PSL UNIVERSITÉ PARIS

Fraunhofer ITWM

ENS DÉPARTEMENT DE GÉOSCIENCES

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à l'École Normale Supérieure

Applications de l'Apprentissage Profond pour le Traitement et l'Interprétation Sismique

Deep Learning for Seismic Data Processing and Interpretation

Soutenue par

Valentin TSCHANNEN

Le 11 juin 2020

École doctorale n°560

École Doctorale STEP'UP

Spécialité

Sciences de la Terre et de l'Environnement

Composition du jury :

Hervé CHAURIS Pr, Mines ParisTech	*Président*
Alessandra RIBODETTI Dr, Université de Nice Sophia-Antipolis	*Rapportrice*
Nicolas THOME Pr, Conservatoire National des Arts et Métiers	*Rapporteur*
Alain RABAUTE Pr, Sorbonne Université	*Examinateur*
Éléonore STUTZMANN Pr, Institut de Physique du Globe de Paris	*Examinatrice*
Matthias DELESCLUSE Dr, École Normale Supérieure	*Directeur*
Norman ETTRICH Dr, Fraunhofer Society	*Invité*
Janis KEUPER Pr, Offenburg University & Fraunhofer Society	*Invité*

ENS ÉCOLE NORMALE SUPÉRIEURE

PSL RESEARCH UNIVERSITY PARIS

CONTENTS

PRÉFACE

Cette thèse est écrite en vue de l'obtention d'un doctorat de l'École Normale Supérieure (ENS), membre de l' Université Paris Sciences et Lettres, sous l'affiliation de l'école doctorale STEP'UP (n°560) en spécialité Sciences de la Terre et de l'Environnement. Elle s'est déroulée entre le Laboratoire de Géologie de l'ENS Paris et le département de Calcul de Haute Performance de l'Institut Fraunhofer pour la Recherche en Mathématiques Industrielles (Fraunhofer ITWM) en Allemagne. Le financement provient d'une bourse de thèse de la Fraunhofer-Gesellschaft qui s'est étendue d'octobre 2016 à mars 2020. L'encadrement du projet de recherche a été assuré par Dr. Matthias Delescluse (ENS), Dr. Norman Ettrich (Fraunhofer ITWM) et Dr. Janis Keuper (Fraunhofer ITWM). La thématique principale de la thèse concerne les applications de l'apprentissage profond, une branche de l'intelligence artificielle, pour le traitement et l'interprétation des données sismiques.

RÉSUMÉ

L'exploration sismique est une méthode de choix pour acquérir une connaissance détaillée de la géologie des bassins sédimentaires et de la croûte terrestre. Cependant, transformer le champ d'onde enregistré en une image précise du sous-sol est une tâche longue et difficile. De nombreux algorithmes sont employés pour traiter le signal, atténuer le bruit et aider à interpréter les données. Ces algorithmes sont conçus par des experts au moyen d'une série d'instructions minutieusement programmées. Ils doivent être soigneusement paramétrés chaque fois qu'ils sont employés et les utilisateurs doivent avoir une bonne compréhension de leurs fonctionnements et de leurs limites. De plus, certaines tâches doivent être effectuées manuellement par des géoscientifiques lorsque qu'elles ne peuvent pas être automatisées correctement par des algorithmes. Ces dernières années, une classe différente d'algorithmes a pris de l'importance. Au lieu de nécessiter des instructions explicites pour résoudre un problème, ils fonctionnent comme des modèles adaptatifs qui peuvent apprendre des données en s'améliorant avec l'expérience. Cette approche est appelée Apprentissage Machine, un sous-domaine de l'Intelligence Artificielle, et parmi ses nombreuses branches, l'Apprentissage Profond apporte actuellement les meilleures performances. L'Apprentissage Profond a ouvert la voie à de nouvelles applications de pointe dans de nombreuses disciplines scientifiques et techniques. Il permet notamment d'automatiser certains processus qui n'étaient jusque-là réalisables que par les humains. Cependant, il peut être difficile de remplir les conditions nécessaires pour utiliser efficacement ces modèles d'apprentissage.

Dans ce travail, nous commençons par identifier les principaux obstacles à l'exploitation du potentiel de l'apprentissage profond. Un premier défi naît de la dépendance de la procédure d'entraînement à l'intervention humaine. Les algorithmes ont besoin d'exemples pour apprendre et pour certaines applications, la préparation de ces exemples peut nécessiter un travail manuel considérable de la part d'experts. Un autre défi réside dans les incertitudes

inhérentes associées aux données sismiques. Le manque de résolution et la présence de bruits conduisent à une non-unicité de l'interprétation. Cela pose des problèmes car des erreurs et des biais humains peuvent être communiqués à l'algorithme. De plus, le manque d'interprétabilité et de garanties théoriques sur l'apprentissage profond sont également des problèmes. Il est difficile de comprendre le processus conduisant la machine à donner sa réponse et le comportement de l'algorithme lorsqu'il est confronté à de nouvelles données n'est pas prédictible.

Suite à ces observations, nous proposons une série de méthodologies qui visent à atténuer ces problèmes et à exploiter les capacités de l'apprentissage machine. Nous démontrons la validité et la faisabilité de nos méthodes sur un ensemble de problèmes d'interprétation et de traitement sismiques. (1) Nous montrons comment, avec l'apprentissage par transfert de la compréhension et la distillation des connaissances, nous pouvons exploiter des éléments déjà connus, soit en utilisant la physique du problème, soit en tirant parti des algorithmes existants, pour entraîner un réseau de neurones sans avoir besoin d'un travail manuel conséquent. (2) Nous présentons des démarches basées sur un apprentissage semi-supervisé où le géologue peut progressivement guider la machine vers une réponse qui lui convienne. Cette approche permet de réduire le besoin d'intervention manuelle et est également utile pour gérer les incertitudes et le caractère non unique de l'interprétation. (3) Nous explorons certaines des possibilités offertes par l'apprentissage non supervisé afin d'accomplir des tâches sans avoir besoin de directives explicites venant des experts, et démontrons les premiers résultats sur des applications pratiques.

Abstract

Seismic reflection is a method of choice to gain a detailed knowledge of the geology of the Earth's crust. However, transforming the recorded wavefield into an accurate image of the subsurface is a long and challenging task. Many algorithms are involved to process the signal, attenuate the noise and help interpreting the data. Those algorithms are designed by experts and are implemented as a sequence of hand-crafted operations that are thought to help solving the imaging problem. They require to be carefully parametrized every time they are employed and practitioners must have a good understanding of their applications range and limits. Additionally, several interpretation tasks have to be performed manually by geoscientists when algorithms fail to deliver good results. In recent years, a different class of algorithms have rose to prominence. Instead of necessitating explicit instructions to solve a problem, they are implemented as adaptive models that can learn from the data by self-improving when given feedback. This approach is named Machine Learning, a subfield of Artificial Intelligence, and among its many branches, Deep Learning currently brings the best performance for complex tasks. Deep learning has set the new state-of-the-art in many applications across numerous scientific and engineering disciplines. It also allows to automate certain processes that were until then only feasible by humans. However, fulfilling the conditions necessary to effectively use these learning models may be difficult.

In this work, we start by identifying the main impediments to harvest the potential of deep learning for geophysical applications. A first challenge comes from the dependency of the training procedure to human intervention. Algorithms need examples to learn and for some applications preparing those examples may require a considerable amount of manual work from experts. Another challenge lies in the inherent uncertainties associated with seismic data. The lack of resolution and the presence of noise lead to uncertainties and non-uniqueness in the interpretation. This raises issues when giving feedbacks to the machine about its perfor-

mance since errors and human biases may be communicated to the algorithm. Additionally, the lack of interpretability and theoretical guaranties of deep learning is also a problem. It is difficult to understand the process leading the machine to give its answer and the expected behaviour of the algorithm when encountering new data can not be predicted with certainty.

Following those observations, we propose a series of methodologies that aim to mitigate these issues and exploit the capabilities of deep learning. We demonstrate the validity and practicability of our methods on a set of challenging interpretation and processing problems. (1) We show how with Transfer Learning and Knowledge Distillation we can benefit from prior knowledge, either by exploiting the physics of the problem or by leveraging existing algorithms, to train a neural network without the need of an extensive manual labour. (2) We present workflows based on Semi-Supervised Learning where the interpreter can progressively guide the machine toward an accepted answer. This approach enables to reduce the need for manual intervention and is also valuable to handle uncertainties and non-uniqueness in the interpretation. (3) We explore some of the possibilities offered by Unsupervised Learning in order to accomplish tasks without the need of an explicit guidance by experts and demonstrate initial results on practical applications.

CHAPTER 1 INTRODUCTION

Solid earth science is the sub-discipline of Geosciences interested in the study of our planet's interior. One of the prominent method to probe the sub-surface is the analysis of seismic waves that can yield detailed images valuable for the study of geological structures as well as giving information about the type of rocks in presence. The remote nature of the measurements combined with the need of ever higher resolution information lead to large datasets that need to be processed in order to improve the signal-to-noise ratio and need to be interpreted despite uncertainties. Machine learning deals with the creation of algorithms that perform specific tasks without having explicitly programmed instructions. They rely instead on the statistics of the data and are becoming increasingly useful to help dealing with very large datasets. In the following we give an overview of seismic exploration techniques, introduce some relevant machine learning disciplines and discuss their application to the processing and interpretation of seismic data.

1.1 Reflection Seismology

While seismology is the study of the Earth using waves produced by natural sources such as earthquakes, reflection seismology uses controlled industrial sources to generate the mechanical energy. In standard surveys, seismic waves possess a penetration depth of several kilometres below the surface and yield a resolution of few tens of meters. They can be acquired both on land and off shore. This make them valuable to the study of processes in the crust and sedimentary basins as well as in various industrial applications such as oil and gas exploration and production, geothermal energy exploitation, carbon dioxide trapping or construction

engineering.

1.1.1 Seismic Data

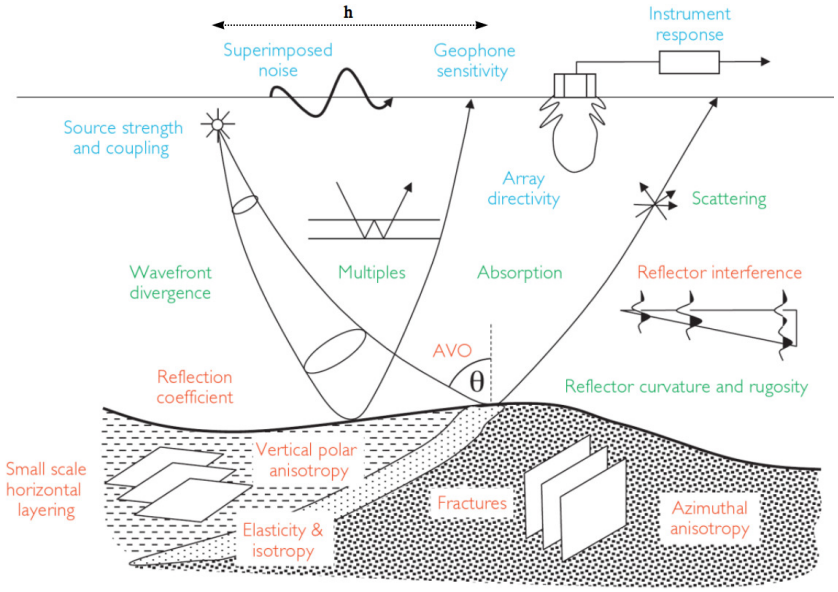


Figure 1.1 – Summary of the factors affecting the seismic amplitudes. Blue terms list acquisition related effects, terms in green highlight general propagation effects while orange terms list the properties affecting the reflection at layer boundaries. (From Simm et al. (2014)).

Seismic Acquisition and Migration

The Earth's interior can approximatively be described as a sequence of layers each consisting in a certain type of rock with given physical properties. In a typical marine acquisition, sources and receivers are placed just beneath the sea surface and towed by a vessel. At a regular pace, sources are triggered and they release acoustic waves that propagate down the water and through the subsurface. At each interface between layers, an impedance contrast reflects part of the seismic energy towards the surface that is recorded by the sensors. As it propagates, the seismic wavefield is modified according to the geometry and properties of the medium. Using equations that describe the propagation of mechanical waves, it is possible to relate the informations contained in the recorded wavefield to the hidden geology of the subsurface. Figure 1.1 summarizes the different factors influencing the seismic response.

According to the convolutional model, seismic traces can be modelled as the convolution of a

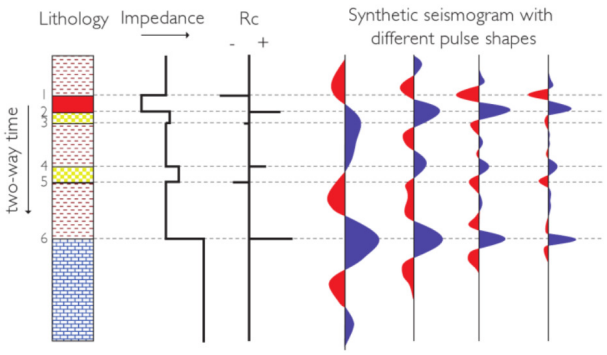


Figure 1.2 – Illustration of the effect of the wavelet on the seismic signal. Synthetic traces generated with the convolution model. The impedance is the product of the volumetric mass density of the medium and the speed of waves. (Taken from Simm et al. (2014)).

seismic pulse with a reflection coefficient series. The reflection coefficient series is related to changes in the physical properties at layers interfaces following Zoeppritz equations. Figure 1.2 illustrates the meaning of the colors in a seismic image. Following the SEG convention (Society of Exploration Geophysicists), a positive reflection coefficient is associated with an increase in the impedance and is associated with a wavelet pick (here in blue). Conversely, a decrease of impedance is associated with a seismic trough (here in red).

It is not straightforward to understand raw seismic data. A vast amount of processing sequences need to be applied in order to transform the wavefield into an image that can be interpreted. We give a brief overview of seismic processing at the end of the section, and only mention here the migration step. After acquisition, a seismic event is located with respect to the position of the source and the receiver as well as the time the wave took to travel from one to the other. The purpose of migration is to re-locate the event to the position it occurred in the subsurface. There exist many types of migration algorithms that differ in the assumptions they make about wave propagation as well as in the domain they operate in, but they all require the knowledge of an accurate estimate of the subsurface velocity (Robein et al., 2010). An overview of how velocity models are obtained is for example given in Jones et al. (2010). Depending on the method, the vertical dimension of the subsurface can either be time or depth. The former corresponds to the two way travel-time of the wave between the source and the receiver and is related to the actual depth via a velocity model. An example of time migration can be seen in Figure 1.3.

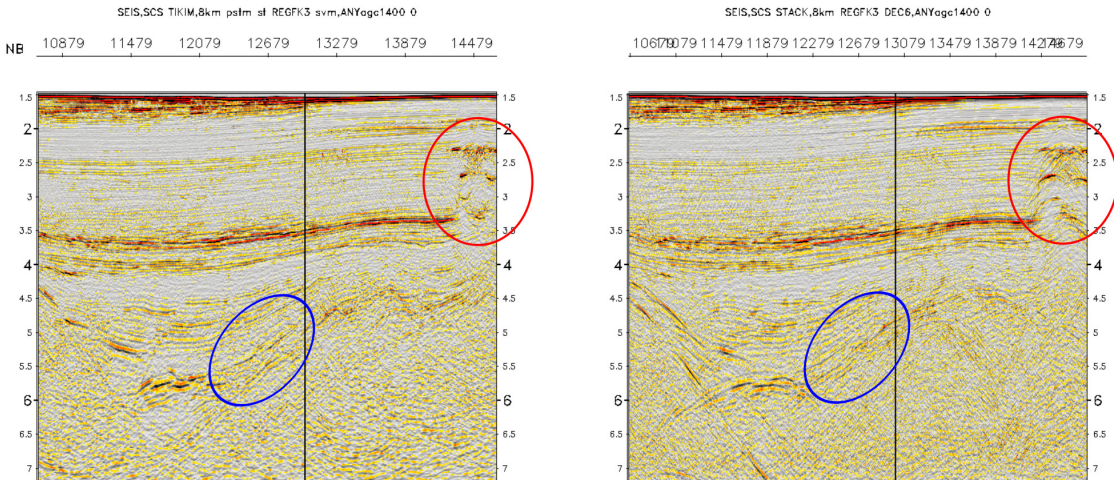


Figure 1.3 – A 2D line from the China Sea before (right) and after (left) Prestack time migration. The red ellipse marks the focusing of some diffraction hyperbolas. The blue ellipse shows a dipping reflector with a corrected slope. **To read this image:** The vertical axis is in seconds two-way travel-time (sTWT from the source to the reflector and back to the receiver). The horizontal axis shows the common depth points number (CDP, which are consecutive physical points imaged by seismic rays of various incidence angles). The distance between two CDPs is 6.25m and the total section is 25km long. The first reflector at 1.5 sTWT is the seafloor, the initial 1.5sTWT corresponding to the propagation in water have been cut. The correspondence between sTWT and depth requires the knowledge of the waves velocity in the media. A propagation of acoustic waves at 1500 m/s in water indicates that the seafloor is at a depth of 1125m. Typical marine sensors (hydrophones) measure the pressure variations (scalar field) caused by the waves. The colorbar of the image is centred around zero (white). (Provided by Delescluse (2020)).

Stack and Prestack Domains

After migration, seismic amplitudes are stored on a regular grid representing a discreet model of the subsurface. A representation of the grid is drawn in Figure 1.4. The local coordinates are inherited from the acquisition set-up. In a standard marine survey, the ship towing the equipments is navigating back and forth following a constant heading and progressively creating a 2D mesh at the sea surface. The sailing direction is called the **Inline** and the perpendicular direction the **Crossline**. The vertical direction can either be Time or Depth depending on the working stage. Datasets are typically recorded over hundreds of inlines and crosslines, during several seconds at a sampling rate of few milliseconds, leading to data sizes that can reach terabytes. It is common to look a the data in 2 dimensions for practical purpose. We give the naming conventions of the different 2D sections through the 3D data in Figure 1.4.

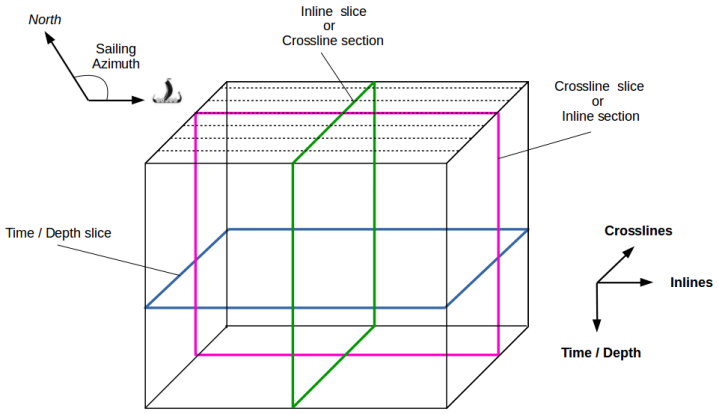


Figure 1.4 – A model of a 3D grid storing stacked seismic amplitudes.

As an example, the image on the right in Figure 1.5 is a crossline section (or equivalently an inline slice). This means that we are looking at the data at a fixed inline, seeing how it changes with time and crosslines.

Because of redundancies in the acquisition, a single point in the subsurface is imaged by more than a single wave. In Figure 1.1, the variable h, named the **offset**, represents the distance between the source and the receiver. Using ray-tracing in a velocity model, the offset can be related to the **incidence angle** θ, which measures at a reflection event the inclination between the incoming wave and the vertical. These extra measurements provide a fourth dimension to the data, called the **prestack** dimension. A collection of vertical seismic traces (extracted at a constant inline and crossline), and changing with time and offset (or angle) is called a **gather** (as an example there are eleven gathers displayed on the left in Figure 1.5). By summing contributions from all offsets (or angles), we fall back to a 3D dataset. This summation process is named stacking and therefore we often refer to the 3D data as a **stack**. We show in Figure 1.5 a dataset in the prestack and stacked domain side-by-side. The prestack signal is useful because its summation enhances the signal-to-noise ratio, and it also enables geophysicists to gain knowledge about the elastic properties of the layers since they relate to how amplitudes change with the incidence angle. More recently, some datasets with a 5^{th} dimension associated with the azimuthal orientation of the incident waves have appeared.

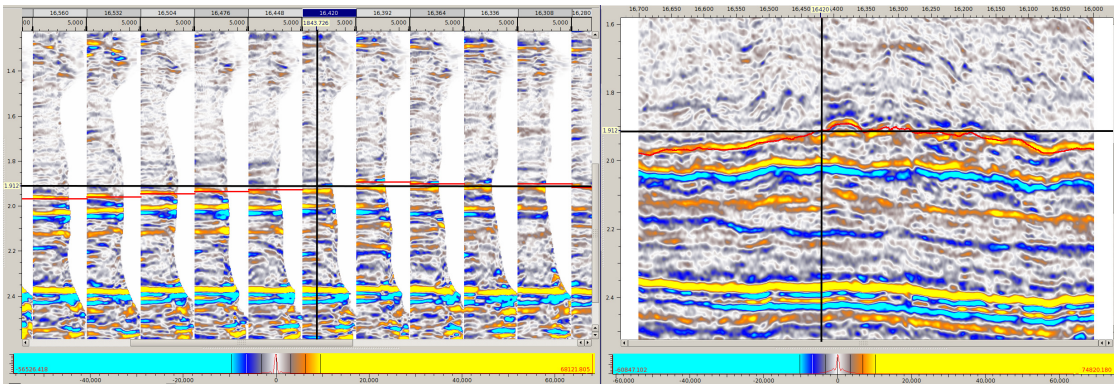


Figure 1.5 – Crossline sections of a seismic dataset. Prestack data is displayed on the left and stacked data on the right. The black crosshair shows the synchronization between both viewers. The vertical axis is time. On the stack, the horizontal axis is the crossline number. At every vertical trace of the stack is associated an offset gather in the prestack domain. The horizontal axis for the prestack data is 2-dimensional. The fast dimension is the offset and the slow one the crossline. We obtain the stack data by summing the prestack traces along the offset dimension.

1.1.2 Processing and Interpretation

Processing

As we mentioned previously, many processing sequences need to be applied in order to transform seismic records into an interpretable image. Those sequences aim at enhancing the events relating to key physical factors (in orange in Figure 1.1) and isolate them from other events generated by factors that do not inform us on the geology (in blue and green in Figure 1.1). There is no single processing workflow that can be applied to every datasets and geophysicists rather adapt to the data and the needs of the interpreters. Yilmaz (2001) gives a large list of existing algorithms and their purpose. Important pre-migration steps are the removal of source effects and the separation of primary and multiple reflections.

Figure 1.2 illustrates the effect that the wavelet has on the seismic signal. This underlines the importance of processing algorithms that transform the source wavelet. In particular, the original source wavelet may not be compact and may be contaminated with echoing effects. By reshaping the wavelet to be near zero-phase and free of ringing, we greatly improve the resolution by sharpening the image and allowing to better see the layer interfaces (see the difference between the first and last trace in Figure 1.2). Multiples, i.e. waves that reflected

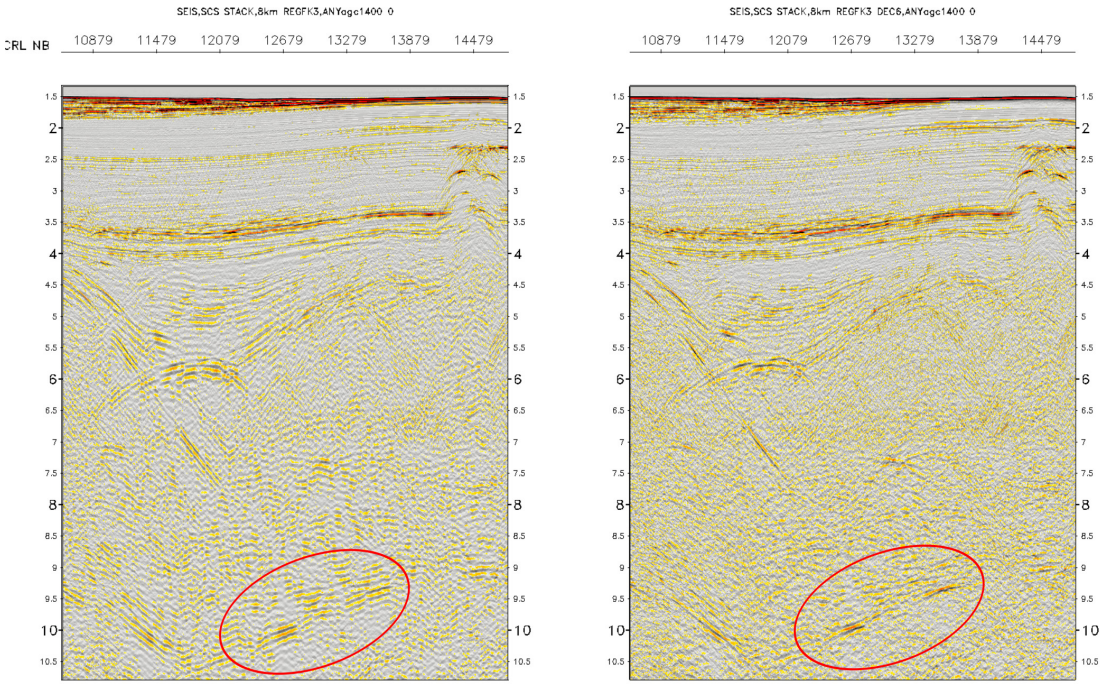


Figure 1.6 – A seismic line before (left) and after (right) deconvolution. The operation sharpens the wavelet. Reflectors, especially in the deeper part, are more clearly defined (see e.g. the reflection marked by the red ellipse). This operation also boosts high frequency noise. (Provided by Delescluse (2020)).

more than once, can obscure reflections by interfering with them or can lead to confusion by appearing as fictive reflections in the data. Figure 1.6 shows the effect of the deconvolution operation on the wavelet and Figure 1.7 illustrates the removal of multiples.

After migration, several processes can be employed to correct the remaining noise that was not accounted for during the pre-processing. Residual move-out and trim-statics are for instance employed to rectify reflections in the prestack domain that are not properly flattened because of errors in the processing velocities. Other important steps involve the use of well data to better assess the shape and phase of the wavelet, to re-position reflections to a more accurate depth and correct the amplitude versus angle behaviour.

Interpretation

Gaining knowledge from seismic data requires to skilfully marry geological and geophysical concepts. Seismic waves only offer an indirect observation of the subsurface, and despite

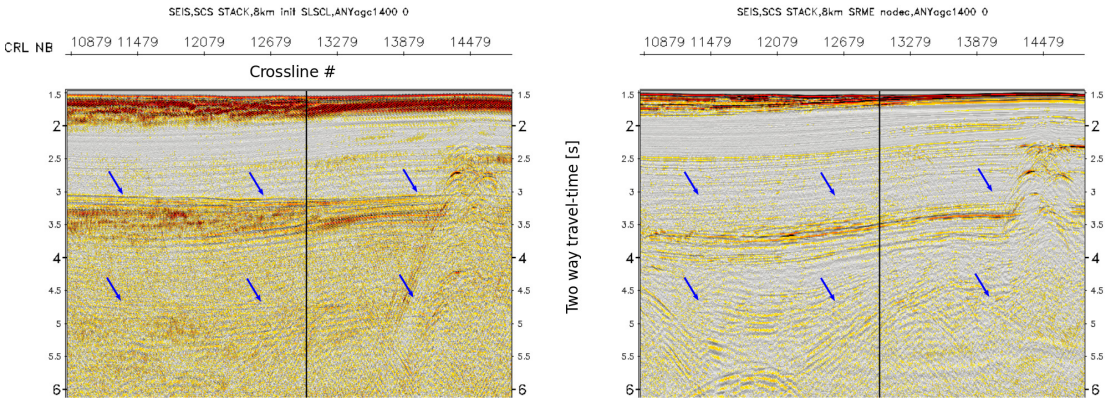


Figure 1.7 – Surface related multiple elimination (SRME, Verschuur et al. (1992)). On the left image, surface multiples of the basin are visible around 3s and 4.5s (blue arrows). Once removed (right), the primary signal is visible.(Provided by Delescluse (2020)).

the processing they remain contaminated with noise and have a limited resolution. Together with the sometimes complex subsurface geology, this explain why interpretation is subject to uncertainties and reaching a consensus between experts is not always possible. Additionally, interpretations are also influenced by the goal of the study, and different aspects will be considered whether one performs natural resources exploration or are interested in the understanding of the tectonic processes. In their book, Bacon et al. (2003) separate the interpretation process into three main groups: structural, geological and quantitative.

Structural geologists are concerned with understanding how structures were formed. Their goal is to be able to tell a chronological story of the different depositional and deformation processes that occurred in order to explain the observed wavefield. They have to make sense of the 3-dimensional structural complexity by looking at either 2D or 3D images. In particular, a large portion of their time is allocated to the picking of events such as horizons, fault networks or channels. An example of manual interpretation of a 2D line is shown in Figure 1.8.

In geological interpretation, one aims to associate the observed reflections and structures with actual rock types. Using general knowledge about the area and the depositional system, one can try to relate observations with the stratigraphy. In addition to seismic, well measurements, when available, are also valuable. Those offer a direct observation of the media and allow to tie some reflections to stratigraphic sequences and position them at their true depth. One can also try to classify areas into groups of similar seismic characters. Those characters, or facies,

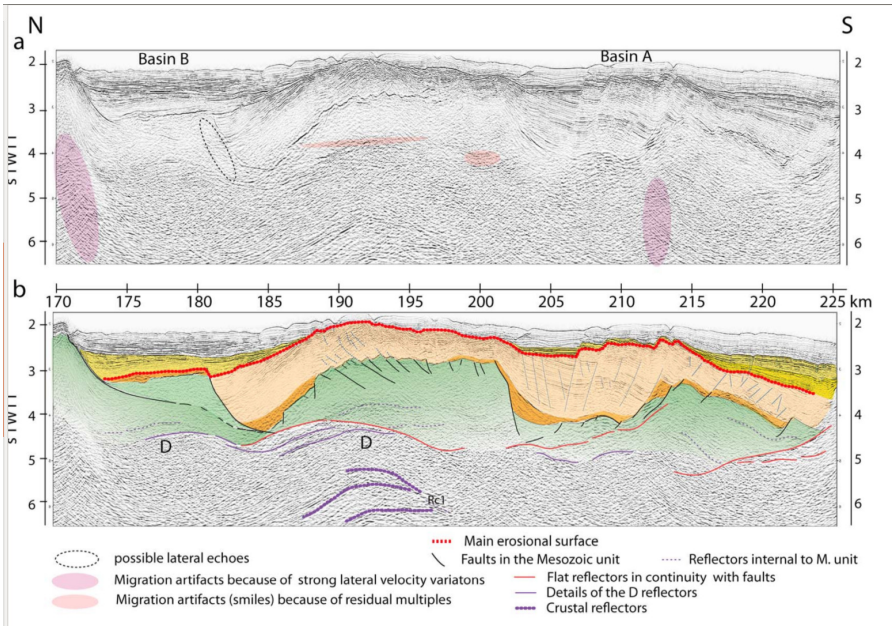


Figure 1.8 – Interpretation of a seismic line from the China sea. (a) Highlights of noisy signals. (b) Interpretation overlayed on the amplitudes. (From Liang et al. (2019)).

provide maps of the lithology and potential distribution of fluids.

Quantitative interpretation uses seismic amplitudes, well logs and knowledge about the wavelet to infer physical properties such as the acoustic and elastic impedances. Using well known equations derived from the physics of wave propagation, one tries to find the value of the parameters that best explain the observed wavefield. Amplitude versus angle (AVA) analyses the response of a reflection in the prestack domain by looking at how amplitudes are affected by the incidence angle (variable θ in Figure 1.1). Different AVA responses may be associated with different lithologies and fluid contents. Inversion is an other approach that aims to convert information from the layer boundaries into information about the layers themselves. In Figure 1.9 we show an example of waveform tomography employed to derive an accurate velocity model.

1.2 Machine Learning and Deep Learning

Machine Learning (ML) and Deep Learning (DL) are fields dealing with the study and design of algorithms that can perform specific tasks without having been programmed with explicit

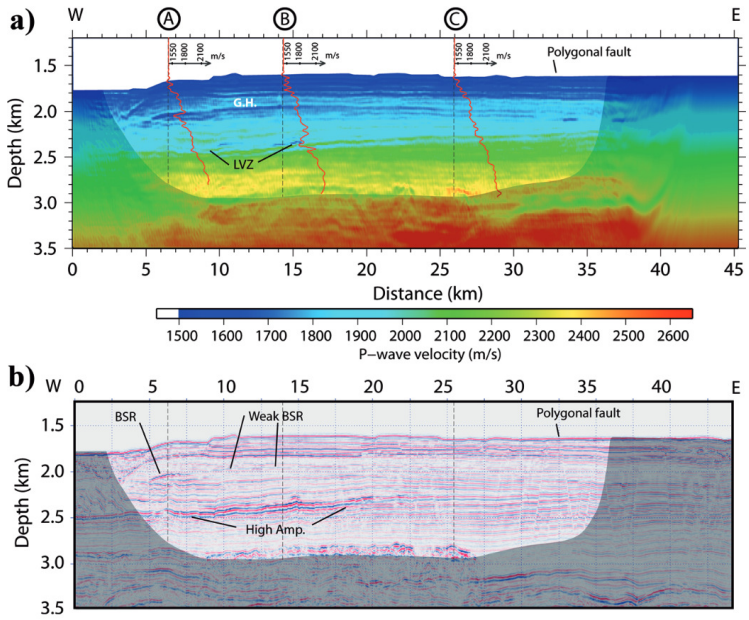


Figure 1.9 – Waveform tomography applied to long-streamer data from the Scotian Slope. (a) Inverted P-wave velocity model. (b) Prestack depth-migrated data shown for comparison. (From Delescluse et al. (2011)).

instructions. To build their computing model, these algorithms instead rely on the patterns they discover in the data and progressively update their parameters to improve their performance. In the following we expose some introductory elements about the topic. We also have a more in depth look at deep learning in Chapter 2.

1.2.1 Concept and Ideas

Conventional Methods *vs* Machine Learning

Let us consider the problem that consists in associating elements from a space $\mathscr{X}$ to elements in a space $\mathscr{Y}$. Given pairs of samples $(x, y) \in \mathscr{X} \times \mathscr{Y}$, the task is to find a function f that successfully maps the elements, i.e. such that $f(x) = y$. Depending on what the spaces represent, the correspondence between $\mathscr{X}$ and $\mathscr{Y}$ can take many forms. In computer vision, the input domain could be natural images and the output domain a set of abstractions describing the objects and symbols present in the image. f is then a map that should reflect the established semantics (*"In this picture, there are two cats and a dog."*). In Geophysics, $\mathscr{X}$ could be a space

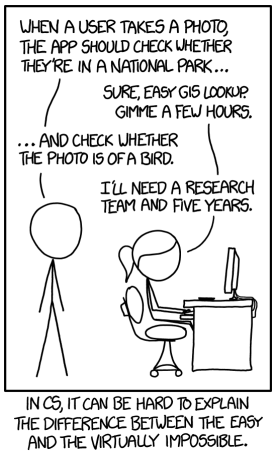


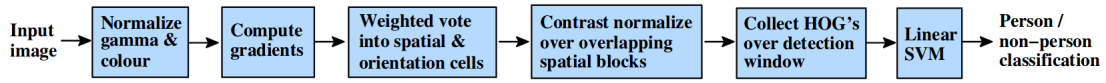
Figure 1.10 – This cartoon was released before the new machine learning revolution (see Section 2.2.1). Explicitly defining a set of rules to perform a task may prove surprisingly difficult even for problems that are trivially handled by humans. GIS stands for Geographic Information System and CS for Computer Sciences. (Source: *xkcd.com/1425/*)

of physical parameters and $\mathscr{Y}$ a space of observations. f would then be a forward operator describing the physics of the problem.

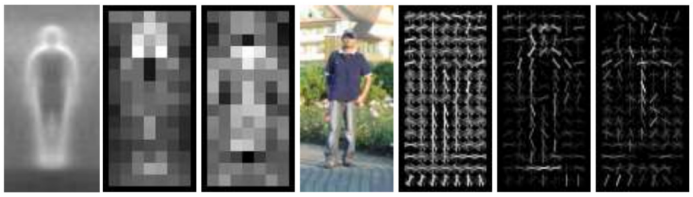
A conventional procedure to build f is to define a set of hand-crafted rules and program them to create an algorithm. In the case of a geophysical forward problem, those rules come from physics and are programmed by a discretization of the governing equations. It is however not always easy to find them, as it is for instance illustrated by Figure 1.10. Conversely, with machine learning, f is a generic function built with a number of free-parameters and asked to find its own rules by learning from the data. There exists many families of machine learning algorithms. Among the most widely used are linear regression models, decision trees, support vector machines or neural networks (see e.g. Bishop (2006)). We explain how the function is learned, in the case of neural networks, in Chapter 2.

Machine learning is therefore appealing since it offers the promise to substitute itself to the difficult problem of finding rules to solve a task. Moreover, hand crafted algorithms are usually highly specific. For example, the instructions to recognize a bird are completely different from the ones to recognize a tree and this work must be done for every new object that one which to find. On the contrary, the same learning algorithm can be used for all objects.

Machine Learning & Deep Learning



(a)



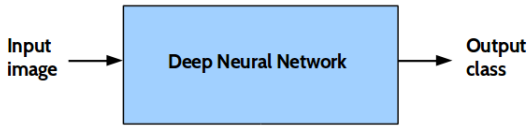
(b)

Figure 1.11 – Example of a ML workflow proposed by Dalal & Triggs (2005) to identify pedestrians in video frames. A preprocessing pipeline successively transforms the input images with predefined filters and trains a simple linear support vector machine (SVM) classifier to solve this complex task. (a) Desciption of the workflow. (b) Examples of explicit feature engineering.

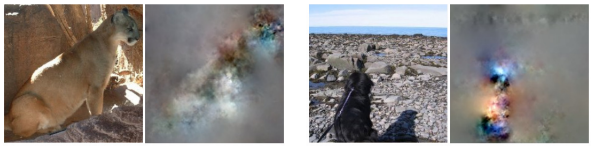
Most machine learning algorithms have a fairly compact design. They receive input data and perform a small number of operations to yield the output data. This equips them with a rather small representation power, i.e. that they are limited in their capacity to extract patterns from the input data and combine them to make a decision. In practice, this means that their application is restricted to simpler problems where the input data contains only limited features and where these features are informative enough to help finding the correct answer in a straight forward manner. To overcome those limitations, researchers have developed strategies to simplify the job of the ML algorithm. In particular, complex pre-processing pipelines are involved in order to transform the data to a more appropriate domain. Many hand-crafted filters have been proposed to emphasis on certain characteristics of the data which are believed to be important to solve the problem. An example of such pipeline is given in Figure 1.11.

Deep learning (DL) is a subset of machine learning, and DL models are employed in a similar fashion as the ML ones. The terminology *deep* refers to the particular design of the algorithms. The relationship between the input and output data is no longer determined by a single or by small number of operations, but by a larger sequence of them. The main representatives of DL are neural networks (Rosenblatt, 1958). This sequential construction, or construction in layers, where each layer performs simple operations and forward the results to the next, turned out

to be a very powerful way to increase the representation power of learning algorithms (LeCun et al., 2015, Schmidhuber, 2015). In particular, DL is not restricted to simpler problems and no longer requires that custom processing pipelines be designed by experts, as illustrated in Fig.1.12. However, a drawback of DL over ML is that it usually requires much more data for learning and the preparation of this data may be very challenging. We go deeper into the details of DL in Chapter 2.



(a)



(b)

Figure 1.12 – Deep learning does not require any specific manually engineered transformations as opposed to machine learning (Figure 1.11). Both the data transformations and the decision making are learned in a unified fashion by the algorithm. (a) Typical workflow in image classification. (b) Examples of transformation learned by the network (taken from Olah et al. (2017)).

Artificial Intelligence ?

The term *learning* is employed to describe the algorithms capacity to self-improve when given feedback. However, the notion of improvement is based on a performance measure specifically designed to evaluate a chosen task. This concept does not involve cognitive notions and the relationship between the term learning applied to machines and learning in the human sense is not obvious (Mitchell, 1997). Similarly, the notion of artificial intelligence (AI) can't be easily defined. Broadly speaking, AI is the field that studies intelligence demonstrated by machines. Many branches of science, engineering and philosophy are involved in this study and deep (machine) learning is only a part of it. Nevertheless, it is common to see the terminology AI

substituted with deep learning in scientific writings or in the media [1].

Because the concepts of intelligence and consciousness are difficult to understand and define, it is not clear what a real AI will look like. It is also not clear whether incremental improvements made in the fields of neural networks and DL will one day lead to the emergence of such AI. In this thesis we prefer to stay clear from this notion and limit the use of this terminology.

1.2.2 Some Applications

The formulation of a ML algorithm is very generic and models are mostly free of a priori constraints which gives them the capacity to adapt to many problems. For this reason, ML can be applied to practically every domains of science and engineering. As long as data are available and that the task we aim to accomplish posses a relationship to these data, it is possible to train a model. Some of the application fields, relevant for geosciences, that benefit from machine learning are computer vision, speech recognition or time series forecasting.

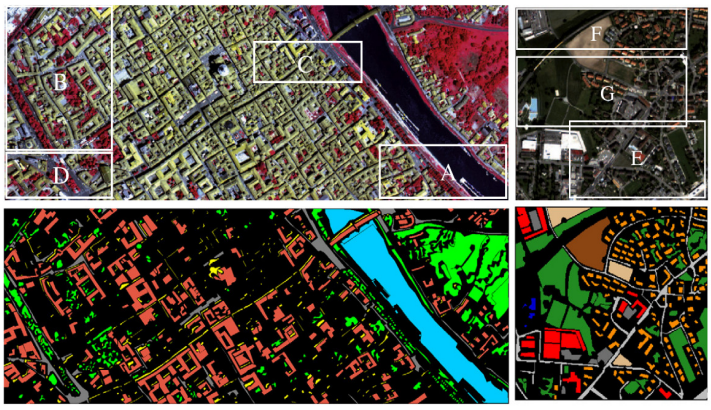


Figure 1.13 – Machine learning based segmentation of different objects (buildings, roads, water, vegetation and shadows) on hyper-spectral satellite images. From Tuia et al. (2011).

Computer vision is the field devoted to creating computer programs that are able to analyse and reason about digital data. These data are traditionally natural images and videos, and when working with recordings of human voices one rather speak of speech recognition. When employing machine learning to tackle those tasks one also refer the process as pattern recog-

[1] e.g. https://www.nytimes.com/2019/10/24/well/live/machine-intelligence-AI-breast-cancer-mammogram.html ; https://www.bbc.com/news/technology-49165713

nition. Those fields share similar goals and largely overlap in their methods and by extension they also apply to other forms of data such as seismic recordings or satellite observations. One of the main application is classification (or segmentation), where the purpose is to associate meta-information to the data. Such information may for example identify the nature and the position of various objects contained in the data. We show an application example in Figure 1.13, where ML is employed to locate as set of objects, such as buildings and vegetation, in satellite images.

Other interests lie for instance in learning the underlying physical process of a phenomenon and use this knowledge to predict the future behaviour of a dynamic system or invert for the physical parameters that best explain the observations. As examples, Krasnopolsky & Fox-Rabinovitz (2006) proposed to use a neural network to forecast the weather and Röth & Tarantola (1994) to invert for a velocity model from seismic data. Those tasks can be performed using physical models instead, but ML and DL approaches are usually much easier to programme and some authors claim that DL can outperform algorithms based on explicit physical laws (see e.g. Araya-Polo et al. (2018)). Once trained, neural networks based algorithms also tend to be computationally much more efficient than their classical counterpart.

1.3 Deep Learning for Seismic Analysis

1.3.1 Motivations and Challenges

Motivations

In the previous sections we gave an overview of seismic data processing and interpretation. Extracting useful information from the recorded wavefield is a long and difficult process that requires the intervention of many experts. We list in the following some of factors explaining the complexity of the task:

- Datasets are usually very large, with millions of traces arranged in a multi-dimensional grid (typically 3D, 4D and even 5D). This is a challenge for writing algorithms that must bear the computational and memory burden. It is also challenging for experts to be able

to explore a dataset in its entirety.

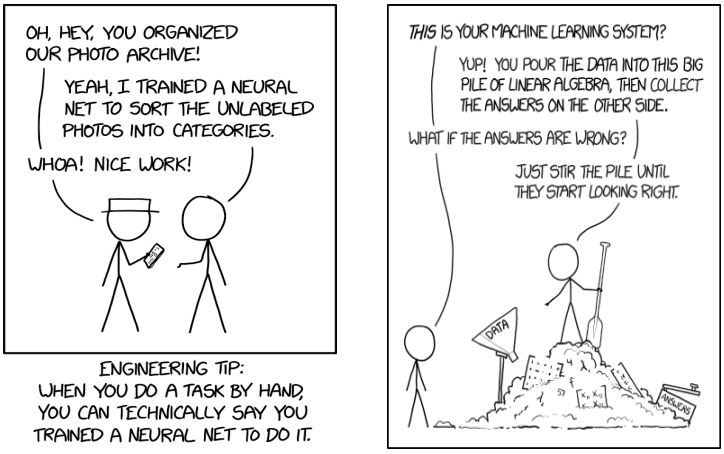
- In order to be interpretable, raw data need to be transformed with a chain of complex processing operations. Each operation is difficult to design and delicate to implement and requires a careful parametrization by experts every time it is employed.
- Qualitative interpretation requires to understand the 3-dimensional structure of the data and to annotate many features over the entire grid. Picking events such as faults and horizons is a tedious and time consuming task. Geologists also often need to look at a large number of attributes designed to help highlighting certain characteristics in the data, which multiply the effective dimension of the problem.
- Quantitative interpretation is also challenging. Clues in the amplitudes are strongly affected by noise and a lack of resolution. It is difficult to write robust algorithms, and they need to be carefully calibrated using various data sources (seismic and wells).

Machine learning, and in particular deep learning, has recently become extremely popular. Learning models have shown a very good potential on a variety of data analysis tasks and have set the new state of the art in many cases (LeCun et al., 2015, Schmidhuber, 2015). Problems that were until recently considered very difficult, such as robust object recognition in images (see Figure 1.10), are now routinely tackled using neural networks (Krizhevsky et al., 2012). Moreover, ML and DL algorithms are versatile. A single model can be used to solve many different problems. They usually require much less efforts from experts to be designed and programmed than traditional algorithms and offer the promise to perform sometimes far better. Additionally, if repetitive and time consuming tasks can be automated, this will allow experts to focus on the more interesting aspects of the interpretation. For these reasons it is worth to look at applications in Geosciences and see if one can alleviate the difficulties listed above. We give in the following some remarks and questions to be investigated:

- Machines are less penalized than humans when dealing with large and high dimensional datasets. Does it mean they have a natural potential to outperform geoscientists in processing and interpretative tasks?
- Can we improve the performances when substituting hand crafted processing algorithms by DL? Once trained, can a learning model automatically adapt to new data without the need for experts to re-parametrize it?

- Can we improve existing automated methods to perform picking by replacing them with DL? Can we use DL to pick features in difficult regions where only humans could succeed so far?
- Can we by-pass physics and use DL to directly and efficiently extract qualitative and quantitative information from the data?

Challenges



(a) (b)

Figure 1.14 – Cartoons making allusions to the dependence on human intervention of machine learning and its empirical nature and lack of explainability. (source: (a) *xkcd.com/2173*, (b) *xkcd.com/1838*).

While deep learning as been shown to outperform traditional methods in many applications, fulfilling the conditions required to efficiently exploit its power can be challenging, especially when working with geoscientific data. Figure 1.14 shows two popular cartoons that depict some problematic aspects of ML and DL.

Preparation of the training data: A first issue lies in the strong dependency of the training process on human intervention. For many applications, in particular in pattern recognition, the observed data are not enough in themselves and one also need to enrich them with a variety of meta-information describing their content. This enrichment, named labelling, is the pillar of many applications. Moreover, for real world applications, it has been discovered that the quantity of labelled data necessary for training was substantial. As an example, in

the famous work of Krizhevsky et al. (2012) to automatically detect objects in images, the training dataset consisted in more than one million images all annotated by hand (Deng et al., 2009). The preparation of such a profusion of examples is a considerable chore. To alleviate this problem, the machine learning community has adopted a largely open-source approach where teams around the world collaborate and big companies voluntary contribute to prepare datasets accessible to all. In seismic exploration however, most of the data and interpretations are kept secret because of their strategic value and getting access to good training datasets may be a major bottleneck for the development of machine learning applications.

Accounting for uncertainties and qualitative assessment: Another difficulty when working with seismic data arises from the approach employed for training. The performance metric, that quantitatively measures the quality of the answer of learning models, is the cornerstone of the procedure. It is used by the machine to update its parameters and by the practitioners to evaluate, improve and communicate about the quality of their algorithm. This measure is very often given as a single score between 0 and 1, where approaching 1 is considered equivalent with approaching exactitude. The assumption behind this paradigm is that one have access to the *true* answer, i.e. that one have perfect knowledge of the output space and a way to quantify it. In the work of Krizhevsky et al. (2012) for instance, this assumption is valid. They indeed have access to a dataset well representative of the diversity of the real world, there is little ambiguity in the labelling process and it is easy to quantify the answer of the machine by simply comparing it with the answer given by humans. In seismic, this assumption is less valid. Besides the difficulty to get access to a large and representative training dataset, there are additional issues brought by the inherent uncertainties in the field. The noise and lack of resolution of seismic data inevitably lead to uncertainties and non-uniqueness of their interpretation. This in turn weakens the significance of any performance metric as errors and human biases will be communicated to the machine. There are also cases where quantifying the quality of a result is difficult. In processing for instance, it is not trivial to tell how much better (or worse) is one version of the transformed data with respect to an other version of the transformed data. This sometimes forces the need of a more qualitative assessment of the results. This is problematic because this assessment is subjective and requires efforts to be communicated, which does not make it well accepted in the data science community.

Additionally, this judgement can't be automated, which makes the training procedure more difficult.

Weak theoretical foundations and lack of interpretability: Another concern often raised when working with deep learning is the lack of theoretical guaranties and explanations. Training an algorithm is a largely empirical process conducted by trials and errors until satisfactory results are obtained (see Figure 1.14b). It is not well understood why algorithms, and in particular neural networks, can perform so well and what is the process that leads them to give their answer. This lack of interpretability may be a problem, especially in geosciences where an interpretation or inversion is never certain and is only accepted by ones peers if one is able to reasonably explain how we came to this result. There is also no definitive guaranties about the general quality of a model. In practice, models are judged by providing a performance score obtained during a blind test. But agreeing that a good score is equivalent to good generalization (i.e. that the model will perform well with any new data), requires the assumption that the blind test data is highly representative of the diversity of the world. Obtaining excellent results during synthetic tests is not a proof that results will be trustworthy on real data.

1.3.2 Thesis Outline

In this thesis, we explore applications of machine learning, and especially deep learning, to solve various problems in seismic processing and interpretation. In particular, we propose methodologies that aim to overcome the challenges associated with the use of deep learning and aim to develop workflows that have practical uses.

We first give an overview of deep learning in Chapter 2 and provide details about the internals of neural networks and the training procedure.

In Chapter 3, we show how with *transfer learning* and *knowledge distillation* one can obtain state-of-the art results in picking tasks without needing to label by hand an extensive amount of data. In particular, we show how synthetic modelling allows us to use physics to create well controlled training datasets and that our trained model successfully handles real data. We also show how one can take advantage of pre-existing auto-picking algorithms to train neural

networks and progressively outperform the results by a noticeable margin.

In Chapter 4, we demonstrate the use of *interactively supervised learning* to execute a challenging interpretation work. We show how human intervention can be integrated together with the automatic process in order to progressively guide the algorithm toward an accepted answer. We also explore the possibilities offered by *unsupervised learning* to see if an algorithm can provide useful information without the need of human intervention.

In Chapter 5, we focus on processing applications. We show how a simple application of unsupervised learning can be used to remove random noise in the data and how supervised learning and transfer learning can be employed to attenuate more complex forms of noise.

We also share in Appendix A the programming techniques and the libraries that we used to generate the results shown in this thesis.

CHAPTER 2 DEEP LEARNING: THEORY AND PRACTICE

2.1 Résumé

Dans ce chapitre nous donnons une vue d'ensemble de l'apprentissage machine profond.

Nous commençons par exposer les origines de ce domaine et expliquons les raisons pour lesquelles cette branche des statistiques à récemment connu un renouveau d'intérêt spectaculaire. Nous introduisons également la technologie principale derrière l'apprentissage profond, les réseaux de neurones à propagation avant, ainsi qu'une de leurs incarnations les plus utiles, les réseaux convolutifs.

Dans une seconde partie, nous rappelons les différents paradigmes qui permettent d'entraîner ces algorithmes et revenons sur la méthode de rétro-propagation des gradients ainsi que sur les techniques d'optimisation des paramètres des réseaux de neurones.

Enfin, nous donnons quelques éléments pratiques sur l'utilisation de ces méthodes. En particulier, nous insistons sur l'importance de la préparation des données d'entraînement et sur le rôle central joué par l'évaluation quantitative des performances. Nous mentionnons également les facteurs à prendre en compte pour le choix de l'architecture du réseau et des paramètres de réglage.

2.2 Deep Neural Networks

In this chapter, we give a basic overview of the field of deep learning. After briefly introducing neural networks, we present the main concepts behind the learning process and we discuss

practical aspects that intervene when resorting to this technology. For a more thorough introduction to the subject, interested readers may refer to Goodfellow et al. (2016) and Nielsen (2015).

2.2.1 History

While deep learning has recently known a considerable renewal of interest, its foundations were laid during the second part of the 20^{th} century. Early developments arose in attempts of creating computational models for the human brain (McCulloch & Pitts, 1943, Rosenblatt, 1958). This inspiration explains why the field often borrows terminologies from neurosciences, such as the naming *artificial neural networks* to designate computing systems inspired by biological neural networks. Further progress were made by refining the architectures of the networks (e.g. Fukushima (1980)) or improving the training procedures (Rumelhart et al., 1985). A large portion of the field is now disconnected from biological influences, and modern algorithms may or may not be working in a manner analogous to the brain (Goodfellow et al., 2016). While research in the field was prolific during this period, deep learning did not gain much popularity outside of the concerned community (LeCun et al., 2015). Among the reasons for this lack of awareness were the computational limitations that restricted applications mostly to small sand-box experiments, as well as the lack of publicly available datasets that could be used to train the algorithms. However, a booming event appended in 2012 when Krizhevsky et al. (2012) won a prestigious image classification competition, by a surprisingly large margin, using deep learning. While their algorithm was not much different from earlier works (e.g. by LeCun et al. (1989)), they benefited from two key factors. First, their implementation was among the first to be compatible with modern graphical processing units (GPUs), leading to computation performances orders of magnitude better than the previous state of the art. Secondly, and maybe most importantly, they trained their network on one of the first very large scale public database. Created by Deng et al. (2009), the *ImageNet* dataset originated after the authors correctly speculated that one of the biggest limitation of deep neural networks did not lie in algorithmic details but rather in the lack of available data examples they could learn from. The original dataset took approximately five years of tedious work to gather and label by hand more than a million images.

Presently, theoretical and applied research in deep learning is extremely active and neural networks are being employed in nearly every existing scientific and engineering fields. Besides the promises of much greater performances offered by neural networks, the exponential growth of the field can also be explained by the very open nature of the community. Most publications and improvements made to software and databases are openly accessible to everyone. Libraries like Keras (Chollet et al., 2015) and cuDNN (Chetlur et al., 2014) enable to build and train neural networks yielding near state of the art results on a benchmark by writing only few lines of code. Another factor explaining the great number of daily new results in the field lies in its highly empirical nature. It is still not well understood why neural networks perform so well and how they could be better designed (Zeiler & Fergus, 2014), and most improvements, or at least new ideas, come from a trial and error process over a virtually infinite set of possible modifications. While these factors make the entry level barrier to use deep learning low, they also bring some challenges. It is indeed difficult to keep up with the evolution of the state of the art and to filter through the many research results that claim superior performances while not always being generalizable to ones own problem (Lipton & Steinhardt, 2018).

2.2.2 Feedforward Networks

The prominent model for building neural networks is the **feedforward** model. Let f^* be a function that maps a variable $\mathbf{x} \in \mathcal{X}$ to a variable $\mathbf{y} \in \mathcal{Y}$, where $\mathcal{X}$ and $\mathcal{Y}$ are respectively the input and output domains, $\mathbf{y} = f^*(\mathbf{x})$. A feedforward network defines a mapping $\mathbf{y} = f(\mathbf{x};\boldsymbol{\theta})$ and learns the values of the parameters $\boldsymbol{\theta}$ such that f best approximates f^*. The term feedforward refers to the direction of the information flow. The function f is designed as a sequence of operations, each taking an input and passing its output to the next operation. The sequence builds a unidirectional link from $\mathbf{x}$ to $\mathbf{y}$. Models that include feedback loops, such as the recurrent neural networks (Rumelhart et al., 1985), are not covered here. A **network** is defined as the sequence of operations expressed as the successive composition of several functions. For a sequence of p operations described by the set of functions $(f^{(i)})_{i=1..p}$, the network is described by the chain $f(\mathbf{x}) = f^{(p)} \circ f^{(p-1)} \circ ... \circ f^{(1)}(\mathbf{x})$.

Individual functions are associated with **layers** and the length p of the chain defines the **depth**

of the network. The function $f^{(1)}$ receiving $\mathbf{x}$ defines the input layer and the function $f^{(p)}$, yielding $\mathbf{y}$, defines the output layer. Intermediate functions are usually said to represent so-called **hidden** layers. One of the earliest and most prominent transformation used in neural networks is the fully connected layer (Rosenblatt, 1958). Let's assume that the function $f^{(l)}$ implements a fully connected transformation from layer $l-1$ to l. The function treats its input data as a one dimensional vector, and performs an affine transform defined as a matrix-vector multiplication followed by a vector addition. We introduce the notations in Figure 2.1. Black arrows indicate the directions of the data flow between individual **units**, also called neurones, represented by blue circles. Layers are composed of individual units and their number defines the **width** of a layer. Layer $l-1$ is composed of n_{l-1} units identified by index i and layer l contains n_l units identified by index j. We note $\mathbf{w}^l \in \mathbb{R}^{n_{l-1} \times n_l}$ the matrix whose individual elements w^l_{ji} are the **weights** connecting the i^{th} unit of layer $l-1$ to the j^{th} unit of layer l. $\mathbf{b}^l \in \mathbb{R}^{n_l}$ is the **bias** vector containing the individual scalar terms b^l_j of the l^{th} layer. Finally we denote by $\mathbf{a}^{l-1} \in \mathbb{R}^{n_{l-1}}$ the output data of layer $l-1$ (and input data to layer l) and $\mathbf{a}^l \in \mathbb{R}^{n_l}$ the output data of layer l. Individual units are responsible for computing individual output elements a^l_j. The exact operation performed by the layer is given by Equation 2.1:

$$a^l_j = \sigma\left(\sum_{k=1}^{n_{l-1}} w^l_{jk} a^{l-1}_k + b^l_j\right) \quad . \tag{2.1}$$

In addition to the affine transformation, an additional non-linear function $\sigma()$ is applied. This function is called the **activation** function and is used to break the linearity between successive layers. Being able to accuratly approximate non-linear processes is important when solving non-trivial problems. Because of the use of this function, the output vector $\mathbf{a}^l$ is called the activation of layer l. The function $\sigma()$ is parameter free and applied independently to every element. Common choices are the sigmoid function ($\frac{1}{1+e^{-x}}$) or the rectifier ($\max(0,x)$) (Hahnloser et al., 2000). Equation 2.1 can also be written in vectorized form:

$$\mathbf{a}^l = f^{(l)}(\mathbf{a}^{l-1}) = \sigma\left(\mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l\right) \quad . \tag{2.2}$$

The complete network is defined by chaining fully connected layers. The width n_1 of the first layer needs to be chosen equal to the size of the input elements $\mathbf{x}$, and n_p should be equal to the size of output elements $\mathbf{y}$. Intermediate widths are hyper-parameters to be chosen by the practitioner (see Section 2.4.3). Given that they are provided with enough capacity (i.e.

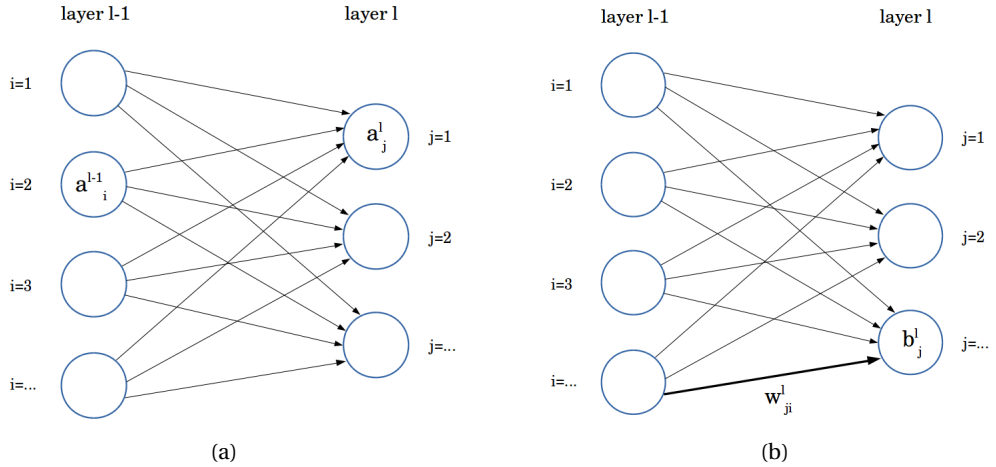


Figure 2.1 – Diagrams representing a fully connected operation transitioning from layer $l-1$ to layer l. The left diagram introduces notations for the data while the right diagram introduces notations for the parameters.

that they have enough parameters and non-linearities), neural networks satisfy the universal approximation theorem (Cybenko, 1989), which means they can be used as proxies to model many of the processes encountered in the real world.

2.2.3 Convolutional Networks

Convolutional neural networks (CNNs) (LeCun et al., 1989) are a class of feedforward networks that use convolutions as a linear transformation instead of the dense matrix multiplications employed in the fully connected layers presented above. CNNs are suited to process data that are represented on a regular grid and that exhibit spatio-temporal characteristics. Examples of such data can be an audio signal discretized at a constant sampling rate and expressed on a 1D grid, or a migrated seismic stack expressed over a 3D grid.

Convolutions are a common transformation in image and seismic processing. Given a time series $\mathbf{x}(t)$, many operations, such as a frequency filtering, can be expressed as a convolution operation. Using an appropriate kernel $\mathbf{w}$, the convolution will morph the input into a new signal $\mathbf{s}(t)$. The operation is defined as the shifting dot product between the reversed input

signal and the kernel:

$$\mathbf{s}(t) = (\mathbf{x} * \mathbf{w})(t) = \int \mathbf{x}(t-\tau)\mathbf{w}(\tau)d\tau \ , \tag{2.3}$$

where τ is the integration variable over the support of $\mathbf{x}$. When expressed on a grid, Equation 2.3 writes as:

$$\mathbf{s}(j) = (\mathbf{x} * \mathbf{w})(j) = \sum_i \mathbf{x}(j-i)\mathbf{w}(i) \ , \tag{2.4}$$

where i is the integration index defined over the support of $\mathbf{w}$ ($\mathbf{w}$ is generally chosen to be smaller than $\mathbf{x}$). Figure 2.2 illustrates how a convolution can be described as a sparse interaction with the input data. For a kernel of size 3, only 3 weights are needed to compute an output element s_j, where n weights (n being the size of $\mathbf{x}$) are required in a dense matrix operation (see Figure 2.1). Therefore, the operation does not depend on the input size, which is an advantageous property when working on large data. Moreover, the weights of the kernel are shared among the input values since they will progressively slide across the entire vector. This leads the operation to be invariant under translation of the input. If, for instance, a convolution operation is used to detect a particular sound in an audio signal, the detection will not be affected whether this sound occurs at the beginning, middle or end of the recording. As a consequence, convolution based algorithms tend to require much less parameters than (dense) matrix based algorithms.

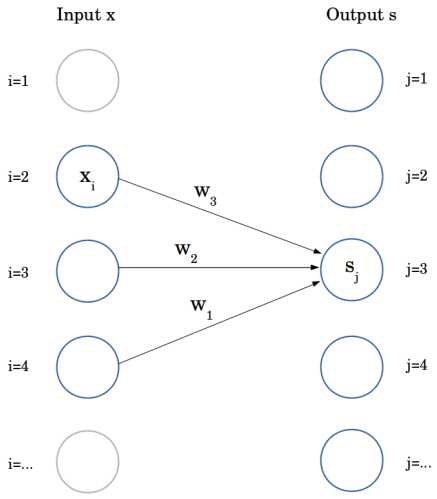


Figure 2.2 – Diagram representing the connectivity in a 1D convolution operation for a kernel of size 3.

Additionally, nodes of the input data grid usually do not store a single scalar value but rather a

vector of values. For examples, a picture stores at every node of the 2D grid a 3-dimensional vector holding values of the RGB color model. For seismic data, nodes of a 3D grid may store amplitudes recorded at various offsets or incidence angles. This extra dimension is refereed to as the **channel** dimension. For an input data containing p channels $\mathbf{X} = (\mathbf{x}_1, ..., \mathbf{x}_p)$, we also define a multi-channel kernel $\mathbf{W}^{(1)} = (\mathbf{w}_1^{(1)}, ..., \mathbf{w}_p^{(1)})$. A convolution is performed in parallel for every channel and the results are summed together as described by Equation 2.5:

$$\mathbf{s}_1(j) = \sum_{k=1}^{p} (\mathbf{x}_k * \mathbf{w}_k^{(1)})(j) \ . \tag{2.5}$$

While a convolution kernel can detect one particular feature, more advance filters will need to detect and combine multiple features to obtain good performances. For this reason, convolutional layers are performing multiple convolutions in parallel, each with their own kernel. If a layer contains q kernels $(\mathbf{W}^{(1)}, ..., \mathbf{W}^{(q)})$, the output data $\mathbf{S} = (\mathbf{s}_1, ..., \mathbf{s}_q)$ will then be composed of q channels as well. The number of kernels q per layer is a hyper-parameter to be chosen when defining the architecture (see Section 2.4.3). Defining also a scalar value $b^{(k)}$ associated with every convolutional kernel, the operation performed by a convolutional layer on a single input is summarized in Equation 2.6.

$$\mathbf{S} = \sigma\left(\left\{\mathbf{X} * \mathbf{W}^{(i)} + b^{(i)}\right\}_{i=1..q}\right) \ . \tag{2.6}$$

If we assume that boundary conditions of the convolution are treated with appropriately, the output signal will have the same spatio-temporal size as the input. However, an essential characteristics of CNNs is that they must be able to access the data at different scales (LeCun et al., 1989). In Figure 2.2 we took the example of a 1D convolution with a kernel of size 3. A layer composed of such kernels will never see more than 3 consecutive input samples at the same time, restricting its ability to analyse only fine scale features. The maximum number of input elements seen by a layer is called the **receptive field**. In order to access coarser scale information, this receptive field needs to be expanded. While increasing the size of the kernels is a possibility, it is not recommended in practice as it will lead to an increase in the number of parameters, making the algorithm less memory and computationally efficient and potentially introducing training instabilities (LeCun et al., 1989). Instead, **pooling** is commonly used to achieve this goal. A pooling operation will decrease the size of the input data by introducing some down-sampling rule. Max-pooling, for instance, will drop every grid node inside of a

defined window, keeping only the node containing the highest value. If the window is of size 2, this will down-sample the signal by the same factor. If we now send the down-sampled signal to a convolutional kernel of size 3, the effective receptive field, measured relative to the original signal, will then be 3×2. Alternating between convolutional layers and pooling layers will progressively increase the effective receptive field. Other mechanisms such as strided convolutions (Long et al., 2015) or dilated (atrous) convolutions (Mallat, 1999) can also be employed to change the effective receptive field of the network.

2.3 Training a Network

2.3.1 Different Forms of Learning

So far, we introduced how neural networks are defined but not how they are trained. At the initial stage, the parameters of a network are chosen at random, making them useless to solve a given task. However, these parameters are not fixed, and networks are given the freedom to modify them in order to improve their performance. This is the reason why we say that they are learning, as opposed to traditional algorithms which use fixed parameters manually engineered by experts. Among the various existing learning paradigms, **supervised learning** and **unsupervised learning** are most often employed to tackle problems in the fields of computer vision and speech recognition (LeCun et al., 2015). Supervised learning requires a training dataset $\mathscr{D} = \mathscr{X} \times \mathscr{Y}$ composed of pairs of inputs and desired outputs. Given this set of points $\{\mathbf{x}, \mathbf{y}\} \in \mathscr{D}$, the neural network f will adjust its weights $\boldsymbol{\theta}$ in order to minimize the distance between the output $\hat{\mathbf{y}} = f(\mathbf{x}; \boldsymbol{\theta})$ and the desired solution $\mathbf{y}$. While supervised learning was empirically shown to be a very effective way to train networks, it relies heavily on the availability and quality of the training data, which poses challenges in practice (see Section 2.4). On the other end, unsupervised learning does not require explicit knowledge of the output space. While it offers the promise to solve issues tied to supervised learning, it is not often used in real world applications as designing an unsupervised training procedure is a difficult task and remains an active area of research (LeCun et al., 2015). In Section 4.3.3 we will see an example use case for unsupervised learning.

2.3.2 Error Back-Propagation

As explained in the previous section, networks rely on an error measure (or equivalently a performance measure) and try to minimize (maximize) it by adapting their parameters. This error measure is a scalar value computed by some loss function L (also called cost, error or objective function). L depends on the output of the network $\hat{\mathbf{y}} \in \mathscr{Y}$ and is chosen to be defined and differentiable in $\mathscr{Y}$.

The gradient of L with respect to $\hat{\mathbf{y}}$ is the vector of the first order partial derivatives. Partial derivatives inform us on how a small change in the input affects the output. The negative gradient of L with respect to $\hat{\mathbf{y}}$, noted $\nabla_{\hat{\mathbf{y}}} L$, indicates in the neighbourhood of $\hat{\mathbf{y}}$ the direction in which L decreases the fastest (Bertsekas, 1997). This means that the gradient holds local information about how $\hat{\mathbf{y}}$ should be changed in order to best decrease the loss function. Changing $\hat{\mathbf{y}}$ can only be done by changing the parameters $\boldsymbol{\theta}$ of the neural network, which means that the quantity of interest is really $\nabla_{\boldsymbol{\theta}} L$. The back-propagation algorithm (Rumelhart et al., 1985) provides a way to propagate the output gradient information backward trough the different layers of the network and to relate it to the gradients computed with respect to the parameters.

The main ingredient of the procedure is the chain rule of calculus. It is a formula that expresses the derivative of the composition of two functions. If f and g are real valued functions such that $y = g(x)$ and $z = f(y) = f(g(x))$, we have:

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \ . \tag{2.7}$$

This rule is generalizable to the multi-dimensional case, and when applied recursively it can propagate the derivatives through the nested composition of several functions. A complete derivation of the back-propagation algorithm can be found for instance in Nielsen (2015).

In practice, numerical libraries for deep learning implement both the mathematical operations needed to build layers as well as their analytical derivatives. In addition of building the main forward computational graph, they also build a second graph with the derivatives for computing the backward pass. This process is called symbolic differentiation (Bergstra et al., 2010).

2.3.3 Optimization

In the previous section, we explained that the gradients contained information useful to decrease the loss function. The purpose of an optimizer is to define rules on how to use those gradients in order to actually modify the values of the network's parameters. Presently, most optimizers are variants of the stochastic gradient descent (SGD) (Bottou, 1998). Gradient descent is a first-order iterative optimization algorithm designed to converge to a local minimum of a function. Given a training dataset $\mathscr{D} = \{\mathbf{d}^{(i)}\}_{i=1..n}$, containing n elements, the global loss function L is defined as the arithmetic mean value of the losses $L^{(i)}$ computed for every $\mathbf{d}^{(i)}$. As a consequence, the gradient with respect to the parameters is given by the following Equation:

$$\nabla_{\boldsymbol{\theta}} L = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L^{(i)} \ . \tag{2.8}$$

Given an initial state for the parameters (random initialization), the method updates them by the following rule:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} L \ , \tag{2.9}$$

where the **learning rate** γ is a parameter controlling the step size of the update. In practice, because n can be very large, we do not compute the complete mean gradient at every iterations but instead select a small random subset of examples, called a **batch**. Hence the stochastic nature of Equation 2.10.

$$\nabla_{\boldsymbol{\theta}} L \approx \frac{1}{m} \sum_{i=1}^{m} \nabla_{\boldsymbol{\theta}} L^{(i)}, m \ll n \ . \tag{2.10}$$

Additionally, most optimizers keep a memory of the gradients of the few previous updates and use them to rescale the learning rate and correct the update direction (e.g. Sutskever et al. (2013), Duchi et al. (2011)). Those modifications were shown to greatly improve the convergence in the presence of noise or when progressing in regions of the loss landscapes with small or large curvature.

The method is said to converge to a local minimum, as opposed to global, because the gradients only carry information about the close neighbourhood of the current position in the optimization landscape. Once converged to a minimum, gradients will be nearly null and parameters will not be updated any more. However, there is no guaranty that a smaller minimum does not exist elsewhere in the landscape since the problems are usually non-

convex. This is a well known limitation of gradient based learning, and understanding and finding better optimization procedures for neural networks is an active field of research (LeCun et al., 2015).

2.4 Practical Methodology

While neural networks are learning by adjusting their parameters on their own, human intervention is nevertheless essential to the process. If from time to time algorithmic developments happen to provide a noticeable improvement of the way networks perform (e.g. Srivastava et al. (2014), Ioffe & Szegedy (2015), He et al. (2016)), it is often observed in practice that the key to successfully apply deep learning resides in factors external to the algorithm itself (Goodfellow et al., 2016). In particular, advanced uses of neural networks require access to an ideal training database containing an exhaustive set of examples labelled with unambiguous and clearly quantifiable meta-information.

It should be noted that research and development efforts put into deep learning and artificial intelligence are considerable, and in only few years from now the fields might be radically different from what they presently are.

2.4.1 Preparing Data

Data are the backbone of artificial intelligence based solutions. For most practical applications, it is not enough to have solely access to the measurements themselves but one also need a variety of meta-informations describing the content as well as the context. As an example, the ImageNet database (Deng et al., 2009), which is one of the reference dataset to help solving object detection problems, currently contains more than 14 million images all manually annotated, sometimes at a pixel level. Preparing such data is usually considered the biggest obstacle and the most important step of deep learning. In geosciences, this stage is especially challenging since data interpretation usually requires expertise and is subject to fundamental limits that make the description of the data uncertain and partially subjective.

2.4.2 Measures of Performance

The viability of a neural network is often judged and communicated using a simple quantitative performance measure. It is common practice to split the training dataset into three parts, the training, validation and evaluation data. The training data is explicitly used as an input to the network to compute and minimize the loss function. The validation data is not directly seen by the network but implicitly influence the training procedure as the user will monitor the performance metric computed on them and optimize for the best architecture and hyper-parameters (see next section). Once the network is deemed trained, i.e. once training loss is low and the evaluation metric yields good results, the final performance is measured on the blind-test data. The performance metric is not necessary the same as the loss function. For instance, in most image classification tasks, the network outputs a distribution representing the probability of the input to belong to each class and the loss function employed is the cross-entropy with respect to the true probability (LeCun et al., 1989, Krizhevsky et al., 2012). On the other end, the performance metric can be the confusion matrix expressing the rate of true positives and detailing the possible types of mistakes between classes.

Data scientists describe the difference between the train performance and the blind-test performance as the **generalization** ability of a network, i.e. its capacity to perform well on new data it has never encountered before. This definition is convenient as it offers a clear quantitative measure that can be communicated and compared against. However, this definition can also be sometimes a source of surprise. When an domain expert tries a deep learning solution on his/hers own data, he/she might observe different performances than the one advertised. It is indeed important to remember that performance is measured with respect to the training dataset and that this choice of dataset can be completely arbitrary. In seismic interpretation for instance, many parameters such as the quality of the acquisition and processing or the geology of the sub-surface will affect the data. Deep learning offers only mild theoretical guaranties as to how a trained network will behave when seeing new data. If performances were judged excellent on synthetic tests, or on a single real dataset, it is not in itself a proof that the method will work on other field recorded data.

Another difficulty encountered when working with geoscientific data is that there is no abso-

lute ground truth. Performance and loss metrics are computed with respect to uncertain labels, which affect their significance. Reaching the global minimum of a loss function computed on such labels means that the network would also have learned to reproduce the noise and possible mistakes they contain.

2.4.3 Architecture, Hyper-Parameters and Regularization

It is important that the user has some degree of understanding of the data and problem at end. This knowledge will determine the choice of the general category of model. For data like seismic amplitudes with a spatio-temporal structure, CNNs are a good choice. The exact architecture, such as the number and type of layers or the number of units per layer needs to be defined. It is common practice to start from a baseline architecture by selecting one of the top performing model in the current literature. While it is commonly accepted that the deeper the networks are the better they perform, one are often limited in practice by both the size and quality of the training data as well as the memory constrains. A very deep architecture developed for well labelled 2D images might not be usable as is for 3D seismic data. If the number of parameters is too small, the network will not have enough capacity to express the full content of the data and will yield poor results. This is called *under-fitting*. On the other end, a network with a very large capacity might be prone to *over-fitting*. This happens when one observe a large drop of performance between the train and test data. In this case, rather than learning to extract meaningful features, a network is simply memorizing the training examples. To prevent this from happening, known strategies involve gathering more and better training data and employing some forms of regularization. Regularization enforces some additional constrains to the learning process, and if chosen adequately it helps to converge to a better solution. Convolutional layers are themselves a form a regularized layers since they are forcing the network to learn multi-scale, spatio-temporal relationships in the data. Other hyper-parameters, such as the choice of optimizer or the schedule of the learning rate, also need to be selected. As previously said, it is common to start from a known baseline model and to make progressive changes in order to better adapt to our own problem. Automatic hyper-parameter tuning methods act as a black-box optimizer searching for the best combination of parameters in a constraint space (Mockus et al., 1978). However,

besides being computationally expensive, those methods rely heavily on the use of a perfect quantifiable performance metric, which is again not trivially obtainable with geoscientific data. The alternative is to tune hyper-parameters by hand. The user will need to build an understanding of the effect of the different parameters on the training loss and validation metric. This process can be tedious, and good algorithms should be relatively robust to the exact state of the hyper-parameters.

Chapter 3 Transfer Learning and Knowledge Distillation for Seismic Interpretation

3.1 Résumé

Dans ce chapitre, nous considérons des étapes de l'interprétation qui nécessitent de parcourir et d'annoter de grandes quantités de données en 3D, et parfois même avant sommation. Bien qu'il existe aujourd'hui des algorithmes qui permettent d'effectuer ces tâches de façon automatique, l'intervention des experts est néanmoins nécessaire dans les régions géologiques les plus complexes. Nous nous intéressons donc ici à la possibilité d'utiliser l'apprentissage profond pour permettre à la machine de fournir de bons résultats y compris dans les cas les plus difficiles.

L'obstacle majeur à l'utilisation de l'apprentissage profond pour ces applications réside dans la nécessité de préparer les exemples nécessaires pour entraîner les modèles. Dans l'idéal, il faudrait avoir accès à une grande quantité de données, représentatives de la diversité de la géologie mondiale, et minutieusement traitées et interprétées par des géoscientifiques. Cependant, en pratique, il est extrêmement difficile d'accéder à un tel ensemble de données et nous proposons donc des façons alternatives d'approcher le problème.

En particulier, nous étudions le recours à l'apprentissage par transfert de la compréhension, ainsi qu'à la distillation des connaissances, pour entraîner des algorithmes tout en se passant de la nécessité d'effectuer la majeure partie du travail à la main. Dans une première application sur la détection des objets diffractants, nous utilisons la modalisation synthétique à partir de modèles physiques pour créer des données d'entraînement et nous montrons qu'un réseau d'apprentissage est ensuite capable d'utiliser ses connaissances pour pointer des données

réelles. Dans une seconde application, nous utilisons un algorithme classique de pointage des failles sismiques afin de fournir une version initiale des exemples d'entraînement. Bien que contenant des erreurs, nous montrons que notre réseau d'apprentissage est capable d'extraire les informations utiles au milieu du bruit et de progressivement améliorer les résultats jusqu'à clairement surpasser l'algorithme traditionnel. Enfin, dans une dernière application, nous explorons les possibilités offertes par l'apprentissage adverse pour aider à réduire les différences statistiques entre données synthétiques et données réelles.

3.2 Overview

In this chapter we consider seismic interpretation tasks that routinely require substantial efforts from engineers to be completed. Those tasks are time consuming because they involve to explore and annotate large 3D, sometimes prestack, datasets. While the community has developed advanced methods to partially automate those process, traditional data processing algorithms are not able to compete with human solutions in area of complex geology or low signal-to-noise ratio (Yilmaz, 2001, Bacon et al., 2003). Conversely, deep learning offers the promise to bridge this automation gap by reaching expert like performances even in the most complex cases. However, currently, the only known viable way to exploit fully the representation power of a deep network is to train it in a supervised manner with a diversified and well labelled dataset (LeCun et al., 2015). Unfortunately, preparing large scale training labels for seismic and other geoscientific data requires huge efforts. We therefore hereby investigate alternative approaches that enables to get results on real data without the need of an extensive manual preparation work. Particularly, we explore the possibilities offered by transfer learning and knowledge distillation. In the first approach, we employ synthetic modelling to create training data and train networks that are then evaluated on field recordings. In the second approach, we use as initial training labels the solution provided by a traditional automatic picker and aim at improving them.

One of the hurdle of deep learning lies in the difficulty to learn a general solution to a problem when the distribution of the training data is not representative enough of the diversity of the real world or when the available labels are scarce and noisy. Transfer learning aims at successfully reusing the knowledge gained by solving one problem to a different but related

(a) (b)

Figure 3.1 – Visualization of the inputs that preferably activate a convolutional kernel from (a) the first layer (b) the twelfth layer of GoogleNet (Szegedy et al., 2015) trained on ImageNet (Deng et al., 2009). Details for computing such images where first described by Erhan et al. (2009) and a modified version by Olah et al. (2017) was used in this example. The main idea is to set up an optimization problem that iteratively transforms an input image (starting from random noise) in order to maximize the activation of a chosen unit.

problem (Pratt, 1993, Bengio, 2012). It is for instance a common practice to pre-train an algorithm on a publicly available dataset and to perform fine-training on a different but semantically similar dataset. This approach relies on the fact that the features extracted by the pre-trained network are fairly universal and that they will also partially express the content of the new data (Yosinski et al., 2014). The benefit is that the fine-training step will require less training examples, reducing the need for manual preparation. Depending on the number of available labels in the new dataset as well as its similarity to the original, one may decide to fine tune either the entire network or only the last classification layer. A similar domain, called one-shot learning, aims at using prior knowledge to learn about new object categories from only one (or few) examples (Miller et al., 2000, Fei-Fei et al., 2006). Those methods make use of the well studied fact that complex signals can often be well described by a composition of a small number of simpler elements, as for instance in the field of wavelet transforms (Donoho, 1995, Mallat, 1999). A limitation of those approaches is that only the most basic feature detectors are well transferable across different data. In a CNN for instance, those basic detectors are learnt in the shallow layers. The deepest layers learn more complex and abstract representations tied to the training samples and are therefore less adequate to well represent different data (see Figure 3.1). Since the true potential of deep learning is reached when the deepest layers are well trained, it might be that transfer learning methods cannot bring state of the art performance on really complex problems, and that an extensive labelling of the new data would still be required. More recently, adversarial regularization has been proposed to

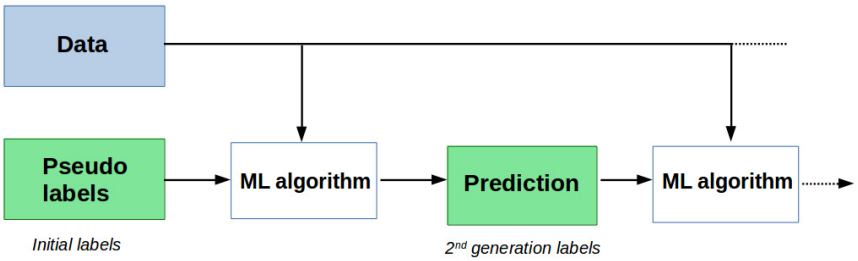


Figure 3.2 – Illustration of the distillation process. Initial labels are obtained by a given method and used to train a first machine learning (ML) algorithm. The trained model yields a prediction on the data that can be treated as new pseudo-labels. Those can serve to train a new ML algorithm, or fine-train the model previously used.

solve the domain adaptation problem in semantic segmentation problems (Liu & Tuzel, 2016, Hoffman et al., 2016, Tzeng et al., 2017).

Another strategy to train algorithms on new data is to use prior knowledge under the form of approximate labels. Such labels, sometimes called pseudo-labels or proxy-labels, are different in the sense that they where not obtained from the standard approach, i.e that they were not prepared manually by a human expert. They may for instance come from a network pre-trained on another dataset, or be computed by a specialized (non-learning) algorithm. Those pseudo-labels are treated as the ground truth and used to train a new network. If the pseudo-labels are good enough to start with, the network might be able to exploit the meaningful information they provide and discard the noisy content, improving the final results. This process is sometimes refereed as knowledge distillation (Lee, 2013, Hinton et al., 2015). Figure 3.2 illustrates several iterations through the distillation process. Mixing together pseudo-labels and manual labels is a common approach in semi-supervised learning (Chapelle et al., 2009).

3.3 Applications

In the following, we present three applications that make use of the concepts introduced in this chapter. In Section 3.3.1, we resort to numerical modelling to create synthetic training data an train a CNN to find small objects in the shallow sub-surface by identifying diffracted waves. In Section 3.3.2, we use the output of a traditional fault auto-picker to train a network , and we show that the method is able to improve on the results. Finally, in Section 3.3.3,

we explore how adversarial training may be employed to improve the transfer learning from synthetic to real data.

3.3.1 Picking Diffractions in Prestack Data

In the following we insert our publication about automatic diffraction picking (Tschannen et al., 2019). We use physical modelling to create a synthetic training dataset without the need of manual labelling and show that the method yields useful results on real data. A similar approach has been employed by Pham et al. (2019) and Wu et al. (2019) to pick channels and faults.

Geophysical Prospecting

EAGE EUROPEAN ASSOCIATION OF GEOSCIENTISTS & ENGINEERS

Geophysical Prospecting, 2020, **68**, 830–844 doi: 10.1111/1365-2478.12889

Detection of point scatterers using diffraction imaging and deep learning

Valentin Tschannen[1,2]*, Norman Ettrich[1], Matthias Delescluse[2] and Janis Keuper[1]

[1]*Fraunhofer Center for Machine Learning and Industrial Mathematics (ITWM), Competence Center for High Performance Computing, 67 663, Kaiserslautern, Germany, and* [2]*École Normale Supérieure, UMR 8538, PSL Research University, 75231, Paris, France*

Received March 2019, revision accepted October 2019

ABSTRACT

Diffracted waves carry high-resolution information that can help interpreting fine structural details at a scale smaller than the seismic wavelength. However, the diffraction energy tends to be weak compared to the reflected energy and is also sensitive to inaccuracies in the migration velocity, making the identification of its signal challenging. In this work, we present an innovative workflow to automatically detect scattering points in the migration dip angle domain using deep learning. By taking advantage of the different kinematic properties of reflected and diffracted waves, we separate the two types of signals by migrating the seismic amplitudes to dip angle gathers using prestack depth imaging in the local angle domain. Convolutional neural networks are a class of deep learning algorithms able to learn to extract spatial information about the data in order to identify its characteristics. They have now become the method of choice to solve supervised pattern recognition problems. In this work, we use wave equation modelling to create a large and diversified dataset of synthetic examples to train a network into identifying the probable position of scattering objects in the subsurface. After giving an intuitive introduction to diffraction imaging and deep learning and discussing some of the pitfalls of the methods, we evaluate the trained network on field data and demonstrate the validity and good generalization performance of our algorithm. We successfully identify with a high-accuracy and high-resolution diffraction points, including those which have a low signal to noise and reflection ratio. We also show how our method allows us to quickly scan through high dimensional data consisting of several versions of a dataset migrated with a range of velocities to overcome the strong effect of incorrect migration velocity on the diffraction signal.

Key words: Seismics, Imaging, Modelling, Signal processing.

INTRODUCTION

Most of the information obtained in exploration seismology comes from specular energy associated with reflections of acoustic and elastic waves at boundaries between geological layers. The resolution of the images is limited by the bandwidth of the seismic wavelet and, in particular, objects whose size is small compared to the dominant wavelength cannot be well resolved. Many small-scale structural details such as reflector discontinuities at fault planes, karst, pinchouts, channel edges, boulders or sand injectites may fall below the resolution power of reflection images and undermine the quality of the interpretation. Yet these objects will provoke the scattering of waves in all directions, a phenomenon called

*E-mail: valentin.tschannen@itwm.fraunhofer.de

diffraction. Due to their truncated Fresnel zone, diffracted waves contain a higher resolution signal and are valuable for an enhanced interpretation and inversion (Khaidukov, Landa and Moser 2004; Moser and Howard 2008; Landa, Fomel and Reshef 2008; Huang, Zhang and Schuster 2015). However, diffracted signal is not often available to the interpreter because its amplitude can be far weaker than reflected signal and is often lost during processing (Khaidukov *et al.* 2004).

It is possible to separate reflected signal from specular signal by taking advantage of the different kinematic properties of the waves interactions (Landa, Shtivelman and Gelchinsky 1987; Kanasewich and Phadke 1988). When encountering a locally planar surface, the energy will reflect in a focused direction depending on the angle of incidence. On the other hand, a diffractor will scatter the energy in all directions. In particular, this distinction will strongly affect the appearance of both signals in the migration-dip angle domain, as this angle is associated with the illumination direction. Using ray-based prestack depth migration in the local angle domain, we can form common image gathers (CIGs) in dependence on the migration dip angle. When computing the diffraction response at a migration point located exactly on a scattering object, given that the correct velocity model is used, the migration operator will align with the diffraction hyperbola and give rise to a flat response in the dip angle CIG. In contrast, when migrating a point located on a reflector, the operator will respond only at a dip corresponding to the structural inclination angle (Audebert *et al.* 2005; Landa *et al.* 2008; Reshef and Landa 2009; Klokov, Baina and Landa 2010; Klokov and Fomel 2012).

Several authors have suggested algorithms to detect scattering points from seismic data. Klokov and Fomel (2012) use a hybrid Radon transform to detect diffracted waves from time domain dip angle CIGs. Arora and Tsvankin (2017) show that discrimination in the dip angle domain is also possible in a transversely isotropic media. Shustak and Landa (2017) and Dafni and Symes (2017) form dip angle CIGs using reverse time migration, making their method applicable in complex geological areas exhibiting strong local variations in the velocities.

In order to detect the diffraction points, those approaches usually rely on filtering out the dips around the estimated specular dip before stacking followed by a visual inspection of the seismic image. This may be a time consuming and also challenging task due to the weak energy of the diffraction signal. For the same reasons, it is challenging to design a detection filter that will be reliable especially in areas of low signal to noise ratio. However, it has now become common practice to resort to supervised machine learning, a branch of artificial intelligence, to solve pattern recognition problems. Rather than requiring a hand crafting of the detection function, this class of algorithms is based on the use of a number of free parameters that will learn from examples. In particular, most of the attention is currently received by the field of deep learning carried by the so-called deep neural networks algorithms that have become the state of the art in solving a variety of recognition tasks (LeCun, Bengio and Hinton 2015). Several authors already applied deep learning to automatically detect structural features in seismic data. Waldeland and Solberg (2017) and Guitton (2018) employed a convolutional neural network (CNN) to, respectively, detect salt bodies and faults from a stack. Pham, Fomel and Dunlap (2018) used a pixel-wise CNN to locate channels networks. Regarding diffraction, de Figueiredo *et al.* (2013) suggested to use a machine learning based *k*-nearest neighbours classifier to detect diffracted signal from common offset gathers and applied their method on ground penetration radar data. Serfaty *et al.* (2017) suggested to distinguish diffracted events from other signals by working with compressed gathers in the local angle domain. After manually labelling a small number of seismic patches according to the dominant object they contain (reflector, fault, point diffractor, migration noise, random noise), they used a network pre-trained on natural images and retrained the last classification layer. The authors show that their method was successful in identifying diffracted waves in their dataset.

While deep neural networks are the best classifiers currently known, resorting to them comes with several difficulties. Probably the most challenging part is to create a training dataset. Not only do we need the seismic amplitudes but also the labels indicating the correct interpretation of those data. In practical applications on natural images, state-of-the-art results usually require to train the networks on millions of examples (LeCun *et al.* 2015). Creating such a dataset, especially in geoscience where expertise is required for the interpretation and where uncertainty and subjectivity will lead to non-unique answers, present a serious challenge. Additionally, the networks are made of many, sometimes millions, of parameters and a number of key hyper-parameters with complex non-linear dependencies. Finding the correct settings is both computationally and manually time consuming. To judge the quality of the obtained solution, authors usually evaluate performances with a blind test on a subset of the dataset that was not used during training. However, in the case where this data subset is statistically similar to the training data, a good test accuracy will not necessarily mean good generalization

performances (LeCun *et al.* 2015). In that case, the network might fail to produce an acceptable answer on a new dataset with a different geology or different processing.

In this work, we present an automated workflow that does not require manual picking of the diffracted waves. We use modelling to design a large synthetic dataset with low interpretation uncertainties and train a CNN from full dip angle gathers to perform high-resolution detection of scattering objects. In particular, we are aiming to identify diffracted signal emitted by point scatterers, that is objects which have a compact shape in all three spatial dimensions. Diffraction may also occur at edges, yielding a different pattern (Trorey 1970), that we do not treated in this work. After evaluation on synthetic data, we show that our network is able to identify diffracted events on a field-recorded dataset, yielding encouraging results as for its generalization capabilities. The first part of our paper gives the reader elements for an intuitive understanding of the diffraction imaging process and the training of a deep neural network. We then give practical details about the synthetic data creation and the training process. Finally we perform a case study on real data and discuss the current limitations of the method as well as further research opportunities.

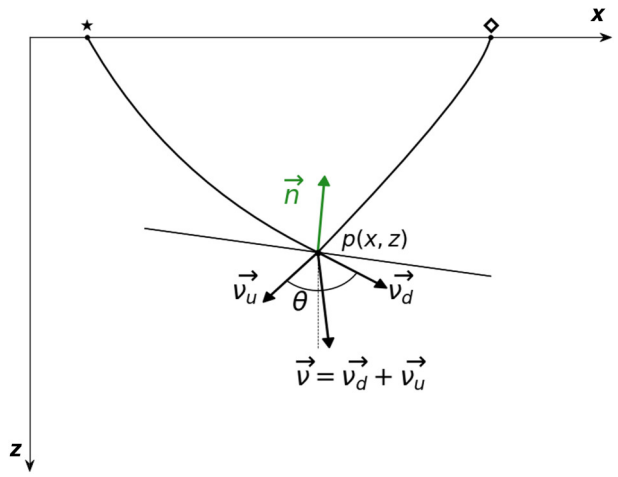


Figure 1 Two-dimensional representation of the local angle domain imaging geometry. A ray pair obtained by shooting from the migration point $p(x, z)$ and reaching a source/receiver pair is drawn. Vectors $\boldsymbol{v}_d$ and $\boldsymbol{v}_u$ are the tangents to the slowness vectors of the down- and up-going rays at p. The dip vector $\boldsymbol{v}$ is defined as the sum of those vectors, and migration dip ν is the angle between $\boldsymbol{v}$ and the vertical. The opening angle θ is the angle between $\boldsymbol{v}_d$ and $\boldsymbol{v}_u$. $\boldsymbol{n}$ is the normal to the locally planar geological reflector.

METHODS

Diffraction imaging

Prestack depth migration aims at mapping seismic events from the acquisition domain to their true position in the subsurface. This method requires the knowledge of an accurate spatially varying velocity field as well as a technique to simulate the propagation and back-propagation of waves. Under the assumption that the dominant wavelength of the seismic wavelet is much smaller than the scale of heterogeneities of the velocity model, we can replace the direct integration of the wave equation by a lighter ray tracing algorithm to compute the propagation. The algorithm we used in our study is working in the local angle domain, treating every node of the imaging grid as a point scatterer (Audebert *et al.* 2005; Moser and Howard 2008; Merten and Ettrich 2015). In the following, we explain the procedure in more detail without accounting for implementation performance specificities. In a two-dimensional (2D) acquisition, we can sort the data as common source gathers where seismic events are located by the source and receiver positions along the sailing line x and the two-way travel time t of the waves. We define our image space with a discrete grid parameterized by x and the depth z and build the migrated image iteratively at every node $p(x, z)$ of the subsurface model. We treat p as a scattering point and start by shooting a ray fan through the velocity model to form ray pairs that, respectively, reach existing source–receiver couples. Figure 1 shows the parameterization of the ray pairs in terms of the opening angle θ and the migration dip angle ν. Then, we compute the travel times along every ray pairs contributing to p (as well as an amplitude correction factor that accounts in particular for the geometrical spreading of the energy along the expanding wavefront) and fetch the corresponding samples in the shot gather. At this stage, for a given velocity field, the migrated data will be depending on four parameters: its position in space given by x and z and the local imaging angles ν and θ.

As the dip angle is directly associated with the illumination direction, reflected and diffracted events will exhibit distinct responses in the dip angle domain. By summing the data along opening angles, dip angle common image gathers (CIGs) are formed by constructive interferences (Landa *et al.* 2008; Klokov *et al.* 2010). Figure 2 illustrates the dip angle response of both a horizontal reflector and a scattering point. For the horizontal reflector, the zero-offset recorded wavefield will show a seismic event at a constant time for every position on the sailing line. Points p_0, p_1 and p_2 represent three migration nodes on a vertical line located at $x = x_k$. From their corresponding migration operators, we observe that nodes

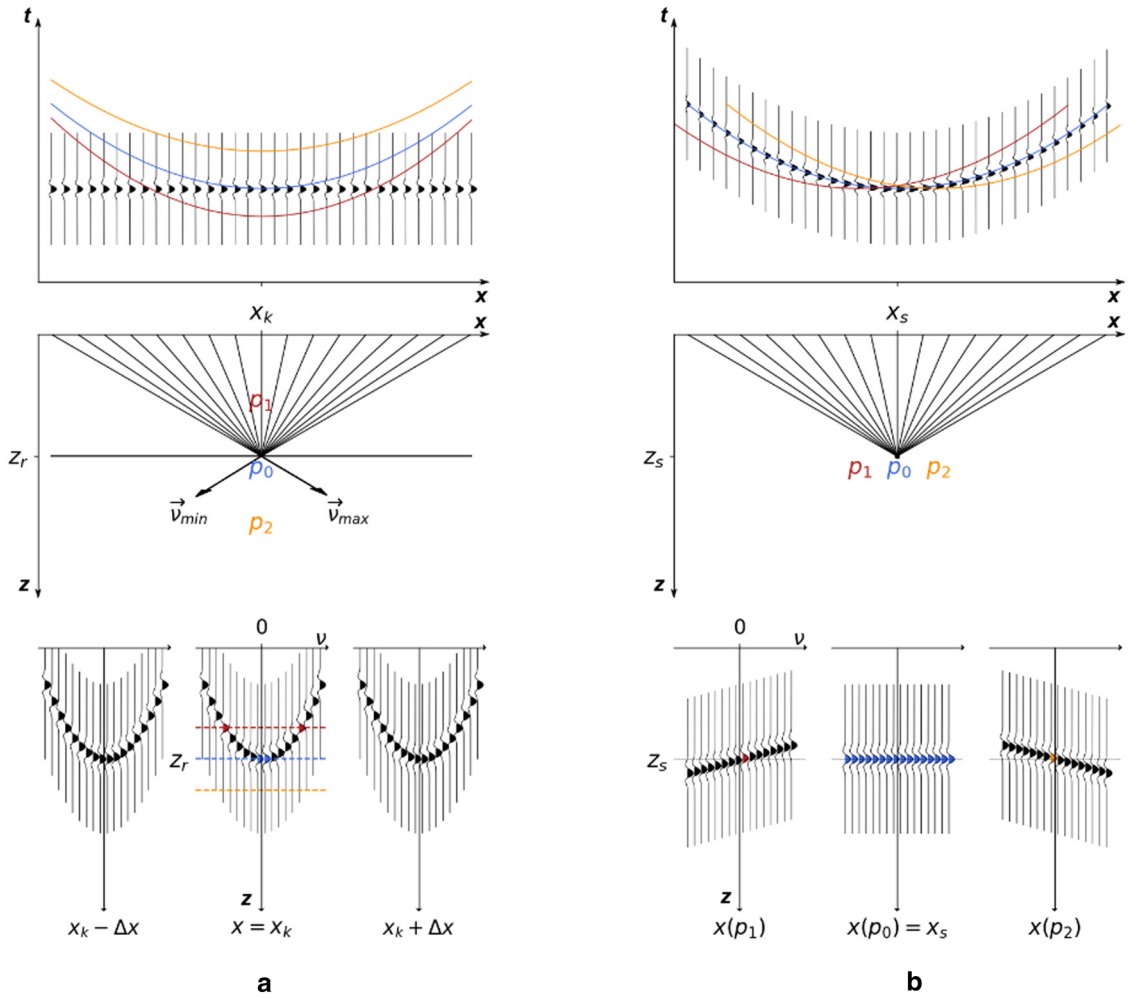


Figure 2 Two-dimensional illustration of the dip angle response, drawn for the zero-offset case, of (a) a horizontal planar reflector at depth $z = z_r$ separating two constant velocity half-spaces (b) a point scatterer in a constant velocity space located in (x_s, z_s). The central row represents the subsurface model. p_0, p_1 and p_2 are migration points. Ray pairs are plotted for p_0 as well as the corresponding minimum and maximum dip-vectors (see Fig. 1). The upper row shows the recorded wave field. Colour-coded diffraction curves of the three migration points are displayed on the seismograms. The bottom panel represents the migrated wavefield sorted as dip angle gathers. Coloured wiggles correspond to the amplitudes picked by the migration operators.

beneath p_0 will not contribute to the imaging of the reflector. The diffraction operator of the node p_0, which is located exactly on the reflector, will encounter the signal only at its apex corresponding to the zero dip angle. Migration operators of the nodes above p_0 will cross the wiggles at positions progressively further away from their apex and contribute to the imaging at progressively larger absolute dips, leading to a parabola-like shape of the signal in the gather. One should note that in the case of a tilted planar reflector, the apex of this pseudo-parabola will be at the dip equal to the plane's normal direction (vector $\boldsymbol{n}$ in Fig. 1). For the point scatterer of Figure 2, things are different as the waves will no longer reflect in a focused direction but will be scattered in all directions. The migration operator computed at p_0, positioned on the diffraction point, will match exactly the diffraction hyperbola of the recorded wavefield. Seismic events will be fetched at every dip angle, creating a flat horizontal response in the gather. When migrating a vertical line on the left of the scattering object, migration surfaces will cross the diffraction hyperbola at dips progressively increasing as the depth of the nodes decreases (and inversely for a line on the right-hand side of the diffractor), creating a flat tilted signature in the gathers. The slope of the response will increase as the lateral position of the line gets away from the diffractor. Similar observations hold

for non-zero offset data. Moreover, we see with this example that even for a short offset range large range dip gathers can be created.

The focusing quality of the method relies on having been able to derive an accurate velocity model. Figure 3 illustrates the sensitivity of the diffracted signal to errors in the velocity field. It is possible to use this strong sensitivity to perform high-resolution velocity analysis (Sava, Biondi and tgen 2005; Fomel, Landa and Taner 2007).

The relatively weak amplitudes of the diffracted waves combined with their strong sensitivity to accurate velocity models and pre-processing steps make their interpretation delicate. The interpreter will need to look through a large amount of prestack data, and potentially through several migrated versions of the same dataset obtained with different velocity models. Additionally, certain geological areas might be very rich in scattering objects, making their identification a time consuming and tedious task. For these reasons, it is preferable to employ an automated method, robust to low signal to noise ratio, and yielding a high-resolution detection. In the next section, we introduce the use of deep learning to solve such problem.

Semantic segmentation

Semantic segmentation deals with the problem of classifying every singular pixel of the data among a set of classes. This approach is a high-resolution extension of data classification, which aims at associating a class label to a group of pixels. Supervised deep learning has now become the method of choice to tackle those problems as it has proven to bring best performances on a broad range of applications (LeCun *et al.* 2015). The main technology behind deep learning is the so-called neural networks. These networks are built as a sequence of layers forming a non-linear, piece-wise differentiable function connecting the input to the output. Every layer is responsible for performing a simple affine transformation on its input and applies an element-wise non-linearity. The power of those networks resides in the way the parameters of the transforms are set. Rather than being manually engineered, the parameters are initially chosen at random and given the freedom to automatically adapt to the data by progressively learning from examples. Stacking several layers is a key to the success of those algorithms since this architectural design allows them to learn a hierarchical representation of the data. The deeper layers will benefit from the work of the previous layers and will be sensitive to progressively more abstract and complex features expressed as a composition of the simpler features learnt by the shallower layers (LeCun *et al.* 2015). Such networks can in theory approximate any function (Cybenko 1989). When the data exhibits a spatial structure and the surrounding information is relevant to understand the local context, a suited choice for the linear transformations is convolutions, and the family of algorithms based on them is called convolutional neural networks (CNNs) (LeCun *et al.* 1998).

In this work, we want to identify and localize diffraction points in the subsurface using seismic data. Learning to identify those elements consists in optimizing a neural network in order to approximate the distribution $\mathcal{D}$ over the domain $S = H \times P$, where H is the space of seismic data and P is the space of probabilities indicating the likelihood for the presence of diffraction points. Our training dataset consists of a collection of patches $d_1, d_2, \ldots, d_N \in S$ drawn from $\mathcal{D}$. In the 2D case, for a given $d \in S$, $d = \{\mathbf{h}^\alpha, \mathbf{p}\}$ is a tuple formed of a prestack seismic amplitudes patch $\mathbf{h}^\alpha(x, z, v)$ and its corresponding mask of probable locations of the scattering points $\mathbf{p}(x, z)$ (see Fig. 4 for an example). In deep learning, we refer to the different prestack sections as channels and call a single channel section a feature map. As an example, when working on natural images, the input data contains three channels, forming a coloured image as a composition of the red, green and blue feature maps. In our binary problem, either there is a diffraction point in this pixel or there is not, the mask $\mathbf{p}$ is defining at every spatial sample the probability mass function $(p, 1-p)$, where $0 \leq p \leq 1$ is the probability of having a scattering point.

Figure 4 is a schematic representation of the CNN architecture we use in this work. The network accepts as input prestack seismic data that it will progressively transform and reshape in order to output a patch matching the shape of the mask. It is composed of four types of layers in charge of performing convolutions, down-sampling, up-sampling and softmax scaling. The trainable parameters are the kernels (and biases) of the convolutional layers. We illustrate in Figure 5 how the first convolutional layer is working. A single input data sample $\mathbf{h}^\alpha = (h_1^\alpha, \ldots, h_{n_\alpha}^\alpha)$ is represented as the concatenation in the channel dimension of n_α feature maps. In our work, n_α is the number of migration dips v. The output of this layer $\mathbf{h}^\beta = (h_1^\beta, \ldots, h_{n_\beta}^\beta)$ is represented as the concatenation in the channel dimension of n_β feature maps. n_β is an architecture hyper-parameter and corresponds to the chosen number of convolution kernels of the first layer. Every single feature map of $\mathbf{h}^\beta$ is obtained by convolving the input data with a different kernel. Equation (1) expresses the exact operation performed:

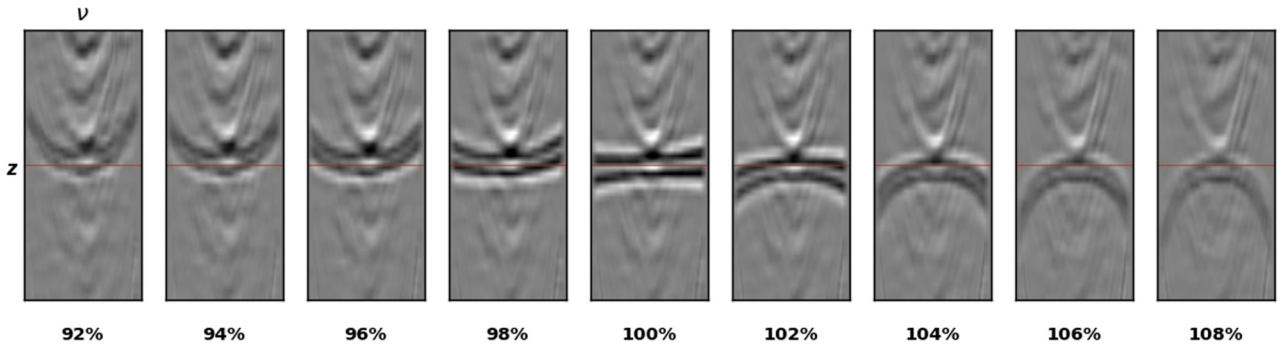


Figure 3 Sensitivity of the diffraction signal to errors in the migration velocity. Every image is a dip angle gather extracted at the same x coordinate located above a synthetic diffraction point. 100% corresponds to the true migration velocity, while remaining percentages correspond to relative perturbations from 2% to 8%.

$$h_i^\beta = \gamma\left(\sum_{k=1}^{n_\alpha} h_k^\alpha * w_k^i + b^i\right), \quad i = 1,..,n_\beta. \tag{1}$$

In the 2D case, the ith weight $\mathbf{w}^i = (w_1^i, \ldots, w_{n_\alpha}^i)$ is a prestack 2D kernel containing the same number of channels as the input layer. A 2D convolution $*$ is performed independently for every channel, and the results are then summed across channels. The ith bias term b^i is added after summation to enable the linear transformation performed by the convolution to be translated from the origin. An element-wise non-linear operator $\gamma(.)$, called the activation function, is applied to break the linearity between the layers in order to increase the approximation capabilities of the network. By repeating equation (1) with n_β weights $(\mathbf{w}^i)_{i=1,..,n_\beta}$ and concatenating the resulting feature maps, we create the new input data for the next layer.

In addition to convolutions, the network also performs spatial down-sampling and up-sampling of the feature maps. The down-sampling is achieved by sliding a small spatial window that selects the largest value and drops the

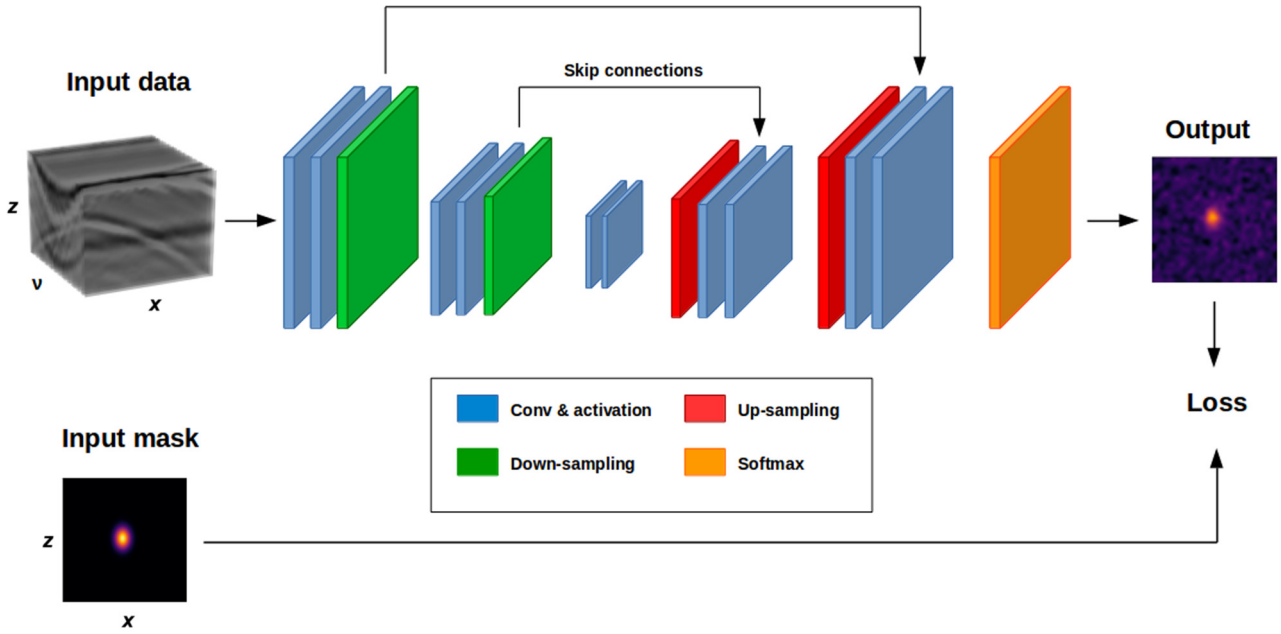


Figure 4 Architecture of our convolutional neural network (designed after Ronneberger, Fischer and Brox 2015). The data flow direction for the forward pass is represented by the black arrows. Input data $\mathbf{h}^\alpha(x, z, v)$ is a 2D prestack data patch and input mask $\mathbf{p}(x, z)$ a 2D patch matching the spatial dimension of the data. Boxes represent multi-channel feature maps colour coded by layer type. Output data $\mathbf{q}(x, z)$ has the same dimension as the input mask.

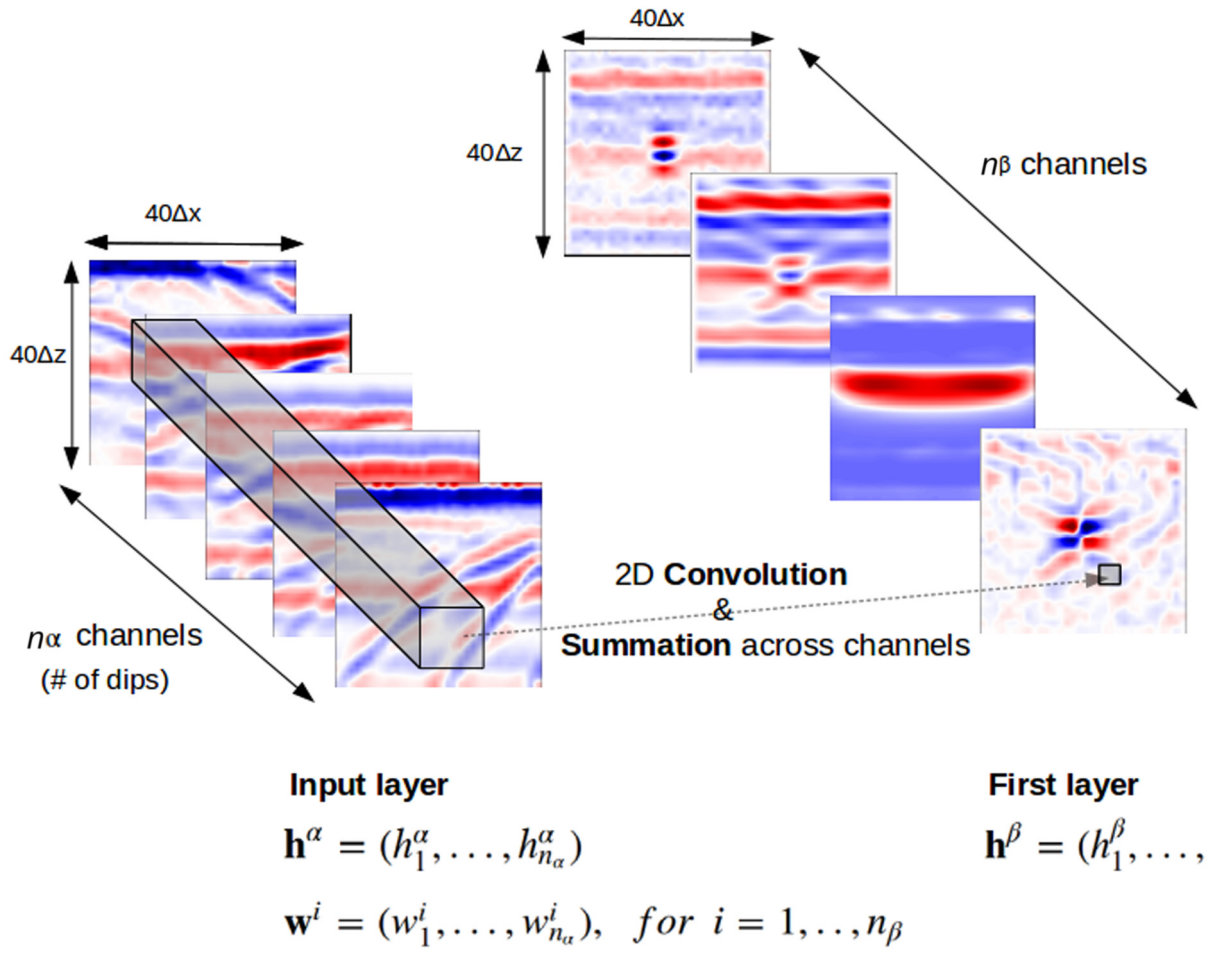


Figure 5 Principle of the first convolutional layer (in the 2D case, without bias adding and activation). The input data is a prestack 2D seismic patch $\mathbf{h}^{\alpha}(x, z, \nu)$ of shape $40\Delta x \times 40\Delta z$ with n_α channels corresponding to the number of dips. The kernels $\mathbf{w}^i$ have the same rank as the input data, and the 2D convolution is performed in the space (x, z) for every channel. An example of prestack kernel is overlay on the input seismic. After the convolution, a 2D output feature map is obtained by summing across channels. Every feature map h_i^{β} is obtained after convolving with a different kernel. The output data $\mathbf{h}^{\beta}$ is formed by the concatenation of the n_β feature maps.

remaining ones. While increasing the non-linearity of the network and forcing translation invariance, this operation also has the effect of expanding the receptive field of the convolutional kernels. By using a small constant spatial shape (e.g. 3×3) through every layer, down-sampling enables the kernels to progressively access a larger area of the data. This characteristic is important in order to learn a multi-scale representation, developing the abstraction power of the network. The up-sampling is the reverse operation and is used to progressively bring back the feature maps to their original spatial shape, which is a requirement since our network should perform a pixel-wise classification. A common technique for up-sampling is to learn the operation using strided transpose convolutions (Long, Shelhamer and Darrell 2014). A final architectural specificity is the use of skip connections (Fig. 4; Ronneberger, Fischer and Brox 2015) to reuse data from shallow layers in the deepest ones. Shallow feature maps are concatenated along the channels with their deeper counterpart of identical spatial size. This will allow the network to make use of both feature maps coming from early layers that contain information close to the original data and feature maps from later layers that contained highly transformed information. In order to map the output of the network to a pseudo-probability distribution expressing the classes diffraction and non-diffraction points, we design the output to be composed of two feature maps and scale every prestack pixels using a softmax layer (e.g. LeCun *et al.* 1998; Fig. 4). The output $\mathbf{q}(x, z)$ is defining at every spatial samples a mass function

$(q, 1-q)$, where $0 \leq q \leq 1$ indicates the confidence of the network in having found a diffraction point.

We called $\mathcal{D}$ the unknown *true* distribution that expresses the probability of having diffraction points in the seismic data, and let $\hat{\mathcal{D}}$ be the distribution computed by our network. Training a neural network consists in optimizing the values of its parameters $w \in \mathbb{R}^m$, where m is the number of free dimensions, in order to increase its prediction performances by bringing $\hat{\mathcal{D}}$ close to $\mathcal{D}$. For a given data point $d \in S$, we evaluate the quality of the prediction by the error measure $l(w; d)$ and define the training procedure as the minimization of the loss function over the finite dataset S:

$$\mathcal{L}(w; S) = \frac{1}{N} \sum_{i=1}^{N} l(w; d). \tag{2}$$

Given a training sample $d = \{\mathbf{h}^{\alpha}, \mathbf{p}\}$, we are concerned with minimizing the error of the network prediction $\mathbf{q}$. A standard pseudo-distance measure between two probability distributions is the cross-entropy (e.g. LeCun *et al.* 1998). It will measure how close is the computed distribution in representing the *true* distribution. In its binary form, the cross-entropy between $\mathbf{p}$ and $\mathbf{q}$ can be expressed as

$$l(w; d) = -p \log(q) - (1 - p) \log(1 - q). \tag{3}$$

Equation (3) shows that cross-entropy is differentiable and convex with respect to $\mathbf{q}$ (but not necessary convex with respect to w), and its minimum is reached at $q = p$. So far, the only computationally tractable way to minimize the loss function of equation (2) is to use a steepest descent algorithm (LeCun *et al.* 2015). Given a position in the optimization landscape for a parameterization state w_t, the method consists of finding the local downhill direction expressed by the negative gradient of the function computed at that point. A move towards a new point of the landscape is done by updating the parameters of the function in this direction, $w_{t+1} := w_t - \eta \nabla \mathcal{L}(w_t, S)$, where the gradient ∇ is the first-order vector derivative and η, called the learning rate, is the hyper-parameter defining the step size of the descent. Since the loss function directly depends only on the last layer of the network, we need to use the derivative chain rule in order to back-propagate the gradient to earlier layers (Werbos 1974). The procedure is repeated iteratively until convergence to a local minimum is obtained. In practice, it is not feasible to compute the gradients for every points of our training set at once, and we rather use a small random subset of the data, called batch, at every iteration. When every example has been seen once by the network, we say it was trained for one epoch. In addition, it is common to keep a moving average of past gradients and use it to influence the latest decent direction for better performances in the case of ill-conditioned loss landscapes (Rumelhart *et al.* 1988). Put together, this minimization procedure is called momentum stochastic gradient descent.

Coming up with the architecture and set of hyper-parameters that perform well on a given dataset can be a tedious task. Most of the field of deep learning is based on empirical findings, and the time needed to design a network is usually spent on hand tuning a number of parameters in order to increase the performances on the testing data. Moreover, the very high dimensionality of the optimization space combined with its non-convexity might provoke the convergence towards a *bad* local minimum. When this happens, one can achieve very good performances on a certain dataset but the network generalization capability will be poor and therefore lead to incorrect results when evaluated on new data with a non-trivially overlapping statistical distribution. In practice, it seems that to overcome these limitations, one needs to train the network with many, sometimes millions, labelled examples (LeCun *et al.* 2015). In the next section, we expose our strategy to create a training dataset using wave equation modelling.

RESULTS

Training on synthetics

Probably the most challenging part in designing a deep learning based application is not building the algorithm but rather preparing the data that will be used for training and evaluation, and most engineering-level applications of convolutional neural networks (CNNs) require to prepare a very large number of examples to produce robust and generalizable results (LeCun *et al.* 2015). While semantic segmentation offers a high-resolution interpretation of the data, it comes at the cost of having to prepare label masks. Such labels are difficult to get by since we need to annotate every pixel of the training data, and applications to seismic images usually require expertise in order to provide an acceptable interpretation. At this end, rather than manually labelling real data, we resort to synthetic modelling to create training and testing sets. Additionally, because of the inerrant uncertainties on geophysical data, manual labelling is prone to errors and subjectivity, while modelling allows us to use physics to control the procedure. As our approach is fully automated, we can cover for a wide range of velocity contrasts and source wavelets, in order to incorporate as much diversity in our training examples as

possible. In the next section, we will evaluate our network performance on high-resolution field-recorded data and we have chosen the modelling parameters accordingly.

We simulated fifty two-dimensional (2D) marine acquisitions with 250 Hz Ricker wavelets using a finite difference integration of the acoustic wave equation on a $\Delta x \times \Delta z = 0.5\text{ m} \times 0.5\text{ m}$ grid. Punctual high acoustic impedance perturbations are added to the layered velocity and density models to simulate the diffraction points. Our labels consist of binary masks indicating the position of the diffractors on the grid by a 1 and 0 elsewhere. In order to allow for uncertainties in the exact position of the scattering points, we convolved the masks with a normalized anisotropic 2D Gaussian (see label patch in Fig. 4). The central point of the Gaussian indicates the most likely position, and surrounding pixels show a progressive decay of the likelihood. After migrating the seismic to gathers with dips ranging from -40° to 40°, we created our training dataset by randomly extracting 200,000 prestack patches of shape $40\Delta x \times 40\Delta z$ with the corresponding masks (see Fig. 4 for an example). To augment the diversity of the training data, we also post-processed them with random band-pass frequency filtering and phase rotation. The architecture of our network is presented in Fig. 4. Every convolutional layer contains twenty-four 3×3 kernels. We trained with a momentum stochastic gradient descent optimizer for 30 epochs with a batch size of 48, using an initial learning rate of 10^{-3}. To regularize the training and try to avoid over-fitting, we perturbed the input patches with additive white noise and applied dropout (see, e.g., Ronneberger *et al.* 2015) and a decay factor of 50×37 every 10 epochs to the learning rate.

To control the quality of the training, we additionally created 10 synthetic datasets with a similar method. Figure 6 shows an example of the application of the trained network on this data. To count and localize the diffraction points found by the network, we run a filter on the predicted attribute to find every local maxima. Our parameters are set such that a local maximum should be detected only above a 0.5 confidence and two local maxima should be separated by at least 2 m. We compared the position of those maxima with those of the synthetic perturbations we added to the model and obtained a rate of a 100% true positives and no false negative. We also ran the network on a dataset imaged with a range of velocities to analyse the sensitivity of the prediction to errors in the migration model. Figure 7 illustrates that the algorithm is resilient to small errors in the velocities and is reaching its highest confidence for velocities close to the true one.

While the evaluation on synthetic data shows good performances, one should be careful before extrapolating and claiming that comparable performances will be achieved on any dataset. It is indeed well known that neural networks can easily over-fit the training data without learning to extract meaningful information. Then, if our blind test data are statistically similar to the train data, one is to expect our evaluation metric to yield good results. However, this is not a guaranty that we have solved our problem by creating a robust network that can generalize well. For instance, in this section, we are using synthetic validation data created with a similar approach than the training ones, which might not be enough for a thorough evaluation. A second case is when we use real examples labelled by an interpreter to train the network and evaluate the performances on the same data few hundreds of meters away. It might not be a guaranty that the evaluation metric will still be good if measured on a new dataset with different geology and processing. Another concern deals directly with the evaluation performance measure and training loss we use. Since the interpretation of our data is uncertain, it is unclear what the truth is and trying to match exactly, uncertain, and sometimes wrong labels might be a problem. In our case, it is difficult to know where the scatterers exactly stand in the subsurface and what is their exact spatial extent. In the next section, we judge the performances of our trained network on field-recorded data.

Field data evaluation

The data we use in this study is a 3.5-km line acquired in shallow waters with a high-resolution, shallow penetration source. It serves in a preliminary study to plan the construction of an offshore wind farm. Since the area is a former moraine, it is expected to contain small-scale debris, brought by a glacier, that need to be avoided while drilling.

Figure 8 shows the result given by the network trained on synthetic data. Since we do not know the true number and the location of the scattering objects, we cannot easily give a quantitative measure of the performance of the network. To assess the results, we investigated the data manually. Figures. 9 and 10 show examples of objects found by the network that we believe to be indeed boulders. In Fig. 11, we can see examples of misclassification. A shallow diffraction point with a noisy prestack response was not at all recognized by the machine, while a cross-shaped signal was, we believe, misclassified as a diffraction. Overall, we are satisfied with the rate of true positives. Most areas highlighted by the attribute seem to correspond to actual diffracted events. Estimating the number of false negatives is more difficult, but the dense coverage observed in Fig. 8

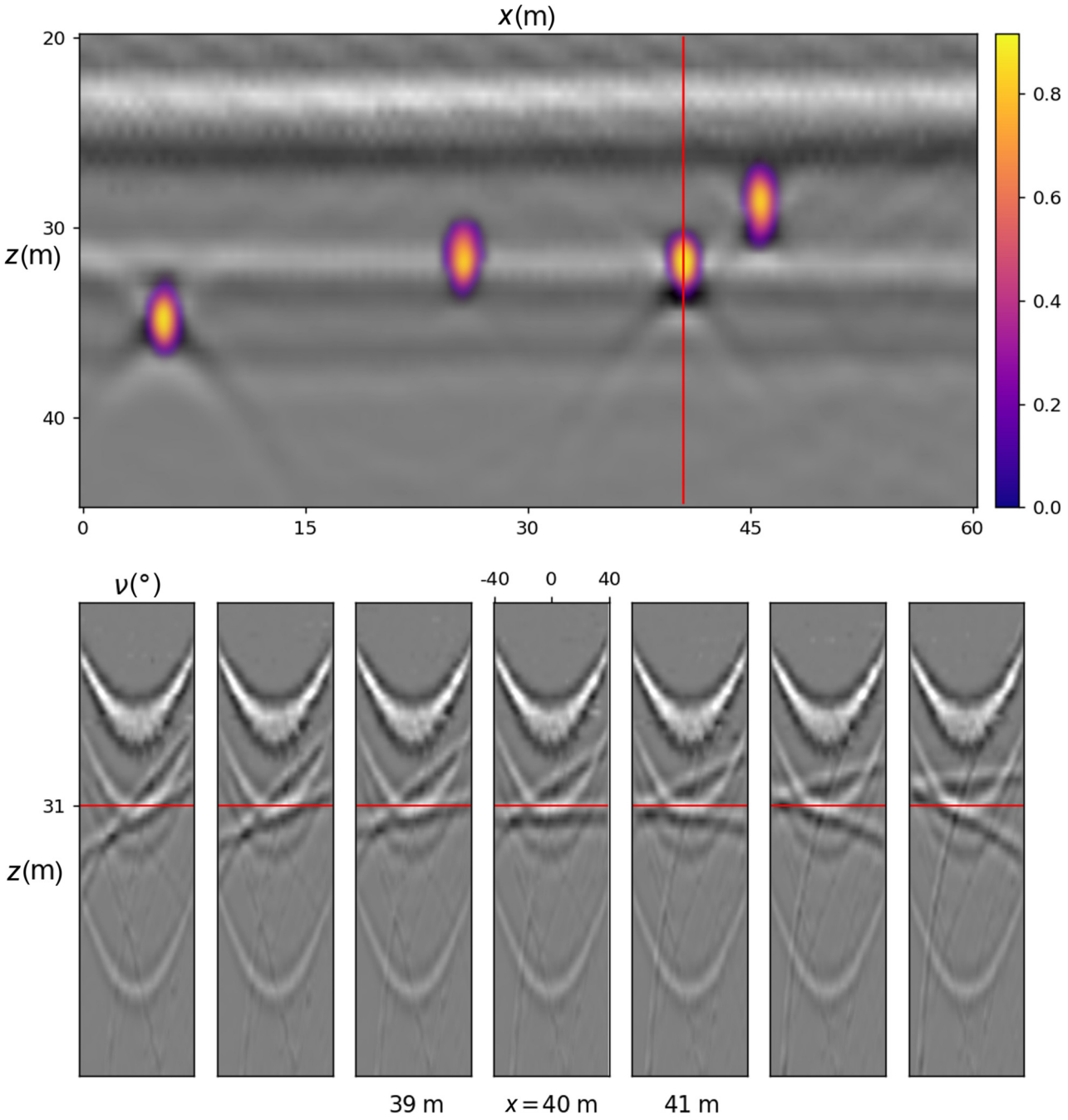


Figure 6 Prediction of the network on synthetic data that was not used for training. The machine confidence in having found a diffraction point is overlaid on the seismic stack. Likelihood inferior to 0.1 is set to be transparent. A vertical red line at $x = 40$ m passes through a diffractor and indicates the location of the central dip-gather displayed at the bottom. Remaining gathers are extracted every meter on the left and right sides. An horizontal line at $z = 31$ m highlights the signature of the scatterer.

gives us confidence that it found a majority of the scattering points.

We tried to incorporate as much diversity as possible in the synthetic data to cover a wide range of possible geologies, but they remain nevertheless a simplification of the reality. Using the confidence attribute, we selected the most probable diffracting objects in the field data and used them to fine-train the network. The results after such training did

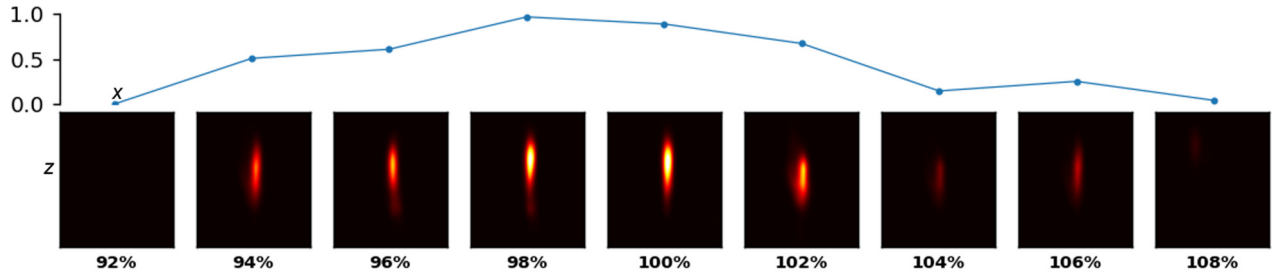


Figure 7 Prediction of the network in a 25 m × 25 m square around a synthetic point scatterer for data migrated with different velocity perturbations as presented in Figure 3. The maximum confidence scores are plotted above their corresponding patch.

not dramatically improve, but we believe that progressively extending our training dataset with real examples found by the network in different datasets will prove useful in further work to increase performances.

DISCUSSION

Mapping seismic amplitudes to the dip angle domain is a useful approach to help separate between diffracted and specular wavefields. Supervised deep learning has become the method of choice to automatically identify patterns in high dimensional data, and in particular convolutional neural networks (CNNs) are a suitable choice for seismic images as they learn, multi-scale, spatial relationships to support their decision. We proposed an automated workflow to detect point scatterers from prestack seismic data using deep learning. We created with wave equation modelling a large and diverse database of examples to feed to the network and showed that our algorithm could successfully transfer the knowledge it acquired from synthetic data to real data. The architecture of the network yields a pixel-wise classification enabling a high-resolution localization of the diffracting objects, while its probabilistic nature allows for some uncertainties in its answer. This method also lets us quickly scan through data migrated with different velocities to overcome the strong sensitivity of the diffraction images to velocity errors. Nevertheless, after carefully evaluating the results on field data, we found few false positives and false negatives and had difficulties to know how to better parameterize the algorithm to avoid those mistakes.

While deep neural networks can outperform every other method in classification tasks, they come with a number of disadvantages and difficulties. Deep learning is mostly empirical and works best when trained in a supervised fashion. It relies on the creation of a vast set of annotated data as well as on trials and errors to tune a large number of hyper-parameters. The datasets are usually prepared manually beforehand, and the performance is judged according to evaluation metrics computed on the training and validation sets. Those requirements are a challenge for applications in seismic interpretation. Because of inherent uncertainties in the data, the interpretation is often non-unique and subjective, and it also requires expertise in geology and geophysics. For this reason, as well as for the fact that most of the interpreted data is not publicly available, it is difficult to create large training databases. It also affects the reach of the evaluation metrics we use since they need to compare the answer of the network to non-perfect, and sometime non-existent, labels provided by interpreters.

To tackle the problem of creating a large labelled training dataset, we used synthetic modelling. This allows us to carefully control the subsurface model and provide an interpretation without manual work. However, synthetic data are a simplification of the reality and cannot account for all of the diversity and complexity that exists in nature. Other authors such as Serfaty *et al.* (2017) suggest to use a network pre-trained on publicly available datasets containing a very large number of annotated natural images. They then only need to label a small number of real seismic examples to fine-train the last layers of the network. This approach seems to work well but has few practical limitations. First, the geometry of the training data restricts the use of pre-trained nets to work on two-dimensional (2D) patches with three channels corresponding to the red, green and blue colour maps. This is a limiting factor for seismic data where structural objects are inherently three-dimensional and where full fold prestack data might bring more information as in this work or in automatic amplitude versus angle classification for instance. Furthermore, while it is understandable that shallow layers that learn to detect high-frequency characteristics such as edges are useful when transferred from natural images to seismic data, it is less intuitive for the deepest layers that have learnt abstract and large-scale concepts. Since the power of deep learning

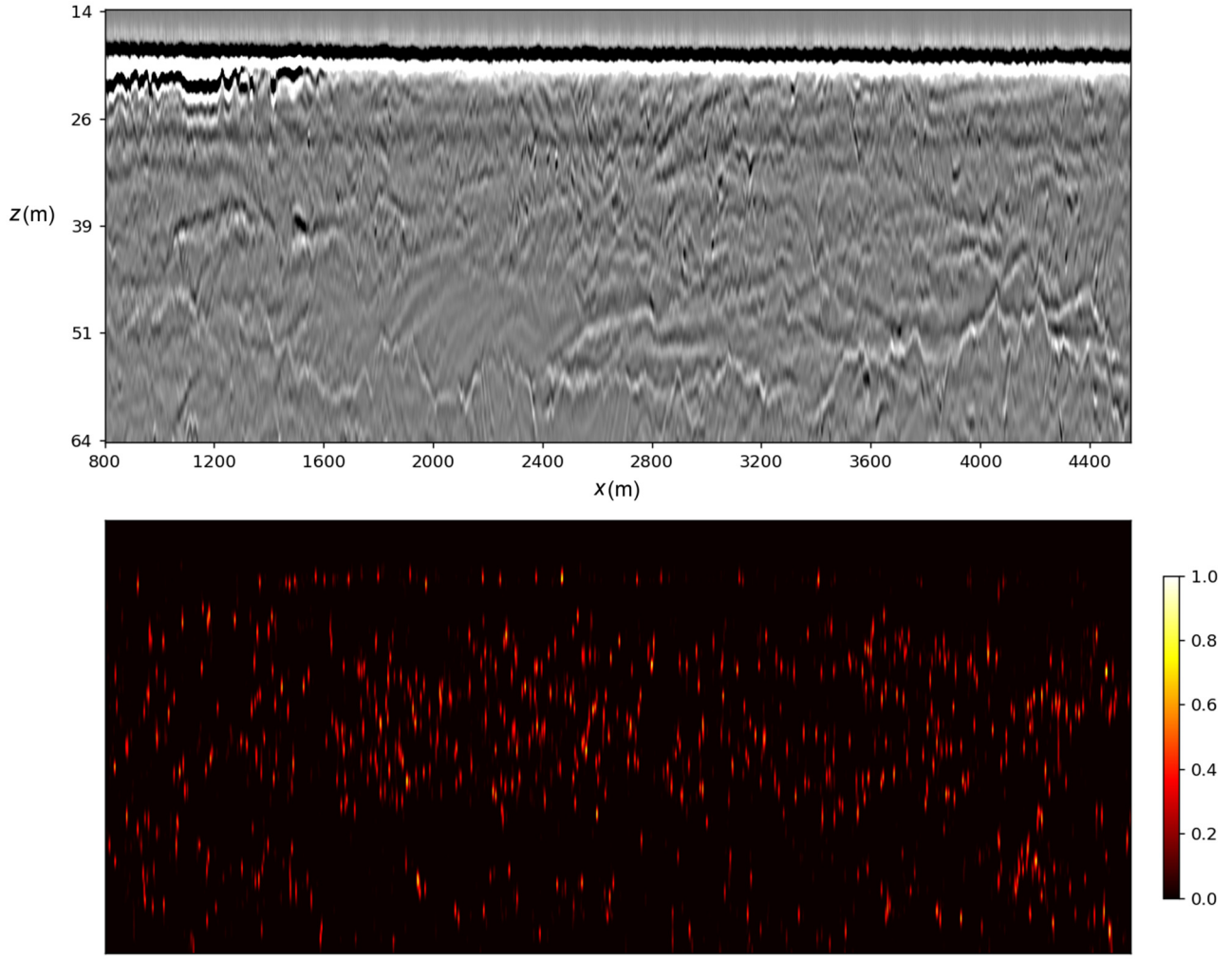


Figure 8 A full stack and the confidence attribute generated by the network. The local minima filter ran on the attribute counted 537 diffracting objects.

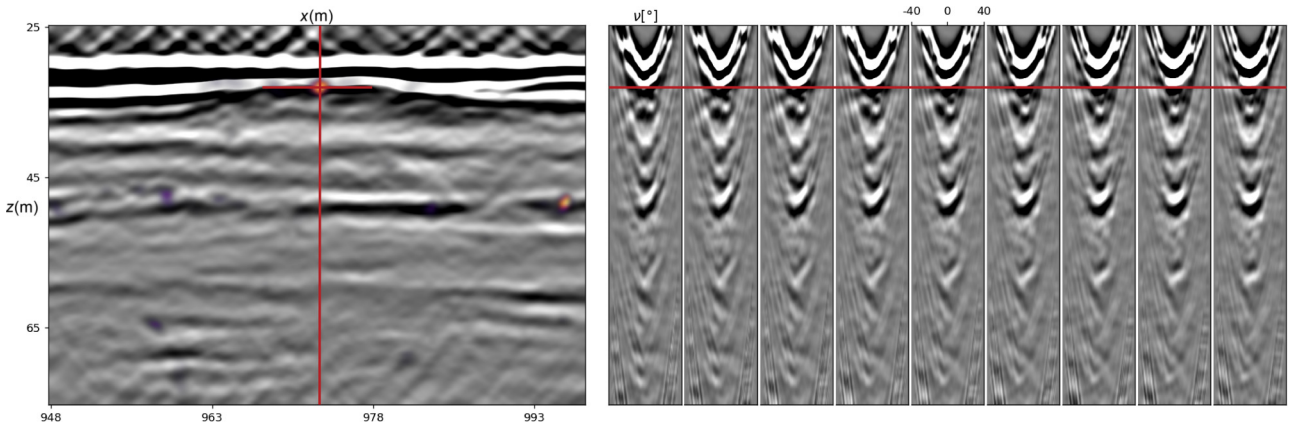


Figure 9 An example of true positive located just beneath the water bottom. The left image is a zoom on the stack with an overlay of the confidence attribute shown in Fig. 8. A vertical red line indicates the position of the central dip CIG displayed on the right side. Other gathers are displayed every 0.5 m around the central gather. The vertical red line highlights the signature of the boulder.

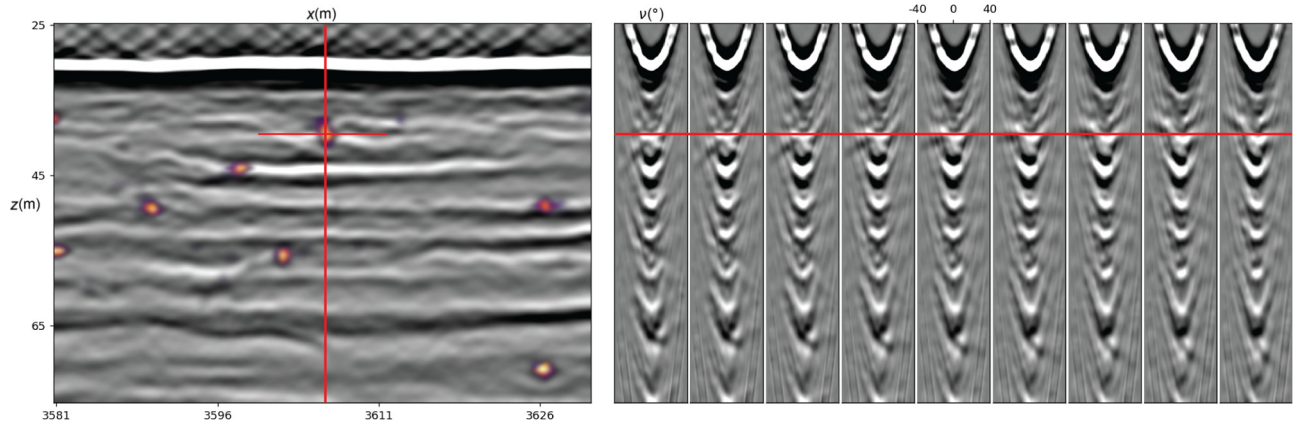


Figure 10 An example of true positive. Disposition of the figure is similar to Fig. 9.

comes from those abstract concepts learnt in the deeper layers of the network, it is unclear whether such pre-trained networks are really taking advantage of the full extent of deep learning and if traditional machine learning methods would not yield similar results. We believe that our approach using synthetic examples to train and progressively adding real examples as they are found by the network to fine train it is a good compromise to have full flexibility in the design in the absence of an ideal training database.

While it is common to judge the performance of a neural network with a numerical evaluation metric, we believe that such conclusion is more difficult to draw with geoscientific data. The labels we provide are often uncertain and sometimes wrong. In our work, for example, it is unrealistic to expect the network to know the exact position of the centre of the diffracting object as well as it exact spatial extent. Therefore, the minimum of the training loss function is probably not indicating the best possible parameterization of the network. Additionally, when evaluating our network on real data we observed a drop in performances compare to the blind evaluation on synthetic data. It is difficult to prove the generalization of the performances of a network. If the blind dataset we use for evaluation is too similar to the training data, a good evaluation score will not necessary extrapolate to all new data. Finally, we argue that providing an evaluation score itself is problematic when working with real data. Again, because of uncertainties and lack of perfect manual interpretations, it is not possible to know the *truth* and therefore to give a 0 to 1 score that is truly meaningful. We think that qualitative judgement by human experts of the machine's findings on field data, while subjective, is still required.

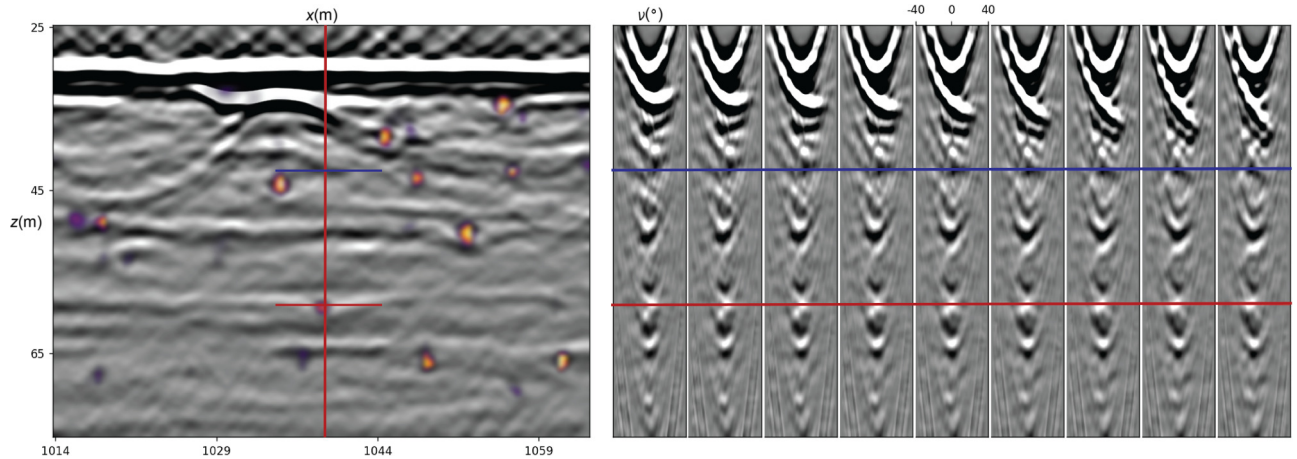


Figure 11 Examples of a false positive and a false negative. The disposition of the figure is similar to Fig. 9. The blue line indicates a probable diffraction that was not recognized by the network, while the red line highlights a cross-shaped signal wrongfully interpreted as a scatterer by the machine.

In further work, we hope to benefit from the creation of a dataset of real examples by progressively incorporating real findings from our method on a variety of field-recorded data. We plan to extend the method to be three dimensional. In theory, such extensions do not pose any problem, the migration will yield three-dimensional (3D) spatial data with 2D azimuthal dip gathers. The CNN will be working in the same fashion but with 3D convolutions and a fourth dimension being channels made of the concatenation of azimuthal migration dip angles. We also plan to improve the precision of the method to perform automated residual velocity analysis, using the results of the network as a misfit criteria to optimize a tomographic inversion.

CONCLUSION

We have introduced a new method to automatically identify scattering points from prestack data migrated using diffraction imaging. We built a database of dip angle common image gathers containing point scatterers using wave equation modelling and trained a convolutional neural network to compute a spatially varying attribute, indicating the machine's confidence in having found diffracting objects in the subsurface. The use of synthetic data was a key in order to provide a variety of examples with their interpretation at minimal manual labour cost. We showed that our trained network could successfully transfer its knowledge on field-recorded data and bring a valuable help to interpreters on an engineering task. Additionally, this automated workflow enables us to quickly scan through different versions of the same dataset to account for potential errors in the migration velocities.

We also discussed some of the challenges associated with the use of artificial intelligence–based algorithms to analyse seismic data. Uncertainties and non-uniqueness of the interpretation as well as the non-guaranty of generalization of the results should be taken into account when evaluating the performance of a network. In particular, we believe that careful inspection by experts, while subjective and qualitative, should be nevertheless carried on a reasonable variety of real datasets before concluding that the problem at hand was solved. We see a good potential in our workflow and hope to prove it valuable in further work for applications to other structural and amplitude related interpretation tasks.

ACKNOWLEDGEMENTS

This project was partially funded by the Federal Ministry for Economic Affairs and Energy of Germany, project 832642. We thank our colleagues from Fraunhofer IWES for preprocessing the marine data used in this study.

DATA AVAILABILITY STATEMENT

Synthetic data: Data available on request from the authors. Real data: Author elects to not share data.

ORCID


Valentin Tschannen
https://orcid.org/0000-0003-1845-5487

REFERENCES

Arora Y. and Tsvankin I. 2017. Analysis of diffractions in dip-angle gathers for transversely isotropic media. SEG Technical Program Expanded Abstracts 2017, 1011–1016.

Audebert F., Froidevaux P., Rakotoarisoa H. and SvayLucas J. 2005. Insights into migration in the angle domain. SEG Technical Program Expanded Abstracts 2002, 1188–1191.

Cybenko G. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **2**, 303–314.

Dafni R. and Symes W.W. 2017. Diffraction imaging by prestack reverse-time migration in the dip-angle domain. *Geophysical Prospecting* **65**, 295–316.

de Figueiredo J.J.S., Oliveira F., Esmi E., Freitas L., Schleicher J., Novais A., et al. 2013. Automatic detection and imaging of diffraction points using pattern recognition. *Geophysical Prospecting* **61**, 368–379.

Fomel S., Landa E. and Taner M.T. 2007. Poststack velocity analysis by separation and imaging of seismic diffractions. *Geophysics* **72**, U89–U94.

Guitton A. 2018. 3D convolutional neural networks for fault interpretation. Presented at the 80th EAGE Conference and Exhibition 2018.

Huang Y., Zhang D. and Schuster G.T. 2015. Tomographic resolution limits for diffraction imaging. *Interpretation* **3**, SF15–SF20.

Kanasewich E.R. and Phadke S.M. 1988. Imaging discontinuities on seismic sections. *Geophysics* **53**, 334–345.

Khaidukov V., Landa E. and Moser T.J. 2004. Diffraction imaging by focusing defocusing: an outlook on seismic superresolution. *Geophysics* **69**, 1478–1490.

Klokov A., Baina R. and Landa E. 2010. Separation and imaging of seismic diffractions in dip angle domain. 72nd EAGE Conference and Exhibition, Extended Abstracts.

Klokov A. and Fomel S. 2012. Separation and imaging of seismic diffractions using migrated dip-angle gathers. *Geophysics* **77**, S131–S143.

Landa E., Fomel S. and Reshef M. 2008. Separation, imaging, and velocity analysis of seismic diffractions using migrated dipangle

gathers. SEG Technical Program Expanded Abstracts 2008, 2176–2180.

Landa E., Shtivelman V. and Gelchinsky B. 1987. A method for detection of diffracted waves on common-offset sections. *Geophysical Prospecting* **35**, 359–373.

LeCun Y., Bengio Y. and Hinton G. 2015. Deep learning. *Nature* **521**, 436–444.

LeCun Y., Bottou L., Bengio Y. and Haffner P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **86**, 2278–2324.

Long J., Shelhamer E. and Darrell T. 2014. Fully convolutional networks for semantic segmentation. *CoRR* abs/1411.4038.

Merten D. and Ettrich N. 2015. GRT angle migration a 5D data mapping problem. 2015 International Conference on High Performance Computing Simulation (HPCS), 577–580.

Moser T. and Howard C. 2008. Diffraction imaging in depth. *Geophysical Prospecting* **56**, 627–641.

Pham N., Fomel S. and Dunlap D. 2018. Automatic channel detection using deep learning. SEG Technical Program, Expanded Abstracts, 2026–2030.

Reshef M. and Landa E. 2009. Post-stack velocity analysis in the dip-angle domain using diffractions. *Geophysical Prospecting* **57**, 811–821.

Ronneberger O., Fischer P. and Brox T. 2015. U-Net: convolutional networks for biomedical image segmentation. arXiv e-prints, arXiv:1505.04597.

Rumelhart D.E., Hinton G.E. and Williams R.J. 1988. Learning representations by back-propagating errors. *Cognitive Modeling* **5**, 1.

Sava P., Biondi B. and tgen J. 2005. Diffraction focusing migration velocity analysis. SEG Technical Program Expanded Abstracts 2004, 2395–2398.

Serfaty Y., Itan L., Chase D. and Koren Z. 2017. Wavefield separation via principle component analysis and deep learning in the local angle domain. SEG Technical Program Expanded Abstracts 2017, 991–995.

Shustak M. and Landa E. 2017. Time reversal based detection of subsurface scatterers. SEG Technical Program Expanded Abstracts 2017, 969–973.

Trorey A. 1970. A simple theory for seismic diffractions. *Geophysics* **35**, 762–784.

Waldeland A. and Solberg A. 2017. Salt classification using deep learning. Presented at the 79th EAGE Conference and Exhibition 2017.

Werbos P. 1974. *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, Cambridge, MA.

3.3.2 Highlighting Faults in a 3D Volume

Introduction

Extracting fault surfaces from a seismic volume is an important step in structural interpretation. Assuming a locally planar geometry, we can characterize a fault with few properties as illustrated in Figure 3.3. The *strike* angle represents the azimuthal orientation of the plane, the *dip* angle measures the inclination between the plane and the horizontal. The *throw* measures the vertical displacements up or down the dip and the *heave* measures the horizontal displacement perpendicular to the strike. Workflows to extract fault surfaces usually consist of several steps. One first needs to delineate the position of the faults before extracting fault surfaces and creating fault objects. In this work, we explore how deep learning can be employed to compute an improved fault delineation attribute without the need for manually picking the data.

Besides manual interpretation, several methods have been developed to highlight faults from 3D seismic images. Most methods are based on the observation that faults represent a discontinuity through lateral reflections, and aim to compute attributes which are sensitive to this property. Semblance (Marfurt et al., 1998) and coherency (Marfurt et al., 1999) are examples of attributes measuring continuity of the waveforms, while variance (Van Bemmel & Pepper, 2000) measures potential discontinuities in the reflections. However, those attributes alone are sensitive to noise and stratigraphic features, such a inclusions, which also corresponds to discontinuities (Hale, 2013). Improved worflows have been proposed by regularizing the attributes, using for instance smoothing along estimated fault strikes and dips. The resulting attribute is usually named the fault likelihood or the fault plane (Hale, 2013, Wu & Hale, 2016, Philit et al., 2019).

Recently, several authors have employed supervised machine learning to approach this problem (Guitton et al., 2017, Huang et al., 2017, Xiong et al., 2018, Zheng et al., 2019, Wu et al., 2019). While most authors require to manually pick a portion of the data to train the algorithm, some propose a more practical methodology. In particular, Guitton et al. (2017) use as initial labels a fault likelihood attribute (Hale, 2013) and train a non-linear support vector machine to reproduce it. They show that the classifier is able to learn the dominant signal of the attribute

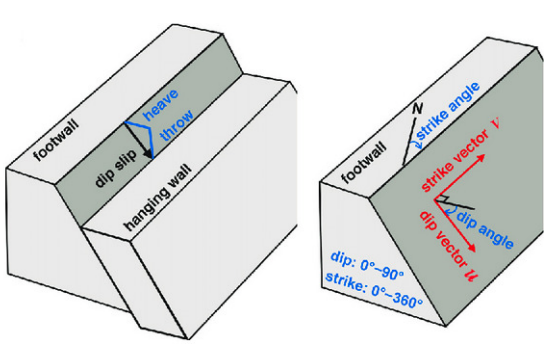


Figure 3.3 – Geometry of a fault plane. Taken from Wu & Hale (2016).

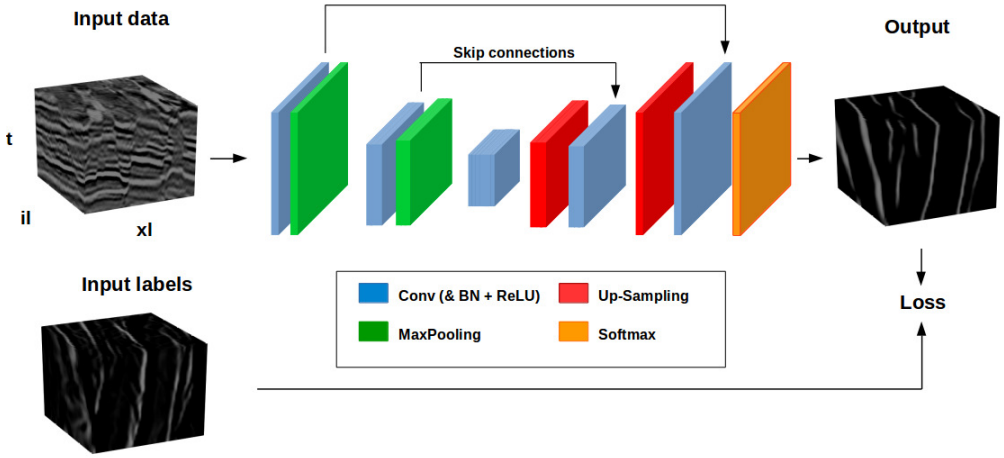


Figure 3.4 – Schematic representation of the CNN we use in this work (after Ronneberger et al. (2015)). The data flow is indicated by the black arrows. Blocks represent 4-dimensional layers outputs, color coded by operation types.

while discarding the less coherent signals, resulting in a better, less noisy, final answer. Wu et al. (2019) employ synthetic modelling to create a dataset of faulted seismic reflections and train a CNN. They show that while the training samples are synthetic, the CNN yields good results on real data. This approach is similar to the one we presented in Section 3.3.1.

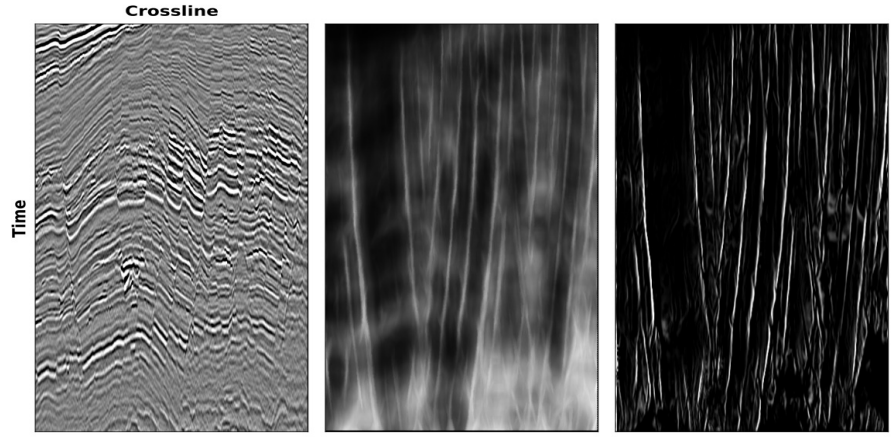
Method and Experiments

In this work, we approach the problem in a similar fashion as Guitton et al. (2017), but using deep learning. As opposed to machine learning methods, deep learning allows for an end-to-end workflow that does not require intermediate calculations such as the computing of an

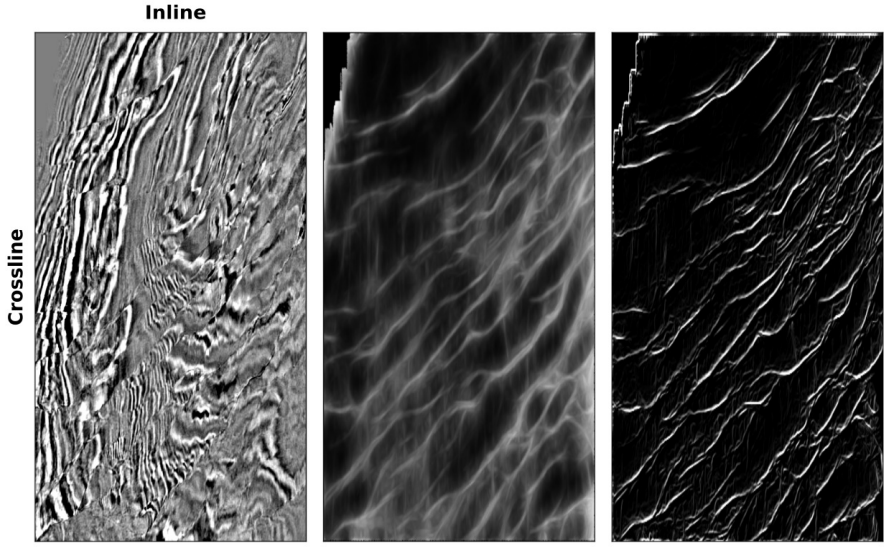
histogram of oriented gradients (see Section 1.2). Deep learning is also an appropriate choice to work with 3-dimensional data, as unlike traditional machine learning algorithms, it does not require to drastically reduce the size of the input data via preprocessing. We use as initial labels a fault plane attribute (Philit et al., 2019) to train a 3-dimensional CNN, and propose an iterative approach to progressively refine the predictions of the network. Figure 3.4 illustrates the method. The network takes as input seismic patches of $64\Delta il \times 64\Delta xl \times 64\Delta t$ (where the dimensions are expressed with respect to the sampling units along the inline, crossline and time directions). It yields a volume with the same shape as the input and is trained by maximizing the similarity between the output and the fault plane attribute.

In order to improve the quality of the network's prediction, we pre-process the initial labels to get a better definition of the faults positions. In particular, we take the power of 3 of the fault plane attribute and perform a thinning operation using a 2D *Sobel–Feldman* operator to compute the gradient along both the inline and crossline directions. The thinned attribute is then defined as the magnitude of positive gradients. In order to align the thinned image with the fault locations, we also shift the volume in the inline and crossline direction. The shift value is found as the *argmax* of the cross-correlation between the thinned and pre-thinned attributes. More advanced methods exist to thin the initial attribute (e.g. Philit et al. (2019)), but we resort to this simple worklow since it partially preserves the thickness, with a gradual increase and decrease of likelihood around the fault position. This results in a smoother learning landscape and provides the network with the notion of uncertainty. We show the result of the operation in Figure 3.5. The thinning we employ robustly preserves the original signal, but it does not highlight faults with a clearly continuous and noise free likelihood. We nonetheless show in the following that the network can learn from those labels and progressively improve the results.

We display the initial training results in Figure 3.6. The network's prediction is scaled to the range $0-1$ and can be interpreted as a confidence or likelihood for the position of the faults. We observe that the continuity of faults is better highlighted in the prediction and that it provides a less noisy answer. More interestingly, the network also seems to recognise faults which are not clearly present in the original fault plane attribute volume. This is for instance highlighted by the red ellipses of Figure 3.6. Despite being trained with partially wrong labels, the CNN is



(a)



(b)

Figure 3.5 – Preprocessing of the fault plane attribute. Examples for (a) an inline section (b) a time slice. The seismic data are displayed on the left. The central images are the original fault plane attribute (Philit et al., 2019). The images on the right show the same attribute after pre-processing.

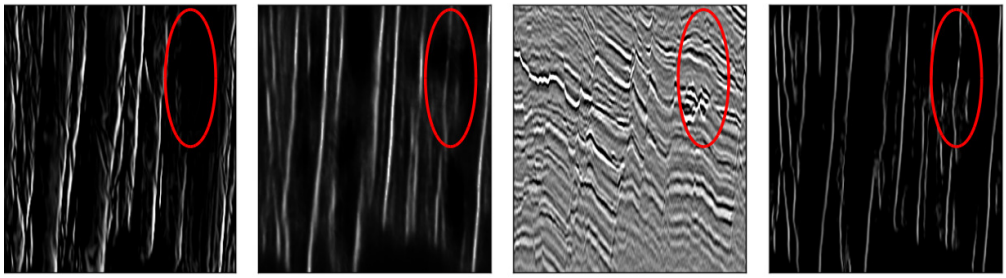
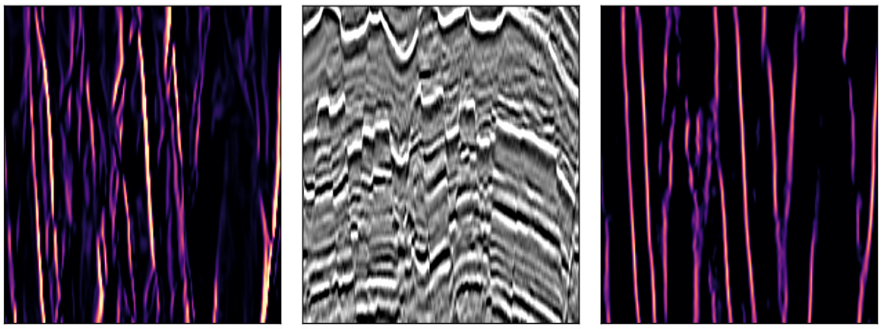


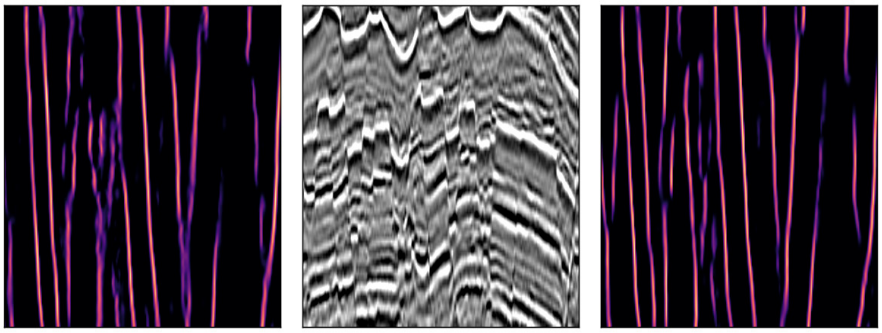
Figure 3.6 – Post-processing of the network's prediction. Example shown for an inline section. The first image from the left shows the original labels, the second image shows the network's prediction, the third image the data and the fourth the post-processed prediction.

robust enough to correct the interpretation in some cases by highlighting new faults, although with a smaller likelihood. From these observations, we propose an iterative approach in order to progressively improve on the results. We call the original fault plane attribute the first generation labels and the network's prediction, after training, the second generation labels. We can now repeat the training of a new CNN with the improved second generation labels. Since the quality of the training information is better, we expect to obtain a better performing network. Additionally, to increase the labels quality, we perform simple post-processing. To boost the likelihood of the newly found faults, we renormalise the prediction using an adaptive histogram equalization algorithm (Pizer et al., 1987, Stimper et al., 2019). We also apply the same thinning procedure as described above. The effect of the post-processing and successive training generations can be seen in Figures 3.6 and 3.7.

In the following, we present the results obtained after the 4^{th} generation training. Each generation requires about 18 minutes of computing time using the library Tensorflow (Abadi et al., 2015) and a Titan X GPU. Running a prediction on the entire input stack takes less than a minute and the overall algorithm runs for about 80 minutes. In Figure 3.8, we show an example of the transformations that are learned by the CNN to process the data. In Figure 3.9 we compare the results between the original thinned fault plane and the network's prediction. On a first order, the final answer is satisfactory as it yields a clean and thin attribute that highlights most major faults of the data.

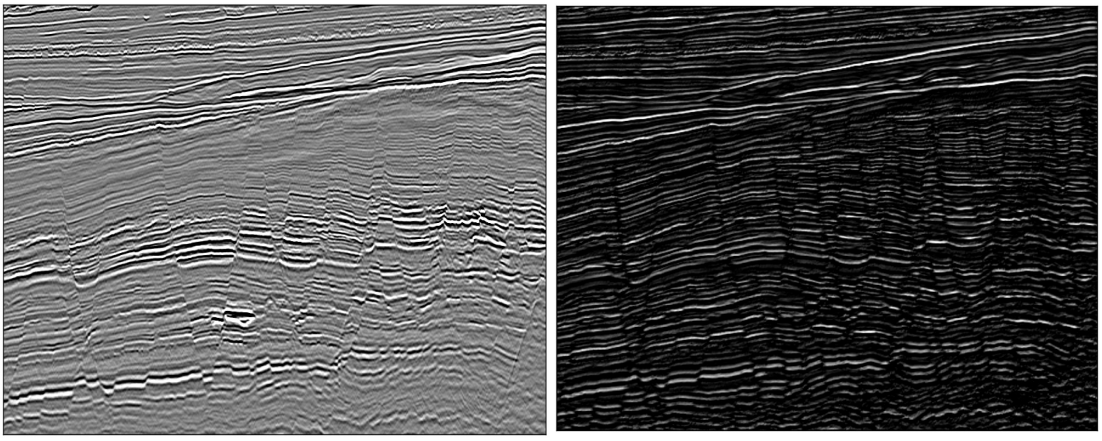


(a)



(b)

Figure 3.7 – Evolution of the network's prediction with successive generations. Lefts images show the input labels, central images the data and right images the predictions. (a) First training generation. (b) Second generation.



(a) (b)

Figure 3.8 – Visualization of a transformation performed by the network. (a) Input data. (b) Activation map for a convolution kernel of the first layer. Waveforms of the reflections are simplified on the transformed data, and faults are more apparent.

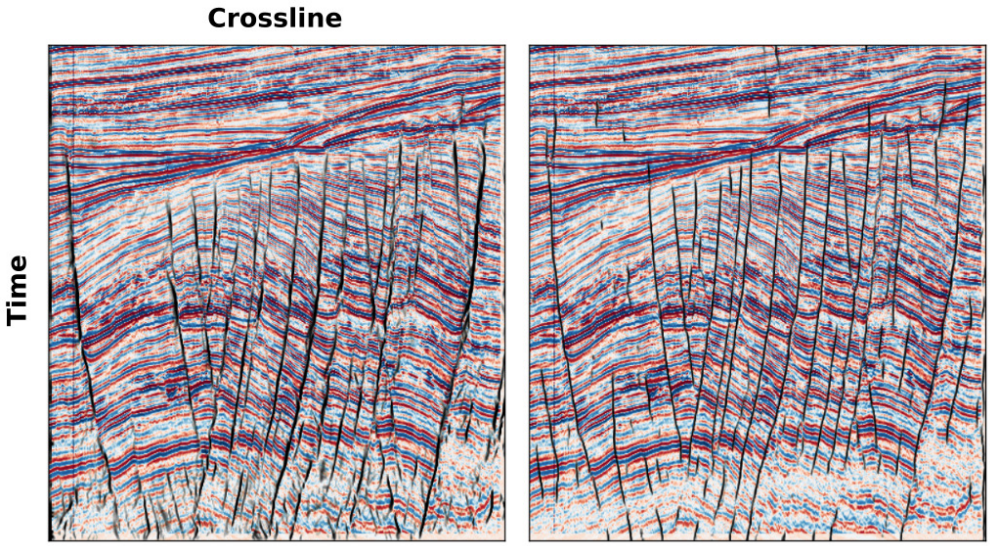


Figure 3.9 – Overlay of the fault attributes on top of an inline section. The original thinned fault plane is shown on the left and the network's prediction on the right.

Discussion

We see with this experiment that the CNN is a very good tool to clean the original fault plane attribute as it does not learn the noise initially present in the labels. False negatives, i.e. faults that are visible in the data but not found by the algorithm are more problematic. Using appropriate post-processing and several training generations, we show that we can nevertheless help the network to find new faults and better define the ones that were not clearly highlighted. We stopped at the $4th$ generation as training longer did not seem to improve the results any-more. Besides the fact that the training labels are partially wrong, other factors that explain why a network can find a fault but not an other (that seems very similar to a human interpreter) are not easily identifiable. The lack of explainability is one of the difficulties associated with deep learning. In order to highlight the remaining missing faults, a possibility is to pick some of them manually and fine-train the network with these new examples (we present this approach in Section 4.3.1).

In this work, we do not train the network to predict additional attributes like the local strike, dip or throw of the faults and these values can later be computed in an independent fashion. It should also be noted that the fault attributes our approach is relying on can only yield good results when the reflections exhibit a good lateral consistency and are dense in the

vertical direction. Picking fault networks in a chaotic area, or inferring fault planes through a thick package of homogeneous properties (and hence free of any clear reflections) remains a challenge for automated methods.

We show few additional results in Figure 3.10.

I am grateful to *Jean-Philippe Adam* and *Sébastien Lacaze* from *Eliis*[1] for providing the seismic data and the fault plane attribute used in this work.

3.3.3 GANs for Improved Synthetic to Real Transfer

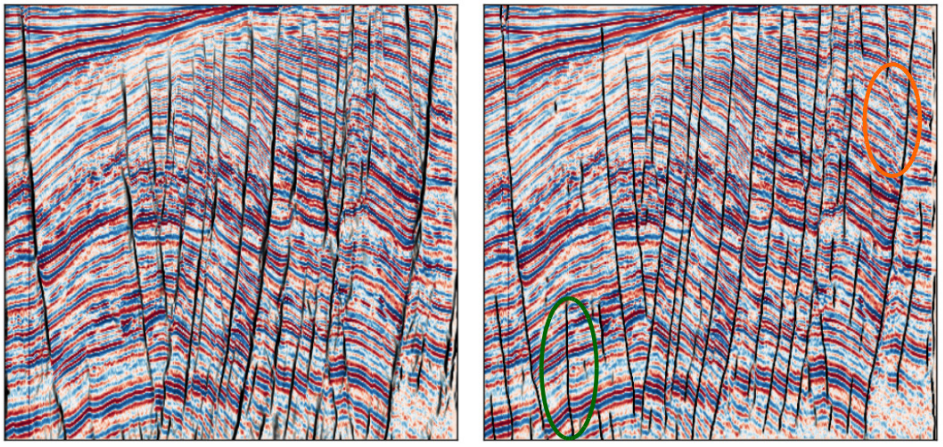
Introduction

This section presents a work in progress done in collaboration with *Ricard Durall-Lopez* from *Heidelberg University* and *Fraunhofer ITWM* and *Janis Keuper* from *Offenburg University*. We aim here to improve the quality of the transfer learning methodology proposed in Section 3.3.1 by trying to reduce the gap between the synthetic data and real data domains.

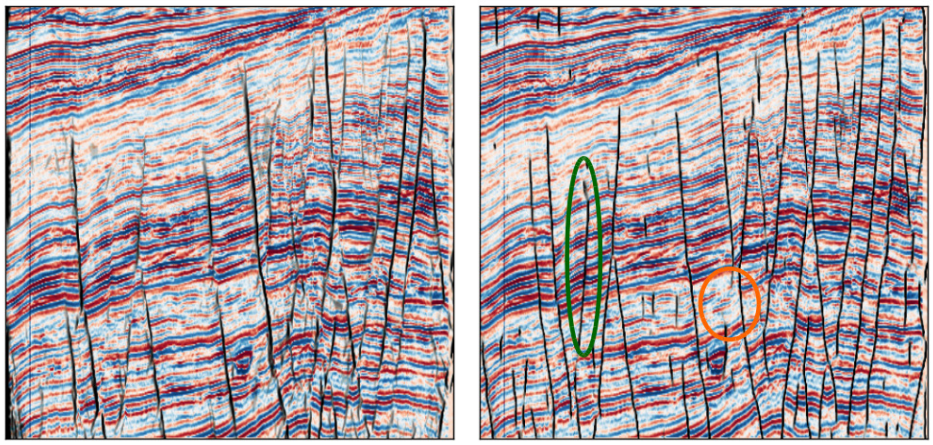
Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) are a class of deep learning models that became instantaneously popular for their relative simplicity and yet very powerful ability to learn data distributions. The main idea is to use two networks that compete against one another, as illustrated in Figure 3.11. Given two domains $\mathscr{A}$ and $\mathscr{B}$, the first network, called the generator $\mathscr{G}$, is responsible for learning an operation that transforms samples from domain $\mathscr{A}$ to a domain $\mathscr{A}'$ that approximates the target domain $\mathscr{B}$. The second network, called the discriminator $\mathscr{D}$, judges the quality of this transformation by trying to distinguish between transformed samples and native samples of the domain $\mathscr{B}$. If the training is successful, the generator has become so good at performing the transformation that even a well trained discriminator can no longer tell the difference between the target samples and the transformed one.

Several authors already employed GANs in a seismic workflow. They are for instance used for trace interpolation (Kaur et al., 2019), full-waveform inversion (Mosser et al., 2018) or forward modelling (Siahkoohi et al., 2019). In this work, we try to improve on the results obtained

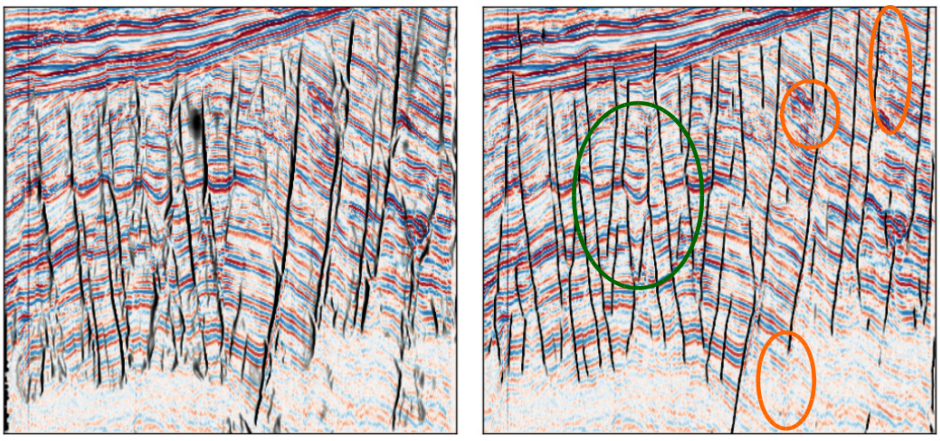
[1] http://www.eliis.fr/



(a)



(b)



(c)

Figure 3.10 – Additional results for some inline sections. The original thinned fault plane is shown on the left and the networ's prediction on the right. Green circles highlights area where the prediction is visibly improved, while orange circles show regions that will require some manual work.

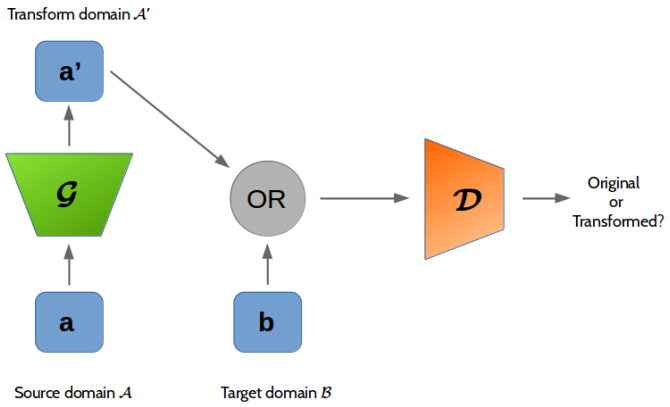


Figure 3.11 – Schematic representation of a GAN's setting (after Goodfellow et al. (2014)). The generator $\mathcal{G}$ is trained to transform data samples that can "fool" the discriminator $\mathcal{D}$. The discriminator is trained to distinguish original samples from transformed ones.

in Section 3.3.1 by creating better synthetic training data. The main idea is to use synthetic modelling to create diffraction signal and train a GAN to synthesize those diffractions into real seismic data. The benefit of this approach is to create more realistic training data, without the need of manually picking examples. This sort of methodology, named *attribute transfer*, is for instance described in Kim et al. (2017), Zhu et al. (2017), Choi et al. (2017).

Method

We employ two prestack datasets, a synthetic one that models seismic data both with and without diffractions, and a real dataset that is (almost entirely) free of diffracted waves. Each sample taken from those datasets come with two meta-information: whether the image is real or synthetic and whether it contains diffraction signal or not. The method is summarized in Figure 3.12. The generator is given two tasks, make the synthetic data to look more real and add a diffraction signal in the data if it's not present (or conversely remove it if it's present). It is the task of the discriminator to judge the quality of those transformations. We name $\mathcal{L}_{disc}$ the discriminative loss, i.e the feedback provided by the discriminator. In addition, we also resort to a cycle consistency loss (Zhu et al., 2017), $\mathcal{L}_{cycle}$, in order to regularize the generator and make sure that the transformed data still retains some of the characteristics of the input data. We name a the input sample and a' the sample transformed by the the generator. We calculate a third sample $\hat{a}$ by sending a' to the generator and define the cyclic loss to minimize

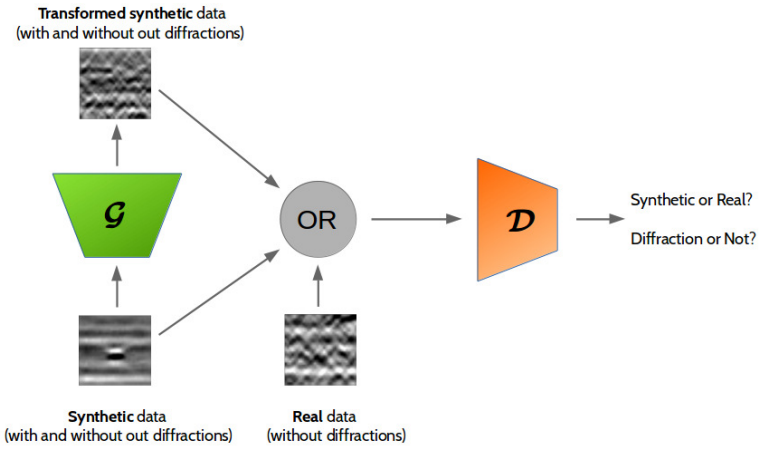


Figure 3.12 – Schematic representation of the method. The generator $\mathcal{G}$ is trained to make the synthetic data more realistic and to add or remove diffraction signal. The discriminator $\mathcal{D}$ is trained to distinguish real samples from transformed synthetics and to guess whether or not a diffraction is present in the transformed sample.

the distance between a and $\hat{a}$. Together, the generator loss writes:

$$\mathcal{L}_{gen} = \mathcal{L}_{disc} + \lambda_1 * \mathcal{L}_{cycle} \tag{3.1}$$

where the scalar λ_1 is a trade-off hyper-parameter. The discriminator must both determine if the data is real or synthetic and if it contains or not a diffraction. The first loss, $\mathcal{L}_{adv}$, is computed using the Wasserstein distance (Arjovsky et al., 2017) and the second loss, $\mathcal{L}_{class}$, with the cross-entropy distance. Together, the discriminator loss writes:

$$\mathcal{L}_{disc} = \mathcal{L}_{adv} + \lambda_2 * \mathcal{L}_{class}, \tag{3.2}$$

where the scalar λ_2 is a trade-off hyper-parameter.

Results and Discussion

Once trained, the generator can be used to insert diffracted waves into real data. Because, in every synthetic examples the diffraction was at the center of the patch, the generator learns to always insert it at this location. Therefore, we can construct a likelihood mask indicating the probable location of the scattering signal and use it to train a segmentation network as we explain in Section 3.3.1. We show initial results in Figure 3.13. A qualitative evaluation of the results suggests a good behaviour of the model. It shows the ability of generating seismic diffraction and combining them within the real images, without destroying

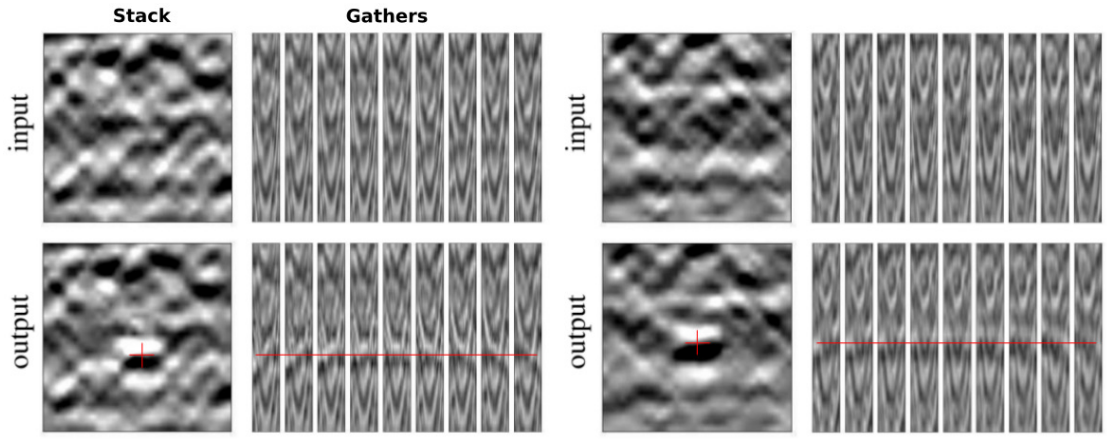


Figure 3.13 – Two examples of real data transformed by the generator in order to add diffraction signal. The data are shown in both the stacked domain (left) and prestack dip-angle domain (right, see Section 3.3.1). The red lines highlight the synthetic diffractions added by our model.

the surrounding reflected signal. The synthetic training data contained diffractions from point perturbations with different acoustic impedance contrasts and this variety in size and intensity of the diffracted waves is displayed by the generator as well.

This approach is promising, but many tests remain to be done. In particular, it is a problem that no perfect quantitative measure enables us to judge the results of the generator. Qualitative evaluation is tedious when working with a large amount of data, and also a bit subjective and uncertain. Overall, we recognize the usefulness of GANs to tackle seismic processing tasks. Adversarial training is a powerful idea that enables to create implicit metrics that could not be expressed well otherwise (such as a measure of how realistic a piece of synthetic seismic data looks like). However, the need to train several networks together as well as the difficulty to provide a clear quantitative evaluation of the final results are hassles that must be kept in mind.

CHAPTER 4 SEISMIC INTERPRETATION VIA INTERACTIVELY SUPERVISED AND UNSUPERVISED LEARNING

4.1 Résumé

Dans ce chapitre, nous considérons des cas qui nécessitent de progressivement affiner l'interprétation à mesure que les géoscientifiques acquièrent une plus grande compréhension de la géologie de la zone d'étude. En conséquence, le domaine d'apprentissage de la machine est lui aussi petit à petit modifié et les résultats fournis par l'algorithme se doivent de refléter ces changements.

L'apprentissage supervisé interactif est un cadre qui permet d'associer de façon efficace l'expertise humaine et la puissance de calcul de la machine. À l'aide d'un certain nombre d'exemples initiaux l'algorithme est entraîné et fournit des résultats préliminaires. L'interpréteur peut alors en contrôler la qualité et choisir d'apporter des corrections, sous forme de nouveaux exemples, afin de guider la machine dans les régions les plus complexes.

En apprentissage non supervisé, il n'est pas nécessaire de préparer des données d'entraînement, l'algorithme est capable d'apprendre seul à partir des données brutes. Cette approche est intéressante car elle évite la dépendance directe de la machine en intervention humaine et peut en théorie fournir des résultats qui sont dénués de biais humains et qui peuvent s'adapter à n'importe quelles nouvelles données. Cependant en pratique il est très difficile d'utiliser cette approche pour résoudre des tâches complexes. De plus, l'expertise humaine est toujours nécessaire afin de régler les différents paramètres de l'algorithme. En pratique, de façon similaire à l'apprentissage supervisé interactif, les experts modifies petit à petit les résultats de la machine en changeant la paramétrisation jusqu'à obtenir une solution jugée satisfaisante.

Dans la suite, nous présentons trois applications concrètes employant ces méthodologies. Dans une première partie nous avons recours à l'apprentissage profond supervisé de maniéré interactive afin de pointer des horizons dans des donnés sismiques 3D. Ensuite, nous utilisons l'apprentissage machine supervisé et non supervisé afin de faire ressortir les différent faciès lithologiques distinguables dans les données de puits et les courbes de variation de l'amplitude avec l'angle d'incidence. Enfin, nous explorons l'apprentissage profond non supervisé et établissions des axes de recherches intéressant à poursuivre dans ce domaine.

4.2 Overview

In the first part of this chapter we consider seismic interpretation tasks that require progressive refining as the experts get to understand better the area and have access to new information, coming for instance from well measurements or additional seismic surveys. As the knowledge and beliefs about the depositional system and the local tectonic history are evolving, geoscientists might change their interpretation and consequently change the learning space of the algorithm. Interactive supervised learning is a framework in which interpreters progressively guide the algorithm to converge toward a solution they trust. The machine will learn and update its parameters as it receives new examples, and the human experts can directly observe the effect on the results. They can then select areas where they disagree and try to correct the model by giving new inputs. Rather than replacing interpreters, this approach aims at augmenting the interpretation workflow by combining artificial and human intelligence (Amershi et al., 2014, Holzinger et al., 2019).

Many research domains, such as medical and biological image analysis, have already experienced with semi-automated algorithms where human experts are used to carefully constrain learning algorithms (e.g. Shyu et al. (1999), Sommer et al. (2011)). These approaches are especially suitable when one cares about explainable artificial intelligence (or at least trustworthy AI), i.e. when one can not be satisfied only from receiving an answer but where one also needs to have a reasonable understanding of how the algorithm got there. This need arises when working in fields where data come with uncertainties. Uncertainties are for instance present in seismic interpretation since the resolution of the data is not sufficient to uniquely resolve the problem and approximations of the physics of wave propagation are being made

during processing. Moreover, human inputs to the algorithm is also a convenient way to bring external knowledge that cannot be trivially streamed to the algorithm. A geologist, for instance, can make use of the knowledge from regional models in order to guide the interpretation in noisy or poorly resolved areas. Additionally, interactive supervised learning is also a good approach when one doesn't have a lot of labelled data to start with. An expert can influence the quality of the algorithm's prediction by carefully selecting few relevant new examples (Chapelle et al., 2009).

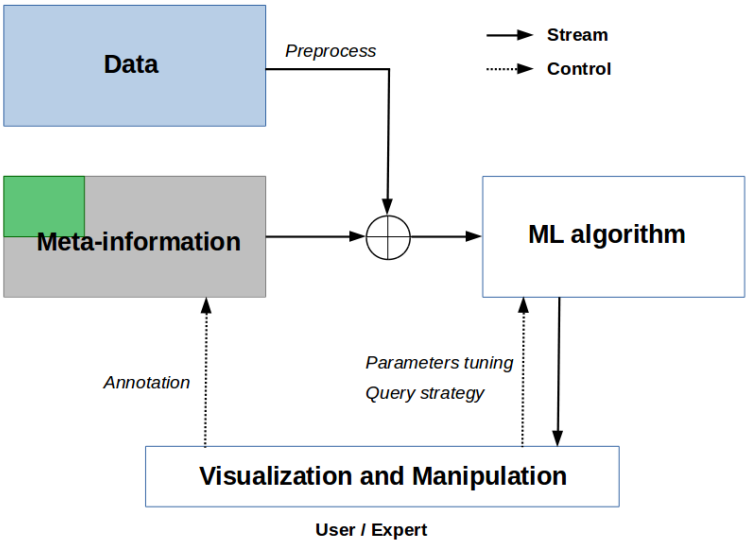


Figure 4.1 – Interactive learning framework. After a quality control (QC), the user can change hyper-parameters of the algorithm and annotate few additional data samples. The learning algorithm is automatically retraining with the new information and sends the new results to the visualization engine for the next iteration of human/machine interaction.

Several approaches are possible to perform interactive supervised learning. A summary of the different methods can be found in Chapelle et al. (2009). In Settles (2009), authors propose an active learning framework where the algorithm selects on its own the unlabelled regions where it would benefit the most from human input. Figure 4.1 presents a typical interactive supervised learning framework.

Unsupervised learning is another approach that aims to let the algorithm discover statistical patterns in the data without the help of external information provided by humans (Bishop, 2006). This approach is appealing as preparing training labels is often a difficult and very time consuming and constitutes a major difficulty to overcome when working with deep

learning. Interestingly, if we could train algorithms without the need for explicit supervision, one could study the results of the machine obtained (mostly) without the introduction of human biases. Additionally, networks trained in this fashion would suffer less from a lack of generalization abilities. Indeed, while supervised learning is limited to the amount of available labelled examples, with unsupervised learning one can in theory always learn to adapt to new observations. Unfortunately present paradigms to train machine learning models without supervision are far from approaching state of the art results obtained by supervised methods on complex problems (LeCun et al., 2015). But they can nonetheless prove useful to tackle smaller and simpler tasks. They can also be employed to pre-train models, that are then fine-trained by human experts (Erhan et al., 2010). Working with unsupervised learning is sometimes analogous to working with semi-supervised learning. While humans are not necessary to label the data, they need to parametrize the algorithm and design the workflow. Obtaining useful results often necessitate to modify hyper-parameters and observe the effect on the results. It is another way to interactively converge toward a useful answer.

4.3 Applications

4.3.1 Picking Horizons in a 3D Volume

In this section we insert the work done in Tschannen et al. (2020) on the picking of horizons in 3D seismic data.

GEOPHYSICS, VOL. 85, NO. 3 (MAY-JUNE 2020); P. N17–N26, 14 FIGS., 2 TABLES.
10.1190/GEO2019-0569.1

Extracting horizon surfaces from 3D seismic data using deep learning

Valentin Tschannen[1], Matthias Delescluse[2], Norman Ettrich[3], and Janis Keuper[4]

ABSTRACT

Extracting horizon surfaces from key reflections in a seismic image is an important step of the interpretation process. Interpreting a reflection surface in a geologically complex area is a difficult and time-consuming task, and it requires an understanding of the 3D subsurface geometry. Common methods to help automate the process are based on tracking waveforms in a local window around manual picks. Those approaches often fail when the wavelet character lacks lateral continuity or when reflections are truncated by faults. We have formulated horizon picking as a multiclass segmentation problem and solved it by supervised training of a 3D convolutional neural network. We design an efficient architecture to analyze the data over multiple scales while keeping memory and computational needs to a practical level. To allow for uncertainties in the exact location of the reflections, we use a probabilistic formulation to express the horizons position. By using a masked loss function, we give interpreters flexibility when picking the training data. Our method allows experts to interactively improve the results of the picking by fine training the network in the more complex areas. We also determine how our algorithm can be used to extend horizons to the prestack domain by following reflections across offsets planes, even in the presence of residual moveout. We validate our approach on two field data sets and show that it yields accurate results on nontrivial reflectivity while being trained from a workable amount of manually picked data. Initial training of the network takes approximately 1 h, and the fine training and prediction on a large seismic volume take a minute at most.

INTRODUCTION

A key step in seismic interpretation is the mapping of the main horizons in the amplitude volume. Horizons are reflection surfaces visible in the data that present a similar character in terms of wavelet shape throughout the survey. Mapping the reflections enables us to analyze the amplitudes to scan for potential fluid anomalies. Highlighting the horizons is also essential to understand how and when the observed geologic structures were formed. At later interpretation stages, surfaces are used to tie the seismic to well logs to relate seismic reflections to actual geologic interfaces and to perform a depth conversion for building geomodels. Dorn (1998) explains that working with full 3D data, rather than with sparse 2D lines, is essential when dealing with complex geology. Typical surveys contain hundreds of inlines and crosslines, which makes the manual interpretation of these surfaces a time-consuming task. For this reason, autotracking tools were developed to help interpreters (Bacon et al., 2003). Working with an autopicker is usually an iterative process in which the interpreter starts by dropping seed points on the desired reflection and gives some key information such as the waveform phase or the expected maximal vertical deviation between two adjacent traces. The tracker uses those hints to extract a 2D surface from the 3D data by finding related waveforms between traces using similarity measures. More seeds are progressively added, and the

Manuscript received by the Editor 23 August 2019; revised manuscript received 8 February 2020; published ahead of production 13 March 2020; published online 08 April 2020.
[1]Fraunhofer Center for Machine Learning and Industrial Mathematics (ITWM), Competence Center for High Performance Computing, Kaiserslautern, Germany and PSL Research University, École Normale Supérieure, UMR 8538, Paris, France. E-mail: valentin.tschannen@itwm.fraunhofer.de (corresponding author).
[2]PSL Research University, École Normale Supérieure, UMR 8538, Paris, France. E-mail: delescluse@geologie.ens.fr.
[3]Fraunhofer Center for Machine Learning and Industrial Mathematics (ITWM), Competence Center for High Performance Computing, Kaiserslautern, Germany. E-mail: norman.ettrich@itwm.fraunhofer.de.
[4]Offenburg University, Institute for Machine Learning and Analytics, Offenburg, Germany. E-mail: janis.keuper@hs-offenburg.de.

completion usually requires the interpreter to finish the most difficult portions manually. Although autopickers work well on high-amplitude reflections with a consistent waveform across the data, in complex structural areas, they usually fail to track across faults with a large throw or to follow weak and chaotic reflections. Many authors have proposed tracking algorithms that try to alleviate those limitations. Gersztenkorn and Marfurt (1999) and Lomask and Guitton (2007) suggest computing 3D structural attributes to highlight faults and stratigraphic elements not readily apparent in the seismic data. Aurnhammer and Toennies (2002) show how one could further constrain the autotracking by incorporating information of the main faults in the area. Alternatively, Pauget et al. (2009) and Wu and Fomel (2018) propose global approaches in which they treat the mapping as an optimization problem and solve it for all the reflections at once. In practice, local approaches often result in misties when the tracks from independent seeds are merged, and global approaches suffer from the curse of dimensionality and might require a lot of manual interventions to constrain the problem in regions containing nonstratified objects such as salt or gas chimneys and chaotic depositions (Hoyes and Cheret, 2011). In addition, traditional similarity measures, such as crosscorrelation, are sensitive to coherent noise in the data (e.g., acquisition and migration artifacts or interference from multiples) and perform poorly in regions with a low signal-to-noise ratio.

In recent years, there has been a big resurgence of interest around the field of deep learning, and in particular, convolutional neural networks (CNNs), to tackle computer vision and waveform analysis problems. Those methods have proven to outperform traditional signal processing and other machine-learning techniques on a large panel of applications (LeCun et al., 2015). Several authors use neural networks for tracking horizons (Harrigan et al., 1992; Kusuma and Fish, 1993; Veezhinathan et al., 1993; Alberts et al., 2000; Leggett et al., 2003), but modern computing power and the accumulation of empirical findings have permitted the emergence of deep neural networks that possess greater potential. Recent applications of deep learning to seismic interpretation include salt classification (Waldeland et al., 2018; Shi et al., 2019), fault detection (Huang et al., 2017; Xiong et al., 2018; Wu et al., 2019; Zheng et al., 2019), diffraction picking (Tschannen et al., 2019), and seismic lithofacies classification (Liu et al., 2019).

Horizon picking is a typical pattern recognition problem, and deep learning is therefore a logical choice to approach it. Peters et al. (2019) use a 2D CNN to track horizons using only a few manual picks to train the algorithm and show that the network can accurately predict the position of the surfaces in field data. In this work, we use a similar approach, but we propose a more in-depth study of the problem and aim to provide a practical methodology for interpreters. We work with a 3D CNN and propose two detailed and challenging case studies, in which we identify the strengths and limitations associated with the use of neural networks to pick horizons. We present a practical and robust workflow to segment several horizons at once in a large seismic volume. Our method does not require any special assumption on the character of the seismic wavelet nor on the spatial continuity of the horizons. We design a 3D-CNN architecture inspired by Ronneberger et al. (2015), which processes data over multiple scales and yields a high-resolution prediction. Because we treat the spatial and temporal dimensions differently, our network provides a stable interpretation while keeping the memory and computational requirements to a workable level. We express the initial manual picks provided by the interpreter as probabilities, and we use the cross-entropy loss to train the network in a supervised manner. The probabilistic formulation allows for uncertainties in the exact location of the predicted horizons. Given a simple masking of the loss function, interpreters have the freedom to label the training data using either seed points or 1D- and 2D-line interpretations. By keeping the total number of free parameters of the CNN small and by using regularization and data augmentation, we show that our method requires only reasonable manual work to prepare the training data. The initial network training time is approximately 1 h, and the prediction on a large seismic survey takes only seconds. We also show how the interpreter can interactively fine train the network by picking a few additional examples in the most complex areas. Finally, we show that our method can be used to extend horizons to the prestack domain to perform higher-quality amplitude analysis. We verify the validity of our approach on two marine data sets that exhibit challenges because of the presence of faults and weak reflections.

METHODS

Semantic segmentation

After prestack time (depth) migration, seismic data are represented as a multidimensional array over a regular grid of inline, crossline, and time (or depth) coordinates. Every element of the array is called a voxel and holds the value of the wavefield amplitude at this position. In the image processing community, given a set of preestablished categories or labels, classification refers to the association of an image with one label. Segmentation goes beyond classification and refers to the association of every pixel of an image with a category. By analogy, with seismic data, we refer to one sample as a patch (i.e., a small cube) extracted from the global volume. Classification aims to associate one class to the entire sample. This is, for instance, the approach taken by Waldeland et al. (2018) to pick salt bodies. To obtain the final prediction on the entire seismic data, Waldeland et al. (2018) assume that the label corresponds to the central voxel of the patch and apply the network on overlapping patches extracted around every voxel of the volume. In segmentation, we also use patches for training, but the network associates one class to every voxel of each patch. This is similar to the approach taken in Wu et al. (2019) and Tschannen et al. (2019) to pick faults and diffracting objects.

Supervised deep learning is now established as the reference method to tackle such problems because it leads to the best results on a wide variety of applications (LeCun et al., 2015). The field of deep learning is almost entirely focused around neural networks, which are a set of algorithms expressed as a computational graph built from a sequence of layers performing simple operations. Rather than being manually engineered, the parameters of the transformations are initially chosen at random and given the freedom to adapt to the data and problem at hand. Chaining several layers is a key feature of these algorithms because this architectural design allows the algorithm to learn a hierarchical representation of the data. The deeper layers build upon the work of the previous layers and are sensitive to progressively more abstract and complex features, expressed as a composition of the simpler features learned by the shallower layers (LeCun et al., 2015). When the data exhibit a spatial structure and the surrounding information is relevant to understand the local context, a suitable choice for the linear transformations is convolutions, and the family of algorithms based on them is called CNNs (LeCun et al., 1998).

In this work, we aim to segment a set of K horizons in a 3D seismic stack. Let $\mathbf{h}(il, xl, t)$ be a seismic sample expressed in the 3D coordinate system (il, xl, t), where il and xl are the spatial coordinates in the inline and crossline directions and t is the vertical temporal coordinate. We express the labels as a 4D hypercube $\mathbf{p}(il, xl, t, K+1)$ representing the probable locations of the K target reflectors. For every voxel, $\mathbf{p}$ holds the discrete probability density function $(p_k \geq 0)_{k=1..K+1}$, such that $\sum_{k=1}^{K+1} p_k = 1$. The terms p_1 to p_K are the probabilities of each horizon, whereas p_{K+1} is the probability of not being any of the desired reflectors.

A neural network depending on the parameters $w \in \mathbb{R}^m$, where m is the number of free dimensions, takes as an input the seismic data $\mathbf{h}$ and outputs a pseudoprobability density function $\hat{\mathbf{p}}_w(il, xl, t, K+1)$ defined at every voxel. Training the network consists of optimizing the values of its parameters to increase its prediction performance so that $\hat{\mathbf{p}}_w$ approaches $\mathbf{p}$. A standard pseudodistance measure between two probability distributions is cross entropy, which measures how close the computed distribution is to represent the *true* distribution. In its discrete form, the cross entropy between $\mathbf{p}$ and $\hat{\mathbf{p}}_w$, summed over space and time, is

$$l(w; p, \hat{p}_w) = -\sum_{il,xl,t} \sum_{k=1}^{K+1} p(il, xl, t, k) \times \log[\hat{p}_w(il, xl, t, k)]. \quad (1)$$

For a training data set $\mathcal{D}$ composed of N pairs of samples $(\mathbf{h}^{(i)}, \mathbf{p}^{(i)})$, the training loss is defined as the average over all samples of the loss computed in equation 1:

$$\mathcal{L}(w; \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} l(w; p^{(i)}, \hat{p}_w^{(i)}). \quad (2)$$

The most commonly chosen approach to minimize the loss function of equation 2 is to resort to an optimizer belonging to the minibatch stochastic gradient-descent family (Robbins and Monro, 1951; LeCun et al., 2015). In this iterative procedure, a random subset of the training data set (called a minibatch) is chosen at every iteration, and the local steepest-descent direction is found by computing the gradients of the loss function with respect to the network's parameters using the back-propagation algorithm (Werbos, 1974). The parameters are updated by taking a step toward this direction, and a new minibatch is selected for the next iteration. The learning rate is an important hyperparameter that defines the step size used for the update. When all minibatches have been seen once by the network, we say that it has been trained for one epoch.

Network architecture

Figure 1 and Table 1 present the architecture of our network. It is a traditional feed-forward convolutional network inspired by Ronneberger et al. (2015). The trainable parameters are contained in the convolutional layers, which transform the data by convolving the input with kernels and adding bias coefficients. Convolving with a single kernel yields a 3D feature map. Because every layer contains several kernels, the output of each layer is 4D with the fourth dimension corresponding to the number of channels (i.e., the number of kernels in the layer). We use rectified linear units $(\max(0, .))$ as activation functions for the output of the convolutional layers. To learn abstract concepts, the network should see the data at different scales.

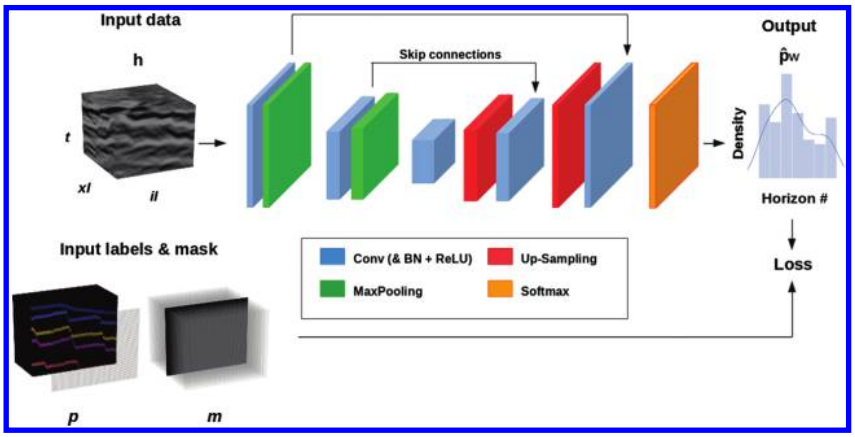


Figure 1. Simplified overview of the 3D CNN used in this work. The data flow is from left to right during the forward pass as indicated by the black arrows. The boxes represent multichannel 4D feature maps (here drawn in 3D for simplicity) color coded by layer type. In this example, the input data $\mathbf{h}$ are interpreted along their central crossline and the mask $\mathbf{m}$ contains ones at the location of the picks along the crossline slice and zeros elsewhere. For simplicity, the output hypercube is represented as a histogram counting the number of voxels associated with each horizon. The exact architecture of the CNN is described in Table 1.

Table 1. Architecture of our CNN, designed after Ronneberger et al. (2015)[5].

Input: Seismic Patch	
01	Conv (5×5×5, 24), BN, ReLU
02	MaxPool (1×1×2)
03	Conv (5×5×5, 24), BN, ReLU
04	MaxPool (1×1×2)
05	Conv (5×5×5, 48), BN, ReLU
06	MaxPool (2×2×2)
07	Conv (5×5×5, 96), BN, ReLU
08	Up-Sampling (2×2×2)
09	Concat; Conv (5×5×5, 48), BN, ReLU
10	Up-Sampling (1×1×2)
11	Concat; Conv (5×5×5, 24), BN, ReLU
12	Up-Sampling (1×1×2)
13	Concat; Conv (5×5×5, 24), BN, ReLU
18	Dropout (30%)
19	Conv (5×5×5, $K+1$), Softmax
Output: Predicted Patch	

[5]Conv stands for convolution, upsampling is performed by nearest neighbor interpolation, BN stands for batch normalization (Ioffe and Szegedy, 2015), ReLU for rectified linear unit, and Maxpool for max-pooling. The size and number of convolutional kernels in each layer are indicated in parentheses. The down and upsampling factors are indicated in parentheses next to the corresponding operations. The blue arrows indicate the skip connections that concatenate the activation maps, along the fourth dimension, which consists of a shallow convolution with its symmetric upsampled counterpart.

The receptive field refers to the effective window size that a convolutional layer is accessing from the original data. For the first layer, the receptive field is equal to the spatial size of the kernels. We use a constant kernel size of $5 \times 5 \times 5$ over the network, and we use max-pooling operations to progressively downsample the data. Max pooling is a sliding window operation that keeps the largest value inside the window and drops the others. A pooling window of $1 \times 1 \times 2$ keeps the inline and crossline dimensions unchanged and downsamples the time dimension by a factor of two. Accordingly, the effective receptive field of the next convolutional layer is twice as large in the vertical direction as in the horizontal ones. Because we aim to classify every voxel of the input data, the network must yield an output that has the same spatial shape as the input. The first half of the network is downsampling the data by a desired rate, and the second half is a symmetric counterpart that progressively transforms the data back to their original spatial shape (see Figure 1). We use nearest neighbor interpolation to perform the upsampling. To compensate for the loss of information caused by the successive pooling layers, we use skip connections (Ronneberger et al., 2015) to inject data from shallow layers to deeper ones (see Table 1).

To choose the number of layers, there is a trade-off between the largest data scale accessible to the network and the memory and computational costs. We consider that for the horizon picking problem, it is more important to access low-frequency information along time than along the inline and crossline dimensions. For this reason, as detailed in Table 1, we alternate between $1 \times 1 \times 2$ and $2 \times 2 \times 2$ downsampling factors. The three pooling layers yield to a total downsampling of $2 \times 2 \times 8$. To regularize the training, we apply batch normalization (Ioffe and Szegedy, 2015) and dropout (Srivastava et al., 2014) and learn the parameters with the Adam optimizer (Kingma and Ba, 2014) using the library TensorFlow (Abadi et al., 2016).

Training data and prediction

When picking horizons, the interpreter needs to build an understanding of the geometry of the area by establishing fault patterns and inferring the depositional and tectonic history of the site. Once the reflectors of interest have been identified, they need to be picked in a dense 3D grid. Most workstations allow this process to be partially automated by letting the interpreter provide information to the tracking algorithm of the reflections that should be followed and iteratively refining the results by making manual adjustments and adding constraints in the mispicked areas. The quality of the automatic tracking will have a big impact on the time needed to complete the task. For our supervised deep-learning approach, we use the picks of the interpreter to create the training labels. As explained in the previous section, the labels should be provided as a hypercube in which the fourth dimension holds the reflector probabilities. For every manually picked horizon, we set the probability of the corresponding voxels to one in the label's hypercube.

To introduce uncertainty in the exact position of the reflectors, we convolve the labels with a normalized 1D Gaussian kernel in the vertical direction (see Figure 2). A difficulty of this approach is that one needs to label every voxel of the training data, which is a tedious task when working with 3D seismic. However, a common solution is to use a masked loss that allows labeling only a subset of the voxels without affecting the training quality (Xu et al., 2015; Peters et al., 2019). In addition to providing the labels, one also defines a binary mask $\mathbf{m}(il, xl, t)$ containing ones for the voxels that are explicitly marked by the interpreter and zeros elsewhere. For instance, if one decides to label an entire inline section, the mask would be a cube of zeros for every inline except for the horizons interpreted on the single inline section in which the mask would contain ones for these voxels (looking at Figure 2, we set to one every voxel of the mask that corresponds to a nonzero probability for at least one of the horizons). If one wishes to highlight horizons with seed points, the mask would contain ones only at the seed locations. Because we incorporate uncertainties in the exact position of the picks by convolving with a Gaussian kernel, we also convolve the binary mask with the same kernel and set to one all voxels whose value is greater than 0.1. The loss function of equation 1 is then modified by masking the cross entropy by an element-wise multiplication:

$$l(w; p, \hat{p}_w) = -\sum_{il,xl,t} m(il, xl, t) \times \sum_{k=1}^{K+1} p(il, xl, t, k) \log[\hat{p}_w(il, xl, t, k)]. \quad (3)$$

In this way, during training, the gradients used to update the parameters will be nonzero only in the picked areas.

Once the network is trained, we run a prediction on the entire seismic stack to obtain a hypercube containing the probability density function for the presence of the horizons at every voxel. Because convolutions and other transformations of our network do not depend on the input data size, we can evaluate the network on data of arbitrary dimensions (Long et al., 2014). Because the entire 3D stack may not

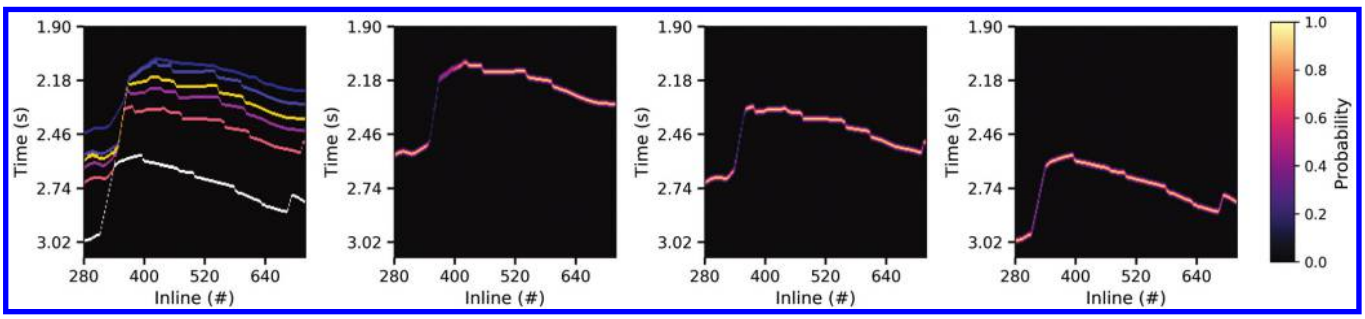


Figure 2. Preparation of the labels for training. The left image shows the picks performed on a 2D section for the different horizons that we want to map. The images on the right are three of the probability slices obtained by convolving in the vertical dimension the picks of individual horizons with a 1D Gaussian kernel. The final labels are created by concatenating every probability slices along an extra dimension. The last slice of the labels corresponds to the "other" class, and its values are chosen such that the total distribution sums to one.

fit in memory, we perform the evaluation in chunks. We split the volume into subcubes, run the segmentation iteratively, and merge the probabilities at the end. To extract an actual 2D horizon surface s_k from the hypercube, we take the position of the maximum probability along the vertical dimension as indicated in equation 4 (see Figure 3):

$$s_k(il, xl) = \underset{t}{\text{argmax}}[\hat{p}_w(il, xl, t, k)]. \quad (4)$$

Because storing the full 4D probability cube can occupy an excessive amount of disk space, one may instead extract the surfaces during the iterative segmentation and only store the horizons. The final horizons are obtained after despiking and smoothing postprocessing procedures.

RESULTS

We evaluate our method on two marine surveys and compare the results of the machine with the interpretations proposed by experts. We create the training data sets by selecting several interpreted lines. We could use seed points instead of full 2D interpretations, but in this way we get more training examples at a minimum extra of manual effort. Seed points may also not be enough to guide the algorithm for difficult reflections that have a low signal-to-noise ratio and that may have a fairly heterogeneous character over the survey. For such reflections, interpreters use their experience and intuition to draw the lines. We also select one interpreted line in a different region for the validation set of each survey. We normalize the seismic volume to the amplitude range $[-1, 1]$, and we extract samples around the training and validation lines with a shape of $[\Delta il, \Delta xl, \Delta t] = [32, 32, 96]$. For both experiments, we use a learning rate of 5×10^{-4} and reduce its value by 33% every 10 epochs. We train the network for 25 epochs with a batch size of 12 samples. To artificially increase the size of the training data sets, we perform a simple data augmentation (Simard et al., 2003) by flipping the inline and crossline directions and by adding white noise with a standard deviation of 0.025 to the input seismic image.

Faults and fine training

The first data set is a stack of 301 inlines × 201 crosslines × 301 time samples with a discretization of 25 m × 25 m × 4 ms. It contains six horizons of interest in a faulted area, and a manual interpretation serves as a reference. We use four inlines to create the training data set. The training takes 47 min on a TITAN X GPU. The final prediction takes less than a minute by splitting the stack in overlapping chunks of 32 inlines. Figure 3 shows the results on two cross sections (not used to prepare the training data) for the top reservoir. The probability attribute can be used to determine areas where the network's prediction is uncertain. The bottom section, for instance, shows that on the other side of the tectonic event (for crosslines less than 1020) is a region of low certainty, although the network is nevertheless able to find the reflection. Figure 4 displays all of the horizons on a crossline section. Although the extracted surfaces do not exactly match the original interpretation, we deem them to be of good quality for a

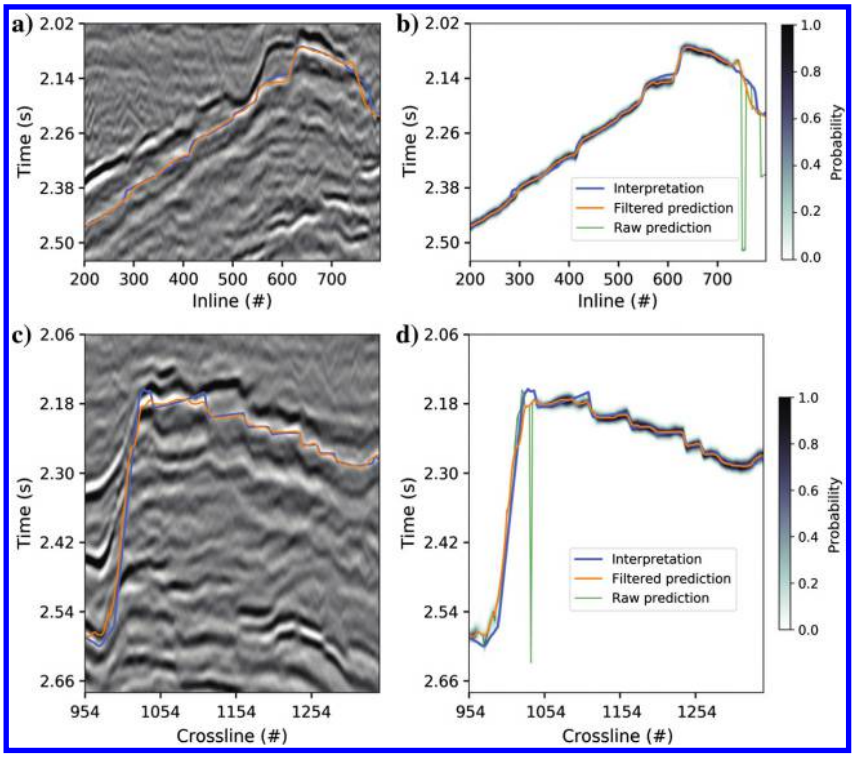


Figure 3. The 2D sections at (a) crossline 1200 and (c) inline 500 through the data and their corresponding probability cube for the top reservoir in (b and d). The interpretation and the machine's prediction are displayed, and we also show the prediction before despiking in semitransparency.

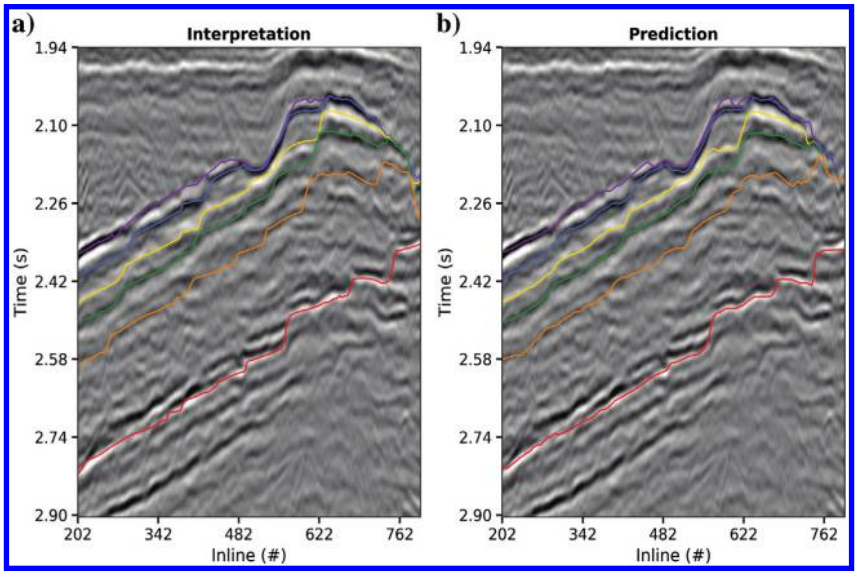


Figure 4. The 2D section at crossline 1200 shows (a) the interpretation and (b) the machine prediction for the six horizons.

first training pass. Even the relatively weak reflectors, such as the reflection highlighted by the orange horizon (the fifth deepest), are picked. Figure 5 shows the top reservoir. Regions of low values in the probability surface highlight the faults and the tectonic folding. By construction of the neural network, we do not impose special assumptions on the lateral continuity of the reflectors and, in particular, fault discontinuities are, in theory, not a problem for the prediction. We see that the faults are better defined on the expert's interpretation. This might be explained by the fact that it was done by hand and that the interpreter picked the reflection from both sides of the discontinuity and filled the gap by linear interpolation, leading to a sharp transition.

If one is not completely satisfied by the results of the first training pass, one can fine-tune the network training. Figures 6 and 7 show the effect of such fine training. In the area marked by the red rectangle in Figure 6a, we place 10 seed points to extract additional training examples. We create a new training data set composed of 10 samples extracted around the seed points, as well as 50 samples randomly chosen from the original training set. We further train the network for approximately 2 min with a learning rate 10 times smaller than the original one. We use a much smaller learning rate because the network is already trained, and we only want to fine tune its weights. We also incorporate samples from the original data set to limit overfitting. This phenomenon happens when the training set is too small and the network becomes overly specialized at recognizing the training data and performs poorly in other areas of the survey. For this reason, we also only reevaluate the fine-trained network in a small region around the seed points. After rerunning the segmentation, we observe that the prediction follows the interpretation more closely.

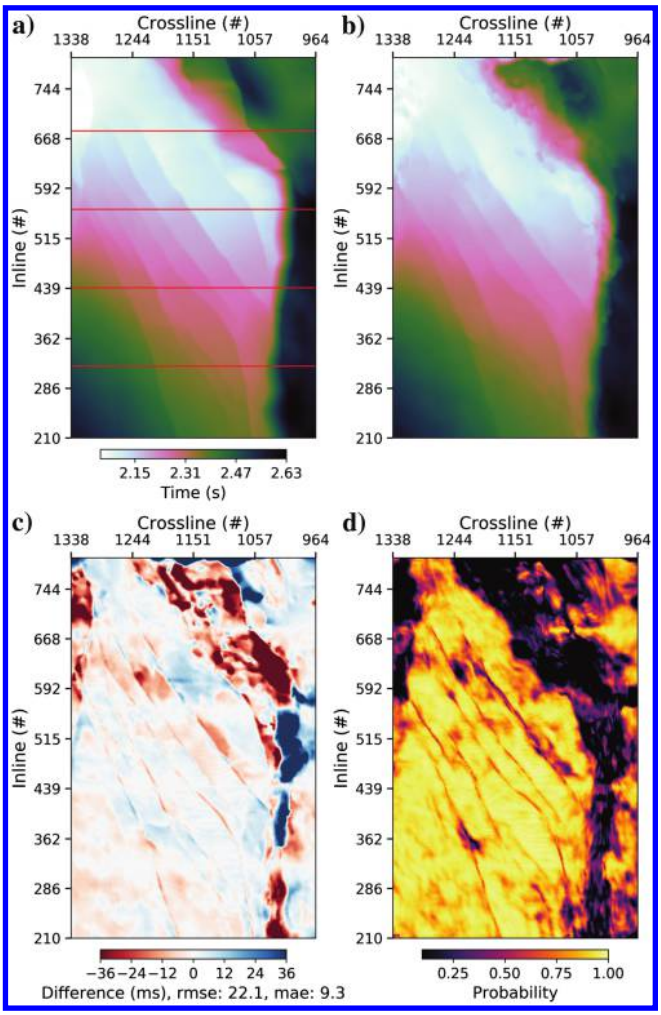


Figure 5. Map views of the top reservoir. (a) Solution provided by the interpreters; the training lines are shown in red. (b) Horizon predicted by our neural network. (c) Difference between the interpretation and the prediction. We also report the root-mean-square error (rms error) and mean absolute error (MAE). (d) Probability associated with the picks.

Extension to prestack seismic data

The second data set consists of prestack angle gathers of dimension 999 inlines × 699 crosslines × 30 angles × 241 time samples with a discretization of 12.5 m × 12.5 m × 2° × 4 ms. It contains six horizons of interest, around a reservoir, interpreted with a combination of handpicking and crosscorrelation-based autotracking. We use one inline and one crossline from a 2° to 12° near-angle stack to create the training data set, and we train the network for 1.03 h. The evaluation runs in less than a minute by splitting the volume in overlapping chunks of 32 inlines. Figures 8 and 9 show the results, obtained by predicting on the near stack, on a crossline section away from the training lines. The autopicking results are similar to the reference baseline, and differences observed on the weaker reflectors are subject to discussion with experts.

In addition, we study how sensitive the network is to changes in the wavelet in the prestack domain. We apply the trained network iteratively on all angle planes from 2° to 60°. Each angle plane is a 3D volume with the same dimensions as the near stack, and the final prediction yields 3D prestack horizons that depend on the incidence angle. We focus on the top of the reservoir shown in Figure 10 and display the corresponding horizon in Figures 11 and 12. We see that the network is correctly picking the horizon up to a certain angle before losing it once the waveform becomes too different from the one observed in the near stack. Extending horizons to the prestack domain is useful to perform an improved amplitude versus angle analysis because the gradient is strongly sensitive to moveout effects.

DISCUSSION

CNNs present several advantages to tackle the horizon picking problem. They do not require strong prior information to operate, and they can adapt to any seismic data. For instance because they work by scanning the entire volume and do not expect a reflection to be found within a certain vertical distance between two neighboring traces, they are appropriate to follow a signal in a faulted area or across a steeply dipping event. In Figure 5d, we observe that the confidence of the network is low at the faults because the reflection at these exact locations is not well-defined, but it nevertheless follows the horizon across the fault blocks. CNNs also have good scalability with respect to the number of reflections to be predicted. When increasing the number of horizons, one only needs to increase the number of channels in the last convolutional layer by the same amount, which results in a minor growth in computational cost.

The fact that our network performs a segmentation, instead of a classification, of the input data is also an advantage. Indeed, whereas segmentation networks need to see each voxel only once, classification networks associate one label to the center of a fixed-size input

patch, and obtaining a prediction over the entire volume requires us to evaluate the network on a few overlapping patches equal to the number of voxels in the data. For standard seismic stacks containing 10^6–10^9 voxels, this induces a considerable overhead that can lead to an evaluation time much greater than the training time itself. Because the evaluation time of our segmentation approach is small, it is suitable to run many successive segmentation on offset or angle planes to follow a reflection in the prestack domain, as shown in Figure 11.

The multidimensional and multiscale nature of CNNs also make them robust classifiers. In Figure 13, we experiment with a 1D network that only analyzes traces along time and observe that, given the same training data, the performance is worse than with the corresponding 3D version. The limited resolution of seismic data, and the various sources of noise, often lead to uncertainties in the interpretation. As such, we believe that exploiting deep learning in an interactive manner, by giving the interpreter the possibility to refine the results by progressively adding new examples to the training data set, is an important part of the presented workflow. By using a masked loss, we give flexibility to the interpreter to pick examples using either seed points or line interpretations, without worrying about the 3D aspects of the algorithm.

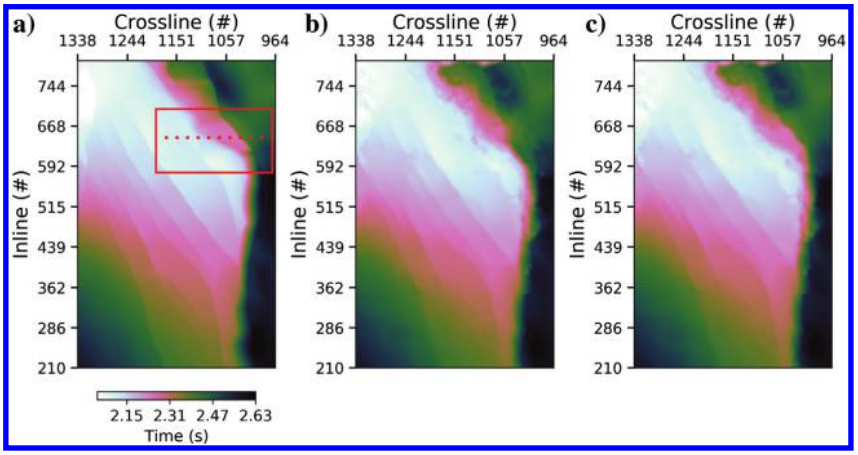


Figure 6. Map views of the top reservoir. (a) Ground truth provided by the interpreters. (b) Horizon picked by our neural network after the first pass of training. (c) Prediction, updated only inside of the red square, after fine-tuning the network. The rms error, computed inside the red square between the interpretation and the prediction, decreased from 36.9 to 15.5 and the MAE from 18.8 to 9.7. The 10 seed points used to fine-train the network are shown by the red dots.

However, there are also challenges in applying deep learning to the horizon interpretation problem. Neural networks are dependent on the quantity and quality of the training data (LeCun et al., 2015). They can adapt to any data set and do not suffer from prior-induced limitations, but they need to be given enough examples before correctly generalizing. Although we show that our method only requires a reasonable amount of training labels, the approach still strongly relies on manual interpretation by an expert. For a seismic volume containing many horizons, providing the initial training samples is an obstacle to overcome. The workflow is also penalized by a slow start because the neural network needs to be trained to convergence before obtaining the firsts results. In Wu et al. (2019) and Tschannen et al. (2019), the authors use synthetic modeling to create large training data sets and train networks that can generalize to real data. However, in this application we aim to pick specific reflections

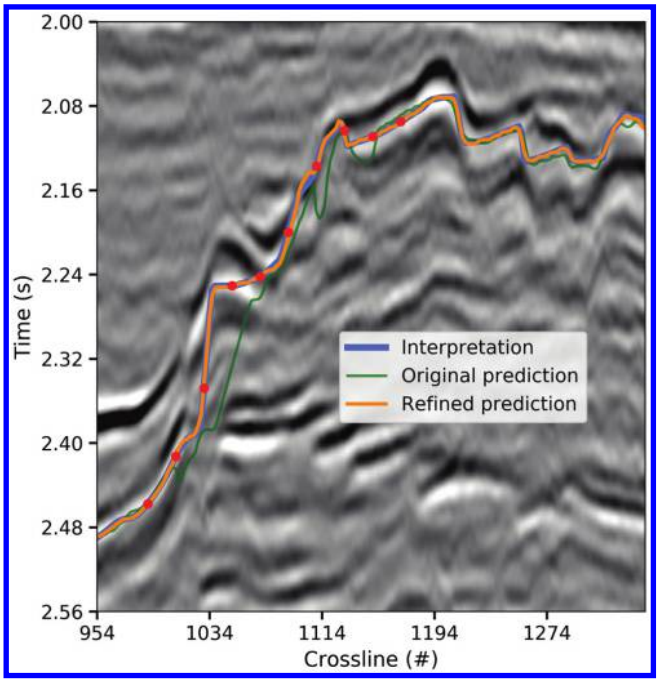


Figure 7. The 2D section at inline 646. We plot the interpretation, as well as the original and refined predictions are shown in Figure 6. The seed points used for fine-training are displayed as red dots.

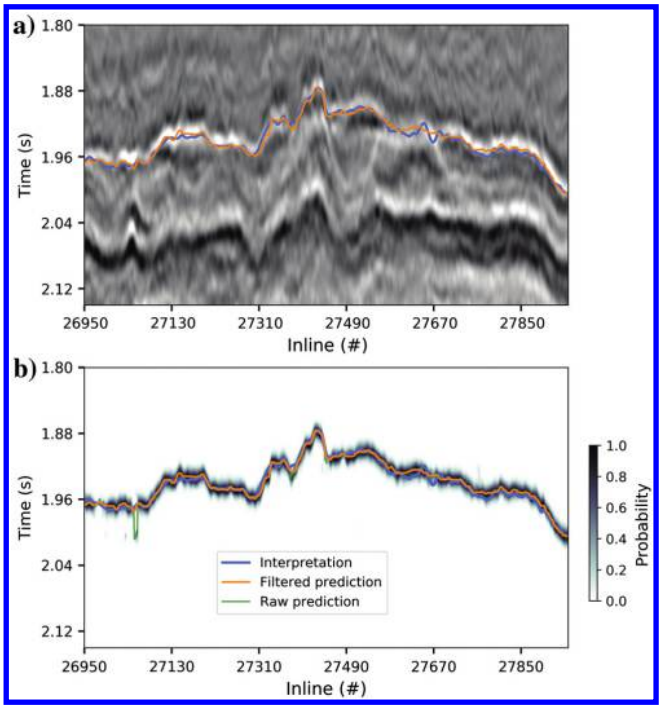


Figure 8. The 2D section at crossline 16301 of (a) the near-angle stack and (b) the probability cube for the top reservoir. The interpretation and the machine's prediction are displayed.

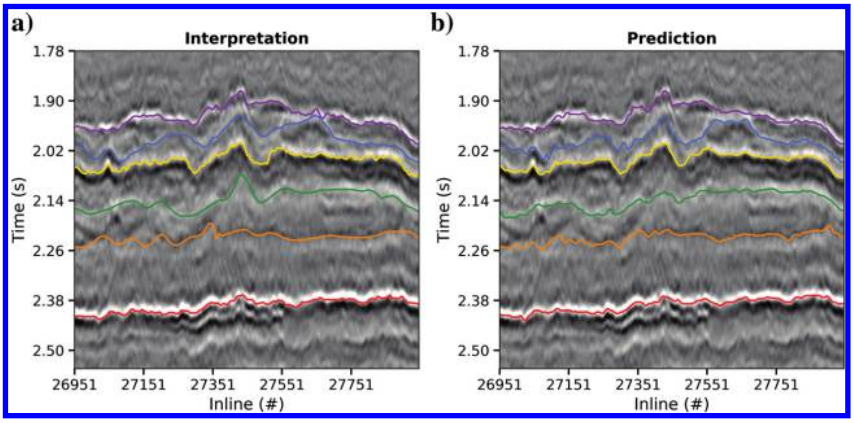


Figure 9. The 2D section at crossline 16301 of the near-angle stack shows (a) the interpretation and (b) the machine prediction for the six horizons.

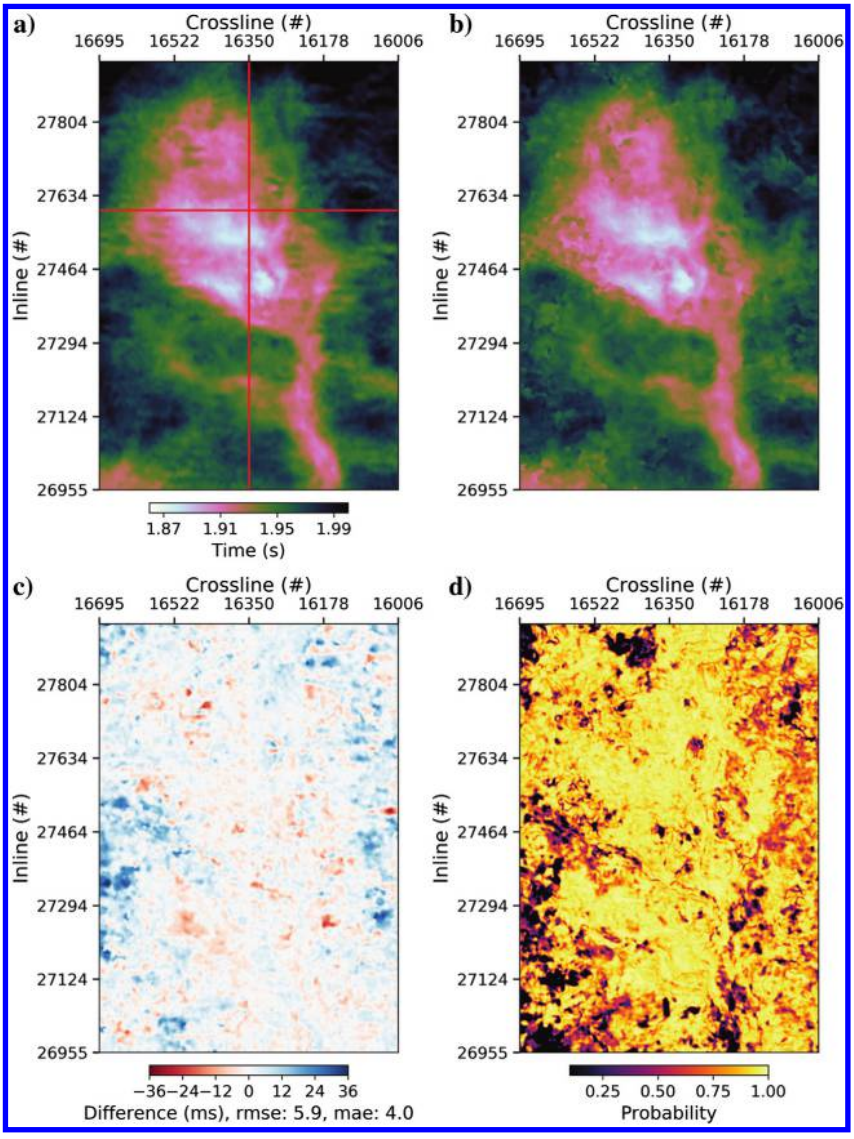


Figure 10. Map views of the top reservoir for the near stack. (a) Solution provided by the interpreters; the training lines are shown in red. (b) Horizon predicted by our neural network. (c) Difference between the interpretation and the prediction. (d) Probability associated with the picks.

and we cannot rely on the exploitation of more abstract and generalizable concepts such as faults or diffractions. By borrowing elements from state-of-the-art architectures (Ioffe and Szegedy, 2015; Ronneberger et al., 2015) and designing the network to have a smaller reach in the inline and crossline directions than in the time direction, we keep the computational and memory requirements low.

Another difficulty lies in the empirical nature of deep learning. Training a CNN requires the tuning of several hyperparameters. To find these parameters, practitioners usually rely on a performance metric evaluated on a validation data set. However in geosciences, labeled data are scarce and sometimes noisy. In our experiments, monitoring the training process using a single test line is not always very informative, and we also find it helpful to use a more qualitative assessment by carefully visualizing the predicted horizons together with the seismic data. We show in Figure 14 the evolution of the training and validation errors with the number of epochs for the first data set. We see that the validation error decreases with the number of epochs in a similar fashion as the training error. This behavior is a good sign that indicates low overfitting and is in accordance with the good quality, on average, of the prediction shown in Figure 5b. However, horizon interpretation is a task that requires precision, and the geology may rapidly change over a data set. The average value of the validation loss does not inform us directly on how the network is performing in the different areas of the data set. For example, in Figure 7, the prediction (in green) is very close to the interpretation except in the steeply dipping area, between crosslines 1000 and 1140. We also see that the validation error is more volatile and on average a bit larger than the training error, which indicates that the network does not perfectly generalize to the entire survey. Using larger training and validation sets would certainly improve the prediction's quality, but a trade-off must be found because increasing the amount of manual labeling is in contradiction to the automatization that is the purpose of the algorithm. Overall, we find that our architecture is not overly sensitive to key hyperparameters such as the learning rate, the number of kernels per layer, and the number of epochs, and we obtain good results on two different data sets with the same parameterization. As stated above, our solution depends on the training data, and we see for instance in Figure 12 that the prediction does not work when the waveform is too different from the one observed in the training data. In this case, to successfully recognize the reflection beyond the polarity reversal angle, one would need to repeat the training procedure using examples picked on a far stack.

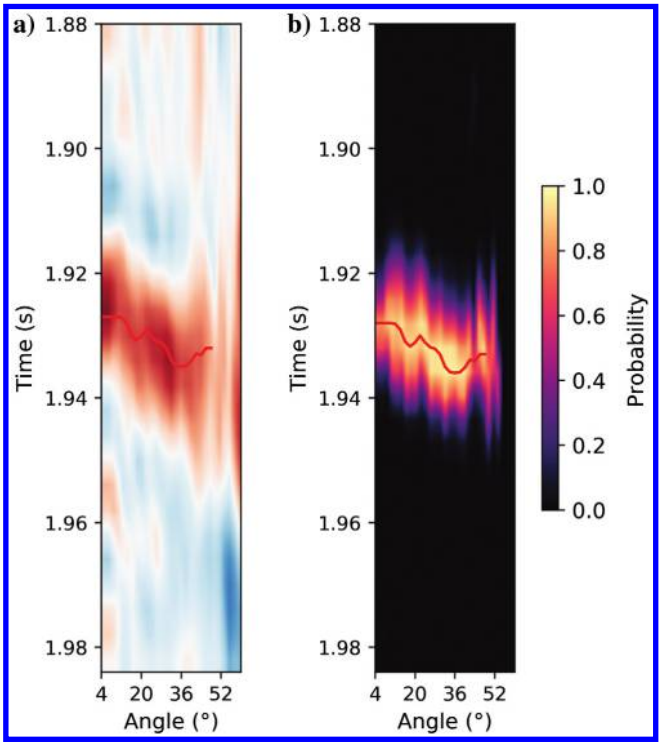


Figure 11. Prediction of the top reservoir across the prestack domain. (a) Angle gather at inline 27208 and crossline 16022; the predicted horizon is drawn in red. The prediction is following the moveout of the reflector up to 40°, and it fails to recognize the event once the waveform becomes too different from the reference near stack, in particular because of stretching. (b) Prediction probability.

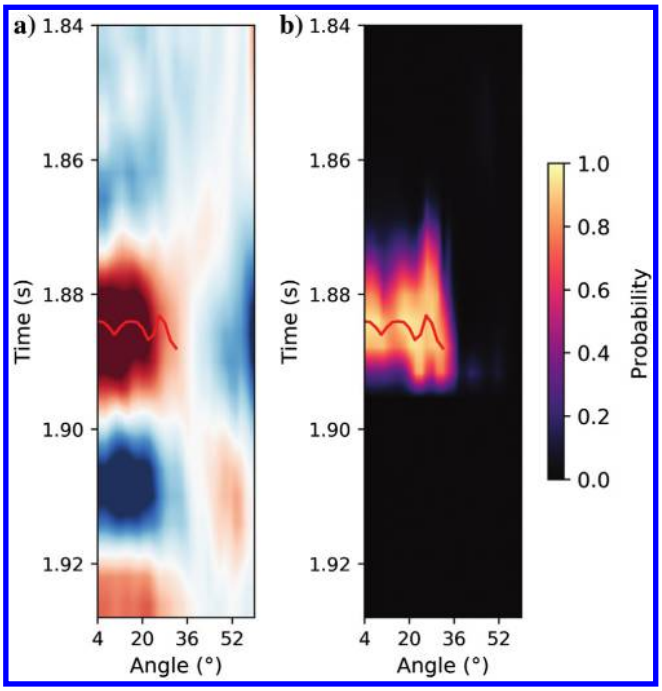


Figure 12. Prediction of the top reservoir across the prestack domain. The setting is similar to Figure 11, for a gather at inline 27600 and crossline 16402. The network correctly picks the reflection until 34°, and it fails to recognize the event once the waveform becomes too different from the reference near the stack because of a polarity reversal.

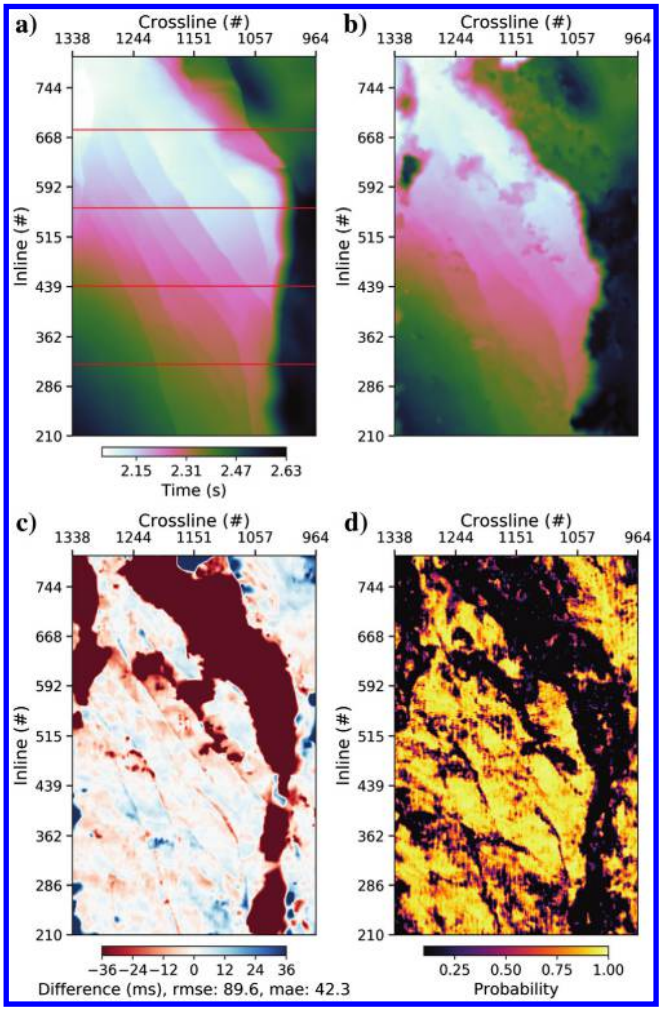


Figure 13. Results obtained for the same horizon as shown in Figure 5 but with a 1D CNN. The architecture of the CNN is the same as the one presented in Table 1, but 3D kernels are replaced by $1 \times 1 \times 5$ kernels. We use training samples of size $1 \times 1 \times 96$ in the inline, crossline, and time dimensions, and we train the network for 40 epochs with the same parameters as for Figure 5.

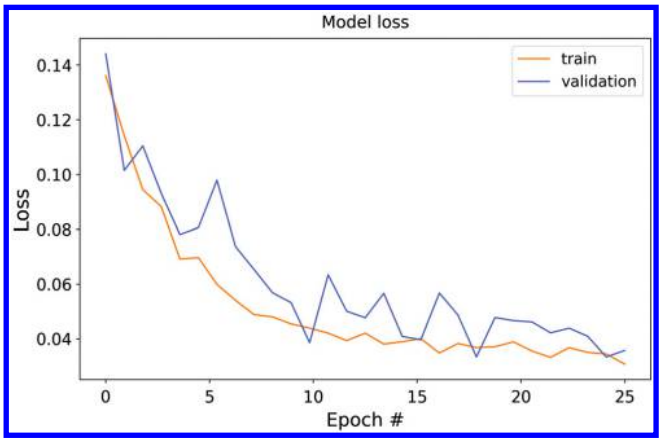


Figure 14. Training and validation errors, for the first data set, monitored over the training epochs.

In addition, CNNs may also suffer from boundary condition artifacts. In Figure 5b, some outliers are visible in the upper part of the predicted surface (at large inlines). Because the network is based on convolutions, performing a prediction on the edge of the volume requires artificially extending the data, using for instance a mirroring condition, which may affect the quality of the output. Finally, although here we treat the multihorizon picking problem, we do not explicitly enforce an ordering of the predictions, and crossings may occur. Because the network is trained using patches, it does not have knowledge of the global coordinate system, and it is nontrivial to make it aware of the relative position of the different samples. We do not address this issue here, and we enforce ordering of the horizons as a postprocessing operation.

CONCLUSION

We have discussed a practical approach to efficiently and simultaneously pick several horizons in a 3D seismic image. We formulate the problem as a segmentation task, in which the position of the reflectors is expressed as a probability distribution, and we use supervised deep learning to solve it. We design an efficient architecture for a 3D CNN that allows us to analyze the data over multiple scales while keeping the memory and computational requirements low. We use a masked loss function to give flexibility to the interpreters in the way they pick the training data. The method requires us to label only a few lines through the survey to yield good initial results, and we show how interpreters can progressively improve the predictions by fine-tuning the network training. Validation on field data shows the potential of the method, and in future work we plan to integrate, within the same end-to-end workflow, the interpretation of other structural features such as faults or salt bodies.

ACKNOWLEDGMENTS

We are grateful to Sharp Reflections for providing us with the data and horizons that we used in this work. We are also grateful for the helpful comments and suggestions provided by three anonymous reviewers.

DATA AND MATERIALS AVAILABILITY

Data associated with this research are confidential and cannot be released.

REFERENCES

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, and M. Isard, 2016, Tensorflow: A system for large-scale machine learning: Proceedings of the 12th Symposium on Operating Systems Design and Implementation, 265–283.

Alberts, P., M. Warner, and D. Lister, 2000, Artificial neural networks for simultaneous multi horizon tracking across discontinuities: 70th Annual International Meeting, SEG, Expanded Abstracts, 651–653, doi: 10.1190/1.1816150.

Aurnhammer, M., and K. D. Toennies, 2002, Horizon correlation across faults guided by geological constraints: Proceedings of the SPIE, **4667**, 312–322, doi: 10.1117/12.467992.

Bacon, M., R. Simm, and T. Redshaw, 2003, 3-D seismic interpretation: Cambridge University Press.

Dorn, G. A., 1998, Modern 3-D seismic interpretation: The Leading Edge, **17**, 1262–1272, doi: 10.1190/1.1438121.

Gersztenkorn, A., and K. J. Marfurt, 1999, Eigenstructure-based coherence computations as an aid to 3-D structural and stratigraphic mapping: Geophysics, **64**, 1468–1479, doi: 10.1190/1.1444651.

Harrigan, E., J. Kroh, W. Sandham, and T. Durrani, 1992, Seismic horizon picking using an artificial neural network: IEEE International Conference on Acoustics, Speech, and Signal Processing, 105–108.

Hoyes, J., and T. Cheret, 2011, A review of global interpretation methods for automated 3D horizon picking: The Leading Edge, **30**, 38–47, doi: 10.1190/1.3535431.

Huang, L., X. Dong, and T. E. Clee, 2017, A scalable deep learning platform for identifying geologic features from seismic attributes: The Leading Edge, **36**, 249–256, doi: 10.1190/tle36030249.1.

Ioffe, S., and C. Szegedy, 2015, Batch normalization: Accelerating deep network training by reducing internal covariate shift: arXiv preprint, arXiv:1502.03167.

Kingma, D. P., and J. Ba, 2014, Adam: A method for stochastic optimization: arXiv preprint, arXiv:1412.6980.

Kusuma, T., and B. C. Fish, 1993, Toward more robust neural-network first break and horizon pickers: 63rd Annual International Meeting, SEG, Expanded Abstracts, 238–241, doi: 10.1190/1.1822449.

LeCun, Y., Y. Bengio, and G. Hinton, 2015, Deep learning: Nature, **521**, 436–444, doi: 10.1038/nature14539.

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner, 1998, Gradient-based learning applied to document recognition: Proceedings of the IEEE, **86**, 2278–2324, doi: 10.1109/5.726791.

Leggett, M., W. A. Sandham, and T. S. Durrani, 2003, Automated 3-D horizon tracking and seismic classification using artificial neural networks: Springer Netherlands, 31–44.

Liu, Z., J. Cao, Y. Lu, S. Chen, and J. Liu, 2019, A seismic facies classification method based on the convolutional neural network and the probabilistic framework for seismic attributes and spatial classification: Interpretation, **7**, no. 3, SE225–SE236, doi: 10.1190/INT-2018-0238.1.

Lomask, J., and A. Guitton, 2007, Volumetric flattening: An interpretation tool: The Leading Edge, **26**, 888–897, doi: 10.1190/1.2756869.

Long, J., E. Shelhamer, and T. Darrell, 2014, Fully convolutional networks for semantic segmentation: arXiv e-prints arXiv:1411.4038.

Pauget, F., S. Lacaze, and T. Valding, 2009, A global approach in seismic interpretation based on cost function minimization: 79th Annual International Meeting, SEG, Expanded Abstracts, 2592–2596, doi: 10.1190/1.3255384.

Peters, B., E. Haber, and J. Granek, 2019, Neural networks for geophysicists and their application to seismic data interpretation: The Leading Edge, **38**, 534–540, doi: 10.1190/tle38070534.1.

Robbins, H., and S. Monro, 1951, A stochastic approximation method: The Annals of Mathematical Statistics, **22**, 400–407, doi: 10.1214/aoms/1177729586.

Ronneberger, O., P. Fischer, and T. Brox, 2015, U-Net: Convolutional networks for biomedical image segmentation: arXiv e-prints arXiv:1505.04597.

Shi, Y., X. Wu, and S. Fomel, 2019, SaltSeg: Automatic 3D salt segmentation using a deep convolutional neural network: Interpretation, **7**, no. 3, SE113–SE122, doi: 10.1190/INT-2018-0235.1.

Simard, P. Y., D. Steinkraus, and J. C. Platt, 2003, Best practices for convolutional neural networks applied to visual document analysis: 7th International Conference on Document Analysis and Recognition, 958–963.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014, Dropout: A simple way to prevent neural networks from overfitting: The Journal of Machine Learning Research, **15**, 1929–1958.

Tschannen, V., N. Ettrich, M. Delescluse, and J. Keuper, 2019, Detection of point scatterers using diffraction imaging and deep learning: Geophysical Prospecting, **68**, 830–844, doi: 10.1111/gpr.v68.3.

Veezhinathan, J., F. Kemp, and J. Threet, 1993, A hybrid of neural net and branch and bound techniques for seismic horizon tracking: Proceedings of the ACM/SIGAPP Symposium on Applied Computing, ACM, 173–178.

Waldeland, A. U., A. C. Jensen, L.-J. Gelius, and A. H. S. Solberg, 2018, Convolutional neural networks for automated seismic interpretation: The Leading Edge, **37**, 529–537, doi: 10.1190/tle37070529.1.

Werbos, P., 1974, Beyond regression: New tools for prediction and analysis in the behavioral sciences: Ph.D. thesis, Harvard University.

Wu, X., and S. Fomel, 2018, Least-squares horizons with local slopes and multigrid correlations: Geophysics, **83**, no. 4, IM29–IM40, doi: 10.1190/geo2017-0830.1.

Wu, X., L. Liang, Y. Shi, and S. Fomel, 2019, FaultSeg3D: Using synthetic data sets to train an end-to-end convolutional neural network for 3D seismic fault segmentation: Geophysics, **84**, no. 3, IM35–IM45, doi: 10.1190/geo2018-0646.1.

Xiong, W., X. Ji, Y. Ma, Y. Wang, N. M. AlBinHassan, M. N. Ali, and Y. Luo, 2018, Seismic fault detection with convolutional neural network: Geophysics, **83**, no. 5, O97–O103, doi: 10.1190/geo2017-0666.1.

Xu, J., A. G. Schwing, and R. Urtasun, 2015, Learning to segment under various forms of weak supervision: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3781–3790.

Zheng, Y., Q. Zhang, A. Yusifov, and Y. Shi, 2019, Applications of supervised deep learning for seismic interpretation and inversion: The Leading Edge, **38**, 526–533, doi: 10.1190/tle38070526.1.

4.3.2 Determination of Geological Facies

Classification from Well Logs

In this section we insert the work done in Tschannen et al. (2017) on the determination of geological facies with well logs.

Facies classification from well logs using an inception convolutional network

Valentin Tschannen[†‡]*, Matthias Delescluse*[‡]*, Mathieu Rodriguez*[‡] *and Janis Keuper*[†]
[†]*Fraunhofer ITWM, Kaiserslautern, Germany*
[‡]*École Normale Supérieure, Paris, France*

SUMMARY

The idea to use automated algorithms to determine geological facies from well logs is not new (see e.g Busch et al. (1987); Rabaute (1998)) but the recent and dramatic increase in research in the field of machine learning makes it a good time to revisit the topic. Following an exercise proposed by Dubois et al. (2007) and Hall (2016) we employ a modern type of deep convolutional network, called *inception network* (Szegedy et al., 2015), to tackle the supervised classification task and we discuss the methodological limits of such problem as well as further research opportunities.

INTRODUCTION

Facies are used by geologists to group together body of rocks with similar characteristics in order to facilitate the study of a basin of interest. Their definition is rather subjective as it depends on the attributes we choose for the classification. One may for instance focus on biological differences by looking at the type of shells present in the samples or we may wish to emphasis petrological characteristics by accounting for the granulometry and the mineralogy. In the case of Oil&Gas reservoirs, porosity and permeability are critical properties to determine since they give indications about the potential volume of fluids that might be stored in a rock and how they will flow during production. We can therefore expect that grains size, shape and density as well as the depositional and compaction history of the rocks will be a dominant factor for the categorization. While the main source of information for defining those facies comes from the observation of core samples under visible and x-ray light, we also have a variety of well log recordings at our disposal. By measuring the acoustic and electrical responses as well as the nuclear radiations of the drilled medium, we can infer properties about its rock matrix and fluid content and indirectly relate them to the porosity, permeability or fluid saturation of the rocks.

Classifying high dimensional data into groups is one of the main branch of the popular field of machine learning. Among the vast panel of methods, current attention is mainly received by so called *deep neural networks*. Learning from experience, those algorithms are able to discover abstract representations and to understand the data in terms of a hierarchy of concepts. They have shown impressive results in a vast panel of supervised classification problems (LeCun et al., 2015).

METHOD

Convolutional networks

Deep networks have recently become the method of choice to solve problems in the fields of computer vision and speech recognition. Their general architecture can be seen as a sequence of layers connected to each other trough a non-linear differentiable function. Each layer is composed of a number of units that independently apply a simple affine transform on their input. The power of such approach resides in the way the coefficients of the transforms are set. Rather than being manually engineered they are given the freedom to adapt to the data by progressively learning from examples. Stacking several layers is a key in the success of those algorithms. Due to the non-linearity applied in between each layers, the learnt coefficients tend to be sensitive to progressively more abstract and complex features in the data as we go deeper in the network. The global perspective of those algorithms however has some inconveniences. They require a huge number of parameters and are difficult to train when working on real world datasets (Lecun et al., 1998).

To overcome those drawbacks, early researchers such as Lecun et al. (1998) proposed to restrict the so called receptive fields of each units to localized regions of the data. By making the observation that the world is compositional, they argued that instead of using the entire input at once to learn the coefficients, units should rather process local groups of samples in a sliding manner. This particular class of networks are named convolutional networks (ConvNets). In addition of being computationally and memory effective (each unit now posses only as many parameters as the size of its receptive field) it also appears to be a more robust way to proceed when it comes to field recordings. In the shallow part, by looking at localised regions of the data, the units typically learn wavelet-like convolutional filters that are useful in detecting basic features. The deeper units will take advantage of the simple feature detectors provided by the previous layers to make their own advanced detectors. Furthermore, by using zoom-out operations (referred in the literature as pooling) in-between the layers, deeper units will look at the data in a progressively more global way (LeCun et al., 2015). As an example, if we assume the gradient of the well logs to be relevant in our problem, it is likely that knowing the gradient over the entire logs will be necessary. Rather than having to learn a large filter made of the concatenation of many differential operators, it is easier to simply learn one operator and apply it in a convolutional manner.

ConvNets are typically designed in a reversed pyramid manner by increasing the number of units as we go to deeper layers. This will progressively transform the original data into a very high dimensional, non-linear space, where clustering, classifications or regressions can be effectively conducted.

Classification

In this work we are interested in employing a ConvNet to solve a supervised classification problem. Using training data coming from wells where the facies sequences were already de-

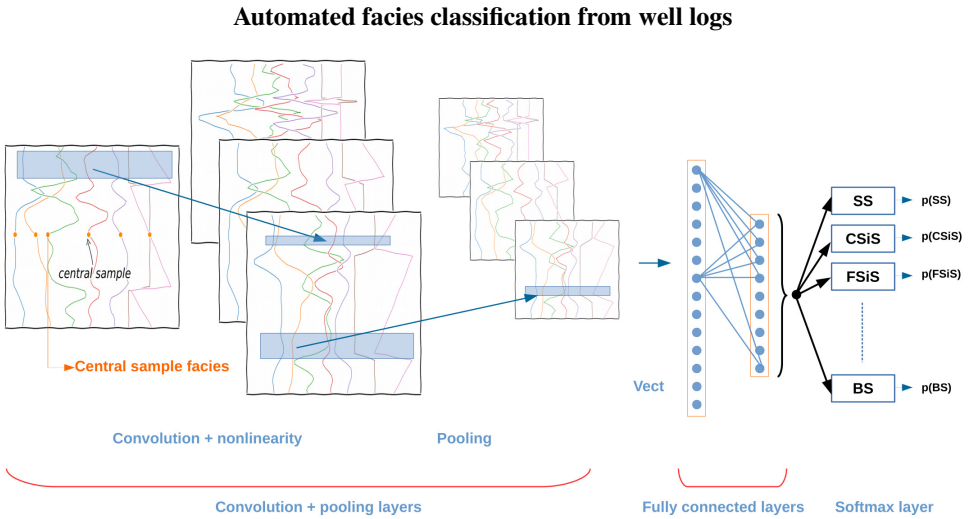


Figure 1: Schematic architecture of a 1 dimensional ConvNet. A short depth-window is extracted from the logs around an example of facies. Filters in the convolution layers are 2-dimensional, in measured-depth (meters or feet) and in channels (number of logs per well) but applied only along depth. The number of filters in the first layer will determine the number of input data to the second layer. Pooling operations reduce the length of the logs by dropping every other samples. On input to the fully connected layers, all filtered well logs are concatenated into a large 1-dimensional series. The softmax layer outputs a vector whose elements form a discrete probability distribution of the facies.

termined by geologists, we train the network to predict facies series that accurately match the experts' interpretation. To this end, in addition to the convolutional and pooling layers described in the previous section, we also need to append fully connected (fc) layers and an output layer. Unlike the convolutional layers, fc layers have access to the entire input at once. Since they come after the deepest convolutional layer, they will be fed with already abstracted individual feature detectors. With the help of additional non-linearities between them, their role is to appropriately combine the abstracted features together in order to solve the supervised task. The output layer is converting the information fed by the fc layers (usually) as a discrete probability distribution of the facies. At each iteration, a random subset of the examples are sent to the network, and using an appropriate objective function, such as the cross-entropy, we compare the predicted labels to the training labels. A global error term is computed and back-propagated though the network to update all the coefficients, in order to minimize the classification error. This approach is called the stochastic gradient descent (Lecun et al., 1998). If the network was properly trained, it should now be able to generalise to new wells where we do not know in advance the facies sequence.

The inception modules

Szegedy et al. (2015) proposed an interesting modification to the convolutional layers. Instead of having a homogeneous setting, they split the layers into four distinct paths (Figure 2) and they introduce the 1×1 convolutions as a cost-effective and efficient feature augmentation technique. The 1×1 convolutions can be seen as a weighting of the different input filters, producing many possible linear combinations that the network can choose from in later layers, while keeping the number of outputs reasonably small. They are also followed by a non-linear activation which further improves the generalization power of the network.

The layout of the inception module can be seen in Figure 2. The data follows each of the four path in parallel before being concatenated at the output. Since it is unclear whether the low frequency or the hight frequency (or both) information contained in the well logs will dominate the learning process, we better let the algorithm decides on its own. As we stack those modules on top of each other, the network progressively learns more and more abstract features from the data which can then be fed to the softmax classifier.

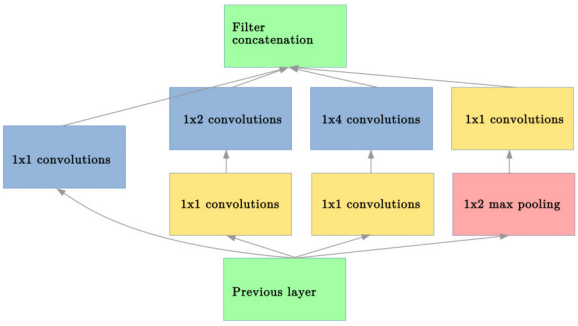


Figure 2: Inception module architecture after Szegedy et al. (2015). On the left, the 1×1 convolution will preserve the vertical resolution of the log sequences. The small kernel convolution will be more sensitive to high frequency informations. The large kernel convolution will be more sensitive to low frequency information. Finally, on the right side, a pooling followed by a 1x1 convolution will perform a sort of low-frequency filtering of the logs, in order to progressively look at more spatially averaged features.

Regularization techniques

The biggest challenge in teaching a deep network lies in preventing over-fitting, whos symptom is a large drop between the training and the testing performances. Rather than learning parameters that capture the nature of the data, it is often observed that a network simply memorizes the training examples. Hence, this results in a very high prediction accuracy on the examples it has already seen, but in poor generalization performances. The most important is to make sure that the problem we want to solve is well-posed and that we have enough training samples. Additionally, many preconditioning and regularization techniques have been developed. Common practice involve a standardization of the input data, an appropriate choice of objective function with the possible addition of penalty terms and the use of dropout (Srivastava et al., 2014) to enforce redundancy in the learnt filters weights. Hyper-parameters, such as the learning rate (step size of the gradient descent) should be carefully tuned by using subsets of the training examples for blind validation. Due to the random initialization of the network and the stochastic nature of the learning phase, results will vary from run to run, and one should aim to design a stable algorithm.

EXPERIMENT

The data we used and the experiment settings originate from a class taught at the University of Kansas (Bohling and Dubois, 2003); (Dubois et al., 2007). An up-to-date implementation of our method with the library `Tensorflow` can be found on Github*. A total number of 11 wells were supplied, each containing 7 logs and a corresponding rock facies series. Out of those 11 wells, 9 were used for training the network and the remaining 2 for a blind evaluation. Among the logs, 5 come from wireline measurements sampled every 1.5 meters (gamma ray, resistivity logging, photoelectric effect, neutron-density porosity difference and average neutron-density porosity) plus an additional 2 geological constraints (non-marine versus marine indicator and relative position) derived from knowledge of the reservoir area. Moreover, we remark that the 2 neutron-density logs are computed using mineralogy dependent coefficients, which may be an additional source of uncertainty.

From observations of core samples, geologists determined that the stratigraphy could be described by 9 different facies: Nonmarine sandstone (SS), Nonmarine coarse siltstone (CSiS), Nonmarine fine siltstone (FSiS), Marine siltstone and shale (SiSh), Mudstone (limestone - MS), Wackestone (limestone - WS), Dolomite (D), Packstone-grainstone (limestone - PS) and Phylloid-algal bafflestone (limestone - BS).

We show in Figure 3 the results obtained for one of the two blind wells and we assess the quality of the machine's prediction with respect to the geologists' classification in term of the F1 score (Figure 4). Visually, the predicted stratigraphy seems reasonable in the first order, as it looks like a median filtered version of the man-made solution. However, an average F1 score of 0.574 and the many observable short intervals in the lithofacies sequence presenting dissimilarities, bring out the difficulties of the network to match the geologists' answer with a high resolution. For instance, if we focus on the packestone layer (PS, in purple) predicted by the machine with a fairly high confidence between 2990m and 3005m, we see a drop in the probabilities at the borders indicating uncertainties in the exact location of the transitions from the above siltstone and the coming wackestone. We also observe that the very fine layers (between 1.5m and 3m thick) of dolomite, mudestone and wackstone found inside the main packestone layer by the geologists were simply not recognized by the machine.

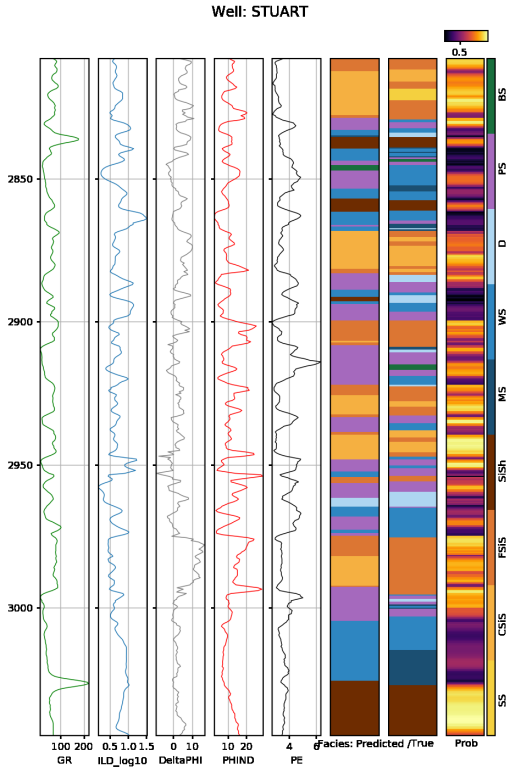


Figure 3: Results of the classification on a blind well. The five first plots show the measured logs as a function of depth (in meters). The two facies series represent the prediction by the machine on the left and the solution proposed by the geologists on the right. The last plot gives the probability with which the machine selected the facies. Bright yellow colors indicate a high certitude (above 70%) while the dark colors indicate uncertain regions (below 50%). The colorbar on the very right gives the correspondences to the facies. (Plot modified from del Monte (2015)).

DISCUSSION

The results published by Hall (2016) and the SEG show that all the *deep learning* methods gave F1 scores below 0.60 and that the *decision tree* like methods fell below 0.65. For a machine learning contest those scores are surprisingly bad, and it is also

* https://github.com/vts21/2016-ml-contest/tree/master/itwm

Pred / True	SS	CSiS	FSiS	SiSh	MS	WS	D	PS	BS	Total
SS	0	14	0	0	0	0	0	0	0	14
CSiS	0	83	24	0	1	0	0	3	0	111
FSiS	0	60	58	0	0	0	0	11	0	129
SiSh	0	0	10	45	9	22	0	1	0	87
MS	0	0	3	4	3	33	0	12	0	55
WS	0	3	3	2	2	92	1	61	2	166
D	0	1	1	0	1	16	47	26	0	92
PS	0	5	4	0	0	28	2	101	0	140
BS	0	0	0	0	0	2	0	4	0	6
Precision	0.00	0.50	0.56	0.88	0.19	0.48	0.94	0.46	0.00	0.56
Recall	0.00	0.75	0.45	0.52	0.05	0.55	0.51	0.72	0.00	0.54
F1	0.00	0.60	0.50	0.65	0.08	0.51	0.66	0.56	0.00	0.52

Figure 4: Confusion matrix for the results shown in Figure 3. The last column (in blue) shows the total number of facies present in the geologists' interpretation. The diagonal (in red) gives the number of facies that were correctly classified by the machine. Off diagonal terms represent the miss-classifications of the machine. E.g, the number in the first row, second column indicates that the machine classified 14 samples as coarse siltstone (CSiS) whereas the geologist classified them as sandstone (SS). Precision is the ratio of true positives to all predicted positives. Recall is the ratio of true positives to all actual positives. The F1 metric weights recall and precision equally.

interesting to note that deep learning approaches do not dominate. Concerning the later observation, we believe that the poorer performances of deep learning is due to a lack of data, as the power of those algorithms comes at the cost of supplying a profusion of training examples. As for the poor match in terms of F1 score, Dubois et al. (2007) point out that the facies zones are more of a continuum rather than a clearly discrete sequence. This means that neighbour classes can be very similar, and a clear frontier does not exist. Moreover, human interpretation is non-unique and subject to errors, which is an additional challenge for geoscientific applications of machine learning as the ground truth is never know for certain.

As previously mentioned, geologists used visual observations of core samples as well as general knowledge about the area to determine their sequence. On the other hand the network was only given access to well data. Since the resolution at the log scale is lower than the resolution at the core scale, this alone should explain why the very thin layers labelled by the geologists are not always recovered by the algorithm. Besides, it is not obvious that the distinctive visual criteria chosen by the experts will also appear in our 7 logs. This could mean that the boundaries drawn by the geologists cannot be completely recovered by the machine, making the problem we want to solve ill-posed. As an example, the confusion matrix in Figure 4 highlights the difficulty encountered by the network to separate the coarse siltstone (CSiS) and the fine siltstone (FSiS). To distinguish between the two clastic sedimentary rocks, petrologist are measuring the grain size. Given the well logs available, it is unlikely that the machine can match the power resolution of the human eye even considering the differences in porosity. It would have been informative to have a sonic log in addition, as grain size, and so the softness of the rock, strongly influences the acoustic propagation.

An other factor to take into account is the proportion occupied by the different facies in the training and validation data. Figure 5 reveals that the algorithm saw almost seven times more examples of coarse sisltstones (CSiS) than dolomites (D). However, despite this, the confusion matrix indicates that classification performances for the dolomites reached an F1 score of 0.66 against 0.60 for the coarse siltstones. Therefore, it seems that the main problem is not the quantity of data but rather the resolution limitations discussed in the previous paragraph.

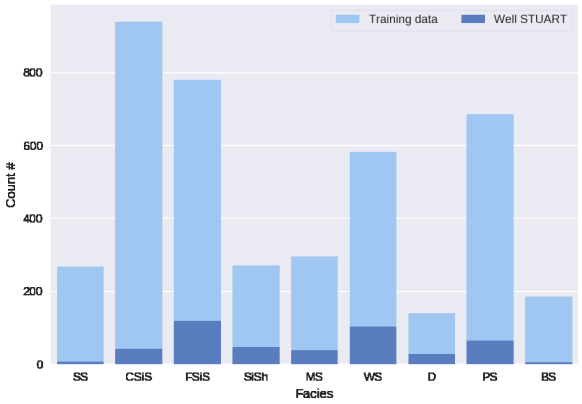


Figure 5: Bar-plot representing the total number of occurrences of each of the 9 facies for both the training data and the geologists interpretation for one of the blind wells (Stuart).

As noted again by Dubois et al. (2007), being able to correctly classify within one neighbour facies is just as good as being correct, and since sources of information given to the geologists and the machine were different, we believe that evaluating the success of the exercise in terms of F1 score alone is of little interest. The machine's classification is not in competition with the humans' one but should rather be evaluated as a complementary information. Discussions with geologists who know the reservoir will indicate whether the global machine interpretation is nevertheless interesting.

CONCLUSION

We proposed to train an inception network to determine the stratigraphy of a reservoir from well measurements. By extracting short sequences from the logs around examples of facies, the algorithm progressively learns a facies - dependent parametrization of the data. We evaluated the prediction performances on blind wells and provided visual and statistical information about the results. The machine's answer is satisfying in the first order but, being deprived of higher resolution core samples data and of additional well measurements, it failed at reproducing the work of the geologists with a high accuracy. In future work we will investigate a form of unsupervised classification called clustering. Instead of teaching our network to recognize the facies chosen by the geoscientists, we may give it the freedom to come up with its own classes and analyse the similarities and differences with the human interpretation. Additionally, it shall be interesting to work with hybrid networks which accept different types of data, from the core scale to the seismic scale (see e.g Wang and Carr (2012)).

REFERENCES

Bohling, G., and M. Dubois, 2003, An integrated application of neural network and markov chain techniques to the prediction of lithofacies from well logs.

Busch, J., W. Fortney, and L. Berry, 1987, Determination of lithology from well logs by statistical analysis: SPE (Society of Petroleum Engineers) Format. Eval.; (United States), **2:4**.

del Monte, A. A., 2015, Seismic petrophysics: Part 1: The Leading Edge, **34**, 440–442.

Dubois, M. K., G. C. Bohling, and S. Chakrabarti, 2007, Comparison of four approaches to a rock facies classification problem: Comput. Geosci., **33**, 599–617.

Hall, B., 2016, Facies classification using machine learning: The Leading Edge, **35**, 906–909.

LeCun, Y., Y. Bengio, and G. Hinton, 2015, Deep learning: Nature, **521**, 436–444.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner, 1998, Gradient-based learning applied to document recognition: Proceedings of the IEEE, **86**, 2278–2324.

Rabaute, A., 1998, Inferring a continuous lithology and mineralogy from multivariate statistical analysis of well-logging data : Examples from some sedimentary structures associated with tectonic plates convergence zones (ocean drilling program leg 134, 156 and 160): Institut des sciences de la terre, de l'eau et de l'espace de Montpellier, Université Montpellier II. Mémoires géosciences-Montpellier.

Srivastava, N., G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014, Dropout: a simple way to prevent neural networks from overfitting.: Journal of Machine Learning Research, **15**, 1929–1958.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, 2015, Going deeper with convolutions: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1–9.

Wang, G., and T. R. Carr, 2012, Methodology of organic-rich shale lithofacies identification and prediction: A case study from marcellus shale in the appalachian basin: Computers & Geosciences, **49**, 151–163.

Clustering from Seismic AVA Curves

Seismic amplitudes carry information about the physical properties of the sub-surface. Analysing their spatial distribution as well as their evolution with the incidence angle (AVA behaviour), yield hints about the lithology and fluids in presence (Simm et al., 2014). In early stages of the exploration process, well information may not be available and amplitude analysis serves as a scanning tool in charge of highlighting potentially interesting regions.

In this section, we look at a possible automatisation method of the amplitude screening using unsupervised clustering. As opposed to the classification approach that we presented in our work on well logs, clustering aims to group elements of a datasets together without the need of an external (manual) help. Those algorithms are based on some notion of similarity and optimize their parameters iteratively until a criteria of convergence is met.

We first perform a synthetic test by clustering AVA curves extracted at an horizon for data modelled with the Zoeppritz equations. We design a wedge model with variations in the shear wave velocities along inlines and variations of thickness along crosslines. We treat every curve independently and cluster them with the *K-mean* algorithm (Hartigan & Wong, 1979). Given a set of n curves $(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n)$ where each curve is of dimension d, the number of recorded incidence angles, the algorithm aims to partition the n observations into a set of k clusters $\mathbf{C} = \{C_1, C_2, ..., C_k\}$. The partitioning is done as to minimize the per-cluster variance, i.e. to solve:

$$\underset{\mathbf{C}}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \| \mathbf{x} - \boldsymbol{\mu}_i \|^2 \ , \tag{4.1}$$

where $\boldsymbol{\mu}_i$ is the mean of points in C_i. Given an initial position of the cluster's centres (centroids) $\boldsymbol{\mu}_1^{(0)}, \boldsymbol{\mu}_2^{(0)}, ..., \boldsymbol{\mu}_k^{(0)}$, a naive approach to optimize the clusters is to iteratively repeat the two following steps:

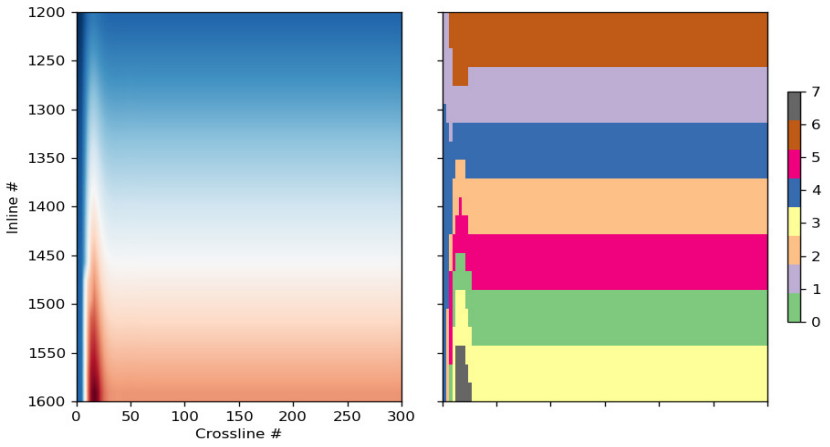
Assignment step: For each observation, compute the Euclidean distance with each centroid and assign it to the closest cluster.

Update step: Calculate the new centroids by taking the mean of the assigned observations:

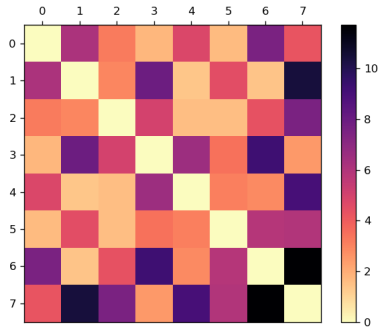
$$\boldsymbol{\mu}_i^{(t+1)} = \frac{1}{| C_i^{(t)} |} \sum_{\mathbf{x}_j \in C_i^{(t)}} \mathbf{x}_j \ , \tag{4.2}$$

where $|C_i^{(t)}|$ is the cardinality of the i^{th} cluster at step t.

The algorithm is not guaranteed to converge toward a global optimum, and the initialization of the centroids position has a strong impact on the results (Hartigan & Wong, 1979). We use the *kmean++* algorithm for initialization (Arthur & Vassilvitskii, 2007), that aims to find good initial positions by spreading out the initial centroids. The stopping criteria is either specified as a maximum number of iterations or as a tolerance over the average intra-cluster variance.



(a)



(b)

Figure 4.2 – Clustering results of the synthetic data for a choice of 8 clusters. (a) Image on the left is a stacked amplitude map extracted along the horizon. Image on the right displays the cluster assignments. (b) Pairwise Euclidean distance matrix between the centroids.

The two key parameters of the approach are the choice of the number of clusters and the preprocessing pipeline applied to the observations. For the synthetic data, we do not perform any special preprocessing other than standardizing the curves with the global amplitude mean and standard variation. We randomly select 20% of the curves to train the algorithm and

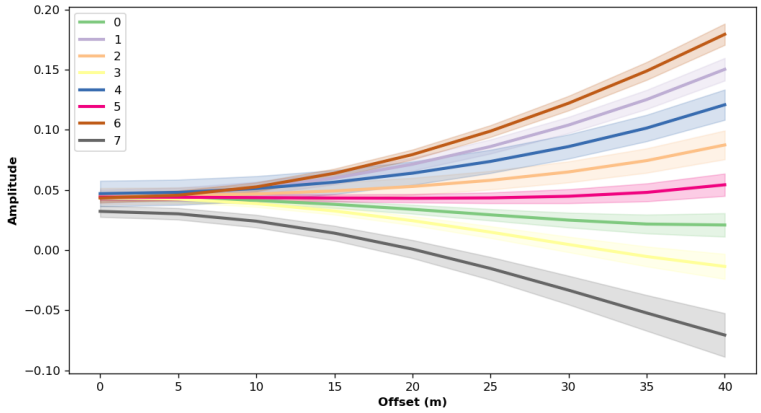
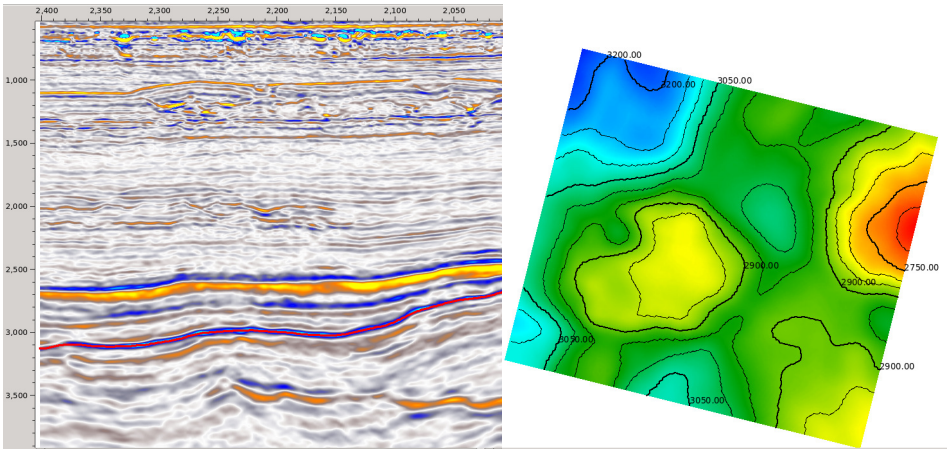


Figure 4.3 – AVA curves extract along the horizon shown in Figure 4.2 and grouped according to the clustering results. Coloured thick lines represent the average curve within each cluster and the semi-transparent area represent the standard deviation.

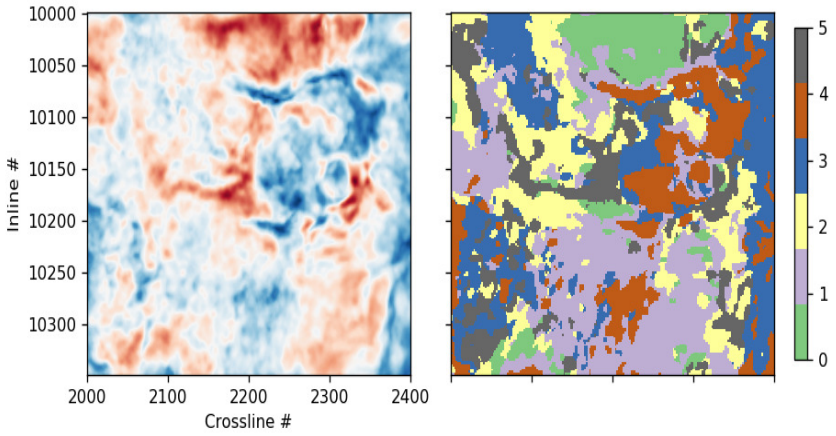
assign every curve to a cluster after convergence. We show the clustering results in Figure 4.2. We observe a continuous partitioning of the data along the inline direction, which indicates the sensitivity of the algorithm to the changes in shear velocities. The signal is more complex at small crossline indexes as the thickness of the reflector decreases and tuning interferences occur. However, we observe a certain consistency of the clustering results as the zones adapt to group together areas with similar average amplitudes. We also display in Figure 4.3 the average AVA curves for each cluster. We see that the AVA-gradient is the main information discovered by the algorithm to cluster the observations.



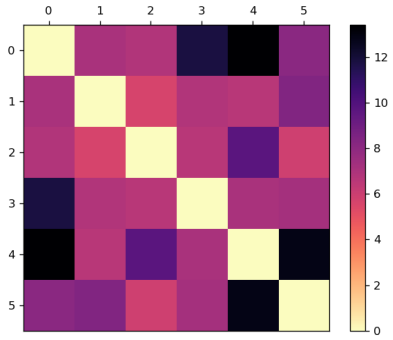
(a) (b)

Figure 4.4 – The BCU horizon mapped in the Volve dataset. (a) A slice at inline 10239. The BCU horizon is highlighted by the red curve. (b) Map view.

We now test the algorithm on curves extracted from the base Cretaceous unconformity (BCU) horizon of the Volve dataset[1] (see Figure 4.4). The worklow we employ is the same as for the synthetic data with the exception that we perform spatial Gaussian smoothing prior to clustering. This preprocessing step helps enforcing some spatial continuity of the predicted clusters. More advanced preprocessing, involving for instance the computing of a series of attributes, can be employed in order to help the algorithm extract more useful information (de Matos et al., 2006). We display the results for two choices of cluster numbers in Figures 4.5 and 4.6. There is a consistency of the results between both cluster numbers, and a first quality control allows us that the answer seems to be at least consistent in terms of average amplitude distribution.



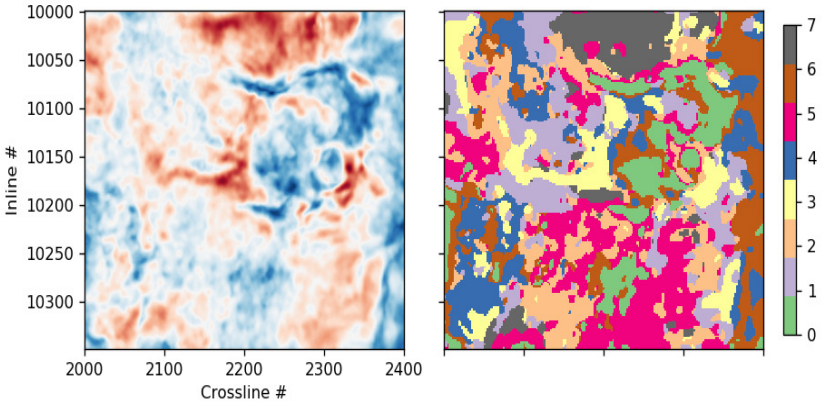
(a)



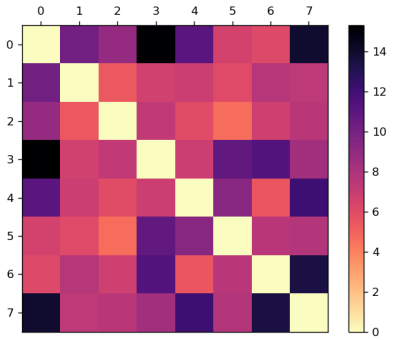
(b)

Figure 4.5 – Clustering results of the real data for a choice of 6 clusters. (a) Image on the left is a stacked amplitude map extracted along the horizon. Image on the right displays the cluster assignments. (b) Pairwise Euclidean distance matrix between the centroids.

[1] https://www.equinor.com/en/news/14jun2018-disclosing-volve-data.html



(a)



(b)

Figure 4.6 – Clustering results of the real data for a choice of 8 clusters. The display is similar to Figure 4.5.

Evaluating the results of a clustering algorithm is often more difficult than performing the clustering itself(Pfitzner et al., 2009). Those approach are explorative and no method come with theoretical guaranties of optimality. Is is usually up to the user to try different configurations and to judge the results (Estivill-Castro, 2002). For instance, without clear external information about the area to constrain the problem, choosing the correct number of clusters is not obvious. Some methods, like hierarchical clustering (Bishop, 2006), do not have a number of cluster parameter and find one on their own. This is often stated as a strength of those methods, however, those come with other hyper-parameters that have an indirect influence on the number of clusters found. For this reason, we believe that K-mean is a good choice as choosing explicitly the number of clusters facilitates the search for the best parametrization. Also, many quantitative criteria exist in order to automatically tune for the best hyper-parameters, but a subjective human evaluation is often needed for real world

applications (Feldman & Sanger, 2007).

Our present approach remains a bit simplistic and more advanced worklows are designed by, for instance, helping the clustering algorithm by feeding it with custom attributes that practitioners believe to be of importance (de Matos et al., 2006). Overall, for seismic applications, it is important that the clustering algorithm be fast and stable to initial conditions in order to facilitate the exploration by the interpreter. We also like algorithms whose parameters have a clear effect on the results as this make their tuning far easier. Together, we believe that seismic clustering methods are only useful so long as the practitioners are able to interpret and judge the results.

Discussion and Outlook

The initial goal of the Section was to develop a workflow for automatically partitioning a dataset into distinctive regions by resorting to both seismic data and well measurements. Unfortunately, the key challenge was to get access to a real dataset with a good quality migrated prestack seismic as well as a number of (calibrated) well logs, together with an in depth interpretation of the data done by experts of the region in order to allow for a serious discussion of the machine's results.

Nevertheless, we studied parts of the problem and already identified some of the main challenges associated with clustering analysis of geophysical data. One of the main point is the difficulty to give a robust qualitative evaluation of the results. Even in the case of a supervised training problem we point out in Tschannen et al. (2017) that a pure data scientist's approach, that does not take into account the specificities of geophysical data due to uncertainties and subjectivity, is not suitable. We also discuss that despite the simplicity of creating an unsupervised clustering workflow to partition seismic data into litho/fluid facies, the need for a subjective evaluation by experts should not be overlooked.

In future work we hope to be working on a rich dataset in order to link information from various geophysical and geological sources and have a look at inversion problems with deep learning.

4.3.3 Unsupervised Deep Learning

Introduction and Method

One of the earliest and simplest way developed to train a neural network without the need for additional meta-information about the data, is to set-up a compression/decompression scheme. As illustrated in Figure 4.7, the idea is to use two networks, one to compress the data and the other to decompress it. Both networks are trained together in order to minimize the decompression error. This is called an auto-encoder (Vincent et al., 2010) and we see that no external information about the data is needed in order to train the network. It is important to choose an architecture that yields an appropriate size for the latent space. If the compression factor is too small, the auto-encoder may learn a trivial identity mapping that will prove useless for further applications. If the compression factor is too big, the network might not be able to recover all of the information. Besides data compression, auto-encoders are also employed for denoising tasks (Vincent et al., 2010) or to pre-train networks before performing a supervised training in the case where only few labelled examples are available (Erhan et al., 2010).

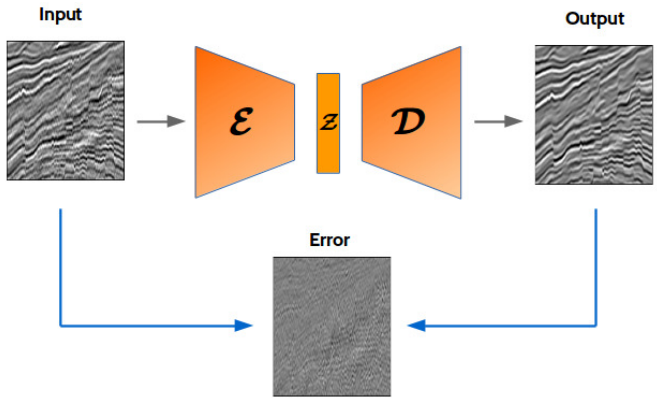


Figure 4.7 – Schematic representation of an auto-encoder (Vincent et al., 2010). Input data is first sent trough an encoder network $\mathscr{E}$ that decreases its spatial shape. It's most compressed form, $\boldsymbol{z}$, is called the latent vector. The second part is a decoder $\mathscr{D}$ that up-samples the data to its original shape. Both parts of the network are trained together in order to minimize the reconstruction error.

In their work, Kingma & Welling (2013) proposed to add an additional constrain on the way the latent representation is learned by re-formulating the problem using variational Bayesian ideas. At the cost of an extra term in the loss function, variational auto-encoders (vae) are

regularized to learn a continuous latent space. This brings interesting properties, in particular, two input samples that are similar in the image domain tend to be mapped to the same region of the latent space and all points sampled from the latent space tend to produce realistic samples when decompressed by the decoder. As an example we show in Figure 4.8 the effect of generating (i.e. transforming latent vectors using the decoder) images by progressively moving through the latent space. We see that the network discovered properties such as the orientation of the digits, and can also smoothly interpolate between distinct shapes.

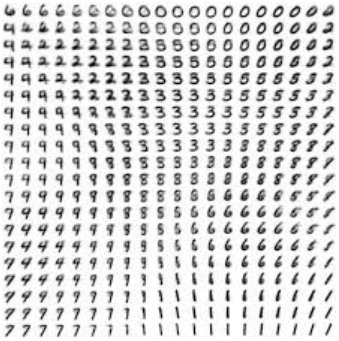


Figure 4.8 – Images generated from the 2-dimensional latent space of a variational auto-encoder trained on the mnist dataset (LeCun et al., 1989). Taken from Kingma & Welling (2013).

Experiment

In this work, we design a 2D convolutional vae. The encoder is composed of two residual blocks (He et al., 2016) and down-samples the data by a total factor of 4. We extract 350 samples from a stacked dataset in the crossline and time direction with a shape of 96×96 pixels, and train the network for 600 epochs with a momentum SGD optimizer. We use the mean squared error in the image space and follow Kingma & Welling (2013) and regularize the latent variables to be normally distributed. In order to regularize the training we use batch-normalization and we randomly perturb the input samples by adding white noise and performing Gaussian blurring. A key hyper-parameter is the trade-off coefficient between the reconstruction error and the latent space constraint. To tune it, one can start with a very small value. This essentially reverts the training setting to the one of a traditional auto-encoder. Once one is able to get a good reconstruction, one can progressively increase the value until one start to observe well behaving generative properties.

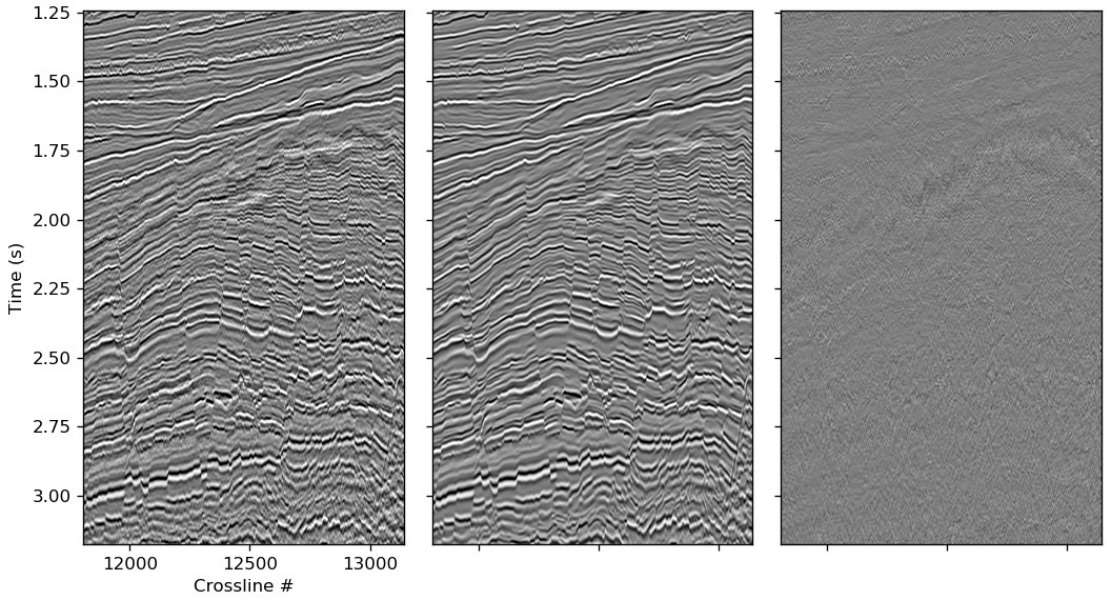


Figure 4.9 – Results of the vae on an inline away from the training area. The image on the left is the original seismic, the central image is the output of the network and the image on the right is the difference between both.

We show in Figures 4.9 and 4.10 the results of passing an inline through the trained network. Because it is much easier to learn structured signal than noisy, non-structured, signal, we observe that the network naturally acts as a denoising algorithm. It seems that most of reflections are correctly preserved and the network didn't blur reflections across faults. In Figure 4.11 we evaluate the generative performance of the network. To do so, we select two input samples x_a and x_b from the seismic dataset. The encoder yields their respective latent representation z_a and z_b and the decoder their reconstructions $\tilde{x}_a$ and $\tilde{x}_b$. Since the latent space is supposed to be continuous and complete, we can linearly interpolate between the points z_a and z_b and observe the effect in the image space using the decoder. In practice, we construct a set of variables $z = (1-\alpha)z_a + \alpha z_b$ for a range of values such that $0 \leq \alpha \leq 1$. We see that the decoder can generate samples that do not exist in the original data but that look fairly realistic. This visual quality control serves to judge whether the training can be considered successful of not.

We also display in Figure 4.12 the results of the activation maximization technique proposed by Erhan et al. (2009) to try to understand the sort of concepts learned by the different parts of the network. To generate those images, we start with random noise in the image domain

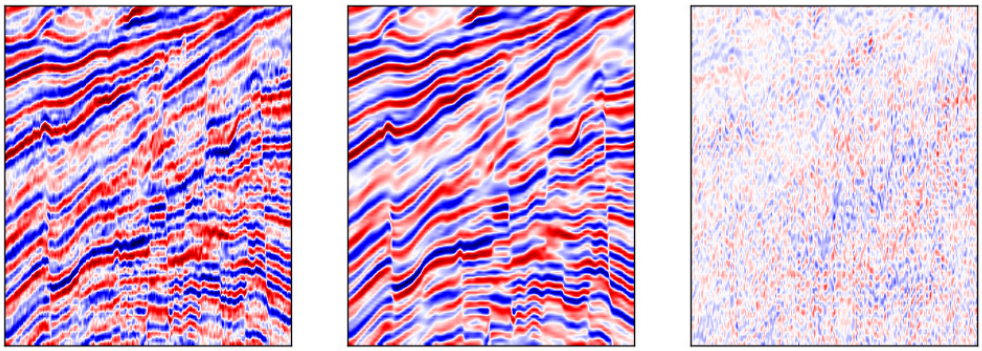


Figure 4.10 – A zoom of the vae results shown in Figure 4.9.

and iteratively update it by back propagation of the gradients computed from a loss function asking to maximize the activation of a certain unit of the network. The textures seen here are rather abstract but suggests that the network did start to learn some concepts from the data.

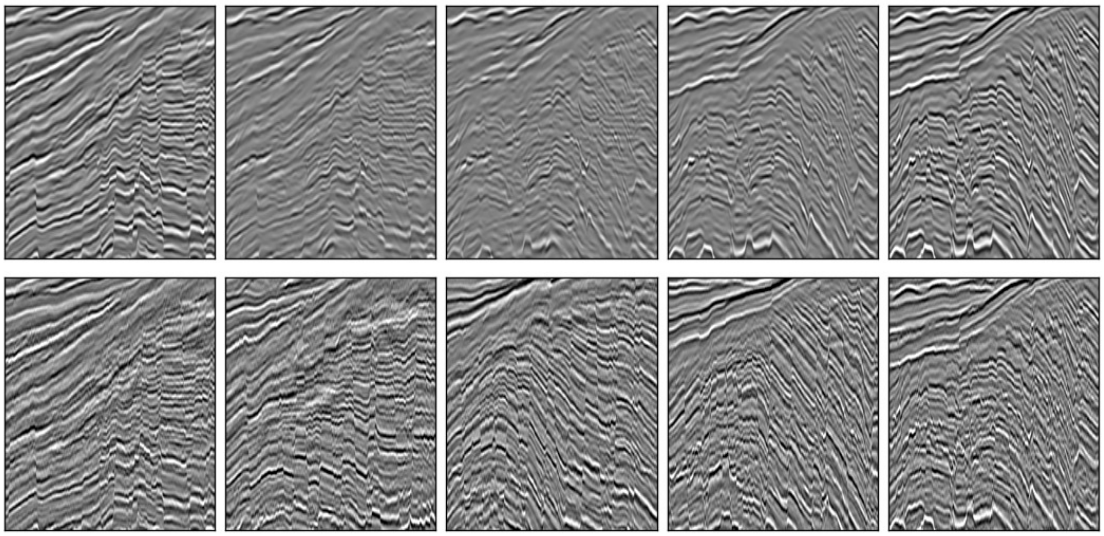


Figure 4.11 – Example of the generative performance of the network. Upper part: images on the far right and left are obtained by decoding real input samples. Central images are obtained by decoding a latent variable linearly interpolated between the latent representation of the real samples. Lower part: real samples taken from the data for comparison. (Best seen as a GIF animation.)

Discussion

While the denoising capability of the network is interesting in itself, variational auto-encoders (and other self-supervised methods) have potential for more applications. Once trained, the

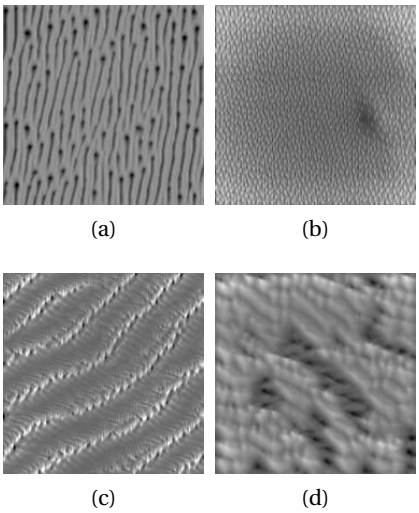


Figure 4.12 – Visualization of the preferred input for some convolutional kernels of the network. Computed after Erhan et al. (2009) (see Figure 3.1 for more details).

encoder may for instance serve as a feature extractor in other applications. In future work, we plan for instance to replace the first half of the network that we use to map horizon surfaces (Section 4.3.1) to see whether this helps to accelerate the convergence. This application would be of interest since our horizon picking method is done with few training examples only and suffers from a slow start. From the generative side, one may imagine applications to, for instance, remove or enhance certain form of coherent signal. In their work, (Burgess et al., 2018) showed on toy examples that a vae could be trained to learn a disentangled latent space. This means that every dimensions of the space can be associated with a clear concept (such as orientation or color). It then becomes possible to manipulate certain characteristics of the data by modifying their latent representation in selected dimensions. As a loose example, Figure 4.13 shows that one dimension of the latent space is mostly sensitive to compact wavefronts associated with near vertical events at the faults.

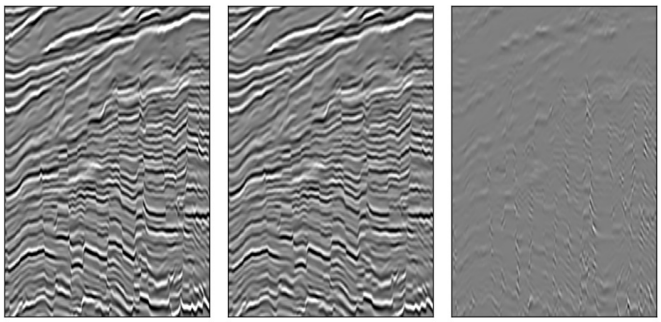


Figure 4.13 – Probing the latent space by muting some of the dimensions. The image on the left is the sample recovered from the full latent vector. The central image was obtained after setting to 0 one of the dimension of the latent vector. The image one the right is the difference between both.

CHAPTER 5 APPLICATIONS TO SEISMIC PROCESSING

5.1 Résumé

Dans ce chapitre, nous nous intéressons aux méthodes de traitement destinées à améliorer la qualité du signal afin de réduire les incertitudes dans l'interprétation. Ces approches nécessitent une attention particulière puisqu'elles transforment la donnée elle même, et par conséquent elles ont une influence sur la totalité des étapes suivantes. Deux difficultés principales naissent de l'utilisation de l'apprentissage machine pour le traitement sismique. Premièrement, comme à chaque fois, il s'agit de trouver des moyens d'apprendre aux algorithmes. En particulier, leur apprendre à distinguer le signal du bruit et à opérer des transformations bénéfiques. De plus, puisque l'utilisateur n'a que peu de contrôle sur la façon d'opérer de la machine, il faut être capable de s'assurer qu'aucun signal important n'a été endommagé ou artificiellement modifié. Cette tâche peut s'avérer difficile devant la grande quantité de données ainsi qu'à cause des incertitudes présentes.

Dans la suite, nous étudions trois applications possibles et discutons de leurs avantages et limitations pratiques. Premièrement, nous décrivons un procédé d'apprentissage non supervisé, basé sur la décomposition en éléments principaux, afin d'atténuer le bruit décorrélé présent dans les données. Dans une deuxième partie, nous nous intéressons au problème d'égalisation spectrale qui permet de comparer entre eux deux jeux de données issues de la même région et enregistrées à des dates différentes. Nous utilisons l'apprentissage profond supervisé et étudions en particulier le cas de l'extrapolation des hautes fréquences. Enfin, nous proposons une méthode de traitement de bout-en-bout des données sismiques avant sommation au moyen de l'apprentissage profond supervisé.

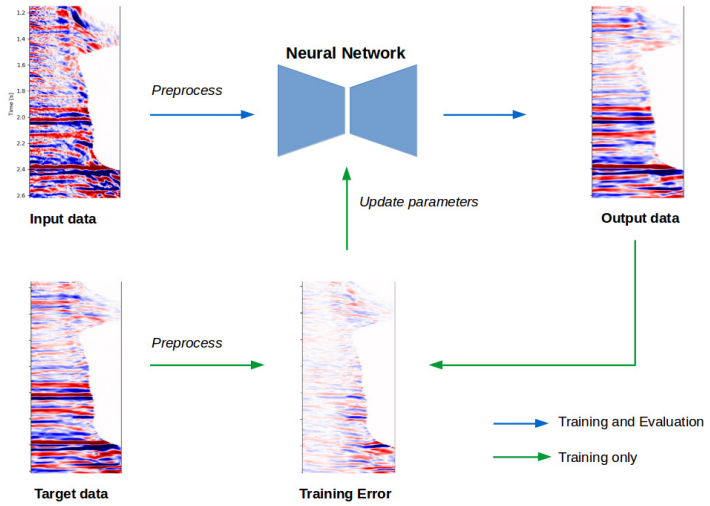


Figure 5.1 – Typical workflow employed to transform data using deep learning. Arrows represent the direction of the data flow.

5.2 Overview

In this chapter we consider seismic processing techniques applied to improve the quality of the data and reduce uncertainties in the interpretation. In particular we focus on post-migration processing of gathers and stacks. Those applications fall into the regression category. Rather than classifying elements of the data into a set of discreet categories like we present in Chapters 3 and 4, the goal here is to transform the seismic amplitudes themselves. These applications therefore require special care since transforming the data will affect all the following processing and interpretation steps. Unlike with traditional algorithms where it is possible to explicitly constrain and parametrize algorithms in order to avoid damaging useful signal, one have a less direct control over transformations done by a learning algorithm. On the other hand, the potential of machine learning is in theory much greater than with traditional processing methods, and better or even new ways to process the data may be made possible.

As often with machine learning, the main challenge is to find ways to train the algorithms. In order to process the seismic data, one must indicate to the network which part of the signal should be removed, which part should be transformed (and how it should be transformed) and which part should be left untouched. We display in Figure 5.1 a typical workflow involving the use of a neural network to process data. We present in this chapter different case studies

and discuss their practicality.

In Section 5.3.1, we present a workflow that could be categorized as unsupervised learning in order to remove random noise from the data. By using simple heuristics, we let the algorithm learn a transform domain in which seismic events have a locally sparse representation. Since it is spatially weakly correlated, random noise will not have a sparse representation in the transformed domain and this is what can be used in order to isolate it from the important signal. We show that the method is effective, but we also point-out some important limitations, such as the fact that noise on real data is rarely fully decorrelated.

In Section 5.3.2 we look at the spectral matching problem. We consider the case where two different seismic datasets are acquired over the same region, and one of them must be transformed in order to account for differences in the acquisition and processing. We treat it as a supervised learning problem by providing pairs of examples from both datasets and ask the network to process the first sample in order to be more similar to the other. In particular we look at the case where high frequencies must be boosted or even extrapolated.

In Section 5.3.3 we study the benefits and drawbacks of deep learning to perform gather conditioning. We treat the problem as a supervised learning task and create pairs of noisy/denoised examples either using traditional denoising workflows or by resorting to synthetic modelling.

Neural networks have the possibility to outperform traditional algorithms, or even allow for new processing methods (such as super resolution (Wang et al., 2019b)). They also offer the promise to process data at a fraction of the manual and computing time required by traditional algorithms. However, we observe that finding good ways to train them is difficult. In particular it is challenging to account for all of the possible variations in the data due to vast possibilities in the underlying geology as well as in the pre-processing sequences, and therefore it is difficult to guaranty optimal performances when applied to new data.

5.3 Applications

5.3.1 Denoising with Pyramidal Kernel PCA

This work was submitted to the journal *Geophysics* and reviewers asked for revisions that we have not yet provided. It proposes a method to attenuate random noise in the data. The end results are similar to the ones we achieved with a variational auto-encoder (vae, Section 4.3.3), but this approach does not make use of deep learning. I did this work at the beginning of my thesis, and a comparison with the vae method illustrates how effective can deep learning be since the vae method was far simpler to implement and test and provides better results.

Introduction

Seismic noise appears both as coherent events which we do not wish to account for and as incoherent ambient energy. The later is produced by the superposition of a variety of unrelated sources, usually spatially distributed and continuous in time, such as wind, ground roll and human activities. In this work, we consider the problem of removing incoherent noise from a seismic stack. We use a discreet wavelet transform (DWT) to perform a time-frequency decomposition of the data and iteratively attenuate the noise through the successive sub-bands using (kernel) principal component analysis (PCA) denoising.

A common approach for denoising is to seek for a transform domain in which the signal and noise can be more easily separated than in the original spatial representation. In particular, sparse decomposition techniques take advantage of the redundancies in a signal to find a lower dimensional subspace spanned by a small number of elements designed to capture most of the data features. Analytical transforms, such as the curvelet transform (Candes & Donoho, 2000), are a very popular choice to decompose the data. In order to accurately match seismic wavefronts at different frequencies and dips, those transforms are usually designed to be anisotropic, multiscale and multidirectional (Donoho, 1993, Hennenfent & Herrmann, 2006). But despite the success of those transforms, researchers have shifted their attention toward data-driven approaches. Instead of relying on a pre-chosen dictionary of basis functions, those approaches attempt to learn the dictionary from examples (Olshausen & Field, 1997). Classic

algorithms are the K-singular value decomposition (K-SVD) (Aharon et al., 2006) or principal component analysis (PCA) (Muresan & Parks, 2003). As the dictionary elements adapt to the structure of the data they usually out-perform wavelet based methods in denoising tasks and are less likely to introduce artefacts (Elad & Aharon, 2006, Mairal et al., 2008). Several such algorithms have already been applied to seismic data: Tang et al. (2012) propose to solve an overdetermined non-linear dictionary learning problem combined with a total variation minimization to achieve the denoising. In Beckouche & Ma (2014) the dictionary is learnt using a variational sparse-representation model while Zhu et al. (2015) remove low frequency noise by combining a sparse K-SVD algorithm with a basis pursuit denoising problem.

The good performance of the previously mentioned methods come at the price of greater computation expenses and the difficulty in tuning the parameters. They require to solve non-linear and sometimes non-convex optimization problems, making the task delicate when applied to real sized seismic data. In this paper, we propose a simple algorithm based on an iterative PCA threshold through a multiscale pyramidal decomposition of the data. We use an isotropic discrete wavelet packet transform (Wickerhauser, 1991) to decompose the noisy seismic into different time-frequency sub-bands (Fig.5.2), and iteratively learn new dictionaries as we go deeper in the pyramid (Fig.5.3). In addition of being well adapted to large datasets, this approach is also particularly efficient when the noise's frequency spectrum fully overlaps the spectrum of the signal. We also extend our algorithm by substituting the PCA by it's non-linear counterpart, the kernel PCA (Schölkopf et al., 1998). We demonstrate in a controlled experiment on field data the efficiency of the method in removing noise while preserving both the fine and coarse structures in the stack.

Patch based learning

In order to differentiate between signal and noise, the learned basis should be sensitive to coherent events and leave the incoherent ones aside. Rather than resorting to a global approach, modern denoising algorithms are comparing group of pixels against each other. By working on small data patches we obtain a more robust estimation of coherent features as we take advantage from the redundancy of the signal and thus are able to represent the data from a reasonably sized dictionary. We follow the patch based dictionary learning methods which

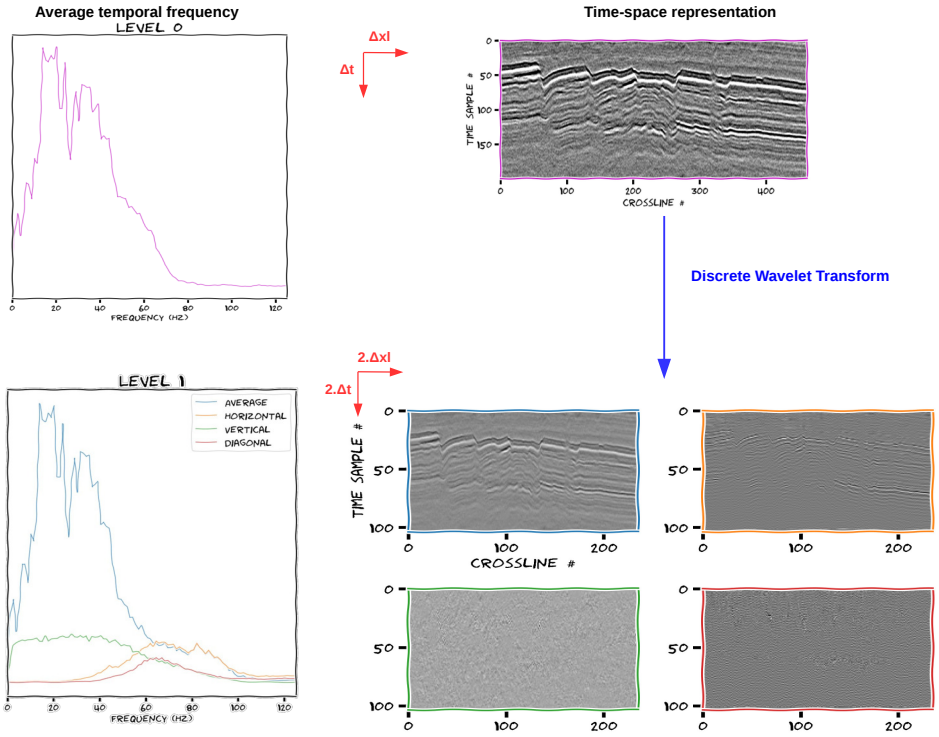


Figure 5.2 – Pyramidal denoising strategy. Using a discrete wavelet transform we iteratively decompose the image from the level 0 (original time - space domain, in purple) to deeper levels where each sub-band has a restricted frequency - wavenumber support. At each level, we independently denoise the sub-bands using (kernel) PCA learning (as explained in Figure 5.3) before going to the next iteration. At level 1, the average sub-band (in blue) is obtained by convolution of the low-pass filter both along time and crosslines (LL). Similarly, the horizontal sub-band (in orange) is obtained by convolution of H along time and L along crosslines (HL), the vertical sub-band (in green) by convolution LH and the diagonal sub-band (in red) by convolution HH.

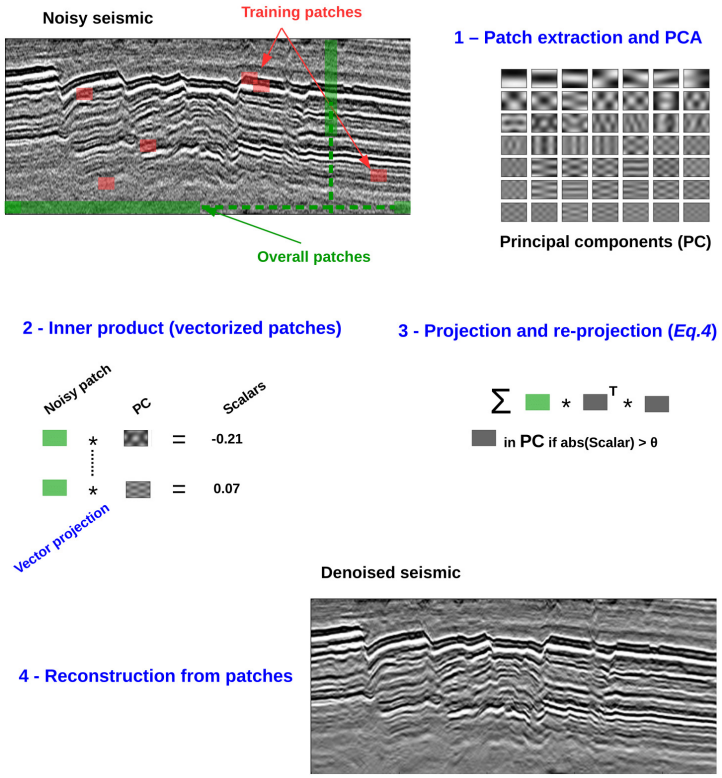


Figure 5.3 – Patch based PCA denoising. For the first step, using a random selection of overlapping patches (in red), we form a dictionary from their principal components. In a second step we decompose the entire image into overlapping patches (in green). For a given patch we compute its inner product with respect to the dictionary elements. Third, we select a sub-dictionary by keeping only the principal components that are prominent (i.e whose inner product is greater than a threshold θ in absolute value) for the description of the patch. The denoising is achieved by successively transforming it back and forth in the sub-space spanned by the reduced dictionary. Fourth, we repeat step 2 and 3 for every green patches and reconstruct the denoised image by arithmetically averaging pixels in overlapping area.

works in two steps. First, for the training phase, we uniform randomly select a given number of possibly overlapping samples and learn a basis able to represent every coherent event they contain. The number of training samples should be large enough to account for all the features present in the data, but also kept minimal to decrease compute time. Second, for the reduction phase, we then decompose the entire image into overlapping patches. Independently for each of them, the algorithm determines which of the basis components are relevant for representing the signal and we assume that the remaining ones are mostly modelling the noise. By successively projecting the samples onto the sub-space spanned by the relevant basis components and re-projecting it onto the original space we achieve the denoising. A typical overlap of 60% is used for the patches. This means that pixels are denoised multiple times, and we use the arithmetic average as the final result, which helps to reduce dictionary artefacts. In the 2D case, we use patches of shape $\sqrt{n} \times \sqrt{n}$. We extract ℓ training samples $\mathbf{x}_p \; for \; p \in [1, \ell]$, and vectorize them to the shape $n \times 1$, where n is the number of pixels per patch. We note by $\mathscr{X} = \{\mathbf{x}_p \in \mathbb{R}^n\}_{p=1,..,\ell}$ the training set. Input patches are standardized for a better conditioning of the learning problem. By subtracting the mean value and dividing by the standard deviation of every patch samples, we make sure that the learning process won't be artificially dominated by an area of strong reflectivity while diminishing the effect of weaker but as much important reflectors.

Principal Component Analysis

For the first phase we consider each of the training patches $\mathbf{x} \in \mathscr{X}$ as an independent observation of the image. These observations have a degree of variability n equal to the number of pixels that compose them. We use Principal Component Analysis (PCA) to learn the basis over the training set. PCA will transform the observations into an orthogonal space whose axes, called the principal components, are maximizing the variance of the data. In this way, the first principal component will point in the direction accounting for as much variability as possible among the training samples, the second one will also point in the direction of maximum variance under the constraint of being orthogonal to the first one, and so on. As seismic data is mostly formed by near horizontal reflectors, we would expect that the most commonly observed amplitude variation among patches is a pick-through (or through-pick)

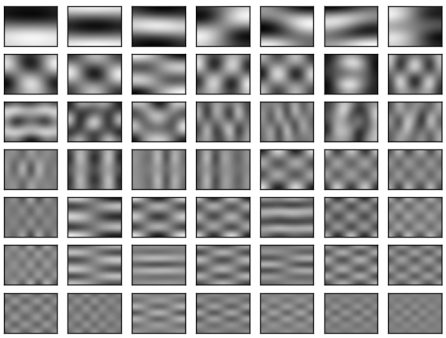


Figure 5.4 – Principal components **V** learned by PCA at the level-0 of the pyramid. The components are sorted by decreasing eigenvalues from top-left to bottom-right.

transition along the time axis (see Fig 5.4). We write as $\bar{\mathbf{x}} = \frac{1}{\ell}\sum_{p=1}^{\ell}\mathbf{x}_p$ the arithmetical average of the training patches and compute their $n \times n$ centred empirical covariance matrix:

$$\Sigma = \frac{1}{\ell}\sum_{p=1}^{\ell}\mathbf{x}_p\mathbf{x}_p^T - \bar{\mathbf{x}}\bar{\mathbf{x}}^T, \tag{5.1}$$

where T denotes the transpose operator. By construction, the covariance matrix is symmetric positive semi-definite. The eigendecomposition of Σ consists in solving the equation $\lambda\mathbf{V} = \Sigma\mathbf{V}$. We obtain a set of tuples $(\lambda_p, \mathbf{v}_p)$, $p \in [1, n]$. The eigenvalues λ_p are positive scalars sorted in decreasing order, and the associated eigenvectors $\mathbf{v}_p \in \mathbb{R}^{n\times 1}$ are the principal axes forming an orthonormal basis. We can now express each of our patches as a linear combination of the principal components:

$$\mathbf{x} = \bar{\mathbf{x}} + \sum_{p=1}^{n}\left\langle \mathbf{x} - \bar{\mathbf{x}}, \mathbf{v}_p \right\rangle \mathbf{v}_p, \tag{5.2}$$

where $\langle ., . \rangle$ is the inner product.

As seismic reflection signal is redundant, only a fraction of the principal components are necessary to represent most of its continuous events, edges and textures. On the other hand, random noise has a minimal co-variance between the observations and is therefore uniformly spread over all the directions of the eigen-basis. In order to remove the unwanted noise we need to identify the principal components required to well represent the seismic observations.

We decompose the entire seismic image into overlapping patches of the same shape as the training patches. We call $\mathscr{A} = \left\{\mathbf{x}_p \in \mathbb{R}^n\right\}_{p=1,..,\ell'}$ the set of all patches. Their number $\ell' > \ell$ depends on the ratio between the patches shape and the image shape and the amount of overlap between patches. The method assumes that the eigenbasis learned over the observations

in X will also be adequate to represent the remaining patches in $\mathscr{A} \setminus \mathscr{X}$. For this reason it is important to use a sufficiently large amount of training data. In the next section we also discuss image partitioning methods for improving the accuracy.

We denote by $\bar{\lambda_p} = \frac{\lambda_p}{\sum_{i=1}^{n} \lambda_i}$ the amount of variance explained by the p^{th} principal components for our training data. Using this quantity resort to a threshold function to drop the eigenvalues which are not necessary to model the coherent signal of a given data sample. Given a sample $\mathbf{x} \in \mathscr{A}$, the importance of the p^{th} principal component for its representation in the eigenbasis is given by the magnitude of the coefficient $\langle \mathbf{x} - \bar{\mathbf{x}}, \mathbf{v}_p \rangle$. We follow Deledalle et al. (2011) and apply a hard threshold θ using the shrinkage function η:

$$\eta(\mathrm{S}) = \mathrm{S}.\Gamma(|\mathrm{S}| \succ \theta), \tag{5.3}$$

an element wise product between some tensor S and a mask Γ whose elements are 1 where entries of S (in absolute value) are greater than θ, zero otherwise. Hence, the denoised version $\tilde{\mathbf{x}}$ of $\mathbf{x}$ is obtained by the successive projection into the reduced sub-space spanned by the relevant principal components and re-projection to the image space:

$$\tilde{\mathbf{x}} = \bar{\mathbf{x}} + \sum_{p=1}^{n} \eta(\langle \mathbf{x} - \bar{\mathbf{x}}, \mathbf{v}_p \rangle) \mathbf{v}_p \tag{5.4}$$

The computational bottleneck in PCA comes from solving the eigendecomposition of the $n \times n$ covariance matrix. Due to the orthogonality constraint, this decomposition remains fast for small n. This partly motivates our strategy of time-frequency decomposition (Fig 5.2) as it enables us to efficiently represent the data from a few number of small training patches by learning several dictionaries at different time-scales.

The main parameter of the PCA denoising is the value of the threshold θ. It is possible to set it empirically, but we rather employ an automated method designed by Zhu & Milanfar (2010). For the case of additive random noise, the authors propose a grid search based on a simple criteria which doesn't require prior knowledge about the noise distribution and favours the preservation of the high frequencies in the signal.

Isotropic Wavelet Packet Decomposition

The success of dictionary methods relies on employing a basis that is able to sparsely represent all features present in the seismic. Yet, if we learn one dictionary for the entire stack it is likely that fine and localized details will not have a sparse representation in this basis or might not even be fully represented. Thus, it is recommended to learn several dictionaries over local regions of the image. Muresan & Parks (2003) perform the PCA denoising inside a sliding square window while Deledalle et al. (2011) use a hierarchical approach where an hybrid basis is computed from global to progressively more local features using a space partitioning technique. Although the computation cost grows linearly with the number of dictionaries to learn, such approaches have been shown to bring significant improvements and are therefore worth the extra effort.

In our work, we employ a wavelet packet decomposition (WPD) (Wickerhauser, 1991) to analyse the seismic data at different time - frequency resolutions. The method splits the input data into sub-bands each with a distinct frequency-wavenumber support. In practice, the 2 dimensional WPD is computed in a cascaded manner by successively passing the image rows and columns through a high pass and a low pass quadrature mirror filters followed by a down-sampling operation (Mallat, 1999). At the first level of the decomposition, applying the high pass (H) and the low pass (L) filters along rows and columns yields to 4 possible sub-bands. The sub-band obtained by low-pass filtering along time and crosslines is said to contain the average details ($LL = a$), high-pass filtering along time and crossline contains the diagonal details ($HH = d$), high-pass along time and low-pass along crosslines the horizontal details ($HL = h$) and low-pass along time and high-pass along crosslines the vertical details ($LH = v$). Indeed, if we apply a high-pass filter along time and low-pass filter along the crosslines, horizontal reflectors will be preserved while nearly vertical ones will be filtered. For the second level of the decomposition, we repeat the procedure for each of the 4 sub-bands, yielding to a total of $2^{2^2} = 16$ sub-bands. Following the naming convention, the sub-band obtained from the original stack by applying only low-pass filters will be denoted LL, $LL = aa$.

Since each sub-band is made of details grouped by frequency/wavenumber category, the features they contain have a stronger correlation than the features in the original image. WPD

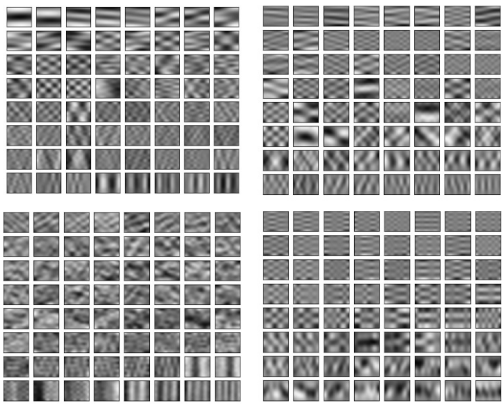


Figure 5.5 – Principal components learned by PCA at the level-1 (a, h, v, d) of the pyramid.

being an orthogonal transform, the band-limited white noise is nearly evenly split between all sub-bands (Donoho, 1993). Additionally, as the decomposition level d increases, from the sub-sampling operation, sub-bands size are decreasing by a factor 4^d (8^d in three dimensions) and the computational burden is hence greatly diminished. Finally as the time and spatial sampling are successively multiplied by two, our constant patch shape will progressively represent a larger area of the seismic, going from local to global analysis.

We propose to denoise the seismic image successively at each level of the pyramidal decomposition. At the level 0, training patches will represent a very small area of the global image, hence focusing on high time/space resolution features. We then compute the first level sub-bands from the image denoised at level 0. We denoise each sub-bands independently. Training patches now cover an area twice as large in time and in crosslines and the features present are better correlated. We stop at the third level of the pyramid, denoise each of the 16 sub-bands and compute the inverse WPD to retrieve the final denoised image.

Kernel PCA

The orthogonality constraint in the PCA algorithm allows only a linear separation between signal and noise. Therefore, we also evaluate its non-linear extension called Kernel PCA (Schölkopf et al., 1998, Mika et al., 1999) to investigate if we can further improve results. Using a non-linear mapping function, the method first transforms the data in a high dimensional space and then performs the linear separation problem in this space. For an appropriate

choice non-linear transform, it is expected that non-linearly separable data in the image space will become linearly separable in the high dimensional feature space. The details of the method are described in Schölkopf et al. (1998). Using what's know as the *kernel trick*, the method gives us access to the projection of an observation $\mathbf{x} \in \mathscr{X}$ on the i^{th} principal component computed in the high dimensional-space without the need to perform explicit computations in this space (Mika et al., 1999). We can therefore apply a hard threshold in the transformed domain by dropping elements with a small projection score. Denoising in the feature space is achieved by an automated selection of a hard threshold. To reconstruct the denoised image we must find the inverse transform to go from the feature space to the image space. This is know as the pre-image problem (Mika et al., 1999). As the transform is non-linear and usually non-invertible, mapping from a low-dimensional space to a high dimensional space, the problem seems difficult to solve. However, Bakir et al. (2003) propose a simple and stable approach by learning the inverse transform solving a kernel ridge regression problem.

Many types of non-linear functions are available and selecting one often brakes down to empirical testing. Our implementation of the denoising algorithm relies on the kernel pca toolkit from `scikit-learn` (Pedregosa et al., 2011), which offers several kernels including polynomial, Laplacian and radial-basis functions (rbf). In our experiment we achieved the best results with the rbf kernel (based on a least-squares distance) and the Laplacian kernel (based on the ℓ_1 distance). Both depend on a single parameter γ and since the rbf kernel is faster to compute, this is the one we retained. Given two points $\mathbf{x}_i, \mathbf{x}_j$ in the image space, the rbf kernel will compute their similarity along a Gaussian curve:

$$k(\mathbf{x}_i, \mathbf{x}_j) = exp(-\gamma \left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2) \tag{5.5}$$

We extend the automated parameter selection of Zhu & Milanfar (2010) to loop over both the threshold and the weighting parameter γ.

Experiments and Discussion

In our experiment, we consider the signal to be any event that has a spatio-temporal coherency (such as primary and multiple energy) and assume that the noise can be modelled by a band-

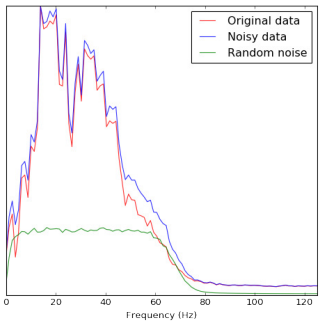


Figure 5.6 – Average normalized amplitude spectrum of the stacks in Figure 5.7, with band-limited gaussian spectrum.

limited normal distribution. Figure 5.7 shows a $0.8s$ marine stack sampled at $4ms$ with 462 crosslines spaced every $12.5m$. In Figure 5.7 we artificially generated broadband Gaussian noise to analyse denoising results both visually and in term of pick signal-to-noise (PSNR) ratio (Welstead, 1999). The average amplitude spectrum of the data and the noise are ploted in Figure 5.6. We compare our algorithm using PCA and kernel PCA with curvelets denosing (Candes et al., 2006) and non-local mean filtering (Buades et al., 2005, Bonar & Sacchi, 2012). We used constant size patches of 7×7 pixels through the pyramid and a biorthogonal 4.4 wavelet (Cohen et al., 1992) with the package `PyWavelets` for the DWP transform.

In Figures 5.4 and 5.5 we display the learned principal components at levels 0 and 1 of the pyramid. In the case of kernel PCA, we do not have access to the eigenvectors but rather to the projection of the data onto the non-linear high dimensional eigenbasis, making any visualization more difficult. Results of the denoising can be seen in Figure 5.7. The PSNR values are given in the caption.

In practice, as the kernel method is computationally demanding, we performed a weak PCA denoising on the levels 0 and 1 of the pyramid and employed the kernel PCA only at the 2^{nd} level only.

Figure 5.7 show our denoising with pyramidal PCA and kernel PCA respectively, compared to results obtained after carefully tuning the curvelets denoising algorithm implemented in Candes et al. (2006) and the non-local mean filter implemented in van der Walt et al. (2014).

The non-local mean filter does not preserve well the signal, in particular the low frequency tex-

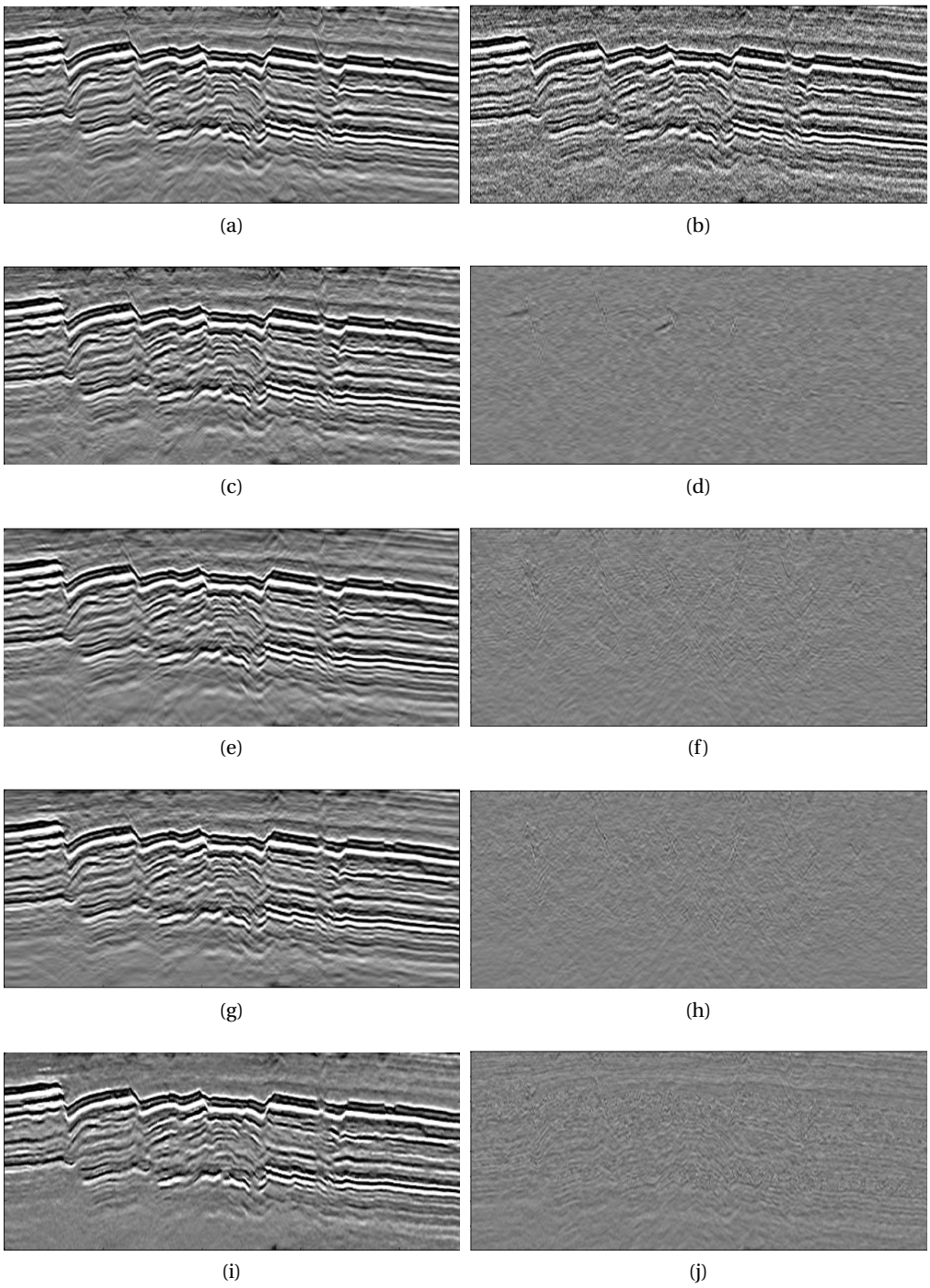


Figure 5.7 – (a) True stack. (b) Noisy stack (PSNR=23.0 dB). (c) KPCA denoised (31.4 dB). (e) Curvelet denoised (27.9 dB). (g) PCA denoised (29.3 dB). (i) Non-local mean filtering (27.1 dB). Difference with the noiseless data for (d) KPCA. (f) Curvelets. (h) PCA. (j) NLM. All figures are displayed on the same grey scale.

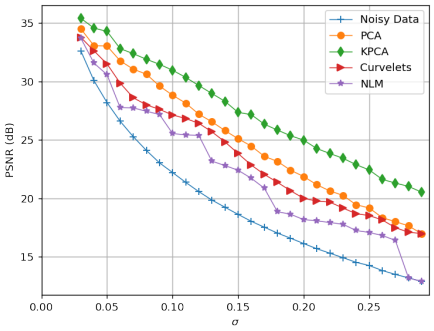


Figure 5.8 – PSNR vs Gaussian noise's standard deviation σ for our pyramidal methods with comparison to curvelets denoising and non-local mean filtering (NLM).

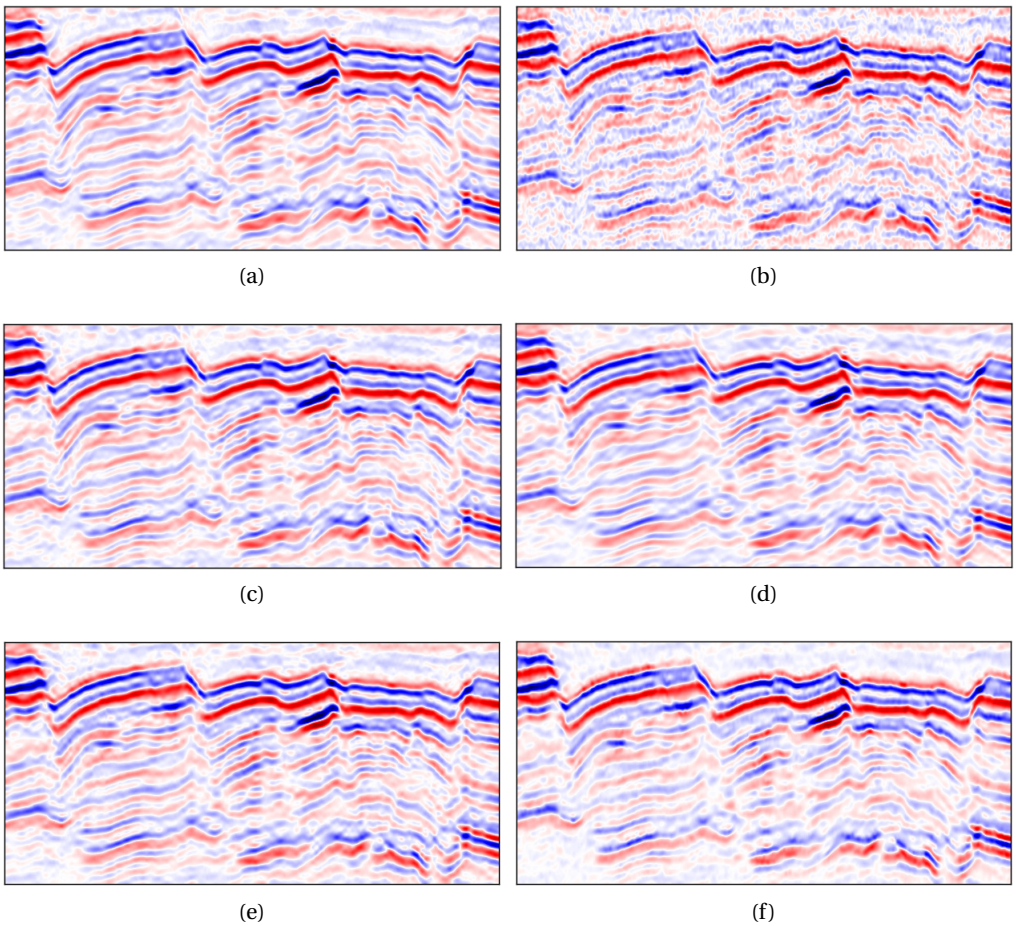


Figure 5.9 – Zoom on the results of Figure 5.7, with a different colormap. (a) True stack. (b) Noisy stack. (c) KPCA denoised. (d) Curvelet denoised. (e) PCA denoised. (f) Non-local mean filtering. Figures are displayed on the same colorbar.

tured area are not successfully recovered and the reflections are blurred across faults. Curvelets denoising is fast and setting the parameters is fairly simple, it removes a large proportion of the noise at the cost of also removing some signal. In particular, it failed recovering exactly the data along the fine and steep faults and it also introduces quite strong artefacts that are particularly visible in the shallow and deep regions of the stack. The pyramidal PCA achieves similar performances to the curvelets. The results might appear slightly more noisy but it introduces much less artefacts in the denoised stack and removes less signal. The kernel PCA, while not being perfect, achieves the best results both visually and in term of PSNR. A zoom on the results can be seen in Figure 5.9.

We additionally conducted an experiment over different noise levels. In Figure 5.8 we display the denoising results in term of PSNR for different standard deviations of the broadband Gaussian noise. For all methods we used a grid search over the parameter space employing the same criteria explained in the first section of the paper. Best results are achieved by the pyramidal kernel PCA method.

Our method has been to be successful in the difficult setting where the noise's bandwidth fully overlaps the one of the reflectivity. Pyramidal PCA achieves performances comparable to curvelets denoising while introducing less artefacts. Pyramidal kernel PCA achieved best results both visually and in term of PSNR, and successfully preserves reflectivity across all frequencies and dips. Our algorithm does require minimal manual parameters tuning while remaining adaptable to the data. In particular, small variations in hyper-parameters such as the patch size, the number of training patches or the choice of wavelet for the DWP transform have a limited impact on the quality of the denoising while the threshold coefficients and the parameter γ are efficiently selected by the method proposed by Zhu & Milanfar (2010). To the cost of a linear expansion in compute time (due to the pyramidal decomposition), we are able to reduce the quadratic cost of the Eigenproblem by requiring to use only small data patches for learning the basis. While PCA is fast and stable, it may only provide a linear separation between signal and noise. On the other hand kernel PCA has a greater adaptability but is computationally more demanding and requires one more parameter to tune. Additionally, since the denoising is performed independently on every sub-band of the decomposition level it would be easy to execute the code in parallel.

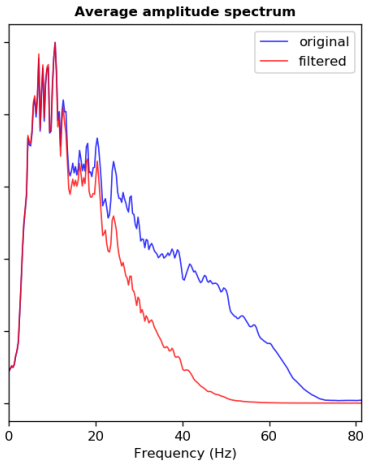


Figure 5.10 – Average amplitude spectrum of both the original and filtered data.

However, a major limitation in our method (and in the ones we refereed to) is that in practice the noise is rarely purely random and often has spatio-temporal coherency, which limits the reach of this algorithm.

5.3.2 Spectral Matching

Introduction

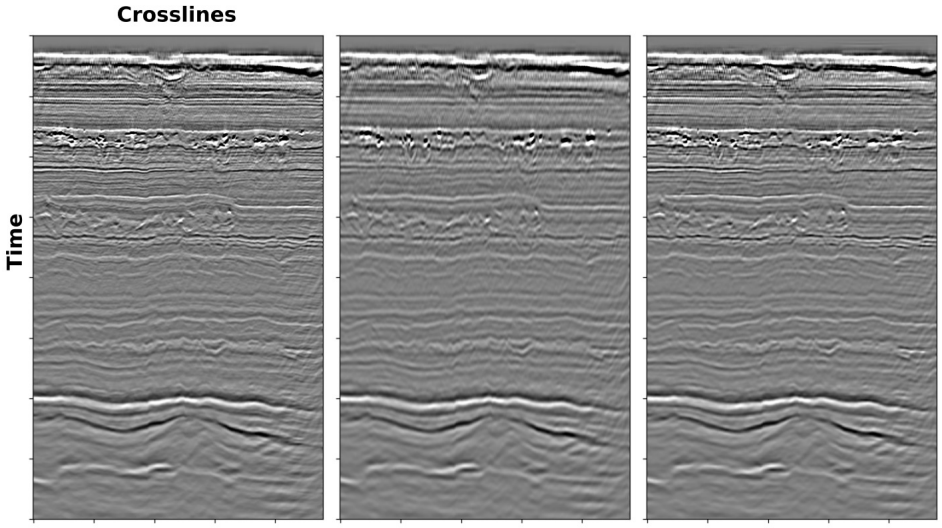
Spectral matching is a technique that aims to modify a signal such that its frequency response is comparable with a target spectrum. It is for instanced employed in certain inversion schemes to match the spectrum of the seismic data to that of well logs (Lancaster & Whitcombe, 2000). It is also an essential step in the processing of time-lapse data (Jack, 1997, Johnston, 2013). Time-lapse (or "4D") seismic refers to the use of multiple seismic surveys, acquired over the same area, in order to detect changes in the physical properties due to the production of a reservoir. In order to accurately compare the datasets, it is important to make sure they have the same frequency content. Because the time-lapse signal is subtle, care must be taken not to destroy it when matching the spectrum of the datasets. In particular, one should try to isolate the 4D signal from the background signal, and only correct for the latter. Because of non-stationary, a matching filter should be able to adapt to the different parts of the data and yield correct results for all depths and lateral positions.

Experiments and Discussion

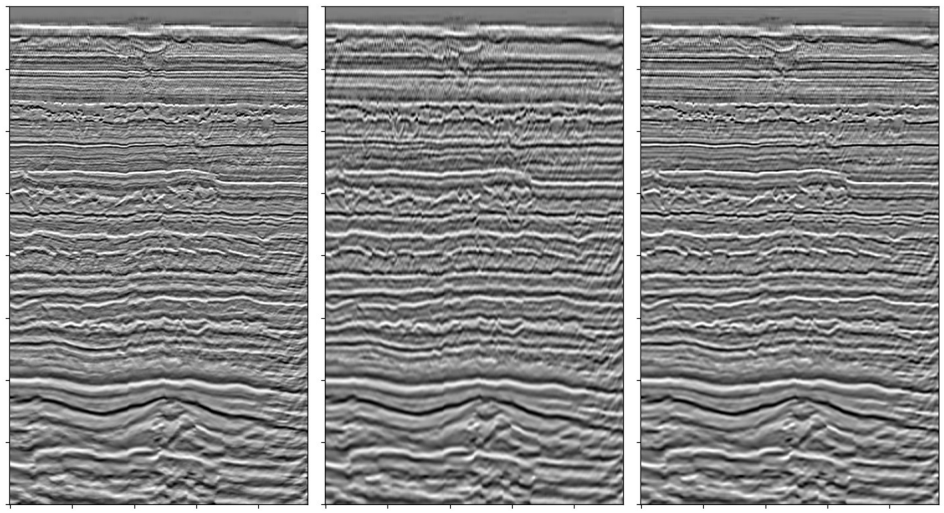
In this work, we perform an initial study using a synthetic test on the Volve open-dataset [1]. The seismic was acquired with ocean-bottom nodes, and we use a full stack of the positive offsets from the acoustic wavefield. For a first experiment, we create a second dataset by applying a low-pass Butterworth filter (Butterworth et al., 1930) to the original data. We show the average amplitude spectrum of both data in Figure 5.10. We build a 2D convolutional neural network with an architecture similar to the one we presented in our work on diffraction picking (Section 3.3.1). The CNN has a maximum stride field of 48 crosslines × 48 time samples. Both datasets have a geometry of 193 inlines × 240 crosslines × 800 time samples with respective sampling of 25m × 25m × 4ms. To create the training data we extract a 1000 pairs of corresponding patches, of size 64 crosslines × 64 time samples, from the first 20 inlines of both datasets. This corresponds to a coverage of approximatively 10%. The input of the network are the filtered patches and we train it with a standard mean-squared-error loss to reproduce the original patches, i.e. to recover the missing high frequencies. We show the results on an inline away from the training data in Figures 5.11 and 5.12. By looking at the spectrum of Figure 5.13, we can see that the network was able to well recover the missing information in the $20Hz-30Hz$ band, and to recover some of the signal in the $30Hz-45Hz$ band.

In order to study the effect of spatial variations, we perform a second synthetic test by applying different low-pass filters in different quadrants of the data. Since the network is trained on patches randomly extracted from the training data and only has a maximum stride field of 64 crosslines × 64 time samples, it is not aware of the relative position of individual data samples. This is problematic if one wish to make it learn a spatio-temporally varying behaviour. While not elegant, a way to give this information to the CNN is to concatenate the absolute position of the patches (in terms of samples #) with the seismic amplitudes. In this way, for every sample, the network will have a direct knowledge about its position in the stack. We use the same network and training configuration as for the first experiment. We show the results in Figures 5.14 and 5.15. We also show the amplitude spectrum of the four quadrants in Figure 5.16. Q1 and Q3 have the same $25Hz$ cut-off frequency, but Q3 is deeper and has therefore a lower frequency content.

[1] https://www.equinor.com/en/news/14jun2018-disclosing-volve-data.html



(a)



(b)

Figure 5.11 – Results for the inline #110. Images on the left come from the original stack, central images from the filtered stack and images on the right are from the filtered stack transformed by the network. (a) Without automatic gain control (agc). (b) With agc.

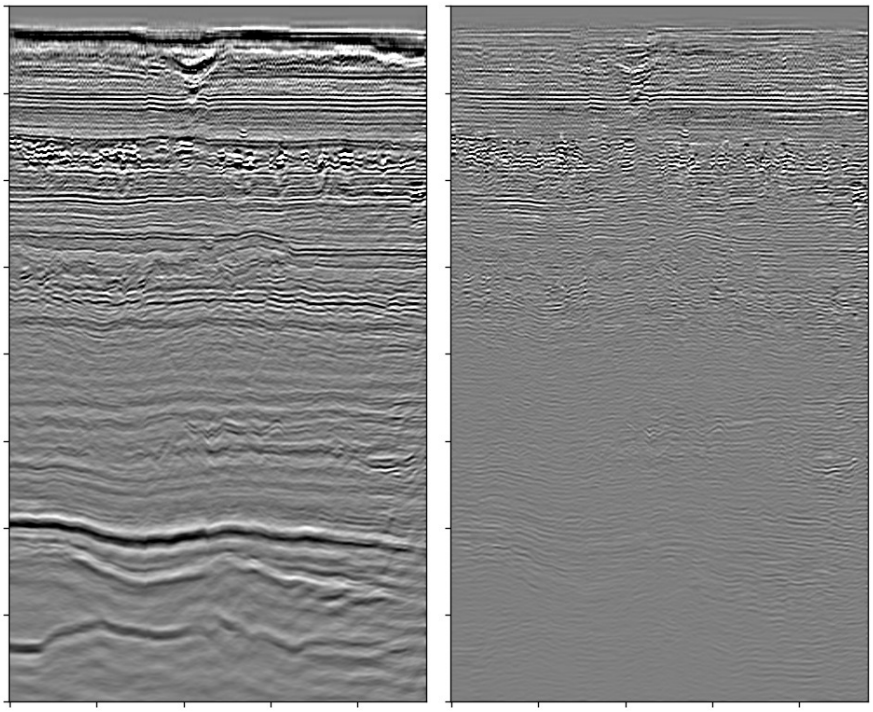


Figure 5.12 – Differences of the original data with the filtered data (a) before, (b) after, processing by the network.

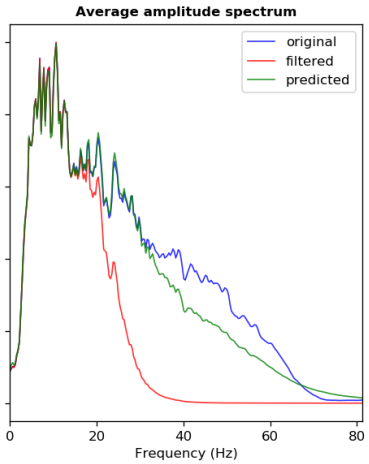


Figure 5.13 – Average amplitude spectrum of the datasets.

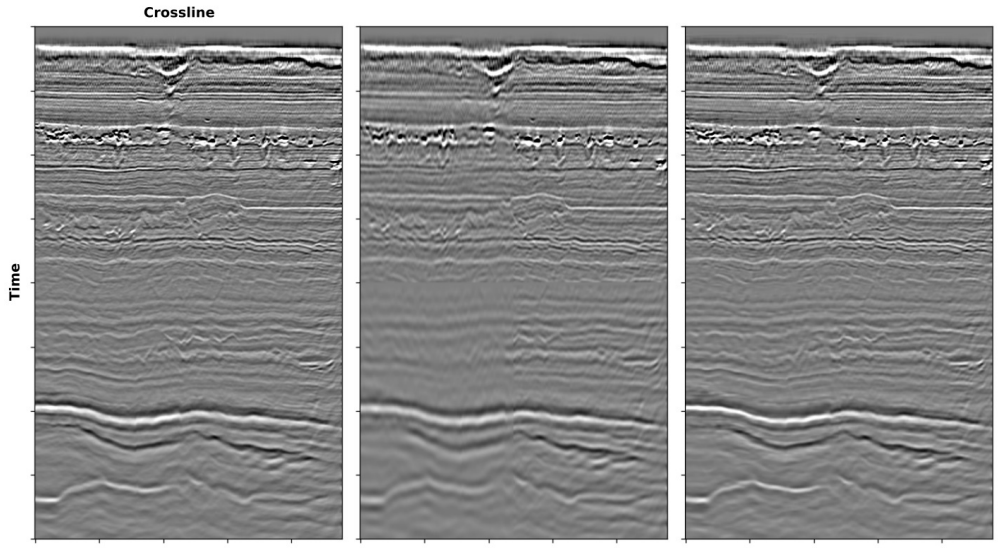


Figure 5.14 – Results for the 110^{th} inline. Images on the left come from the original stack, central images from the filtered stack and images on the right are from the filtered stack transformed by the network. For the filtering, we applied a different low-pass filter to the four different quadrants. We used an order 8 Butterworth filter with a cut-off frequency of $25Hz$ for the top-left quadrant ($Q1$), $40Hz$ for the top-right ($Q2$), $13Hz$ for the bottom left ($Q3$) and $25Hz$ for the bottom-right ($Q4$).

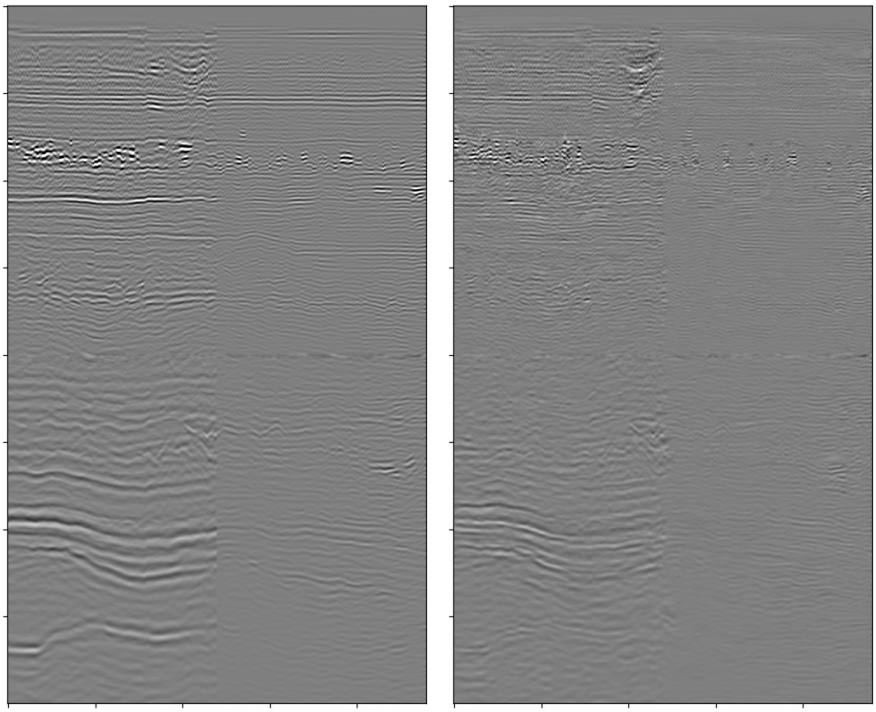


Figure 5.15 – Differences of the original data with the filtered data before (left) and after (right) processing by the network.

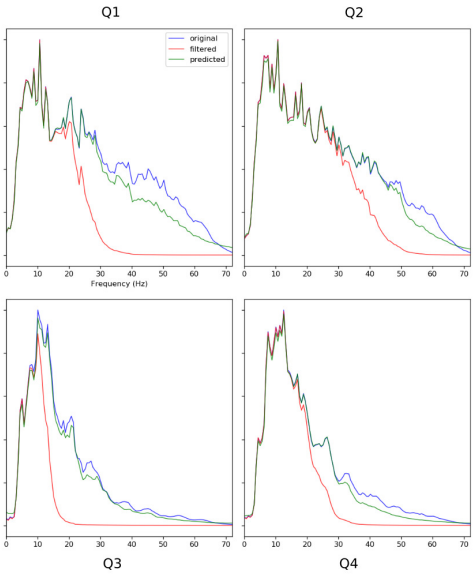


Figure 5.16 – Average normalized amplitude spectrum of the datasets computed over the four quadrants described in Figure 5.14.

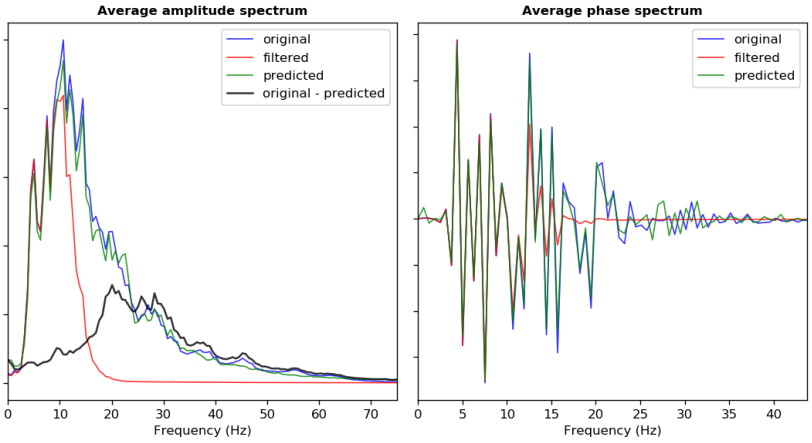
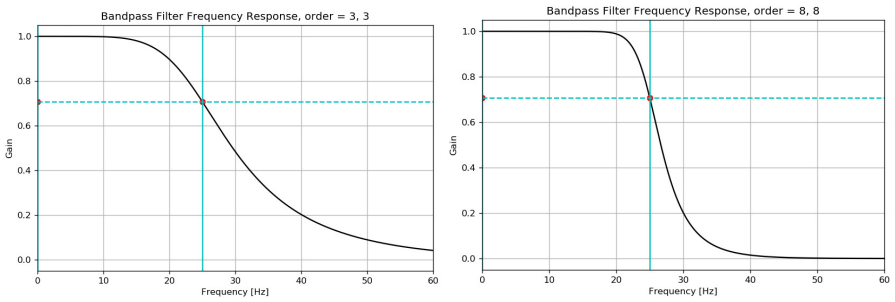


Figure 5.17 – Average normalized amplitude spectrum and phase spectrum of the datasets computed over the third quadrant ($Q3$) described in Figure 5.14.



(a) (b)

Figure 5.18 – Frequency supports for two $25Hz$ low-pass Butterworth filters with an order of (a) 3 (b) 8.

While the network is able to infer some of the missing high frequencies, we see on Figure 5.15 that the reconstruction error is still large. The test set-up is non trivial because we applied a hard cut-off frequency, and recovering the competently missing energy is most likely a very ill-posed problem. In particular, we see in Figure 5.17 that the network was not able to correctly infer the phase of the high frequency components.

For this reason, we repeat the experiment, but using order 3 Butterworth filters (see Figure 5.18). We present the results for the same test inline away from the training region in Figures 5.19 and 5.20. This time, the difference plots show that the error is much weaker and Figures 5.21 and 5.22 show that both the amplitude and phase spectrum are accurately matched.

Conclusion

Experiments show the potential of deep learning to approach the spectral matching problem. Initial results are encouraging, but we see that the method is presently enable to cope with the missing highest frequencies. It is not obvious whether or not the clues present in the low frequency filtered data are really enough to truly recover the original time resolution. An overview of deep learning methods applied to super resolution can be found in (Wang et al., 2019b). In particular, more advanced loss functions, such as adversarial training (Goodfellow et al., 2014), could be tested to see whether one can push the results further. Additionally, when applied in the context of time-lapse seismic, it is important to find a way to only match the background while preserving the signal. Usual methods are to extract training examples

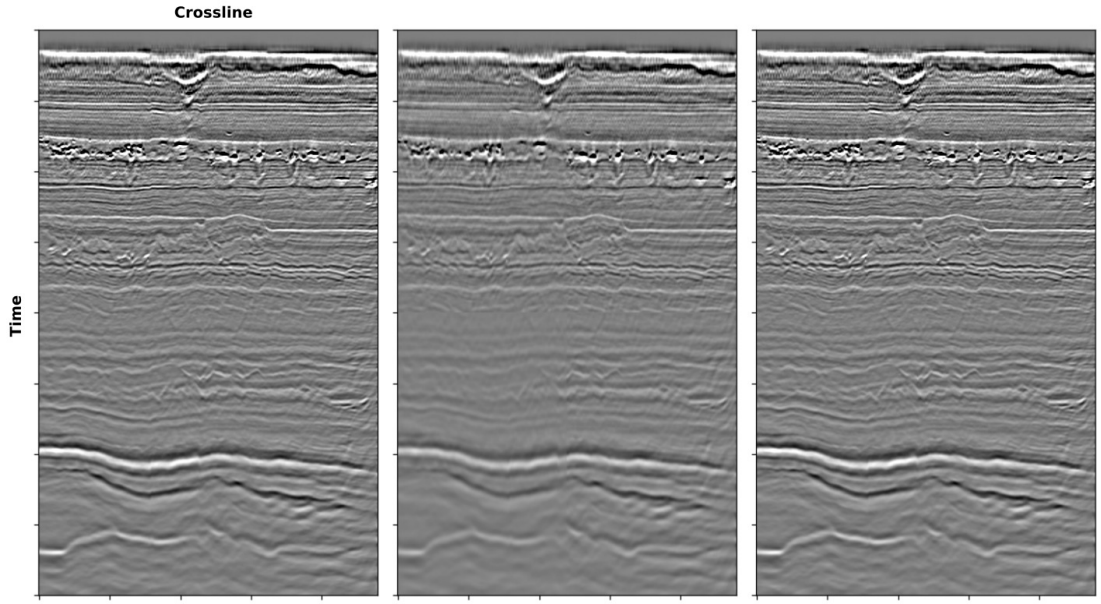


Figure 5.19 – Results for the 110^{th} inline. The set-up of the image is the same as Figure 5.14, with the difference that an order 3 Butterworth filter was used instead.

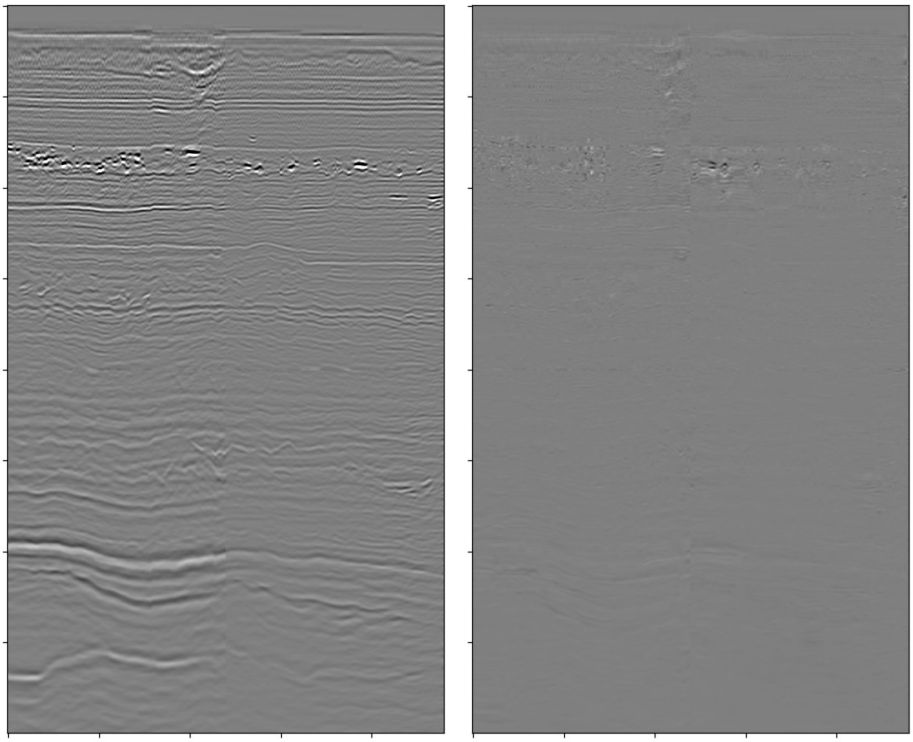


Figure 5.20 – Differences of the original data with the filtered data before (left) and after (right) processing by the network.

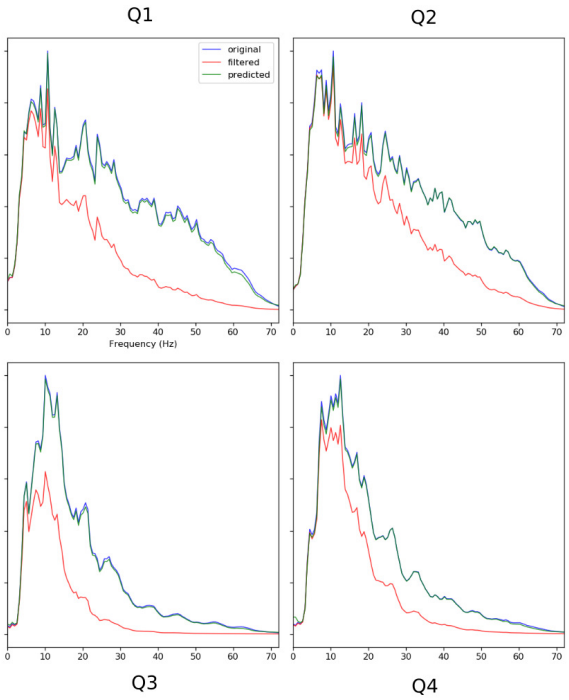


Figure 5.21 – Average normalized amplitude spectrum of the datasets computed over the four quadrants described in Figure 5.19.

away from the reservoir area, but this approach might not be sufficient.

5.3.3 Gather Conditioning

Introduction

After migration, geophysicists and geologists are tasked to understand the subsurface by performing structural and geological interpretation of the data as well as analysing amplitudes and running inversions. The success of those exercises is directly depending on the quality of the data and the goal of seismic processing is to enhance the useful signal and separate it from unwanted perturbations. We give in Section 1.1.2 a brief overview of the different processing steps performed on a dataset. Processes applied prior to migration are destined to provide good results on average over the entire dataset. Once the data have been migrated and the area of interest narrowed down, processing becomes more targeted and is usually done with a final goal in mind (like structural interpretation or inversion). While for a long time the

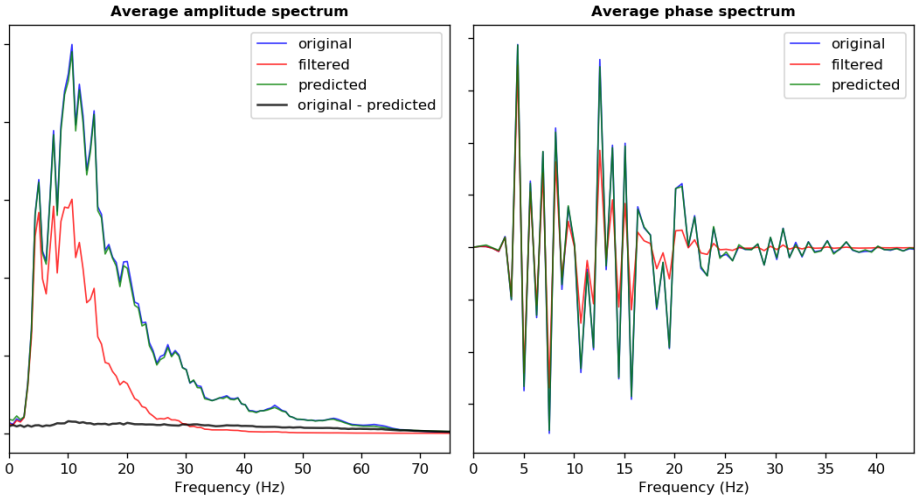


Figure 5.22 – Average normalized amplitude spectrum and phase spectrum of the datasets computed over the third quadrant ($Q3$) described in Figure 5.19.

standard delivery of a migration job was a set of partial stacks, it is now considered important to access data in the prestack domain since most of the remaining issues can only be corrected before gathers are stacked (Cook et al., 2016, Shea, 2012). In the following, we present a typical post-migration conditioning workflow applied to a field recorded dataset and we evaluate the ability of a neural network to learn to process the data in an end-to-end fashion. We also discuss the advantages and problems associated with the use of deep learning for targeted end-to-end data conditioning.

Experiment

The test dataset we use in this work was acquired in the North Sea and imaged with a Kirchhoff prestack time migration. An example gather is shown in Figure 5.23. Despite the pre-migration processing, we see that several problems remain with the data. For instance, we observe a number of parabolic events between $1.0s$ and $1.8s$, associated with multiple reflections, interfering with the primaries. Some events are not completely flat, which indicates that the processing velocities were not completely accurate, and there is also noise contamination. We devised the following conditioning workflow to alleviate those problems, with the software *PreStack-Pro* [2]:

[2] https://sharpreflections.com/

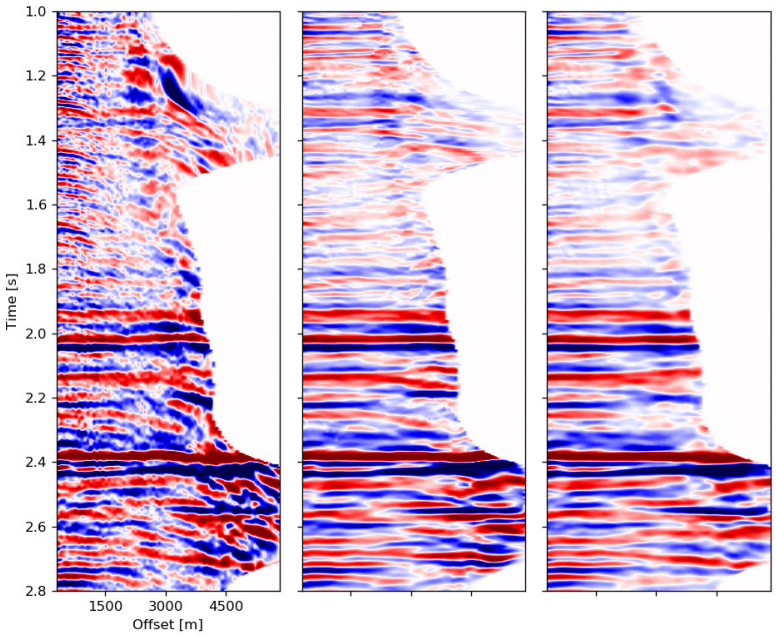


Figure 5.23 – Conditioning results for a gather located away from the training lines. Left: Noisy gather. Middle: Gather processed with conventional algorithms. Right: Gather processed by the neural network.

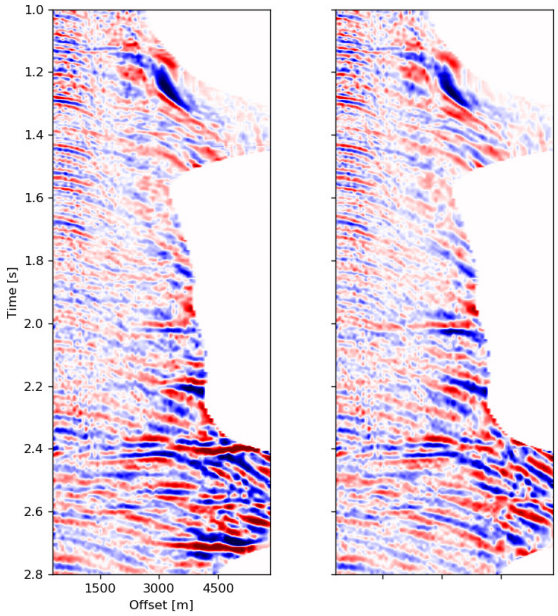


Figure 5.24 – Difference between the noisy and conditioned (left), and noisy and predicted (right) gathers shown in Figure 5.23.

- Angle mute (remove (post-critical) energy recorded at an incidence angle greater than 40°)
- Automatic Gain Control (AGC - reversible equalization of the amplitudes to be able to distinguish weak events in the $\tau - p$ domain)
- Parabolic Radon mute (remove residual multiples)
- Linear Radon mute (linear noise removal)
- Inverse AGC
- Semblance weighted spatial filtering (3D, per offset plane, random noise attenuation)
- Dip steered structurally consistent anisotropic diffusion filter (2D, per gather, random noise attenuation)
- Time variant trim statics (mathematical correction of residual move-out in the gathers)

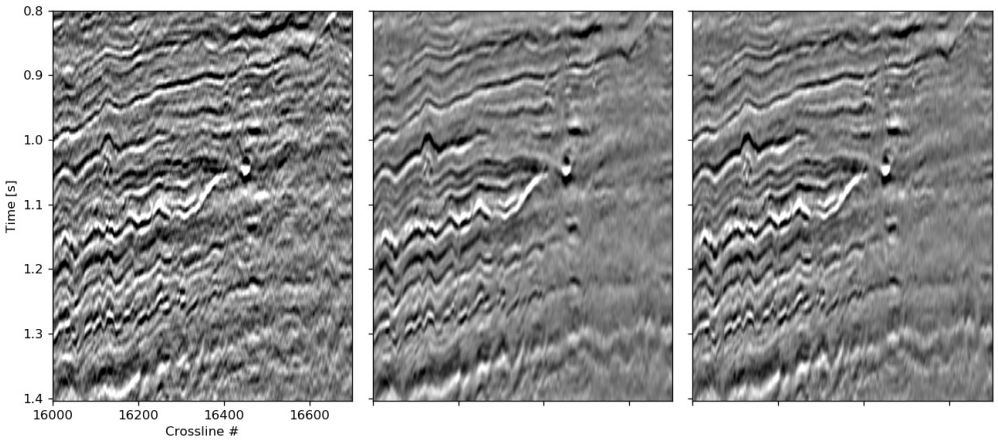
We design and train a neural network in a similar fashion as in Section 3.3.1. Our training data consists in pairs of input/output gathers before and after conditioning. Once trained, we evaluate the network on gathers away from the training area. We display results in Figures 5.23, 5.24 and 5.25.

Additionally, we also evaluate the trained network with a blind test. We use another marine dataset from the North Sea and directly apply the network without further training with the new data first. We compare the results with a traditional conditiong workflow similar to the one we presented above in Figure 5.26.

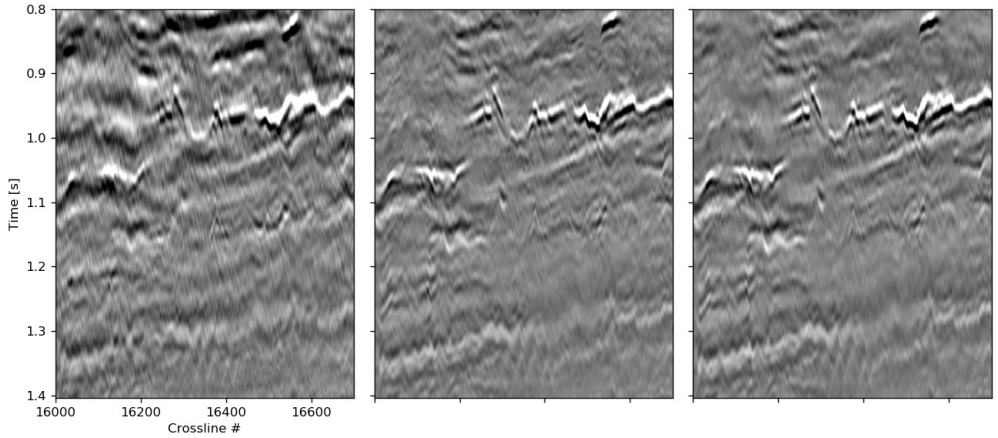
At a first look, the results provided by the network look promising. Even when applied on a dataset it was not trained on, the network seems to be able to perform a complex denoising flow. On Figure 5.26, we even see that the jittering noise present in the near offsets was corrected by the network while it is not completely removed by the traditional conditioning flow. The stacks shown in Figure 5.25 reveal that the network perform as well as the denoising worklow, and was able to preserve the relevant signal.

Discussion

There are several advantages in using deep learning to perform an end-to-end conditioning of the data. Traditional workflows involve between 5 and 10 algorithms applied sequentially.

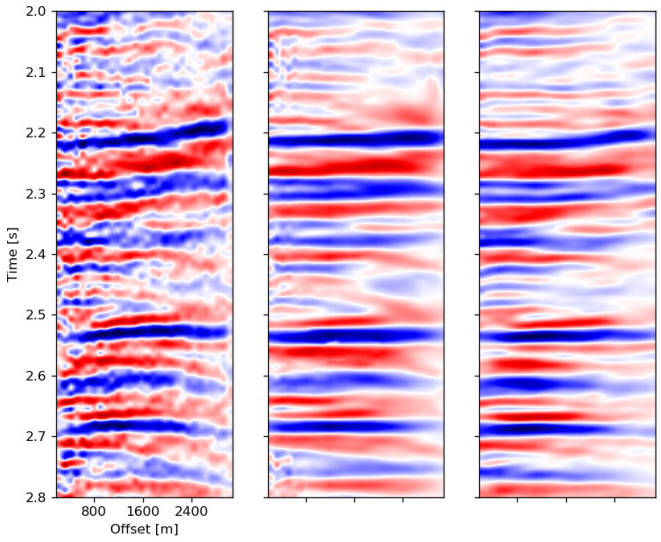


(a)

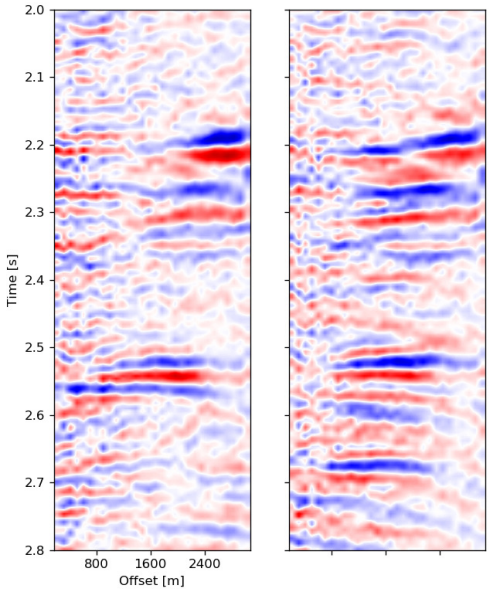


(b)

Figure 5.25 – Comparison for inline slices away from the training area. (a) Near offset stack (b) Mid offset stack. The images on the left are the input data, central images the ones processed with the traditional algorithms and on the right processed by the network.



(a)



(b)

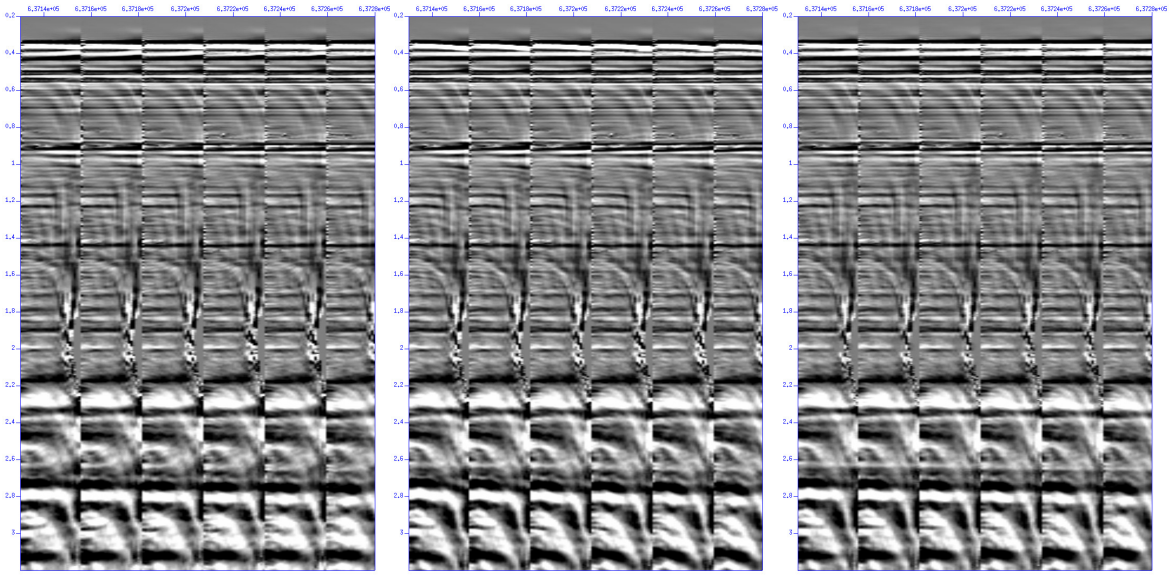
Figure 5.26 – Evaluation of the network on a blind dataset. (a) From left to right: Noisy gather, gather denoised by a conventional workflow, gather denoised by the neural network. (b) Left: difference between the noisy and traditionally conditioned gathers, right: difference between the noisy and network-conditioned gathers.

Each of them come with a set of parameters that the geophysicists must tune in order to obtain the best results. On the other hand, once trained, a neural network is parameter free. Additionally, many algorithms, like the Radon transform, are computationally heavy and induce a processing time of several hours every time they are used. For a neural network, after an initial training time of few hours, the processing time for new datasets is reduced to minutes only.

The adaptable nature of neural networks also make them ideal to process datasets with strong variations in the signal and noise over the survey area. Because of the large number of parameters in a traditional workflow and the long computing time, one usually select only one set of parameters for the entire survey. However, those parameters may not be adequate everywhere. A network, in theory, can adapt to the data and provide a tailored processing to every area.

However, there are also serious limitations to the method we present here. Data conditioning is target oriented and partially subjective. Geophysicists adapt their workflows to the particularities of the dataset. They focus mainly on a restricted region that was identified as interesting. They also adapt to the different needs and will likely have a different approach whether the final goal of the study is to perform structural picking or to run an inversion. In their standard form, trained networks can only yield one solution and one therefore lose the flexibility which is sometimes essential in interpretation. Moreover, because of the difficulty to gather and prepare training data, one can only train a network with so many examples. When provided with a new dataset with different a acquisition setting and pre-processing, recorded over an area with a different geology, it is unlikely that the network can provide an optimal answer. We see for instance in Figure 5.26 that the alignment of certain reflections (like the blue one at $2.2s$) is not as good with the network. This is likely because the training data did not have moveout effect as severe as in this data. In these conditions, it is also difficult to completely trust that the network will preserve all the important information.

Maybe, for those reasons, employing a network to perform an end-to-end conditioning is a little ambitious at first. There are many things to take into consideration, and performing a trustworthy quality control of the results as well as being able to identify and correct the



(a) (b) (c)

Figure 5.27 – Field data test for the alignment of gathers. (a) Input data (b) Alignment with a traditional algorithm (c) Alignment with a neural network. Courtesy of Breuer & Ettrich (2020)

problems brought by the network is difficult. In a similar work, Breuer & Ettrich (2020) focus instead on a single processing step, looking at the alignment problem. They employed the same method as we described above, but with pairs of input/output gathers that differ only in the alignment of the reflections in the offset domain. They created a training dataset using synthetic modelling and evaluated the results on real data. Figure 5.27 shows that they were able to get better results than with a traditional algorithm, especially in the complex area where multiples (that should not be flattened) interfere with the primaries, or where there is a polarity reversal. Because a single step is involved it is simpler to perform a rigorous quality control. While results are very encouraging, they also have similar problems. In particular, since the training data are simplified synthetics, it is difficult to guaranty that the most complex features in the real data will be preserved by the network. It is for instance not certain yet that the AVO (amplitude vs offset) behaviour is well preserved after transformation by the network.

Conclusion

Replacing parts or all algorithms in a processing workflow with neural networks is an appealing idea. Comparatively, deep learning requires much fewer parameters and since it is a data driven method, it can learn to adapt to the data and, in theory, does not suffer from intrinsic limitations like most traditional algorithms. Networks are also computationally much more efficient than a sequence of traditional algorithms. Moreover, an end-to-end approach may be beneficial over a sequence of processing steps as small errors due to processing artefacts may eventually accumulate and become problematic.

On the other hand, there are challenges not easy to overcome. Once trained, the user is left with a single solution and if the results are not good it is not trivial to improve them. Typically the method employed to improve a network's results is to gather more training examples. However, in seismic, it is not easy to do so and it is unlikely that one can have access to an ideal training dataset rich enough to cover all the cases that can be encountered in practice.

One possible source of improvement for further work will be to look at variational methods (Kingma & Welling, 2013), in combination with the segmentation method we use in this work, in order to provide answers under the form of a set of possible solutions, so that the user can select the preferred one. Using an adversarial regularization (Goodfellow et al., 2014) may also help the network to learn a more complex representation of the data and to adapt better to new datasets.

CHAPTER 6 CONCLUSION AND OUTLOOK

6.1 Conclusion

Results of the Thesis

We began by exposing the main promises offered by deep learning to solve advanced processing and interpretation tasks in seismic reflection, while putting those into perspective with the main issues for its usage in geosciences. The ability of convolutional neural networks (CNNs) to assimilate large and multi-dimensional data and to learn to perform complex tasks make them ideal to tackle the many processes involved in seismic analysis. However, fulfilling the necessary conditions to be able to exploit their impressive performance is not a trivial task, and learning models also come with inconveniences compared to traditional methods. In particular, we identified several key aspects that must be accounted for when working with geoscientific data:

- For some applications, such as the identification of geological structures in the data, preparing a training dataset may require an enormous amount of manual work.
- Because of uncertainties and a lack of resolution, designing an informative performance metric to train networks is sometimes challenging in geosciences.
- Understanding how a network came to a result and guaranteeing that its good accuracy generalizes to any new data is a non-trivial task.

In order to design algorithms that have practical uses, we developed workflows that took into account those challenges by resorting to various paradigms and demonstrated them on real case studies.

Transfer learning and knowledge distillation: In Tschannen et al. (2019) (Section 3.3.1), we used the physics of the propagation of waves in order to create a synthetic training dataset for the automatic detection of diffracting objects and showed that the knowledge acquired by the network could be successfully transferred on real data. Synthetic modelling allowed us to have a full control over the experimentation space, and to provide training information without the need of an extensive and tedious manual labelling process. Our trained CNN could extract information from prestack gathers and yield accurate results on a field dataset, highlighting several hundreds of scattering objects in few seconds only. This demonstrates a certain generalization capability of neural networks that are able to go beyond the simplified perception they got from the synthetic model and extract patterns in real data.

In Section 3.3.2, we made use of a fault picking algorithm developed using a traditional approach and showed that the results it provided could serve as an initial answer for training a 3-dimensional CNN. By iteratively training a new CNN using the results of the previous iteration, we showed how the distilled knowledge helped to progressively improve the quality of the predictions, and this without resorting to manual editing. This demonstrates a certain resilience of neural networks to inaccuracies in the training data and an ability to improve despite a partially erroneous feedback.

In Section 5.3.3, we experimented with the adaptation capabilities of CNNs to process new datasets. We trained a model by providing it with pairs of noisy and conventionally denoised gathers and experimented to see if the gained knowledge could prove useful for the end-to-end processing of a new dataset. This task is challenging because the signal and the noise present in the data are strongly affected by the geology and the pre-processing operations applied to them. Despite facing a very different dataset, our network gave a reasonable answer. There are however many remaining problems to solve before seeing the apparition of a one button algorithm that can perform full denoising on any new data.

Supervised and semi-supervised learning: In Tschannen et al. (2020) (Section 4.3.1), we showed how an interpreter could progressively help refining the predictions of a network by guiding it in the more complex areas. We derived an automatic horizon picker that could be initially trained with a reasonable amount of examples and that could successfully extrapolate

horizon surfaces in a complex 3-dimensional environment. By controlling the quality the results, the interpreter can then focus on regions where he or she disagrees with the machine and fine-train it in order to progressively converge toward an acceptable answer. In addition of limiting the amount of manual work, this approach also has the advantage to give more control to the expert and therefore allow him/her to be more confident about the results and to guide the interpretation in uncertain regions.

In Tschannen et al. (2017) (Section 4.3.2) we identified some of the pitfalls that machine learning scientists should avoid when working with geoscientific data. When using supervised learning to classify data, like the litho-fluid facies classes from well logs, it is important to keep in mind the possible sources of uncertainties. Unlike with traditional machine learning benchmarks, there is not always a clear correspondence between the data and the chosen labels and we argued about the importance there is to understand those uncertainties.

Unsupervised learning: In Sections 5.3.1 and 4.3.3 we showed how unsupervised learning could be used to denoise the data. This form of learning is appealing because it does not necessitate human intervention to prepare a training dataset. This means that it is not limited to learn only a subset of the output domain, like supervised approaches, and could therefore in theory adapt to any data. We obtained good results with random noise attenuation, however, some more advanced processing, such as the removal of coherent signals, are still out of the reach of unsupervised methods.

In Section 4.3.2, we looked at clustering, i.e unsupervised classification, to find trends in the amplitudes along an horizon based on their AVA response. Synthetic tests showed the results to be coherent, and real data tests gave interesting results, although those results can only be interesting if the experts are able to interpret them. Those form of approach are often considered to be free of human biases, but we are not sure if this is really the case. It is true that in this setting, humans are not the one that directly give the feedback necessary to learn, but experts are still responsible for designing the algorithm, choosing the performance metric and setting some key hyper-parameters which inevitably influence the results.

Remarks

In this thesis we certainly were able to appreciate the impressive capacity of deep learning to perform complex tasks and we demonstrated its usage on a variety of subjects. By designing worklows aimed at alleviating the complications brought by its usage with seismic data, we obtained algorithms that are practical and that yield good results on challenging real world applications. However, a number of factors were not properly accounted for and some things could have been done differently.

First, it was maybe a mistake to mainly focus on pattern recognition tasks such as the picking of diffractions, horizons or faults. In those applications, the key factor is the preparation, by hand, of the training datasets. We were able to circumvent this by employing appropriate stratagems, but in order to truly harness the potential of deep learning, one would have needed to access much larger and much more diversified labelled datasets. Other applications that (at least in appearance) rely much less on human labelling, such as spectral matching (Section 5.3.2) and super resolution (Wang et al., 2019a) or waveform inversion (Araya-Polo et al., 2018) would have been easier to investigate from the deep learning perspective.

Related to the first point, some applications would require a much greater number of tests in order to be validated. For instance, in the case of end-to-end gather conditioning presented in Section 5.3.3, we saw that the processing performed by the network on previously unseen data was in appearance very good. However, many subtle processes are at play and gaining confidence that no useful signal was removed or artificially distorted is not a trivial task. It remains unclear how a network behaves when extrapolating its knowledge to a new configuration and if this extrapolation can truly be trusted.

I also probably spent too much time worrying about those limitations and should have accepted to limit myself to simpler test cases. In deep learning research and applications, the most important presently seems to be the first to publish, and concerns about generalization and explainability are often considered negligible since they seem become less and less relevant in practice as more training data is being prepared and new heuristics for designing and training better models are being found.

6.2 Outlook

Multi-integration

We looked at several applications independently from one another. It would be interesting to try to unify interpretative tasks together in a single algorithm. For instance by training a CNN to recognize several features at once in the data such as faults, salt domes, horizons and channels.

Interactive Platform

Integrating machine learning algorithms into a software built for data visualization and interpretation would greatly facilitate the work of a practitioner. Having access to interactive 2D and 3D viewers with labelling tools, and being able to modify the hyper-parameters and to add new examples to a learning model and observe in real time the effects on the results of the machine would be a great help. In this setting, it would be possible to explore for an optimal synergy between the machine and the expert.

Variational and Adversarial Methods

We began to explore the possibilities offered by variational and adversarial methods in Sections 4.3.3 and 3.3.3. It would be interesting to go further and see how variational methods could be employed to provide results as a distribution of possibilities instead of a single answer. Adversarial methods also seem to provide results that substantially improve the state-of-the-art in some applications. Such applications are for instance low and high (spatial and temporal) frequencies extrapolation (Wang et al., 2019a, Halpert, 2018, Lu et al., 2018), highly efficient forward modelling (Moseley et al., 2018), or cheap, easy and accurate full waveform inversion (Araya-Polo et al., 2018, Yang & Ma, 2019).

Uncertainty Quantification

In order to develop the use of machine learning based applications in geosciences it will be important to have a better understanding of the uncertainties associated with the predictions

of algorithms. Trying to quantify uncertainties associated with the labelling process may help understanding how human biases and errors are communicated to the machine. Additionally, using appropriate frameworks such as Bayesian learning (Neal, 2004) could help practitioners to determine which predictions can be trusted or not.

Tighter Integration with Wave Propagation Physics

In Section 3.3.1 (Tschannen et al., 2019) we used synthetic data created using mathematical modelling of the physics of wave propagation to train a network and latter evaluate it on real data. It would be interesting to go further and integrate explicitly the prior-knowledge coming from physics into the learning procedure. This would be an alternative to the black-box approach and could yield more interpretable models and increase our confidence in their reliability.

Deep 3D Networks

It is often observed that the deeper the networks are, the better their results. Writing very deep 3D-CNNs to handle large seismic datasets presently remains a challenge as the available memory on accelerators is too limited and the computational complexity is multiplied by a factor greater than $n\log(n)$ (where n is the size of the third dimension). Parallelisation schemes to distribute the training over multiple nodes are being developed but some issues still need to be addressed before they can be effectively used (Keuper & Preundt, 2016). It will be interesting to try those methods and see the potential of a very deep network for seismic processing and inversion.

APPENDIX A ELEMENTS OF PROGRAMMING

Analysing and visualizing geoscientific data can quickly become a challenge in terms of computing time and memory resources. When developing research algorithms, one therefore should address those obstacles and build the programs on solid foundations if we hope to scale our work to real world applications. Writing such technologies represents a heavy task and requires profound expertise in the fields of high performance computing and software development. However, when working with limited human resources and under tight budget and time constraints, building them from scratch is impracticable. Nevertheless, we will see in the following how one can benefit from a vast set of tools developed by the scientific community, and released under friendly licences, to achieve good algorithmic performances while requiring only moderate knowledge and efforts.

In Geosciences, data are typically recorded over space and time, and come with a variety of meta-information, such as physical units and geographic coordinates, that should be readily accessible when needed. We can roughly decompose the analysis workflow in two main steps. First, the data are collected from the sensors, pre-processed, and used in a computationally heavy algorithm such as imaging and waveform inversion in seismology or model simulations in climate sciences. Secondly, the results of those simulations are used in further post-processing and visualisation operations to help the scientists derive knowledge from them. In this work, we focus only on the second step. Our data consist mainly of migrated seismic volumes stored on a regular grid. Typical volume sizes range from few hundreds of megabytes for 2D lines to hundreds of gigabytes for prestack multi-azimuth gathers. Any process that we aim to apply beyond toy examples will need to work on data that do not fit in RAM (random access memory).

Presently, one of the most widely used programming language to do research with scientific data is *Python*. Its popularity can be explained by its simplicity and by the large and active community around it. Since it is an interpreted, high-level, language, users do not have to deal with complex low-level concepts such as explicit memory management and can write working programs with a high productivity. While simplicity comes at the cost of performance issues, there exists a large ecosystem of scientific and high performance computing libraries such as *SciPy* (Jones et al., 2001), *NumPy* (van der Walt et al., 2011), *Scikit-Learn* (Pedregosa et al., 2011) or *Dask* (Rocklin, 2015) that offer simple Python application programming interfaces (api) that call behind the scenes efficient code written in compiled languages like *C* and *FORTRAN*. When using those libraries, it is possible to write scripts that will scale well to medium and reasonably large size problems.

In this chapter, I present the main tools I used to produce the results of my thesis. I followed many of the recommendations laid out by the *PANGEO* project [1]. This is by far not an exhaustive list, and I only present succinctly the packages. Interested readers should refer to online documentations and community forums for more information.

A.1 Getting Started

Before starting to write scripts it is important to set-up proper working environments to ensure that one will have access to the latest version of the programming language and have flexibility to install packages, move and copy data, open graphical interfaces and run applications on either CPUs (central processing units) or GPUs (graphical processing units). I used three distinct machines: my laptop, a single workstation holding 12 cores and 2 GPUs and a GPU cluster built with *Open Carme* (Straßel et al., 2018). Those types of machines come with a trade-off between freedom and convenience as well as memory and compute power. Working on a cluster will give us access to the best computing performances but some restrictions might affect the working conditions. In particular, one usually does not have any control on the software stack and one is entirely relying on the system administrator to get access to the tools needed. Some clusters run on old operating systems and are often not compatible with modern packages. One might also have limited access rights that can restrict permissions to

[1] https://pangeo.io/

freely upload and download data to the cluster. On the other end, when working on personal machines, one have the freedom to configure it at will but one is limited by the hardware and also carry the responsibility for the good functioning of the system as well as data back-ups.

All the computers I used were managed by a *GNU-Linux* operating system (OS). These OS are popular in the scientific community for their great flexibility, and most of the tools a programmer need are readily available and come with documentation as well as online support. It is in particular convenient to work simultaneously on several remote machines and to transfer data using the secure shell protocol (ssh). One can also automate many time consuming tasks by writing *shell scripts*. For instance, the following command typed into a console terminal will transfer data between my laptop and a remote computer over the *Fraunhofer* network:

```
$ scp ./mydata.h5 tschannen@***.itwm.fhg.de:/media/storage/DATA
```

Most of the scientific packages do not come with the default OS's python installation and should be installed separately. A convenient python distribution is offered by *Anaconda*, as it comes with the *Conda* package management system that enables to easily install and update modules while taking care of dependencies. Conda can also be used to maintain several hermetic environments, for instance to run some legacy software with Python 2 without affecting a Python 3 installation.

To write Python scripts and organize them in a project, one can resort to an integrated development environment (IDE) such as *Spyder* or *Atom*. These come with a set of useful features like syntax highlighting and integrated debugger and profiling tools. We can also do quick testing in an interactive Python shell (*IPython*) that bear conveniences like tab completion. *Jupyter notebooks* (Pérez & Granger, 2007) are designed for presenting and sharing research results by providing interactive computing and visualization capabilities as well as supporting several text formatting and web page creation languages like *markdown*, *latex*, *HTML* or *CSS*. To track the successive modifications made to the source code, one should use a version control system such as *git*. In addition of providing a safe back-up mechanism, it is also a handy tool for coordinating work among several programmers contributing to the same project, and keeping up-to-date versions of the scripts on different machines.

A.2 Data Formats

The dominant file format to store and exchange seismic data is the *SEG-Y*. It is an open standard that specifies how both the data and meta-data should be written as a hybrid text and binary file. While revisions of the standard are made from time to time, it is a little rigid due to its age, and some design choices like the trace headers are not the most convenient to work with. It also suffers from a lack of explicit constraints which results in the existence of essentially several standards instead of just one. Additionally, the format is only used by the seismology community and there is therefore little tools that can operate directly with it in the Python ecosystem.

One of the most popular file format in the geoscience community is the *NetCDF4*, a format specially created to work with scientific data represented as multi-dimensional arrays with meta-information. It builds on top of the *HDF5* format and is designed to store very large files with possibly dynamic sizes and organized under a *Unix*-like tree-based file system. In order to convert data between both formats, I used the segy parser *segyio* to load data into memory and write it to a netcdf4 archive with the *NetCDF4* library. This operation can be written inside a loop in order to read and write data by chunks, avoiding out-of-memory errors. For very large files, the operation can be sped-up by starting several processes that each read and write a portion of the data in parallel.

Additional formats that store other smaller geophysical data such as well logs or horizon picks are usually text formats and can be efficiently handled with the *Pandas* (McKinney, 2010) library.

A.3 Lazy Labelled ND-Arrays

Typically, migrated seismic data are represented as multidimensional arrays and come with a variety of meta-information, such as physical units and project geometry, that should be readily accessible when needed. The standard python library to operate on multi-dimensional arrays is *NumPy* (van der Walt et al., 2011). For the user, Numpy ndarrays are n-dimensional matrices containing elements of the same type (e.g.floating point values). Those arrays support

fast linear algebra operations and are compatible with a large number of vectorized algorithms for instance available in the *SciPy* package (Jones et al., 2001). They also provide convenient fancy indexing operations for non-trivial data slicing and merging.

However, those arrays come with some limitations, most notably they can only work in memory and they do not support labels. This limits their usage to relatively small datasets and also forces to write classes around them to keep track of meta information. To solve those issues, projects like *xarray* (Hoyer & Hamman, 2017) offer to create out-of-memory data structures that support lazy computing, chunked processing and labelling with meta-data. Xarrays can be created directly from NetCDF4 files: `ds = xr.open_dataset('./dataset.nc',group='gathers/angles')`. Figure A.1 shows the content of an xarray storing seismic gathers. The wavefield itself is contained under the field *Data variables*, and the information of the dimensions are stored under the fields *Dimensions* and *Coordinates*. We see that the wavefield is expressed over 4 dimensions and, for instance, the prestack dimension is represented by incidence angles from 4° to 60° with a resolution of 2°. Creating the xarray do not load any data into main memory but rather generates a view of the dataset. A convenient feature of those arrays is the possibility to select data by coordinates rather than indexes. The following code snippet demonstrates this by extracting the first angle gather from the dataset by using both coordinate based and index based slicing:

```
assert np.allclose(ds.amplitudes.sel(ilines=27200,xlines=16002).values ==
                                        ds.amplitudes[0,0,...].values)
```

Another feature is the lazy nature of the arrays. Data is never loaded into memory unless explicitly asked. Before performing operations, one need to specify how the array should be chunked in order for the process to work only on sub-parts of the data that can fit in memory. `ds = ds.chunk(dict(ilines=20,xlines=20,angles=-1,time=-1))` defines such data subsets as hyper-cubes of 20 inlines by 20 crosslines. To compute a near angle stack from 4° to 20° we run the following code:

```
stack = ds.amplitudes.sel(angles=slice(4,20)).sum(dim='angles')
```

At this stage no computation as been performed, the code snippet instead built a computational graph linking the input data (the on-disk NetCDF4 prestack dataset) to the output

```
<xarray.Dataset>
Dimensions:     (angles: 29, ilines: 501, time: 76, xlines: 349)
Coordinates:
  * ilines      (ilines) int32 27200 27201 27202 27203 ... 27698 27699 27700
  * xlines      (xlines) int32 16002 16004 16006 16008 ... 16694 16696 16698
  * angles      (angles) int32 4 6 8 10 12 14 16 18 ... 46 48 50 52 54 56 58 60
  * time        (time) float64 1.8e+03 1.804e+03 1.808e+03 ... 2.096e+03 2.1e+03
Data variables:
    amplitudes  (ilines, xlines, angles, time) float32 ...
Attributes:
    description:  Angle gathers
```

Figure A.1 – Print of an *xarray.Dataset* object pointing to angle gathers.

```
<xarray.DataArray 'amplitudes' (ilines: 501, xlines: 349, time: 76)>
dask.array<shape=(501, 349, 76), dtype=float32, chunksize=(20, 20, 76)>
Coordinates:
  * ilines   (ilines) int32 27200 27201 27202 27203 ... 27697 27698 27699 27700
  * xlines   (xlines) int32 16002 16004 16006 16008 ... 16692 16694 16696 16698
  * time     (time) float64 1.8e+03 1.804e+03 1.808e+03 ... 2.096e+03 2.1e+03
```

Figure A.2 – Print of an *xarray.DataArray* object pointing to a 3D stack.

data (a near stack). The variable *stack* is itself an xarray DataArray structure that inherited the correct coordinates and chunking meta-information (see Figure A.2). Several such operations can be chained together, progressively building a more complex graph. The graph can then be executed, and the output is either returned into memory or saved as additional data into the NetCDF4 dataset.

A.4 Good Performance Computing

As explained earlier, writing efficient python code mostly consists in using correctly the well tested and maintained libraries available in the scientific stack. Such libraries perform well because they were written by domain experts and run the heavy computations using behind the scenes calls to more appropriate languages. However, one limitation is that we are restricted to the available routines and some operations highly tailored for our data may not be available as is, or not easily expressible as a combination of those routines. In this case one could write our own routine in an appropriate language and offer a python api for it. But this approach is not easily feasible and takes time to carry out. In this section we instead present few tools that are developed to offer an easier approach to code acceleration and parallelization.

We introduced in the previous section the xarrays object. An additional feature that we did not mention is that they integrate well with the out-of-core parallel computing library *Dask* (Rocklin, 2015). Dask can be employed for both shared memory and distributed memory computing. The latest is used to parallelize an application to be run on a cluster, but we will showcase only the former which enables to make use of the multiple cores available on most laptops and workstations. As an example we consider the trace envelope computation. This attribute is based on the Hilbert transform and is helpful for instance to discriminate stratigraphic sequence boundaries or thin-bed tuning effects (Roden & Sepulveda, 1999). We compute it with the following code:

```
1 import numpy as np
2 from scipy.signal import hilbert
3 def trace_envelope(x,axis=-1):
4     return np.sqrt(np.square(x) + np.square(hilbert(x).imag))
```

Using Dask we can decorate the *trace _ envelope* function so that it operates lazily on our variable *stack* from the previous section. Evaluating `stack_env = dask.delayed(trace_envelope)(stack)` will create a computational graph that Dask's scheduler can automatically parallalize at run time.

Other tools exist to overcome the interpreted nature of python by enabling to compile portions of the code without resorting to an other language. *Cython* (Behnel et al., 2011) is a package used to write *C* extensions for Python with minimal efforts. The following example computes the root mean square (rms) amplitude of a trace:

```
0 %%cython
1
2 import cython
3 cimport cython
4 import numpy as np
5 cimport numpy as np
6 from libc.math cimport sqrt
7
8 DTYPE = np.float64
9 ctypedef np.float64_t DTYPE_t
10
```

```
cpdef np.ndarray[DTYPE_t, ndim=1] rms_amplitude_cython(np.ndarray[DTYPE_t
                                        , ndim=1] trace,
    unsigned int halfwindow=11):

    cdef unsigned int N = trace.size

    cdef np.ndarray[DTYPE_t, ndim=1] zs = np.zeros((halfwindow,))
    cdef np.ndarray[DTYPE_t, ndim=1] pad = np.concatenate((zs,trace,zs))
    cdef np.ndarray[DTYPE_t, ndim=1] rms_signal = np.empty((N,))

    cdef unsigned int wsize = 2*halfwindow+1
    cdef double sq = 0.
    cdef unsigned int i, j

    for i in range(N):
        sq = 0.
        for j in range(i,i+wsize):
            sq += pad[j]**2
        rms_signal[i] = sqrt(sq/wsize)

    return rms_signal
```

The library allows to define *C* types and the line %%cython is a *jupyter magic* command (see next section) that compiles the function and wraps it to be a callable from python. Figure A.3 shows that the cython version is more than 360 times faster than the equivalent naive python implementation (it is also about 1.7 times faster than the convolution based vectorized numpy version).

Numba (Lam et al., 2015) is another tool that implements a just-in-time compiler for transforming a subset of python and numpy into fast machine code. In some cases it can be very easy to use as minor modifications to the original code can bring a dramatic speed increase. The following code implements the same operation as above in python. The only difference with the naive implementation is the *decorator* added on top of the function definition.

```
@numba.jit(nopython=True)
def rms_amplitude_numba(trace, halfwindow=11):
    N = trace.size
```

```
%timeit rms_amplitude_naive(yo)
%timeit rms_amplitude_np(yo)
%timeit rms_amplitude_cython(yo)
%timeit rms_amplitude_numba(yo)
```

```
7.33 ms ± 59.5 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
34.4 µs ± 428 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
20.3 µs ± 277 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
22.9 µs ± 819 ns per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Figure A.3 – Jupyter magic command timing four implementations of the same function.

```
    zs = np.zeros((halfwindow,),dtype=trace.dtype)
    pad = np.concatenate((zs,trace,zs))
    rms_signal = np.empty_like(trace)
    wsize = 2*halfwindow+1

    for i in range(N):
        sq = 0.
        for j in range(i,i+wsize):
            sq += pad[j]**2
        rms_signal[i] = sqrt(sq/wsize)

    return rms_signal
```

Performances reported by Figure A.3 shows that the speed is comparable to the cython implementation.

For completeness we also give the numpy version of the above function:

```
def rms_amplitude_np(trace, halfwindow = 11):
    ws = 2*halfwindow + 1
    trace2 = np.power(trace,2)
    window = np.ones(ws)/float(ws)
    return np.sqrt(np.convolve(trace2, window, 'same'))
```

Another important family of tools to write efficient code are profilers and debuggers. Profilers let you dissect the memory usage and compute time of individual instructions, giving insights about potential non desired data copy and highlighting the computational bottlenecks that might be worth optimizing. Figure A.4 shows the result of a line by line profiling, with the

```
Timer unit: 1e-06 s

Total time: 0.031103 s
File: <ipython-input-37-102da2791206>
Function: rms_amplitude_naive at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def rms_amplitude_naive(trace, halfwindow=11):
     2         1          8.0      8.0      0.0      N = trace.size
     3         1         14.0     14.0      0.0      zs = np.zeros((halfwindow,),dtype=np.float)
     4         1         29.0     29.0      0.1      pad = np.concatenate((zs,trace,zs))
     5         1          4.0      4.0      0.0      rms_signal = np.empty_like(trace)
     6         1          2.0      2.0      0.0      wsize = 2*halfwindow+1
     7
     8       602        391.0      0.6      1.3      for i in range(N):
     9       601        412.0      0.7      1.3          sq = 0.
    10     14424       9811.0      0.7     31.5          for j in range(i,i+wsize):
    11     13823      19447.0      1.4     62.5              sq += pad[j]**2
    12       601        980.0      1.6      3.2          rms_signal[i] = sqrt(sq/wsize)
    13
    14         1          5.0      5.0      0.0      return rms_signal
```

(a) Naive implementation.

```
Timer unit: 1e-06 s

Total time: 0.00098 s
File: <ipython-input-33-022a459eb264>
Function: rms_amplitude_np at line 1

Line #      Hits         Time  Per Hit   % Time  Line Contents
==============================================================
     1                                           def rms_amplitude_np(trace, halfwindow = 11):
     2         1          3.0      3.0      0.3      ws = 2*halfwindow + 1
     3         1        835.0    835.0     85.2      trace2 = np.power(trace,2)
     4         1         42.0     42.0      4.3      window = np.ones(ws)/float(ws)
     5         1        100.0    100.0     10.2      return np.sqrt(np.convolve(trace2, window, 'same'))
```

(b) Vectorized implementation with numpy.

Figure A.4 – Line-by-line profiling of two implementation of the same function with IPython's *%lprun* profiler.

IPython profiler `%lprun`, of the root mean square function for the naive and numpy based implementations. Without surprise we see that most of the performance gain was obtained by replacing the loop by vectorized operations. Figure A.5 displays the result of the memory profiling of the numpy based implementation with the `%memit` magic. The IPython debugger allows for full inspection of the code, by navigating through the different layers of function calls and letting us inspect the state of each variable as they are encountered by the interpreter.

A.5 Data Pipelines for Deep Learning

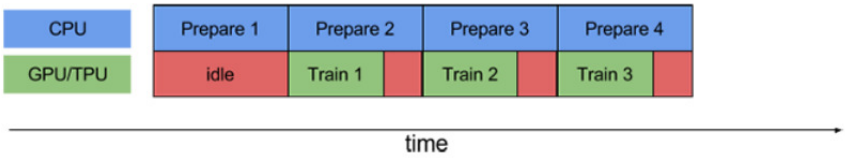
Presently, the fastest way to train a deep neural network is to resort to Graphical Processing Units. GPUs are high throughput devices composed of hundreds of cores and thousands of

```
Line #    Mem usage    Increment   Line Contents
================================================
    21  497.359 MiB  497.359 MiB   @profile(precision=3)
    22                             def rms_amplitude_np(trace, halfwindow = 11):
    23  497.359 MiB    0.000 MiB       ws = 2*halfwindow + 1
    24  573.449 MiB   76.090 MiB       trace2 = np.power(trace,2)
    25  573.656 MiB    0.207 MiB       window = np.ones(ws)/float(ws)
    26  649.930 MiB   76.273 MiB       return np.sqrt(np.convolve(trace2, window, 'same'))

peak memory: 677.09 MiB, increment: 256.11 MiB
```

Figure A.5 – Line-by-line memory profiling of a function with IPython's *%memit* magic.

(a) Without pipelining

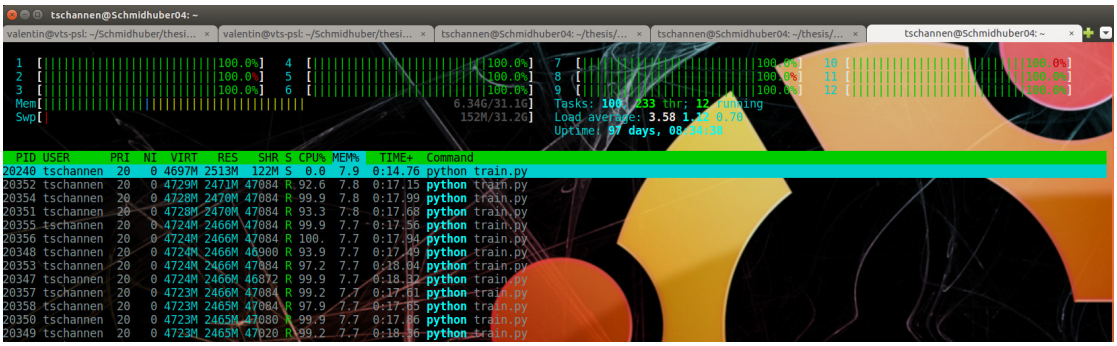


(b) With pipelining

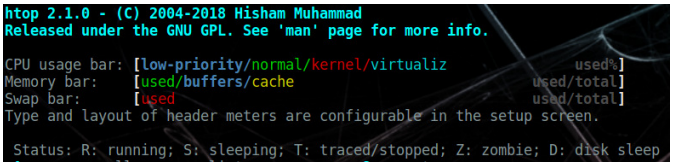
Figure A.6 – Graphic illustrating the effect of pipelining on the CPU and GPU usage (taken from Abadi et al. (2015)).

hardware threads which make them ideal for running linear algebra operations in parallel. A number of libraries have been developed to build neural networks and train them on GPUs. Two of the most popular are *Tensorflow* (Abadi et al., 2015) and *PyTorch* (Paszke et al., 2017). They come with a set of predefined layer types as well as with many low level operations that can be used to create custom layers. Users define a network by programming a computation graph expressing the data flow as well as the operation types. Those libraries also provide a set of optimizers, like the stochastic gradient descent (SGD), and compute the back-propagation using automatic differentiation. The user does not have control over the compute performance of the library, but it is essential that (s)he builds an efficient data pipeline. Figure A.6 illustrates how a naive pipeline can negatively affect training performances by starving the GPU.

As seen in the previous sections, we start with a large seismic dataset stored on disk in a



(a)



(b)

Figure A.7 – Monitoring cpus activity with the *htop* command.

NetCDF4 or HDF5 archive. Training neural networks is done in a stochastic manner by sending at every iteration a batch of samples, which each consists in smaller subset of the entire data. First, the CPU asks to the file system to fetch a batch from disk and brings it to the host memory. It then performs some optional operations and sends the batch to the GPU. With pipelining, several CPU cores work in parallel to fetch and transform the data, they queue them in a buffer and several threads dequeue the buffer to feed the GPU as soon as possible, in order to limit idle time. The following script is an example for a skeleton of a python *generator* than can be provided to Tensorflow's dataset api via the instruction `tf.data.Dataset.from_generator`. It assumes that the data is stored on disk as an HDF5 file and specifies how every samples should be extracted. The locations for the extraction are stored in the variable `self.indices` and can represent for instance the 3-dimensional position slice indexes of data sub-cubes extracted along a training inline.

```
0 import h5py
1
2 class DataGenerator:
3     """ """
4     def __init__(self, side, ...):
5         ...
6         self.side = side
```

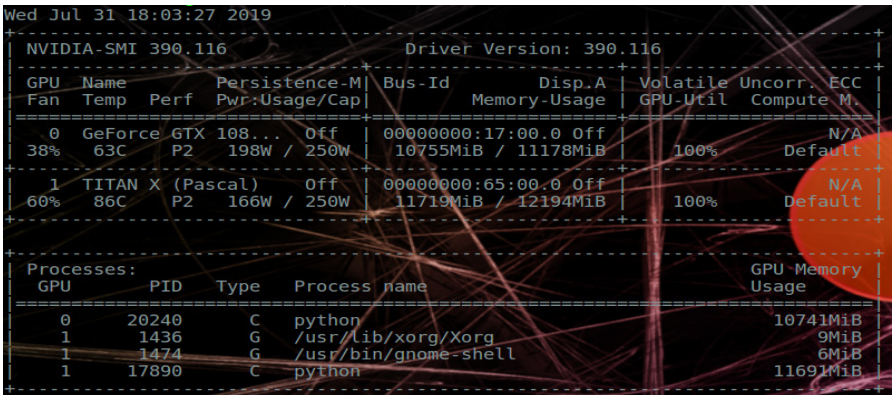


Figure A.8 – Monitoring the GPU usage the the command *watch -n 1 nvidia-smi*.

```
7           self.indices = self._get_training_slice_indices()
8
9       def __call__(self):
10          with h5py.File(HDF5_PATH,'r') as h5f:
11              reader = h5f[self.dataset_name + '/stack']
12              for idx in self.indices:
13                  cube= reader[idx]
14                  cube = preprocess_data_cube(cube)
15
16                  yield (cube, idx)
```

Another possibility, if the above approach is too slow, is to serialize the data before hand in *TFRecords* binary files and write a generator that reads those files. In Figure A.7, we see how one can use *multiprocessing* to read data with the above generator, pre-process them, and write them to a set of TFRecords in parallel. This approach as the advantage to cache the preprocessing operation that does not need to be performed any more at every training iteration, and it also saves the data format conversion time.

To make sure that the pipeline we designed is efficient, we can monitor the GPU activity (see the column *Volatile GPU-Util* in Figure A.8) and verify that there is only little idle time.

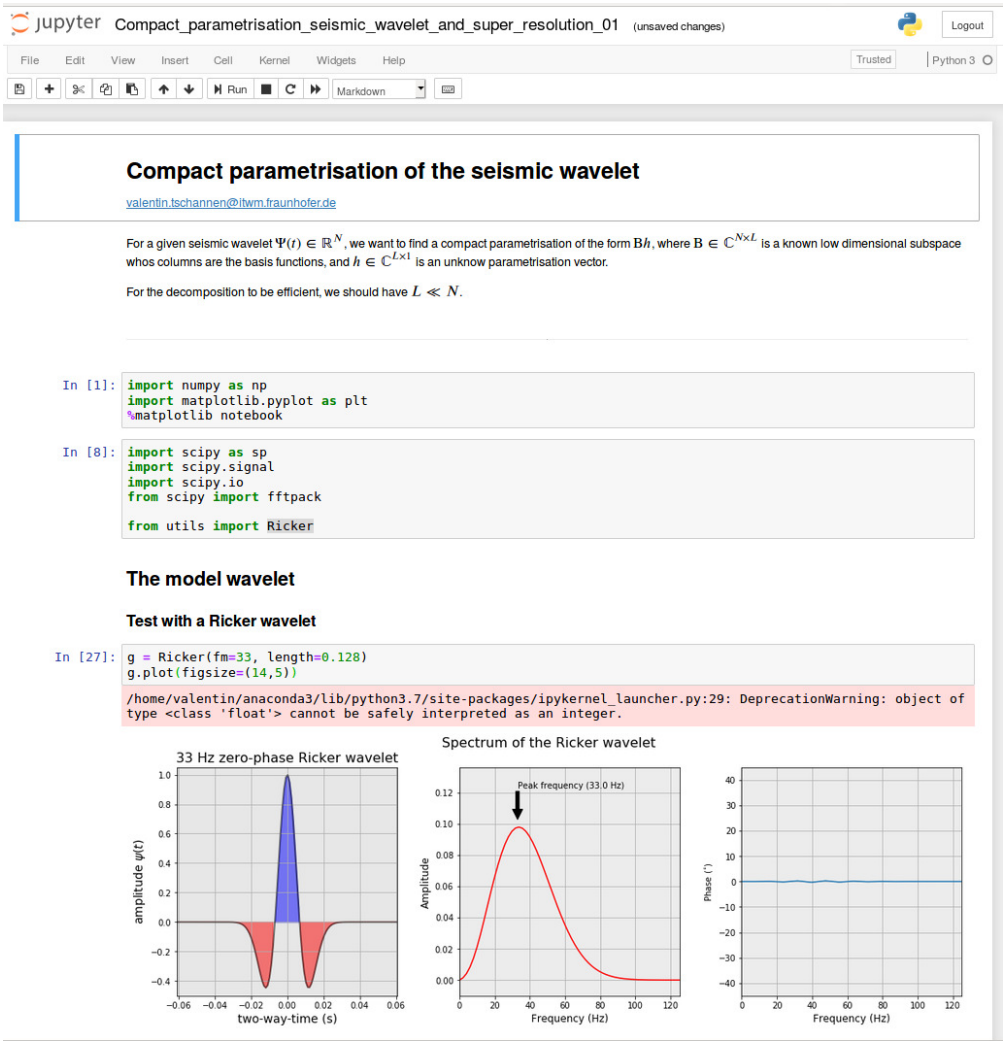


Figure A.9 – Example of a *Jupyter* notebook.

A.6 Visualizing and Reporting

Visualizing the results and organizing them to present our work to colleagues and record the experimental flow is important. We already introduced the Jupyter notebooks (or JupyterLab) (Pérez & Granger, 2007) that can run code snippet from many languages including python, markdown, latex, CSS, HTML, or javascript. This offers the possibility to create organized lab reports (see Figure A.9) that can be easily shared with colleagues, run and modified on cloud services or published under the form of blog posts for people to read in their web browser.

The default plotting library for python is *Matplotlib* Hunter (2007), I used it to create many of

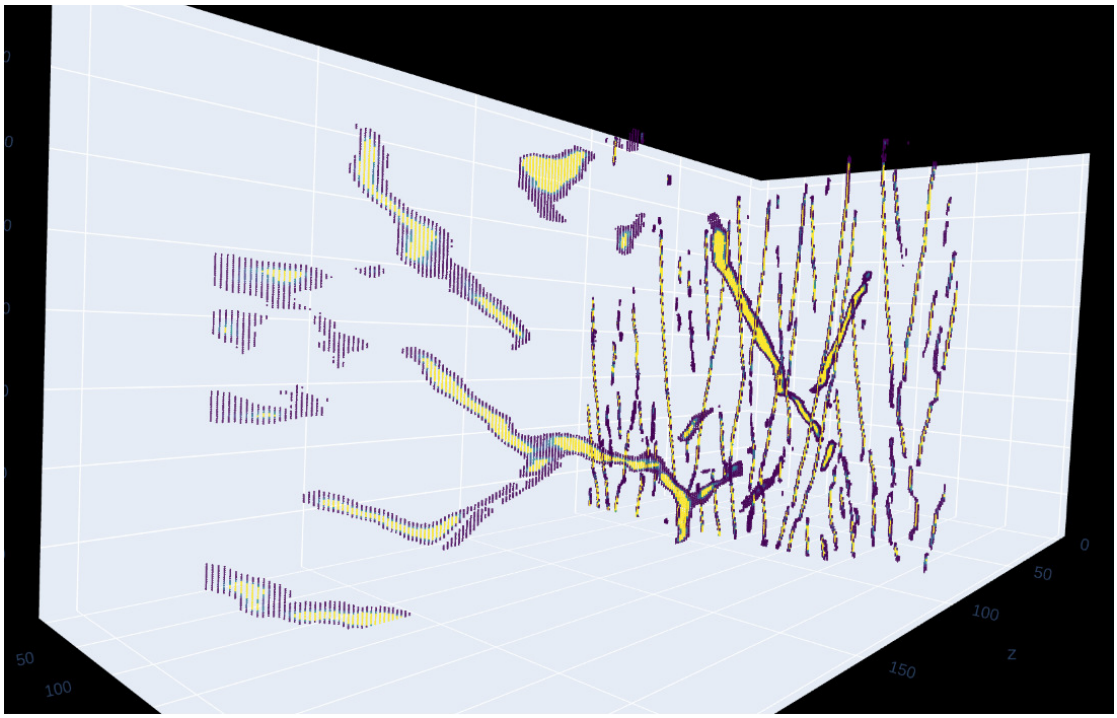


Figure A.10 – 3D viewer in a Jupyter notebook with Plotly.

the figures seen in this thesis. Recently, more modern libraries like *Plotly* (Inc., 2015) or *Bokeh* (Bokeh Development Team, 2019) have appeared. They are based on a JavaScript engine and are powerful for interactive data visualization in a web browser (see e.g. Figure A.10).

BIBLIOGRAPHY

Abadi, M. et al. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Aharon, M., Elad, M., & Bruckstein, A. (2006). K -svd: An algorithm for designing overcomplete dictionaries for sparse representation. *Signal Processing, IEEE Transactions on*, 54(11), 4311–4322.

Amershi, S., Cakmak, M., Knox, W. B., & Kulesza, T. (2014). Power to the people: The role of humans in interactive machine learning. *AI Magazine*, 35(4), 105–120.

Araya-Polo, M., Jennings, J., Adler, A., & Dahlke, T. (2018). Deep-learning tomography. *The Leading Edge*, 37(1), 58–66.

Arjovsky, M., Chintala, S., & Bottou, L. (2017). Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Arthur, D. & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms* (pp. 1027–1035).: Society for Industrial and Applied Mathematics.

Bacon, M., Simm, R., & Redshaw, T. (2003). *3-D Seismic Interpretation*. Cambridge University Press.

Bakir, G. H., Weston, J., & Schölkopf, B. (2003). Learning to find pre-images. In S. Thrun, L. K. Saul, & B. Schölkopf (Eds.), *NIPS* (pp. 449–456).: MIT Press.

Beckouche, S. & Ma, J. (2014). Simultaneous dictionary learning and denoising for seismic data. *GEOPHYSICS*, 79(3), A27–A31.

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2011). Cython: The best of both worlds. *Computing in Science & Engineering*, 13(2), 31–39.

Bengio, Y. (2012). Deep learning of representations for unsupervised and transfer learning. In *Proceedings of ICML workshop on unsupervised and transfer learning* (pp. 17–36).

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., Warde-Farley, D., & Bengio, Y. (2010). Theano: A cpu and gpu math compiler in python.

Bertsekas, D. P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3), 334–334.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bokeh Development Team (2019). *Bokeh: Python library for interactive visualization*.

Bonar, D. & Sacchi, M. (2012). Denoising seismic data using the nonlocal means algorithm. *Geophysics*, 77(1), A5–A8.

Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9), 142.

Breuer, A. & Ettrich, N. (2020). Correcting residual moveout in seismic gathers with deep learning. personnal communication.

Buades, A., Coll, B., & Morel, J.-M. (2005). A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05 (pp. 60–65). Washington, DC, USA: IEEE Computer Society.

Burgess, C. P., Higgins, I., Pal, A., Matthey, L., Watters, N., Desjardins, G., & Lerchner, A. (2018). Understanding disentangling in beta-vae. *arXiv preprint arXiv:1804.03599*.

Butterworth, S. et al. (1930). On the theory of filter amplifiers. *Wireless Engineer*, 7(6), 536–541.

Candes, E., Demanet, L., Donoho, D., & Ying, L. (2006). Fast discrete curvelet transforms. *Multiscale Modeling & Simulation*, 5(3), 861–899.

Candes, E. J. & Donoho, D. L. (2000). *Curvelets: A surprisingly effective nonadaptive representation for objects with edges*. Technical report, Stanford Univ Ca Dept of Statistics.

Chapelle, O., Scholkopf, B., & Zien, A. (2009). Semi-supervised learning. *IEEE Transactions on Neural Networks*, 20(3), 542–542.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., & Shelhamer, E. (2014). cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*.

Choi, Y., Choi, M., Kim, M., Ha, J.-W., Kim, S., & Choo, J. (2017). Stargan: Unified generative adversarial networks for multi-domain image-to-image translation.

Chollet, F. et al. (2015). Keras. https://keras.io.

Cohen, A., Daubechies, I., & Feauveau, J.-C. (1992). Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 45(5), 485–560.

Cook, D., Constance, P., Singleton, S., & Harris, P. (2016). Introduction to special section: Seismic data conditioning. *Interpretation*, 4(2), SGi–SGi.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4), 303–314.

Dalal, N. & Triggs, B. (2005). Histograms of oriented gradients for human detection.

de Matos, M. C., Osorio, P. L., & Johann, P. R. (2006). Unsupervised seismic facies analysis using wavelet transform and self-organizing maps. *Geophysics*, 72(1), P9–P21.

Deledalle, C.-A., Salmon, J., & Dalalyan, A. S. (2011). Image denoising with patch based pca: local versus global.

Delescluse, M. (2020). Processing and interpretation of a 2D seismic line from the China see. Personal communication.

Delescluse, M., Nedimović, M. R., & Louden, K. E. (2011). 2D waveform tomography applied to long-streamer mcs data from the scotian slope. *GEOPHYSICS*, 76(4), B151–B163.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*.

Donoho, D. L. (1993). Wavelet shrinkage and w.v.d.: A 10-minute tour. In *Progress in Wavelet Analysis and Applications* (pp. 109–128).

Donoho, D. L. (1995). De-noising by soft-thresholding. *IEEE transactions on information theory*, 41(3), 613–627.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul), 2121–2159.

Elad, M. & Aharon, M. (2006). Image denoising via learned dictionaries and sparse representation. In *CVPR*.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., & Bengio, S. (2010). Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb), 625–660.

Erhan, D., Bengio, Y., Courville, A., & Vincent, P. (2009). Visualizing higher-layer features of a deep network.

Estivill-Castro, V. (2002). Why so many clustering algorithms: A position paper. *SIGKDD Explor. Newsl.*, 4(1), 65–75.

Fei-Fei, L., Fergus, R., & Perona, P. (2006). One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence*, 28(4), 594–611.

Feldman, R. & Sanger, J. (2007). *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge university press.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4), 193–202.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672–2680).

Guitton, A., Wang, H., & Trainor-Guitton, W. (2017). Statistical imaging of faults in 3D seismic volumes using a machine learning approach. In *SEG Technical Program Expanded Abstracts 2017* (pp. 2045–2049). Society of Exploration Geophysicists.

Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., & Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789), 947.

Hale, D. (2013). Methods to compute fault images, extract fault surfaces, and estimate fault throws from 3D seismic images. *Geophysics*, 78(2), O33–O43.

Halpert, A. D. (2018). Deep learning-enabled seismic image enhancement. In *SEG Technical Program Expanded Abstracts 2018* (pp. 2081–2085). Society of Exploration Geophysicists.

Hartigan, J. A. & Wong, M. A. (1979). Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1), 100–108.

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

Hennenfent, G. & Herrmann, F. J. (2006). Seismic denoising with nonuniformly sampled curvelets. *Computing in Science & Engineering*, 8(3), 16–25.

Hinton, G., Vinyals, O., & Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hoffman, J., Wang, D., Yu, F., & Darrell, T. (2016). Fcns in the wild: Pixel-level adversarial and constraint-based adaptation. *arXiv preprint arXiv:1612.02649*.

Holzinger, A., Plass, M., Kickmeier-Rust, M., Holzinger, K., Crişan, G. C., Pintea, C.-M., & Palade, V. (2019). Interactive machine learning: experimental evidence for the human in the algorithmic loop. *Applied Intelligence*, 49(7), 2401–2414.

Hoyer, S. & Hamman, J. (2017). xarray: N-D labeled arrays and datasets in Python. *Journal of Open Research Software*, 5(1).

Huang, L., Dong, X., & Clee, T. E. (2017). A scalable deep learning platform for identifying geologic features from seismic attributes. *The Leading Edge*, 36(3), 249–256.

Hunter, J. D. (2007). Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3), 90–95.

Inc., P. T. (2015). Collaborative data science.

Ioffe, S. & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jack, I. (1997). *Time-Lapse Seismic in Reservoir Management*. Society of Exploration Geophysicists.

Johnston, D. H. (2013). *Practical Applications of Time-lapse Seismic Data*. Society of Exploration Geophysicists.

Jones, E., Oliphant, T., Peterson, P., et al. (2001). SciPy: Open source scientific tools for Python. [Online; accessed <today>].

Jones, I., of Geoscientists, E. A., & Engineers (2010). *An Introduction to Velocity Model Building*. Eage Publications.

Kaur, H., Pham, N., & Fomel, S. (2019). *Seismic data interpolation using CycleGAN*, (pp. 2202–2206).

Keuper, J. & Preundt, F. (2016). Distributed training of deep neural networks: Theoretical and practical limits of parallel scalability. In *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)* (pp. 19–26).

Kim, T., Cha, M., Kim, H., Lee, J. K., & Kim, J. (2017). Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 1857–1865).: JMLR. org.

Kingma, D. P. & Welling, M. (2013). Auto-encoding variational bayes.

Krasnopolsky, V. M. & Fox-Rabinovitz, M. S. (2006). Complex hybrid models combining deterministic and machine learning components for numerical climate modeling and weather prediction. *Neural Networks*, 19(2), 122–134.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12 (pp. 1097–1105). USA: Curran Associates Inc.

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A llvm-based python jit compiler. In *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC* (pp.7̃).: ACM.

Lancaster & Whitcombe (2000). *SEG Technical Program Expanded Abstracts*. Society of Exploration Geophysicists.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4), 541–551.

Lee, D.-H. (2013). Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks.

Liang, Y., Delescluse, M., Qiu, Y., Pubellier, M., Chamot-Rooke, N., Wang, J., Nie, X., Watremez, L., Chang, S.-P., Pichot, T., Savva, D., & Meresse, F. (2019). Décollements, detachments, and rafts in the extended crust of dangerous ground, south china sea: The role of inherited contacts. *Tectonics*, 38(6), 1863–1883.

Lipton, Z. C. & Steinhardt, J. (2018). Troubling trends in machine learning scholarship. *arXiv preprint arXiv:1807.03341*.

Liu, M.-Y. & Tuzel, O. (2016). Coupled generative adversarial networks. In *Advances in neural information processing systems* (pp. 469–477).

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3431–3440).

Lu, P., Morris, M., Brazell, S., Comiskey, C., & Xiao, Y. (2018). Using generative adversarial networks to improve deep-learning fault interpretation networks. *The Leading Edge*, 37(8), 578–583.

Mairal, J., Elad, M., & Sapiro, G. (2008). Sparse representation for color image restoration. *Trans. Img. Proc.*, 17(1), 53–69.

Mallat, S. (1999). *A wavelet tour of signal processing*. Elsevier.

Marfurt, K. J., Kirlin, R. L., Farmer, S. L., & Bahorich, M. S. (1998). 3-d seismic attributes using a semblance-based coherency algorithm. *Geophysics*, 63(4), 1150–1165.

Marfurt, K. J., Sudhaker, V., Gersztenkorn, A., Crawford, K. D., & Nissen, S. E. (1999). Coherency calculations in the presence of structural dip. *Geophysics*, 64(1), 104–111.

McCulloch, W. S. & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4), 115–133.

McKinney, W. (2010). Data structures for statistical computing in python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 51 – 56).

Mika, S., Schölkopf, B., Smola, A., Müller, K.-R., Scholz, M., & Rätsch, G. (1999). Kernel pca and de-noising in feature spaces. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II* (pp. 536–542). Cambridge, MA, USA: MIT Press.

Miller, E. G., Matsakis, N. E., & Viola, P. A. (2000). Learning from one example through shared densities on transforms. In *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662)*, volume 1 (pp. 464–471).: IEEE.

Mitchell, T. M. (1997). Does machine learning really work? *AI magazine*, 18(3), 11–11.

Mockus, J., Tiesis, V., & Zilinskas, A. (1978). The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129), 2.

Moseley, B., Markham, A., & Nissen-Meyer, T. (2018). Fast approximate simulation of seismic waves with deep learning. *arXiv preprint arXiv:1807.06873*.

Mosser, L., Dubrule, O., & Blunt, M. (2018). Stochastic seismic waveform inversion using generative adversarial networks as a geological prior. In *First EAGE/PESGB Workshop Machine Learning*.

Muresan, D. D. & Parks, T. W. (2003). Adaptive principal components and image denoising. In *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*, volume 1 (pp. I–101–4 vol.1).

Neal, R. M. (2004). Bayesian methods for machine learning.

Nielsen, M. A. (2015). *Neural networks and deep learning*, volume 25.

Olah, C., Mordvintsev, A., & Schubert, L. (2017). Feature visualization. *Distill*, 2(11), e7.

Olshausen, B. A. & Field, D. J. (1997). Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision Research*, 37(23), 3311 – 3325.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., & Lerer, A. (2017). Automatic differentiation in pytorch.

Pedregosa, F. et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Pérez, F. & Granger, B. E. (2007). IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3), 21–29.

Pfitzner, D., Leibbrandt, R., & Powers, D. (2009). Characterization and evaluation of similarity measures for pairs of clusterings. *Knowledge and Information Systems*, 19(3), 361.

Pham, N., Fomel, S., & Dunlap, D. (2019). Automatic channel detection using deep learning. *Interpretation*, 7(3), SE43–SE50.

Philit, S., Lacaze, S., & Pauget, F. (2019). Innovative automatic fault detection using a volume 3D scanning method. In *SEG Technical Program Expanded Abstracts 2019* (pp. 1923–1927).

Pizer, S. M., Amburn, E. P., Austin, J. D., Cromartie, R., Geselowitz, A., Greer, T., ter Haar Romeny, B., Zimmerman, J. B., & Zuiderveld, K. (1987). Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3), 355–368.

Pratt, L. Y. (1993). Discriminability-based transfer between neural networks. In *Advances in neural information processing systems* (pp. 204–211).

Robein, E., of Geoscientists, E. A., & Engineers (2010). *Seismic Imaging: A Review of the Techniques, Their Principles, Merits and Limitations*. Education tour series. EAGE Publications.

Rocklin, M. (2015). Dask: Parallel computation with blocked algorithms and task scheduling. In K. Huff & J. Bergstra (Eds.), *Proceedings of the 14th Python in Science Conference* (pp. 130 – 136).

Roden, R. & Sepulveda, H. (1999). The significance of phase to the interpreter: Practical guidelines for phase analysis. *The Leading Edge*, 18(7), 774–777.

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

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6), 386.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation*. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

Röth, G. & Tarantola, A. (1994). Neural networks and inversion of seismic data. *Journal of Geophysical Research: Solid Earth*, 99(B4), 6753–6768.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61, 85–117.

Schölkopf, B., Smola, A., & Müller, K.-R. (1998). Nonlinear component analysis as a kernel eigenvalue problem. *Neural Comput.*, 10(5), 1299–1319.

Settles, B. (2009). *Active learning literature survey*. Technical report, University of Wisconsin-Madison Department of Computer Sciences.

Shea, B. (2012). Pre-stack seismic is essential for quantitative amplitude interpretation. Seismic Profile, issue 1.

Shyu, C.-R., Brodley, C. E., Kak, A. C., Kosaka, A., Aisen, A. M., & Broderick, L. S. (1999). Assert: A physician-in-the-loop content-based retrieval system for hrct image databases. *Computer Vision and Image Understanding*, 75(1-2), 111–132.

Siahkoohi, A., Louboutin, M., & Herrmann, F. J. (2019). The importance of transfer learning in seismic modeling and imaging. *Geophysics*, 84(6), A47–A52.

Simm, R., Bacon, M., & Bacon, M. (2014). *Seismic Amplitude: An interpreter's handbook*. Cambridge University Press.

Sommer, C., Straehle, C., Koethe, U., & Hamprecht, F. A. (2011). Ilastik: Interactive learning and segmentation toolkit. In *Biomedical Imaging: From Nano to Macro, 2011 IEEE International Symposium on* (pp. 230–233).: IEEE.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1), 1929–1958.

Stimper, V., Bauer, S., Ernstorfer, R., Schölkopf, B., & Xian, R. P. (2019). Multidimensional contrast limited adaptive histogram equalization.

Straßel, D., Tschannen, V., Pfreundt, F.-J., & Keuper (Fehr), J. (2018). Og-hpc 18: Carme - an open source framework for multi-user, interactive machine learning and data analytics on distributed (gpu) systems.

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. In *International conference on machine learning* (pp. 1139–1147).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Tang, G., Ma, J.-W., & Yang, H.-Z. (2012). Seismic data denoising based on learning-type overcomplete dictionaries. *Applied Geophysics*, 9(1), 27–32.

Tschannen, V., Delescluse, M., Ettrich, N., & Keuper, J. (2020). Extracting horizon surfaces from 3d seismic data using deep learning. *Geophysics*, 85(3), N17–N26.

Tschannen, V., Delescluse, M., Rodriguez, M., & Keuper, J. (2017). Facies classification from well logs using an inception convolutional network.

Tschannen, V., Ettrich, N., Delescluse, M., & Keuper, J. (2019). Detection of point scatterers using diffraction imaging and deep learning. *Geophysical Prospecting*, in press.

Tuia, D., Pasolli, E., & Emery, W. J. (2011). Using active learning to adapt remote sensing image classifiers. *Remote Sensing of Environment*, 115(9), 2232–2242.

Tzeng, E., Hoffman, J., Saenko, K., & Darrell, T. (2017). Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7167–7176).

Van Bemmel, P. P. & Pepper, R. E. (2000). Seismic signal processing method and apparatus for generating a cube of variance values. US Patent 6,151,555.

van der Walt, S., Colbert, S. C., & Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2), 22–30.

van der Walt, S., Schönberger, J. L., Nunez-Iglesias, J., Boulogne, F., Warner, J. D., Yager, N., Gouillart, E., Yu, T., & the scikit-image contributors (2014). scikit-image: image processing in Python. *PeerJ*, 2, e453.

Verschuur, D. J., Berkhout, A., & Wapenaar, C. (1992). Adaptive surface-related multiple elimination. *Geophysics*, 57(9), 1166–1177.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., & Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(Dec), 3371–3408.

Wang, B., Zhang, N., Lu, W., & Wang, J. (2019a). Deep-learning-based seismic data interpolation: A preliminary result. *GEOPHYSICS*, 84(1), V11–V20.

Wang, Z., Chen, J., & Hoi, S. C. H. (2019b). Deep learning for image super-resolution: A survey.

Welstead, S. T. (1999). *Fractal and wavelet image compression techniques*. SPIE Optical Engineering Press Bellingham, Washington.

Wickerhauser, M. V. (1991). Lectures on wavelet packet algorithms. In *Lecture notes, INRIA* (pp. 31–99).

Wu, X. & Hale, D. (2016). 3D seismic image processing for faults. *Geophysics*, 81(2), IM1–IM11.

Wu, X., Liang, L., Shi, Y., & Fomel, S. (2019). Faultseg3D: Using synthetic data sets to train an end-to-end convolutional neural network for 3D seismic fault segmentation. *GEOPHYSICS*, 84(3), IM35–IM45.

Xiong, W., Ji, X., Ma, Y., Wang, Y., AlBinHassan, N. M., Ali, M. N., & Luo, Y. (2018). Seismic fault detection with convolutional neural network. *Geophysics*, 83(5), O97–O103.

Yang, F. & Ma, J. (2019). Deep-learning inversion: A next-generation seismic velocity model building method. *GEOPHYSICS*, 84(4), R583–R599.

Yilmaz, Ö. (2001). *Seismic data analysis: Processing, inversion, and interpretation of seismic data*. Society of exploration geophysicists.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? In *Advances in neural information processing systems* (pp. 3320–3328).

Zeiler, M. D. & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818–833).: Springer.

Zheng, Y., Zhang, Q., Yusifov, A., & Shi, Y. (2019). Applications of supervised deep learning for seismic interpretation and inversion. *The Leading Edge*, 38(7), 526–533.

Zhu, J.-Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2223–2232).

Zhu, L., Liu, E., & McClellan, J. H. (2015). Seismic data denoising through multiscale and sparsity-promoting dictionary learning. *GEOPHYSICS*, 80(6), WD45–WD57.

Zhu, X. & Milanfar, P. (2010). Automatic parameter selection for denoising algorithms using a no-reference measure of image content. *IEEE Transactions on Image Processing*, 19(12), 3116–3132.

RÉSUMÉ

Acquérir des connaissances sur la géologie de la subsurface terrestre grâce à l'imagerie sismique est un processus long et parfois fastidieux. De nombreux algorithmes sont utilisés pour transformer le signal, atténuer le bruit et aider à interpréter l'image. Ces algorithmes sont conçus par des experts et nécessitent d'être soigneusement paramétrés. De plus, de nombreuses tâches doivent être effectuées manuellement par les géoscientifiques lorsque les algorithmes ne parviennent pas à fournir de bons résultats. Ces dernières années, l'apprentissage profond, un sous-domaine de l'intelligence artificielle, a pris une grande importance. Il a été montré que les modèles d'apprentissage surpassent les algorithmes traditionnels dans de nombreuses applications à travers un grand nombre de disciplines scientifiques. Ils permettent également d'automatiser certains processus qui n'étaient jusque-là réalisables que par des humains. Cependant, il peut être difficile de remplir les conditions nécessaires pour exploiter leur potentiel. Dans cette thèse, nous identifions les principaux obstacles à l'utilisation de l'apprentissage profond, notamment ceux de l'incertitude sur l'interprétation des données et de la dépendance de l'apprentissage en exemples fournis par des experts, et proposons une série de méthodologies visant à les surmonter. Nous démontrons la validité et la faisabilité de nos méthodes sur un ensemble de problèmes d'interprétation et de traitement sismique.

MOTS CLÉS

Analyse Sismique – Apprentissage Profond

ABSTRACT

Gaining knowledge of the geology of the Earth's subsurface with seismic reflection is a long and challenging process. Many algorithms are employed to transform the signal, attenuate the noise and help interpreting the image. Those algorithms are designed by experts and require to be carefully parametrized. Additionally, many tasks have to be performed manually by geoscientists when algorithms fail to deliver good results. In recent years, deep learning, a subfield of artificial intelligence, has rose to prominence. Learning models have been shown to outperform traditional algorithms in many applications across numerous scientific disciplines. They also allow to automate certain processes that were until then only feasible by humans. However, fulfilling the necessary conditions to exploit their potential may be challenging. In this thesis, we identify the main impediments to the use of deep learning, in particular the uncertainties in the interpretation of the data and the dependency of the training procedure on examples supplied by experts, and propose a series of methodologies that aim to overcome them. We demonstrate the validity and practicability of our methods on a set of challenging seismic interpretation and processing problems.

KEYWORDS

Seismic Analysis – Deep Learning