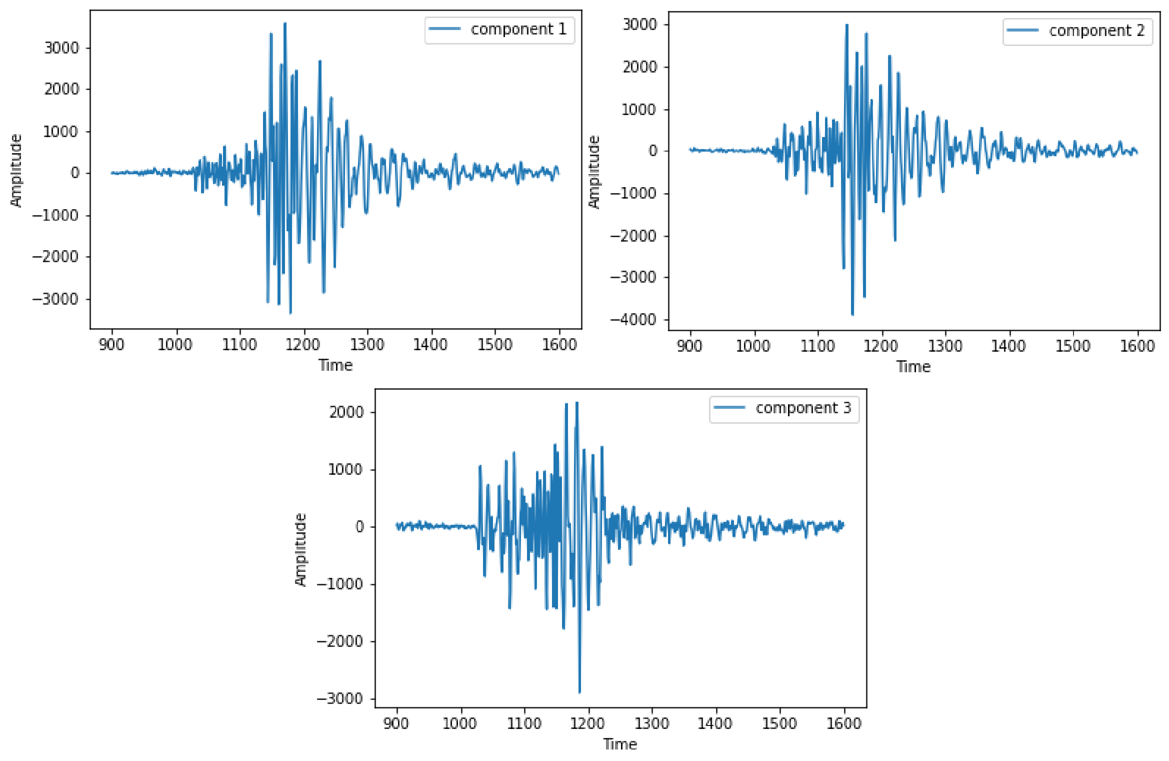


Figure 13: Three-component signals of a sample

Then, the outputs are the probability distributions of noise, P-wave and S-wave (Figure 14).

IV.2 Generating a simulated database

IV.2.1 SPECFEM2D: A numerical simulation tool for forward and adjoint seismic wave propagation

SPECFEM2D [7] 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

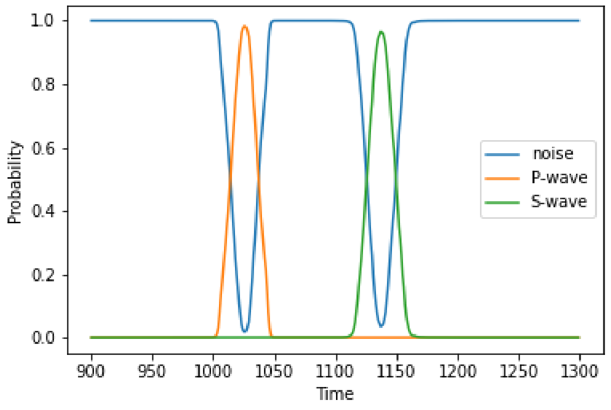


Figure 14: Probability distributions of noise, P-wave and S-wave obtained by prediction

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 create a training database for our model.

Some specific model parameters can be varied in order to achieve a diverse data set for our training 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.. The simulation can be confirmed to work well by verifying the spectrogram results, the forward wave field and also the seismogram output (Figure 15).

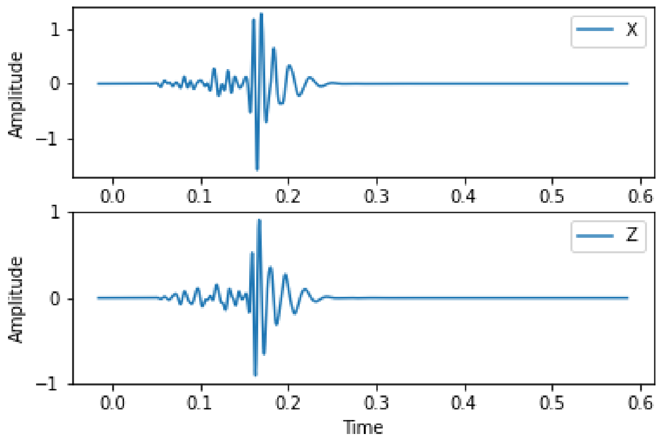


Figure 15: A seismic waveform example simulated by SPECFEM2D

IV.2.2 Data augmentation

In order to increase the size of our database, we use SPECFEM2D to simulate the seismograms with the series of 24,001 time points and then randomly extract the parts of 10,001 time points for each original sample. 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 database size will increase 10 times) and the rate coefficient $\alpha \in [0, 1]$. 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 16 shows some extraction cases from a original seismogram.

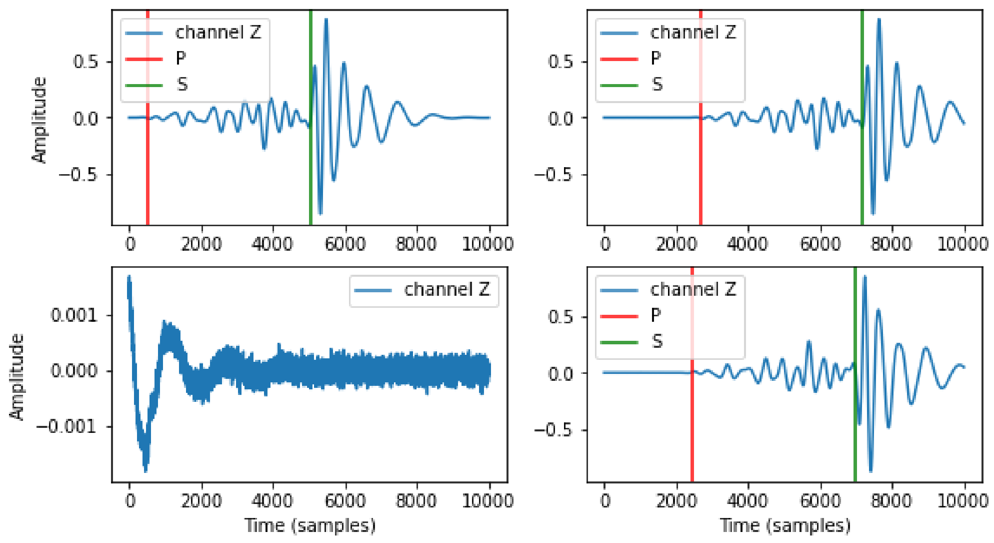


Figure 16: Different seismograms randomly selected from the original one. Both seismograms at the top are selected from the essential part, the one at the bottom left are selected from the part which does not contain the arrival times, and the one at the bottom right is selected from the essential part 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. 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. In the experiments shown in [10], they compared performances of the training in three cases (Figure 17):

1. No transfer training (B-only): they train a prediction model only with the data set B;

2. Similar transfer training: they train the model on data B using the transfer learning with the pre-trained model on data A (similar with B).

3. Dissimilar transfer training: the pre-trained model on the dissimilar data A is used to transfer the prediction model on data B.

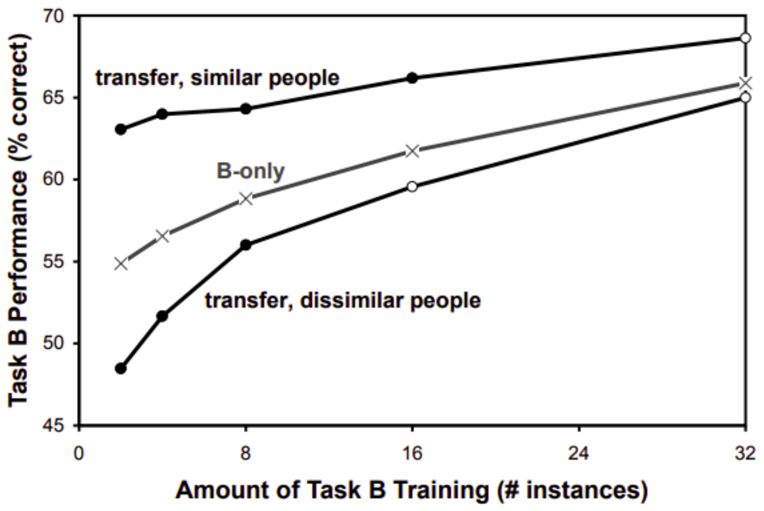


Figure 17: Effects of transfer learning in three cases
Taken from [10]

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. 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.

Once we have created a data set with the simulation tool, we consider a training-validation set with the size of 3890 and a test set with the size of 480. We consider this time the neural network taking only one channel, that means we apply only one filter for a single channel in the input layer instead of three for three channels. 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 two channels that we do not use.

Now we want to compare the performance of the network on the data test by learning features from the training data in different approaches. We will compare the neural network trained from scratch according to our data (Figure 18) with the network using transfer learning in the different ways (Figure 19, 20, 21).

In the scenario of Figure 19, we load weights from the whole pre-trained model into our new model and fine-tune all of these weights.

In Figure 20, 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.

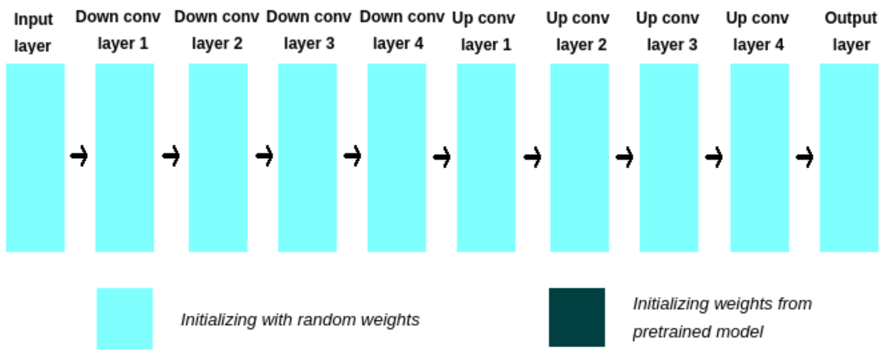


Figure 18: Training all layers from scratch (initializing random weights)

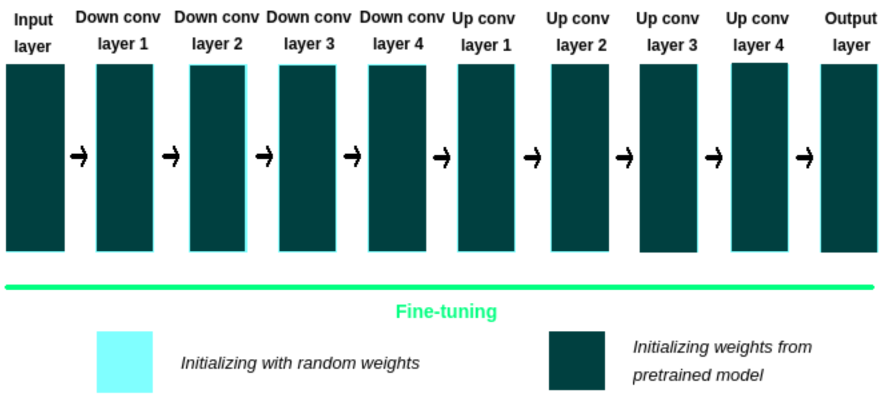


Figure 19: Initializing weights from all layers of pre-trained model and fine-tune all layers

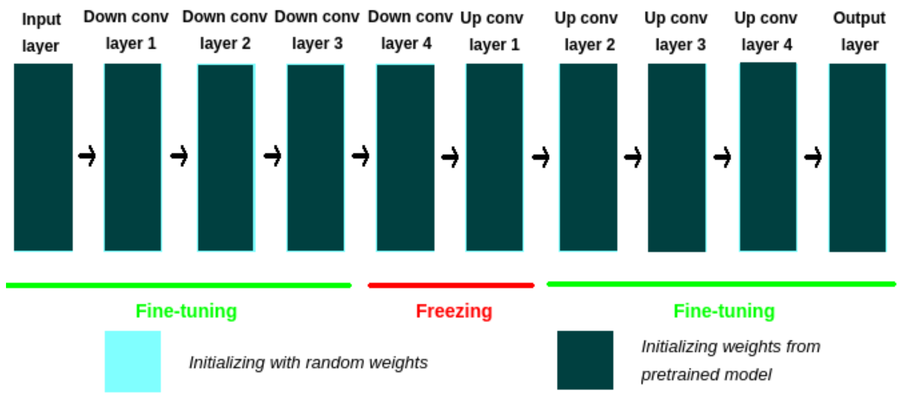


Figure 20: Initializing weights from all layers of pre-trained model and freezing two deepest layers

In the following configuration shown in Figure 21, 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.

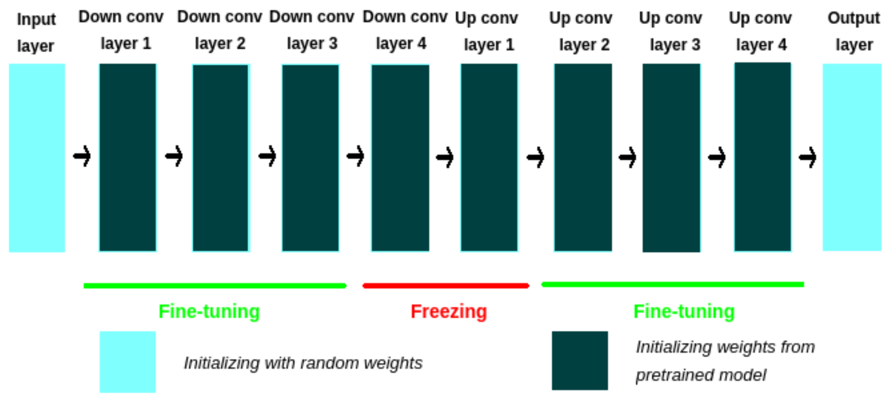


Figure 21: Initializing weights from only encoder-decoder layers of pre-trained model and freezing two deepest layers

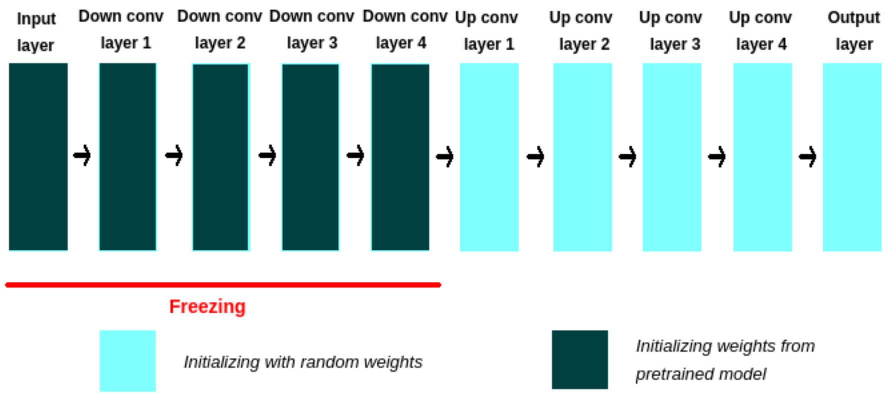


Figure 22: Initializing weights from only input and encoder layers of pre-trained model and freezing these layers

IV.3.3 Results

IV.3.3.1 Training and validation loss

Figure 24 and 25 show the training and validation loss in the different cases.

interpret loss ....

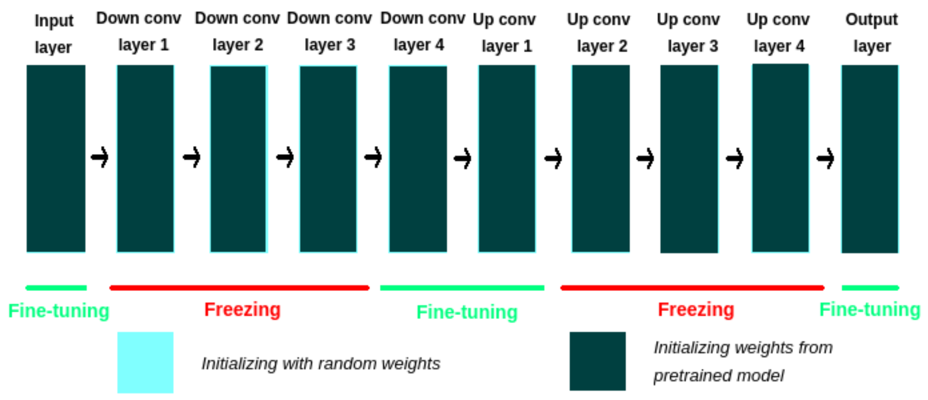


Figure 23: Initializing weights from the entire of pre-trained model and freezing some encoder and decoder layers

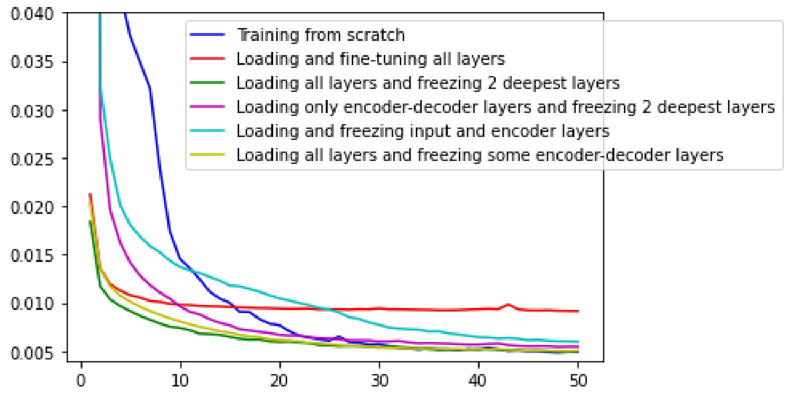


Figure 24: Comparison of training loss in different cases

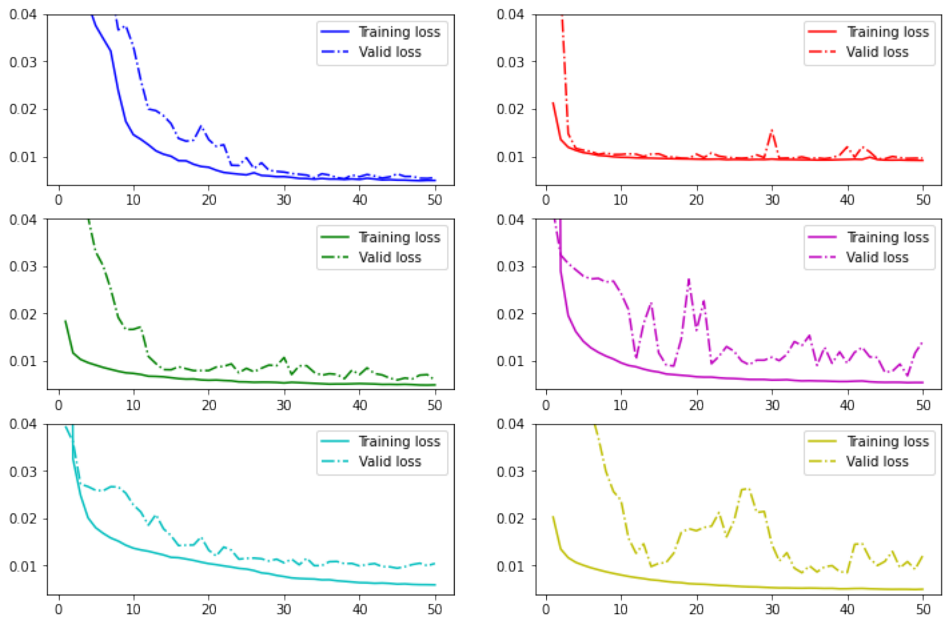


Figure 25: Training and validation loss in different cases

IV.3.3.2 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.2.2.2). 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 1 shows the scoring metrics in the 4 different cases.

		Precision	Recall	F1-score
Training from scratch	P-phase	0.694	0.612	**0.65**
	S-phase	0.965	0.98	0.973
Loading and fine-tuning all layers	P-phase	0.595	0.535	0.564
	S-phase	0.983	0.987	0.985
Loading all layers and freezing 2 deepest layers	P-phase	0.636	0.548	0.589
	S-phase	0.983	1	**0.991**
Loading only down-upsampling layers and freezing 2 deepest layers	P-phase	0.562	0.518	0.539
	S-phase	0.967	0.963	0.965
Loading and freezing input and encoder layers	P-phase	0.54	0.414	0.469
	S-phase	0.98	0.985	0.982
Loading all layers and freezing some encoder-decoder layers	P-phase	0.64	0.557	0.596
	S-phase	0.998	0.98	0.989

Table 1: Scoring metrics for different cases

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

IV.4 Database improvement by adding realistic topography

Part V

Application to the real data set

V.1 Application of semi-supervised learning to unlabeled data

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

V.1.2 Pseudo-labelling

V.1.2.1 Anomaly detection using robust regression methods

V.1.2.2 Correcting pseudo labels using Support Vector Machine

V.2 Results

Part VI

Conclusion and perspectives

[3]
[13]

References

[1] N. Adaloglou, *Deep learning in medical imaging: 3d medical image segmentation with pytorch*, 4 2020. [Online; posted 02-Apr-2020].

[2] ——, *Intuitive explanation of skip connections in deep learning*, 3 2020. [Online; posted 23-Mar-2020].

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] C. Versloot, *Understanding transposed convolutions*, 9 2019. [Online; posted 29-Sep-2019].

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

[14] 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.