An introduction to distributed training of deep neural networks for segmentation tasks with large seismic datasets

(July 23, 2021)

GEO-2021-0130

Running head: Distributed training of NNs

ABSTRACT

Deep learning applications are drastically progressing in seismic processing and interpretation tasks. However, the majority of approaches subsample data volumes and restrict model sizes to minimize computational requirements. Subsampling the data risks losing vital spatio-temporal information which could aid training while restricting model sizes can impact model performance, or in some extreme cases, renders more complicated tasks such as segmentation impossible. We illustrate how to tackle the two main issues of training of large neural networks: memory limitations and impractically large training times. Typically, training data are preloaded into memory prior to training, a particular challenge for seismic applications where the data format is typically four times larger than that used for standard image processing tasks (float32 versus uint8). Based on an example from microseismic monitoring, we illustrate how over 750 GB of data can be used to train a model by using a data generator approach which only stores in memory the data required for that training batch. Furthermore, efficient training over large models is illustrated through the training of a seven-layer UNet with input data dimensions of 4096×4096 (~7.8M parameters). Through a batch-splitting distributed training approach, training times are reduced by a factor of four. The combination of data generators and distributed training removes
any necessity of data subsampling or restriction of neural network sizes, offering the opportunity of utilisation of larger networks, higher-resolution input data or moving from 2D to 3D problem spaces.
INTRODUCTION

The use of deep learning (DL) has seen a resurgence in its application to geophysical problems over the past decade. Last century’s investigations into the potential benefits of DL methodologies were hampered by technological limitations (Dean et al., 2018). Nowadays, access to reasonably powerful compute is freely available with certain cloud providers even offering free GPU provisions in their experimentation environment, for example CoLab. Alongside this, “tech giants” have open-sourced deep-learning packages “en masse”, such as Google’s TensorFlow package (Abadi et al., 2016) and Facebook’s PyTorch package (Paszke et al., 2019). These advancements have significantly lowered the bar for incorporating DL approaches into research projects and, as such, have contributed to the surge in development of deep learning applications for the geoscience domain. Furthermore, training data and pretrained models have become increasingly more available.

While DL methodologies have seen a resurgence across all fields of seismology, and wider geoscience applications, the use of computer vision procedures in particular has been shown to be incredibly useful for seismic processing and interpretation problems, where the ‘input’ data can be treated as an image. Kaur et al. (2020) illustrated the use of cycle generative adversarial networks for groundroll suppression in land seismic data while Yu et al. (2019) illustrated the potential of convolutional neural networks (CNN) for seismic denoising of random and linear noise signals, as well as multiple suppression. The use of neural networks (NNs) for interpretation of seismic cubes has been extensively tested over the last five years with promising results being offered from many different approaches varying in both preprocessing, NN architecture and postprocessing. For example, Hami-Eddine et al. (2017) investigate the use of a growing NN for an unsupervised clustering
procedure to accelerate seismic interpretation, while Wu et al. (2018) illustrated the use of a CNN for identification of faults within a 2D window from a seismic volume.

DL is not just making waves in the active seismic community, it has also begun making headway in passive seismic applications through the introduction of new, more reliable procedures for event detection. From a single station viewpoint, i.e., where traces are handled independently of one another, recurrent NNs have been shown to be particularly powerful in offering an alternative to the commonly used short-time average, long-time average detection procedure (e.g., Zheng et al., 2018; Birnie and Hansteen, 2020). While from an array point of view, both Stork et al. (2020) and Consolvo and Thornton (2020) have illustrated how CNNs can be used for detecting an events arrival within a certain time-space bounding box.

Despite great advancements being made on tailoring NN architectures for geophysical applications, one large drawback remains: training of large NNs is memory and time expensive. As such, the majority of deep learning applications for seismic datasets require subsampling of the data (Alwon, 2018). A solution to this is to train NNs in a distributed manner, across multiple GPUs, as has been adopted for other big data, deep learning tasks. For example, Zhang et al. (2019) use a distributed training scheme for automatic speech recognition, while Adamski et al. (2018) utilise a similar distributed scheme for teaching a machine to win Atari games via deep reinforcement learning. Furthermore, in the medical domain, distribution of training has allowed for model training where, for legal reasons, the data cannot all be stored in one location (Chang et al., 2018).

Using a passive monitoring scenario, this paper walks through the design, implementation and deployment of a deep learning problem that leverages on the ability to distribute
the NN training, allowing efficient training of a large NN (∼ 7.8M trainable parameters) with a large (> 750 GB) training dataset.

**DATASET**

Similar to the development of many processing, imaging and inversion algorithms, in this study our approach is developed on synthetic data and tested on a field dataset. The field dataset comes from a permanent reservoir monitoring (PRM) system deployed on the seabed at the Grane field in the Norwegian sector of the North Sea. The PRM system consists of 3458 sensors, three-component geophones with a hydrophone, arranged in a pseudo-gridded-style with a sparser “crossline” backbone as illustrated in Figure ?? . The receiver spacing is approximately 50 m along the cables (inline) and 300 m between the cables (crossline). Continuously recording at a 500 Hz sampling rate, almost 2.4 TB of passive seismic data is collected every day.

The system is primarily used for reservoir and overburden monitoring with active seismic surveys. However, it has also been shown to provide invaluable additional information by using it for passive monitoring. For example, drill bit localisation during drilling campaigns (Houbiers et al., 2020; Zarifi et al., 2021) and interferometric velocity modelling (Zhang et al., 2019).

To-date no seismic events have been recorded due to subsurface movement. However, in the summer of 2015 during a drilling campaign, energy waves resulting from a liner collapse were captured in the seismic data. An in-depth analysis of this event was performed by Bussat et al. (2018) using a subset of the receivers. The z-component of this event, hereinafter referred to as the G8-event, is illustrated in Figure ?? and used in this study.
for the benchmarking of the developed ML detection procedure.

**METHODOLOGY**

Defining a clear problem statement is fundamental for the development of any new algorithm, whether ML related or not. For the passive monitoring scenario the problem statement we investigate in this paper is how to develop a **real-time** event detection procedure that uses the full array. Two other key elements in the development of ML approaches include: the training dataset and the model architecture. Below we discuss in detail how the problem is set up, how training data are chosen and how the model architecture is adapted for the use case.

**Solution design**

For the microseismic scenario, events are typically below a SNR of one and therefore a lot of standard processing measures leverage the additional spatio-temporal information that can be captured by using array processing procedures as opposed to trace-by-trace methods. For example, there are a number of different stacking procedures that have been shown to improve detection procedures by increasing the SNR, such as envelope stacking (Gharti et al., 2010) or semblance stacking (Chambers et al., 2010).

Figure ?? illustrates how microseismic event detection can be considered as a computer vision task, whether as a classification, object detection, or image segmentation task. Considering the full array, the identification of the signal within a certain time window can be considered as an image segmentation tasks where each pixel represents a single point in time, \( t \), and space, \( x \). Therefore the task is to determine for each pixel in the image whether
it is part of a seismic event or not, i.e., a binary classification per pixel.

Sliding window approaches have proven very popular in previous image segmentation tasks on post-stack seismic data. They work particularly well due to the uniform spatial sampling in processed seismic sections from active acquisitions meaning that all windows whether 2D or 3D maintain the same distance between samples. However, this is not the case when working with pre-migrated data, as is often the case in passive monitoring. Figure ?? Scenario A schematically illustrates how a rudimentary, spatio-temporal windowing procedure, the most commonly applied in seismic DL applications, could be implemented for raw passive data on a pseudo-gridded-geometry analogous to the Grane geometry. For this approach, one must only consider/optimise the number of stations to include in the window and the time range on which to span. However, due to the irregular spacing between receivers, there is little consistency in the relationship between event arrivals across the different windows.

Scenario B offers a number of more sophisticated alternatives to Scenario A. As illustrated in Figure ??, receiver groups could be selected in multiple ways: by inline grouping, a radius-based approach from central receivers or a nearest-neighbour approach. A number of design decisions must be considered with these approaches: the number of receivers per group; the number of models to be created (e.g., one per group); how to handle over-utilization of receivers where they are grouped into multiple groups; as well as the obvious, which grouping method to use. For the inline and radius-based approaches the number of stations would change between each window therefore requiring different NN models per group. Fixing the number of ‘neighbours’, as illustrated by the neighbour-based approach, would remove the complication of varying input dimensions. However, this would still introduce inconsistencies in the spatial distribution of arrivals, particularly at the edges of
The alternative to splitting the data is to develop an image segmentation procedure that uses all 3458 sensors simultaneously. This removes the complications of determining the optimal receiver groupings (and number of models), however it introduces computational complexities due to the size of each data “observation”. To provide a comparison, most imaging recognition tasks utilise input dimensions of $256 \times 256$ (Deng et al., 2009). Other DL applications on seismic data have ranged from input windows of $24 \times 24$ (Ma et al., 2018) to $100 \times 100$ (Guo et al., 2018) to $128 \times 128 \times 128$ (Wu et al., 2019). The number of traces in our example, 3458, is substantially larger than most input dimensions, as such the remainder of the paper will focus on how to efficiently train NNs with large input dimensions.

**Data creation**

In the seismic space there are three main options for gathering training data: field data collection, laboratory recorded data, or synthetically generated data. A good training dataset must have a large volume of data available, be similar to the data onto which the trained model will be applied and be simple to label. Largely to avoid the tedious annotation procedure typically associated with supervised learning approaches, in this study synthetic datasets were generated for training the model. Historically, synthetic datasets have been heavily used in the development and benchmarking procedures of new algorithms, and the importance of using realistic synthetics to accurately depict how an algorithm will perform on field data cannot be overstated (Birnie et al., 2020). Similarly, to train an ML model that is robust for application to field data, the training data must provide an efficient representation of the variety of waveforms and noise types that exist in such recordings.
however at a reasonable creation speed. In this section we discuss how we have generated a
diverse dataset of realistic synthetic seismic recordings for training and evaluation purposes.

Using traveltimes and the standard convolutional modelling approach, synthetic datasets
are generated using the workflow as illustrated in Figure ???. First, the source location is
randomly selected from a cube in the subsurface centered around the top of the reservoir.
The source parameters: wavelet type (Ricker or Ormsby), frequency content (central fre-
quency of between 20 and 30 Hz), and SNR (of between 0.5 and 2.0) are also randomly
selected. The wavelet is then generated and the wavefield data are created via convolutional
modeling with a scaler accounting for amplitude decay due to geometrical spreading.

Noise is an ever-persistent challenge in seismic field data handling. To make the synthet-
ics representative of field data, synthetic colored noise models are generated using statist-
ics observed from previously collected passive recordings. The frequency spectrum of the
recordings are grouped into 5 Hz bands representing the percent of total energy within each
band. This is used to scale the colored noise model such that it has a similar frequency
content to recorded noise, similar to the approach of Pearce and Barley (1977). The colored
noise model is then scaled spatially to represent the spatial distribution in energy as typ-
ically observed on the array, e.g., higher amplitudes around the vicinity of the production
platform.

As well as forming the base of the synthetic seismic dataset, the wavefield data are used
to generate the matching “label” dataset for training and evaluation purposes. As event
detection is a binary classification, the labels are either zero or one where one indicates
that a wavefield of interest is present. An event’s arrival is classified anywhere where the
wavefield energy is greater than a specified amount depending on the wavelet type and
frequency content.

For simplifying experimentation of the NN architecture, to be discussed below, the length of each synthetic dataset is 4096 time samples which equates to 8.192 s, given the 2 ms sampling interval. Assuming the energy bands for noise spectrum and the array geometry are preloaded, it takes 1.7 s from start to end of the generation procedure of a single seismic section (when computed on a 2.9 GHz, 6-core Intel Core i9 machine with 32 GB RAM).

Model architecture

The U-Net architecture of Ronneberger et al. (2015) has become the workhorse for most image segmentation tasks on seismic data, following on from its successful application in medical imaging. The standard U-Net architecture follows the form of a contracting (left) path and an expansive (right) path as illustrated in Figure ???. The contracting path has the ability to capture context and consists of repeated blocks of: two $3 \times 3$ convolutions each followed by a rectified linear unit (ReLU) and a $2 \times 2$ max pooling operation with a stride of 2 for downsampling. On the other hand, the expansive path enables precise localization and consists of repeat blocks of: two $3 \times 3$ convolutions each followed by a rectified linear unit (ReLU) and a $2 \times 2$ upsampling convolutional layer with a stride of 2. One of the defining features of the U-Net architecture is the ‘copy and crop’ operation that acts across the layers, illustrated by dashed arrows, which connects the contracting and expansive paths.

As noted by Ronneberger et al. (2015) in the original U-Net study: “To allow a seamless tiling of the output segmentation map ..., it is important to select the input tile size such that all $2 \times 2$ max-pooling operations are applied to a layer with an even x- and y-size.”.
the number of sensors in the Grane PRM system, when halfed becomes an odd number, 1729, therefore it is not possible to make a U-Net without altering the input dimensions. An additional 638 null traces were added to the array such that the input dimension became 4096 - a binary number meaning that we can divide by two all the way down to one. These input images are now orders of magnitude larger than Ronneberger et al. (2015)’s study, whose experiment used images of 512×512 pixels.

In the original U-Net study, four layers were used, reducing the data dimensionality down to 32 at the base of the NN. For the Grane example, an additional three layers are required to reduce the data down to the same dimensions. For the convolution steps we begin with four filters at the top layer, multiplying by a factor of two at each reduction step. The incorporation of the additional layers and following the filter methodology, the resulting model has ~ 7.8M number of trainable parameters.

**IMPLEMENTATION**

The large dimensions of the data are not the only “size” complexity arising in this use case due to the data types involved. Typically images are stored with a data type of uint8 while seismic data is stored with a float32 data type. Therefore, a seismic section with the same dimensions as an image is four times larger, affecting memory requirements for NN training. This complexity presents a challenge when loading data into memory for training the NN. For the majority of image segmentation tasks the full training set is loaded into memory prior to training. In this experiment, each labelled seismic section is 108 MB and therefore it is not feasible to load a large number of training and validation samples into memory.

TensorFlow’s dataset functions offer a manageable solution to the memory limitation
challenges encountered due to the datasize. This allowed the storing of only the required data samples per step, therefore removing any necessity to reduce the size of the model or the input data dimensions.

In the data creation section above we argued for the use of synthetic datasets for training purposes. However, there are two approaches to how this can be implemented. Firstly, data can be pre-made, written to file and read in as needed. Alternatively, a data generator can be implemented that creates data on-the-fly. For this specific use case, we calculated that it would take $\sim 4$ hours and 756 GB of storage for the first option, additionally taking 2 s per file to be read in - assuming the data are stored as a TensorFlow Tensor. The second option has the advantage that no additional storage is required however the data would need to be re-generated every epoch. In this case, the generation time is similar to the loading time and as such there is little difference in the processing time of either approach (considering only the reading time for the first approach). Therefore, due to the lowered storage requirements, we choose to implement the second approach of generating the data on-the-fly. The data generator was seeded with the seismic section number such that the same data was generated per epoch and could be replicated at any future point.

**TRAINING**

The model has $\sim 7.8$M trainable parameters with 6000 seismic sections per epoch with an additional 1000 seismic sections generated for validation. Using a single machine with a large GPU *, a single training sample takes $\sim 28$ s. Therefore for one epoch, excluding validation, on a single GPU machine takes $\sim 47$ hours.

---

*6 core, 112GiB, 1xNvidia V100 GPU*
Parallelization of the training regime can drastically decrease the total training time and is a functionality available in both the two biggest machine learning Python libraries: TensorFlow and PyTorch. In this example, we use a batch-splitting (data parallelism) approach implemented by using TensorFlow Estimators with four workers as illustrated in Figure ???. A separate evaluator node is also added to our resource pool such that training is not paused during the validation steps. We follow a synchronous updating procedure requiring each worker to complete its batch and return weight updates to the chief before workers can begin on the next batch of training samples. Using four workers of the same specs as the GPU machine in the serial example, with an additional evaluator node, training time for one epoch is reduced to under 12 hours. Note, some additional compute time is introduced due to both communication and waiting (due to the synchronous training mode).

The training is run on cloud resources and orchestrated using Kubernetes. The training scripts were written and tested locally on small, dummy datasets before being incorporated into a custom Docker Image. A cluster of cloud compute resources were commissioned, in this case five GPU machines with the specs as previously described. A fileshare was mounted to the resources containing the necessary files for the synthetic data creation - geometry and noise energy frequency bands - allowing access to the files as if they were locally stored. Distributed training is initialised via applying a Kubernetes .yaml file to the cluster. The .yaml contains all the necessary information regarding file paths, number of resources to use for training and validation, as well as the additional Python inputs such as number of training samples per epoch, snapshot frequency, and range of synthetic parameters. Once the Kubernetes job has been initiated, the required number of pods are created, in our case one chief, three additional training pods and an evaluator, and the training job begins.

The model is saved at every checkpoint during the training procedure allowing analysis
of the model while training is ongoing. Training ran for approximately 6 days, covering 12 epochs (i.e., 18,000 training steps of 4 samples each), before the model was deemed sufficiently trained via a qualitative analysis of detection performed on newly created (i.e., blind) synthetic recordings. Figure ?? illustrates the progression of the model’s accuracy and loss with respect to the evaluation dataset, as well as the chief’s loss, over the training period.

**Evaluation**

Once sufficiently trained, a number of new synthetic datasets were generated that the model was not exposed to during the training period covering a range of different event locations. Figure ?? illustrates the performance of the trained network on predicting the event arrival for three events of the same magnitude (SNR=0.4), one to the NorthEast of the array, one below the center of the array, and one to the SouthWest. While the moveout patterns are significantly different the network manages to accurately detect the arrivals. Figure ?? zooms in on the recordings from different sections of the receiver array, illustrating how the detection procedure accurately handles the varying amplitude of arrivals across the array as well as the varying local moveouts. In both Figures ?? and ??, there is little-to-no additional noise in the detection arising from the heightened noise levels around the platform site.

A similar analysis is run to evaluate the sensitivity of the trained segmentation model to varying SNRs. Figure ?? illustrates how the detection procedure can handle low SNR events. As expected, decreasing the SNR of arrivals results in increasing noise in the detection procedure. Down to an SNR of 0.2 the arrival shape is clearly visible within the prediction section however at an SNR of 0.1 the event arrivals are no longer easily identifiable.
To ensure the trained model is applicable to field data, it is applied to the previously
described G8 event. Figure ?? displays the 8 s seismic recording with the event alongside
the UNet predictions. The blue box highlights the arrival on a particularly noisy receiver
grouping while the red box indicates the arrival on a quieter group of receivers at the edge
of the array. The event is clearly detected across the majority of receivers without detecting
the platform noise that begins halfway through the recording.

DISCUSSION

The aim of this study was to investigate possible methodologies for the training and ap-
lication of large deep NNs on seismic datasets without the requirement of subsampling
or windowing. The ability to handle larger input data dimensions as well as train larger
models offers the opportunity for capturing additional spatio-temporal information from
the seismic data - a well documented approach for enhancing SNR. The solution design
section of this paper, in particular Figure ??, highlighted the complications in developing
a generic model to be applied on either receiver lines or by windowing the array - for this
specific use case. As such, the simplest path was to process the full array in one go and
leverage technological advances to allow the training of such a large model.

There also exist a number of other use cases which naturally permit windowing however
that may benefit from using larger windows. Fault detection is one such task that is often
reduced to a 2D problem despite the ‘original’ 3D subsurface data volume. For example, Guo
et al. (2018) extract 2D slices from a 3D seismic cube, explicitly stating: “The dimension
reduction from 3D to 2D is to reduce the time to train the CNN.” Similarly, Ma et al.
(2018) provide 24×24 images with an inline, crossline and time section as input channels
to a 2D NN, rendering the problem pseudo-3D. However, they stop short of using a full 3D
input. Distributing the training allows the possibility of using larger input data dimensions (increasing window sizes or adding an additional dimension) therefore, either capturing a larger spatio-temporal area or offering the opportunity to use higher resolution data.

In this paper, due to the ability to hold a single batch and the model in the memory of a GPU card, we illustrated a synchronous, batch-splitting distributed training scheme - arguably the simplest of the distribution strategy. However, many alternative distribution procedures are available which may further speed up training or to handle situations where the model is too large to be held in memory. For example, Zhu et al. (2020) use a model parallelism approach for an image segmentation task where increasing the model size was shown to improve the model’s accuracy. Ben-Nun and Hoefler (2019) provide a good overview of the different distribution schemes possible. Additionally, there are a number of other strategies that could be considered for reducing computational requirements or further speeding up training. For example, reducing the precision of the data and model parameters, incorporating a data compression procedure (e.g. Jaderberg et al., 2014), or using other specialised hardware, such as tensor processing units (Jouppi et al., 2017) and field programmable gate arrays (Chen et al., 2014). These will all influence the model’s performance, the extent to which is subject to future work.

A trade-off can occur between input data dimensions and model size where, as opposed to subsampling data, a smaller, simpler model is used. For example, in the microseismic event detection use case both Stork et al. (2020) and Consolvo and Thornton (2020) train CNNs to detect a time-space box in which an arrival is detected. The smaller computational requirement allows for a faster training procedure; however, less information can be derived from the models’ predictions. For object detection the returned information is that of a bounding box with the same “arrival time” for all receivers as opposed to segmentation
procedures which detect arrival times per trace. Wu et al. (2019) provide another example of where a smaller network has been utilised. They used a simplified UNet with a reduced number of layers for a 3D fault detection procedure which allowed for significant “savings in GPU memory and computational time”. The procedure for efficient training detailed in this paper provides the opportunity to increase model dimensions while still keeping a reasonable training time.

It should be noted here that not all deep learning applications on seismic data require subsampling. For example, should the same segmentation procedure developed in this paper be adapted for a different, smaller permanent array, such as the 50 receiver array at Aquistore (Stork et al., 2018), the input dimensions would be smaller than those used in the original UNet implementation rendering any discussion on subsampling unnecessary. However, these use cases are becoming rarer, particularly with the adoption of densely sampled fiber optic cables for permanent monitoring.

The success of a model is highly dependent on its training data, and the use of synthetic datasets for training has become common-place in seismic deep learning procedures (e.g. Huang et al., 2017; Pham et al., 2019; Wu et al., 2019; Cunha et al., 2020). In traditional synthetic data usage for developing and benchmarking algorithms it has been shown that the more realistic the synthetic data the better for understanding uncertainties and identifying pitfalls (Birnie et al., 2020). However, there is a trade-off between similarity to field data and computational cost which is particularly applicable when developing the large volume of datasets required for training deep learning models. In this use case, the GPU resources were not leveraged for the synthetic data generation and we found that generating the waveform data via a wave propagation procedure would be too computationally expensive for what we classified as a reasonable generation time - sub 2.0 s. As such, we
used a simple convolutional modeling procedure incorporating geometric spreading while assuming a homogeneous velocity model for the traveltime computations. Similarly, for the incorporation of noise in the dataset, the generation of realistic noise models (i.e., non-stationary, non-Gaussian, non-white noise), such as via a covariance-based approach (Birnie et al., 2016), was deemed too timely. Therefore, an approach similar to that of Pearce and Barley (1977) was used which generates a stationary noise model that accurately replicates the frequency content of recorded noise. As of yet, an analysis has not been published to show the trade-off between the complexity/reality of synthetics and the performance of the trained model. In this use case the resulting network produces acceptable predictions on this field dataset, but more testing is required to fully assess the model’s performance on a wider variety of field data with varying noise and event properties.

Despite many advancements in detection algorithms over the years, Skoumal et al. (2016) highlighted how computational cost is a big barrier preventing the majority of these algorithms making it into a production toolbox. One of the key criteria of such a detection algorithm is its real-time applicability. While the training took 6 days using four GPU machines, detection can be performed in under 3.0 s on a 2.9GHz, six-core Intel Core i9 machine with 32GB RAM for an 8.0 s recording segment. Therefore, once trained, the model can be used for real-time monitoring applications without any requirement of large computational resources or parallelization across multiple machines. Following the successful implementation of the distributed procedure, future work will focus on improving the model’s performance, in particular focusing on detecting the full hyperbolic arrival. This investigation will consider the diversity of the training dataset; the optimum model architecture (e.g., number of filters, layers, etc.); the optimum loss function; as well as any benefits of varying the training procedure, such as drop out and learning rate.
CONCLUSION

The majority of deep learning applications for seismic data involve the subsampling or windowing of the dataset. In this paper, we illustrate how through the distribution of training, larger networks can be efficiently trained, removing the need for subsampling and/or windowing. Illustrated on a microseismic monitoring use case, the paper walks through the stages of the deep learning project, from synthetic training data creation to adapting a standard model architecture to distributed model training and finally to model evaluation using both synthetic and field datasets. While illustrated on a scenario where data windowing is non-trivial, not windowing data, or using larger windows than previously possible, has great potential for other segmentation tasks such as fault and horizon detection.

ACKNOWLEDGMENTS

The authors would like to thank the Grane license partners Equinor Energy AS, Petoro AS, Vår Energi AS, and ConocoPhillips Skandinavia AS for allowing to present this work. The views and opinions expressed in this abstract are those of the Operator and are not necessarily shared by the license partners. The authors would also like to thank Ahmed Khamassi and Florian Schuchert for their invaluable support on the data science elements of this project, as well as Marianne Houbiers for her insightful discussions on the application of DL for passive monitoring.
REFERENCES


Bussat, S., M. Houbiers, and Z. Zarifi, 2018, Real-time microseismic overburden surveillance at the grane prm field offshore norway: First Break, 36, 63–70.


medical imaging: Journal of the American Medical Informatics Association, 25, 945–954.


Pham, N., S. Fomel, and D. Dunlap, 2019, Automatic channel detection using deep learning: Interpretation, 7, SE43–SE50.

Ronneberger, O., P. Fischer, and T. Brox, 2015, U-net: Convolutional networks for biomed-


Zhang, W., X. Cui, U. Finkler, G. Saon, A. Kayi, A. Buyuktosunoglu, B. Kingsbury, D.


FIGURE LIST

- Figure 1: Array information for the permanent reservoir monitoring system deployed over the Grane field in the Norwegian North Sea. The array is separated into “inline” sections as represented by the colorscale. (a) the array geometry overlaid on the field’s polygon with the black triangle indicating the location of the platform. (b) the number of sensors per line, while (c) the distribution in distances between neighbouring sensors per “inline”.

- Figure 2: Bandpassed data of the G8 event recorded over the full PRM array (a). The blue box in the center corresponds to the zoomed in data segment shown in (b) highlighting the event arrival at the same time as an onset of platform noise. The red box corresponds to the zoomed in data segment shown in (c) from a quieter section of the array.

- Figure 3: schematic illustrating how microseismic event detection can be considered as a computer vision task, either as full image classification, object detection, or image segmentation.

- Figure 4: Schematic illustrating possible approaches to windowing of the seismic data prior to developing DL models.

- Figure 5: Workflow of the generation and labelling of synthetic data.

- Figure 6: Seven layer UNet architecture.

- Figure 7: Comparison between a single process for training vs a distributed process using a data parallelism strategy with four workers. The evaluator node is not illustrated.
Figure 8: Progression of (a) the model accuracy and (b) the loss during training.

Figure 9: UNet detection’s on synthetic seismic events with source origins in different subsurface locations: NorthEast of the array, below the center of the array and to the SouthWest of the array, as illustrated by red crosses in the array map. The top panel shows the synthetic data, the middle panel shows the labels corresponding to the synthetic and the bottom panel shows the UNet’s detection.

Figure 10: Magnified results from the event below the center of the array as illustrated in ???. The first row comes from receiver lines in the West of the array, the middle row includes the two inlines closest to the platform in the center of the array, and the bottom row includes receivers in the furthest East lines.

Figure 11: SNR investigation on the performance of the trained UNet with the event always originating from the same subsurface location.

UNet event detection on the Grane G8 liner collapse event. The blue box in the center corresponds to the zoomed in data segment and detections shown in the bottom left column highlighting the event arrival at the same time as an onset of platform noise. The red box corresponds to the zoomed in data segment and detections shown in the bottom right column from a quieter section of the array.
Figure 1: Array information for the permanent reservoir monitoring system deployed over the Grane field in the Norwegian North Sea. The array is separated into "inline" sections as represented by the colorscale. (a) the array geometry overlaid on the field's polygon with the black triangle indicating the location of the platform. (b) the number of sensors per line, while (c) the distribution in distances between neighbouring sensors per "inline".

439x322mm (59 x 59 DPI)
Figure 2: Bandpassed data of the G8 event recorded over the full PRM array (a). The blue box in the center corresponds to the zoomed in data segment shown in (b) highlighting the event arrival at the same time as an onset of platform noise. The red box corresponds to the zoomed in data segment shown in (c) from a quieter section of the array.

254x177mm (300 x 300 DPI)
Figure 3: Schematic illustrating how microseismic event detection can be considered as a computer vision task, either as full image classification, object detection, or image segmentation.

300x98mm (150 x 150 DPI)
Figure 4: Schematic illustrating possible approaches to windowing of the seismic data prior to developing DL models.

226x134mm (144 x 144 DPI)
Figure 5: Workflow of the generation and labelling of synthetic data.
Figure 6: Seven layer UNet architecture.

255x85mm (150 x 150 DPI)
Figure 7: Comparison between a single process for training vs a distributed process using a data parallelism strategy with four workers. The evaluator node is not illustrated.
Figure 8: Progression of (a) the model accuracy and (b) the loss during training.
Figure 9: UNet detection's on synthetic seismic events with source origins in different subsurface locations: NorthEast of the array, below the center of the array and to the SouthWest of the array, as illustrated by red crosses in the array map. The top panel shows the synthetic data, the middle panel shows the labels corresponding to the synthetic and the bottom panel shows the UNet's detection.

362x437mm (59 x 59 DPI)
Figure 10: Magnified results from the event below the center of the array as illustrated in 9. The first row comes from receiver lines in the West of the array, the middle row includes the two inlines closest to the platform in the center of the array, and the bottom row includes receivers in the furthest East lines.

171x171mm (150 x 150 DPI)
Figure 11: SNR investigation on the performance of the trained UNet with the event always originating from the same subsurface location.

365x441mm (59 x 59 DPI)
Figure 12: UNet event detection on the Grane G8 liner collapse event. The blue box in the center corresponds to the zoomed in data segment and detections shown in the bottom left column highlighting the event arrival at the same time as an onset of platform noise. The red box corresponds to the zoomed in data segment and detections shown in the bottom right column from a quieter section of the array.

381x381mm (300 x 300 DPI)
DATA AND MATERIALS AVAILABILITY

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