Bottom Sea Estimation with Deep Learning Methods

Jean-Michel Amath Sarr[*]
IRD
jm.amath@icloud.com

Timothée Brochier[†]
IRD
timothee.brochier@gmail.com

June 15, 2018

Abstract

Terabytes of acoustic multispectral data are routinely collected by oceanographic vessel on fisheries resources campaign assessment in West African waters. The vessel sends out pulses of various frequency's acoustic waves in the water, those waves are reflected back to the source when they meet diverse organisms (fish, plankton, etc) or more generally solid objects. We call echogram (echo more informally) the corresponding signal. Since there is a large variety of elements interfering with the acoustic signal it is hard (we rely on the work of experts) to interpret and identify the various organisms that are found by this procedure. Nowadays bottom sea recognition is still partially done by experts in a semi automated approach. The overall goal of this work is to investigate the effectiveness of deep learning methods to automate the task of bottom sea estimation.

Contents

1 Introduction **2**

2 Material and Methods **3**
1 Data . . . 3
2 Supervised Learning . . . 4
2.1 Regression . . . 4
3 Model Selection procedure . . . 7
3.1 Experiment Design . . . 7
4 Models . . . 7
5 Linear Regression . . . 7
6 Multilayer Neural Networks . . . 7

[*]A thank you or further information
[†]Corresponding author

6.1 Overview 7
6.2 Building Blocks 8
7 Convolutional Neural Networks 9
7.1 Overview 9
7.2 Building Blocks 10
8 Recurrent Neural Networks 13
8.1 Overview 13
8.2 Building Blocks 14

3 Results **16**

4 Discussion and further work **16**

1. Introduction

With the increased availability of environmental sensors, their increasing precision and memory storage capacity, the volume and complexity of data collected from environmental observations has risen. In the context of under water marine observations by the mean of active acoustics method, multi frequency echo sounder data have often been used to make inference for fish and plankton classification. However, the first operation needed before any inference, is to identify bottom sea position.

Early echo sounders typically have broad beams (30-60 degrees) which make depth measurements inaccurate when surveying irregular seafloor. Indeed in this situation a given ping will ensonify a large area of the bottom, and if it?s irregular, the depth recorded could be anywhere within the ensonified area. [1]. Another limitation of single beam echo sounder is that only one depth value per ping is returned. Thus, the lack of measurements density make it a serious drawback to map complex seafloor surfaces accurately. Hence, this result in an inaccurate picture of the seafloor topography.

However, for fish abundance estimations, an accurate knowledge of the bottom depth is needed, as the fish abundance is usually estimated by integration of the acoustic signal from bottom to surface. Because acoustic signal is very strong near the bottom, a little error in bottom estimation can lead to large error in fish abundance estimates.

For these reasons, the task of bottom sea recognition with single beam echo sounder is still done in a semi automatic way, involving a lot of human decision making in a tedious and time consuming process. These limitations has led to the development of multi beam depth sounders, which became available in the late 1970?s, but which are still not used for official fish stock estimations due to increased complexity of data analysis. Thus, while state of the art methods [2] currently use multi beam sonar data for seafloor topography, in this work we are interested in leveraging the large body of single beam data that were, and continue to be, recorded. Indeed today there is huge data bases full of those data, that we can?t use because the human intervention would be tremendous.

Nowadays with the increase of computer power and the rise of big data, deep learning methodologies have showed an edge in numerous tasks from computer vision to natural language processing [6]. In this paper we investigate how well neural networks perform for bottom sea recognition. Furthermore, we evaluate if we could learn models that would extract enough human insight from labeled data, to make the procedure fully automated on unseen data.

The outline of this paper is structured as follow: first we review the general methodology used in machine learning, specifically for supervised learning tasks. Secondly we present our data and some widely used models Linear Regression, Multi Layers Neural Networks, Convolutional Neural Networks and Recurrent Neural Networks. Finally, we present the result and discussed how these networks could be used to predict the bottom sea on further campaigns.

2. Material and Methods

1. Data

Data has been provided by IRD (Institut de Recherche et Développement), it consists of two datasets corresponding to two campaigns that took places in 2011 and 2015 in the North-West African coastal waters.. We can summarise the relevant information as follows: After having discussed with the experts' team it appeared that the following variables were relevant to learn from data: Latitude, Longitude, Echogram, and CleanBottom. For our purpose, following the expert advice we only select the lowest frequency (18 kHz) to draw the bottom line (Mainly because it goes deeper). The sonar is fixed on the ghull of the vessel and send regularly acoustic pulse toward the bottom, at four distinct frequencies (18 kHz, 38 kHz, 120 kHz, 200 kHz) Each pulse is called a ping. Each variable has a value at every ping.

- Depth is a vertical grid with a regular spacing and each cell correspond to a value of depth in meters
- Echogram is associated with Depth, in fact for every ping and every depth there is an echogram value.
- CleanBottom are the values of the bottom set by the expert.
- Position : latitude and longitude.

In summary, data can be viewed as a snapshot of the water where at each time (ping) we have diverse values. Hence we subset our dataset using those variables. In figure 1 we illustrate the echogram and the bottom sea as well for two sample from the 2011 and 2015 campaign that we will be using for model selection.

2. Supervised Learning

Supervised learning is a part of machine learning dealing with problems that we can frame as follows:

We have a treatment that input some data X and output some data Y, and we want to learn a function f able to approximate the treatment. There is two popular type of supervised learning problems :

- Regression problems
- Classification problems

The only difference lie in the types of the output Y, in fact if Y can be represented as categorical data (finite number of classes), we better use a classification setting. Otherwise if Y can vary infinitely, a regression procedure is recommended. Often the function to learn has learnable parameters that, combien with the data make a prediction. To evaluate the quality of a prediction a loss function is used, then, to find the best f, an optimization procedure is run to minimize the loss.

2.1 Regression

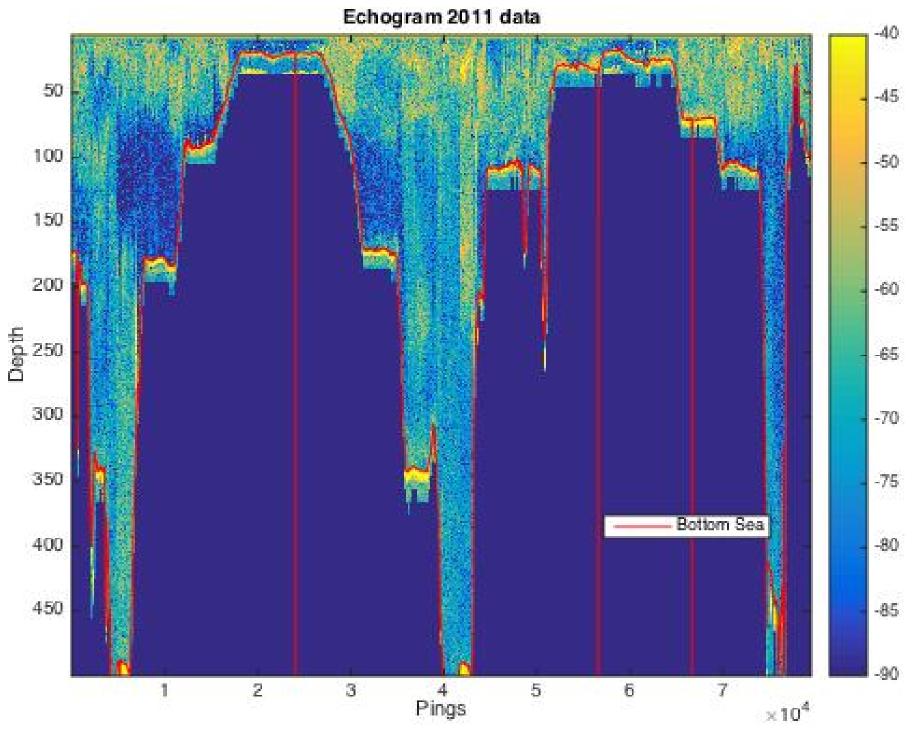
Before starting any modeling we needed to know the kind of problem we may refer on. Clearly, we can frame it as a supervised learning problem, where the variable to predict is CleanBottom that we rename Y, doing so, our explanatory variable will be X a DataFrame. In summary

- Y = CleanBottom.
- X = Latitude, Longitude, Echogram.

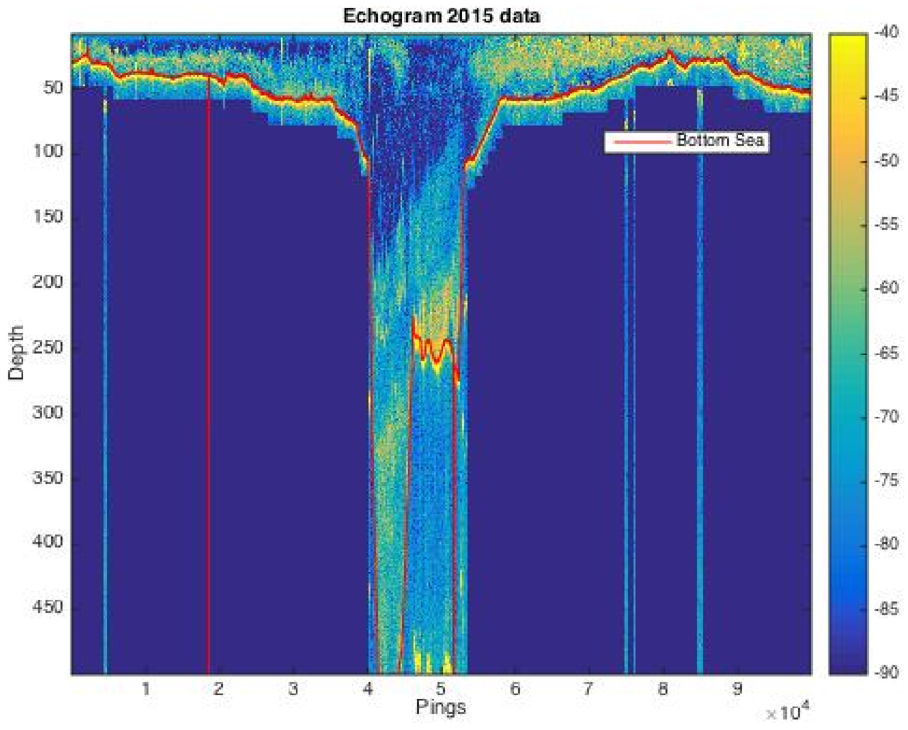
In a first time, we opted for a regression problem, in other words we want to evaluate the value of the bottom Y given X. Our error will be measured in meters (we used the mean absolute error as cost function) Hence we tried several models:

- Linear Regression (LR)
- Shallow Neural Network (SNN)
- Deep Neural Network (DNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)

The main difference between the SNN and the DNN is the number of hidden layers. In the following paragraphs we will argue the relevancy of each of these models in our setting, and give a concise description of their key components.



(a) *As we see there is bad pings, resulted from measurements error.*



(b) *We observe that sometimes the echogram have value even when the bottom sea is rather shallow.*

Figure 1: *Echograms are represented, the deep blue color correspond to Nan values. Since the multi frequency data are collected during the whole campaign the process is memory expensive, thus only relevant parts of the echogram were saved.*

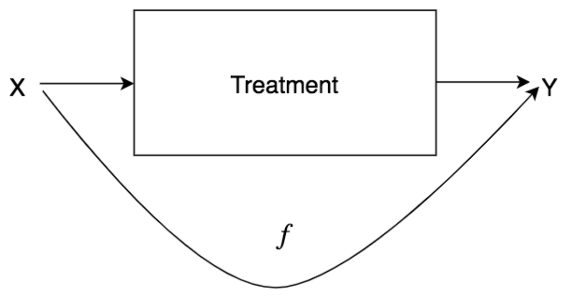


Figure 2: *The treatment is approximated by a function f.*

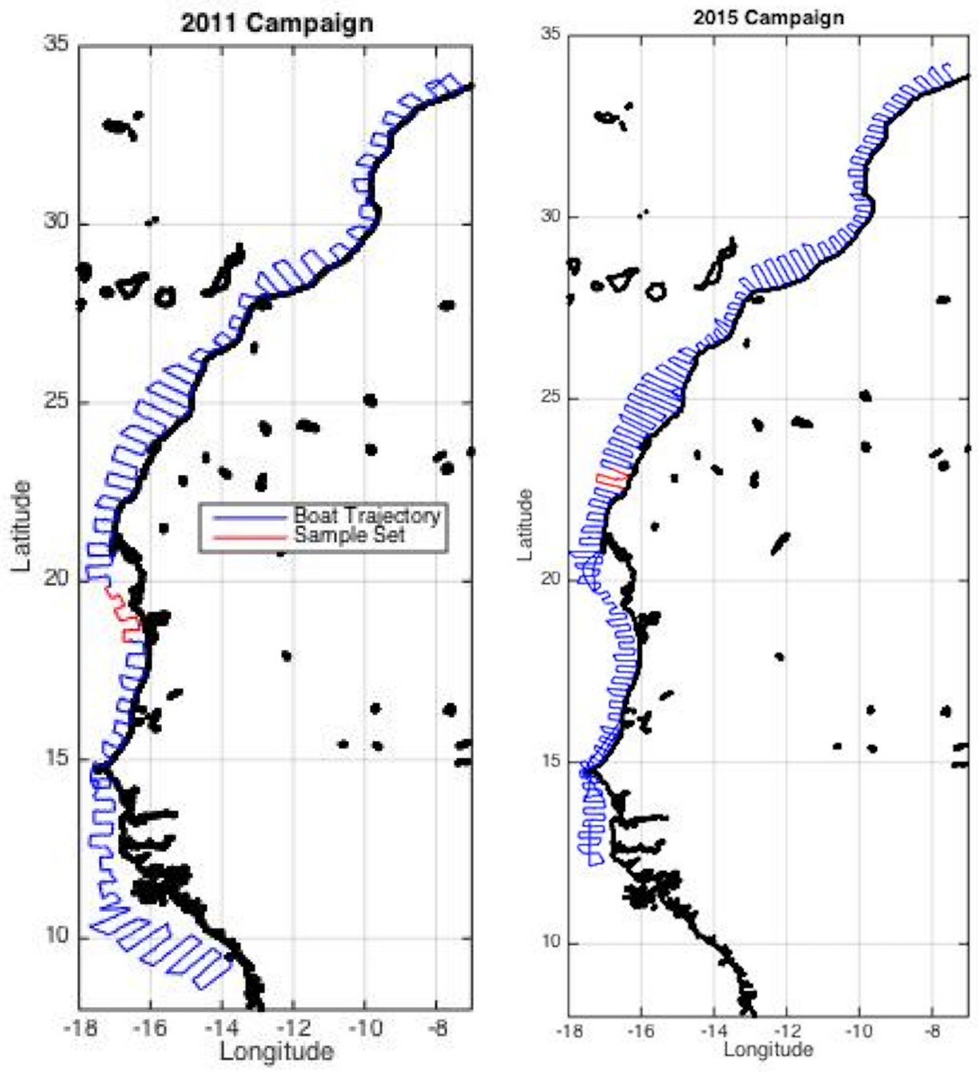


Figure 3: *Origin of both sample, 2011 and 2015. Note that the 2011 campaign expand further in the South while the 2015 campaign has a finer granularity. In red we can observe the sample that we use for model selection. As we can notice, the two dataset were sample from different regions.*

3. Model Selection procedure

Here we follow the standard training-development-test scheme for model evaluation [3]. However since our problem is specifically to learn a function

$$f(Latitude, Longitude, Echogram) \mapsto CleanBottom$$

able to make prediction on new dataset, we need our dev/test set to come from the same dataset. In fact we conduct two experiment with the same design.

3.1 Experiment Design

Both experiment are the same reversing the data used, let illustrate it.
data selection

- Sample training set (100 K datapoints) from 2011 dataset
- Sample dev/test set (100 K datapoints split into two subsets of equal dimension) from 2015 dataset

model selection

- Train each of the five models on the training set
- Tune hyperparameters on the dev set and select the best model
- Evaluate generalization error on the test set

Once we finish the experiment we will get a best model and it's error. Then we make the second experiment to obtain another best model and it's error. At this point we have potentially two models that stand out. To select the best one, we evaluate relative performance, see result figure 15

4. Models

5. Linear Regression

The Linear Regression Model assumes that there is a linear relationship between the explanatory variable X and the output Y. Namely

$$Z = W \cdot X + b \tag{1}$$

where Z is the predicted output, W a vector of parameters of the same shape than X and b a scalar. The parameters to learn are $\{W, b\}$.

6. Multilayer Neural Networks

6.1 Overview

The hypothesis behind the use of neural networks (NN) is that the treatment to model can be described in terms of simpler treatments. Each neuron representing a specific treatment of the data, see figure 4. Hence, a neural network is a composition of such simpler treatments see figure 5. The hyperparameter of neural network are:

- numbers of layers
- numbers of neurons per layer
- activation function for each layer

By hyperparameters we mean that those parameters are determined before hand, by the modeler, as opposed as the learnable parameters which depends on the data. In general the more hidden layers there is in a neural network the deeper we say it is. Thus, the term deep learning make no surprise. The mathematical back up of the efficiency of these networks are found in [4] where it is proven that neural networks can arbitrarily approximate any continuous function on a compact. In other words, any treatment that can be described as a continuous mapping of the data can be learn by a neural network. However, the proof is not constructive, meaning that we don't know in advance how many neuron will be necessary to find a good approximation. Consequently different architectures have emerged taking advantage of the data structure, with the general remark that deeper NN have a better representation capacity.

6.2 Building Blocks

As discussed in the section supervised learning, the parameters are learnt in an iterative fashion, in general there is two major step to train NN, feed forward propagation and backpropagation. Practically, backpropagation is tedious to derive, so modern libraries such as TensorFlow, Keras have automated it, then only forward propagation is needed to train a NN. In this context we will describe the general forward propagation formulas for a multilayer NN. Given a NN with the following hyperparameters: [1]

- L: number of layers
- $n^{[l]}$: numbers of neuron in the layer l
- $g^{[l]}$: activation function of the layer l

We consider $X = a^{[0]}$, the parameters to learn for layer l are : $W^l, b^{[l]}$. Hence we have the general formulas.

$$Z^{[l]} = W^{[l]T} \cdot a^{[l-1]} + b^{[l]} \tag{2}$$

$$a^{[l]}] = g^{[l]}(Z^{[l]}) \tag{3}$$

In the next paragraphs, CNN, and RNN, are described.

[1]We follow the notation of [5]

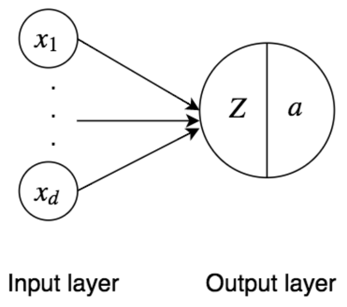


Figure 4: *Single neuron where Z is computed as in the equation (1), $a = g(Z)$ is the final output, g is the activation function, and $\{x_1, \cdots, x_d\}$ is an example .*

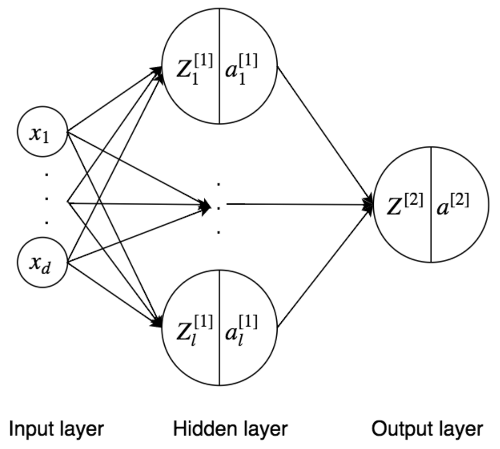


Figure 5: *One hidden layer neural network*

7. Convolutional Neural Networks

7.1 Overview

Traditional pattern recognition algorithms to treat images or speech were built using a feature extractor, then a classifier. Is it possible to use NN to classify speech and images ? Well, we have seen that theoretically it is possible to learn very complex function with NN, but traditional architectures encounter some problems to work with data such as images, speech or time series.

- Images usually haves hundred of variables (one for each pixel), and fully connected architectures will have indeed lots of parameters to learn which is costly (in terms of memory and computation)
- NN simply ignore the topology of the data, for instance the organization of the pixels in an image is not taken into account. They can be presented in any order without influencing the output. Whereas in the case of images, variables are highly correlated.
- NN also lack build-in invariance with respect to local distortions. Meaning that a small change in the input could affect the output.

These are the main considerations behind the emergence of CNN. Indeed, CNN are based one three architectural ideas: sparsity of connections, parameter sharing and equivariant representation ([7], [6]). We will illustrate those ideas in the following section, describing the building blocks of a CNN.

7.2 Building Blocks

Convolution operation
The first operation to consider is the convolution operation that is described in the following figure

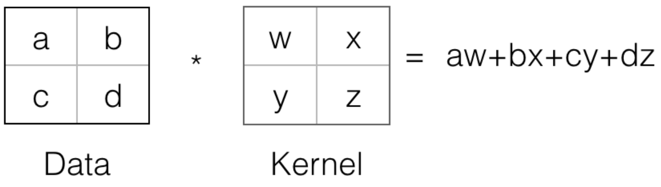


Figure 6: *Convolution operation; each element of the data is multiplied with the corresponding element of the kernel the results are summed up.*

Note that this operation is not the real mathematical convolution but is instead referred as the cross-correlation operation, which is a flipped version of the convolution. In fact the cross convolution miss the associativity property, that can be useful to prove theoretical result, but is not useful in the context of machine learning where we seek computational efficiency. Convolution has three hyperparameters:

- f: size of the kernel
- s: stride
- p: padding

For instance in the figure 6 the kernel is of size 2×2 and the stride of size 0 with no padding. In fact the figure 7 describe how the convolution operation work with a stride $s = 1$. A problem with this procedure is that the corner are taken into account just once, also the result of a convolution has a shrinked shape. The role of the padding parameter is to prevent this by adding a strip of zeros all around the image. There are two types of padding essentially: valid and same.

- valid: does not affect the data
- same: pad the data so that the output is of the same shape as the entry

The equation to figure the shape of the result a convolution operation is as follow:

$$\left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n+2p-f}{s} + 1 \right\rfloor \tag{4}$$

Where n is the height and with of the input, p is the larger of the padding, f the hight and width of the kernel, and s the stride. In short a strided convolition looks as in the figure 7.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} * \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 12 & 14 \end{bmatrix}$$

Figure 7: *Strided convolution; the result of a strided convolution is obtained by streaming the kernel through the data.*

As we discussed earlier traditional pattern recognition algorithm in computer vision include a feature detector and a classifier. In the following example [figure 8] we will show that the convolution operation can serve as a feature detector.

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}$$

Figure 8: *Convolution on an image, pixels are numbers. From left to right:* 6×6 *image,* 3×3 *kernel, and* 4×4 *image*

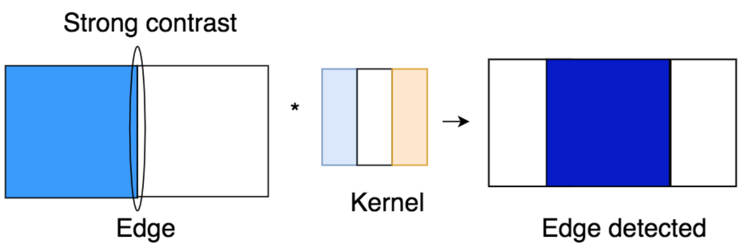


Figure 9: *Convolution as a feature detector to illustrate the figure 8*

As said earlier images, speech have strong local structure, thus taking the topology of the data into account may result in some improvements. One of the advantage of the convolution operation is that it exploits local correlation and combine local feature to recognize spatial (image) or temporal object (speech). In the example above the convolution highlighted the contrast in an image. Thus the topology of the data is taken into account.
One might ask how have we found such a kernel for this task of detecting edges. We would answer that researchers have found it by trial and error. But in the Deep Learning era, kernels are parameters and can be learned in a NN, that's

precisely what a CNN does, hence the remaining question is how generally this operation is used in a NN, and what is a CNN whatsoever ? Well, to answer this question we need to present the building block of a CNN: a convolution layer.

Convolution layer

The main difference between a convolutional layer and a standard layer is the linear operation as you may see comparing the equation (2) and (5) :

$$Z^{[1]} = W^{[1]} * a^{[0]} + b^{[1]} \tag{5}$$

$$a^{[1]} = g^{[1]}(Z^{[1]}) \tag{6}$$

To get a visual understanding of what happens here, refer to figure 10

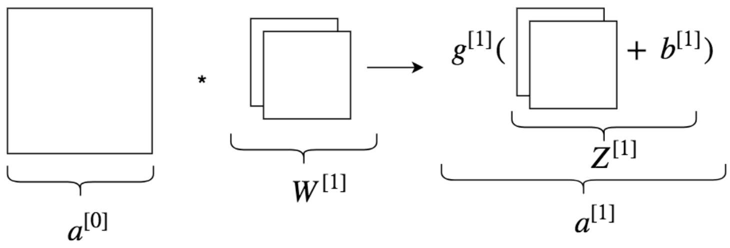


Figure 10: *A convolution layer; an image is convoluted by two filters, and we applied an activation function to the result adding a bias parameter*

As you may have noticed a new hyperparameter has shown up, that is the number of filter. Since each filter can serve as a feature detector, multiple filter can detect different things, (e.g horizontal edge detector and vertical edge detector in an image). In a NN the weights to learn are the coefficients of the kernel and the biases ($W^{[1]}, b^{[1]}$). The interests of this layer compared to a standard fully connected layer are multiple:

- On the efficiency side, such a layer is memory and computation efficient (less parameters, and the convolution operation is less costly than a matrix multiplication)
- The parameters are shared, for instance in the case where the data is an image, chances are that a feature detected (vertical edge) in one part of the image may be useful in another part as well.

Well, we have addressed most of the problems we put light on except those related to distorsions. We have defined a distorsion as a small change in the data. A function is invariant to distorsion or stable if a small change in the input does not change the output much, we find this concept also in machine learning as stability or regularization ([3] chapter 13). Then we can transpose the intuition behind using concepts from physics: for instance if someone put a ball in a hole, the ball is in a position of stable equilibrium, instead if the ball is at the top of a mountain, then the equilibrium is unstable and a small change will perturb that equilibrium. When a NN is distorsion invariant that mean that it is stable, for example a digit written character classifier is stable when it is able to recognize a letter whatever who has written it. To tackle this challenge a convolution layer often perform a pooling operation.

Pooling operation There is two major pooling operation, max pooling and average pooling set up using two parameters (filter size and stride) let see an example

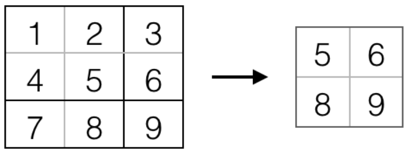


Figure 11: *Max Pooling: a kernel of size* 2×2 *stream the data finding the max at each step*

The max pooling operation is one of the most used in practice. How this relate to our distorsion consideration. Suppose the data to which the pooling is applied come from the a previous convolution layer, then a salient feature will certainly be strongly activated (recall the edge detection example). The result of a pooling layer is less interested by the position of the feature than it's presence, reducing noise around the feature to detect and keeping the interesting information, hence reducing variance.

Note that a complete convolution layer is composed of convolution and pooling operation. At this stage we have all the building blocks to give an example of a full CNN. We will succinctly present one of the first historical CNN LeNet-5 that has been use in hand written digit recognition [7], in fact an image of written digit is inputed and the network output a class prediction $(1, 2, \cdots, 10)$.

Nowadays multiple architecture have emerged with tremendous success in computer vision:

- AlexNet
- VGGNet
- ResNets
- Inception Network

The last model we will present is the RNN

8. Recurrent Neural Networks

8.1 Overview

RNN have been used in supervised sequence labeling with success in the industry, it is a task referring to learning to map a sequence of data to a

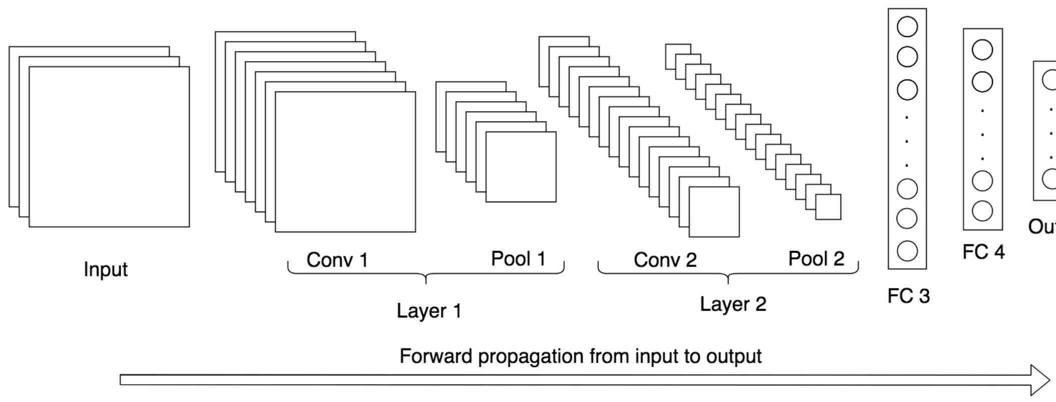


Figure 12: *LeNet-5 architecture, the first two layers are convolution layers, they include convolution and pooling operation, their output is then sent to two layers of fully connected neurons, to finally make a prediction. The arrow illustrate the flow of data through the network.*

sequence of labels. Some examples may include:

- speech recognition; sound speech $\mapsto$ sentence
- machine translation; text in english $\mapsto$ text in french
- sentiment analysis; comment on a movie $\mapsto$ number of stars

As before we might ask: why not using a standard NN to perform sequence labeling ? Well, NN don't take the structure of the data into account, each point is considered independent from the others, thus contextual information may be ingored. For example in speech recognition the same speech can be pronounced with different accent at various speed. Taking two sequence arbitrarily at different time step of a recording might not provide much information about the whole speech.

To tackle those issue RNN share parameters. To expand on that let's come to the building blocks.

8.2 Building Blocks

We already have all the material to give an example of a RNN, but let first introduce some notation.

- T_x, T_y: respectively size of the input x and the output y
- $x^{<1>}, \cdots, x^{<T_x>}$: a sequence
- W_{aa}: parameter shared across the network
- W_{xa}: parameter to ponder x
- W_{ya}: parameter to predict y
- b_a, b_y: bias of the model

In the litterature there is often two ways of illustrate RNN, with fold graph as in figure 13

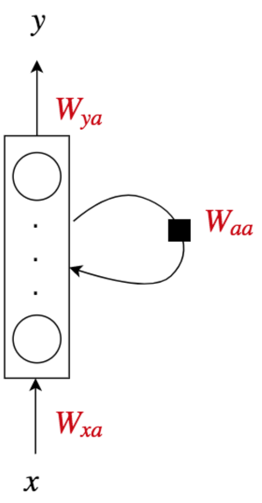


Figure 13: *RNN with fold graph. In red the learnable parameters. The rectangle with circles inside represents a layer of a NN*

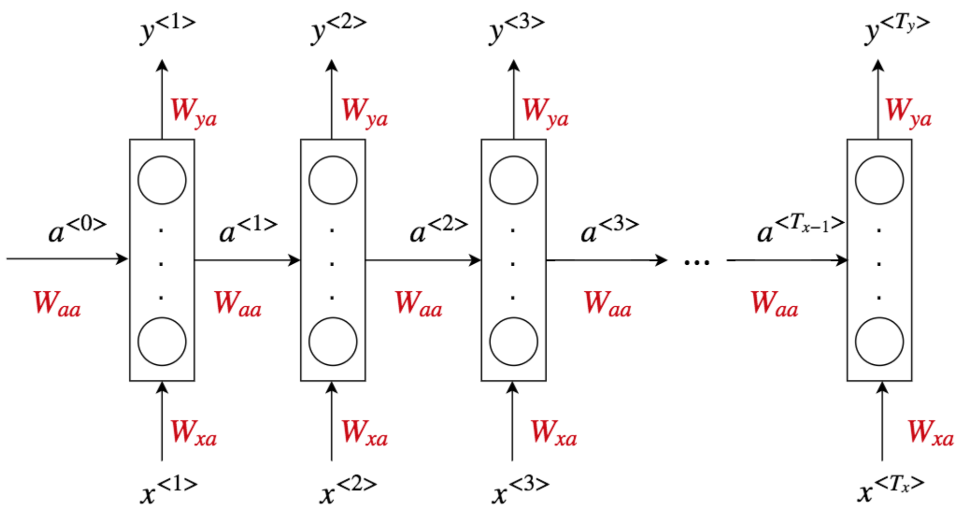


Figure 14: *RNN with unfold graph. The sharing parameter propriety is clearly displayed. Then the parameters learned must be balanced with respect to the sequences*

Let's define the forward propagation equations in the general case. The figure 14 may be useful to interpret them:

$$a^{<t>} = g_1(W_{aa} \cdot a^{<t-1>} + W_{xa} \cdot x^{<t>} + b_a) \tag{7}$$
$$y^{<t>} = g_2(W_{ya} \cdot a^{<t>} + b_y) \tag{8}$$

With g_1, g_2 being activation functions. It may not be the same, the non linearity we need to predict y may be different than the one for computing adequate shared parameters across the network. Note that we present an example of a *many to many* RNN architecture to handle sequence with multiple input and multiple output, as it is the case in machine translation or speech recognition.

Other problems may require *many to one* and *one to many* architecture type.

Perhaps, RNN are very effective for sequence labeling, but they have certain flaws:

- Firstly, it is hard to catch long term dependencies when we have very long sequences. In fact the sensitivity to contextual element tend to exponentially decay when we have more we have recurrent unit. In the literature this problem is referred as vanishing gradients [8]. For instance in machine translation the state of a verb in a sentence (singular/plural) can be determined long before, and chances are that the network just "forget" it.
- Secondly in their actual representation RNN do only take into account information from the past, in essence the recurrent units are only connected in one direction.

To cope with these challenge some elements were added. Long Short-Term Memory [9] and Gated Recurrent Unit (GRU) [10] have been shown capable of storing and accessing information over very long timespans, by redesigning the standard recurrent layer around *memory cell* units. Since they achieve comparable performance [11], we will use GRU, which are less complicated. To access all contextual information (past and future) Bidirectional Recurrent Neural Networks (BRNN) were proposed [12].The main difference difference being that recurrent units in BRNN are connected in both directions.

3. Results

As discussed above the two experiment were run, and we find the best model i.e the one that got the lowest development error, it turned out to be the DNN in the first experiment and the RNN in the second experiment see figure 15. To further consider the prediction made by these models we compare them to the original bottom to better understand what they have learnt, as you can see in figure 20, we only present the best models to have a sense of what is globally learnt, because the predicted bottom do not vary very much from a visual perspective. Finally the best model overall seems to be the RNN, the figure **??** compare the overall development error on both experiment and made the best performance clear.

4. Discussion and further work

Given the way we design the learning procedure, it means that the model that is the more able to adapt to a new dataset, (by adapting we mean getting a better prediction performance in a regression setting) is the RNN. In other words this architecture seems to better exploit whatever training data it has to make

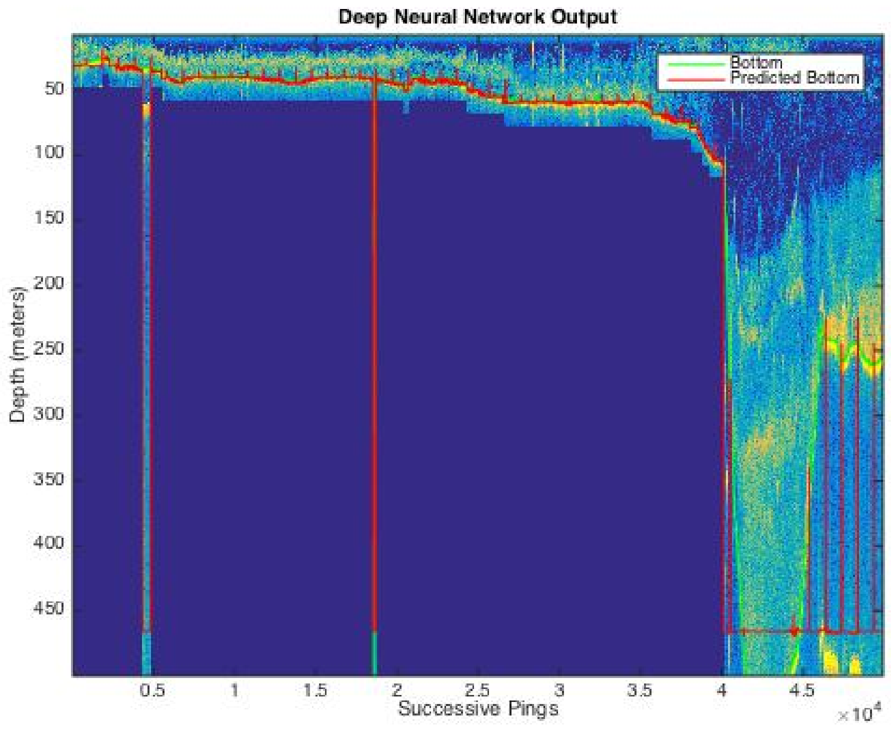
prediction on unseen data overall.
Note that the overall modest performance of these predictors is manly due to the fact that they have been trained on a non representative section of the campaing. In other words, the training set is to small to capture the full complexity of the whole dataset. The map in figure 3 reveal it pretty clearly. Furthermore, before deploying the RNN for bottom sea estimation without human supervision on further data we must think at what we might find acceptable as an output, and run a final experiment. It would consist of training the RNN on the full 2011 data and appreciate wether or not we have reached the acceptable performance on the 2015 campaign. The reverse experiment must also be successful, **only then** we will be in a position to say that our learning algorithm has reached acceptable performance for fully automated bottom sea estimation. If this approach reveal to not be satisfactory, a classification setting must be considered.

Models	**Experiment 1**		**Generalization Error**	**Figures**
	Train_2011-Error	Dev_2015-Error	Test_2015-Error	
LR	5.71	237064568.8		
SNN	5.72	33.14		17 - b
DNN	6.65	31.85	31.04	17 - a
RNN	12.27	34.56		18 - a
CNN	6.19	51.70		18 - b
Models	**Experiment 2**		**Generalization Error**	
	Train_2015-Error	Dev_2011-Error	Test_2011-Error	
LR	30.25	3077754954.5		
SNN	22.69	71.26		19 - b
DNN	33.30	97.05		19 - a
RNN	20.09	46.63	41.30	20 - a
CNN	30.35	117.33		20 - b

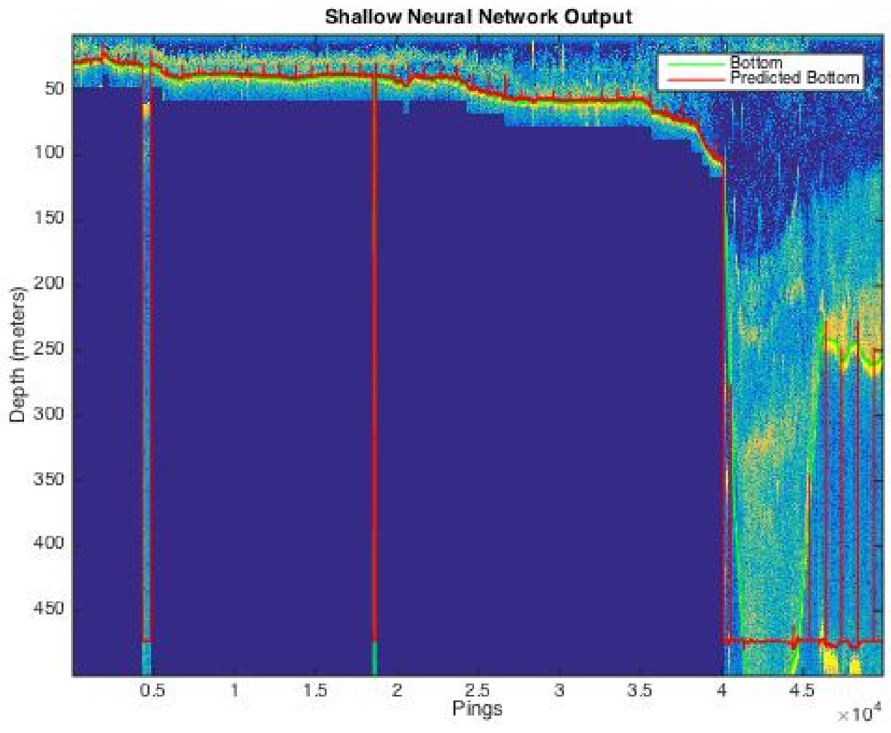
Figure 15: *Summary of the experiments. As a reminder in the first experiment the training set was sampled from 2011. The data from 2015 was split in a development and test set of equal size. The development set was used to tune hyperparameters. We highlighted the best model in green. Finally the test set was used to compute the generalization error. All the numbers are mean absolute error in meters.*

Models	Experiment 1 Dev-Error	Experiment 2 Dev-Error	Average Error
DNN	31.85	97.05	64.45
RNN	34.56	46.63	40.595

Figure 16: *Comparing the best two models of the experiment 1 and two averaging their development error we conclude that the overall best model to consider using the full dataset is the RNN.*

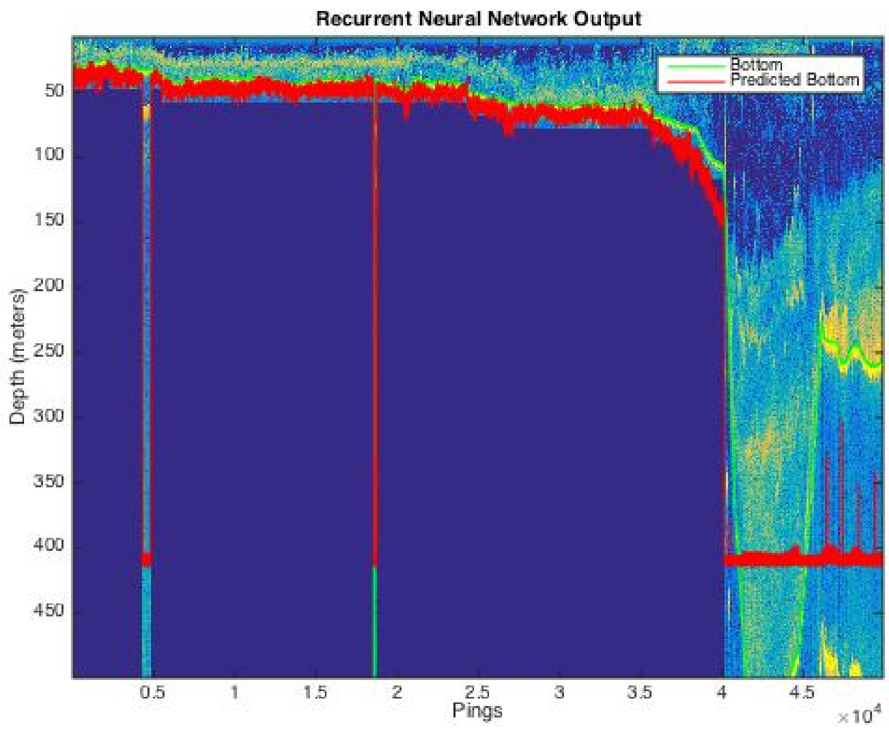


(a) *The DNN closely follow the real Bottom, but is affected by bad pings, and seem to have difficulty when the bottom sea is much shallower than the last non Nan value of the echogram.*

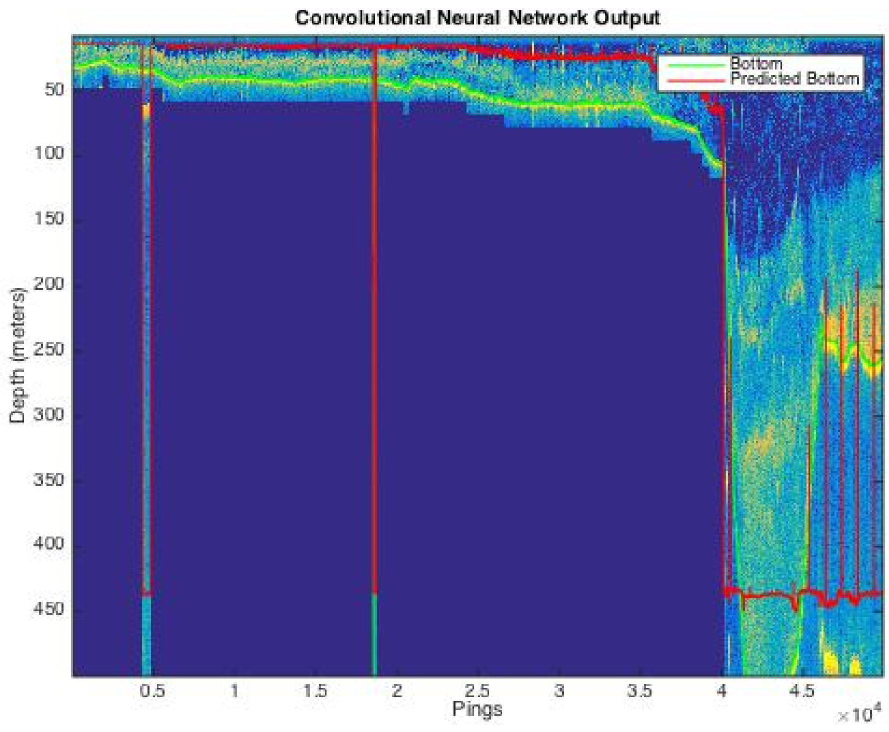


(b) *The SNN output a very similar prediction than the DNN*

Figure 17: *Experiment 1 - DNN and SNN*

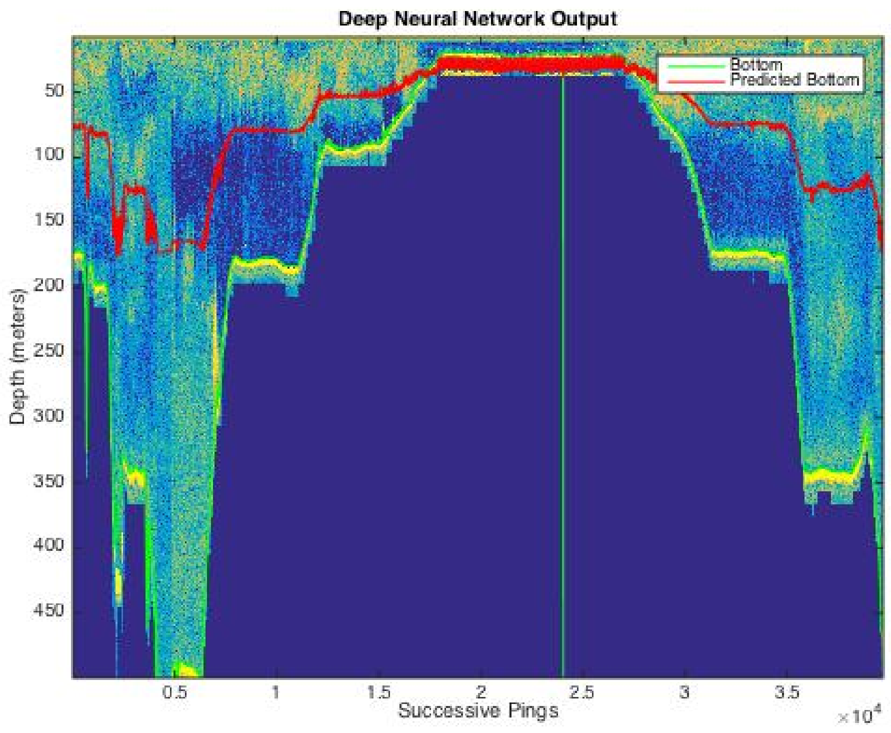


(a) *The RNN is slightly below the seafloor*

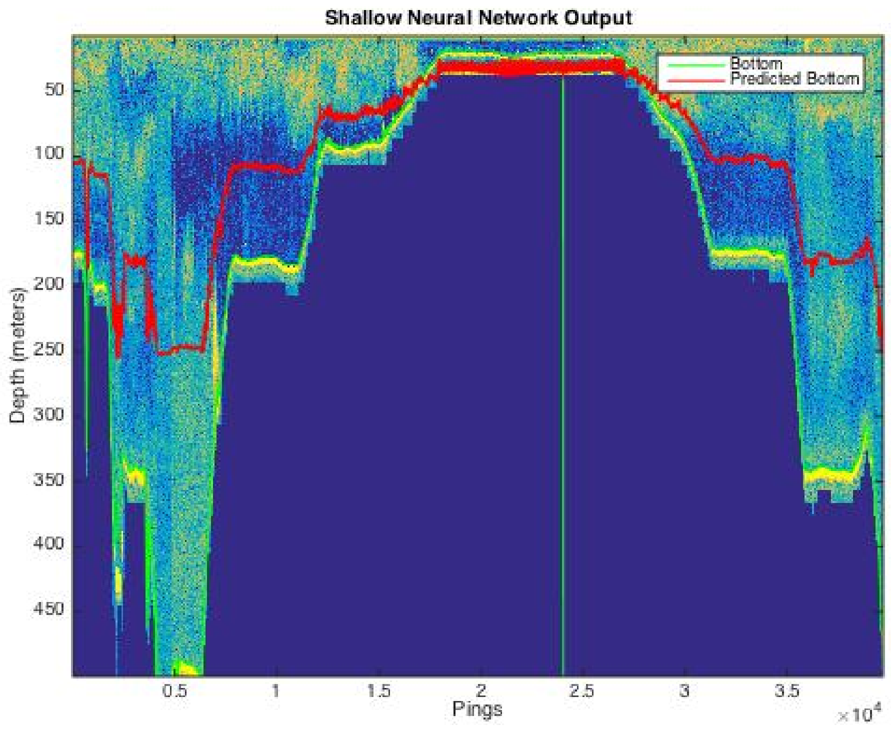


(b) *The CNN is systematically above the seafloor.*

Figure 18: *Experiment 1 - SNN and CNN*

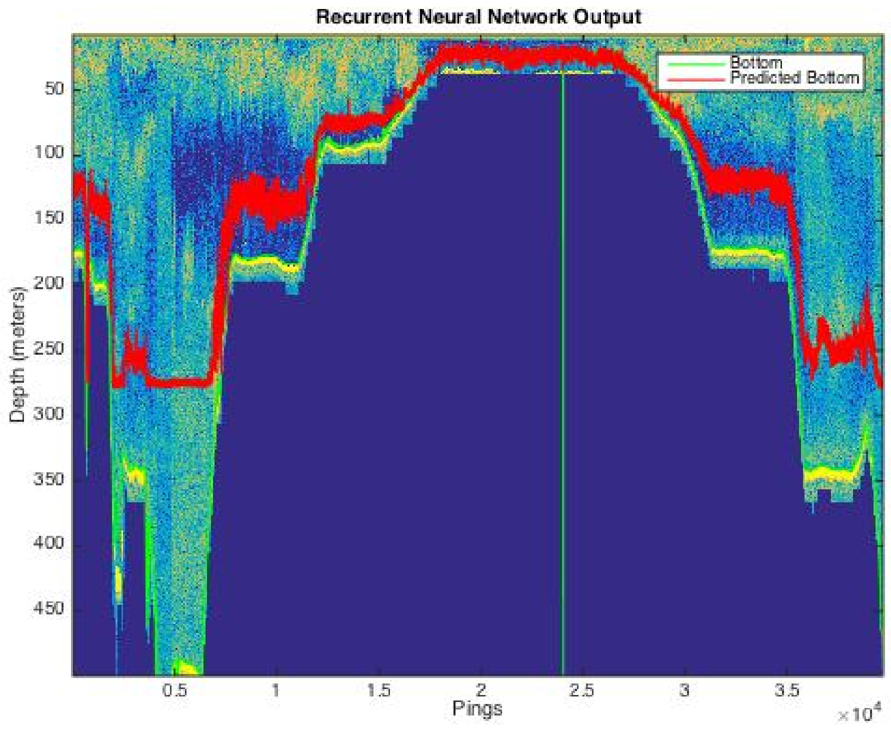


(a)

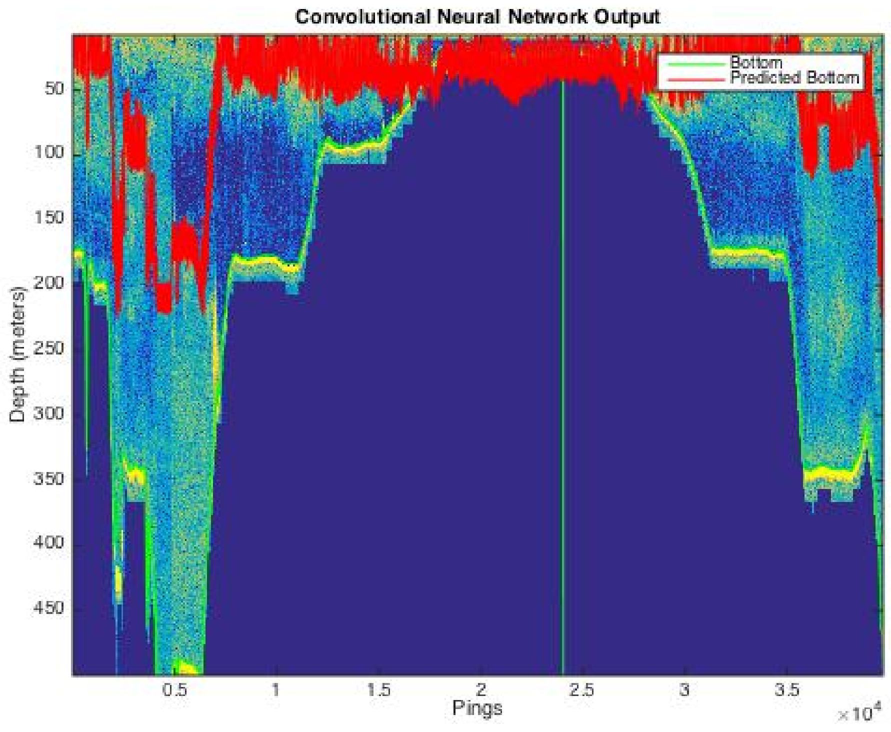


(b)

Figure 19: *Experiment 2 - DNN and SNN. As we can see the result are very similar.*



(a)



(b)

Figure 20: *Experiment 2 - SNN and CNN*

References

[1] Larry A. Mayer *Frontiers in seafloor mapping and visualization*, Marine Geophysical Researches, 2006

[2] B. R. Calder and L. A. Mayer *Automatic processing of high-rate, high-density multibeam echosounder data*, Geochem, Geophys. Geosyst, 2003

[3] S Shalev-Shwartz, S Ben-David, *Understanding machine learning: From theory to algorithms*, Cambridge university press, 2014

[4] G Cybenko, *Approximation by superpositions of a sigmoidal function*, Mathematics of control, signals and systems - Springer, 1989

[5] Andrew Ng, *Neural Networks and Deep Learning*, deeplearning.ai - Coursera - Massive Open Online Course

[6] Ian Goodfellow and Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT Press, 2016

[7] Y. Bengio, Yann Lecun *Convolutional Networks for Images, Speech, and Time-Series*, ResearchGate, 1997

[8] Y. Bengio, P. Simard, and P. Frasconi. *Learning long-term dependencies with gradient descent is difficult.*, IEEE Transactions on Neural Networks, 5(2):157?166, 1994

[9] S. Hochreiter and J. Schmidhuber. *Long Short-Term Memory.*, Neural Computation, 9(8):1735?1780, 1997

[10] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. *On the properties of neural machine translation: Encoder-decoder approaches.*, arXiv preprint arXiv:1409.1259, 2014

[11] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, arXiv:1412.3555v1, 2014

[12] M. Schuster and K. K. Paliwal. *Bidirectional Recurrent Neural Networks.*, EEE Transactions on Signal Processing, 45:2673?2681, 1997