Universität Bremen

MarDATA HELMHOLTZ SCHOOL FOR MARINE DATA SCIENCE

Zentrum für Technomathematik

AWI ALFRED-WEGENER-INSTITUT HELMHOLTZ-ZENTRUM FÜR POLAR- UND MEERESFORSCHUNG

Unsupervised Deep Machine Learning Methods to Discriminate Icequakes in Seismological Data from Neumayer Station, Antarctica

Dissertation

zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

vorgelegt von

Louisa Kinzel

Erstgutachter: Prof. Dr. Dr. h.c. Peter Maaß

Zweitgutachter: Prof. Dr. Hans-Georg Stark

Datum der mündlichen Prüfung: 05.03.2024

Abstract

Unsupervised machine learning methods are gaining attention in the seismological community as more and larger datasets of continuous waveforms are collected, since they tackle the challenging task of learning without using hand-labelled data. In this thesis, we present some neural network approaches to the task of unsupervised learning, and apply them to discriminate and cluster different seismological events, including icequakes and earthquakes. We focus mainly on contrastive learning, which has recently been showing great success in the field of computer vision and other domains, and we transfer these methods to the domain of seismology. We implemented and tested data augmentation strategies for seismological data, and applied the contrastive learning method SimCLR as well as the deep clustering method DEC. For this purpose, we created and partially labelled different datasets containing various waveforms including many icequakes detected by an STA/LTA algorithm on continuous waveform recordings from the geophysical obervatory at Neumayer station, Antarctica. We demonstrate the effectiveness of our approach using quantitative evaluation on the labelled dataset as well as qualitative evaluation of the clustering on a larger unlabelled dataset.

Zusammenfassung

Unüberwachte Methoden des maschinellen Lernens gewinnen in der Seismologie zunehmend an Aufmerksamkeit während immer mehr und größere Datensätze an kontinuierlichen Wellenformen gesammelt werden, da sie die schwierige Aufgabe des Lernens ohne manuell annotierte Daten bewältigen. In dieser Arbeit stellen wir einige Ansätze mit neuronalen Netzen für die Aufgabe des unüberwachten Lernens vor und wenden sie an, um verschiedene seismologische Ereignisse, insbesondere Eisbeben und Erdbeben, zu unterscheiden. Wir konzentrieren uns vor allem auf das kontrastive Lernen, welches in letzter Zeit große Erfolge auf dem Gebiet der Bildverarbeitung und in anderen Bereichen erzielt hat, und übertragen diese Methoden auf den Bereich der Seismologie. Wir haben Strategien zur Datenerweiterung für seismologische Daten implementiert und getestet und wenden die kontrastive Lernmethode SimCLR sowie die Deep Clustering Methode DEC an. Zu diesem Zweck haben wir verschiedene Datensätze mit unterschiedlichen Wellenformen erstellt und teilweise gelabelt, darunter viele Eisbeben, die von einem STA/LTA-Algorithmus auf kontinuierlichen Wellenformaufzeichnungen des geophysikalischen Obervatoriums der Neumayer-Station in der Antarktis erkannt wurden. Wir demonstrieren die Effektivität unseres Ansatzes anhand einer quantitativen Bewertung des annotierten Datensatzes sowie einer qualitativen Bewertung der Cluster-Analyse anhand eines größeren, nicht annotierte Datensatzes.

Acknowledgements

Firstly, I am very grateful to all of my supervisors: Peter Maaß for the mathematical research advice, and Vera Schlindwein and Tanja Fromm for the geophysical research advice and for giving me the opportunity to go on an antarctic research expedition. I thank all of you for your continued encouragement and support in my efforts to complete this PhD thesis.
My PhD project was funded through the Helmholtz School for Marine Data Science (MarDATA). Thank you to everyone involved, especially to our coordinator Enno Prigge.
I also thank all of my colleagues who helped through this time, including but not limited to Martin, Carola, Yvonne, Marlo, David, Janek and Vladimir. Your support during more difficult times (and during all times) was always greatly appreciated.
Thank you also to my friend Moritz and to my father Joachim for helping me proofread this thesis. Your efforts and feedback helped me a lot.
Finally, I am most grateful to my husband Jannik for his continued support and love during this time.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The relatively new field of cryoseismology utilizes icequakes, seismic events which originate in the ice or at the ice-bed-boundary, to monitor ice sheet dynamics over time (e.g. Podolskiy and Walter, 2016 and references therein). The geophysical observatory at Neumayer station, Antarctica (Wesche et al., 2016), has recorded a time series of over 20 years of seismological data with hundreds of seismic events a day, including numerous icequakes. These seismic events consist of a wide range of cryogenic events, local, regional and distant earthquakes, and considerable amounts of noise signals of unknown origin including instrumental, human or weather induced disturbances. Systematic event recognition and classification to create a catalogue of icequakes at Neumayer station has not been attempted before, although there have been other studies using various methods at other ice sheets (e.g. Kufner et al., 2021, Barcheck et al., 2020, Roeoesli et al., 2016, Olinger et al., 2019). Recent studies have shown that the strongly tidally modulated occurrence of icequakes allows for conclusions on the coupling of ice shelves to pinning points that hold the smaller East Antarctic ice shelves in place (Pirli et al., 2018). Specifically, thin ice shelves may only be in contact with pinning points and produce icequakes during low tides; whereas thick ice shelves will preferably move during high tide, when the ice shelf is lifted and friction on the pinning point is reduced to allow for motion and related icequake generation. The 20 year seismological record of Neumayer station provides the chance to study such kind of ice dynamics of the past two decades, but this requires fast and reliable automated algorithms. For this purpose, we aim to utilize the power of modern machine learning algorithms to analyze the dataset efficiently.

In principle, there are two main branches of machine learning applicable to this problem: supervised learning (e.g. Hudson et al., 2019, Hammer et al., 2015) and unsupervised learning (e.g. Seydoux et al., 2020, Mousavi et al., 2019). For supervised learning, the algorithms rely on data labelled by human experts to define a discriminator that also works for samples that have not been annotated by humans or seen by the algorithm. Unsupervised learning is a tool that is mainly useful for data exploration, where the event types of interest are yet to be identified. The most relevant method in unsupervised learning is the clustering task, where signals are automatically grouped into similar classes based on some measure of similarity. In seismology, there are frequently new data sets to be analysed which stem from a variety of different survey locations including fault zones, volcanoes or, as in our example, ice masses. For each of these

data sets, the occuring event types differ and the target events of interest need to be identified first. Ideally, an algorithm screens a dataset for events and based on the detected events, creates groups or clusters with similar event types. These can afterwards be further analysed by humans for e.g. the source mechanism or time evolution. We therefore explore unsupervised learning methods for icequake detection in this thesis.

In all machine learning methods, features are required to automatically and effectively discriminate different seismic signals. One way of extracting these is using expert knowledge to hand-craft features that are useful for the task at hand, for example the duration of an event or the frequency range. However, the features may be biased towards certain event types that the expert expects, and may not be useful to discriminate other events. To avoid this, another way of handling the feature extraction is to work with the raw signal directly and use a trainable neural network (NN) to automatically learn useful features.

In supervised learning, this feature extraction task is automatically learned together with the classification task. In unsupervised learning, however, training a useful NN feature extractor is more challenging. The goal is to learn general-purpose features that may be useful for a variety of different tasks; in our case, we are trying to learn features that are useful for a clustering task, i.e. we want to separate different groups of events from each other.

Icequakes can have a variety of source mechanisms, resulting in a diverse spectrum of ice-related events recorded at stations in ice-covered environments (Podolskiy and Walter, 2016, Anandakrishnan and Alley, 1997, Sinadinovski et al., 1999, Wiens et al., 2008, Lough et al., 2015). Some icequakes are related to tidal mechanisms, when vertical displacement causes bending at the grounding line (Hammer et al., 2015). Another source mechanism is ice sliding across the bed rock at pinning points (Pirli et al., 2018). Typically, icequake events that originate from the same source mechanism tend to be highly correlated and in case of tide induced events even predictable. The high similarity fuels the hope that unsupervised learning algorithms can be successfully applied.

Previously, Hammer et al. (2015) proposed using Hidden Markov models to discriminate icequakes from earthquakes in the Neumayer seismological archive. However, the method had a high computational cost and suffered from low flexibility, which makes it unsuitable to analyse the lifetime dataset of the Neumayer station or transfer the trained model to different station settings.

Other studies like Provost et al. (2017) used a random forest classifier to discriminate between different seismic signals like rockfalls and earthquakes, which similarly could be applied to discriminate icequakes from earthquakes and other signals. They rely on a hand-labelled dataset to train the classifier. Further, they use a hand-selected feature set (including for example features like duration, skewness of the signal, frequency range, etc.), thereby relying on expert knowledge to select suitable discriminatory features.

More recently, Seydoux et al. (2020) used an unsupervised deep learning approach to automatically discriminate seismic signals from noise. They use a clustering approach with neural networks, in particular wavelet scattering networks, to define the features. Other authors used Deep Embedded Clustering (DEC) (Xie et al., 2016) to cluster earthquake signals into different groups, often working with spectrograms as input to the neural networks (Mousavi et al., 2019, Jenkins et al., 2021). In this approach, an autoencoder is used along with a clustering loss

that encourages the latent features to be more clustering friendly. Here, the seismograms are encoded by a neural network into a feature space where the clustering occurs, while another neural network, the decoder, ensures useful features.
On the other hand, especially in the field of computer vision and image recognition the learning capabilities of neural networks are advancing fast. In particular, self-supervised methods show increasingly competitive performance even without using huge labeled datasets. In general, self-supervised learning methods work by defining an auxiliary task with labels that can be extracted from the data itself, without relying on manual labels. That is, they use supervised learning strategies in an unsupervised setting.
One form of self-supervised learning is contrastive learning, which relies on data augmentation to learn useful data representations. For data augmentation, samples are modified in a predefined manner, e.g. by adding noise or flipping/shifting the sample. It has recently shown great success with methods like SimCLR (Chen et al., 2020) and MoCo (He et al., 2020), which have been developed and tested in the field of image recognition, but also applied to other fields like audio processing (Al-Tahan and Mohsenzadeh, 2021). To the best of our knowledge, it has not yet been applied to seismology. Data augmentation is standard in computer vision for images, but requires some novel methods for seismological data. Based on this, contrastive learning algorithms implement an instance-level discrimination task: the learned representations of two views (i.e. two augmented versions) of the same sample are trained to be similar, while at the same time dissimilar to other samples.
In this thesis, we explored two approaches of unsupervised deep learning to the seismological setting: Firstly, unsupervised clustering with DEC and secondly, unsupervised deep feature learning and subsequent clustering with SimCLR, which is the main contribution of this thesis. In particular, we adapt recent advances in self-supervised deep learning to the domain of seismological time series analysis, with the aim of classifying icequake events by applying the contrastive feature learning method SimCLR to the seismological dataset from Neumayer Station, Antarctica. These methods result in learned feature extractors that may be used for different purposes:

- Clustering, i.e. grouping the data into clusters in order to identify event types of interest for further analysis
- as a pre-trained model to train a classification model in a semi-supervised fashion in order to create an event catalogue.

In this thesis, we focus on the first application and apply a clustering method after the feature learning with SimCLR. Specifically, our main contributions are

- Testing the applicability of contrastive learning strategies in the context of seismology,
- Providing a hand-labelled dataset for evaluation,
- Introducing novel data augmentation strategies in seismology.

The results of this work are presented in this thesis as a monograph. We start by introducing unsupervised deep learning methods along with the relevant machine learning background in chapter 2. Afterwards, we explain the data collected at Neumayer station and our pre-processing

of it in chapter 3. Finally, chapter 4 shows our experiments and chapter 5 contains some concluding remarks.

The parts of this thesis relating to the application of SimCLR to Neumayer data are also part of a published article (Kinzel et al., 2024):

> **Louisa Kinzel**, Tanja Fromm, Vera Schlindwein, and Peter Maass. Unsupervised Deep Feature Learning for Icequake Discrimination at Neumayer Station, Antarctica. *Seismological Research Letters*, 95(3): 1834–1848, 01 2024. doi: 10.1785/0220230078.

Chapter 2

Unsupervised Deep Learning

Machine learning is often first introduced in the context of supervised learning, where the data is already grouped into ground truth classes (labelled) by human annotation. In the following, we motivate the central concepts of deep learning (e.g. *loss functions*, *learning* and *training*) from a supervised standpoint, but later, these same concepts will also be applied to the unsupervised setting where the data does not have labels attached to them. The difference lies mainly in the definition of the loss function, which has to be adapted when we do not have labels available. Different supervised and unsupervised learning approaches work with different loss functions, but the general learning framework of defining a loss function and using it to optimizing the parameters of the neural network stay the same.
Therefore, in order to discuss unsupervised deep learning approaches, we start by introducing the basic concepts and terminology of machine learning and neural networks in section 2.1. This chapter is based on the equivalent chapter of my Master's thesis "Deep Learning for Picking Seismic Arrival Times at Neumayer Station, Antarctica" (Granzow, 2020). For background, we also refer to the textbooks Shalev-Shwartz and Ben-David (2014), Bishop (2006), and Murphy (2012). Afterwards, we will introduce the concept of unsupervised learning, in particular clustering analysis. Finally, we explain some approaches to the clustering task using neural networks in section 2.2.

2.1 Background: Machine Learning and Neural Networks

In supervised machine learning, we usually deal with a dataset of observations $X = \{x_1, \dots, x_N\} \subset \mathcal{X}$ and labels $Y = \{y_1, \dots, y_N\} \subset \mathcal{Y}$ (called the *training data*), with a vector space $\mathcal{X}$ of possible inputs and a vector space $\mathcal{Y}$ of possible outputs. The formal framework of statistical learning (see for example Shalev-Shwartz and Ben-David, 2014, sec. 2.1) assumes that there is some underlying unknown probability distribution $p(x, y)$ over the product space $\mathcal{X} \times \mathcal{Y}$, that describes the probability of observation-label pairs. The goal is to establish a model $f : \mathcal{X} \to \mathcal{Y}$ that represents the training data well in some way, i.e. $f(x_i) \approx y_i$. The set of possible models f is denoted by $\mathcal{F}$, called the *hypothesis space*. Also consider a loss function $\ell : \mathcal{Y} \times \mathcal{Y}$ that defines a similarity measure between a predicted value $f(x_i)$ and the true value y_i. Then the *expected*

risk (see e.g. Shalev-Shwartz and Ben-David, 2014, sec 3.2.2, eq. 3.3) is defined as

$$\mathcal{L}(f) = \mathbb{E}_p[\ell(f(x), y)] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(f(x), y)p(x, y)\, dx\, dy. \tag{2.1}$$

In theory, the expected risk is what a model should aim to minimize. But because the assumed underlying distribution is unknown, we need to consider ways to empirically approximate the risk. This leads to the definition of the *empirical risk* (see e.g. Shalev-Shwartz and Ben-David, 2014, sec 3.2.2, eq. 3.4):

$$L(f) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i), y_i). \tag{2.2}$$

If we assume that the training dataset consists of independent and identically distributed (i.i.d.) samples, then the expectation of the empirical risk is the expected risk.
Statistical learning algorithms thus aim to minimize the empirical risk, by searching for a minimum within the selected hypothesis space $\mathcal{F}$:

$$f^* \in \arg\min_{f\in\mathcal{F}} L(f) = \arg\min_{f\in\mathcal{F}} \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i), y_i). \tag{2.3}$$

In the machine learning community, the optimization for a model $f \in \mathcal{F}$ is referred to as *training* the model.
Note that the selection of the hypothesis space $\mathcal{F}$ has an important impact on the final model. The decision of restricting the search to a specific hypothesis space in the first place introduces a certain bias in the model selection. This may seem like a restriction, but is in fact advantageous in practice: Selecting a model that is too powerful may lead to *over-fitting*, meaning that it may represent the training data well but not generalize to other data. That is why usually, researchers hold back a proportion of the training data as a *validation set* in order to make sure that the trained model generalizes well. The selection of the model class also allows the researcher to incorporate prior knowledge of the task at hand, for example by using convolutions for the task of image recognition.
Another way to reduce overfitting is the use of regularization (e.g. Shalev-Shwartz and Ben-David, 2014, sec. 13), i.e. adding a term to penalize overly complex models to the optimization objective:

$$L(f) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(x_i), y_i) + \lambda\|f\|^2 \tag{2.4}$$

for some norm $\|\cdot\|$.
In the following, we will introduce a possible way of defining the hypothesis class, namely the neural network (see e.g. Hinton and Salakhutdinov, 2006). It is the fundamental tool of deep learning and it is what gives the field the name of "deep" learning, the reason for which will become clear later. We will restrict ourselves to the case of parameterized models $f = f_w$ with some high-dimensional parameter w, and thus slightly change notation and denote the empirical risk as $L(w)$, a function depending on the parameter choice w. The loss function for an individual datapoint $\ell(f(x_i), y_i)$ may be denoted as $L_i(w)$.
The preferred method to minimize loss functions of this form is *stochastic gradient descent* or

mini-batch gradient descent. A standard gradient descent algorithm would at each step update the weights by taking a step in the negative gradient direction:

$$w \leftarrow w - \eta \nabla_w L(w) = w - \eta \frac{1}{n} \sum_{i=1}^{n} \nabla_w L_i(w), \tag{2.5}$$

where η is a step size parameter and in machine learning is often called the *learning rate*.
For large datasets however, it can be very inefficient to compute the full gradient. Stochastic gradient methods avoid this problem by replacing the gradient with a stochastic approximation: Consider an unbiased stochastic gradient estimator $G(w, \xi)$ such that $\nabla_w L(w) = \mathbb{E}[G(w, \xi)]$ where ξ is a random variable with a certain distribution. Stochastic gradient methods would require the following assumptions (see e.g. Ghadimi and Lan, 2013):

- It is possible to generate i.i.d. samples of the random variable ξ.
- At every iteration t we can generate a sample of the stochastic gradient $G(w_t, \xi_t)$ such that $\nabla_w L(w_t) = \mathbb{E}[G(w_t, \xi_t)]$.
- The variance of the stochastic estimator is uniformly bounded, i.e. for any t,

$$\mathbb{E}[\|G(w_t, \xi_t) - \nabla_w J(w)\|^2] \leq \sigma^2 \tag{2.6}$$

The stochastic gradient method then updates the parameter at each step according to

$$w_{t+1} = w_t - \eta_t G(w_t, \xi_t) \tag{2.7}$$

with a step size η_t. The step size is often chosen to be constant $\eta_t \equiv \eta$. Under the above assumptions, the convergence properties of stochastic gradient methods have been studied in the literature, see e.g. Ghadimi and Lan (2013).
In particular, for the problem at hand the gradient of the loss function is in the form of an average over all datapoints

$$\nabla_w L(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w L_i(w) \tag{2.8}$$

so that the gradient may be approximated by a randomly selected single datapoint loss $\nabla_w L_i(w)$ where

$$\mathbb{E}[\nabla_w L_i(w)] = \nabla_w L(w). \tag{2.9}$$

The weights are then updated based on this estimate:

$$w \leftarrow w - \eta \nabla_w L_i(w). \tag{2.10}$$

A compromise between computing the full gradient and using a single example is mini-batch gradient descent (which is, in fact, usually used in practice). Here, a small subset (*mini-batch*) of the training examples is used instead of just one single example, giving the unbiased estimate

$$\nabla_w L(w) \approx \frac{1}{M} \sum_{i \in S} \nabla_w L_i(w) \tag{2.11}$$

with a randomly selected index set S of size M. This method is more expensive in each step, but usually leads to smoother and faster convergence. The collection of steps until all samples were used for the optimization step once is called an *epoch*.
In practice, it is common to use more sophisticated versions of stochastic gradient methods like Adam (Kingma and Ba, 2015) or RMSProp (Hinton and Tieleman, 2012). These use an adaptive learning rate and momentum, where not only the current gradient, but also the gradients of previous steps are used.

2.1.1 Neural Networks

We will now consider the case where the input and output spaces of the model are the real numbers $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}^k$. A neural network (NN, e.g. Hinton and Salakhutdinov, 2006, LeCun et al. (2015)) is a special type of model (i.e. function) $f^w : \mathbb{R}^d \to \mathbb{R}^k$, namely a composition of relatively simple functions:

$$f^w = f_1 \circ \cdots \circ f_n. \tag{2.12}$$

Each f_i is called a *layer*. The superscript w indicates that the function is parameterized by some parameters w. This specific form of model, where the function is made up of a composition of simpler functions, is what gives the field of deep learning its name: the more layers, the "deeper" the model. In the case of multiple layers, we speak of a *deep* neural network and *deep* learning.
We start by describing the simplest version of a neural network, namely the *feed-forward fully connected* neural network (if there are at least two layers, we also speak of a *multi-layer perceptron*, MLP). Figure 2.1 illustrates such a network (this form of visualization is usually referred to as the *computational graph*). The nodes in the graph are sometimes called *neurons*, inspired by the similarities to a biological neural network.

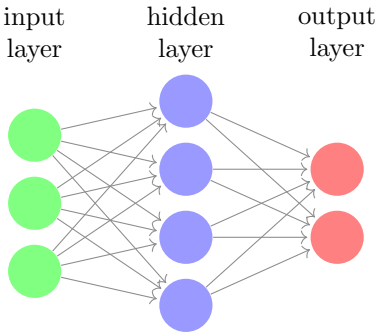


Figure 2.1: Example of a fully connected two-layer feed-forward neural network with 4 hidden nodes

In the case of fully connected (or *dense*) layers, the functions $f_i : \mathbb{R}^{n_1} \to \mathbb{R}^{n_2}$ are of the form

$$f_i(x) = g(W_i x + b_i), \tag{2.13}$$

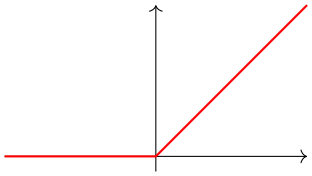
where $W_i \in \mathbb{R}^{n_2 \times n_1}$ is called the *weight matrix* and $b_i \in \mathbb{R}^{n_2}$ is the *bias vector*. These are the trainable parameters (sometimes referred to simply as *weights*) of the network and for notational convenience, they are often collected in a single parameter w. The function g is called *activation function* or simply *activation* and plays the very important role of introducing nonlinearity: Without it, the whole network would collapse into a single linear function. It is a nonlinear scalar function $g : \mathbb{R} \to \mathbb{R}$, applied to the vector $W_i x + b_i$ component-wise. Common examples include the Rectified Linear Unit (ReLU) activation (Glorot et al., 2011)

$$g(z) = \max(0, z), \tag{2.14}$$

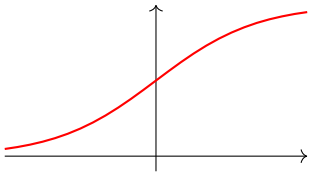
the sigmoid (or logistic) function

$$g(z) = \frac{1}{1 + e^{-z}}, \tag{2.15}$$

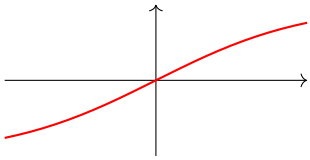
and the hyperbolic tangent $g(z) = \tanh(z)$, as depicted in Figure 2.2. For a more comprehensive overview, we refer to e.g. Goodfellow et al. (2016).



(a) ReLU $g(z) = \max(0, z)$



(b) logistic function $g(z) = \frac{1}{1+e^{-z}}$



(c) hyperbolic tangent $g(z) = \tanh(z)$

Figure 2.2: Examples of activation functions

The number of hidden layers in a neural network as well as the number of nodes in each layer is a design choice and affects the representative power of the neural network. In general, larger networks might be preferable because they can represent complicated functions more accurately. In fact, it is known that under mild assumptions on the activation function, any continuous function on a compact subset of $\mathbb{R}^n$ can be approximated arbitrarily well by a feed-forward neural network with a single hidden layer if there are just enough neurons in that layer. This result is known as the *universal approximation theorem* (Cybenko, 1989, Hornik, 1991). Unfortunately, the result is not very practical. In machine learning, we do not actually know the true function we are trying to approximate, but instead we only have samples of it and want to learn the parameters to approximate it. While the theorem ensures the existence of good parameters, it does not say anything about the *learnability* of them. Furthermore, the number of hidden neurons N would have to be very large. That is why in practice, one often uses deeper networks (i.e. more hidden layers) and more complicated layers. It can be desirable to keep the number of layers and weights small in order to keep the network simple and prevent over-fitting to the training examples.

Loss Functions

Depending on the task the neural network is used for, the output layer is processed accordingly. The two most common uses include classification and regression, but in the case of deep clustering or contrastive learning, many different kinds of loss functions may be used.
For example in the case of **classification**, the last layer would have as many nodes as there are classes and the network would be trained so that the one with the largest value 'wins', i.e. the example is classified as that class. One common way to achieve this is the *softmax cross entropy classifier*. In this case, the output, which is some real vector $z = f^w(x)$, is turned into a probability distribution (that is, a vector with values between 0 and 1 that sum up to 1) by applying the softmax function $h : \mathbb{R}^k \to \mathbb{R}^k$ where

$$h_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}. \tag{2.16}$$

Each entry can then be interpreted as the probability that the example belongs to that class. In the case of classification, the label is the desired output distribution: The probability of the correct class should be 1, while all other probabilities should be 0. Therefore, we can now apply the *cross-entropy* loss. The cross-entropy between a true distribution $p(x)$ and an estimated distribution $q(x)$ is defined as

$$H(p, q) := -\sum_x p(x) \log q(x). \tag{2.17}$$

In this case, we are comparing the true distribution

$$p(x) = [0, \ldots, 0, 1, 0, \ldots, 0] \tag{2.18}$$

with the output of our neural network

$$q(x) = \frac{1}{\sum_k e^{z_k}} [e^{z_1}, \ldots, e^{z_c}]. \tag{2.19}$$

The loss for the i-th training example would thus be

$$L_i(w) = -\log \left(\frac{e^{z_{y_i}}}{\sum_k e^{z_k}} \right), \tag{2.20}$$

where y_i is the true label for that example.

The total loss for this classifier is then the average over the individual data point losses, and it is dependent on all weights of the network, which we shall denote w:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} L_i(w). \tag{2.21}$$

Another common task in machine learning is the **regression** task, i.e. the prediction of a target value. In this case the output layer would be a single number $z = f^w(x)$ and the network would be optimized with a similarity measure in $\mathbb{R}$. The most common loss function is the

quadratic loss:

$$L_i(w) = (f(x_i) - y_i)^2. \tag{2.22}$$

Another possible choice is the absolute value loss:

$$L_i(w) = |f(x_i) - y_i|. \tag{2.23}$$

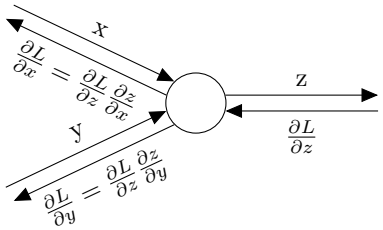
A combination of the two is the huber loss (Huber, 1964), which is a quadratic loss for small values and a linear loss for larger values:

$$L_i(w) = \begin{cases} \frac{1}{2}(f(x_i) - y_i)^2 & \text{for } |f(x_i) - y_i| \leq \delta \\ \delta|f(x_i) - y_i| - \frac{1}{2}\delta^2 & \text{for } |f(x_i) - y_i| > \delta \end{cases}. \tag{2.24}$$

We will discuss more loss functions for the case of unsupervised deep learning later in section 2.2.

Training a Neural Network: Backpropagation

When training the neural network one needs an efficient way to compute the gradient of the loss function L of the output with respect to the weights. This is done with the *backpropagation* algorithm (see e.g. Goodfellow et al., 2016), which is essentially applying the chain rule backwards through the network. The idea can be illustrated as follows:



This diagram shows a single neuron in the network. For each layer, we can compute the local gradient of the output of the node with respect to the inputs in advance; in the example above, we have the output z and inputs x and y with local gradients $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial y}$. Now when computing the gradients of the output L, we start from the very last node in the network, where the gradient is trivially $\frac{\partial L}{\partial L} = 1$. Going backwards through the network, we encounter situations like the one illustrated above: From the left, we have the incoming gradient $\frac{\partial L}{\partial z}$ of L with respect to z. By the chain rule, the gradients $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial y}$ can be computed by multiplying with the local gradient:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}, \qquad \frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}. \tag{2.25}$$

We can apply this process to each layer in the network, until all gradients are computed.

2.1.2 Convolutional Neural Networks

Convolutional neural networks (CNNs, LeCun et al., 2015) are extremely widely used in deep learning applications. They are most popular for their use in image recognition, but have also been applied to one dimensional time series data such as the seismic records studied in this thesis. Their strength lies in the fact that they are able detect local structures in the data by using rather small, local filters.
One typical way to design convolutional neural networks is to start with a series of convolutional layers, combined with pooling layers and regularization layers, followed by a series of fully connected layers. In this standard architecture, the first (convolutional) part serves as a *feature extractor*, while the second (fully connected) part serves as a regressor or classificator, depending on the task at hand.

Convolutional Layers

Convolutional neural networks use an operation known as *discrete convolution*. It is derived from the usual (continuous) convolution that is defined for function $x, w : \mathbb{R} \to \mathbb{R}$:

$$(x * w)(t) := \int_{-\infty}^{\infty} x(\tau) w(t - \tau) d\tau. \tag{2.26}$$

Note that the convolution is commutative, i.e. $x * w = w * x$. In the context of convolutional neural networks however, it is common to view the first argument as the *layer input* while the second one is the *kernel* or *filter* that is to be learned by the network. The result is called *feature map*.
When working with data on a computer, the data will usually be discrete instead of continuous, e.g. in form of vectors or matrices. In that case, x and w are defined on the integers $\mathbb{Z}$ and the integral is reduced to a sum, defining the discrete convolution:

$$(x * w)(t) := \sum_{a=-\infty}^{\infty} x(a) w(t - a). \tag{2.27}$$

In practice, the input x and kernel w have only finite support, so that the infinite sum is actually a finite one. In neural networks it is also common to choose the support of the kernel to be much smaller than the input. This allows the network to learn local structures in the data.
We are also often dealing with two dimensional input data X defined on $\mathbb{Z} \times \mathbb{Z}$, so that we also define a two dimensional kernel W and the discrete convolution becomes

$$(X * W)(t, s) = \sum_{a} \sum_{b} X(a, b) W(t - a, s - b). \tag{2.28}$$

This may also be generalized to even higher dimensions.
Additionally, one might also deal with data that has multiple channels, e.g. RGB channels for coloured images or different spatial components of seismograms. In those cases, the kernel is adapted accordingly to include the channel dimension without convolving over that dimension. It is common for convolutional layers to have multiple kernels in order to learn different features

in the input.
Another change that researchers make to the basic form of convolution in practice is to change the *stride*. Intuitively, convolution can be imagined as "sliding" the kernel over the input, and computing the sum of the element-wise product at each step. The stride determines how the filter slides over the input: A stride of 1 means that the filter is moved one pixel at time as it was defined above, whereas a stride of 2 means two pixels at a time and so on.
A convolutional layer thus has three important hyper-parameters: The number of kernels, the size of the kernels and the stride. Another design choice when applying a convolutional layer is how to treat the boundaries: it is common to pad the boundary of the input to this layer with zeros to control the size of the layer output. Another possibility is to only include those convolutions where the filter and the input overlap completely (*valid* padding).

Pooling Layers

Pooling layers are commonly used in conjunction with convolutional layers to reduce the spatial dimension of the feature maps. The depth dimension remains untouched. To reduce the input by a factor of e.g. 2, one would apply a filter of size two with a stride of 2 and only retain the maximum (or average) activation of each window. When using the maximum operation, this layer is known as a *max pooling* layer, and when using the averaging operation, the layer would be called *average pooling*.

Regularization Layers

It is common to apply some special kind of layers which act as a regularization within a neural network in order to prevent the model from overfitting and to improve the learning behaviour. The most common examples include *dropout* (Srivastava et al., 2014) and *batch normalization* (Ioffe and Szegedy, 2015).
When using dropout, at each training step a neuron is either kept with probability p or dropped with probability $1 - p$, meaning it is set to zero for that step, effectively reducing the network to a smaller one. After the step, the neuron is set back to its original value. This is usually only done during training time, while at test time, the full network is used.
Batch normalization has empirically proven to be an effective technique to improve the learning speed and stability of a neural network in practice. Batch normalization is a step that normalizes the input to each layer. Ideally, this normalization would be done by the mean and variance of the entire training set; However when using stochastic mini-batch optimization techniques, it is more practical to use the mini-batch mean and variance. This kind of layer thus learns the normalization parameters with each training step.

2.2 Unsupervised Deep Learning

Unsupervised machine learning generally refers to machine learning settings where the data does not come with labels as would be the case in supervised learning. This is for example the case when we have a dataset of unknown images, or a long time series of not-yet analysed seismogram recordings. The goal of unsupervised machine learning algorithms is to find hidden patterns

in the data, usually for an exploratory analysis of the dataset. One of the most important types of unsupervised learning is *cluster analysis*, where we aim to split the given dataset into groups of similar instances such that the instances in each group are similar to each other, and at the same time not (as) similar to instances in other groups. This task is particularly challenging in the case of high dimensional data like images and seismograms, since the semantic relationship between different instances is very difficult to determine based solely on the values of individual pixels or seismogram samples. This is why in high dimensional spaces, machine learning algorithms generally work on some sort of feature vector derived from the instance (this is true both in unsupervised and in supervised learning). There are two approaches two derive this feature vector: Manually, for example by extracting spectral information, color information, shape information, or the like; or automatically using a neural network. The machine learning algorithm (in this case, clustering algorithm) is then applied on these features.
In the following sections, we will first describe the clustering task in general, and specifically the k-means clustering algorithm, which is the algorithm we used in our experiments. Afterwards, we will introduce some deep learning approaches to learn features in an unsupervised manner, in order to generate the features needed for a clustering analysis.

2.2.1 Clustering Analysis

To describe the clustering task more precisely mathematically, consider a dataset $Z = \{z_1, \dots, z_N\}$. Then the *clustering task* is to partition the dataset into k clusters $S_1, \dots, S_k$. That is, S_i are nonempty subsets of Z such that each element in Z is in exactly one of the subsets S_i, or equivalently, $S_1, \dots, S_k \subset Z$ such that:

$$S_i \neq \emptyset \qquad (S_i \text{ are nonempty}) \tag{2.29}$$

$$S_i \cap S_j = \emptyset \text{ for all } i \neq j \qquad (S_i \text{ are pairwise disjoint}) \tag{2.30}$$

$$\bigcup_{i=1}^{k} S_i = Z \qquad (S_i \text{ cover the whole dataset}). \tag{2.31}$$

To achieve this partitioning, there exists a multitude of clustering algorithms. For a comprehensive survey, see e.g. Xu and Tian (2015), Ezugwu et al. (2022). They can be broadly classified into different categories:

- **centroid-based clustering**. These rely on a pre-defined number of cluster centroids as a kind of prototype for each cluster, which are generally iteratively refined by minimizing a certain loss function. They include for example the popular k-means algorithm (Lloyd, 1982) and Gaussian Mixture Models (GMM) with Expectation-Maximisation (EM) (Dempster et al., 1977), as well as variants of these.

- **density-based clustering**. These works with the assumption that clusters are collections of objects that occupy dense regions in the data space, separated by regions of smaller density. They includes for example DBSCAN (Ester et al., 1996) and OPTICS (Ankerst et al., 1999).

- **hierarchical clustering**. These build a hierarchical structure of nested clusters, ranging

from a single cluster containing the whole dataset to as many clusters as there are objects in the dataset. They start either from the single cluster which is successively split into separate clusters (*divisive clustering*, DIANA (Sarle et al., 1991)), or from many single-object clusters which are successively merged into larger clusters (*agglomerative clustering*, e.g. Ward-method (Ward, 1963), single linkage, complete linkage).

Our experiments are all based on the k-means algorithm, which we will describe in some more detail now.

The k-means Algorithm

k-means clustering (Lloyd, 1982) aims to partition the dataset $Z = \{z_1, \dots, z_N\}$ into k clusters $S_1, \dots, S_k$, each with a cluster centroid μ_j, such that the sum of squared distances to the cluster centroids is minimal. Here, the distance is measured by the euclidian distance. k-means thus aims to find

$$\arg\min_{S_1,\dots,S_k} \sum_{j=1}^{k} \sum_{z \in S_j} \|z - \mu_j\|_2^2. \tag{2.32}$$

The centroid is defined as the mean of the datapoints in its cluster:

$$\mu_j = \frac{1}{|S_j|} \sum_{z \in S_j} z, \tag{2.33}$$

where $|S_j|$ is the number of datapoints in S_j.
Algorithmically, this is achieved by first randomly initializing the cluster centroids (e.g. by choosing samples of the dataset $\{z_i\}$ as initial centroids) and then iterating two steps:

1. assign each sample to the closest centroid,
2. recompute centroids as the mean of samples in its cluster.

The iteration continues until assignments do not change anymore. Note that the algorithm does not guarantee an optimal solution, and depends heavily on the initial choice of centroids. There are many different variants of the algorithm, for example k-median clustering (where the median is used instead of the mean) or fuzzy C-means clustering, where each datapoint is allowed a "fuzzy" probability of belonging to each cluster (soft clustering) instead of assigning each datapoint to a single centroid. An improvement over k-means clustering is the k-means++ algorithm, which is an algorithm for choosing the initial centroids in a more sophisticated way.

High Dimensional Clustering

In a clustering analysis, the goal is to split a dataset into groups of similar instances based on some measure of similarity. When applied to a dataset containing images or, in our case, seismograms, the groups should be split based on some semantic meaning. Here, it is not immediately clear on what basis the algorithm should split the data into. Consider, for example, a dataset of images containing pictures of cats and dogs. Some of the cats and dogs are standing up, some are lying down, of some we only see the face, some are outside and some are on a bed, they have different fur colours and pattern, and so on. In this instance, if a human is asked to

split the dataset into two groups, the most natural way would probably be to separate it into 1. pictures of cats and 2. pictures of dogs. But who said that we wouldn't want to split the dataset in different ways: inside vs. outside pictures, light vs. dark fur, standing up animals vs. lying down animals, or whatever other categories we might think up?
This is the challenge in developing automated clustering algorithms for high-dimensional data like images. In *natural clustering*, we make the assumption that different instances can be naturally associated with some categories (like, for examples, cats and dogs). We imagine some representation space where the different categories occupy different regions of the space, and closeness in representation space corresponds to semantic similarity of the associated instance. That is, the categories form clusters where the similarity, or closeness, within each category is higher than the similarity between different clusters. Further, when two such clusters or categories are close together in this representation space, the categories are similar in meaning (think, for example, the cluster of cat images and the cluster of dog images, lying close together when embedded within a larger dataset containing further images of horses, birds, and penguins).
Since classical clustering algorithms do not work well in "pixel space" (i.e. directly on the image or seismogram), it is helpful to extract features from the objects in question before applying the clustering algorithm. One possibility is manual feature extraction, where we extract information like spectral content, color (for images) or envelope shape (for seismograms) and apply the clustering algorithm on these features. Another possibility is using neural networks for feature extraction. The latter is what we want to explore further in the remaining sections, and we describe two general neural network approaches for learning features from unlabelled data.

Learning Features with Neural Networks: supervised vs. unsupervised learning

Traditionally, deep learning was considered to work best in a supervised setting. That is, the deep learning algorithm is provided with a dataset that consists of data samples $X = \{x_1, \ldots, x_N\}$ along with the corresponding labels $Y = \{y_1, \ldots, y_N\}$. Generally, the performance of supervised learning models increases with the size of the dataset. That means that in this supervised deep learning setting, we have a need for large labelled datasets. However, labelling a large dataset is not an easy task for multiple reasons. Firstly, the process of labelling a large dataset is quite time-consuming and therefore expensive. Secondly, the person labelling the data is introducing a personal bias into the data, since it is not always straightforward to assign a label to a specific data sample: see, for example, our examples of weird seismic events (or non-events) in the data section, Figure 3.3.
We frequently use the example of computer vision and/or image analysis in this thesis, since it is a field where many new deep learning methods are developed and tested. Here, again, in computer vision we have the "luxury" of having access to multiple large labelled benchmark datasets, like for example the popular Imagenet Jia Deng et al. (2009), on which many new algorithms are tested first. In seismology, the community has started to establish such benchmark datasets especially for the task of phase picking (see e.g. Münchmeyer et al., 2022). For the task set in this thesis (i.e. the detection of icequakes in continuous seismological records) however, no such benchmark datasets are established.

Therefore, if we want to tackle the problem of icequake detection with deep learning, there are two main options: Either the creation of a large labelled dataset by hand and subsequent application of supervised learning strategies, or the use of unsupervised deep learning methods. In recent years, unsupervised deep learning methods have been developed that show quite remarkable success and they have experienced rising popularity ever since. There are different approaches to unsupervised deep learning, and we have explored two specific ones in this work: Deep Embedded Clustering (Xie et al., 2016) as a representative of deep clustering methods and SimCLR as a representative of contrastive learning methods. Deep clustering focuses on utilizing deep neural networks for the task of clustering high dimensional data, and employ different strategies to learn features that are suitable for the clustering task. Contrastive learning on the other hand focuses on learning good general-purpose features in an unsupervised manner, i.e. without using human-annotated data. It is not specific to the clustering task, but clustering is one possible way to use the learned features.
One challenge all of these unsupervised deep learning strategies have to deal with is the problem of collapsing representations, which happens when we try to use neural networks in a too naive way. For example in clustering, we may try to minimize the distance of data samples within each cluster; but when bringing a learnable neural network as a feature extractor into the equation, a trivial solution might be to map all samples to the same value. Then the condition is satisfied trivially, but the learned neural network is not useful at all. Therefore, care always has to be taken to avoid such a collapse of representations, and different algorithms use different strategies which we will discuss below.

2.2.2 Deep Embedded Clustering

The first unsupervised deep learning method we explore is called Unsupervised Deep Embedding for Clustering Analysis (DEC, Xie et al., 2016), and slight variations of the original method. It starts from using a so-called autoencoder to encode the datapoints into a feature vector. These features are further trained using a cluster-refining loss which encourages the feature representation to exhibit a strong cluster structure, i.e. short distances within clusters, and large distances between clusters. This strategy has inspired several follow-up works (e.g. Guo et al. (2017), Guo et al. (2018)), which improve on this original idea.

Autoencoders

An autoencoder (Hinton and Salakhutdinov, 2006) is a special type of neural network architecture which consists of two parts: the *encoder* and the *decoder*. The encoder acts on the data and transforms it into a feature vector (somtimes called *latent* space). The decoder acts on these feature vectors and transforms them back into the data space. This neural network structure is then trained to reconstruct any given input sample, for example using a squared-error loss. Figure 2.3 shows visual representation of an autoencoder neural network.

More formally, the encoder is a neural network $f_\theta : \mathcal{X} \to \mathcal{Z}$ that maps from data space $\mathcal{X}$ to a latent space $\mathcal{Z}$, and the decoder is a neural network $g_\phi : \mathcal{Z} \to \mathcal{X}$ that maps from latent space $\mathcal{Z}$ to data space $\mathcal{X}$. In practice, both data space and latent space are usually vectors of real numbers, the latent space being smaller than the data space: $\mathcal{X} = \mathbb{R}^N$, $\mathcal{Z} = \mathbb{R}^M$ with $M < N$.

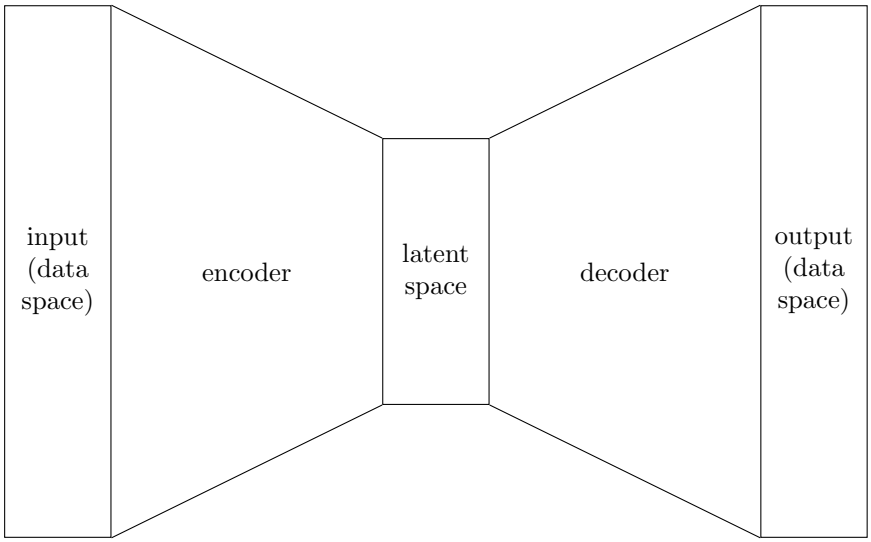


Figure 2.3: A visual representation of an autoencoder neural network

The autoencoder is often trained with a squared error loss. Using the notation from before, we collect all weights in a common vector $w = (\theta, \phi)$ and denote the dataset as $x_i, \dots x_n \in \mathcal{X}$. An individual data example is $x_i = (x_i^1, \dots x_i^N)$, and its reconstruction is $y_i = g_\phi(f_\theta(x_i)) = (y_i^1, \dots y_i^N)$. The loss of this data example x_i is the squared distance between x_i and y_i:

$$L_i(w) = \sum_{k=1}^{N} (x_i^k - y_i^k)^2, \tag{2.34}$$

and the total loss again is taken over all data examples:

$$L(w) = \frac{1}{n} \sum_{i=1}^{n} L_i(w). \tag{2.35}$$

We also use convolutional layers in autoencoder neural network architectures (*convolutional autoencoder*). Here, in the decoder we use so-called transposed convolutional layers, which allow the upsampling of a smaller feature map to a larger feature map. For more details, we refer to e.g. Dumoulin and Visin (2016).

Cluster Refinement Loss

Now DEC (Xie et al., 2016) is based on an autoencoder, and adds in a cluster assignment hardening loss, which encourages the latent space to be cluster-friendly by fine-tuning the cluster assignments.
The training starts with a pre-training of the autoencoder using the usual methods, before adding in the DEC loss. After pre-training, we are given embeddings z_i of datapoints x_i (i.e. $z_i = f_\theta(x_i)$). Using k-means in latent space, we initialize the centroids $\mu_j \in \mathcal{Z}$. Now we improve this clustering by iterating two steps:

- compute a soft assignment between the embedded data points and the centroids

- update the parameters of the encoder f_θ and the cluster centroids by learning from high-confidence assignments.

For the first step, to compute the soft assignment we measure the similarity between embedded points $z_i = f_\theta(x_i)$ and centroids μ_j using a t-distribution with degrees of freedom α:

$$q_{ij} = \frac{\left(1 + \frac{\|z_i - \mu_j\|^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}}{\sum_{j'} \left(1 + \frac{\|z_i - \mu_{j'}\|^2}{\alpha}\right)^{-\frac{\alpha+1}{2}}}. \tag{2.36}$$

Here, we interpret q_{ij} as the soft cluster assignment, i.e. it is the probability of assigning sample i to cluster j.
For the second step, we aim to refine the clusters by learning from high confidence assignmets with the help of an auxiliary target distribution. This is done by minimizing the Kullback-Leibler divergence, a common measure for the similarity of two probability distributions, between the soft assignments and the target distribution (which is still to be defined):

$$L = \mathrm{KL}(P||Q) = \sum_{i,j} q_{ij} \log \frac{p_{ij}}{q_{ij}}. \tag{2.37}$$

The target distribution is chosen in a way that achieves the following three objectives:

1. strengthen predictions
2. put more emphasis on high confidence points
3. normalize the loss distribution of each centroid (to deal with imbalanced clusters, for example one very large cluster)

For DEC, the authors chose the following target distribution:

$$p_{ij} = \frac{q_{ij}^2 / f_j}{\sum_{j'} q_{ij}^2 / f_{j'}}, \tag{2.38}$$

where $f_j = \sum_i q_{ij}$ are the soft cluster frequencies. Here, raising the soft cluster assignment probabilities q_{ij} to the second power corresponds to point 2, putting more emphasis on high confidence points. Normalizing by frequency of each cluster corresponds to point 3.
The training is then done by optimizing the neural network as normal, by computing the gradient of the KL-divergence loss with respect to z_i and μ_j and using mini-batch gradient descent and backpropagation to pass the gradients with respect to z_i to the encoder parameters θ.
For our experimental details and results on this method, see section 4.2.

2.2.3 Contrastive Learning

Now, we turn to our second kind of unsupervised deep learning methods, contrastive learning. The field of so-called "self-supervised" learning has quickly been gaining popularity in the last couple of years. In self-supervised learning, we use methods from supervised learning,

but without using actual human-annotated labels. The "labels" used are instead pseudolabels generated from the input data itself. A special case of self-supervised learning is contrastive learning. SimCLR Chen et al. (2020) is a "**Sim**ple framework for **C**ontrastive **L**earning of **R**epresentations" and was a very impactful work in the field of contrastive learning, by providing a simple formulation for the contrastive learning task, and extensive numerical studies showing the impressive results. Many subsequent developments in contrastive learning build on it.

Broadly speaking, contrastive learning methods work by *contrasting* similar data samples to dissimilar ones. We speak of *positive pairs* for similar samples and *negative pairs* for dissimilar samples. Mathematically, in contrastive learning we need for each data point x a probability distribution to sample a positive paired data point $x^+ \sim p^+(\cdot|x)$ and a probability distribution to sample a negative paired data point $x^- \sim p^-(\cdot|x)$. Often in contrastive learning, we use data augmentation, i.e. slight perturbations of an input image or seismogram, to create the similar samples. The objective of contrastive learning, achieved by defining the loss function accordingly, then is to train the representation of similar samples to be close in feature space and the representation of dissimilar samples to be far apart in feature space.

Different contrastive learning approaches differ in their way of defining the positive and negative probability distributions, the computation of the representation (i.e. the neural network architecture), and the specific loss function used to achieve the contrasting objective. The SimCLR framework is one approach, but there are others including MoCo (**Mo**mentum **Co**ntrast, He et al. (2020)) and BYOL (**B**ootstrap **Y**our **O**wn **L**atent, Grill et al. (2020)). In the following, we will describe a general contrastive learning framework following the description of Le-Khac et al. (2020), before detailing some of the specific approaches.

The data samples x, i.e. input images or seismograms, live in some input space $\mathcal{X}$. Sometimes it is helpful to think of the input sample and contrasting sample in terms of *query* and *key* respectively, similar to how it was described in He et al. (2020) and formalized in (Le-Khac et al., 2020, definition 1):

Definition 1. *A* query q *and* key k *are specific views of input samples.*

Here, the term *view* refers to augmented versions. The query is a view that we are comparing to the keys. If a key is similar to the query, the pair (q, k) is considered a *positive pair*, and if the key is dissimilar to the query, the pair (q, k) is considered a *negative pair*. The notion of similarity and dissimilarity is formalized by a similarity distribution (Le-Khac et al., 2020, definition 2):

Definition 2. *The* similarity distribution *is a joint distribution over pairs of input samples* $p^+(q, k^+)$. *A pair of query* q *and key* k^+ *is called a* positive pair *if it is sampled from this distribution.*

The notion of dissimilarity, i.e. when to consider a pair of query and key a *negative pair*, depends on the specific approach. Sometimes, a pair of query and key (q, k^-) is sampled explicitly from a dissimilarity distribution $p^-(q, k^-)$, and in other cases the dissimilarity distribution is not explicit but the pair (q, k^-) is considered negative if it is not sampled from the similarity distribution.

Another integral part of a contrastive learning method is the model. It transforms the input samples into the feature space (Le-Khac et al., 2020, definition 4):

Definition 3. *The* encoder *is a transformation f_θ with parameters θ that maps input samples $x \in \mathcal{X}$ to its features h the feature space $\mathcal{H}$:*

$$f_\theta : \mathcal{X} \to \mathcal{H}. \tag{2.39}$$

That is, the features are computed as $h = f_\theta(x)$.

The features are then further embedded in a projection space by the projection head (Le-Khac et al., 2020, definition 5):

Definition 4. *The* projection head *is a transformation g_ϕ with parameters ϕ that maps the features $h \in \mathcal{H}$ into a projection space $\mathcal{Z}$;*

$$g_\phi : \mathcal{H} \to \mathcal{Z}. \tag{2.40}$$

That is, the projections are computed as $z = g_\phi(h)$.

The projection head is only used in the training procedure, but is not needed to compute the features. This is done because contrastive learning was observed to work better this way: the loss is not applied directly on the feature space, but on the projection instead.
In order to train the encoder and projection head, we make use of some contrastive loss (Le-Khac et al., 2020, definition 6):

Definition 5. *The contrastive loss is a loss function that operates on the projections z and is designed in such a way that the projections of similar pairs (z, z^+) are encouraged to be close and the projections of dissimilar pairs (z, z^-) are encouraged to be dissimilar.*

Note that the contrastive learning framework is generally independent of the type of similarity distribution, and could therefore also be applied in a supervised setting (e.g. Khosla et al. (2020)): In this case, two input samples could be considered similar if they belong to the same ground-truth class, and dissimilar if they belong to different classes. The goal in contrastive learning is that positive pairs are semantically similar, i.e. they represent the same idea, but not similar in input space, in order to let the model learn semantic information. This could be achieved using human-annotated labels, but can also be achieved through other means like data augmentation. While the human-annotating approach is easy and effective, it also suffers from the problems we described before (i.e. it is expensive, it introduces bias). The unsupervised approach using data augmentation on the other hand can make use of huge unlabelled datasets without the need for labelling efforts, but more thought has to be put into how to achieve the desired queries and keys with similar semantic meaning but different input data.

Contrastive loss functions

Here, we will describe some of the loss functions used in different contrastive learning approaches (see e.g. Le-Khac et al., 2020 for an overview). Generally, contrastive loss functions consist

of two parts: A *scoring function* which measures the agreement between the query and key, and the actual loss function that encourages closeness for similar samples (i.e. maximize the agreement), and the opposite (minimize the agreement) for dissimilar samples.
Generally, the scoring function may be based on one of two measures of agreement: distance or similarity. Consider two embedded vectors q and k (Note that we use the notation q and k for the query and key both in input space and in embedded space).
The *distance* can be measured simply by a metric in the embedding space, for example the euclidian (L2-) distance

$$\text{dist}(q, k) = \|q - k\|_2 \tag{2.41}$$

or the L1-distance:

$$\text{dist}(q, k) = \|q - k\|_1. \tag{2.42}$$

The *similarity* on the other hand can be measured by the dot product:

$$\text{sim}(q, k) = q^T k. \tag{2.43}$$

The dot product however is unbounded and dependent on the length of both vectors, so that if this similarity was used directly, when training the embedded vectors q and k could be made arbitrarily small or large to minimize or maximize the agreement. What is used instead therefore is the *cosine similarity*, which normalized the two vectors before computing the dot product:

$$\text{sim}(q, k) = \frac{q^T k}{\|q\|\|k\|}. \tag{2.44}$$

This similarity is bounded between -1 and 1 and is 0 if and only if the two vectors are orthogonal.
Now there are several possible loss functions based on these scoring functions. Firstly, so-called energy-based margin losses are based on the distance scoring function. The first time a contrastive loss function was used, it was based on so-called energy-based models (Cun and Huang, 2005) and introduced by Chopra et al. (2005) (later also in Hadsell et al. (2006)). It takes as input a pair of query q and key k, and if it is a negative pair, it only contributes to the loss function if their distance is within some margin m:

$$\mathcal{L}(q, k) = \begin{cases} \text{dist}(q, k)^2 & \text{if } (q, k) \text{ is a positive pair} \\ \max(0, m - \text{dist}(q, k)^2) & \text{if } (q, k) \text{ is a negative pair} \end{cases}. \tag{2.45}$$

This loss function results in minimizing the distance between positive pairs, while maximizing the distance between negative pairs if the key is within some radius of the query. The total loss function is then taken as the sum over all such pairs (q, k).
On the other hand, the so-called *triplet-loss* (Schultz and Joachims, 2004, Weinberger et al., 2006) acts not on a pair of query and key, but instead on a triplet of query q, positive key k^+ and negative key k^-:

$$\mathcal{L}(q, k^+, k^-) = \max(0, \text{dist}(q, k^+)^2 - \text{dist}(q, k^-)^2 + m). \tag{2.46}$$

Here, again, the distance between the query and the positive key is minimized, while the distance between the query and negative key is maximized, and loss only acts on triplets that fall below a certain threshold (margin) m. Again, the total loss is computed over all such triplets (q, k^+, k^-), which are drawn from the input in some way yet to be specified.
In both of these cases, the choice of how to draw these query-key pairs or query-positive key-negative key triplets is important and quite difficult. The difficult task here is to find pairs that are similar enough such that they provide a useful learning signal. Such "mining techniques" are an integral part of the methods using these loss functions, and if they are not successful, lead to slow convergence of the models.
On the other hand, there are probabilistic loss functions based on noise-contrastive estimation (NCE, Gutmann and Hyvärinen (2010)) that work with the similarity-based scoring functions. These can be motivated from the softmax classification function and include for example the *normalized temperature scaled cross entropy loss (NT-Xent)* loss which is used for example in SimCLR. This loss acts on a given query q, a positive key k^+, and a set of keys $\mathcal{K}$ which contains k^+ and consists of negative keys otherwise:

$$\mathcal{L}(q, \mathcal{K}) = -\log \frac{\exp(\text{sim}(q, k^+)/\tau)}{\sum_{k \in \mathcal{K}} \exp(\text{sim}(q, k)/\tau)}, \tag{2.47}$$

where τ is a temperature parameter.
In the following, we will describe a couple of specific contrastive learning methods, namely SimCLR (which we will use for our experiments), Momentum Contrast (MoCo, He et al., 2020), and Bootstrap your own latent (BYOL, Grill et al., 2020).

SimCLR: A Simple framework for Contrastive Learning of Representations

The SimCLR framework Chen et al. (2020) is one of the most popular contrastive learning approaches. We will describe this framework in detail as we used it for our experiments. It consists of the following main components (referring to the general framework defined above):

- **data augmentation**. SimCLR samples positive and negative pairs by using data augmentation as follows. Let $x_i \in X \subset \mathcal{X}$ be a data sample. Then

  $$\tilde{x}_i = T(x_i) \tag{2.48}$$

  is a view (augmented version) of x_i. Here, T is the operator to perform the augmentation (which may consist of different transformations like distortion and adding noise). In SimCLR, the augmentation is always performed twice to get independently sampled augmented versions of the same data point x_i, i.e. $\tilde{x}_i$ and $\hat{x}_i$ (query and key). That means in this case, the query and key are sampled from the same distribution. Two such augmented samples are considered a positive pair if they originate from the same sample, and a negative pair otherwise.

- **encoder network**, or *backbone* f_θ. This is the neural network that maps the input samples x_i to a feature space $\mathcal{H}$:

  $$f_\theta : \mathcal{X} \to \mathcal{H}. \tag{2.49}$$

The parameters of the neural network are called θ. The features of each data sample are computed by this neural network:

$$h_i = f_\theta(\tilde{x}_i). \tag{2.50}$$

These features are what we ultimately use as the data representation. The network f_θ is usually chosen to be a convolutional neural network (CNN).

- **projection head** g. The projection head further maps the features (representations) to another space, the projection space $\mathcal{Z}$:

$$g_\phi : \mathcal{H} \to \mathcal{Z}. \tag{2.51}$$

The projection head's parameters are called ϕ. For each feature, we compute the projections:

$$z_i = g_\phi(h_i). \tag{2.52}$$

In SimCLR, the projection head is chosen to be a one hidden layer MLP.

For training, we sample a minibatch of N input samples, then apply the data augmentation operator twice on each input sample, to get two different views of each sample. We then have a batch of $2N$ views of input samples which we denote as $\{x_1, \ldots, x_{2N}\}$. A pair of samples is considered a positive pair if it originated from the same input sample and is considered a negative pair otherwise. Here, negative pairs are not sampled explicitly, but instead all other samples in the batch are treated as negative samples. Note that this means there might be other input samples in the batch that are, in fact, semantically similar to the query, but are then wrongly discouraged from having similar representations. SimCLR relies on the assumption that most other input samples in the batch belong to a different class.
From the input views, we use the encoder and projection head to compute the embedded vectors $\{z_1, \ldots, z_{2N}\}$. On these we compute the loss, which is chosen as the the normalized temperature-scaled cross entropy loss (NT-Xent) as introduced before (equation (2.47)):

$$\mathcal{L}_{ij} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \exp(\text{sim}(z_i, z_k)/\tau)}. \tag{2.53}$$

with the temperature τ and the cosine similarity:

$$\text{sim}(z_i, z_j) = \frac{z_i^T z_j}{\|z_i\| \|z_j\|}. \tag{2.54}$$

The final loss is the sum of all $\mathcal{L}_{ij}$ over all positive pairs (in both directions, i.e. both (i, j) and (j, i)).
For our experimental details and results on this methods, see section 4.3.

Momentum Contrast for Unsupervised Visual Representation Learning (MoCo)

Another contrastive learning framework is called **Mo**mentum **Co**ntrast (MoCo) (He et al., 2020). It is quite similar to the SimCLR framework, but differs in some crucial points.

in MoCo, the query x^q is again sampled from the input data and its embedding is computed through a query encoder network f_q. The keys are computed through a separate encoder network, which is not, like the query encoder network, updated by gradient descent updates, but is instead a momentum-updated version of the query encoder network. The key dictionary is also implemted as a queue, such that the framework reuses keys from previous batches. This allows the dictionary to be much larger than the mini-batch size while keeping the computation managable. Positive and negative pairs are again obtained from a data augmentation pipeline, where two different random views of the same sample are computed through data augmentation, and two views are considered a positive pair if they originate from the same sample.
To be more precise, consider a query sample x^q, of which we compute an embedding with the query encoder network: $z^q = f_q(x^q)$. This is *contrasted* with the key dictionary $z_1^k, \dots, z_K^k$. The key dictionary is computed through the key encoder network f_k, which is the result of a momentum-update of the query encoder f_q:

$$\theta_k \leftarrow m\theta_k + (1-m)\theta_q, \tag{2.55}$$

where θ_k and θ_q are the parameters of the key and query encoder networks, respectively, and m is the momentum update parameter. In experiments, it was determined that a rather large m (i.e. $m \approx 0.999$, resulting in a slowly changing key encoder) is the most beneficial. During training, for each minibatch we compute the corresponding keys with the current key encoder network. Note that therefore the key dictionary consists of keys which were computed with previous, now outdated key encoders. In each iteration, the oldest batch of keys in the dictionary is discarded.
The loss function used is the NT-Xent with dot product similarity:

$$\mathcal{L} = -\log \frac{\exp((z^q)^T z^{k_+}/\tau)}{\sum_{i=1}^{K} \exp((z^q)^T z_i^k/\tau)} \tag{2.56}$$

with temperature τ, the embeddings of query z^q and positive key z_+^k and dictionary of keys $z_1^k, \dots, z_K^k$.

Bootstrap your own latent (BYOL)

Bootstrap your own latent (BYOL) (Grill et al., 2020) is a special case of contrastive learning in that it does not use negative samples, but instead only contrasts the query with positive keys. This is surprising, since intuitively, we could assume this leads to a collapsed solution, where all inputs are mapped to the same value, which is not useful at all.
In BYOL, the general problem is described as follows. The idea that positive pairs should have a similar representation is a *prediction* problem in the sense that one side of a positive pairs should be able to be predicted by the other side. This, however, leads to collapse, as a constant mapping fully satisfies this prediction problem, since the constant values are trivially predictive of one another. The contrastive learning approaches we have discussed so far address this by instead considering a *discrimination* problem: We have a query with positive and negative keys, and from the query, they learn to discriminate between the positive and negative keys.
In BYOL, the collapsing solution problem is addressed in a different way. While the query is

encoded as before with a so-called *online* network (which is updated as usual with stochastic gradient descent), the corresponding positive key is encoded by a so-called *target* network. The first idea would be to keep this target network fixed but randomly initialized, such that the online network learns to predict the target representation of it's own positive pair. Indeed they show that this already leads to a successfully learnable framework, which performs better than a randomly initialized framework. However they achieve a much better performance when the target network is a slowly updated version of the online network (similar to MoCo). Intuitively, this is a process of predicting the target encoding with the online encoding, and then iterating this procedure, essentially bootstrapping its own latent representations by using an old version of the online network as target representation.
To formally describe the framework, consider an input sample $x \in \mathcal{X}$. From it we produce two different independent augmented views $\tilde{x}$ and $\hat{x}$. The online network consists of an encoder f_θ, a projector g_θ, and additionally a predictor q_θ. For ease of notation, we denote here the parameters of all components of the online network as θ, but each component uses its own subset of the parameters. For the target network, the same architecture is used, but with different parameters ξ, with the exception of the predictor, which is only present in the online network. The target network parameters ξ are updated as a momentum update of the online parameters θ:

$$\xi \leftarrow m\xi + (1-m)\theta. \tag{2.57}$$

From the two augmented views $\tilde{x}$ and $\hat{x}$, we get the encoded projections by applying the encoder and projector in sequence, from the online network and target networks respectively:

$$\tilde{z} = g_\theta(f_\theta(\tilde{x}))$$
$$\hat{z} = g_\xi(f_\xi(\hat{x})).$$

On only the online projection $\tilde{z}$, we additionally apply the predictor q_θ to get the prediction $\tilde{y} = q_\theta(\tilde{z})$ (which should predict $\hat{z}$).
Both the prediction and target projection are ℓ_2-normalized before applying the mean squared error as the loss function:

$$\mathcal{L}_{\theta,\xi} = \| \frac{\tilde{y}}{\|\tilde{y}\|_2} - \frac{\hat{z}}{\|\hat{z}\|_2} \|_2^2 = 2 - 2\frac{\langle \tilde{y}, \hat{z} \rangle}{\|\tilde{y}\|_2 \|\hat{z}\|_2}. \tag{2.58}$$

Finally, the loss is symmetrized by following the same procedure with $\tilde{x}$ and $\hat{x}$ switched (i.e. with $\hat{x}$ as the online prediction, and $\tilde{x}$ as the target) to get $\mathcal{L}'_{\theta,\xi}$, and adding the two components:

$$\mathcal{L} = \mathcal{L}_{\theta,\xi} + \mathcal{L}'_{\theta,\xi}. \tag{2.59}$$

Then at each training step, the loss $\mathcal{L}$ is used for a stochastic gradient descent update to the online parameters θ, and the target parameters ξ are updated via the momentum update (equation 2.57).

2.3 Data Augmentation for Seismological Data

Data augmentation is an integral part of most contrastive learning methods, but is also heavily used in many other neural network applications. Since the application of such methods to seismological data is still new, no standard repertoire of data augmentation strategies exists. For this thesis, we developed and implemented 4 data augmentation strategies, which we will explain in the following sections.
In general, data augmentation on time series data (like seismic data) has to be done carefully, because, compared to images, it is not as easy to decide visually whether a given augmentation leaves the class label untouched. For example, an important characteristic of seismic data is its frequency content, but this is not readily determined by looking at the waveform with the human eye.

2.3.1 Additive Gaussian Noise

Here, we simply add Gaussian noise of a certain variance σ^2, i.e. $\mathcal{N}(0, \sigma^2)$ distributed noise, to each data point in a seismogram (Figure 2.4).
Formally, we obtain a noise sample x^{noise} by sampling $x_i^{\text{noise}} \sim \mathcal{N}(0, \sigma)$. The new sample is then defined as

$$\tilde{x} = x + x^{\text{noise}}. \tag{2.60}$$

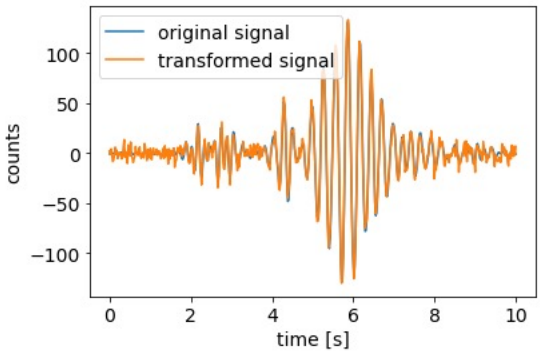


Figure 2.4: Example of Gaussian Noise: random noise is added to the seismogram.

2.3.2 Amplitude Stretching

Here, we augment the data by changing the amplitude (Figure 2.5). We manipulate the amplitude pointwise, but in contrast to the Gaussian Noise, the amount of stretching varies not randomly, but smoothly across the seismogram. Following the idea of Um et al. (2017), this is implemented as a pointwise multiplication with a discretized cubic spline varying around 1, which warrants a smooth variation of the stretching factor across the seismogram.
To be more precise, we sample $s \in \mathbb{R}^d$ that is a discrete version of a curve which varies around 1. This will be achieved as follows. We sample a set $\{S_1, \ldots, S_k\}$ of k support points evenly spaced across the length of the sample from a normal distribution centered at 1, i.e. $S_i \sim \mathcal{N}(1, \rho^2)$.

Then we interpolate these using a cubic spline to get a continuous function varying around 1. Subsequently, we discretize this function by sampling $s = [s_1, \ldots, s_d]$ evenly spaced at each of the d sampling points and pointwisely multiply the data by this:

$$\tilde{x} = x * s = [x_1 s_1, \ldots, x_d s_d]. \tag{2.61}$$

We use the same stretch factor across all three channels of each seismogram.

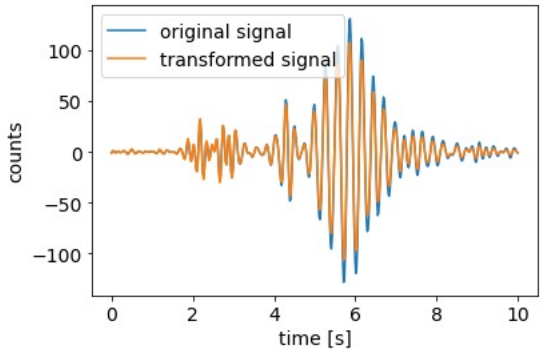


Figure 2.5: Example of Amplitude Stretching: The amplitude of the data is manipulated. Note how the amplitude stretching varies over time with stretching between seconds 2-3, and compression between seconds 5-8.

2.3.3 Time Stretching

Here, we manipulate the data examples in time by doing a global dilation by a factor. The factor is drawn from a uniform distribution between $1 - \eta$ and $1 + \eta$. Figure 2.6 shows the effect: The original and stretched seismograms match at 5 s in the center of the waveform. For a stretching factor of 1.1 for example, the sample at 4 s would be moved to 3.9 s, the sample at 3 s to 2.8 s, the sample at 6 s to 6.1 s and so on, stretching all times by 10 % relative to the central sample at 5 s.

More formally, this is achieved as follows. Since the original time series $x = [x_1, \ldots, x_d]$ is sampled at evenly spaced time steps, the values for the time steps may be chosen arbitrarily; for simplicity and to keep close to the implementation, we will use the indices, but starting at 0, i.e. $t_1 = 0, \ldots, t_d = d - 1$. Then we sample a stretching factor uniformly at random $\alpha \sim \mathcal{U}(1 - \eta, 1 + \eta)$ from an interval around 1. We define new time steps as

$$\tilde{t}_1 = d\frac{\alpha}{2}, \quad \tilde{t}_n = d - d\frac{\alpha}{2} = d\left(1 - \frac{\alpha}{2}\right) \tag{2.62}$$

and $\tilde{t}_2, \ldots, \tilde{t}_{n-1}$ evenly spaced in between. Then we compute samples $\tilde{x}_i$ by interpolating at these new time steps. When compressing the data, we are left with the problem of handling the boundary. Ideally, this could be solved by getting the actual data from the time series, since we do have access to that; however, since that is more complicated to implement, we compute the boundary by simply reflecting the data.

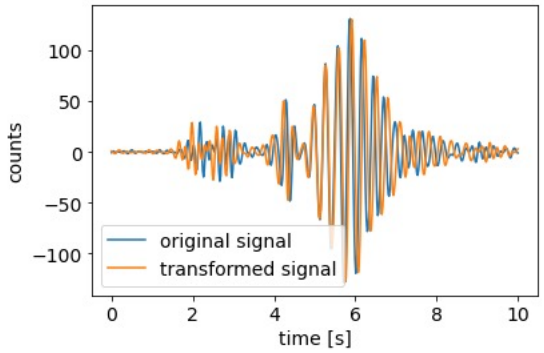


Figure 2.6: Example of Time Stretching: The data is stretched in time uniformly, keeping the middle point fixed.

2.3.4 Time Warping

Instead of time-stretching the seismogram uniformly, we also use a time-varying factor similar to the amplitude stretching strategy, which we call time warping (Figure 2.7). This results in time-compressing the signal in some areas, while time-stretching it in others.
Similar to Time Stretch, we achieve this by interpolating the time series $x = [x_1, \ldots, x_d]$ with time steps $t_1 = 0, \ldots, t_d = d - 1$ at new time steps. The new time steps are computed by generating a curve $s = [s_1, \ldots, s_d]$ varying around 1 in the same manner as for Amplitude Stretch (i.e. computing a discretized cubic spline, with the interpolation points drawn from a $\mathcal{N}(1, \xi^2)$ distribution). Then we compute the cumulative sum of s and in order to avoid extrapolating, constrain the result to be between $t_1 = 0$ and $t_d = d - 1$, which can be easily achieved by rescaling. These are then the new time steps at which to interpolate $\tilde{x}$.

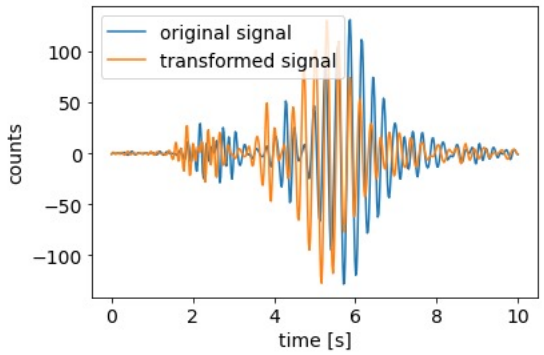


Figure 2.7: Example of Time Warping: Time Warp: The data is stretched/compressed variably over time. Note the time compression around second 0-6, then time stretching around seconds 7-10.

2.4 Evaluation Strategies

The question of how to evaluate the introduced machine learning methods is crucial. For this purpose, we created the labelled datasets NICE-eval-3 and NICE-eval-6, and compare the automatically computed clusters to the ground-truth labels. We keep the weights of the encoder fixed and compute the features for the dataset (h_i from equation (2.50)).
Firstly, we run a clustering algorithm (k-means) on the features with the number of clusters equal to the number of ground truth classes. We then evaluate the resulting clusters by comparing them to the ground truth classes in the following ways:

Clustering Accuracy

To compute the clustering accuracy, we simply compute what percentage of data examples are classified correctly (accuracy). Since in principle, the clusters may be arbitrarily assigned to ground-truth classes, we find the matching that yields the highest accuracy value. This can be efficiently achieved using the Hungarian matching algorithm Kuhn (1955).

Normalized Mutual Information

Normalized mutual information (NMI, Strehl and Ghosh, 2003) is a common measure for evaluating clustering performance. Mutual information is a concept from information theory which measures the amount of statistical dependence of two random variables. Normalized mutual information is scaled to lie between 0 (no mutual information) and 1 (perfect matching). This is another common metric for evaluation of clusterings when the ground-truth is available.

Additionally, we found that looking at a clustering with more classes gives some valuable insight (see Results section 4.3.2), which is why we introduce another metric that is based on another clustering with the k-means algorithm, but allowing for double the number of clusters.

Double Homogeneity

The homogeneity score (Rosenberg and Hirschberg, 2007) measures whether the clustering assigns only data points of a single ground truth class to each cluster, i.e. allowing one ground truth class to be split into mutiple clusteres. The score is computed via the conditional entropy of the ground truth class distribution given the proposed clustering. If each cluster contains data points of only a single ground truth class, the conditional entropy is 0 and the homogeneity score is 1. If on the other hand the ground-truth class distribution within each cluster matches the overall ground-truth class distribution, the homogeneity score is 0.

For the evaluation strategies, we use implementations from scikit-learn (Pedregosa et al., 2011).

Qualitative Evaluation Using Visual Tools

Furthermore, to make the evaluation more comprehensive, we include an evaluation on the unlabelled dataset NICE-application. The use of the labelled dataset has the advantage that quantitative measures can be computed. However, there are also some concerns using the labelled data for evaluation. Firstly, the labelled dataset is rather small and the manual labels (especially within the icequake class) may not be perfect. Secondly, the labelled dataset was obtained using a less sensitive STA/LTA trigger, so that it contains fewer hard-to-classify events. In general, it is difficult to devise useful performance metrics without using labels. Most evaluation scores like e.g. the silhouette score measure the amount of similarity within each cluster and between different clusters, where similarities within the clusters should be high and similarities between different clusters should be low. In our case, however, the features space itself is what is to be evaluated, so that it is not immediately clear whether similarity in this feature space is a useful tool.
Therefore, we also follow a qualitative approach, and evaluate the clustering of the unlabelled dataset based on

- the visual distribution of events in feature space and
- the events in each cluster.

To evaluate the visual distribution of events in feature space, it is common in the unsupervised deep learning literature to use t-distributed stochastic neighbour embedding (t-SNE, Van Der Maaten and Hinton, 2008). t-SNE produces a two- or three-dimensional embedding of a higher dimensional space (in our case, feature space), which can then be easily visualized on paper or on a screen (see e.g. Figure 2.8). Intuitively, it works by computing an embedding that most accurately matches the pairwise distances between data points. We can visualize this embedding as a two-dimensional scatter plot where each event is represented by a single point. We use the implementation from scikit-learn Pedregosa et al. (2011) with default parameters.
Further, to directly associate the scatter plot with the events in each cluster, we decided to develop a tool to augment this usual visualization of clusters. In addition to checking the visual separation of clusters, we can directly associate each point in the scatter plot to its corresponding waveform by hovering the mouse over it and showing the waveform in a separate plot. Figure 2.8 shows two screenshots of the visualization tool.

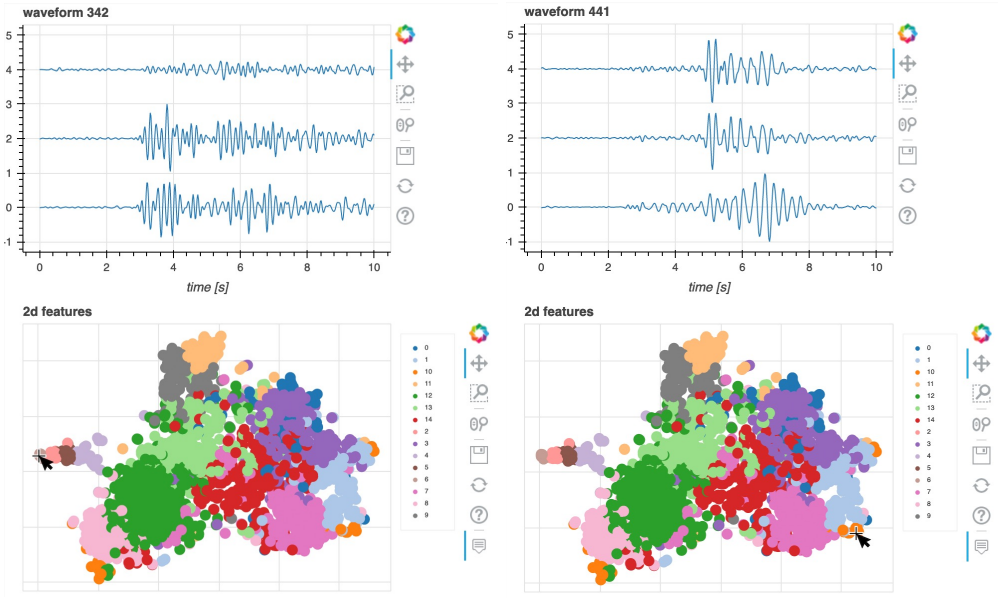


Figure 2.8: Screenshots of the visualization tool.

Chapter 3

Data

The geophysical observatory at Neumayer Station III maintains, among other scientific instruments, three permanent seismological stations in Antarctica, which have been recording continously for over 20 years (Wesche et al., 2016). The data is available at `http://doi.org/10.14470/NJ617293`, and we use parts of this data for our experiments. In this chapter, we first describe the seismological observatory at Neumayer station, and then show our steps of preprocessing the data before applying the machine learning algorithms.

3.1 The seismological network at Neumayer station

The permanently maintained seismological stations at Neumayer Station III are called VNA1, VNA2 and VNA3 and yield continuous seismic records with 50Hz sampling frequency (Fromm et al., 2018). The surroundings of the network and the three stations are depicted in Figure 3.1, along with Neumayer station itself.
Over the lifetime of the observatory the sensors were updated, and in the following we describe the current setup which has been active since 2010. The sensors for all stations VNA1, VNA2 and VNA3 are three-component Guralp CMG-3ESP broadband seismometers with a flat response between 120 s and 50 Hz. The continuous data streams are digitized by a Q330 data logger at a sampling rate of 50Hz, made available in real time, and archived for future use.
The sensor for VNA1 is located in a firn cave whose depth increases yearly due to snow accumulation, and is today about 13 m deep. The firn cave is located about 1.5 km south of Neumayer station on the Ekström Ice Shelf.
VNA2 is a located on the ice rise Halvfarryggen (colloquially called "Watzmann") about 50km south-east of Neumayer station. In addition, it is the central sensor of a seismological array with an aperture of 1.8 km which consists of 3 concentric rings of vertical short-period Mark L4 seismometers.
VNA3 is located around 90km south-west of Neumayer on the ice rise Sörasen (colloquially called "Olymp").
For this thesis, we focused on the data from the station VNA2. This is because surrounding array may be useful for future studies by providing more information on events like azimuth and slowness and therefore enable locating the event origin. So far, however, our methods only

use single station recordings. The data is in counts, and is proportional to ground velocity. As a first preprocessing step, all data was filtered by a 3-8 Hz Butterworth bandpass filter. This is a common processing step in seismology and years of experience have shown that this is the frequency band best showing icequake events in the Neumayer data.

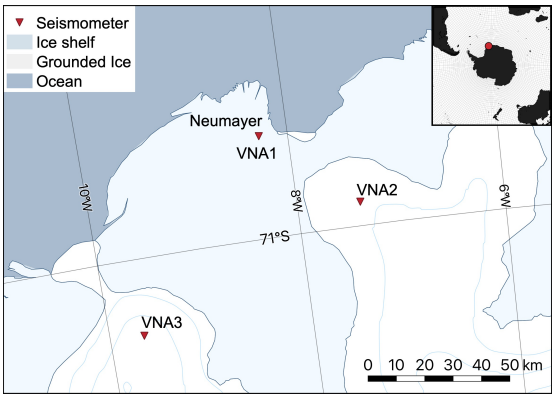


Figure 3.1: The seismic network around Neumayer station, Antarctica. Note in particular the location of VNA2, which is the station that provides the data used in this study.

Figure 3.2 shows some sections of the time series at different time scales: a whole month as a helicorder plot (Figure 3.2a), and a five-minute excerpt (Figure 3.2b). Fig. 3.2a shows the tidal influence on icequakes: the diagonal bands of high amplitude indicate increased icequake activity caused by tides. Fig. 3.2b shows individual seismic signals of interest. The recordings include many icequakes, but also earthquakes and so-called spikes, which are non-seismic signals that can be caused by electronic disturbances or recentering of the sensor. We are interested in the numerous icequakes. There are different types of icequakes with highly similar waveform appearances for each group. Therefore, we have two goals: The primary goal is to separate icequakes from other event types; and the secondary goal is to cluster the icequake signals into groups of the same type.

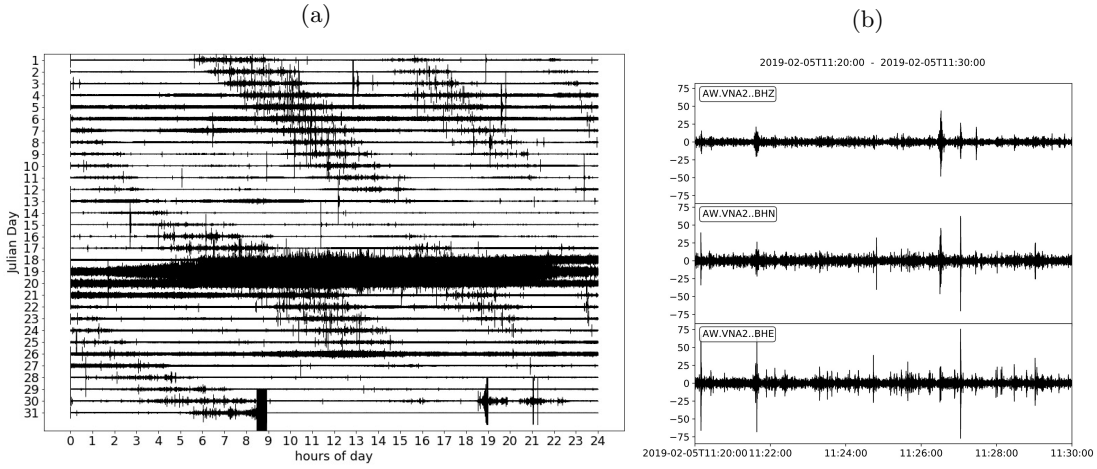


Figure 3.2: Example seismograms of station VNA2 at different time scales filtered with a 3-8Hz bandpass filter. (a) is a helicorder seismogram for the month of January 2019. Note the increased icequake activity clustered in diagonal bands, indicating a tidal cyclicity of icequake signals. (b) shows a 5 minute long excerpt of the three-components seismogram showing indvidual icequakes of interest.

3.2 Datasets pre-selected with STA/LTA

To reduce the amount of pure noise data examples, we pre-selected events by an STA/LTA trigger. Starting from the continuous data stream filtered by a 3-8 Hz Butterworth filter, by visual inspection, we decided on an STA length of 2 seconds, and an LTA length of 10 seconds. After running the STA/LTA trigger, we gain a dataset of short seismograms by cutting out 10 s long slices of the time series starting 4 s before the trigger. These length parameters were also chosen by manual inspection, and are chosen such that all relevant icequakes are contained within the slices.

We experimented with different trigger thresholds, and found that it was more beneficial to train the feature extractor on a less sensitive trigger (i.e. more clear events, higher threshold), while it is still possible to subsequently transform and cluster with a more sensitive trigger. Based on this experience, we extracted different datasets corresponding to different trigger thresholds. We call our datasets the **N**eumayer **ICE** (NICE) datasets, and Table 3.1 shows an overview.

The training dataset is called NICE-train and contains data from the years 2011-2019 with a high trigger threshold of 4 to produce a data set of clear, high-quality but unlabelled data for training.

Further, for the subsequent clustering we use data from 2019 and a lower threshold of 3 to include weak events. This dataset we call NICE-application.

Finally, for evaluation of the performance of the clustering, we manually produce a labelled dataset with clear events exceeding a detection threshold of 4.5. These were hand labelled in two different ways: Firstly, NICE-eval-3 with three classes: earthquake, spike, icequake. Secondly NICE-eval-6, where the icequakes are further divided into 4 different types of icequake

we call A, B, C and D (Figure 3.3a-d), resulting in 6 classes (earthquake, spike, A, B, C, D). Most prominent is a type of icequake that we call type 'A' (298 events, Figure 3.3a) with a clear surface wave train (Rayleigh waves) following a weak P phase arrival. Event types 'B', 'C' and 'D' have less clearly defined phase types with varying degrees of impulsiveness. Type 'B' (24 events, Figure 3.3b) is characterized by a strong first arrival on the east component followed by an approximately linearly decreasing envelope. Type 'C' (120 events, Figure 3.3c) has a less impulsive first arrival, and the envelope is first increasing and then decreasing for roughly the same amount of time. Type 'D' (96 events, Figure 3.3d) has a very impulsive arrival on the north component, followed by a more smooth waveform appearance on the east component. We also labeled 128 spikes, which appear as extremely short, sharp events on the filtered seismogram. In addition, the STA/LTA trigger also detected 105 events that are difficult to classify even for human experts. Some examples are shown in Figure 3.3e-g, and they were excluded from the dataset.

Table 3.1: Overview of the different **N**eumayer **ICE** (NICE) datasets.

Name	STA/LTA thresh.	# of examples	year	labels
NICE-train	4	100838	2011-2019	no
NICE-application	3	72036	2019	no
NICE-eval-3	4.5	878	2019	EQ, Spike, Icequake
NICE-eval-6	4.5	878	2019	EQ, Spike, A, B, C, D

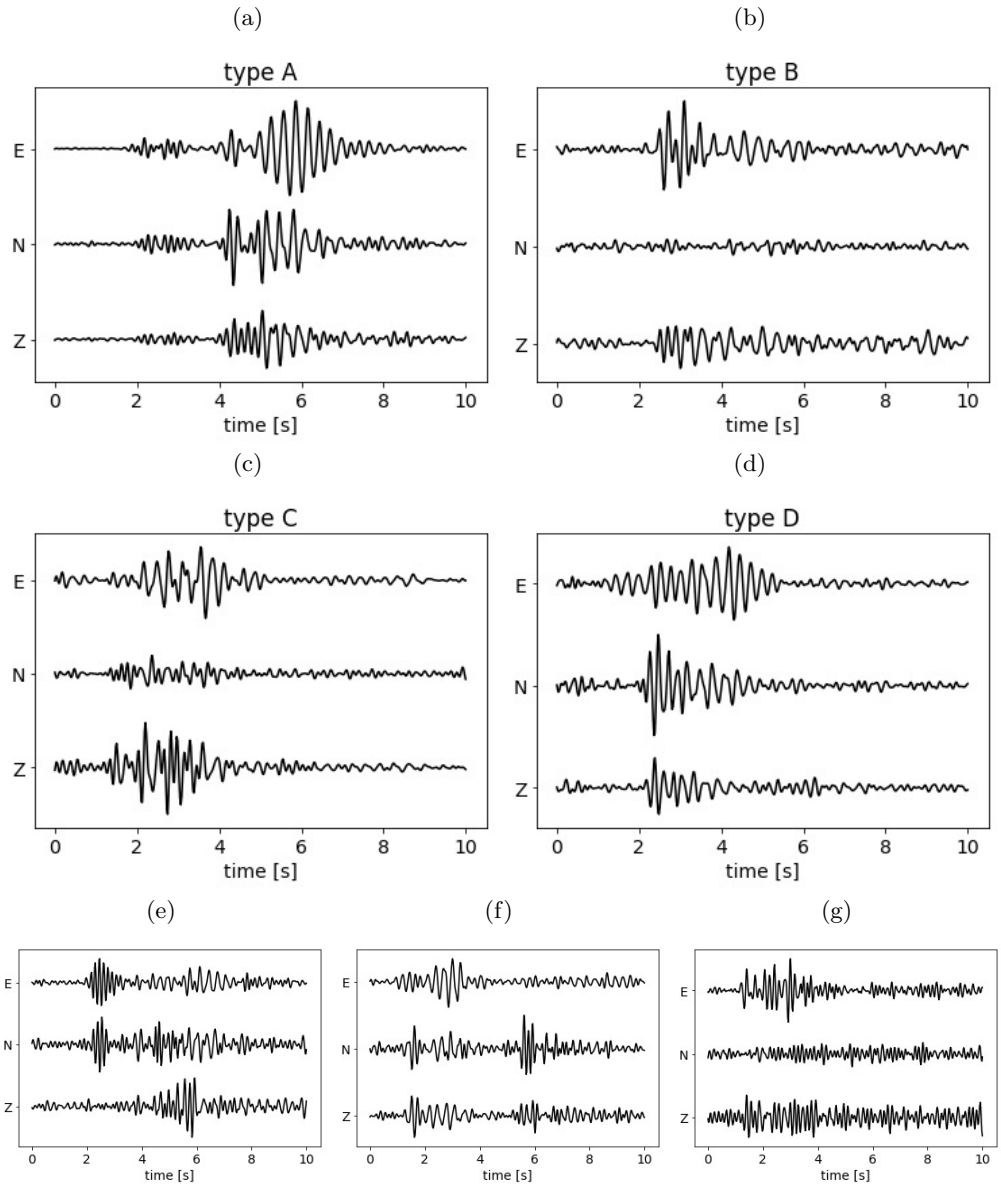


Figure 3.3: Some examples of events in the datasets. (a)-(d) are prototypes of the icequake classes of interest, types A, B, C and D respectively, as described. (e)-(g) are examples of more difficult events, which cannot be labeled by human experts, but need to be handled by the algorithm.

Chapter 4

Experiments

4.1 Explanation of the History of my Work

To understand the experiments section in my work, it will be helpful to first explain the timeline of the work.
In the beginning, the objectives and main emphasis of the project were not yet clearly set, so that the first task was to find and explore possibly suitable methods to analyze the dataset at hand. The original premise was a dataset of about 20 years of seismological recordings from the three permanent stations VNA1-3, which includes numerous icequakes that have not been analyzed so far, with the goal of creating a consistent icequake catalogue for this dataset. However, the working group at AWI also continuously collects additional seismological data in past, current and possible future temporary projects at various locations, including icequake studies around Neumayer, but also potentially Ocean-Bottom-Seismometers (OBS) or studies at volcanic sites. New seismological data is continuously being collected by seismologists at AWI but also around the world. Some ideas for analyzing such datasets include active learning, bayesian deep learning, transfer learning, unsupervised learning (either with or without neural networks) and random forest apprpdfoaches with manual feature extraction. Some of these ideas have been explored to varying degrees in the literature, but the task of applying machine learning approaches to seismological data is still very much in an "experimental phase". Many new methods are tried and put on the market, but there exist no clear best practices and not even consistent benchmark datasets have been established.
In our specific case, we had to decide between focusing on the original task of creating an icequake catalogue for the historical Neumayer dataset from the permanent stations, or whether to also take into account the numerous project datasets with different kinds of seismological signals.
After some experimentation and discussion, we came to the conclusion that one of two possibilities would be worth pursuing:

- For the task of creating an icequake catalogue for the historical Neumayer dataset, it would be best to implement an *active learning* or *transfer learning* approach, which are deep learning techniques that are specifically designed to deal with a small amount of labels in large amounts of data. Here we would need to put some work into manually

labeling at least a small amount of the seismological data in order to train the machine learning algorithms.

- When taking the diverse project data into account, it would be quite appealing to develop an algorithm that works in a fully *unsupervised* fashion. This is the somewhat more ambitious route, where no data is manually labelled and the algorithm should automatically find and group different types of icequakes in the data. It is usually intended as a first data exploration step.

After some more consideration and discussion, we decided to experiment with the fully unsupervised approach. The main reason was the prospect of a fully automated icequake-catalogue-algorithm for newly collected datasets, which may be used and re-used for any projects in the future.
Of the available unsupervised learning methods, we further decided to go for the option of unsupervised *deep* learning as opposed to working with more traditional methods on manual features, for two reasons:

- The field of unsupervised deep learning is a quite active field in the machine learning community with many new methods being developed, many of which have not yet been applied to the domain of seismology. This was an opportunity to close that gap.
- Manual features have the potential of introducing bias since the features are already designed by the expert with the expected classes in mind.

Of the available unsupervised deep learning approaches, we first focused on the deep clustering method DEC, which was one of the most influential deep clustering methods in recent years.
After a couple of months of testing this method, trying different sub-datasets and trying to tune the optimal parameters, we were not quite satisfied with the results. Also so far, we had been evaluating the clustering results in absence of a labelled dataset only by visually checking examples in the clusters for similarity. This led us to the realization that it would be essential to make some effort of manually labelling a certain amount of data, in order to be able to objectively evaluate any method by testing it on this dataset. For this purpose, we spend some weeks manually checking the dataset for icequakes and labeling a small amount of these for the creation of the labelled dataset. In this course we decided to focus our efforts on one of the stations, VNA2, to evaluate the methods. VNA2 was selected because of the surrounding seismic array which could potentially be used for further analysis and evaluation of the results. Further, for the purpose of evaluating the methods not only on the labelled dataset but also on the larger amount of unlabeled data, we developed the visualization tool (see section 2.4), which allowed us to directly visualize the events in each cluster.
After the work of manual labelling and evaluation considerations, since the results so far from the deep clustering method DEC had not been satisfactory, we decided to focus on a different new unsupervised deep learning method: contrastive learning. This was a very new tool being developed primarily in the field of computer vision, and it showed remarkable results, so that we decided it was worth trying to transfer these results to the domain of seismology. For this purpose, it was first necessary to develop data augmentation strategies for seismology, which is an integral part of contrastive learning methods, but also quite useful for other deep learning

applications.
After developing and implementing these augmentation strategies, we applied mainly the contrastive learning approach SimCLR, and did extensive tests on the labeled dataset as well as the larger unlabeled dataset using different evaluation strategies, in order to get the best results from this method.
With this historic evolution of the project in mind, below we first describe our results from the DEC experiments, which are not very expansive since they were abandoned after they were deemed not satisfactory enough. Note that also we show our results on the newly defined datasets, which were only computed after we decided to not pursue this approach further. Afterwards, we explain our experiments with SimCLR. In the discussion afterwards, we will address the possibilities of further experiments to extend this work, specifically further experiments for the DEC method which would be needed for a fair comparison of the two approaches we pursued.

4.2 DEC on Neumayer Data

For completeness, we show here the results of Deep Embedded Clustering on data from Neumayer station. We show results on the datasets we described in chapter 3. Note that the experiments were originally done on different, not so curated, datasets. Therefore, the decision to abandon this line of work was not based on these results specifically, but on other results. Also, the results may well be improved with more careful parameter tuning.
For our DEC experiments, we used an autoencoder based on a simple one-dimensional convolutional neural network architecture based on an architecture that was previously used in seismological deep learning models Ross et al. (2018). The detailed architecture is shown in Table 4.1.

Table 4.1: Model architecture of the convolutional autoencoder used for DEC experiments, adapted from Ross et al. (2018). Here, we use a convolutional autoencoder with two fully connected layers. After each convolutional layer, we apply batch normalization. The latent space is the output of layer 4, and the output of layer 5 has the same size as the input to layer 4 (i.e., it is the same architecture in reverse).

layer	type	# channels	filter size	stride
1	Conv	32	21	2
2	Conv	64	15	2
3	Conv	128	11	5
4	linear	50		
5	linear	(valid)		
6	TransConv	64	11	5
7	TransConv	128	15	2
8	TransConv	3	21	2

We trained two DEC models: One on NICE-application (unlabelled data from 2019), and

one on NICE-eval (hand-labelled dataset with data from 2019). We start by showing the results on the labeled dataset NICE-eval, then show the results for the unlabeled dataset NICE-application.

4.2.1 Performance on Labeled Dataset

For the DEC model on the labeled dataset, we pre-trained the Autoencoder on the NICE-application dataset, and then ran the DEC training on the NICE-eval dataset. We used 6 classes, the same number of classes as we manually identified.
Figure 4.1 shows a two-dimensional t-SNE visualization of the features of the dataset NICE-eval-6. The different clusters are very sharply separated, since DEC specifically trains the features to behave that way. However, only one cluster corresponds with the manual labels: The earthquakes are mostly well-separated from the rest of the classes. The spikes and the different types of icequakes, however, are not separated into clusters by this method, which is part of the reason why we stopped working on this method.

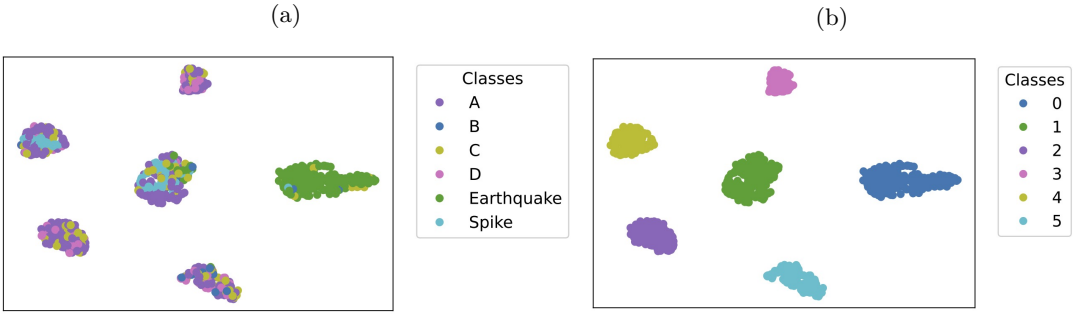


Figure 4.1: A two dimensional visualization of the features of the dataset NICE-eval-6 with (a) six classes manually labelled and (b) clustering produced by DEC with six classes. The embedding is produced with t-SNE.

4.2.2 Unsupervised Clustering Evaluation

For the DEC model on the unlabeled dataset, we used the NICE-application dataset for both the pre-training of the autoencoder, and for the subsequent DEC training. Here we use 15 classes, which is slightly more than the number of event classes we would expect in this dataset. In our SimCLR experiments (which we will describe shortly), we included more comprehensive testing to determine the optimal number of clusters. For this recomputation of our DEC results, we decided to simply use the same number of clusters as we determined there.
We plot histograms of event occurrence over time for the month of March 2019 (Figure 4.2a) and the last couple of days of March 2019 (Figure 4.2b) together with the tide (CATS2008 model, Howard et al., 2019) and wind speed (ERA5-Land dataset, Muñoz-Sabater, 2019). Furthermore, we plot a t-SNE representation of a subset of the features with colors indicating the cluster (Figure 4.2c).
The results again show well separated clusters (Figure 4.2c), since DEC specifically trains the features to behave that way. This, however, does not say anything about the quality of these

clusters. Therefore, we look to the time distribution of events in each cluster (Figures 4.2b and 4.2b). Here, we can see that different clusters show clearly different behaviour. Specifically, clusters 1, 2, 10, 11, and 14 show bursts of activity around March 9 and March 14-15, coinciding with prominent storms during this time. All other clusters seem to show some tidally controlled patters, which indicates that they likely contain some icequakes. We see these same patterns in the results of our SimCLR model, and will go into some more detail in the corresponding section below.

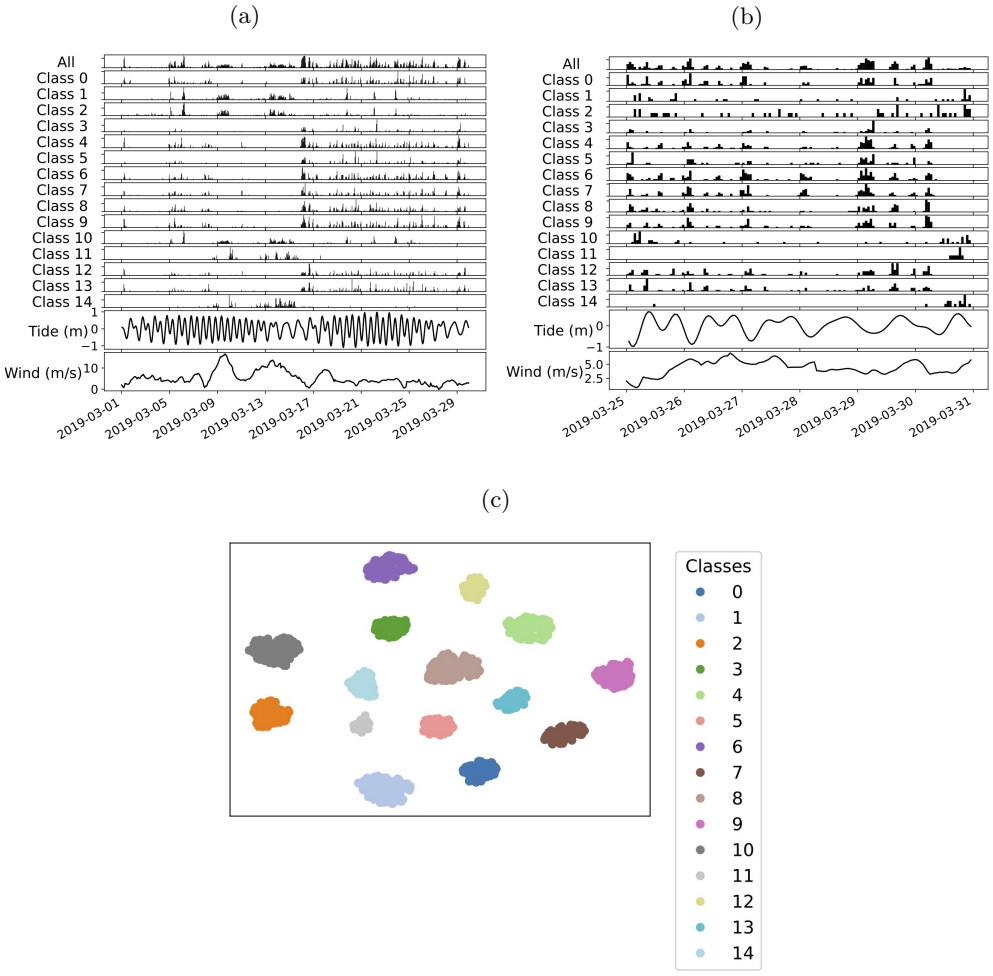


Figure 4.2: Histograms of event times and tide for (a) the month of March 2019 and (b) the days March 25-30. Note the differences in time distributions of the different classes. Also note the peak in activity of certain classes during falling tide. (c): A t-SNE visualization of (a subset of) the features of NICE-eval-unlabelled. Colors indicate the predicted cluster.

In Figure 4.3, we further plot a t-SNE representation of the features of the labelled dataset under the DEC model trained on the unlabelled dataset. We see that again, the earthquakes are well separated from the rest of the classes, but the spikes and different types of icequake are not well separated.

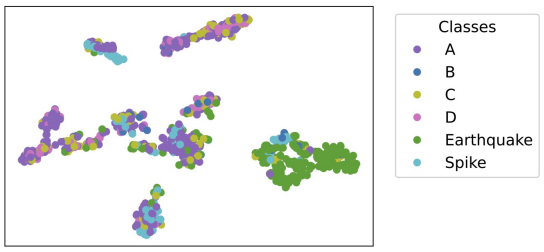


Figure 4.3: A t-SNE representation of the features of the labelled dataset under the DEC model trained on the unlabelled dataset.

All of the above results have to be treated with caution, since they have not undergone rigorous parameter tuning for reason we laid out in section 4.1. The following observations are, however, also confirmed by other publications using DEC for clustering seismological data (e.g. Jenkins et al., 2021, Ozanich et al., 2021).
In general, we notice that DEC separates the cluster very well, but it is less clear whether the separated clusters are actually useful. This is especially clear in Figure 4.1a, where we can clearly see one well separated cluster which consists of earthquake events, but in the other clusters, it is impossible to distinguish clusters belonging to specific ground-truth classes. In particular, not even the spikes are well separated from the icequakes, even though we expected these to be rather easy to separate. This indicates that the clusters being well separated also in the unsupervised case does not neccessarily imply a good clustering in data space, and more analyses must be done like e.g. our analysis of the event times histograms.
We found that DEC is generally heavily dependent on that initial autoencoder and k-means clustering. The subsequent DEC training works to considerably sharpen those clusters in feature space, but that does not neccessarily mean an improvement of the clustering in data space if the initial clustering is not very good in the first place. DEC tries to improve on the initial clustering by putting more emphasis on the high-confidence assignments, which means that the initial clustering would have to be decent at least for the high-confidence samples. In the experiments however, it is not clear whether this is always the case.

4.3 SimCLR on Neumayer Data

In contrast to DEC, SimCLR is not a clustering method. We use SimCLR to train a feature extractor, and then subsequently use a clustering method (k-means) on these features. This is in contrast to DEC, where the clustering is learned at the same time as the features, so that the clustering influences the learning procedure. Therefore, our approach to unsupervised clustering here (SimCLR then k-means) is a two-step approach, whereas DEC is a unified clustering and feature learning framework.
Our SimCLR models were trained on the NICE-train dataset and evaluated on the NICE-eval and NICE-application datasets as introduced in chapter 3. As the neural network encoder (backbone) for SimCLR training, we use a one dimensional version of the ResNet18. To decide on a neural network architecture for the encoder network, we performed our first experiments with a neural network architecture that was used previously in a seismological setting (Ross et al., 2018), see Table 4.2. We experimented with optimizing in the neighbourhood of those parameters through hyperparameter search. Better results were obtained when using a one dimensional version of another popular neural network architecture from the literature, the ResNet architecture (He et al., 2016). We use the implementation of a one dimensional ResNet by Hong et al. (2020), and the architecture is depicted in Figure 4.4. For the projection network, we use a two-layer MLP with 128 nodes each and a ReLU activation.

Table 4.2: Model architecture of the CNN from Ross et al. (2018). CBP = convolution, batch norm, pooling; FB = fully connected, batch norm; F = fully connected

layer	1	2	3	4	5	6
type	CBP	CBP	CBP	FB	FB	F
#channels	32	64	128	256	256	1
filter size	21	15	11	-	-	-

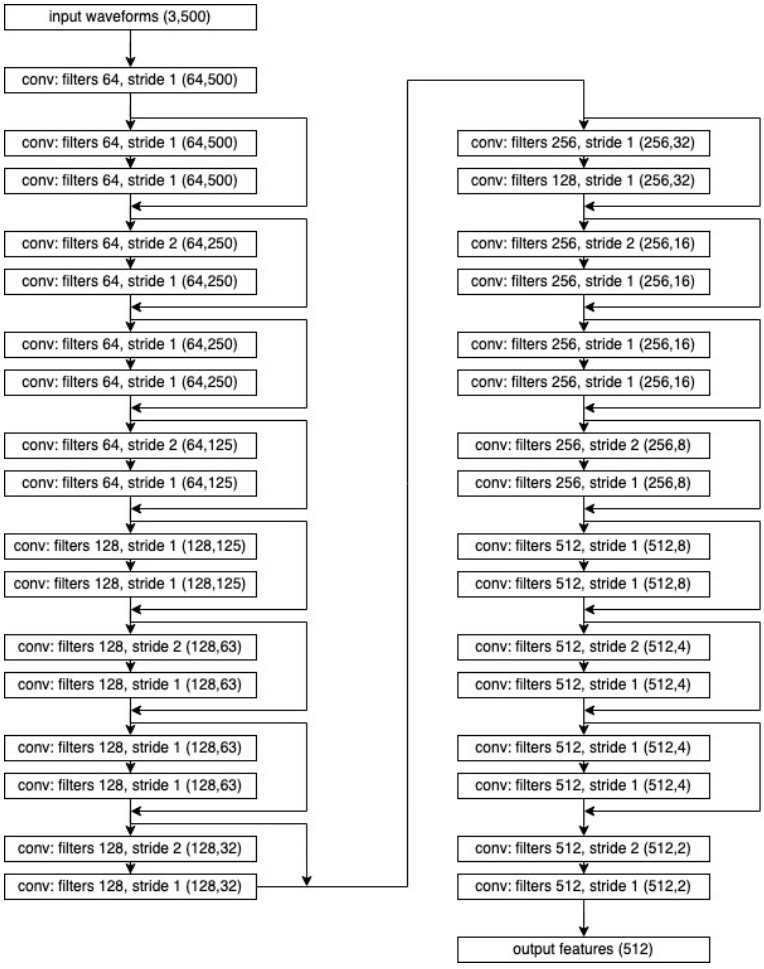


Figure 4.4: ResNet Architecture.

The hyperparameters were chosen as the default hyperparameter of SimCLR (temperature $\tau = 0.5$, weight decay 10^{-6}).
In the following, we analyze different data augmentation strategies to determine their hyperparameters. Then, we evaluate on the labelled dataset (NICE-eval) before analysing the performance on the larger unlabelled dataset (NICE-application).

4.3.1 Testing Data Augmentation Strategies

First, we set the level at which to perform each augmentation strategy, i.e. the hyperparameters σ, ρ, η and ξ. Our experience has shown that physically sensible levels of augmentation do not always lead to the best performance. Therefore, instead of setting the augmentation levels manually, we tested different levels of each augmentation strategy by training the SimCLR framework using only this strategy.
To evaluate the performance, we train the model for 50 epochs and subsequently compute the features of the labelled dataset NICE-eval-3. On these features, we compute the different evaluation metrics. Since the training depends on random initialization, we ran the training three times for each parameter and averaged the performance. The results are summarized in

Figure 4.5.

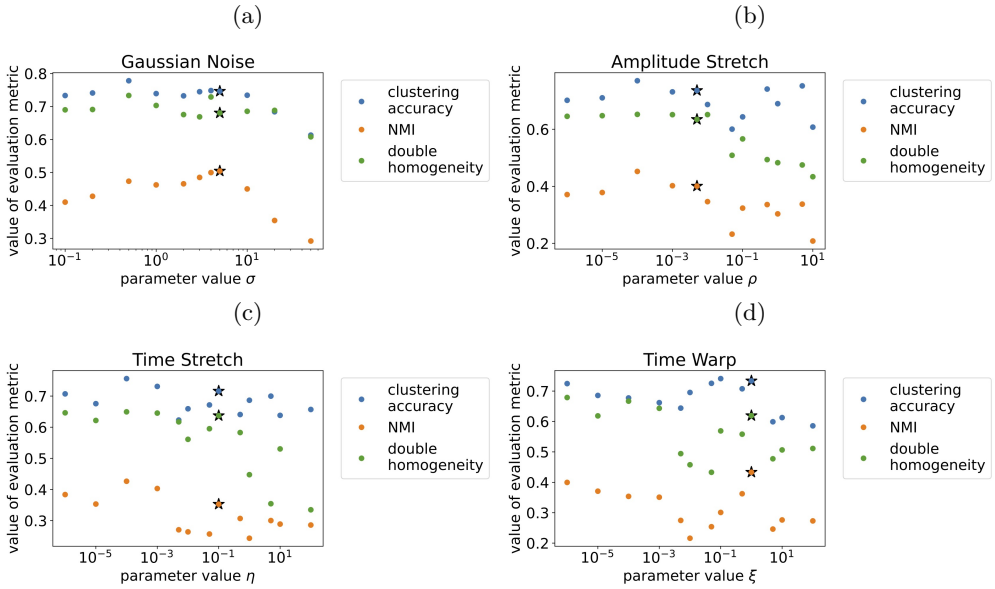


Figure 4.5: We test the level of each augmentation by varying the parameters (σ, ρ, η, ξ for the respective strategies, as introduced above). The chosen values are marked by stars.

The parameter ranges we tested include values that are physically not meaningful, since they result in either almost unchanged or entirely distorted seismograms. We make the assumption that the contrastive learning method benefits from data augmentations which are as strong as possible while still leaving the modified seismogram belonging to the same class, since it has been found (Chen et al., 2020) that contrastive learning benefits from strong data augmentation. Therefore, we choose the highest parameter value which still shows good performance.
For Gaussian Noise, the NMI curve shows a clear peak of performance at $\sigma = 5$. The other evaluation metrics show a roughly constant performance for parameter choices smaller than this, but a decline in performance for higher values. Since we generally lean towards higher parameter choices for physical reasons, we choose the parameter $\sigma = 5$.
For Amplitude Stretch, clustering accuracy and NMI show a peak at $\rho = 0.0001$, while the double homogeneity metric again shows roughly constant performance for smaller values but a sharp decline in performance after $\eta = 0.005$. Therefore, again based on the assumption of prefering higher parameter values, we choose the highest parameter that still leads to good performance under the double homogeneity metric, i.e. $\rho = 0.005$.
For Time Stretch, similar to Amplitude Stretch, the clustering accuracy and NMI show a peak at $\eta = 0.0001$, and double homogeneity shows a sharp decline in performance after $\eta = 0.1$. Therefore, for the same reason, we choose $\eta = 0.1$.
For Time Warp, the performance is unstable. We do again observe a slight decline for high values after a peak in performance at $\xi = 1$, which is therefore the parameter we choose.
In summary, we do generally see a decline in performance for high parameter values (entirely distorted seismograms), but especially for smaller parameter choices where the seismograms are almost unchanged, the results are mixed. Therefore, we use these tests as an indication, but we

also use our knowledge and assumption about the contrastive learning method (i.e. that higher parameters are preferable) to make our choices. We also emphasize that this hyperparameter choice is not too critical, since changing these hyperparameters within the tested range leads to very similar final results.

4.3.2 Evaluation of Augmentations in Pairs

To gain more insight into the effect of different augmentations, we perform further numerical tests where we combine the augmentation strategy in pairs, following the same approach as Chen et al. (2020). We test each combination of two augmentation strategies with the levels as determined before. This analysis is meant to evaluate the relative importance of each augmentation strategy.

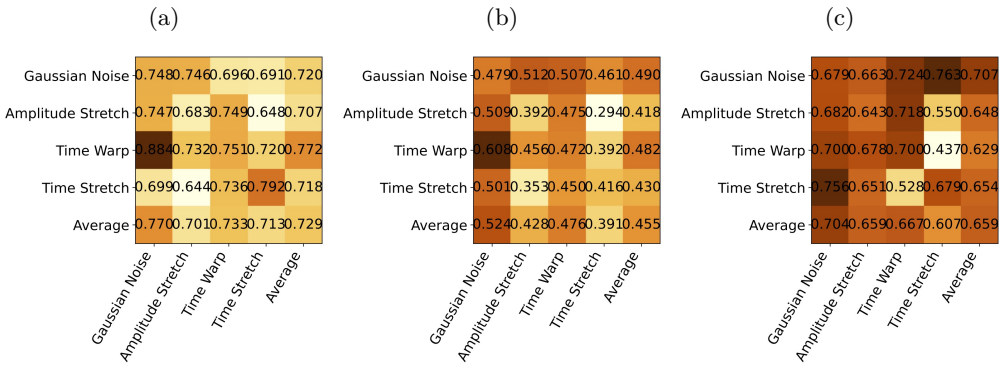


Figure 4.6: Evaluation of augmentations in pairs for different evaluation metrics. (a) clustering accuracy, (b) NMI, (c) double homogeneity. The color represents the performance values: the darker, the better. When combining two of the same augmentation (on the main diagonal from top left to bottom right), only the one augmentation was used; off the diagonal, the augmentations were used in sequence. On the left is the first augmentation, on the bottom the second augmentation. In the last rows and columns, we compute the average for that specific augmentation over all pairs.

In Figures 4.6a and 4.6b we see one augmentation combination stand out: Time Warp with Gaussian Noise. It does not, however, stand out as much in Figure 4.6c. To investigate this discrepancy, we take a closer look at the models in question in Figure 4.7.

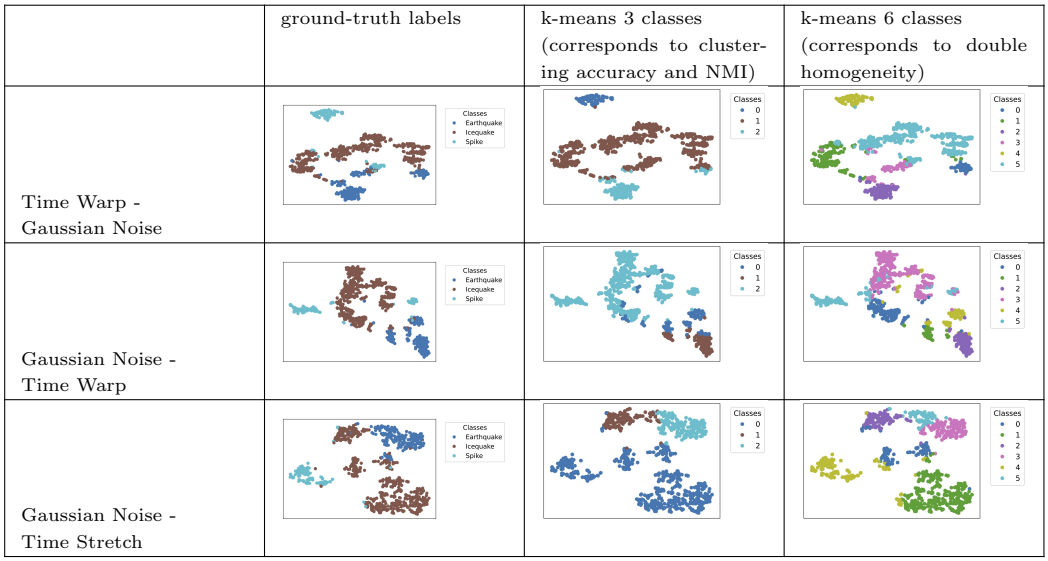


Figure 4.7: t-SNE representations of the features of the labelled dataset, using the models resulting from the different augmentation combinations.

As Figure 4.6 had indicated, the Time Warp - Gaussian Noise - model is quite successfully clustered into the 3 ground truth classes by the k-means algorithm with 3 classes. However, visually, the separation of ground truth classes is not much better than, for example, the Gaussian Noise - Time Warp - model, although the 3-class-clustering metric is much worse. This may be due to effects of the 2D-embedding, but may also point to a flaw in the metric. For this reason, we introduced the 6-class clustering metric (double homogeneity), which seems the preferable metric in this case. The Gaussian Noise - Time Stretch - model is the best-performing model under the double homogeneity metric. This is because when looking at the clustering with 6 classes, most clusters contain events that belong to the same ground-truth class.

Now in Figure 4.6, the rows and columns corresponding to Gaussian Noise generally lead to the highest (darkest) accuracy values. This means that Gaussian Noise seems to be the most effective augmentation strategy. This may be due to the fact that it represents real seismological variability best: Different noise levels occur naturally on seismograms.

In Figure 4.6c, combination of Time Stretch with Gaussian Noise shows a strong perfomance. The combination of Time Stretch with Time Warp, however, shows a rather bad perfomance; this was to be expected as combining two different time-manipulating strategies does not make much sense.

Lastly, Amplitude Stretch shows a relatively consistent performance. This augmentation strategy seems seismologically somewhat more plausible than for example Time Warping, as variation in the amplitude is common in seismology.

We also note that composing different augmentation strategies is important, as evidenced by the fact that the best performing models are off the diagonal. Overall, many different combinations show comparable performance.

For the final analysis, we are using a model trained with all four data augmentation strategies, but randomly selecting different augmentation strategies at each training step. There are sev-

eral reasons for this: Combining multiple augmentation strategies is how contrastive learning is generally done in the literature (e.g. Chen et al., 2020, Al-Tahan and Mohsenzadeh, 2021), and also we expect that more variability in the augmentation strategies leads to better performance. Indeed, the model trained with all four augmentation strategies outperforms any of our models trained with two augmentation strategies on the double homogeneity metric.

4.3.3 Performance on labelled dataset

From now on we consider a model trained with a combination of all four data augmentation strategies: Gaussian Noise level $\sigma = 5$, Amplidute Stretching level $\rho = 0.005$, Time Stretching level $\eta = 0.1$ and Time Warping level $\xi = 1$. We combine these strategies at each training step as follows: We randomly and independently choose with probability 0.5 whether Gaussian Noise, Amplitude Stretch and either of Time Warp or Time Stretch is used or not. The two time-manipulating strategies Time Warp and Time Stretch we do not combine. Instead, if we choose to do one of the time-manipulating strategies, we select Time Stretch or Time Warp with equal probability. This process leads to sometimes using three augmentations, sometimes any two augmentations, and sometimes only one or no augmentation. We also randomize the order of augmentations. The model was trained for 1000 epochs.
To evaluate the quality of the learned features on the manually labelled dataset, we use the quality metrics as described in the evaluation section. We compute these metrics on the dataset NICE-eval-3, i.e. treating all four icequake classes as a single class (resulting in three classes icequake, earthquake, and spike). As a simple baseline, we compare our features from SimCLR to those computed by a principal component analysis (PCA). For the PCA, we use the implemention from scikit-learn (Pedregosa et al., 2011) with 50 components. The results are summarized in Table 4.3. Note that this table is of limited quantitative value, but indicates that our learned method outperforms classical data analysis methods like PCA.

Table 4.3: Evaluation on the labelled dataset NICE-eval-3. Our learned SimCLR model clearly outperforms the classical data analysis tool PCA.

	PCA	SimCLR
Clustering accuracy	0.30	0.73
NMI	0.09	0.43
Double homogeneity	0.16	0.77

Figure 4.8 shows a 2-dimensional representation of the features of the labelled dataset with the colors indicating ground-truth labels. The embedding was produced by t-SNE (Van Der Maaten and Hinton, 2008). Note that the absolute axis values are not of interest and therefore omitted.
The scatter plot shows that the ground truth classes are mostly well separated in the feature space, especially if we only care about separating icequakes from earthquakes and spikes. The different types of icequakes are much more difficult to separate, which was expected since this holds true even for human analysts.
When clustering the data with k-means into six classes, the icequakes are all put into a single

cluster, and the earthquake class is split into multiple classes. This indicates that the algorithm finds a higher variability within the earthquakes than within the icequakes.

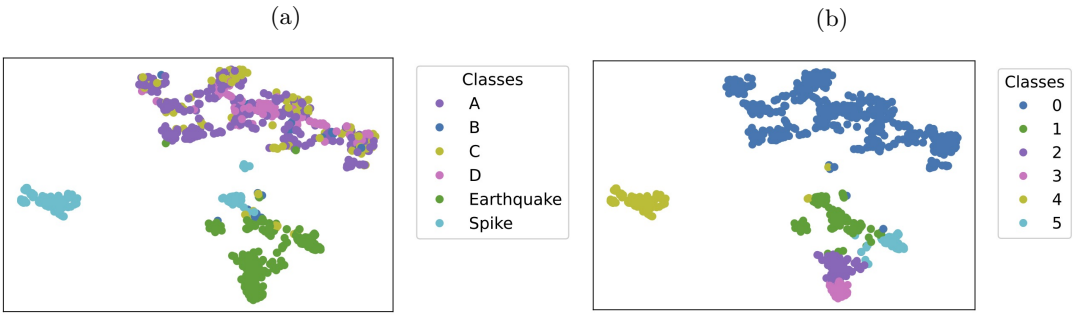


Figure 4.8: A two dimensional visualization of the features of the dataset NICE-eval-6 with (a) six classes manually labelled and (b) clustering produced by k-means with six classes. The embedding is produced with t-SNE. Icequakes, earthquakes and spikes are well separated from each other. The different types of icequakes are not well separated, which was expected since this holds true even for human analysts. The clustering puts all icequakes in a single cluster and splits the earthquakes into different clusters.

4.3.4 Unsupervised Clustering Evaluation

As a last step, we now apply the algorithm to the unlabelled data set NICE-application. This is an important step, since it is what the algorithm is intended for. We ran the k-means algorithm with a pre-defined number of clusters on the features of the dataset NICE-application, i.e. the events detected in 2019. Then we examine whether properties of the extracted groups differ and event types of different physical origin can be discriminated.

Choosing the Number of Clusters

Firstly, choosing the number of clusters to use in the k-means clustering is not trival. In k-means clustering, the number of clusters is a parameter that has to be decided on beforehand. To determine the optimal number of clusters, we ran the k-means algorithm with this parameter between 2 and 20, and computed the average within-cluster entropy in each case. Intuitively, the entropy represents how much variation any single cluster exhibits. This is averaged over all clusters.

The trend is shown in Figure 4.9. The average within-cluster entropy is expected to decrease with rising number of clusters, and the optimal number of cluster is assumed to be at a point where the graph shows a sharp turn ("elbow"). An elbow can be seen at around 12 clusters. We round that number up to 15 clusters, since similar cluster can be still joined together in a subsequent manual analysis.

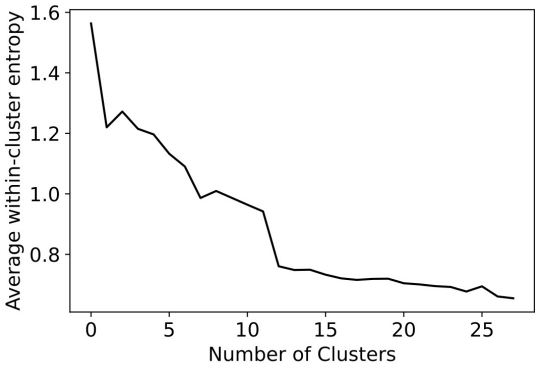


Figure 4.9: Average within-cluster entropies over the number of clusters.

Now, we analyse the performance of the k-means clustering with 15 classes. For that purpose, we plot histograms of event occurrence over time for the month of March 2019 (Figure 4.10a) and the last couple of days of March 2019 (Figure 4.10b) together with the modelled tide (CATS2008 model, Howard et al., 2019), and wind speed and 2m temperature from an ERA5 reanalysis dataset (Muñoz-Sabater, 2019). Furthermore, we plot a t-SNE representation of a subset of the features with colors indicating the cluster (Figure 4.10c). We also plot histograms of event occurrence times of classes 5 and 12 together with a catalogue containing earthquakes with a distance of less than 30 degrees manually picked by analysts at Neumayer station (Figure 4.10d). Finally, we show some examples of waveforms of each cluster (Figure 4.11).
We can clearly distinguish four larger groups of events. The first group, consisting of clusters 2, 3, 6, 7, 9, 10 and 13, shows a dependency on the tide. These icequake clusters coincide with the falling tide (Fig 4.10b, marked in red) and correspond to the large cluster on the right of the scatter plot (Figure 4.10c). The tidally controlled occurrence pattern combined with short duration signals suggests different types of icequakes as likely origin. Tidally-controlled icequakes are well known and have been observed with other methods in detection rates (e.g. Pirli et al., 2018) and seismic noise levels (Fromm et al., 2023).
The second group consists of clusters 0, 1, 4, 8, and 14 and lacks this tidal occurrence pattern. Events occur more irregularly in time, partly in bursts. Clusters 4 and 1 show prominent bursts of activity around March 9 and March 14-15, coinciding with prominent storms during this time, which indicates that they likely contain wind-related waveforms. The waveforms (see Figure 4.11) are not easy to identify as a certain type of seismic event. They correspond to the large cluster on the left of the scatter plot.
Generally, the first half of the month is characterized by more wind and higher temperatures, and there is a drop in temperature around March 16. This correlates with more events in the "icequake" group. This is likely due to a decreased noise level leading to higher event detection rates of the STA/LTA detector, and possibly also due to the lower temperature inducing icequakes as has been found in other studies (e.g. Olinger et al., 2019).
The third group, clusters 5 and 12, are relatively evenly distributed over time indicating that they most likely contain earthquakes. This can also be seen in the waveforms for these particular classes (Figure 4.11). To verify whether these classes contain earthquakes, we plot their histograms again in a direct comparison to an earthquake catalogue from Neumayer station

(Figure 4.10d). We only included earthquakes with a distance of less than 30 degrees, because they are most likely to be seen in the 3-8 Hz filtered seismogram. We notice a rough correlation of the two histograms, which confirms that these two clusters indeed contain real earthquakes. The correlation is weaker in the second half of the month, likely due to a higher number of weak events detected by STA/LTA that are either misclassified or go undetected in the earthquake catalogue. The scatter plot (Figure 4.10c) shows these clusters in the middle between the two larger clusters.

The fourth group only consists of one cluster 11, that contains clearly discernable spikes (see Figure 4.11). They occupy a small, separate portion on the scatter plot.

While we cannot quantify the performance of the algorithm on this larger unlabelled data set, we can show that it is possible to discriminate different clusters with clearly different behaviours and likely different source processes of the seismic events. This indicates that the clustering is useful in automatically creating systematic event catalogues.

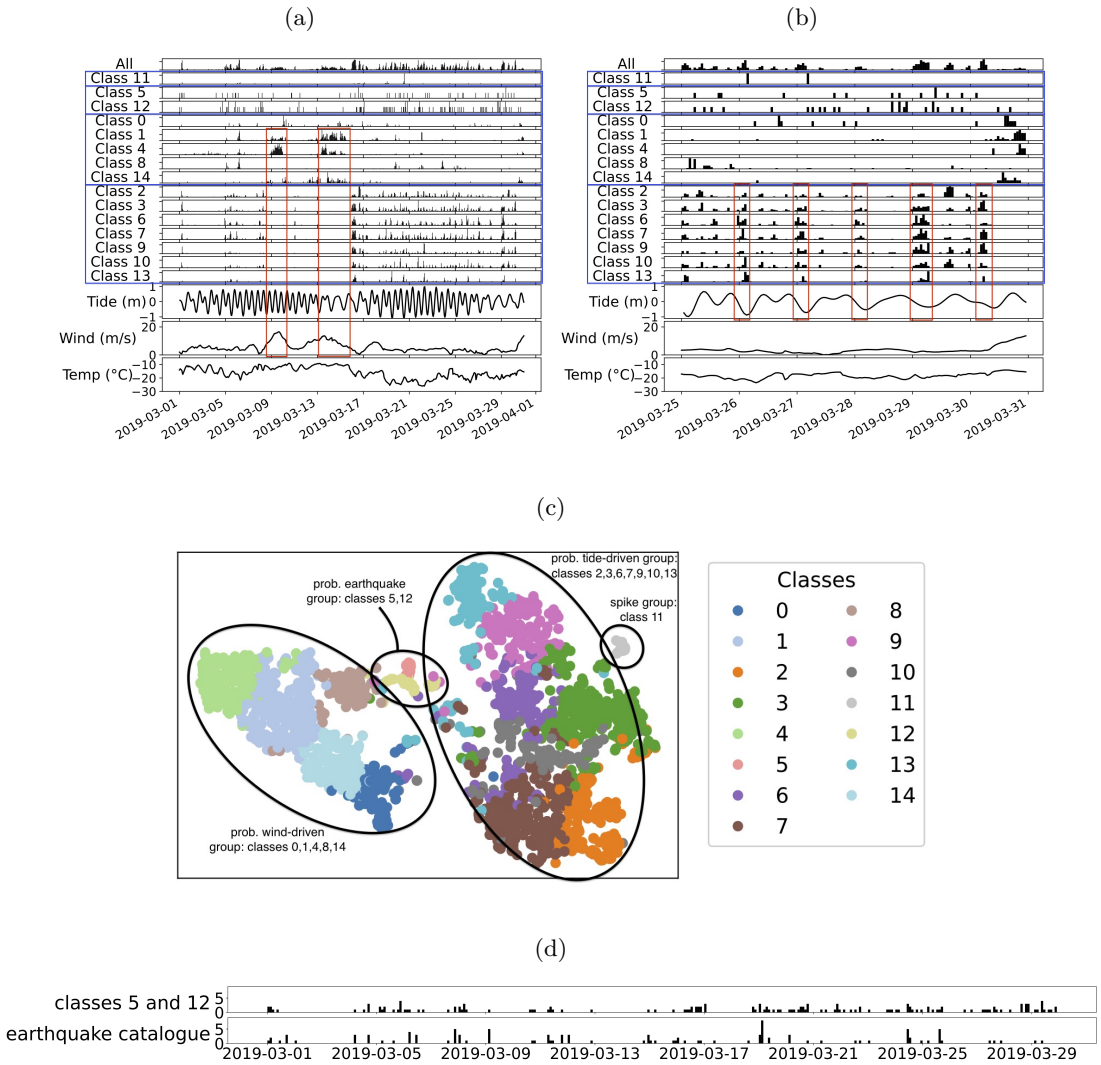


Figure 4.10: Histograms of event times, tide, wind and temperature for (a) the month of March 2019 and (b) the days March 25-30. Note the differences in time distributions of the different classes. Also note the peak in activity of certain classes during falling tide. (c): A t-SNE visualization of (a subset of) the features of NICE-application. Colors indicate the predicted cluster. (d): Histogram of event times of classes 5 and 12 together with event times of Neumayers earthquake catalogue.

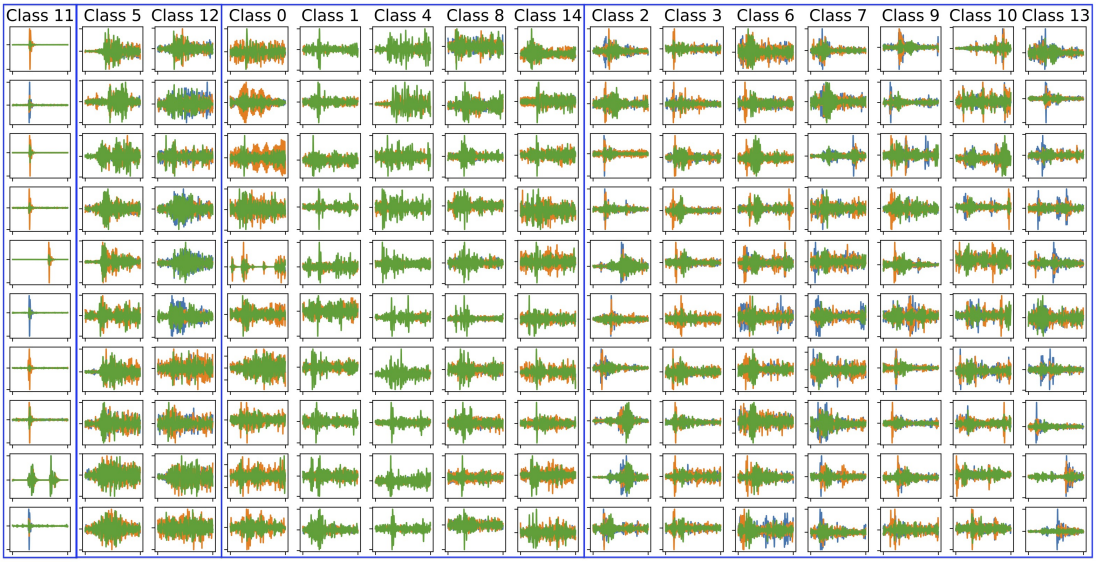


Figure 4.11: Some examples of events in each cluster. The three components are plotted on top of one another in a single plot.

Chapter 5

Conclusions

The main contribution of this thesis is the introduction of contrastive learning to the analysis of seismic datasets, and our analyses show that it can effectively contribute to the arsenal of machine learning methods available to analyse large seismic datasets. Our work was a first attempt of using SimCLR for a seismological dataset. We showed that this approach can help identify different types of seismicity and thus has the potential to speed up the exploration of new seismological datasets. In particular, we differentiated some clusters whose temporal occurrence patterns show some correlation with the tide, wind, and an earthquake catalogue, and thus show clearly different characteristics. Clustering on our labelled dataset also showed that the model separates icequakes from earthquakes well, and that there was more variability within the earthquakes than within the icequakes.

Our work on SimCLR provides a case study on how to apply SimCLR for unsupervised learning to real life applications. This is a valuable contribution also to the machine learning community, as many publications are focused on simulated data or data from academic experiments. In particular, we included a somewhat extensive numerical testing scheme for determining suitable hyperparameters in the data augmentation step. As usual for real life data, the findings are not as clear and obvious as one might hope for. Our final strategy for determining was based on the assumption that larger augmentation parameters, i.e. parameters leading to more pronounced changes in the data, are preferable. Hence, we did choose maximal parameters which still produced good results for the training data. Nevertheless, we want to emphasize that over a comparably large interval for these hyperparameters the differences in the achievable accuracies were minimal or within limits one might expect when dealing with real data. This left some freedom in choosing final values for the hyperparpameters, but it is also a good indication for the robustness of the methods.

Furthermore, our work was the first attempt at clustering the long-term data from the Neumayer geophysical observatory. We started by using the Deep Clustering framework DEC, which we subsequently abandonded in favor of pursuing the contrastive learning approach with SimCLR. Subsequently, there have been several attempts in the seismological machine learning research community of using DEC for seismological clustering (Jenkins et al., 2021, Snover et al., 2021, Ozanich et al., 2021). For example Jenkins et al. (2021), even though they included far more extensive parameter tuning than we did, found that the DEC method does not considerably

improve their clustering compared to a more simple deep clustering approach only using an autoencoder and a Gaussian Mixture Model (GMM), and Ozanich et al. (2021) found no improvement of DEC over the GMM approach.
The DEC method is heavily dependent on the initial autoencoder and clustering, meaning it should work well at improving the clustering of relatively simple datasets where this approach already works well. The SimCLR method on the other hand is in itself independent of the clustering task, and tries to provide a good feature extractor independent of what these features will be used for. In contrast to DEC, the features of SimCLR are learned independently from the clustering, which means that the features may be learned on a different dataset to the dataset on which the clustering is done. In our case, this improved the results since the training could be done on a dataset containing slightly clearer events than the dataset we did the clustering on.
There are many ways to extend the work of this thesis, which were not possible within the time frame of my work.
Firstly, it is neccessary to perform extensive tuning of the DEC method to get the best achievable results out of it. Only then it would be possible to do a proper and fair comparison of the two unsupervised deep learning approaches used in this thesis.
Secondly, there are numerous possible variations or extensions of both methods presented here, since unsupervised deep learning has received plenty of attention in the last couple of years. To only give some examples, one might try combining contrastive learning with deep clustering, and incorporating the clustering already into the neural network training. Furthermore, we may develop more or different augmentation strategies for contrastive learning on seismological data. Another approach would be to use the SimCLR features not only for clustering, but instead as a pre-training step for supervised or semi-supervised learning. This is particularly the case for datasets where some labelled data may already be available.
When applying machine learning to a seismological dataset, we must always consider which kind of approach is best suited for the particular dataset. For example the methods presented here, i.e. unsupervised learning, are primarily a data exploration tool, meaning they are well suited for a first analysis of a new dataset where we do not yet know what kind of events to expect. We applied it here to the lifetime dataset from the Neumayer geophysical observatory, but it can be used for any newly collected seismological dataset. For the Neumayer observatory data, it might be more beneficial to use a (at least semi-) supervised approach, since some of the events we see there are already well known and studied.
Overall, we provide with this thesis a case study on the use of contrastive learning on real datasets for the machine learning community, and a new tool for the exploration of datasets for the seismological community.

Bibliography

Haider Al-Tahan and Yalda Mohsenzadeh. CLAR: Contrastive Learning of Auditory Representations. In *International Conference on Artificial Intelligence and Statistics. PMLR 130*, pages 2530–2538, 2021.

S. Anandakrishnan and R. B. Alley. Tidal forcing of basal seismicity of ice stream C, West Antarctica, observed far inland. *Journal of Geophysical Research: Solid Earth*, 102(B7): 15183–15196, 1997. doi: 10.1029/97jb01073.

Mihael Ankerst, Markus M. Breunig, Hans Peter Kriegel, and Jörg Sander. OPTICS: Ordering Points To Identify the Clustering Structure. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 49–60, 1999. ISBN 9781581130843. doi: 10.1145/304182.304187.

C. Grace Barcheck, Susan Y. Schwartz, and Slawek Tulaczyk. Icequake streaks linked to potential mega-scale glacial lineations beneath an Antarctic ice stream. *Geology*, 48(2):99–102, 2020. doi: 10.1130/G46626.1.

Christopher M Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 1 edition, 2006. ISBN 0387310738.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, pages 1597–1607. PMLR, 2020. URL `http://proceedings.mlr.press/v119/chen20j.html`.

Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, pages 539–546, 2005. ISBN 0769523722. doi: 10.1109/CVPR.2005.202.

Yann Le Cun and Fu Jie Huang. Loss functions for discriminative training of energy-based models. In *AISTATS 2005 - Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, page 34, 2005. ISBN 097273581X.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989. doi: 10.1007/BF02551274.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. doi: 10.1111/j.2517-6161.1977.tb01600.x.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, mar 2016. URL `https://arxiv.org/abs/1603.07285v2`.

Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

Absalom E. Ezugwu, Abiodun M. Ikotun, Olaide O. Oyelade, Laith Abualigah, Jeffery O. Agushaka, Christopher I. Eke, and Andronicus A. Akinyelu. A comprehensive survey of clustering algorithms: State-of-the-art machine learning applications, taxonomy, challenges, and future research prospects. *Engineering Applications of Artificial Intelligence*, 110:104743, 2022. doi: 10.1016/j.engappai.2022.104743.

Tanja Fromm, Alfons Eckstaller, and Jölund Asseng. The AWI Network Antarctica – Alfred-Wegener Institute, Germany. *Summary of the Bulletin of the International Seismological Centre*, pages 22–36 (2309–236X), 2018. doi: 10.5281/zenodo.1156983.

Tanja Fromm, Vera Schlindwein, Veit Helm, and Vera Fofonova. Observing tidal effects on the dynamics of the Ekström Ice Shelf with focus on quarterdiurnal and terdiurnal periods. *Journal of Glaciology*, pages 1–11, mar 2023. doi: 10.1017/jog.2023.4.

Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013. doi: 10.1137/120880811.

Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315—-323, 2011. URL `https://proceedings.mlr.press/v15/glorot11a.html`.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN 978-0-262-03561-3.

Louisa Granzow. *Deep Learning for Picking Seismic Arrival Times at Neumayer Station, Antarctica*. Master's thesis, University of Bremen, 2020.

Jean Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap Your Own Latent A New Approach to Self-Supervised Learning. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 21271–21284, Red Hook, NY, USA, 2020. Curran Associates Inc. doi: 10.48550/arXiv.2006.07733.

Xifeng Guo, Long Gao, Xinwang Liu, and Jianping Yin. Improved deep embedded clustering with local structure preservation. In *Proceedings of the 26th International Joint Conference*

on Artificial Intelligence, pages 1753–1759, 2017. ISBN 9780999241103. doi: 10.24963/ijcai.2017/243.

Xifeng Guo, En Zhu, Xinwang Liu, and Jianping Yin. Deep Embedded Clustering with Data Augmentation. *Acml*, 95:550–565, 2018. URL `http://proceedings.mlr.press/v95/guo18b/guo18b.pdf`.

Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 297–304, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL `https://proceedings.mlr.press/v9/gutmann10a.html`.

Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1735–1742, 2006. ISBN 0769525970. doi: 10.1109/CVPR.2006.100.

Conny Hammer, Matthias Ohrnberger, and Vera Schlindwein. Pattern of cryospheric seismic events observed at Ekström Ice Shelf, Antarctica. *Geophysical Research Letters*, 42(10): 3936–3943, 2015. doi: 10.1002/2015GL064029.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. ISBN 9781467388504. doi: 10.1109/CVPR.2016.90.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum Contrast for Unsupervised Visual Representation Learning. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9726–9735, 2020. doi: 10.1109/CVPR42600.2020.00975.

G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. doi: 10.1126/science.1127647.

Geoffrey Hinton and Tijmen Tieleman. RMSPROP: Divide the Gradient by a Running Average of its Recent Magnitude. *Coursera: Neural Networks for Machine Learning*, 2012.

Shenda Hong, Yanbo Xu, Alind Khare, Satria Priambada, Kevin Maher, Alaa Aljiffry, Jimeng Sun, and Alexey Tumanov. HOLMES: Health OnLine Model Ensemble Serving for Deep Learning Models in Intensive Care Units. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1614–1624, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379984. doi: 10.1145/3394486.3403212.

Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. doi: 10.1016/0893-6080(91)90009-T.

Susan L Howard, Laurence Padman, and Svetlana Erofeeva. CATS2008: Circum-Antarctic Tidal Simulation version 2008. *U.S. Antarctic Program (USAP) Data Center*, 2019. doi: 10.15784/601235.

Peter J. Huber. Robust Estimation of a Location Parameter. *The Annals of Mathematical Statistics*, 53(1):73–101, 1964. doi: 10.1214/aoms/1177703732.

Thomas S. Hudson, Jonathan Smith, Alex M. Brisbourne, and Robert S. White. Automated detection of basal icequakes and discrimination from surface crevassing. *Annals of Glaciology*, 60(79):167–181, 2019. doi: 10.1017/aog.2019.18.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org, 2015.

William F. Jenkins, Peter Gerstoft, Michael J. Bianco, and Peter D. Bromirski. Unsupervised Deep Clustering of Seismic Data: Monitoring the Ross Ice Shelf, Antarctica. *Journal of Geophysical Research: Solid Earth*, 126(9), 2021. doi: 10.1029/2021JB021716.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/cvprw.2009.5206848.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. In *Advances in Neural Information Processing Systems*, pages 18661–18673, 2020.

Diederik P. Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

Louisa Kinzel, Tanja Fromm, Vera Schlindwein, and Peter Maass. Unsupervised Deep Feature Learning for Icequake Discrimination at Neumayer Station, Antarctica. *Seismological Research Letters*, 95(3):1834–1848, 01 2024. doi: 10.1785/0220230078.

Sofia Katerina Kufner, Alex M. Brisbourne, Andrew M. Smith, Thomas S. Hudson, Tavi Murray, Rebecca Schlegel, John M. Kendall, Sridhar Anandakrishnan, and Ian Lee. Not all Icequakes are Created Equal: Basal Icequakes Suggest Diverse Bed Deformation Mechanisms at Rutford Ice Stream, West Antarctica. *Journal of Geophysical Research: Earth Surface*, 126 (6), 2021. doi: 10.1029/2020JF006001.

H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):8397, 1955. doi: 10.1002/nav.3800020109.

Phuc H. Le-Khac, Graham Healy, and Alan F. Smeaton. Contrastive Representation Learning: A Framework and Review. *IEEE Access*, 8:193907–193934, 2020. doi: 10.1109/ACCESS.2020.3031549.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015. doi: 10.1038/nature14539. URL `https://doi.org/10.1038/nature14539`.

Stuart P. Lloyd. Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Amanda C. Lough, C. Grace Barcheck, Douglas A. Wiens, Andrew Nyblade, and Sridhar Anandakrishnan. A previously unreported type of seismic source in the firn layer of the East Antarctic Ice Sheet. *Journal of Geophysical Research F: Earth Surface*, 120(11):2237–2252, 2015. doi: 10.1002/2015JF003658.

S. Mostafa Mousavi, Weiqiang Zhu, William Ellsworth, and Gregory Beroza. Unsupervised Clustering of Seismic Signals Using Deep Convolutional Autoencoders. *IEEE Geoscience and Remote Sensing Letters*, 16(11):1693–1697, 2019. doi: 10.1109/LGRS.2019.2909218.

Jannes Münchmeyer, Jack Woollam, Andreas Rietbrock, Frederik Tilmann, Dietrich Lange, Thomas Bornstein, Tobias Diehl, Carlo Giunchi, Florian Haslinger, Dario Jozinović, Alberto Michelini, Joachim Saul, and Hugo Soto. Which Picker Fits My Data? A Quantitative Evaluation of Deep Learning Based Seismic Pickers. *Journal of Geophysical Research: Solid Earth*, 127(1), 2022. doi: 10.1029/2021JB023499.

Joaquín Muñoz-Sabater. ERA5-Land hourly data from 1950 to present. Copernicus Climate Change Service (C3S) Climate Data Store (CDS), 2019. URL `https://cds.climate.copernicus.eu/doi/10.24381/cds.e2161bac`. (Accessed 2023-06-21).

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012. ISBN 0262018020.

S. D. Olinger, B. P. Lipovsky, D. A. Wiens, R. C. Aster, P. D. Bromirski, Z. Chen, P. Gerstoft, A. A. Nyblade, and R. A. Stephen. Tidal and Thermal Stresses Drive Seismicity Along a Major Ross Ice Shelf Rift. *Geophysical Research Letters*, 46(12):6644–6652, 2019. doi: 10.1029/2019GL082842.

Emma Ozanich, Aaron Thode, Peter Gerstoft, Lauren A. Freeman, and Simon Freeman. Deep embedded clustering of coral reef bioacoustics. *The Journal of the Acoustical Society of America*, 149(4):2587–2601, 2021. doi: 10.1121/10.0004221.

Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825—-2830, 2011.

Myrto Pirli, Sebastian Hainzl, Johannes Schweitzer, Andreas Köhler, and Torsten Dahm. Localised thickening and grounding of an Antarctic ice shelf from tidal triggering and sizing of cryoseismicity. *Earth and Planetary Science Letters*, 503:78–87, 2018. doi: 10.1016/j.epsl.2018.09.024.

Evgeny A. Podolskiy and Fabian Walter. Cryoseismology. *Reviews of Geophysics*, 54(4):708–758, 2016. doi: 10.1002/2016RG000526.

F. Provost, C. Hibert, and J. P. Malet. Automatic classification of endogenous landslide seismicity using the Random Forest supervised classifier. *Geophysical Research Letters*, 44(1): 113–120, 2017. doi: 10.1002/2016GL070709.

Claudia Roeoesli, Agnes Helmstetter, Fabian Walter, and Edi Kissling. Meltwater influences on deep stick-slip icequakes near the base of the Greenland Ice Sheet. *Journal of Geophysical Research: Earth Surface*, 121(2):223–240, 2016. doi: 10.1002/2015JF003601.

Andrew Rosenberg and Julia Hirschberg. V-Measure: A conditional entropy-based external cluster evaluation measure. In *EMNLP-CoNLL 2007 - Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 410–420, 2007.

Zachary E. Ross, Men Andrin Meier, and Egill Hauksson. P Wave Arrival Picking and First-Motion Polarity Determination With Deep Learning. *Journal of Geophysical Research: Solid Earth*, 123(6):5120–5129, 2018. doi: 10.1029/2017JB015251.

Warren S. Sarle, Leonard Kaufman, and Peter J. Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis. *Journal of the American Statistical Association*, 1991. doi: 10.2307/2290430.

Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In *Advances in Neural Information Processing Systems*, pages 41–48, 2004. ISBN 0262201526.

Léonard Seydoux, Randall Balestriero, Piero Poli, Maarten de Hoop, Michel Campillo, and Richard Baraniuk. Clustering earthquake signals and background noises in continuous seismic data with unsupervised deep learning. *Nature Communications*, 11(1):3972, 2020. doi: 10.1038/s41467-020-17841-x.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, New York, NY, USA, 2014. ISBN 9781107298019. doi: 10.1017/CBO9781107298019.

C. Sinadinovski, K. Muirhead, M. Leonard, S. Spiliopoulos, and D. Jepsen. Effective discrimination of icequakes on seismic records from Mawson station. *Physics of the Earth and Planetary Interiors*, 113(1):203–211, 1999. doi: 10.1016/S0031-9201(99)00005-9.

Dylan Snover, Christopher W. Johnson, Michael J. Bianco, and Peter Gerstoft. Deep clustering to identify sources of urban seismic noise in long beach, California. *Seismological Research Letters*, 92(2A):1011–1022, 2021. doi: 10.1785/0220200164.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Alexander Strehl and Joydeep Ghosh. Cluster ensembles - A knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2003. doi: 10.1162/153244303321897735.

Terry T Um, Franz M J Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data Augmentation of Wearable Sensor Data for Parkinson's Disease Monitoring Using Convolutional Neural Networks. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, ICMI 2017, pages 216–220, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5543-8. doi: 10.1145/3136755.3136817.

Laurens Van Der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008.

Joe H. Ward. Hierarchical Grouping to Optimize an Objective Function. *Journal of the American Statistical Association*, 58(301):236–244, 1963. doi: 10.1080/01621459.1963.10500845.

Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in Neural Information Processing Systems*, pages 1473–1480, 2006. ISBN 9780262232531.

Christine Wesche, Rolf Weller, Gert König-Langlo, Tanja Fromm, Alfons Eckstaller, Uwe Nixdorf, and Eberhard Kohlberg. Neumayer III and Kohnen Station in Antarctica operated by the Alfred Wegener Institute. *Journal of large-scale research facilities JLSRF*, 2(A85):1–6, 2016. doi: 10.17815/jlsrf-2-152.

Douglas A. Wiens, Sridhar Anandakrishnan, J. Paul Winberry, and Matt A. King. Simultaneous teleseismic and geodetic observations of the stick-slip motion of an Antarctic ice stream. *Nature*, 453(7196):770–774, 2008. doi: 10.1038/nature06990.

Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, pages 478–487, New York, NY, USA, 2016. ISBN 9781510829008.

Dongkuan Xu and Yingjie Tian. A Comprehensive Survey of Clustering Algorithms. *Annals of Data Science*, 2(2):165–193, 2015. doi: 10.1007/s40745-015-0040-1. URL `https://doi.org/10.1007/s40745-015-0040-1`.