2020-08-12

Deep Learning to Predict Ocean Seabed Type and Source Parameters

David Franklin Van Komen
Brigham Young University

Deep Learning to Predict Ocean Seabed Type and Source Parameters

David Franklin Van Komen

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Deep Learning to Predict Ocean Seabed Type and Source Parameters

David Franklin Van Komen
Department of Physics and Astronomy, BYU
Master of Science

In the ocean, light from the surface dissipates quickly leaving sound the only way to see at a distance. Different sediment types on the ocean floor and water properties like salinity, temperature, and ocean depth all change how sound travels across long distances. Hard sediment types, such as sand and bedrock, are highly reflective while softer sediment types, such as mud, are more absorptive and change the received sound upon arrival. Unfortunately, the vast majority of the ocean floor is not mapped and the expenses involved in creating such a map are far too great. Traditional signal processing methods in underwater acoustics attempt to localize sources and estimate seabed properties, but require *a priori* decisions and fall victim to ill conditioning and non-linear relationships between the unknowns and are computationally expensive. To address these problems, a deep learning method is proposed to distinguish between seabed types while also predicting source parameters such as source-receiver range from simulated training data. In this thesis, several studies are presented that explore the effectiveness of convolutional neural networks to make predictions from two types of sounds that propagated through the ocean: impulsive explosions and ship noise. These studies show that time-series signals and spectrograms contain sufficient information for deep learning, and additional preprocessing for feature extraction is not necessary. Training data considerations, such as randomness in the network weights and inclusion of representative variability are also explored. In all, this study shows that deep learning is a useful tool in underwater acoustics and has significant potential for seabed parameter estimation.

Keywords: underwater acoustics, source localization, seabed sediment prediction, ocean modeling, machine learning, deep learning

ACKNOWLEDGMENTS

It has been a privilege to study at BYU and I am grateful for the experiences that have come my way while attending this institution. So many different people have impacted me on this journey and I cannot possibly thank every one by name in this acknowledgements section. To everyone, thank you for all of the assistance and friendship you have shown over the years. I am truly grateful.

My first thank you goes to Dr. Tracianne Neilsen. Even before she became my advisor as an undergraduate, the acoustics course I took from her shaped the direction I was heading. She encouraged me to join the acoustics research group, and though I initially declined, she offered a position when I later came to my senses. Her expertise in acoustics has been invaluable as I've worked on this research. However, her kindness, encouragement, and passion have impacted my life in so many positive ways.

I'm also thankful for the members of my committee. Dr. David Knobles was the primary investigator of the research on the contracts that supported this research. The original underwater simulation code, weekly discussions on research progress, and expertise have all been enormously helpful during this research. I'm appreciative of Dr. Mark Transtrum for his advice and encouragement as well as showing me how much I enjoy predictive modeling and mathematical physics through the courses I took from him. I am also grateful for Dr. David Wingate for his expertise and for instilling a love of machine and deep learning within me.

I'm also grateful for the other acoustics faculty members that have helped me along the way. Dr. Brian Anderson, Dr. Scott Sommerfeldt, Dr. Kent Gee, and Dr. Timothy Leishman have all taught me principles that will help me through my life and career.

Another thank you goes to my friends and office mates Brian Patchett, Reese Rasband, and Adam Kingsley. Thank you for making our office a fun and productive space with all of the different stories, jokes, homework help, and discussions. Thank you to the students that assisted with and are continuing underwater acoustics research: Mason Acree, Kira Howarth, Stephanie Herron, Daniel Mortenson, Gabriel Fronk, Hadassah Griffin, and Jasper Forman. I'm also thankful to all of the other students in the acoustics research group at BYU.

I also must mention and give thanks to my family. Thank you to my parents for their constant support and for raising me to love understanding how the world works. Thank you to my siblings for their encouragement and friendship. Thank you to my extended family members for your support as well. I could not have been able to complete this without all of my family members.

Again, thank you to everyone I already mentioned and to those that I didn't mention. This journey could not have been completed without your support and kindness.

This research was supported in part by the Office of Naval Research, SBIR Grant #N68335-18-C-0806. Another portion of this work was funded by ONR contract number #N00014-19-C-2001. The Seabed Characterization Experiment 2017 was funded by the U.S. Navy Office of Naval Research. A large thanks the captains and crews of RV Endeavor is also owed. Also, thanks to the support from the BYU Department of Physics and Astronomy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In ocean acoustics, one of the factors that affects sound propagation is the seabed. Different sediment types allow for different amounts of reflection and absorption when sound interfaces with the sediment. As the sound travels through the water, reflections from the surface and seabed combine through constructive and destructive interference. Examples that illustrate the impact of the seabed on the received sound are found in Sec. 2.2. Because different seabeds link different effects on the sounds observed, seabed mismatch can impact technologies like SONAR. Thus, proper interpretation of the SONAR output when identifying objects in the ocean requires knowledge of the seabed. A machine learning approach to seabed identification has the potential to provide quick estimates of the seabed type that can allow for safer and more efficient SONAR operation in the ocean.

Because of the impact of the seabed on sound propagation, efforts to localize acoustic sources and estimate the ocean environment through acoustic methods are active areas of research. Challenges arise due to nonlinear relationships between the unknowns, high-dimensional search spaces, and large uncertainty due to ill-conditioning of the inferred parameters. Another challenge is the variability that also exists across multiple parameters such as sediment type (sand, mud, silt, clay, etc.) and thickness of sediment layers, the depth of the ocean, the depth-dependent speed of sound in the water, and many other parameters. Efforts to solve the source localization and environmental estimation problems have used two different types of algorithms to make predictions from data: model-driven and data-driven. The differences between these two highlight the potential for machine learning models in underwater acoustics.

1.1.1 Model-driven Solutions

Model-driven architectures are typically implemented by explicit relationships and rules that dictate a response. Model-driven solutions estimate model parameters by minimizing a cost function that represents the data-model mismatch. Model-driven approaches incorporate large bodies of knowledge that have been obtained through extensive research and iterative testing. For example, some classical systems can be described by the Newtonian laws of physics, which are model-driven as they can be represented as mathematical equations. More complex models can have more parameters, which are generally found by trial and error in many optimization schemes that minimize data-model mismatch. These models require *a priori* decisions such as data sampling rates, the search algorithm, bounds on parameters, whether to vary individual or multiple parameters at once, and what qualifies as a "good" data-model match. Many algorithms have been developed for model-driven solutions, such as simulated annealing, genetic algorithms, Markov Chain Monte Carlo sampling, and Bayesian approaches. Underlying model-driven approaches are parameterized based on the physics.

Propagation models for the ocean are based upon solving the wave equation with boundary conditions that represent the ocean environment. There are models that solve this problem through range dependent and range independent methods. Common range-independent ocean propagation models is the normal mode model ORCA[2] (used in this thesis) and KRAKEN.[3] Common range-dependent models include PE-RAM[4,5] which take an iterative approach through range-stepping. These propagation models are used in various model-driven algorithms to find source and environmental parameters.

The model-driven solutions in underwater acoustics use physics-based propagation models given a specified parameterization of the ocean and strive to minimize the model-data mismatch. This differs from strictly data-driven solutions (explained in depth in Sec. 1.1.2) in that they require an assumed model of sound propagation in their calculations. Some examples of model-driven solutions applied to underwater acoustics include matched-field processing[6] and matched field inversion.[7] Different optimization algorithms have been applied such as genetic algorithms,[8] Monte Carlo estimation algorithms,[9] simulated annealing,[10] improvements by applying broadband sound propagation,[11] a fast Gibbs sampler approach,[12] and more recently maximum entropy[13] and trans-dimensional Bayesian inference.[14] These optimization techniques can provide solutions, though some drawbacks are the need for information used to decide bounds and priors as well as computational expense due to their iterative nature, which requires running the propagation model for each model-data comparison. Estimation of environmental parameters using these traditional methods have seen success,[15,16] though the challenges remain.[14,16,17]

1.1.2 Data-driven Solutions

Data-driven approaches can make predictions based on patterns and features present within large datasets without an underlying mathematical model. Machine learning techniques are data-driven as they use input data to learn features that allow predictions to be made. These data-driven models do not need specific rules of physics hard-coded within, as they "learn" what pieces of the data are important through extensive iteration. The largest limitation to data-driven approaches is the amount of data required to learn the features needed to make accurate predictions. In ocean acoustics, the shortage of unclassified measured labeled data and the expense of creating such a dataset led this research to use data simulated with a propagation model for the studies in this thesis, as detailed in Sec. 2.2.

For the data-driven models used in this thesis, supervised deep learning (a subset of machine learning that structures algorithms into multiple layers) through neural networks is used. Supervised approaches allow the networks to learn to predict labels based on features observed in the labeled training data. For example, a label used in this thesis is seabed class. All data samples used as training must be labeled with their seabed class for the network to learn how to predict seabed class. Some unsupervised approaches can be used to categorize unlabeled data into distinct classes by finding divisions, but these divisions identified by the algorithm require additional interpretation and are not used in this thesis.

Machine learning techniques are actively being applied to the problem of acoustic source localization in the ocean. Early efforts in air acoustics, such as the work of Steinberg *et al.*[18] that introduced the idea of using neural network techniques to localize an acoustic point source in a homogeneous medium using single- and two-layer neural networks, highlight potential for such techniques. Underwater acoustic efforts include those by Lefort *et al.*[19] and Niu *et al.*[20,21] and found that machine learning classifiers for source range outperform traditional methods such as inversion and matched field processing. Others, such as Huang *et al.*[22] and Wang *et al.*[23] have had similar success with using neural networks to localize sources in the ocean. Other efforts include deep transfer learning between simulated and measured data,[24] direction-of-arrival estimation,[25] classification of underwater targets,[26] vessel detection and range estimation using cepstrums,[27] and using normalized acoustic matrices calculated through vertical line array data to estimate source range in deep oceans.[28]

Machine learning has also been used for estimation and prediction of ocean environmental parameters. Early efforts included the use of neural and statistical classifiers by Michalopoulou *et al.*,[29] artificial networks on transmission loss by Benson *et al.*,[30] and global and hierarchical approaches by Stephan *et al.*[31] Recently, Piccolo *et al.*[32] used generalized additive models for predicting compressional sound speed and attenuation from time-domain features. Frederick *et al.*[33] also used physics-based modeling and machine learning to classify the seabed. A more detailed review of machine learning and its applications in various fields of acoustics can be found in Bianco *et al.*,[34] and the deep learning methods used in this thesis are discussed further in Chapter 2.

1.2 Thesis Overview

This thesis presents research analyzing the potential of deep learning models to distinguish seabed types using recorded and simulated sound signals. Chapter 2 provides a foundation necessary to understand the deep learning models used throughout the various studies presented as well as the software used for deep learning (Sec. 2.1). Four "canonical" seabeds ordered by bottom loss are presented and parameterized as the basis for data simulation (Sec. 2.2.1).

Chapter 2 also details the data simulation process and the different sound sources used with deep learning to differentiate seabed types and provide predictions on source parameters. Two different types of sound sources are used: pressure time-series recordings of explosive charges and spectrograms of surface ships. Explosive charges have been useful for seabed characterization[35] as an impulsive source.[36] Surface ship sources have also been used for seabed characterization[37–40] as a well-modeled broadband noise source.[41] Example simulations of the different sound sources are also presented to demonstrate how environmental and source parameter differences change the received sound. For both of these source types, the deep learning methods are trained on simulated data then tested on both simulated and measured data.

Chapter 3 is a preliminary study using simulated explosive sound source recordings using feed-forward neural networks (FNNs) and convolutional neural networks (CNNs). It compares the results between the two network types when both utilize pressure time-series as inputs (Sec. 3.3). The work presented in this chapter was submitted and published in *Proceedings of Meetings on Acoustics* under the title "A convolutional neural network for source range and ocean seabed classification using pressure time-series."[42]

Chapter 4 extends the study in Ch. 3 by applying the networks to measured explosive charges. This study shows how well the CNNs can generalize training on only simulated data by application to measured data (Sec. 4.3). The standard is also set and used for the rest of the studies of training multiple networks of the same architecture to provide a better statistical analysis of network performance (Sec. 4.3.2). The work presented in this chapter was published in *JASA Express Letters* under the title "Seabed and range estimation of impulsive time series using a convolutional neural network."[43]

Chapter 5 continues the work of Ch. 4 by examining potential impact of ocean variability on the CNN performance. Specifically, the effect of using training datasets simulated with different degrees of ocean variance is evaluated for CNNs making predictions on measured pressure time-series explosive charges. Two simulated datasets are presented, where one contains upward-refracting, downward-refracting, and isovelocity ocean sound speeds with a single ocean depth and the other contains multiple ocean depths but only contains upward-refracting ocean sound speeds (Secs. 5.2.1 and 5.4). The work presented in this chapter constitutes the first time deep learning is applied to both source range and seabed type and is being prepared for submission to the *IEEE Journal of Oceanic Engineering*.

Chapter 6 shifts to the moving surface ship sound sources and provides preliminary evidence that deep learning methods can learn to make predictions on seabed type, ship speed, and closest point of ship approach (CPA range). Due to the lack of measured data, this study also shows model validation on separate training datasets with source parameters near and extending beyond the original bounds of the training set. The trained models are then applied to the one available measured data sample. At the time of writing, the work presented in this chapter is being prepared for submission to *The Journal of the Acoustical Society of America*'s special issue on machine learning.

Chapter 7 summarizes the main conclusions from the research presented in this thesis. Additionally, a summary of additional future work is given.

Chapter 2

Methods

2.1 Machine Learning Background

Machine learning is a data analysis method that builds predictive models (networks) through the idea that systems can be dynamically learned from data through the identification of patterns. In recent years, machine learning interest has exploded as computational hardware has increased in speed and capacity. Though multitudes of model architectures, techniques, optimizations, and methods are being published in the community almost daily, only a subset of deep learning approaches are used throughout this thesis. Because deep learning is a relatively new field in underwater acoustics, the feed-forward neural network (FNN) and convolutional neural network (CNN) methods are of particular interest. FNNs and CNNs are also convenient because of their relative simplicity in implementation.

The FNN[44, 45] is an artificial neural network that takes a set of inputs and maps them to outputs through learned linear transformations. This neural network is typically organized by a number of layers of "neurons" (a vector of numbers) where the first layer is the input, the last layer is the output, and any layers between are hidden layers. The particular value of a neuron at layer j is given by O_j through the following:

$$O_j = f(\text{net}_j), \tag{2.1}$$

$$\text{net}_j = \sum_{i=1}^{\text{nodes}} w_i x_i + b \tag{2.2}$$

where w_i is the weight applied to the previous node output x_i, b is a bias term, and $f(\text{net}_j)$ is an activation function chosen by the network designer (such as a sigmoid function or a rectified linear unit[46]). The network learns these weights and biases through iterative steps by calculating the error of predictions against the true output (known commonly as "loss"). This loss value is then used through back propagation (a gradient descent algorithm throughout all values) to update the parameters. The number of layers, the number of neurons, and other such values that must be decided before training are known as hyperparameters and are generally chosen through trial and error.

FNNs are useful for learning on data features that are representative of any raw phenomena, but they begin to fail on grid-structured data (verified by a preliminary investigation published as Van Komen *et al.*[47] in *Proceedings of Meetings on Acoustics*). Grid-structured data are represented discretely across an independent axis, such as time or frequency. Example's of grid-structured data in acoustics include pressure time-series data and spectrograms (pressure information divided into frequency bins via a Fourier transform across time). Features such as peak signal amplitude, average amplitude, or any other number of descriptions of a signal, are useful for describing acoustic phenomena but must be extracted either through determined algorithms or by hand. Raw grid-structured data, such as pressure recordings, are data rich and could contain features that a deep learning model can use to make better predictions.

The models used to handle grid-structured data throughout this thesis are CNNs.[48] CNNs have significantly more hidden values and parameters throughout multiple layers of the network than shallow models, like a simple FNN. For example, some of the most common CNN-based networks in the literature have millions of learnable parameters, like AlexNet,[49] VGGNet,[50] and GoogLeNet.[51]

The CNN uses a "convolution" that slides a learned filter across the data and calculates the dot product between the values of the filter with the input data. The convolution operation produces an activation map that can then be passed through another set of convolutions or directly input into a FNN (though the final combined network is still referred to as a CNN). CNN parameters are learned through the same iterative method as FNNs: loss calculation and backwards propagation through a gradient descent algorithm.

This thesis shows that CNNs have significant potential for classifying ocean seabeds and predicting source information from time-series pressure and spectrogram data.

2.1.1 PyTorch Framework

To implement the deep learning models and facilitate learning without needing to program all of the mathematical transformations, the open-source deep learning framework PyTorch was used throughout the studies presented in this thesis. PyTorch was written with "Pythonic" (follows accepted Python standards) syntax for usability and provides a simple interface into the CUDA libraries to interface with GPU hardware to accelerate learning. PyTorch includes a vast library of common mathematical operations used in deep learning, data processing tools, learning algorithms, and other tools necessary to build and train a machine learning model.

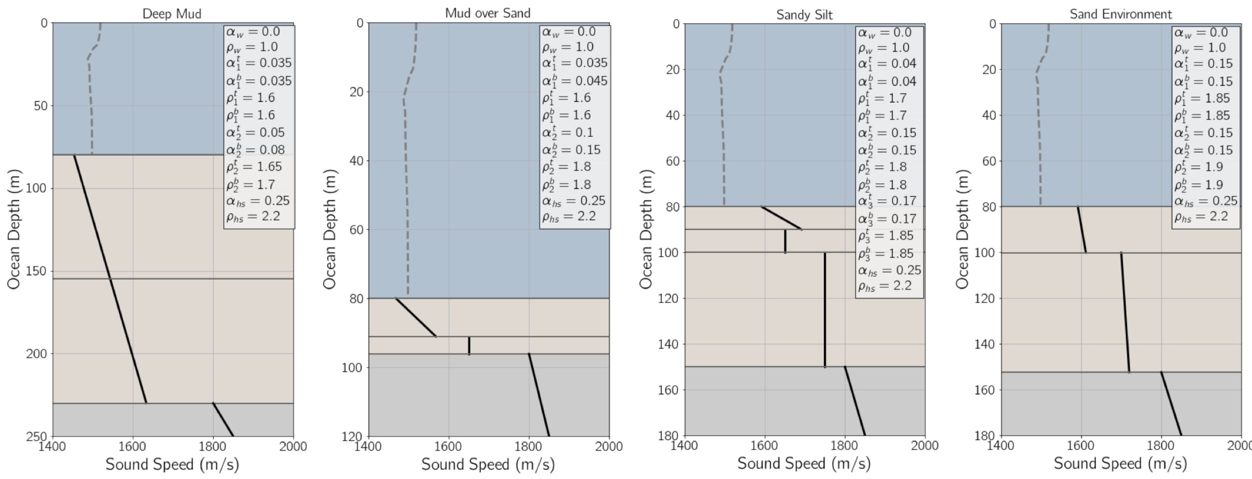
Appendix C contains a configuration file with all available options for the codes used for CNN learning. Each of the subsequent chapters contain details about the particular model structure and other operations used within PyTorch.

2.2 Synthetic Data Generation

Due to the lack of labeled field data and the need to generate sufficient amounts of training data for deep learning, synthetic data are generated. The synthetic data depends on (1) the source type and (2) the ocean environment. For the studies presented in this thesis, two different types of noise sources are simulated in different ocean environments. The first source type is known as Simulated Underwater Sound (SUS)[35,36] sources, which are explosive charges detonated at certain depths in the ocean. The second type is ship of opportunity (SOO) spectrograms. These two sources provide different types of data to be used in CNNs as the SUS charges are pressure waveforms represented only in time while the SOO spectrograms give pressure information across time and frequency.

To simulate the sound propagation through an ocean environment, the range-independent normal-mode model for acousto-elastic ocean environments ORCA[2] is used to calculate the mode functions at discrete frequencies. These mode functions can then be used to calculate the ocean response or Green's function that can be combined with source information to calculate a received signal. This response is also calculated with the assumption of azimuthal symmetry in cylindrical coordinates. David Knobles provided an initial MATLAB script for the data simulation, and all subsequent code that handles data generation relies on that original code.

The factors that affect sound travel are (1) the water depth, (2) the sediment layers on the sea floor and (3) the depth-dependent sound speeds in the ocean. Although the ocean contains an infinite variety of seabeds, these initial studies use a consistent set of seabeds. Four "canonical" seabeds were identified from the literature with different sediment layers for the ocean simulation. These seabeds are labeled as (1) deep mud, (2) mud over sand, (3) sandy-silt, and (4) sandy. The "deep mud" environment (seabed 1) comes from geoacoustic inversion on data from the Gulf of Mexico by Knobles *et al.*[11] The "mud over sand" environment (seabed 2) comes from maximum entropy statistical inference on data collected during the SUS circle experiments in SBCEX.[13] The "sandy silt" environment (seabed 3) comes from geoacoustic inversions done in the New England Bight by Potty *et al.*[52] The "sand" environment (seabed 4) comes from a study on sandy seabeds done by Zhou *et al.*[53] The parameters from those papers were then input into ORCA to calculate the normal modes. A visualization of these seabeds and their parameters is shown in Fig. 2.1.



(a) Deep mud **(b)** Mud over sand **(c)** Sandy silt **(d)** Sand

Figure 2.1 Parameterization of the four canonical seabeds. The thicker black lines indicate the depth-dependent sound speed. The attenuation, α, and density, ρ are listed in the legend with subscripts indicating water (w), layers (1-3), and the lower half-space (hs). These environments increase in reflectivity and are numbered 1 through 4 from left to right for the use of the machine learning algorithms.

These seabed diagrams show a vertical "slice" from the surface of the ocean to the sediment layers that constitute the ocean floor. The water column (in blue) has a dashed line showing a representative sound speed profile, typical of a shallow ocean. The water is assumed to have constant attenuation, α_w, and density, ρ_w, listed in the legend. Each sediment layer l is parameterized by a thickness h_l in meters, compressional sound speed c_l at the top and bottom of the layer in m/s, density ρ_l at the top and bottom in g/cm^3, and compressional attenuation α_l at the top and bottom in dB/λ. The bottom boundary, or half-space (hs), is defined by the compressional sound speed, attenuation and density. These parameters are all fed into the normal mode model ORCA to simulate the frequency-dependent response of the ocean.

Propagation in the ocean water also needs to be modeled when simulating data. The most important property is sound speed profile (SSP) which contains the sound speed associated with each depth point. In general, the SSPs for the studies presented in this thesis are based off of measurements taken during the 2017 Seabed Characterization Experiment (SBCEX 2017).[54] These SSPs are nearly linear with a slight positive slope and range between 1469 and 1472 m/s. The water depth over the area measures between 72 and 78 m deep. The exact sound speed profiles used when generating data are detailed in the studies presented in later chapters.

With the ocean environments (including the seabed and SSPs) parameterized, simulations are then performed to simulate SUS charges and SOO spectrograms. What follows in this section is a basic description of how the synthetic data generation script works and how it handles the two different types of sources.

2.2.1 General Procedure for Data Generation

Organization for the data generation script is necessary to effectively create large datasets. First, the data generation script reads multiple inputs through a configuration file. These inputs dictate parameters such as which source type to use, the length of the data, the positions of receivers, source-receiver ranges, seabed types, ocean sound speed profiles, and more. The program then sets up the parameters it needs for calculations later, like frequency arrays and time arrays. Then the simulation process loops through all seabed and SSP combinations.

For each combination, the program then loads in the seabed and SSP parameterizations and then calls ORCA to obtain the mode functions ($\phi(z)$, where z is depth in the ocean) for each frequency requested. The mode functions are then used to calculate the Green's function for that particular environment. Equation 16 in Ref. [2] shows how the calculated modes are summed to form the pressure field, $p(r,z)$ produced by a source at a depth z_s:

$$p(r,z) = \sqrt{\frac{2\pi}{r}} e^{i\pi/4} \frac{1}{\rho_s} \sum_n \frac{\bar{\phi}_n(z)\bar{\phi}_n(z_s) e^{ik_n r}}{\sqrt{k_n}} \tag{2.3}$$

where $\phi_n(z)$ is the n-th mode function evaluated at depth z, k_n is the n-th mode eigenvalue, ρ_s is the density at the source, and r is the horizontal range between source and receiver. The program then splits to simulate either the SUS charges or SOO spectrograms modeled as point sources. When it finishes, it saves the data so that it can be loaded later for machine learning purposes.

2.2.2 SUS Charge Simulations

The SUS charge model as proposed by Chapman[36] and refined by Wilson *et al.*[35] was used to mimic the charges that were recorded at SBCEX 2017. The model provides a pressure time-series representation of the charge's source signal at a range of 1 meter. This time-series signal is converted to the frequency domain via a Fourier transform. This spectrum is multiplied by the Green's functions at each frequency. Then an inverse Fourier transform yields the simulated pressure time-series. This process is done across all selected SUS charge depths and specified source-receiver ranges. Information about the specific ranges and source depths are provided in subsequent chapters for the studies performed.

An example of the simulated SUS charges across three different source-receiver ranges for all four canonical seabeds are shown in Figure 2.2. This figure demonstrates differences in received sounds with these different seabeds and ranges. As range increases, the amplitude of the received signals drop as expected. The simulated signals also show the relative reflectivity of the chosen canonical seabeds as the received pressures with the sandy seabed are much louder than those in the deep mud seabed. The shape of the signals and their ringing are also distinct across seabed and range. If these differences between seabed and source-receiver range can be seen by the naked eye in plots like these, then a CNN should be able to make the predictions as well. A comparison to measured SUS charges can be seen in Fig. 4.1. Studies involving predictions with SUS charges like these are discussed in Chapters 3, 4, and 5. These studies use normalized levels which discards the relative amplitude information resulting in a more challenging problem.

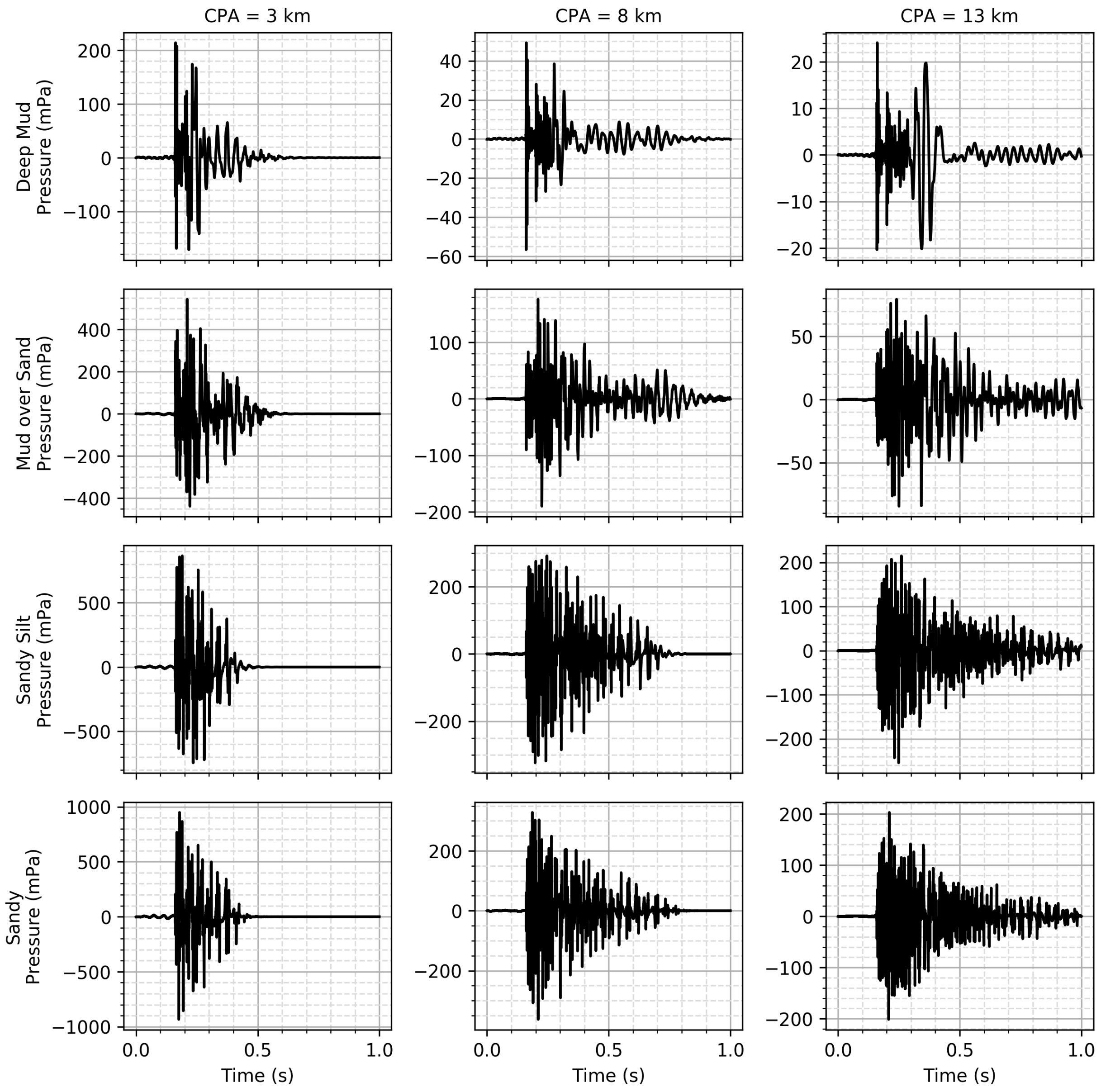


Figure 2.2 Simulated SUS charges for four different seabed types (rows) and three different source-receiver ranges (3, 8, and 13 km, columns). These signals were simulated at a source depth of 18.3 m below the surface of the ocean. Note the different pressure scales in the *y*-axis for the signals.

2.2.3 Ship of Opportunity Spectrogram Simulations

The SOO spectrograms differ from the SUS charges because they have dependencies in both frequency (f) and time (t). The source spectrum of radiated ship noise is simulated using the ensemble source model described by the equation in Sec. 3 of Wales and Heitmeyer.[41] This equation provides the sound pressure level (SPL) spectrum across frequencies in dB re $1\mu\text{Pa}/\sqrt{Hz}$, which needs to be converted back to complex pressure densities. The following transform is applied to convert to pressures:

$$P(f) = p_{\text{ref}} 10^{\text{SPL}(f)/20} \tag{2.4}$$

where p_{ref} is 1μPa. Since phase information is not available through SPL, a random phase is added to make the $P(f)$ complex. Each random phase is selected between 0 and 2π and applied through Euler's formula to convert the absolute pressure density $P(f)$ to a complex pressure density $\tilde{P}(f)$ with random phase. Due to the slow speed of the ship, the range of the ship varies slowly, so the Doppler shift effects can be ignored (see Eq. (2) in Ref. [55]).

The challenge with SOO simulation is calculating source-receiver range as the ship moves. The ship is assumed to travel with a constant velocity in the y direction over a specified time window (Fig. 2.3 shows the perspective). To calculate the horizontal source-receiver ranges, the closest point of approach (CPA) is assigned to the center of the time window, then the starting point (at the bottom of Fig. 2.3) is determined that keeps CPA in the center. As the y position changes through time, the range is calculated at each discrete time step, t. For each range calculated, $\tilde{P}(f)$ is drawn (with new random phase each time) and multiplied by the Green's function for each particular range. This creates a matrix of complex pressures $\mathbf{P}(f,t)$ where $¶_{ij} = \tilde{P}(f_i,t_j)$.

This procedure is repeated across all seabed and SSP combinations for multiple different ship speeds and CPAs to create a simulated dataset. Figure 2.4 shows generated ship spectrograms for the four canonical environments with three CPAs of 3, 8, and 13 km and the same speed of 15 kts. The large "U" shape for the CPA of 3 km is distinctive of SOO spectrograms and the curvature

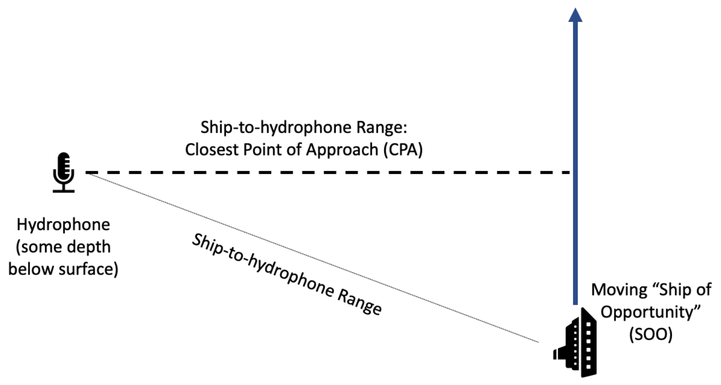


Figure 2.3 A simple diagram of a ship of opportunity's motion in relation to the receiver. This diagram is shown as if the viewer is looking down on the ocean surface. The ship is traveling with a constant velocity in the y-direction and the ship-to-hydrophone range is calculated for each time step.

of the "U" is related to the distance and ship speed. The levels decrease as CPA increases, as for the SUS charges in Fig. 2.2. The "U" shape also flattens out considerable as CPA increases, as all ship-reciever ranges are much closer to the CPA than at nearer CPAs.

To visualize the difference between speeds, another set of simulated spectrograms are presented in Fig. 2.5. In these spectrograms, the CPA is 3 km and the speeds are 5, 15, and 25 kts. The first noticeable difference between the different shapes is the sharpness of the "U" shape increasing and speed increases.

SOO spectrogram plots do not show environmental differences as easily as the plots for SUS charges do (see Fig. 2.2). At the frequencies shown, the deep mud and mud over sand environments have very similar shapes and levels. Some differences can be noted by the naked eye, such as deep mud levels decreasing more at the edges of the plots than mud over sand. The differences in sandy silt and sandy are even harder to detect in the pictures, though some small differences can be seen upon close inspection (particularly near the edges and the faint differences between lines). The similarity between spectrograms for pairs of seabed types provides a challenge for the machine learning models to distinguish between the different seabeds. Chapter 6 is a study using simulated

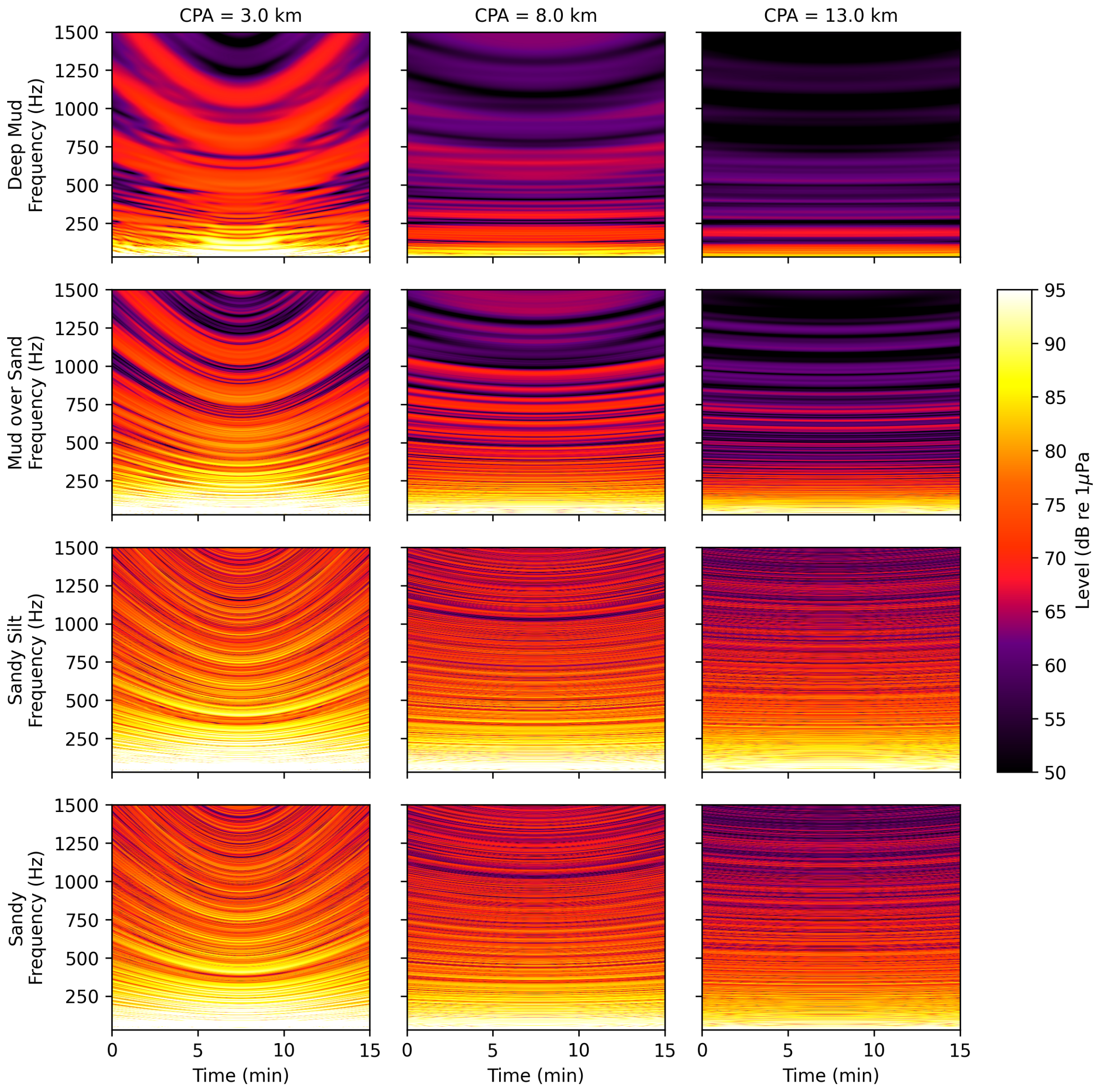


Figure 2.4 Simulated SOO spectrograms for four different seabed types and three different values of closest point of approach (3, 8, and 13 km). For these spectrograms the simulated ship is traveling at a constant velocity of 15 knots.

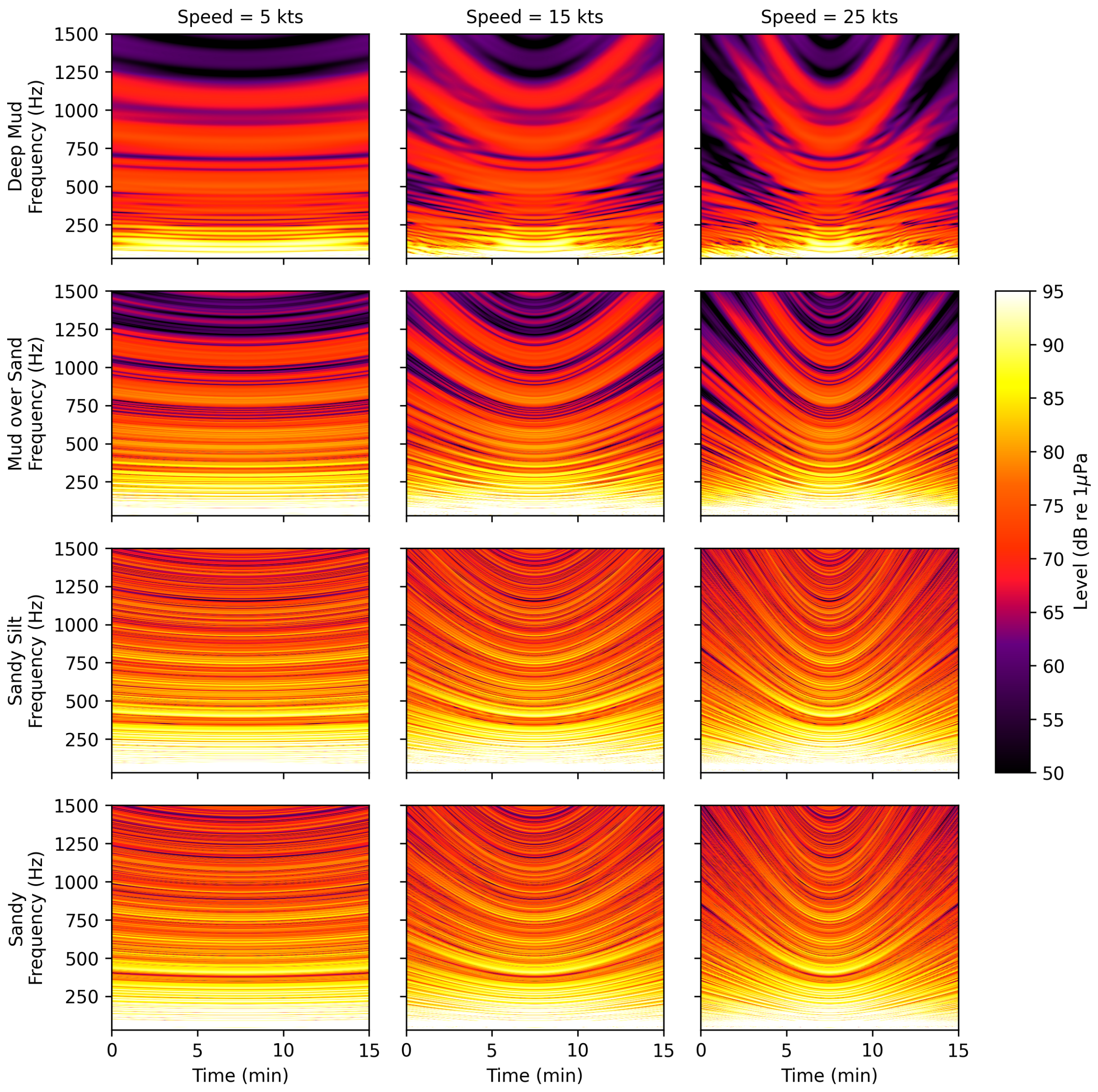


Figure 2.5 Simulated SOO spectrograms for four different seabed types (rows) and three different ship speeds (5, 15, and 25 kts, columns). For these spectrograms the simulated ship's closest point of approach is 5 km.

and measured SOO spectrograms to distinguish seabed type and predict CPA and speed. In that study, the source level of the measured SOO spectrogram at 1 m was unknown (S_0 in the equation in Sec. 3 of Wales and Heitmeyer[41]), so the spectrograms were normalized which discards range information in the relative amplitudes.

2.3 Summary

The deep learning methods presented in Sec. 2.1 are used throughout the rest of the studies in this thesis. Each study reports specific hyperparameters used to set up the models and during training. Simulated data methods presented in 2.2 are used to generate datasets used for training and validation of the machine learning models throughout all studies in this thesis. The four canonical seabeds used in this study are shown in Fig. 2.1. As with the models, specific source parameters used to generate the data are explained within their respective studies.

Chapter 3

Study: Preliminary Work using SUS Charges

The first tests using SUS charges were done using only simulated data. A feed-forward neural network (FNN) attempt that compared learning on extracted features and learning on pressure time-series was published in *Proceedings of Meetings on Acoustics*.[47] The results from that study indicated that FNNs perform better on extracted features than on the time-series waveforms. The next study shifted to using convolutional neural networks (CNNs) on the pressure time-series and was published as a separate article in *Proceedings of Meetings on Acoustics* as "A convolutional neural network for source range and ocean seabed classification using pressure time-series" in December 2019.[42] Comparisons between FNNs and CNNs were also shown in this work. The relevant work from that paper is presented in this chapter. The introduction (Sec. 3.1) of this work has been reduced for inclusion in this thesis as Chapter 1 contains a more exhaustive literature review.

3.1 Introduction

The goal of this paper is to expand beyond using a simple feedforward neural network (FNN)[45] for predicting environmental information from full pressure time-series through the use of convolutional neural networks. This paper contains preliminary results on a simulated dataset. Future work will use more sophisticated networks with simulated and real datasets.

3.2 Method

Predicting seabed type and source range with a CNN requires training data. This section illustrates how the data were generated, how specific features summarizing the data were selected, and how the network was designed for this problem.

3.2.1 Data Simulation and Feature Extraction

For this experiment, data were generated with SUS[36] charges in environments simulated by the range-independent ocean model ORCA.[2] The four different environments (muddy, mud over sand, sandy slit, and sand) are shown in Figure 2.1. The charges were simulated at 30 equally-spaced ranges ($r = 0.5 - 15$ km) away from a receiver. The charges were also simulated at three different depths ($z_s = 4$, 18.3, or 35 m). The environment type, range value, and explosion depth are the values that the network attempts to learn to predict. In the case of regression, the network attempts to predict the actual values independently as a regression problem, though the classification case attempts to predict which unique combination of values the sample belongs to.

The SUS charges were also simulated with 20 different water column profiles sampled from real and typical shallow-water measurements in an ocean 80 m deep to provide variability in the samples. The signals are approximately 14 seconds at a sampling rate of 1,000 Hz making each consist of approximately 14,000 discrete pressures. The 14 seconds was chosen to maintain absolute travel time within the signal which provides extra information about the range.

Before the data goes into the network, the dataset first needs to be normalized. For this particular case, the data were normalized by the absolute maximum of the entire dataset. This normalization not only places all values between -1 and 1, but it also maintains relative amplitude across the whole dataset. Another method considered is normalizing each sample individually by its own maximum, but in that case the relative amplitude information is lost.

3.2.2 Convolutional Neural Network Topology and Hyperparameters

The CNN used to predict environment and source range was built in Python with the PyTorch[56] package. PyTorch[1] is an open-source deep-learning platform that uses native Python syntax to quickly prototype, train, and test networks of any configuration. PyTorch also automates the required algorithms to perform gradient descent, learning rate scheduler components, and other useful algorithms making it ideal for proof-of-concept and production work. The CNN used consisted of 4 convolutional layers and 2 linear operations. The hyperparameters corresponding to each of the network layers can be found in Table 3.1.

The network was trained with the Adam optimizer[57] using a learning rate of 0.001 that annealed via a cosine function (a function included in PyTorch[56]) over 1,500 epochs. The loss function chosen was mean squared error loss,[58] which is useful for predicting the true physical values through regression. The learning rate was annealed to allow the network to approach the optimal weights early in training and then refine them later in training. The 7,200 data samples were split into training and testing datasets with a random 80/20 split (5,760 training samples, and 1,440 testing samples randomly divided each training instance). The results of training this network are shown using 1,440 testing samples not used during training.

[1]More information about how to use PyTorch can be found on their website at https://pytorch.org/.

Table 3.1 Details of the CNN. Layers in the network along with operation type, kernel size, number of channels, stride, and what type of activation function was used at the end of the layer. In the case of the linear operations, the kernel size indicates the number of output nodes. "Conv1D" is an abbreviation for one-dimensional convolution.

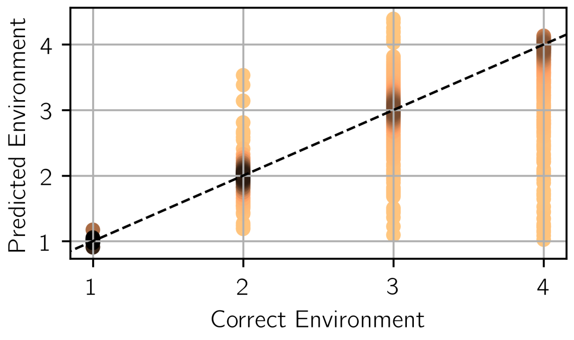
Layer	Layer Operation	Kernel Size	Channels	Stride	Activation	Batch Norm
1	Conv1D	16	3	4	ReLU	Y
2	Conv1D	8	9	4	ReLU	Y
3	Conv1D	8	18	4	ReLU	Y
4	Conv1D	8	27	4	ReLU	Y
5	Linear	500	N/A	N/A	ReLU	Y
6	Linear	3 or n_{classes}	N/A	N/A	Linear or Softmax	N

3.3 Results

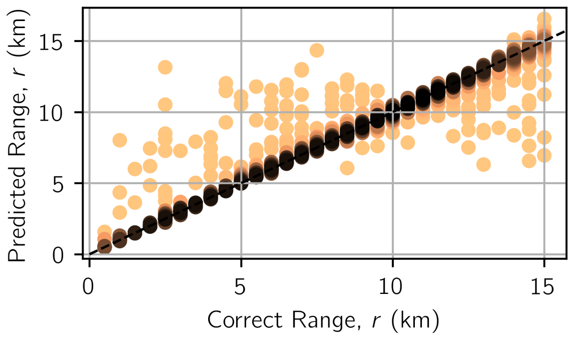
The results have been divided into two sections. The first sections shows the performance of a FNN on this dataset of simulated pressure time series. (A previous study with an FNN was limited to 135 data samples.[47]) The second section provides the results for the CNN detailed in Section 3.2.2 on the dataset.

3.3.1 Feedforward Neural Network Results

To provide a baseline result, the dataset was input into a FNN. This FNN consisted of two hidden layers with 5,000 nodes each, was trained across over 300 epochs, and predicted a number between 1 and 4 for a seabed type, and value in km for the source-reciever range of the signal. Figure 3.1 shows the predictions the trained network made on environment class and source range independently on the validation data. The colors in Figure 3.1 are used to show the density of the predictions since there are over one thousand points on each plot. The FNN was asked to only predict the



(a) Predicted vs correct environment.



(b) Prediction vs correct range.

Figure 3.1 Prediction results from the FNN. There are 1,440 different points on these plots, so the colormap scale shows the density of points (the darker the color, the denser the scatter plot). When the environment predictions (Figure 3.1a) are rounded to the nearest integer, an 85% accuracy is achieved. The source range predictions (Figure 3.1b) have a RMSE of 1.19 km when compared to the true range. The dashed line shows the values that would be obtained if the network made an exactly correct prediction.

environment number and the source range to better match the previous study on FNNs,[47] where more information about setting up a FNN can be found.

To determine the accuracy of the environment class, the predicted environment value was rounded to the nearest integer and compared against the true environment. With this rounding, the network was 85% accurate on environment. The network correctly identifies the first environment, but as soon as more reflectivity is added, the network struggles, especially with the sandy silt and sandy seabeds. However, the dark color near the correct answer indicates a high density of points near there, so only a small number of these samples are actually incorrect as indicated by the 85% accuracy. The source-receiver range predictions had a root mean squared error (RMSE) of 1.19 km when compared to the true range. The FNN tends to over-predict the range until around 8 km and then it begins to under-predict the range. However, the FNN still gets close to correct range on most of the training samples.

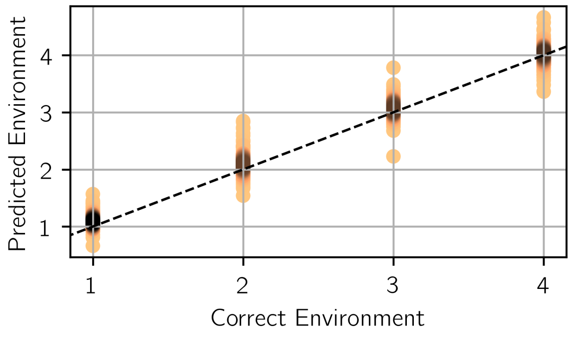
If this FNN network was instead configured for classification, the FNN was 71.2% accurate on the 120 unique range-seabed classes. These results are significantly better than the results that used pressure time-series instead of extracted features found in a previous study using FNNs.[47] This improvement could be due to the larger training dataset or the inclusion of absolute travel time in the signals.

3.3.2 Convolutional Neural Network Results

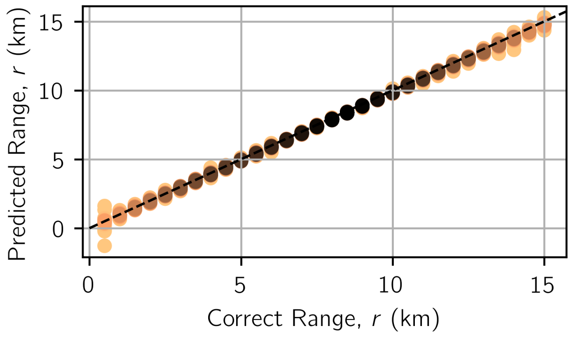
The CNN was trained on the same dataset but was designed and trained to learn environment number, source range r, and source depth z_s. Figure 3.2 shows the results of the CNN on the same size of validation dataset from a different random split. A visual indicator that the network has learned well is how clustered the different values are and how dark the points get near the correct answer line.

These figures show interesting trends when compared to the FNN. First, in Fig. 3.2, the predictions all appear closer to the true value across all samples and all predictions. As for the seabed environment prediction (as seen in Fig. 3.2a), there are several predictions that are outside the dense cluster, but the majority are correctly predicted. The error in range prediction (as seen in Fig. 3.2b) increases towards the min and max ranges, but are still relatively close to the correct values. This increase is likely due to the fact that there are not training data points beyond these extremes to help the CNN distinguish these ranges. In comparison with the FNN results, which had some wildly inaccurate predictions, the CNN predictions are mostly correct or incorrect by only a small margin. The source depth predictions (as seen in Fig. 3.2c) show a distinct grouping around the correct source depth.

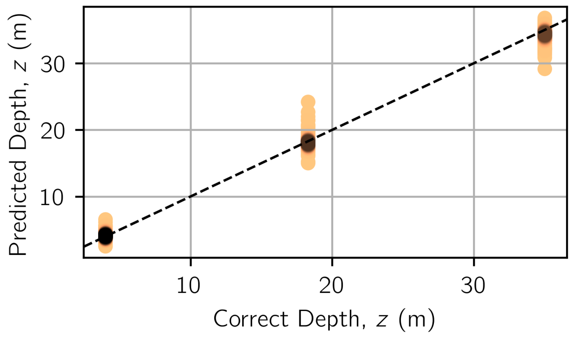
Comparison of the accuracy confirms the superiority of the CNN to the FNN. The RSME for the CNN range predictions is 0.1798 km—an order of magnitude better than the predictions of the FNN. The accuracy of the environment prediction is 97%, which is 13% higher than that of the FNN. The CNN also predicts a third label, the source depth, with an RMSE of 0.7633 m. The CNN



(a) Predicted vs correct environment.



(b) Predicted vs correct range.



(c) Predicted vs correct source depth.

Figure 3.2 Prediction results from the CNN. There are 1,440 different points on these plots, so the colormap scale is used to show density of points (the darker the color, the denser the scatter plot). When the environment predictions (Figure 3.2a) are rounded to the nearest integer, a 98.96% accuracy is achieved. The source range predictions (Figure 3.2b) has a RMSE of 0.1798 km when compared to the true range. The source depth predictions (Figure 3.2c) has a RMSE of 0.7633 m. The dashed line shows the values that would be obtained if the network made an exactly correct prediction.

performed much better than the FNN even when predicting an additional label.

The results described come from the CNN structured for regression, but the final output layer can be modified for classification. Beforehand, each sample must be labeled into a class representing its combination of source-receiver range, source depth, and environment. This creates 360 unique classes for the network to learn. The same 80/20 training/testing data split was used. The CNN was 97% accurate in classifying the 1440 testing samples, which is 26% higher than the FNN on triple the number of classes.

3.4 Conclusion

This paper has shown that CNNs have the potential to learn source range and environmental classification from time-series waveforms. The CNN on the waveforms performed better than an FNN applied to the same set of data. This improvement implies a significant advantage to using convolutions on the grid-like structure of pressure time-series data. This study also lays the ground work for developing deeper and more complex networks, such as U-net CNNs[59] and recurrent neural networks (such as long short-term memory networks[60]), which have also shown success in image and speech recognition. Future work seeks to expand on these results by increasing the amount of data, increasing the number of environments used, application of noise to training and testing data, and applying the networks to real measured data.

Chapter 4

Study: CNN Predictions on Measured SUS Data

The study presented in this chapter extends the work done with CNNs in Chapter 3 by applying trained CNNs to measured SUS data. The contents of this chapter were published in *JASA Express Letters* as "Seabed and range estimation of impulsive time series using a convolutional neural network" in April 2020.[43] The introduction (Sec. 4.1) of this work has been reduced for inclusion in this thesis as Chapter 1 contains a more exhaustive literature review.

4.1 Introduction

In this paper, a machine learning model known as a convolutional neural network (CNN) is used to simultaneously predict source range and seabed type. This CNN is trained on synthetic pressure time waveforms and, as a proof of concept, applied to waveforms measured on a single pressure sensor[1] from an impulsive SUS source[35,36] in the New England Mudpatch during the 2017 Seabed Characterization Experiment (SBCEX2017).

4.2 Background and Methodology

Several steps are needed before a CNN can simultaneously predict source range and environment type on a real-world measurement. First, the CNN training data needs to reflect the real-world testing data and incorporate the variability likely to be found. The real-world testing data used in this work is described in Sec. 4.2.1. Second, due to the lack of real-world labeled data, simulated training data was generated using a verified source spectrum and a propagation model, as described in Sec. 4.2.2. Finally, the training data are used to tune the hyperparameters of the CNN. The CNN model architecture and tools used to build and train the model are presented in Sec. 4.2.3.

4.2.1 Intensity Vector Autonomous Recorder in SNCEX2017

SBCEX2017[54] was conducted on the New England Mud Patch centered at [408′ N, 705′ W] in spring of 2017. The Intensity Vector Autonomous Recorder (IVAR) system deployed by the Applied Physics Laboratory, University of Washington recorded the waveforms of SUS Mk64 charges deployed at ranges of 2-13 km. Three to five SUS were deployed at each location. More details about the system, the recordings, locations of the SUS stations, and the experiment conducted can be found in Ref. [1].

While IVAR has multiple sensors for measuring both pressure and particle velocity, the data used here are restricted to the pressure sensor located 1.32 m above the seabed. Examples are displayed in Fig. 4.1(a) and (b). Because the signals were downsampled to 5000 Hz and isolated to one second, the simulated training data are each one second long and have a sampling frequency of 5000 Hz. To facilitate the generation of trained data for this study and to provide proof of concept, only the data collected from a single pressure sensor IVAR was used; additional SBCEX2017 sensors can be used in future studies.

The environment at SBCEX2017 informed the simulation of the training data. The SUS charges

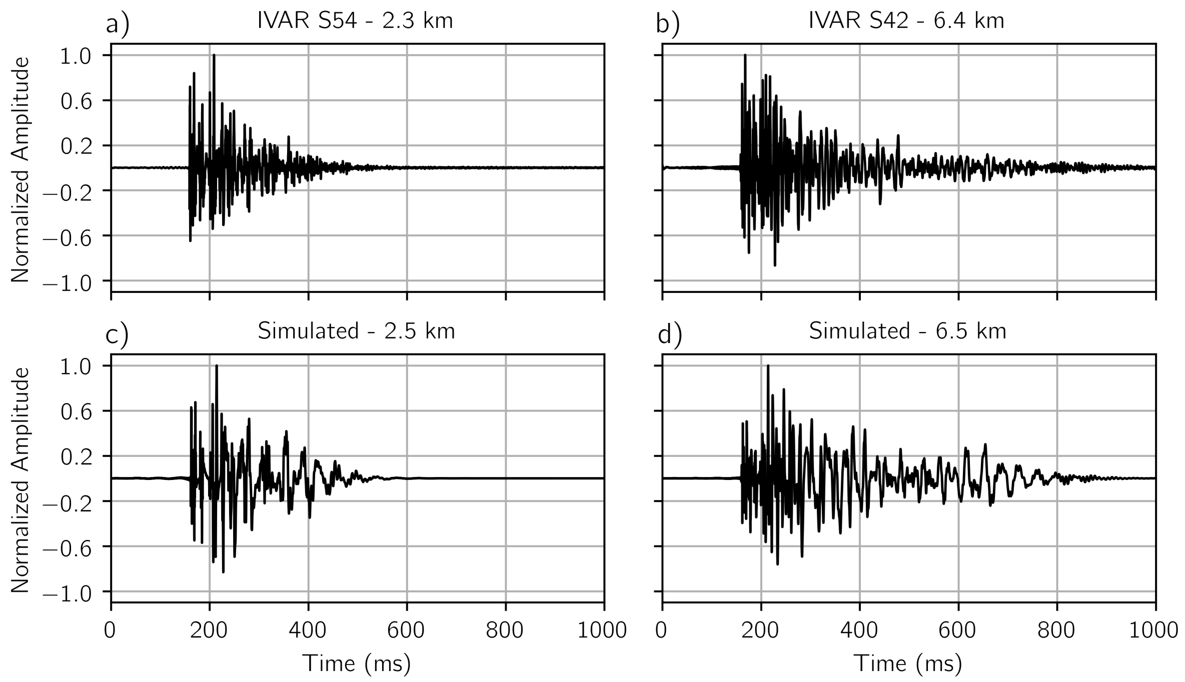


Figure 4.1 Normalized pressure waveforms measured on IVAR from SUS station S54 (a), and S42 (b) (the exact positions are shown in Fig. 3 of Ref. [1] Fig. 3) and two simulated, normalized waveforms, in a representative environment, at 2.5 (c) and 6.5 km (d) away from the source.

were deployed in an area with an average ocean depth of 74 m and a nearly isospeed sound speed profile in the water (as seen in Fig. 2(b) in Ref. [1]): varying from 1467.8-1468.6 m/s. The New England Mud Patch has an uppermost sediment layer consisting of fine-grained mud-like material.[61] Thus, training data are simulated for a variety of linear sound speed profiles and four different of seabed types, one of which was inferred from the analysis of the SUS waveforms on a different receiving array.[13]

4.2.2 Synthetic Data Generation

Machine learning models require a significant amount of training data to identify and learn patterns. Because of the lack of labeled field data, synthetic training data is used. The range-independent, normal-mode model ORCA[2] model was used to generate the ocean's impulse response at frequencies up to 2500 Hz. This impulse response is convolved with a simulated SUS Mk64 signal spanning a bandwidth of 5-2500 Hz to produce the simulated time-series waveform using the model in Wilson *et al.*[35] Examples of the simulated samples are shown in Fig. 4.1(c) and (d).

The time series were simulated using four different seabeds representing deep mud, mud over sand, sandy silt, and sand environments. The "deep mud" environment comes from a study done in the gulf of Mexico by Knobles *et al.*[11] The "mud over sand" environment comes from maximum entropy statistical inference on data collected during the SUS circle experiments in SBCEX.[13] The "sandy silt" environment comes from geoacoustic inversions done in the New England Bight by Potty *et al.*[52] The "sand" environment comes from a study on sandy seafloors done by Zhou *et al.*[53] A visualization of these seabeds and values for sound speeds, densities, and attenuations is shown in Fig. 1 of Van Komen *et al.*[42]

A variety of sound speed profiles were also used in the simulation to expand the training dataset. Fifty different linear, downward-refracting sound speed profiles were used with water depths between 73-75 m, representative of the measurements taken at SBCEX2017. These profiles (spanning 1465 - 1470 m/s) were selected to add variability that was present in the measurements. This variability also allows the machine learning model to better generalize the internal feature extraction used for prediction.

Source and receiver parameters were selected to cover the variability in the measured IVAR data. The signals were simulated to be received on a hydrophone located 1.3 m from the ocean floor at ranges between 0.5 and 15 km at 0.5 km intervals. The true IVAR-to-SUS charge ranges (as reported by station locations in Ref. [1]) were not used in the simulations so as to test the model's ability to generalize on the range label. Because the nominal source depth of the SUS charges is 18.3 m, the simulations were performed at eight different source depths spanning 10.3 to 24.3 m from the surface to account for any variability.

With the four seabed types, 50 sound speed profiles, 30 ranges, and eight source depths, a dataset of 48,000 signals was generated. The one-second signals, normalized and sampled at 5000 Hz, were aligned with the arrival time occurring at approximately the same time in each sample, as illustrated in Fig. 4.1. Thus, the data contain no information about absolute travel time or absolute amplitude.

4.2.3 Machine Learning Model Architecture and Training

The machine learning model employed in this study is a Convolutional Neural Network (CNN). A CNN is chosen due to its ability to find patterns in gridded data, which makes it an ideal candidate for analysis on time-series waveforms. Using a CNN also eliminates the need for data preprocessing because a CNN learns to extract features necessary for accurate predictions. More information on CNNs can be found in Goodfellow *et al.*[58] The CNN architecture used in this study is a simple network with four convolutional layers and two fully connected layers and is the same network architecture found and explained in more detail in Van Komen *et al.*[42]

To implement and train the machine learning models, the Python package PyTorch[62] is used. PyTorch automates many of the functions used in machine learning for developing and training models and provides an interface for learning on a GPU, a necessity for training quickly on 48,000 training samples. PyTorch also provides the algorithms needed for the models to learn, such as the Adam[57] optimizer used to train the network's weights.

The dataset of 48,000 simulated signals were labeled with the source-receiver range (in km) and a number representing the seabed type. The dataset was randomly divided into a training-validation split of 95%/5%, leading to 45,600 training and 2,400 validation samples. This split was chosen because the final testing dataset is the IVAR signals. Validation errors for this network are reported in Van Komen *et al.*[42]

For a single training session, the CNN was trained over 200 epochs. The learning rate began at 0.001 and was annealed via a cosine function to allow the algorithms to make large adjustments in the early epochs and then make smaller adjustments in the later epochs. These hyperparameters were selected over several trial runs and led to sufficiently low validation error; the results on the IVAR SBCEX2017 data (Sec. 4.3) also confirm the selections. Unfortunately, there is no special formula or method for selecting hyperparameters, so there is a possibility that other hyperparameters could give more precise results, but for the sake of consistency, these parameters were chosen and used across all tests in this study.

The network was trained multiple times to get a broader picture of the potential the CNN has to make predictions on the IVAR SBCEX2017 data. This process of training multiple networks was chosen to account for the random initialization of weights and the random training-validation split. Ten instances of this training-validation split led to ten different networks that were then applied to the IVAR SBCEX2017 data. The predictions from these ten different networks are shown and discussed in Sec. 4.3.

4.3 Results and Discussion

The results section has been divided into two subsections: results from a single trained network applied to the IVAR SBCEX2017 data and results from multiple trained networks. The division of the results is to illustrate how different random initializations and splits of training data can lead to different predictions. The network, in all cases, attempts to make predictions via regression and outputs two values for each sample: the range (in km) and a number representing the seabed type.

4.3.1 Individual Network Results

After training the CNN on the simulated data, the 37 IVAR SBCEX2017 data samples were passed through the network to obtain predictions of range and seabed type. The predictions of the CNN from two separate networks are shown in Fig. 4.2. The predicted ranges are compared to the

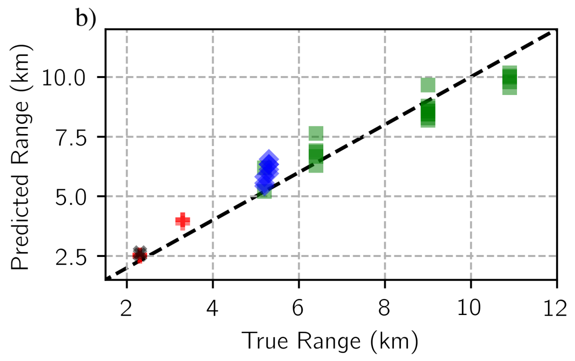


Figure 4.2 (color online) Predictions from one training instance of the CNN on the 37 data samples of SUS signals recorded on IVAR. The "True Range" corresponds to the measured distance; the diagonal dashed line indicates where the "ideal" predictions should lie. The color/symbol indicates the predicted seabed type. The red pluses indicate the "sandy" seabed, the blue diamonds indicate the "sandy-silt" seabed, and the green squares indicate the "mud over sand" seabed while the black crosses indicate an environment out of scope.

measured IVAR-to-SUS ranges spanning 2-12 km. If the network predictions matched the true ranges, the points would lie exactly on the diagonal line. In both cases, the signals of range 7 km and closer are overpredicted by the network, and those farther away are closer to the truth or slightly underpredicted. In these cases, the root mean squared error for all range predictions is 0.84 in (a) and 0.7 km in (b).

For each data sample, the CNN also yields a number corresponding to a prediction of the seabed type. The predicted seabed type is a decimal number, which is rounded to the nearest integer to easily display the results. In Fig.4.2, the different symbols/colors indicate which seabed type is predicted. For the two cases shown, the CNN predicts that the signals farther than 5 km from IVAR are identified as being from the "mud over sand" seabed (green squares), which most closely resembles the New England Mudpatch area. However, the network is predicting environments that are more reflective: the "sandy silt" environment (blue triangles), the "sandy" environment (red pluses), and even a "5th" environment (black crosses) which does not exist for the closer ranges. This variability likely ties to the physics of the sound propagation. One potential explanation is the variation in propagation paths. Another potential explanation is that the longer ranges sense primarily the upper portion of the seafloor, and, when the range is shorter, deeper features of the seafloor have more impact the propagation. Thus, the "mud over sand" seafloor represents what influences propagation to ranges greater than 6 km, while the tendency towards a seafloor with less bottom loss at short ranges indicates that the deeper features of the "mud over sand" seafloor are likely too absorptive.

4.3.2 Multiple Network Results

After training and applying the network that produced the predictions shown in Fig.4.2, nine more networks were trained to test how well the network architecture generalizes with multiple initializations. The number of networks (ten) was selected because adding more instances did not signficantly change the distribution of the results, which are displayed using violin plots. Violin plots show the median (white dot), inter-quartile range (black rectangle), and the full distribution of the data (in color). The violin plots for results from the ten trained networks are shown in Fig. 4.3, where (a) shows the distribution of ranges and (b) shows the same for environment number.

The violin plots for range (Fig. 4.3a) show how well the network learned to predict IVAR-to-SUS ranges using the simulated training dataset. The median ranges (white dots) follow the same trend as the expected values. For several of the stations, the centers of the distributions are close to the expected values (S54, S41, S42, and S37), while the other centers are off by 1-2 km, with closer ranges (S60 and S40) being overpredicted and lonfer ranges (S36 and S44) being underpredicted. These results show that network is learning how to predict ranges, though there is certainly room for improvement.

The violin plots for seabed type (Fig. 4.3b), reflect the trend seen in the Fig. 4.2. For the two closest stations (Stations S54 and S60 at approximately 2.4 and 3.3 km), the network never predicts a seabed below three. The next two stations (S41 and S40 at 5.3 and 5.2 km, as well as some results

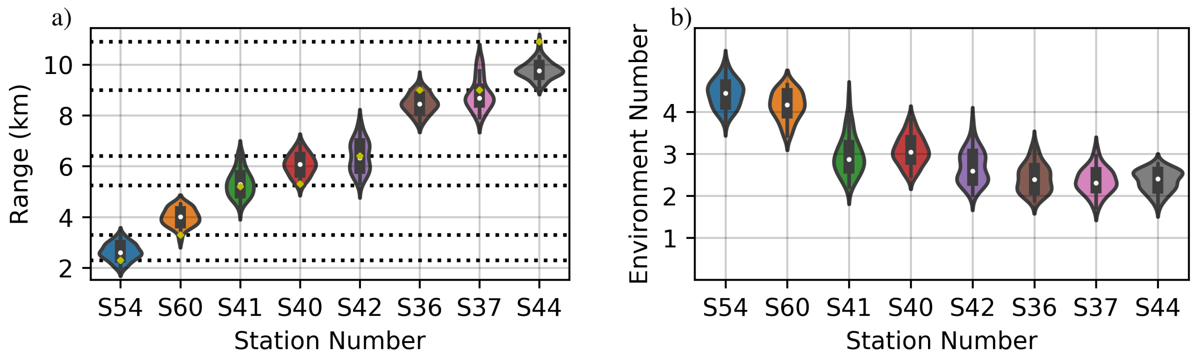


Figure 4.3 (color online) Violin plots (a combination of probability density and box plots) showing the distributions of the predictions a CNN made on IVAR data samples for a) range and b) seabed type separated by SUS station number. (A map of the stations is shown in Fig. 3 of Ref. [1].) The yellow diamonds show the expected values, with the horizontal dotted lines highlighting the true range. The seabed types are numbered as 1: deep mud, 2: mud over sand, 3: sandy silt, and 4: sandy.

from S42 at 6.2 km) also predict seabeds closer to three than the expected two. However, the results for the remaining stations show the majority of their distributions approach the "mud over sand" environment with some extension to the "sandy silt" environment. In addition, the distribution of the seabed type predictions becomes narrower as range increases. These plots suggest that the network learns how to predict the seabed type, but some additional information needs to be included in the training dataset to increase the efficacy of the network at close ranges.

4.4 Conclusion

This paper has shown that a CNN can be trained on simulated data to make seabed type and range predictions simultaneously on real-world data. As IVAR-to-SUS range increases, the mean distribution of range predictions tends to be underpredicted as the environmental predictions tend towards the correct seabed. At ranges less than 5 km, a more correct answer for range tends to be coupled with a prediction of a more reflective seabed type. Although predictions from this network are not perfect, these results show the potential for machine learning models to make range and seabed predictions simultaneously if the training dataset represents the variability of the real world data. The violin plots shown here also provide one way of tracking uncertainty due to random initialization of the network.

Future work will seek to improve the results by refining the simulated dataset to include more variation in environment and range to allow for further generalization. For example, only downward refracting sound speeds were used in this study, though some measurements also show that at some locations an isovelocity profile was observed. This variation was not included in these simulations, so a larger dataset that takes varying ocean sound speeds could prove beneficial. As with all applications of machine learning, there is also the possibility of developing deeper and more advanced networks to improve results.

Chapter 5

Study: Exploration of Different Training Datasets on CNN Predictions

This study continues the work done by training CNNs with simulated SUS data and then making predictions on measured SUS data. In this particular study, more attention is given to developing a proper training dataset that accounts for variations in the measured data. The contents of this chapter are being prepared for publishing in *IEEE Journal of Oceanic Engineering* with the tentative title "A CNN for impulsive time series: Training data considerations." The introduction (Sec. 5.1) of this work has been reduced for inclusion in this thesis as Chapter 1 contains a more exhaustive literature review.

5.1 Introduction

This study seeks to use a CNN to simultaneously identify an ocean seabed type and predict source-to-receiver ranges on measured data while only training on simulated data. This synthetic training data consists of one-second pressure time-series from an signal underwater sound (SUS) charge. Networks are trained and validated on these synthetic signals and then tested on signals recorded during the Seabed Characterization Experiment 2017. The impact of the training data on the seabed type and range predictions is explored and yields several important lessons for future training of deep learning algorithms in ocean acoustics.

5.2 Background and Methodology

Because applications of deep learning models are vast and varied, a discussion on the specific steps and decisions made for this study is presented. The rationale behind the synthetic data generation process as well as the inspiration for the synthesized environments are discussed in Section 5.2.1, followed in Sec. 5.2.2 by a description of the choices made in preparing the simulated and measured data for training, validation, and testing. Section 5.2.3 discusses the software packages used for the deep learning algorithms as well as the network topology used for this study. Sections 5.2.4 and 5.3 address how the machine learning models were trained and tested.

5.2.1 Data Generation

The Sediment Characterization Experiment[1] (SBCEX) was conducted on the New England Mud Patch centered at $[40^\circ 28' \mathrm{N}, 70^\circ 35' \mathrm{W}]$ in spring of 2017. The Intensity Vector Autonomous Recorder (IVAR) was located in 74 m of water to measure SUS charges deployed at various locations between 2 km and 11 km in range during SBCEX. Though IVAR was equiped with multiple sensors, the 37 recorded pressure time-series data are used for testing the deep learning networks in this study. The design and location of this SUS charge experiment (in an environment with a mud over sand seabed[13]) inform the training data simulations.

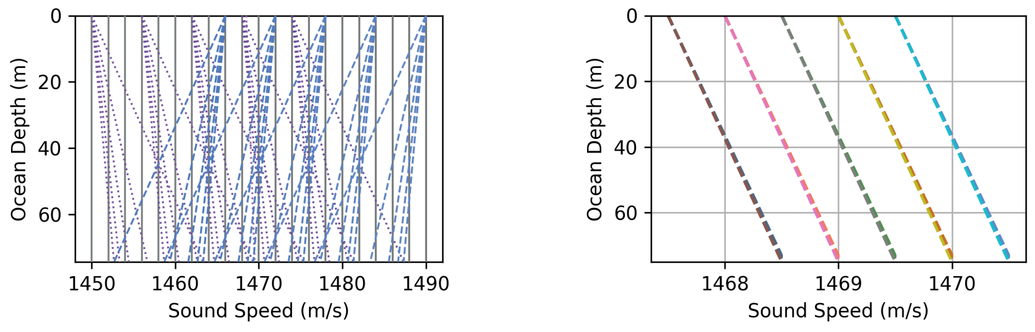
Machine learning models require a significant amount of data for training to be able to identify and learn patterns, and the 37 samples from IVAR are insufficient for training. To mitigate such a limitation, synthetic data samples were generated for training. Using ORCA[2] model, depth-dependent mode functions in range-independent environments are calculated through upward- and downward-reflecting plane wave reflections instead of solving parabolic equations (such as the range-dependent PE-RAM[4,5] software), resulting in a faster computation time. The calculated mode functions can then be used to generate the Green's function of the ocean for specified source and receiver locations. The range-independent ORCA model is sufficient for this case because, although the SBCEX seabed is not range independent, variations are gradual (See Fig. 3 in Ref.[1]). Additionally, the frequencies used in this experiment do not exceed 2,500 Hz, and the maximum range of the measured data is approximately 11 km. The frequency-dependent Green's functions are multiplied by the SUS charge spectral model and an inverse fast Fourier transform is performed to obtain a pressure time-series waveform.

The simulated SUS[36] charge model as proposed by Wilson *et al.*[35] was used to mimic the charges at SBCEX. The SUS charges were simulated at 30 discrete source-receiver ranges from 0.5 to 15 km at 0.5 km intervals. These ranges not only cover the measured ranges of 2-11 km, but also provide data outside that range. The depth of the SUS charge was recorded to be at 18.3 m in SBCEX, but to provide variability, the SUS charge depth was simulated to be between 10.3 and 24.3 m at 2 m intervals. The receiver was set to be at the measured IVAR depth of 1.32 m above the ocean floor.

The ocean response from ORCA depends on the selected seabed and sound speed profile. Although a myriad of possible seabed configurations exist, the goal is to test if a CNN can distinguish between a few seabed types. Thus, in this investigation, four different seabeds—representing deep mud, mud over sand, sandy silt, and sand—were used in the simulations. The "deep mud" environment (seabed 1) comes from a study done in the gulf of Mexico by Knobles *et al.*[11] The "mud over sand" environment (seabed 2) comes from maximum entropy statistical inference on data collected during the SUS circle experiments in SBCEX.[13] The "sandy silt" environment (seabed 3) comes from geoacoustic inversions done in the New England Bight by Potty *et al.*[52] The "sand" environment (seabed 4) comes from a study on sandy seabeds done by Zhou *et al.*[53] A visualization of these seabeds and values for sound speeds, densities, and attenuations is shown in Figure 2.1. The "mud over sand" seabed is the target seabed for the IVAR data as it comes directly from an inversion performed at SBCEX.[13] These four seabeds were chosen to be distinct and ordered from least to most reflective.

To investigate the impact of training data on the network, two different simulated datasets were used. The two datasets were generated with the different sets of sound speed profiles (SSPs) shown in Fig. 5.1. In the first simulated dataset (SIM1), a static ocean depth of 74.4 m was chosen to mimic an average depth of the ocean during the SBCEX experiment. The SIM1 signals were generated for each of the 72 SSPs in Fig. 5.1a. This collection of linear upward refracting, downward refracting, and isovelocity SSPs was selected to include a variety of propagation conditions The second simulated dataset (SIM2) was generated with featured 50 sound speed profiles all upward refracting with approximately the same slope as was measured during SBCEX 2017. The five unique SSPs in Fig. 5.1b were assigned ten depths between 73 and 75 m to simulate ocean depth variability. To test different variations, a third dataset was uses; the synthetic signals from SIM1 and SIM2 were combined to create a dataset denoted as SIM1+2 as a means for testing the impact of training the models with all variations explored. The resulting differences provide insights into considerations needed to generate a sufficiently representative training data.

This study seeks to predict between four seabed types and predict a range corresponding to the data. With the four different seabeds, 30 ranges, and eight source depths along with the varying ocean sound speed profiles, SIM1 had 69,120 signals and SIM2 had 48,000 signals. Although these training datasets are smaller than some machine learning projects, which use hundreds of thousands of samples in a single dataset, the sizes of these two datasets are considerably larger than the number of real-world test samples (37).



(a) SIM1 SSPs **(b)** SIM2 SSPs

Figure 5.1 (Color online.) The various sound speed profiles (SSPs) used to generate the SIM1 and SIM2 datasets. In the SIM1 set, 72 various upward (dashed lines), downward (dotted lines), and isovelocity (solid lines) profiles are used with a static ocean depth of 74.4 m. In the SIM2 set, only upward velocity profiles are used and the water depth varies from 73 and 75 m, resulting in 50 SSPs for SIM2.

5.2.2 Data Preparation

After data generation, the data is prepared for input into the machine learning model. The primary preparation step is data normalization. Normalization is important for efficient machine and deep learning, especially when the inputs are large numbers. When input values have a large values and, perhaps more importantly, large variance, a much longer time is required for the algorithm to learn. The machine learning community does not have a "one-size-fits-all" solution for normalization, so this study will use maximum normalization.

In this study, each synthetic signal was normalized by its own maximum. This normalization removes relative pressure information across the dataset. This normalization was done primarily to reduce training time as the weights would only need to learn how to handle small, unitless numbers instead of large values in Pa. This approach also mimics common techniques in image-based machine learning which normalizes pixel values to be between 0 and 1. However, this type of normalization incurs a potential for a loss of accuracy as all variation in maximum amplitude due to different propagation distances is lost as well. This hindrance further shows the power of the

CNN to localize sources and classify seabed types on measured data by making the problem more difficult to solve as the networks never see the true maximum value of the signals nor their relative loudness. Normalizing by a static pressure value across all signals could have satisfied the need for small inputs and preserved relative amplitudes, however, individual normalization was selected as a way to test the network and because the real-world data were normalized in the same way.

In addition to the normalization, the time duration of each sample was reduced to one second. After calculating a "direct" arrival time with the average sound speed in the ocean, the signals were extracted so that the direct arrival time was at about $t = 20\mu$s. This extraction is another hindrance for the network, as all absolute arrival time is removed through this process. The absolute arrival time, like the absolute amplitude of the signal, provides range information. However, the removal of travel time from the signals also supplies a more realistic example. As absolute time is often not known, new signals could be extracted from a recording and quickly aligned in this manner. Examples of the resulting normalized one-second pressure time series are shown in Fig. 1 in Ref X (submitted JASA-EL, will update with proper reference before submission). These synthetic signals form the training dataset.

5.2.3 Machine Learning Model and Software

For this study, a convolutional neural network (CNN) is used. A CNN uses the convolution operation where a learned "filter" is slid across filtered data to find patterns. A CNN eliminates the need for feature extraction preprocessing on the data as it finds its own "features" in the data. (More information on CNNs can be found in Goodfellow *et al.*[58]) The CNN architecture used in this study is a network with four convolutional layers and two fully connected layers (one hidden and one output) with a rectified linear unit (ReLU) activation function after each internal layer. A regression output trained via mean squared error is used for source range and seabed number. The exact specifications of the network architecture can be found and are explained in more detail in Van

Komen *et al.*[42]

The model was built, trained, and tested in the Python programming language. In particular, the PyTorch[62] open-source framework was used. PyTorch was written with a focus on usability and speed which makes it a great candidate for prototyping models quickly and efficiently in native Python while providing the tools to learn on GPUs. PyTorch includes a vast library of common mathematical operations used in neural networks, data processing tools, and learning algorithms.

To train the network, the stochastic optimization method called Adam[57] was used. As with the procedure detailed in Van Komen *et al.*,[42] a cosine annealing learning rate was used, though the method of warm restarts as detailed in the original paper by Loshchilov *et al.*[63] was not used as the current PyTorch framework opted to not include that implementation.

5.2.4 Training the Model

The PyTorch library was used for training (in particular, Python 3.6.9 with PyTorch version 1.4.0). To accelerate learning, an NVIDIA Tesla T4 was used during the learning phase. With a GPU, training time decreases by a factor of 5 when compared to an Intel Xeon CPU e5-2630 with two allocated processors (approximately 75 minutes to 15 minutes) on one of the datasets.

For the networks employed in this paper, a training period of 200 epochs was used for the SIM1 and SIM2 training datasets and 150 epochs for the SIM1+2 dataset (due to the increase in training data samples, this was done to reduce computation time). The fixed number of epochs was selected without early stopping to give each network the same amount of time to learn. Section 5.3.1 shows the results of self- and cross-validation during and after training. Each instance of training had a different random initialization of weights and used a starting learning rate of 0.001 and a batch size of 32.

5.3 Testing the Model

Simply training the model is not enough to know how well the model has learned. This section provides a discussion on the methods used to evaluate the model's performance and to provide a baseline for how the model learns on the different datasets used in this study. Section 5.3.1 discusses model validation which occurs using simulated datasets similar to the one used in training. Section 5.3.2 discusses model generalization through application of the trained networks to measured data.

Both the validation and generalization quantify performance with the same metrics. The first metric applied for range is root mean squared error, RMSE $= \sqrt{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2/n}$ with $\hat{y}_i$ being the prediction, y_i being the truth, and n being the number of samples. The second applied for range is mean absolute percentage error: MAPE $= \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)/y_i$. For seabed type, accuracy makes more sense than RMSE or MAPE because of the discretized nature of seabed representation being used. The accuracy is calculated by rounding the prediction to the nearest integer and determining if it equals the target number defined in Sec. 5.2.1.

5.3.1 Validation

One measure to ensure the model learns is through a validation dataset. A validation dataset generally comes from the same source as the training dataset or is a selection of the data not used in training and is useful for determining how a model behaves during training. If a model does not give satisfactory results on a validation dataset, there is little reason to believe that performance on another dataset will be better.

While training, learning was paused at certain intervals, and the network was given the validation dataset for making predictions and calculating loss values to monitor for potential overfitting. One clear sign of overfitting is when the loss on the training data continues to decrease while the loss on the validation dataset increases. This difference implies that the network is learning to model the training data too well and cannot accurately model other data. For the datasets in this work, both the training and validation loss values continually decreased across all training periods, indicating that that model was not overfitting during training.

k-fold Cross-Validation

The first method employed for network validation is the *k*-fold cross-validation method.[64] This method divides a dataset into *k* random blocks of near-equal size and then trains and tests *k* times. During each of these tests, the network is trained on $k-1$ blocks of the dataset excluding one block and then tested on the withheld block. The network is then reinitialized with random weights and trained again with the next set of blocks repeated $k-1$ times.

The *k*-fold cross-validation method was performed with $k = 4$ for SIM1 and SIM2 datasets. Networks trained on SIM1 predicted on average 99.89% of the seabeds correctly and obtained a RMSE of 0.05 km on source-receiver range predictions. Networks trained on SIM2 predicted on average 100% of the seabeds correctly and obtained a RMSE of 0.01 km on source-receiver range predictions. According to these results, the network can more accurately make predictions on holdouts from the SIM2 dataset when trained with data from SIM2. However, due to the differing physics, these results do not necessarily mean that one dataset is better than another, so further validation and testing is required.

Validation Across Datasets

Another validation method used involved training networks on one dataset and testing on the other dataset. To calculate these results, ten instances of the CNN were trained on a 95/5% split for each dataset. Multiple instances were selected to account for random initialization of network parameters and to get a statistical average of the network's potential. The results of these validation tests are outlined in Table 5.1 and indicate the important connection between the variablity in the training

Table 5.1 Validation results across multiple runs for the trained networks as defined in Sec. 5.3.1. The "Training" column refers to either "4-fold"(the *k*-fold cross validation method outlined in Sec. 5.3.1) in the dataset indicated in the "Testing" column or the training dataset that was then tested in the other dataset (detailed in Sec. 5.3.1).

Training	Testing	Seabed Accuracy	Range RMSE
4-fold	SIM1	99.89 %	0.0464 km
4-fold	SIM2	100 %	0.00993 km
SIM1 Trained	SIM2	99.99 %	0.185 km
SIM2 Trained	SIM1	74.31 %	1.92 km

and testing datasets. These results show that SIM2's collection of slightly upward refracting ocean sound speed profiles does not contain enough variability to account for the propagation through upward refracting and isovelocity profiles in SIM1 leading to mispredictions in the seabed and source-receiver range. An interesting result from this validation is that networks trained on the SIM1 dataset can make accurate predictions on the seabed and on average predict within 0.2 km on the range even though the ± 2 m variation in ocean depth in SIM2 were not included in SIM1.

With these validation results completed on the simulated datasets, the trained networks were then tested on real-world data to show how the networks generalize what was learned from simulations. The measured dataset is described in Sec. 5.3.2, and Sec. 5.4 shows the predictions of the trained networks on that measured data to provide a case study that models trained on simulated data can make reasonable predictions on real world data.

5.3.2 Generalization

To obtain generalization results, a dataset of measured SUS charge signals was used. The data was recorded in an area of the New England Mudpatch centered at [$40°28'$ N, $70°35'$ W] during the 2017 Seabed Characterization Experiment (SBCEX2017). The Intensity Vector Autonomous Recorder (IVAR) system deployed by the Applied Physics Laboratory, University of Washington recorded

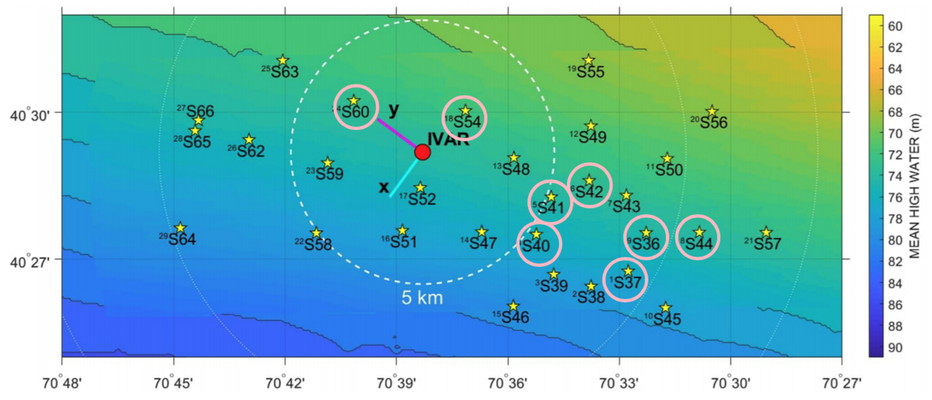


Figure 5.2 (Color online.) Map of SBCEX stations. The encircled station numbers indicate the stations included in the measured dataset. The large circle near the words "IVAR" shows the location of the IVAR system. The color map indicates the mean water depth at each position. This figure was originally published in Ref.[1]

the waveforms of small 31 kg SUS explosive charges deployed at ranges of 2-13 km. Figure 5.2 shows a map of the SBCEX experiment with circles around each of the stations selected for this study. Three to five SUS signals were collected at each station for a total of 37 signals.More details about the system, the recordings, and the experiment conducted can be found in Ref.[1]

While IVAR has multiple sensors for measuring both pressure and particle velocity, only signals from the pressure channel located 1.32 m above the seabed in used in this paper. As with the simulated training data, each sample of the IVAR data was normalized between -1 and 1. The results of inputting the 37 IVAR samples into the trained CNNs are presented in Sec. 5.4.

5.4 Results and Discussion

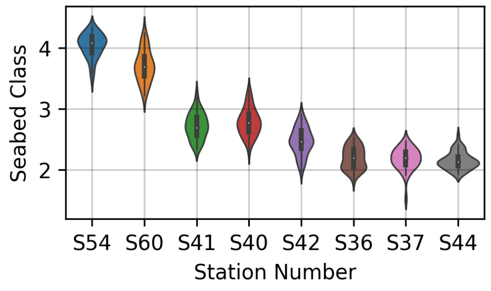
The networks trained on the synthetic datasets are applied to 37 measured IVAR signals taken from eight different ranges. For each synthetic dataset (SIM1, SIM2, SIM1+2), ten CNNs are trained for simultaneous seabed type and range predictions. The predictions for seabed type and range from the thirty different networks are presented in Secs. 5.4.1 and Secs. 5.4.2, respectively. For comparison, the networks were also configured and trained to predict either seabed or range in Sec. 5.4.3. The comparisons across the datasets yields insights for future network training.

5.4.1 Seabed Prediction Results

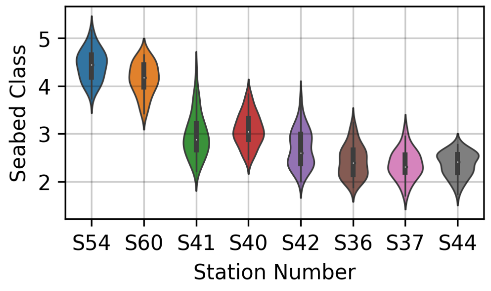
First, comparisons are made between the seabed predictions. The seabed type predictions across each station where the SUS charges were deployed are shown in Figure 5.3 as violin plots. A violin plot contains the median prediction (white dot), a box and whisker plot (in black) to show inner and outer quartile ranges, and a normalized distribution of the predictions (in color). The distribution shows the probability density of the predictions, smoothed by a kernel density estimator. The left-to-right order of the stations indicates increasing range. In the case of the IVAR data, the "expected" value should be two. ("Expected" is used here instead of "true" due to the many different parameterizations of the seabed from SBCEX.[35])

A clear trend is consistent across all three training datasets in Fig. 5.3.: the seabed type predictions approach seabed "two" for data from the farther the stations. For the first two stations (S54 and S60 at approx. 2.4 and 3.3 km, respectively), the seabed type predictions are much closer to four (the sandy seabed) and even approaching a "5th" seabed not included in training. The next two stations (S41 and S40 at 5.3 and 5.2 km, respectively), as well as some predictions from S42 (at 6.2 km) are closer to three (the sandy silt seabed). The stations beyond S42 (S36 at 9.0 km, S37 at 9.0 km, and S44 at 10.9 km) give predictions much closer to seabed type two in all cases with the SIM1 training providing the closest and tightest results.

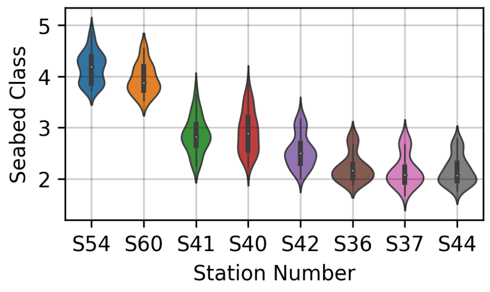
This range-dependent trend relates to the discretized representations of the seafloor selected for this work. At longer ranges, propagation is primarily influenced by upper portion of the seafloor, while the closer ranges retain more information about propagation effects due to the deeper portions of the seafloor. Thus, the "mud over sand" seabed parameterization more directly represents the effective seafloor at ranges greater than 6 km, and the tendency of the network to predict a more



(a) Networks Trained on SIM1



(b) Networks Trained on SIM2



(c) Networks Trained on SIM1+2

Figure 5.3 (Color online.) Seabed type predictions of 30 separate networks (ten for each dataset trained to predict seabed and range simultaneously) on the measured IVAR data taken at multiple stations. The horizontal axis shows the station numbers, which are ordered from nearest to farthest source-receiver range. The vertical axis is the seabed type number. These results are shown via violin plots, which show the median (white dot), box and whisker blot (in black), and a normalized distribution of the predictions (the "violins" in color).

reflective seafloor at the closer ranges indicates that deeper sediment properties are more reflective than seabed type 2.

While the violin plots provide lots of visual information about the CNN predictions and uncertainties, another way to quantify the performance is through seabed type accuracy and RMSE, as described in Sec. 5.3 Table 5.2 provides accuracy (relative to expected seabed type 2) and RMSE for the different networks trained with different simulated datasets in different modes. The results in "Multi" prediction mode are used in this section, while the "Single" prediction mode results are

Table 5.2 Seabed prediction errors and accuracy across training datasets and prediction modes. The "Single" prediction mode refers to networks that were trained to only predict on the seabed type while the "Multi" prediction mode refers to networks that were trained to simultaneously predict both range and seabed types. The "accuracy" is calculated by rounding each prediction to the nearest integer and comparing it to the expected value of 2 for the representative "mud over sand". Root mean square error (RMSE) helps quantify the repeatability of the predictions. The bold values indicate the "best" errors and accuracy.

Training	Mode	Accuracy	RMSE
SIM1	Single	80.45%	0.528
SIM1	Multi	49.19%	1.015
SIM2	Single	63.78%	0.833
SIM2	Multi	30.54%	1.305
SIM1+2	Single	**88.38%**	**0.376**
SIM1+2	Multi	42.70%	1.146

addressed in Sec. 5.4.3. From the rows labeled as "Multi" prediction mode, the networks trained on SIM1 have the have the highest seabed accuracy, with an accuracy of 49% (over the 31% and 43% accuracy of SIM2 and SIM1+2 respectively). In all cases, however, the seabed accuracy is improved when the network is trained only to predict seabed type, as discussed in Sec. 5.4.3.

5.4.2 Range Prediction Results

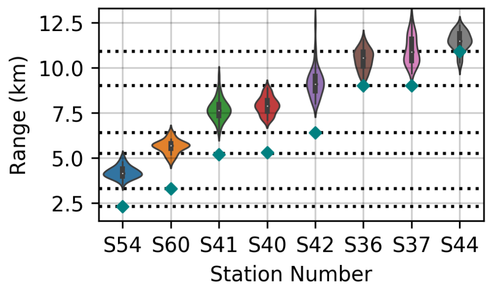
As with the seabed prediction results in Sec. 5.4.1, the range predictions are presented from the same thirty different networks (ten for each of the three datasets trained for simultaneous range and seabed predictions). Similar violin plots of the predictions sorted by station number are shown in Figure 5.4 so that the distribution of predictions can be easily seen for the hundreds of predictions presented. For these stations, the diamonds and dotted horizontal lines indicate the true source-receiver ranges for each station.

The training dataset makes a difference in the network predictions on range. The first observation from these results is that the networks trained on SIM1 (Fig. 5.4a) tend to overpredict the source-receiver ranges, though the overprediction tends to decrease as the source-receiver ranges decrease (S44 is much closer to the correct range than S54, for example). This overprediction is likely correlated with the network's tendency to predict a harder seabed for data from these stations. The networks trained on SIM2 tend to make predictions much closer to the true range values, and Table 5.3 confirms that assessment (with a RMSE of 0.742 km vs. 2.197 and 2.132 km for SIM1 and SIM1+2, respectively). However, the SIM2-based predictions tend towards underestimating the farther ranges (particularly S44) though not by the same amount of overprediction seen in the networks trained on SIM1 (at their worst). This improvement in training with SIM2 is possible evidence that accounting for various ocean depths and less SSP variability helps the network learn ranges more accurately.

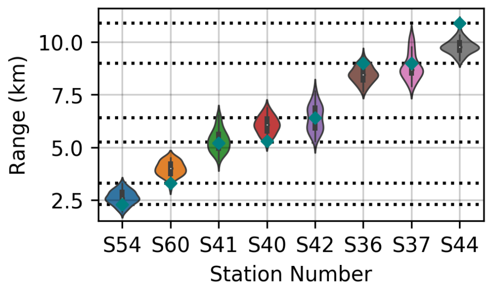
An unexpected observation comes from the networks trained on SIM1+2 (Fig. 5.4c) when compared to the other results (Figs. 5.4a and 5.4b). Deep learning models tend to be "data hungry," leading to a common thought that more data leads to better predictions. However, Table 5.3 shows that the metrics of the combined SIM1+2 dataset in the "Multi" mode has a performance between SIM1 and SIM2 rather than improved results. Perhaps the large number of variable SSPs and static ocean depths provided by SIM1 overwhelmed the learning, making proper range predictions more difficult. This evidence confirms that the quality of training data is an important factor in learning and that the ocean variability included in simulated training data should be representative of the real-world system that is being modeled.

5.4.3 Comparison to Individual Predictions

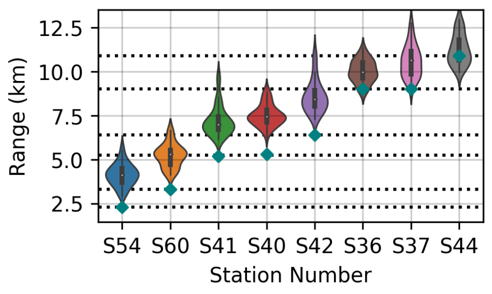
All of the previous tests were done with networks configured to simultaneously predict seabed and range. To further investigate the potential of the networks to learn from these synthetic datasets, another set of networks were trained to learn only one label: either range or seabed.



(a) Networks Trained on SIM1



(b) Networks Trained on SIM2



(c) Networks Trained on SIM1+2

Figure 5.4 (Color online.) Range predictions of 30 separate networks (ten for each dataset trained to predict seabed and range simultaneously) on the measured IVAR data taken at multiple stations. These plots are similar to those in Fig. 5.3. The diamonds and dashed lines represent the "true" source-receiver range at each station.

Though simultaneous prediction is more convenient, the thought behind this separation was that the networks could learn to focus specifically on features or patterns closely associated with the single specified label.

The results of the networks trained to only predict seabed are presented in Fig. 5.5 and some immediate differences are seen when compared to the results of the networks trained for simultaneous predictions (Fig. 5.3). The first difference is the tighter distributions of predictions in all cases, but especially for the networks trained on SIM1 (Fig. 5.5a) and SIM1+2 (Fig. 5.5c) where the seabed predictions for stations S41 and beyond are much closer to the expected seabed type 2. These tighter distributions appear to indicate the network learned something identifiable about

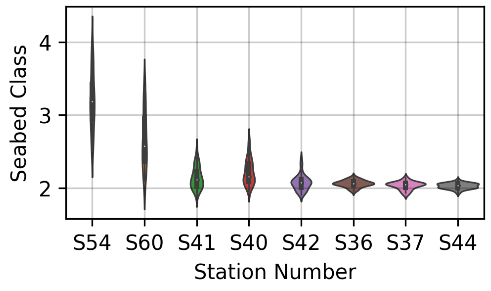
Table 5.3 Range prediction errors across training datasets and prediction modes. The "Single" prediction mode refers to networks that were trained to only predict on range while the "Multi" prediction mode refers to networks that were trained to predict on both range and seabed types. Two error values are presented here: root mean square error (RMSE) and mean absolute percent error (MAPE). The bold values indicate the "best" errors.

Training	Mode	RMSE	MAPE
SIM1	Single	2.256 km	42.55%
SIM1	Multi	2.197 km	41.34%
SIM2	Single	**0.704 km**	**10.41%**
SIM2	Multi	0.742 km	10.99%
SIM1+2	Single	1.965 km	37.57%
SIM1+2	Multi	2.132 km	35.42%

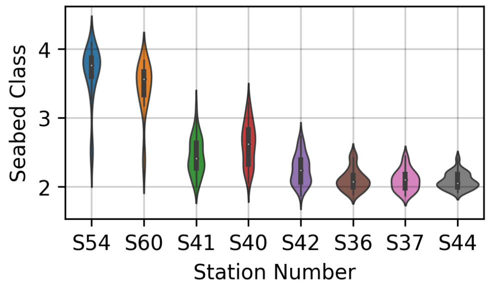
the seabed. For the closest range, the very extended distribution likely means that none of the four seabed types in the training data represent a reasonable effective seabed.

As confirmed by Table 5.2, cases of the "Single" (only seabed) predictions outperformed the "Multi" (both seabed and range) predictions. The accuracy of seabed predictions increases by approximately 30-45% across all training datasets. The RMSE also significantly decreases. Thus, these networks make more accurate predictions on seabed type when only asked to learn one label for this particular network design. As with the "Multi" predictions, SIM1 and SIM1+2 provide the most accurate seabed predictions, with SIM1+2's accuracy surpassing SIM1 by 8%. Thus, for the seabed-only predictions, the larger variability of SSPs in the training yields tighter distributions in seabed type predictions. This improvement in seabed prediction in "Single" mode could also be explained by the relatively small network size, as there may not be enough parameters for the network to learn how to predict both seabed type and range predictions simultaneously.

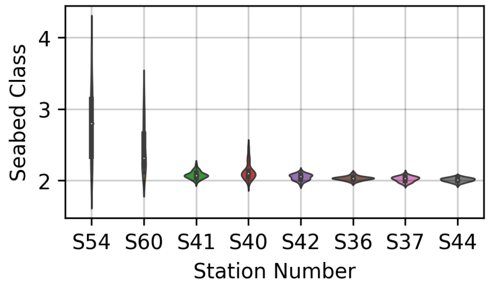
A similar comparison can be made with the range predictions. Distributions of results for the range-only predictions are similar to the the "Multi" regression results in Fig. 5.4. In Table 5.3, the



(a) Networks Trained on SIM1



(b) Networks Trained on SIM2



(c) Networks Trained on SIM1+2

Figure 5.5 (Color online.) Seabed predictions of 30 separate networks (ten for each dataset trained to predict only seabed type) on the measured IVAR data taken at multiple stations. These plots are similar to those in Fig. 5.3.

RMSE and MAPE are shown for the "Single" regression case. Results using SIM2 (identified as the most accurate in Sec. 5.4.2) show an improvement of 0.04 km in RMSE and 0.58% in MAPE. The nearly negligible improvement when the network only predicts the range is surprising as the the seabed predictions demonstrated an enormous improvement by comparison when switching to "Single" predictions.

A potential explanation for this small change could be in how the loss was implemented during training these networks. A summed MSE loss function (MSE was calculated for each label and then summed) was used in the "Multi" regression mode. Although the optimizer is attempting to get that loss value to zero in training, the scale of the learn-able labels possibly caused the network to focus on the label with the highest numbers. Each dataset has source-receiver ranges extending from $0.5 - 15$ km while the seabed types extend from $1 - 4$. A larger error in range would be more apparent in summed MSE than a similarly-scaled error in seabed type. Potential fixes for this could be to combine scaled errors for each label or scale the labels before training the network, both of which are beyond the scope of this study.

5.5 Conclusion

The results of this study provide evidence that convolutional neural networks (CNNs) trained on synthetic data have the potential to learn to make predictions on measured data, when care is taken regarding the training dataset. Specifically, one-second, normalized pressure time series were simulated for a variety of sound speed profiles, four seabed types, and a collection of source ranges and depths. The resulting simulated datasets were used to train CNNs for seabed type and range predictions. The trained CNNs were then applied to SUS charge signals measured during SBCEX 2017.

These results indicate that training on synthetic data is a reasonable start for making predictions on real-world measurements. Sufficient measured data to train a deep learning model would be expensive to collect and label, while simulated data only requires computer time. In this particular study, the sizes of the simulated training datasets were several orders of magnitude larger than the measured dataset on which final predictions were made.

This study has also yielded insights into considerations for generating simulated datasets to train neural networks to make seabed type and range predictions. Both the source-receiver parameters and the environmental parameters used in the simulations must account for the variability in the measured data. In particular, seabed type predictions in this work were better when significant variability was included in the water sound speed profiles used to generate the training dataset. Source-receiver range predictions, however, improved when ocean-depth variability was included.

This study highlights the importance of accounting for ocean variability in training machine and deep learning algorithms for marine applications.

This paper also contributes to the discussion of how to honestly evaluate the performance of machine and deep learning algorithms. Similar to many traditional optimization algorithms, most machine and deep learning networks provide an answer without an indication of the associated uncertainty. Care must be taken to account for the randomness associated with initializing the network weights and the training/validation split. In this paper, multiple instances of each network were executed on each training dataset, and the resulting predictions were displayed as a violin plot indicative of the statistical distribution of the results. Reporting the root mean squared error and mean absolute percentage over the results is also useful to quantify the prediction uncertainties.

Avenues exist for reducing the uncertainty of the seabed and range predictions. One potential investigation would be to use a deeper network, as the one used in this study consisted of only three convolutional layers and one hidden layer. A deeper network has the potential for more parameters to learn how to make better predictions. Likely a multi-regression approach to learning would also improve the results as the network could potentially allocate more parameters to learning individual labels. Another future step would be increasing the complexity of the training datasets by adding more source-receiver ranges, varying the ocean depth further while using the upward- and isovelocity sound speed profiles, and varying parameters in the seabed floor. One significant area of needed future research is to determine how to include more seabed types. Studies are needed to discern how to best choose from among the infinite number of possible seabed parameterizations to build a robust system for using deep learning in ocean acoustics.

Chapter 6

Preliminary Work Using Ship of Opportunity Spectrograms

This study shifts focus from impulsive SUS signals to cargo ship noise. The regular passage of ships in the ocean makes them a commonly occurring noise source, leading to the term "ships of opportunity" (SOO). The SOO noise is presented as spectrograms, which contain sound energy across time and frequency. At the time of writing, this work is in preparation for submission to the *Journal of the Acoustical Society of America*, Special Issue on Machine Learning, with the tentative title "Preliminary seabed classification and source parameter estimation using deep learning on broadband ship noise." The introduction (Sec. 6.1) of this work has been reduced for inclusion in this thesis as Chapter 1 contains a more exhaustive literature review.

6.1 Introduction

Successes of machine learning in underwater acoustics have prompted multiple questions about what can be learned from acoustical sources. A well-modeled broadband noise source exists in the form of ship of opportunity (SOO) noise.[41] Due to the machinery that propels a cargo ship forward, a constant broadband source of noise is radiated through the ocean. A model for the spectral shape of SOO noise was developed by Wales and Heitmeyer.[41] The interaction of the SOO noise with the ocean environment impacts the received noise. In addition to having a broadband source model, another thing that makes SOO an attractive sound source is that they tend to travel along straight tracks at a constant velocity (along shipping lanes), which simplifies the modeling procedure. Also, the AIS database[65] of all cargo ships gives sufficient information to estimate the actual range and speed corresponding to measured SOO spectrograms.

SOO sources have been used for seabed characterization in past studies. Battle *et al.*[66] applied near-matched-field processing for geoacoustic inversion to ship noise and noted that even quiet ships radiate enough noise for basic geoacoustic inversion. Nicholas *et al.*[67] used an L-shaped array to perform environmental inversion and matched-field tracking and noted that the vertical leg of the array provides environmental information and ship range-depth information while horizontal leg provides ship bearing information. Heaney[37] notes the challenges of predicting the full ocean response due to uniqueness of solution, sensitivity to mismatch, and variability with range and instead moves to prediction of average field levels and time spreads of sound in the ocean. Heaney then uses SOO noise to perform geoacoustic inversion of ship radiated noise from a single hydrophone in shallow water. Park *et al.*[38] used time-domain inversion techniques with time-reversal in a ray-based forward model to estimate geoacoustic parameters.

Other studies have expanded the geoacoustic inversion problem through waveguide invariant extraction and other statistical estimation techniques. Gervaise *et al.*[55] used inversion techniques on SOO noise to extract the waveguide invariant for geoacoustic inversion, though they concluded that attenuation cannot be estimated accurately with their inversion scheme and that density and sound speed estimation required injection of *a priori* information about the bottom. The same study notes that if ship range varies slowly enough, than effects from the Doppler shift can be ignored (see Eq. (2) in Ref. 55). Crocker *et al.*[68] measured SOO noise with an autonomous underwater vehicle and used Bayesian methods to obtain posterior probability distributions for the properties of a lower half-space. Byun *et al.*[69] used ray blind deconvolution to estimate channel impulse response to inform matched-field processing. Muzi *et al.*[39] used a vertical line array and time-averaged SOO noise to estimate passive bottom loss and noted that incorrect bottom loss estimation is a major source of error in SONAR performance prediction. Xu *et al.*[40] used joint time-frequency inversion for seabed parameters with ship noise recorded on a VLA.

SOO noise has also been used for source paramater estimation, such as source range. In a study done in the Gulf of Mexico, Koch and Knobles[70] and Stotts and Koch[71] used 15 min of noise from a moving ship measured on a horizontal line array to obtain simultaneous geoacoustic and source track parameter values. Gemba *et al.*[72] calculated channel impulse responses from 2.5 sec of SOO noise measured on a VLA to calculate arrival time. Niu *et al.*[20,73] and Ozanich *et al.*[25] recently used deep learning techniques for SOO localization.

This paper proposes to extend such research through a deep learning approach to see if SOO spectrograms can be used in convolutional neural networks (CNN) to make distinctions between different seabed types. Simultaneously, the models are trained to make source parameter estimation such as the ship's closest point of approach and speed. Three different multi-task CNN architectures are employed to investigate the impact of CNN depth and parameter count on generalizability. A comparison of input data type is also presented to decide if complex, magnitude or magnitude squared spectral values work best. Synthetic SOO spectrograms are used during training due to a lack of labeled field data and testing is performed on separate synthetic datasets and measured SOO noise. These tests show the potential for CNNs to distinguish seabed type and source speed and distance from SOO sources.

6.2 Background

The CNNs are trained on synthetic data, one measured SOO spectrogram was available for an initial test of the networks' generalizability. This measured spectrogram is from the Kalamata, and was recorded during the Seabed Characterization Experiment in 2017 (SBCEX2017)[54] on a vertical line array in the New England Mud-Patch. Though many sensors were present during the experiment,[54] the data used in this study were collected at a vertical line array (VLA) deployed by Marine Physical Laboratories/Scripps Institute of Oceanography (MPL/SIO). The Kalamata was recorded by their VLA2, which was located in the SE region of the experimental area of the study at [40.442N, 70.527W]. VLA2 consists of 16 hydrophones, with the lowest being 3 m from the ocean floor and a spacing of 3.75 m. For this preliminary study, only the recording from channel 8 (33 m from the ocean floor, in approximately 75 m of water) is used.

The Kalamata ship was found (from AIS data) to have a closest point of approach to the VLA of 3.29 km and was traveling at 19.9 kts. The measured Kalamata spectrogram from channel 8 of VLA 2 is shown in Fig. 6.1a. The spectrogram is calculated from an input signal sampled at 25,000 Hz with 50% overlap time-averaging with a time window size of $2^1 3$ samples. Due to low-frequency background noise, the 300-1500 Hz band is used for this work. The resulting spectrogram spans 15 minutes with the closest point of approach placed at 7.5 minutes, which centers the characteristic "U" shape of ship spectrograms. Though the ship noise was present in recordings longer than 15 minutes, the time limit was selected to reduce computation time and storage needs of the synthetic datasets (Sec. 6.3.1).

The authors realize that one data sample is not sufficient to test deep learning models. In an effort to address this concern, two synthetic datasets are generated for validation purposes and multiple model architectures are explored. Multiple models for each model architecture type are trained to provide a more statistically accurate estimation of model performance.

6.3 Methods

Supervised deep learning models require training datasets with a large number of labeled samples. Due to the scarcity and expense of real-world labeled ship of opportunity (SOO) spectrograms, synthetic data were used during the training and validation steps. Thousands of ship of opportunity spectrograms were simulated in four different seabed types with water profiles representative of the SBCEX2017 experiment for numerous ship speed and closest points of approach (CPA) to provide datasets for training and validation of CNNs. These trained CNNs are then applied to measured data as preliminary evidence of how SOO spectrograms can be used to obtain estimates of seabed type and source parameters. Details regarding these steps are provided in this section.

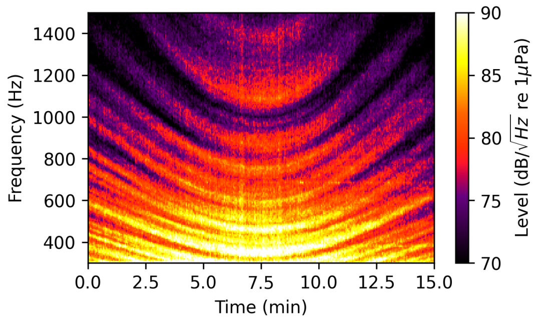
6.3.1 Synthetic Data

The simulations model received SOO noise using a modeled source spectrum and frequency-response of the ocean waveguide. The source spectrum of radiated ship noise is simulated using the ensemble source model described by the equation in Sec. 3 of Wales and Heitmeyer.[41] While this equation yields spectral levels in decibels at 1 m, these levels are converted to pressures (in Pa) and assigned a random phase. The ocean response is modeled with ORCA,[2] a range-independent, normal-mode model for acousto-elastic ocean environments. The ocean environment is simulated for different combinations of sound speed profiles (SSPs) and four seabed types. The four seabed types selected for this study are shown in Fig. 2.1: 1) deep mud,[11] 2) mud-over-sand,[13] 3) sandy-silt,[52] and 4).[53] The ocean SSPs are taken from measurements made during SBCEX2017 with random ocean depths selected between 73 and 78 m to account for changes in bathymetry across the area spanned by the experiment.

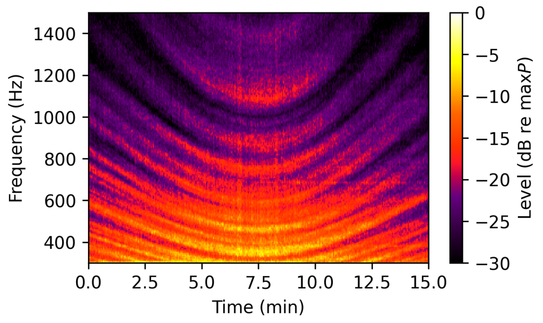
The SOO is assumed to travel in a straight line at a constant velocity. The SOO source is modeled with a quasi-static assumption corresponding to a point source at discrete locations throughout time, and no Doppler shift was accounted for due to the relatively slow speed of the SOO. The closest point of approach (CPA) is defined to be the point where the range between the ship and the hydrophone is at a minimum. For this study, the CPA assumed to occur at the center of the time array, t. Range positions over the entire interval can then be calculated based on time and velocity. The frequency array, f is is selected to generate the source spectrum of the ship through the Wales-Heitmeyer model.[41] The frequency-dependent Green's functions, calculated through the mode functions from ORCA and the range, are multiplied by the source spectrum to calculate the received complex spectrum, $\tilde{P}(f)$ at each time position. By iterating through each time position, the spectra are then combined to form the spectrogram, $\mathbf{P}$, with $\mathbf{P}_{ij} = \tilde{P}(f_i, t_j)$. This process is repeated for all selected combinations of seabed type, SSP, ship speed, CPA range, and effective ship source depth to generate the synthetic dataset.

The frequency and time dimensions of the generated spectrograms must match those of the final data being tested. In this study, decisions were guided by considering a prominent SOO event recorded by MPL/SIO VLA2: the passage of the Kalamata (details in Sec. 6.2). The measured Kalamata spectrogram is shown in Fig. 6.1a for the 300-1500 Hz band. When extracting the Kalamata spectrogram from the measured data, a significant amount of environmental noise was observed below 300 Hz. The characteristic "U" shape of SOO spectrograms was also clearest across 15 minutes as measured by the hydrophone of Channel 8 (33 m above the ocean floor) in the VLA. These observations, coupled with computational limits such as available computer memory and storage capacity, led to the simulation of the 300-1501.7 Hz band (with a Δf spacing of 6.1 Hz) and a 15 min span measured (with time steps intervals of 3 sec).

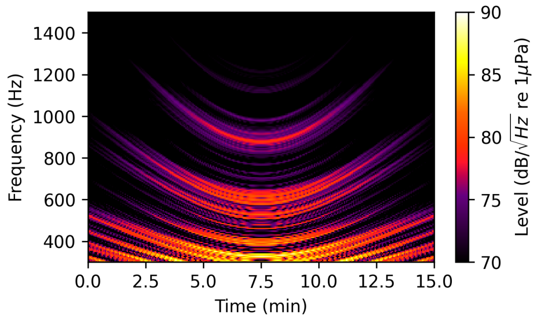
An example spectrogram generated for the mud-over-sand bottom at the same CPA and with the same speed as the measured ship is shown in Figure 6.1c for comparison with the measured spectrogram from the Kalamata ship (Fig. 6.1a). The difference between the measured and synthetic levels is caused by the source level of the Kalamata, which is unknown. To circumvent



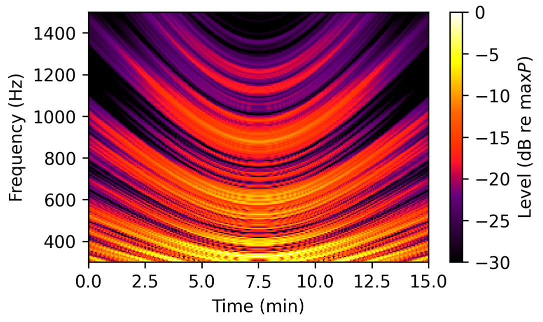
(a) Measured Kalamata Spectrogram



(b) Normalized Kalamata Spectrogram



(c) Synthetic SOO Spectrogram



(d) Normalized Synthetic SOO Spectrogram

Figure 6.1 (Color online) Example ship of opportunity spectrograms. (a) Absolute and (b) normalized spectrograms of the Kalamata measured on channel 8 of the MPL/SIO VLA2 during SBCEX 2017. (c) Absolute and (d) normalized synthetic spectrograms for a ship travelling 20.0 kts with a closest point of approach (CPA) of 3.3 km— similar to the Kalamata—, using the mud-over sand seabed type. The "normalized" spectrograms have the reference pressure be the maximum pressure of the spectrogram instead of 1μPa

that limitation, each spectrogram was normalized by its absolute maximum. The corresponding normalized spectrograms are shown in Figs. 6.1d and 6.1b. This normalization discards information about the range that is contained in the absolute levels, thus presenting an additional challenge for the deep learning models as the range information contained in relative and absolute amplitude is lost.

As only one recording of the Kalamata is avialable, synthetic data are used for training and validation. The synthetic data need to contain sufficient variability to account for the real-world

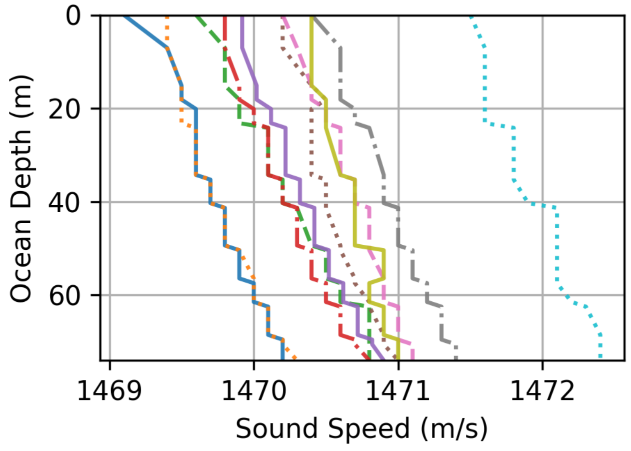


Figure 6.2 (Color online) Ten SSPs measured during SBCEX. To create the 50 training and 30 validation SSPs, five and three, respectively, random water depths were used with each SSP, .

conditions that exist in the measured data, as discussed in Van Komen *et al.*. To provide variability, the training dataset was generated with the following parameters. To represent the ocean, 50 different sound speed profiles (SSPs) were used in combination with the four designated seabeds. The SSPs were drawn from ten measurements made during SBCEX, as shown in Fig. 6.2 with each having five water depths randomly drawn over 75 m$\pm$0-3 m (while preserving the SSP gradient). The distribution of the randomly assigned ocean depths across the 50 SSPs is shown in Fig. 6.3(d). The 50 SSPs combined with each of the four seabeds creates 200 separate environments used for simulation.

For each of these 200 ocean environments, nine ship speeds were selected, nine CPAs were selected, and two ship depths were selected. The combinations of these source parameters and the 200 environments produced a dataset containing 32,400 (200 environments $\times$ 9 CPAs $\times$ 9 sounds speeds $\times$ 2 depths) samples for training. Previous internal studies showed that using randomly drawn source parameters improved the generalizability of the network as long as a sufficient number of samples were selected over the entire span of the labels of interest. To accomplish these goals, multiple set of the source parameters was used for each of the 200 environments. The nine CPA values for each environment included three fixed values used (3, 8, and 13 km) and six additional ranges randomly drawn between 0.5 and 15.5 km. The ship speed values included three fixed values (18, 20, and 22 kts) and six random values drawn between 16 and 24 kts. The two ship depths were randomly drawn to be between 6.5 and 9.5 m for each environment. The distributions of these source parameters over the 200 environments are shown in Fig. 6.3. The peaks in the distributions for CPA and ship speed correspond to the fixed values, selected to ensure that each of the 200 environments contained several of the same values as a means of consistency and for visual inspection.

To test the ability of the networks to interpolate between labels used in the training data samples and extrapolate beyond the edges of the training labels, two validation datasets were generated using the same procedure as the training dataset. The first validation dataset test the ability of the networks to interpolate. This first dataset (validation dataset 1) of 5,400 samples contains source parameter values similar to those encountered in the training dataset but with slight adjustments. Thirty SSPs were used (from the same ten measured SSPs but with only three random depths each). Four ship speeds were selected (18 and 22 kts for every environment and two randomly drawn between 16 and 24 kts), and nine CPAs were drawn (4, 9, and 14 km for every environment along with six random CPAs between 0.5 and 16 km). The remaining parameters were the same. The distribution of the water depths and source parameters used in validation dataset 1 are shown as dashed lines in Fig. 6.3. Validation dataset 1 measures the ability of the trained networks to interpolate.

The ability of the networks to perform at the edges of the training labels and extrapolate beyond the training labels are tested with validation dataset 2. For this second validation dataset, the same 30 SSPs were used as for validation dataset 1, but the CPAs (1, 2, 14, and 15 km for every environment with eight random CPAs between 0.5 and 16 km) and ship speeds (16 and 25 kts for every environment with two random CPAs between 14 and 26 kts) were chosen to be closer to the

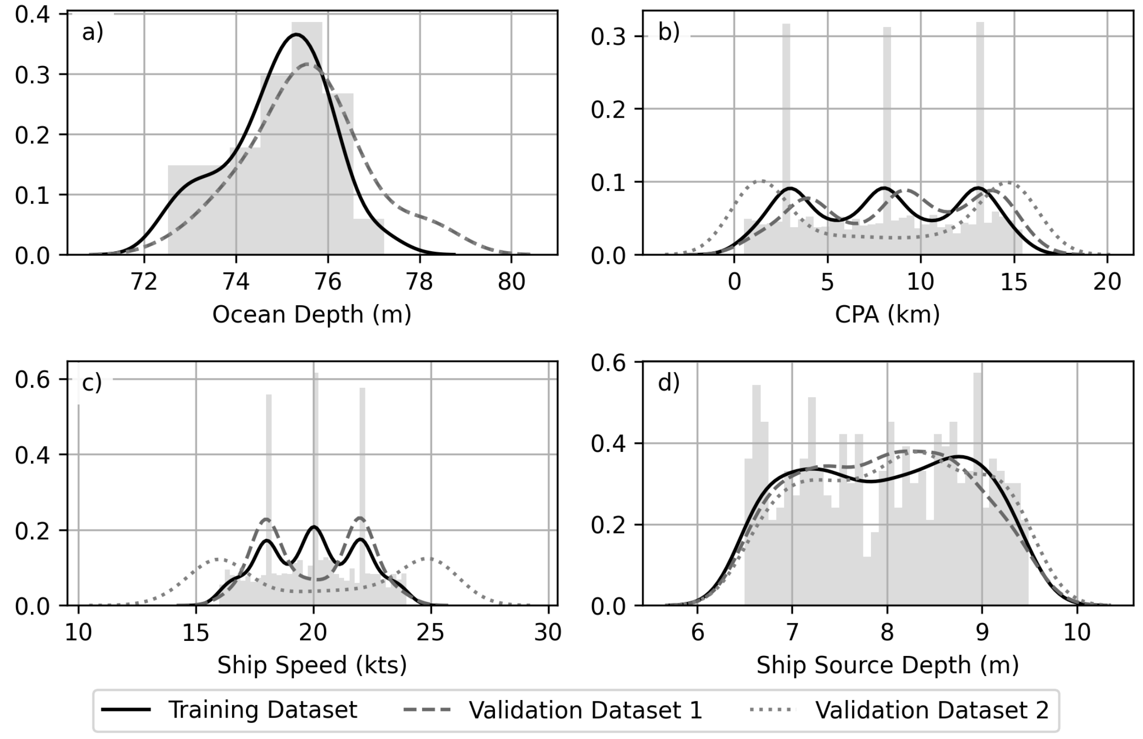


Figure 6.3 Normalized histograms (shaded gray areas) and distributions (solid lines) of the random parameters selected for the training dataset: (a) distribution of ocean depths over the 50 SSPs, (b) distribution of CPAs, (c) distribution of speeds, (d) distribution of source depths over the 200 environments. The large peaks correspond to values that were selected for each environment. The dashed and dotted lines represent the distributions of the various parameters for the two validation sets.

edges of the values used for the training dataset. Validation dataset 2 contains 8,640 samples. The distribution of the water depths and source parameters used in validation dataset 2 are shown as dotted lines in Fig. 6.3. Results of training and validation are discussed in Sec. 6.4.1.

Data and Label Preparation for Training

The manner in which SOO spectrograms should be used as input to the CNNs is studied in this paper. The measured SOO spectrograms come from a discrete fast Fourier transform on the received time-dependent signals resulting in complex values in the frequency domain. The synthetic SOO spectrograms are modeled in the frequency domain as complex numbers. This work evaluates if CNNs learn better on these complex values (real and imaginary parts) $\mathbf{P}$, the absolute value of those values $|\mathbf{P}|$, the squared pressure $|\mathbf{P}|^2$. For $\mathbf{P}$, the real and imaginary parts are split into two channels, where the real part is channel 0 and the imaginary part is channel 1. This is analogous to the red, blue, and green channels present in digital images. $|\mathbf{P}|$ and $|\mathbf{P}|^2$ are input as single channels, similar to the single black and white channel in digital images. As mentioned in Sec. 6.3.1, all samples are also divided by their absolute maximum before being passed to the network.

Another preparatory step was the normalization of the CPA and ship speed labels (a common practice with neural networks[74]). Instead of the networks learning their raw values in meters and knots, the networks learn scaled labels. Since CPA in the training set can vary between 0.5 and 15 km and ship speed can vary between 16 and 24 kts, this forces the values to be on the same scale, which has the potential to expedite learning and mitigate potential biases that due to the different scales. This scaling corresponds to taking the distributions of each label (Fig. 6.3) and scaling and shifting each distribution to lie between 0 and 1. For each set of labels, y, the scaled labels, $\hat{y}$, are found via

$$\hat{y}_i = (y_i - \min \mathbf{y})/(\max \mathbf{y} - \min \mathbf{y}), \tag{6.1}$$

where y_i correspond to the individual label of data sample i.

6.3.2 Convolutional Neural Network Topology

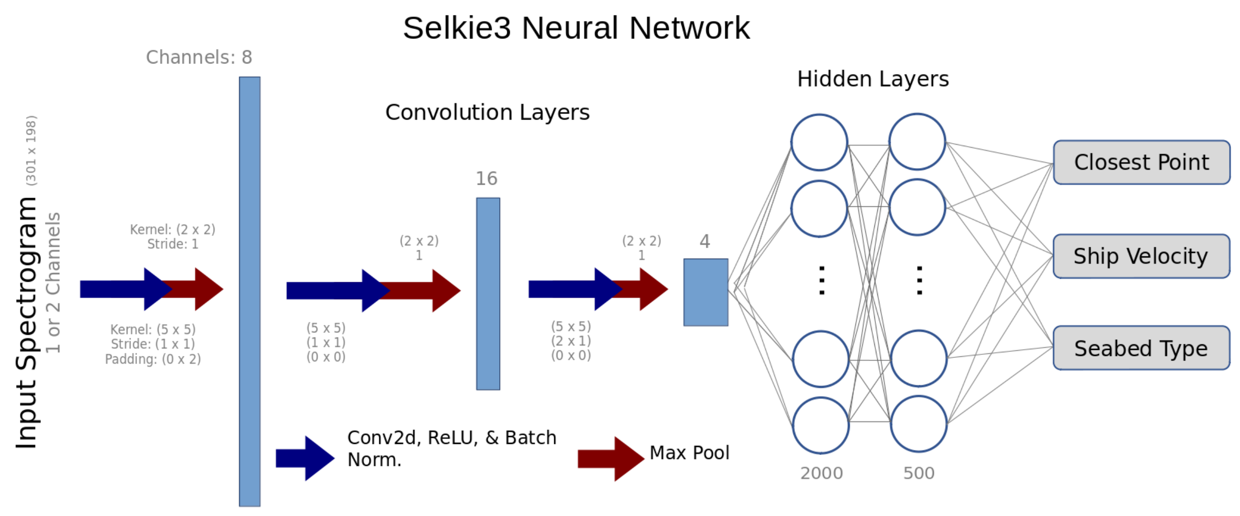
For this study, three different neural networks were selected to make predictions on seabed type, closest point of approach, and ship speed. The first two CNNs were hand-designed and incorporate different numbers of convolutional layers, with the full network architectures shown in Fig. 6.4. The third CNN is an implementation of "AlexNet" presented in Krizhevsky *et al.*[49] originally designed to classify between 1,000 different classes of images in the ImageNet[75] database.

Though various opinions on CNN design exist in the community, the two hand-designed CNNs were designed with simplicity in mind and to compare the effect of depth and number of parameters on predictions. The first CNN (Selkie3) has three convolutional layers with each followed by maximum pooling layers. These convolutional layers are followed by two hidden layers before reaching the output layer. The second CNN (Selkie5) differs by having five convolutional layers with no pooling after any of the layers. In both networks, the convolutional layers are followed by batch normalization. Selkie3 was designed to include pooling to reduce the number of features in each layer while using small filters to reduce the number of learnable parameters. Selkie5 uses more convolutional layers to give the model more learnable parameters and to let the stride parameters of the convolutions reduce the features within each layer, rather than allowing that reduction come directly from the pooling layers. Selkie5 also has larger hidden layers than Selkie3 because more parameters in the hidden layers can allow more complex linear transformations. Selkie3 has 4.3 million parameters while Selkie5 has 14.2 million parameters.

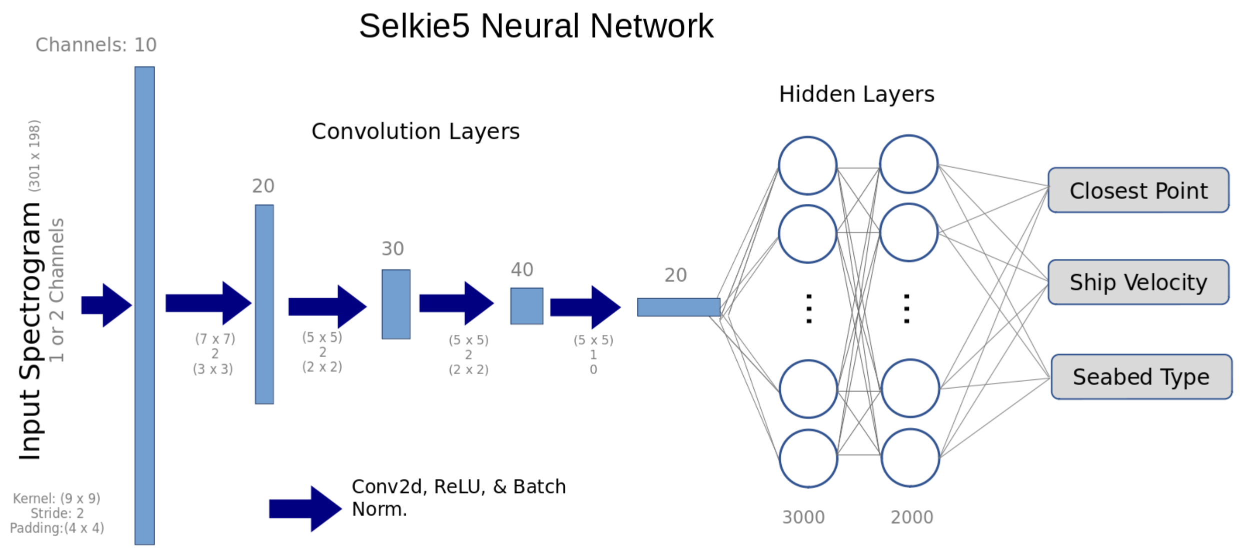
The implementation of "AlexNet" used in this paper comes from using half of the network size depicted in Figure 2 in Ref.[49] and is referred to as "HalfAlexNet" throughout the current study. Krizhevsky *et al.* mentions division of the network for simultaneous training across two GPUs, and one of those halves was implemented for this study. Even with just half of the original AlexNet there are over 15.6 million learnable parameters.

6.3.3 Training Convolutional Neural Networks

The implementation of the neural networks mentioned in Sec. 6.3.2 was done in Python 3 using the open-source PyTorch framework[62] (version 1.5.0). PyTorch was written using Python-native syntax with a focus on speed and usability and provides the tools required for loading data, building models, and training on a GPU using algorithms standard to the machine learning community. In particular, this study uses the PyTorch implementation of the Adam optimizer[57] for stochastic optimization during training. As with the procedure detailed in Van Komen *et al.*,[42] a cosine annealing learning



(a)



(b)

Figure 6.4 (Color online) Network topology for the two hand-designed CNN networks in this study. (a) Selkie3 features three convolutional layers each followed by a maxpooling step. (b) Selkie5 features five convolutional layers with no pooling steps. Each network also has two hidden layers before the output layer.

rate was used to limit the variable learning rate used by the Adam optimizer. (The method of warm restarts in the original paper by Loschilov *et al.*[63] was not used.) The annealing learning rate was implemented to prevent over-fitting

Because of the random initialization of the weights and splits of the dataset into batches, ten instances of each network type were trained for each type of input. Training multiple networks allows a statistical approach to evaluating the model's performance, as every new instance of a model is initialized with random values for all parameters. Each network was trained for 50 epochs. The networks were trained on an NVIDIA Tesla T4 GPU to accelerate learning.

Multi-task Learning

The final decision for CNN implementation is how to assign outputs for the final layer. The way networks learn to make predictions is through a loss function that determines how incorrect the predictions are compared to the truth. Through the selection of a loss function, these outputs can be trained to make different types of predictions such as regression or classification. Prior work using one-second pressure time series[42,43] and towed tonals used a regression approach to learning both the seabed type and source labels. Though some success was seen using regression to predict a seabed type, regression loss functions generally rely on a definition of distance between the truth and prediction. While differences in range, speed, or depth have a physical connection to distance, the choice to represent the seabed type as discrete numbers does not. The seabed types (used in Refs. 42, 43 and the current study) were ordered from highest to lowest bottom loss to provide a sequential progression, but the "distance" between seabeds 1 and 2 is not the same as the "distance" between seabed 2 and 3. Thus, in this work, the seabed type is found through classification while the ship speed and CPA range are found via regression.

To accomplish this combination of prediction types, the multi-task method proposed in Kendall *et al.*[76] was implemented which weighs losses for individual tasks by considering the homoscedastic uncertainty through learning the weighting. This multi-task method works by calculating an original loss value for each of the outputs of the network (mean squared error for regression and cross entropy for classification) and then scaling each loss by a learnable parameter. When building the network, the final output layer needs to include enough neurons to accomplish each of the tasks. For this particular study, six output neurons were used where the first four were reserved for classification on the seabed type and the last two were for CPA and ship speed. Though more details on this implementation of scaling loss can be found in Ref.,[76] we also implemented a variable transformation[1] of the learnable parameters σ_i via $\eta_i = 2\log\sigma_i$. This transformation changes Eq. (7) in Ref.[76] to

$$\sum_{n=1}^{i} \frac{1}{2\sigma_i}\mathscr{L}_i + \log\prod_{n=1}^{i}\sigma_i = \frac{1}{2}\sum_{n=1}^{i}(e^{-\eta_i}\mathscr{L}_i + \eta_i) \tag{6.2}$$

where $\mathscr{L}_i$ is the loss obtained for each task and i represents the task. This transformation avoids numerical instability as the original equation limits $\sigma \in \mathbf{R}_{>0}$ but Eq. 6.2 allows $\eta \in \mathbf{R}$.

Thus, during training, the network learns how to properly scale the loss values for each of the tasks. In this particular study, the initial η_i were selected to be 1.0, 0.5, and 0.8 for seabed type, CPA, and ship speed respectively. No exhaustive testing was performed on what the initial η_i should be. However, because these values are learnable parameters, the network adjusts them to minimize overall loss as with the model parameters so these η_i were sufficient.

6.4 Results and Discussion

The results of predictions from the trained neural networks are presented in this section. In particular, model validation and model generalization are presented. As explained in Neilsen *et al.*, there is a distinct difference between validation and generalization of CNN models especially in the context of synthetic and measured data. For the validation results, we present model predictions on synthetic SOO spectrograms with similar parameters as the training data. For the generalization results, we present model predictions on a measured SOO spectrogram. Metrics for model performance include

[1]This suggestion was given by Tony-Y on the PyTorch online forums

Table 6.1 Results from the k-fold cross validation tests where $k = 5$. The bolded numbers indicate the highest performing network for that metric.

Model	Seabed Accuracy	CPA RMSE	Speed RMSE
Selkie3	99.55%	0.33	**0.61**
Selkie5	98.98%	0.34	0.63
HalfAlexNet	**99.84%**	**0.2825**	1.04

root mean squared error (RMSE), mean absolute percent error (MAPE), and accuracy.

6.4.1 CNN Validation on Synthetic Data

Two methods of validation are presented in this section. The first is k-fold cross validation,[64] where the training dataset is divided into k random splits. The network trains on $k-1$ of those splits and tests on the remaining split. The network is then reinitialized and training is done on the next combination until all k blocks have been used for testing.

To validate that all three networks are learning relationships between the features of the training dataset and the six outputs, k-fold cross validation was done with $k = 5$ on the real and imaginary input data. The k-fold validation results, in Table 6.1, show that for seabed type predictions, HalfAlexNet has the highest accuracy, though Selkie3 and Selkie5 trail by less than a percentage point. For CPA predictions, HalfAlexNet has the lowest RMSE, though Selkie3 and Selkie5 differ by approximately 0.05 km. For ship speed, Selkie3 has the lowest RMSE at 0.61 kts while HalfAlexNet performs considerably worse. These results show that overall the networks perform similarly and are learning features that allow for estimations of seabed type, CPA, and ship speed, though more testing is required. These models trained through k-fold cross-validation are not used for further testing in this study.

The second validation involves testing the model on different separate datasets simulated in the same manner as the training dataset, but with different values. For these validation tests, the models were trained with all 32,400 samples in the training dataset, instead of a random subset as in the *k*-fold cross-validation. These fully trained models are then applied to the synthetic validation datasets (Sec. 6.3.1) the measured data (Sec. 6.2).

Ten instances of each model were trained with each input data type to account for uncertainties associated with the random initialization of the weights in the models. The results of the ten instances of each model applied to the 5,400 samples in validation dataset 1 are given in Table 6.2 as the average accuracy of the seabed classification and the RMSE and MAPE of the CPA and ship speed. The best scores across the various networks are in bold.

Validation dataset 1 has a similar distribution of ranges and speed, as shown in Fig. 6.3(b) and (c), but different the distribution of ocean depths was skewed slighted higher, as shown in Fig. 6.3(a). Application of the trained networks to validation dataset 1, thus, investigates of the performance of the networks in a presence of slight ocean depth mismatch. For validation dataset 1, all seabed predictions have an accuracy above 90%, which indicates the model is learning features required for seabed classification. When analyzing the individual results, it was noticed that the misclassifications across all networks generally were from distinguishing between the sandy-silt seabed and the sandy seabed. Looking at spectrograms for the same ship speed and CPA at the different seabeds for the selected frequencies (see Fig. 2.4), the sandy-silt and sandy seabeds appear nearly identical to the naked eye due to the similarities in propagation over the distances involved. However, this misclassification occured only in less than 500 of the 5,400 samples. The network that performed the best on seabed predictions was HalfAlexNet using $|\mathbf{P}|$ as input, though the $|\mathbf{P}|^2$ inputs performs less than 0.2% worse. However, Selkie3 and Selkie5 barely trail behind HalfAlexNet on seabed predictions. These validation results are significant because Selkie3 has around 10 million fewer parameters than Selkie5 and HalfAlexNet and performs similarly.

As for CPA predictions, HalfAlexNet performs the best once again, though none of the RMSE values in the other networks extend beyond 0.6 km, and the MAPE values fall between 3.22 and

Table 6.2 Results from ten instances of each model on validation dataset 1 containing 5,400 samples, which used similar speeds and CPAs to the training dataset, but had different ocean depths (see Fig. 6.3). The "input" column indicates the type of input data used for that set of tests. For all model and method combinations, ten networks were trained to produce these results and the accuracy, RMSE, and MAPE values come from 54,000 predictions (10 CNNs $\times$ 5,400 samples). The bolded numbers indicate the best value of the metric across all networks and methods.

Model	Input	Seabed Accuracy	CPA RMSE	CPA MAPE	Speed RMSE	Speed MAPE
Selkie3	$\mathbf{P}$	92.98%	0.60	5.20%	0.88	3.10%
	$\lvert\mathbf{P}\rvert$	99.11%	0.52	4.16%	0.73	2.47%
	$\lvert\mathbf{P}\rvert^2$	99.03%	0.50	3.99%	0.68	2.32%
Selkie5	$\mathbf{P}$	92.98%	0.60	5.33%	0.90	3.22%
	$\lvert\mathbf{P}\rvert$	99.18%	0.45	3.60%	0.64	2.10%
	$\lvert\mathbf{P}\rvert^2$	99.00%	0.46	3.72%	0.65	2.18%
HalfAlexNet	$\mathbf{P}$	91.95%	0.57	5.44%	0.97	3.56%
	$\lvert\mathbf{P}\rvert$	**99.70%**	0.36	3.24%	0.52	1.73%
	$\lvert\mathbf{P}\rvert^2$	99.56%	**0.36**	**3.22%**	**0.49**	**1.63%**

5.44% across the board with small deviances. The same is seen in speed metrics. HalfAlexNet once again performs the best when making these predictions, though for the other networks, the speed prediction RMSE values are less than 0.88 km and MAPE varies between 1.63% and 3.56%.

Validation dataset 2 has the same ocean depth mismatch as validation dataset 1, but the distribution of the CPAs and speeds are also different from the training dataset, as shown in Fig. 6.3. The average network performance on the 8,640 samples of validation dataset 2 is summarized in Table 6.3. In this case, the scores are generally worse, though that was expected due extreme nature of the speed and CPA selection used. The networks were never trained on speeds outside 16-24 kts, and few CPAs were included outside the 1-15 km range, while the networks are now making predictions on data simulated at or beyond those ranges. When operating on dataset 2, the networks are making predictions at the edges of and beyond the source parameters used in training. For these edge cases, the networks are still near or above 90% accurate on seabed classification. Once again, HalfAlexNet performs the best on seabed classification, though Selkie5 and Selkie3 still have accuracy greater

Table 6.3 Results from ten instances of each model on validation dataset 2 containing 8,640 samples, which had more speeds and CPAs near and beyond the boundary of those values in the training dataset (see Fig. 6.3 for parameter distributions). The "input" column indicates the type of input data for that set of tests. For all model and method combinations, ten networks were used to produce these results and the accuracy, RMSE, and MAPE values come from 86,400 predictions (10 CNNs $\times$ 5,400 samples). The bolded numbers indicate the best value of the metric across all networks and methods.

Model	Method	Seabed Accuracy	CPA RMSE	CPA MAPE	Speed RMSE	Speed MAPE
Selkie3	$\mathbf{P}$	89.63%	0.80	13.21%	1.50	5.54%
	$\lvert\mathbf{P}\rvert$	97.49%	0.72	11.35%	1.28	4.55%
	$\lvert\mathbf{P}\rvert^2$	97.24%	0.73	10.90%	1.24	4.45%
Selkie5	$\mathbf{P}$	92.88%	0.78	13.38%	1.45	5.35%
	$\lvert\mathbf{P}\rvert$	98.03%	0.60	9.12%	1.03	2.10%
	$\lvert\mathbf{P}\rvert^2$	97.45%	0.65	10.13%	1.06	2.18%
HalfAlexNet	$\mathbf{P}$	91.68%	0.74	13.15%	1.55	6.10%
	$\lvert\mathbf{P}\rvert$	**98.47%**	**0.54**	8.57%	**0.93**	3.26%
	$\lvert\mathbf{P}\rvert^2$	97.83%	0.54	**8.05%**	0.93	**3.22%**

than 97% for these edge cases.

The biggest increase in error between the two validation datasets is in the CPA and speed predictions, as expected since many of them extend beyond the ranges used during training. When looking at individual network predictions, the speed and CPA ranges tend to be underpredicted at the largest ranges and speeds, while they are generally overpredicted at the smallest ranges and speeds. In part, these trends can be be attributed to the label normalization process mentioned in Sec. 6.3.1. Since the smallest speed in the training datset, for example, was 16 kts, that value was normalized to 0 while the largest speed of 24 kts was normalized to 1. Any true values less than 16 would then have their normalzed value as less than 0, which the network never needed to predict during training. The same holds for values above the maximum. This label normalization is beneficial for training and testing on datasets within the bounds of the training set, but potentially becomes a detriment for datasets that contain samples beyond those bounds. However, it is currently unknown if that is the only reason why this occurs and requires further investigation. For the purposes of this study, the label normalization suffices as the measured data falls within the bounds of the training dataset.

An interesting observation from both validation datasets is how the different data input methods produce different results. In general, the differences in results based on input type is larger than differences due to the network type. When using the complex **P** matrix, the models have the lowest seabed accuracy across all networks. The prediction results when using $|\mathbf{P}|$ or $|\mathbf{P}|^2$ are close for all network types, but $|\mathbf{P}|$ does slightly better than $|\mathbf{P}|^2$ on seabed predictions while the opposite is true for CPA and speed. This difference could be due to the squaring of the $|P|$ causing larger differences between features that the networks couldn't learn in other input method. These validation tests emphasize the importance of input data type and have shown the performance of the networks in the presence of slight ocean depth mismatch and for source labels near and beyond the edges included in the training data.

6.4.2 CNN Generalization on Measured Data

After testing the networks on the validation data, the networks were applied to the spectrogram from the Kalamata (described in Sec. 6.2). AIS data for the Kalamata during this time indicates an approximate CPA of 3.3 km and speed of 19.9 kts. The expected seabed prediction is the mud-over-sand seabed, as it was obtained from a maximum entropy bayesian approach for the SBCEX2017 area[13] or the deep mud, which has similar surficial properties including an angle of intromission. For the $|\mathbf{P}|$ and $|\mathbf{P}|^2$ inputs, the Kalamata spectrograms were obtained via time-averaging while **P** was not. Table 6.4 shows the full results of the predictions when the values mentioned are considered the true labels for the signal.

As for CPA and speed predictions, the best predictions come from the Selkie3 model, which is surprising due to its worse performance on the validation datasets than Selkie5 and HalfAlexNet (Sec. 6.4.1), though Selkie5 gives a lower speed MAPE. To further investigate these differences, Fig. 6.5 shows the individual prediction distributions of each network. Figs. 6.5b and 6.5a show

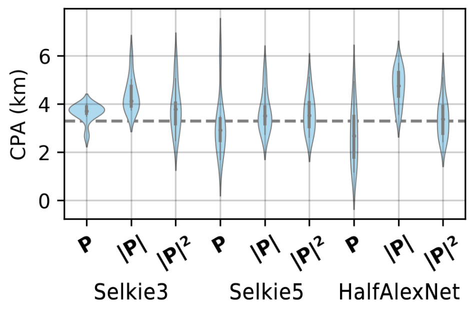
Table 6.4 Results from testing on the Kalamata sample. The RMSE and MAPE are calculated from one sample across ten networks.

Model	Method	CPA RMSE	CPA MAPE	Speed RMSE	Speed MAPE
Selkie3	$\mathbf{P}$	**0.50**	**14.41%**	**0.90**	4.08%
	$\|\mathbf{P}\|$	1.28	33.21%	1.45	6.01%
	$\|\mathbf{P}\|^2$	1.07	25.21%	1.20	5.25%
Selkie5	$\mathbf{P}$	1.15	24.94%	0.97	**3.85%**
	$\|\mathbf{P}\|$	0.86	18.53%	1.44	5.98%
	$\|\mathbf{P}\|^2$	0.81	19.66%	1.71	7.28%
HalfAlexNet	$\mathbf{P}$	1.29	32.79%	2.14	9.75%
	$\|\mathbf{P}\|$	1.58	42.85%	2.33	9.46%
	$\|\mathbf{P}\|^2$	0.79	19.83%	2.59	12.17%

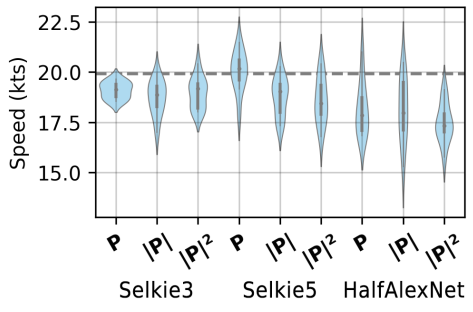
violin plots (a combination of normalized distributions and box-and-whisker plots) of CPA and speed predictions. It is clear that Selkie3 trained on the **P** input data has a tighter distribution for speed and CPA than the other networks.

Even though some of the medians of other networks are closer to the truth (the dashed horizontal line), the tighter distribution shows more confidence and repeatability from Selkie3. This could be an example of the 14+ million paramaters present in Selkie5 and HalfAlexNet leading to overfitting on the **P** synthetic data. Though they perform the best in the validation results (Sec. 6.4.1) drawn from the same statistical distribution as the training data, they do not generalize as well as Selkie3 on measured data. This serves as a caution for using the deepest networks presented in deep learning literature, as for these types of problems there could be a tendency of overfitting.

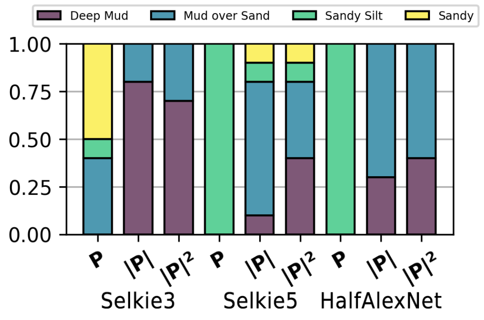
The seabed type classification results are depicted differently. Figure 6.5c shows a set of stacked barcharts of the predictions for the seabed class. These barcharts should be evaluated by rememberring that the key propagation feature for the frequencies used (300-1500 Hz) is likely the angle of intromission, due to the speed of sound at the top of the seabed being less than the speed of sound of the water at the bottom of the ocean. With this in mind, the combination of deep mud and mud-over-sand predictions given are reasonable. From this figure, it is clear that the noise in the



(a) CPA Predictions



(b) Speed Predictions



(c) Seabed Predictions

Figure 6.5 (Color online) Prediction results from the three models and the three input data types. (a)Violin plots (a normalized probability distribution kernel with the median and quartile ranges) of the CPA predictions. (b) Violin plots of the speed predictions. (c) Stacked barchart showing the percentage of predictions for each seabed type.

complex **P** inputs increases the difficulty in getting an accurate seabed prediction. However, the time-averaged $|\mathbf{P}|$ and $|\mathbf{P}|^2$ inputs identify a seabed type with an angle of intromission 100% of the time for Selkie3 and HalfAlexNet and 80% of the time for Selkie5. For seabed type predictions on the Kalamata measured data, **P** is not a reasonable input type. However, further testing on more measured data is required to conclusively determine which data input type is the correct choice for these predictions in all situations.

In Sec. 6.3.1, it was mentioned that the true source level of the Kalamata ship was unknown (see Fig. 6.1 for a comparison). Due to this unknown source parameter, the simulated levels do not match the measured levels. The required normalization used to account for this difference also discarded information about range which could have impacted the results seen here. One idea for future implementation is to include a source level parameter as a label for the CNN to learn. In addition, the measured spectrogram contains significantly more noise than the sumulated spectrograms, which could be a source of the prediction errors. The addition of random noise during training on the synthetic data could potentially teach the network to ignore such noise.

6.5 Conclusion

The results of this study provide evidence that CNNs trained on synthetic ship-of-opportunity (SOO) spectrograms have the potential to make predictions on seabed type and source speed and closest point of approach. Specifically, 15 min SOO spectrograms spanning 300-1500 Hz were simulated with four seabed types, ten measured sound speed profiles with different ocean depths, and a variety of source speeds and CPAs on a single hydrophone. The synthetic training dataset was used to train CNNs to make these predictions. The trained CNNs were then applied to additional synthetic testing datasets and one measured spectrogram from the ship Kalamata.

The results indicate that SOO spectrograms contains extractable features and patterns that CNNs can identify to make predictions. Though the networks struggled identifying differences between signals with a deep mud and mud over sand seabed or a sandy silt and sandy seabed at times, results were reasonable across the different tests. The networks even performed well on synthetic data near the edges of the training distribution regardless of the data input method.

The results on the measured data also provide evidence that the networks can learn from synthetic data and make appropriate predictions on measured data. Though only one sample was used, the ten instances of each dataset type provided insights that a smaller network better generalizes to the nosier measured data. The smallest network was able to consistently provide good CPA and speed predictions while also providing reasonable seabed predictions. However, the input data type impacted the seabed type predictions. When using the real and imaginary part of the complex pressure, **P**, the networks chose a more reflective environments no matter the size of the network, though the smallest network did make a few correct predictions. With both $|\mathbf{P}|$ and $|\mathbf{P}|^2$ as inputs, the networks chose the deep mud and mud over sand environments for the measured data, which is reasonable due to the angle of intromission present in these seabeds.

Future work can expand on these findings by including noise in the training data. Even with the time-averaged measured data, background noise is still visible noise. Introducing such noise into the training data could help the network learn to ignore noise in the input data. Another improvement would be using additional measured SOO spectrograms for testing. This study was limited by only allowing a single receiver depth of 33 m, so further use of data recorded during SBCEX 2017 would be to use additional receiver depths. Another future topic is the inclusion of more seabed types so that networks could begin to learn how to predict seabed parameterizations instead of just selecting between distinct seabed types.

Chapter 7

Conclusions

Throughout the studies presented in this thesis, multiple lessons have been learned regarding using deep learning to predict seabed type and source parameters. Chapter 3 demonstrated that CNNs have the potential to make such predictions and perform better than FNNs on the raw data. Chapter 4 applied networks trained on simulated data to measured data and found that good predictions can be made when the simulated training data represents the measured source and environment. Chapter 5 took that finding a step further to show that adequate sampling of the search space needs to be included in the training data due to differences between real-world data and simulated data. Chapter 6 showed the potential of using spectrograms of ship noise to make such predictions and extended the training data simulation by including random source parameters to more adequately sample the search space. Each study in this thesis has presented conclusions regarding their specific findings. (See Secs. 3.4, 4.4, 5.5, and 6.5.)

One general finding from the research shown in this thesis is the power of CNNs to minimize preprocessing. In all of the studies shown, minimal preprocessing was applied to the input data. In the case of the SUS charges, they remained in the time-domain and were only normalized by their maximum. No component analysis, frequency-domain transforms, or other signal processing was done before the signals were input to the CNNs. The SOO spectrograms, which were only normalized by their maximum as well, came from a FFT on the received signals. The use of pressure time series and spectrograms removes the need for preprocessing techniques, which are generally time consuming and computationally expensive when performed on large datasets. The reason preprocessing is not necessary is that CNNs have the potential to extract features necessary for predictions directly from time-domain waveforms or spectrograms.

Another important finding throughout this research had to do with the generation of simulated training data. Intermediate testing and the results seen in Chapters 5 and 6, showed that randomly sampling source parameters was more effective than evenly spacing the values (e.g., 0.5 - 15 km at 0.5 km intervals). This randomness potentially teaches the networks to not focus on hitting those exact intervals but instead how to best regress in a more generalizable fashion. More research is required to determine exactly what type of distributions are best for each source parameter, but some randomness improves generalizability.

This thesis has also provided insight regarding the use of output layers in CNNs to make predictions. Regression and classification approaches were both used for prediction, and different labels do better with each one. Regression makes sense and works well for labels that are smoothly varying quantities, such as source-receiver range or ship speed. Those quantities are known from the labels, and the distance between one and two meters is the same as two and three meters, which is the principle undelying regression. However, for labeling seabed type, no continuously varying value contains the same consistent definition of the distance between seabeds. Though the four canonical seabed types were sorted by by the overall amount of bottom loss, the difference between deep mud (1) and mud over sand (2) was not necessarily the same as between mud over sand (2) and sandy silt (3). Initial attempts to apply regressing to the seabed type label gave reasonable results (See Secs. 4.4 and 5.5.), but a classification approach is more appropriate (Sec. 6.5).

To combine different types of outputs, such as regression and classification, a multi-task approach was used (Sec. 6.3.3). Multi-task learning allowed the loss functions (mean squared error for regression and cross entropy loss for classification) to be combined and scaled in a learnable way. Success was seen with the multi-task method for simultaneously predicting seabed type, source range, and ship speed. When regression was used instead of multi-task (as done in the SUS studies, Sec. 5), predictions from networks trained on just one parameter performed better than models that had to perform multiple predictions (Secs. 5.4.1 and 5.4.2), though that could be attributed to the network size, dataset size, or other factors and requires further research.

Network size and structure is also a factor to consider when using CNNs to make predictions. In the case of the SUS charges as input data, a network with few layers was used (Sec. 4.2.3). This network is tiny compared to some of the models found in literature (such as AlexNet,[49] VGGNet,[50] and GoogLeNet[51]), but the results were very promising. Three different network sizes were used on the SOO noise study in Chapter 6 (Sec. 6.3.2), and the network with four million parameters performed comparably to the networks with over 14 million parameters and appears to generalize better. These results illustrate that large networks do not always guarantee better results than smaller networks when making predictions on the SUS charges or SOO noise. Perhaps more complicated tasks in the future that attempt to predict many individual prameters in the sediment layers will require deeper and more sophisticated networks, but the small networks worked remarkably well for these studies.

7.1 Future Work

This thesis has shown the potential for deep learning in underwater applications, and future work using machine deep projects is very promising. Current projects in Dr. Neilsen's research group are investigating the use of more complicated and advanced network structures, such residual networks, recurrent networks, echo-state networks, and attention-based networks. These different models have the advantage of incorporating the context available through the time dependent inputs to inform predictions. Letting the models incorporate that context could remove the need for the input data to be aligned (such as the closest point of approach being in the center, or the impact of received SUS charges being a few milliseconds past zero).

Other machine learning methods beyond deep learning are also important for this research. The use of support vector machines in creating a representative set of ocean sound speed profiles is being researched to identify how to condense days worth of measurements into a reduced basis set that can be used for training data simulation. Clustering algorithms could also be used as a visualization method or to evaluate methods for dividing data, such as in classifying signals. Ensemble learning could also provide a clearer look at the uncertainty of predictions. This list will continue to grow as the machine learning and deep learning fields are vast, with new techniques being published regularly.

Exploration of input data is also an important research topic for these applications. Questions regarding the use of multiple hydrophones (in a horizontal or vertical line array, for example) as extra input channels could provide more localization information similar to the multi-receiver beamforming approach used in acoustic signal processing. The question regarding whether networks perform better on frequency-domain spectrograms or time-domain pressue signals for the same source should be addressed. Other decisions about the optimal input data sampling frequency, frequency limits and spacing, length of signals, etc.) also need to be investigated. Further exploration into the distributions of the source parameters and ocean parameters to be included in simulating the synthetic training data is also an active topic of research. The use of other source types should be investigated along with using additional simulation methods, such as a range-dependent model.

To extend deep learning techniques to estimate individual parameters in the ocean seabed, parameter sensitivity and sloppiness will need to be carefully considered. The type of input data and choice of labels can be informed through the use of the Fisher information matrix and manifold boundary approximation methods. These tools investigate the sensitivity of ocean and source parameters and quantify the information content in the input data about each parameter. These methods can also be used to investigate what parameters (like sediment density) make the most significant change to ocean acoustics and provide guidance on how to select seabeds that are ordered in a physical manner and acoustically distinct.

There are many things to consider when it comes to future research into deep and machine learning in underwater acoustics. With the rise of a plethora of applications from science to business, new approaches will always be available to try and improve upon. This thesis has laid the foundation for these future studies and shown that deep learning techniques have significant potential for predicting environmental and source parameters in underwater acoustics.

Appendix A

ORCA Python Interface

The modeling code ORCA[2] was originally written in Fortran. The original program was designed to take in an input file, run calculations to determine the mode functions, then return the values in plain text files. Modifying the original input files is tedious especially when hundreds or thousands of calls need to be made to ORCA to calculate the mode functions in different environments and different frequencies. To improve workflow and speed, an interfaces was developed so that ORCA could be used natively in Python. "ORCA Interface" (Sec. A.1), is a whole package that contains the entire functionality of ORCA and calls the original compiled binary. The source code is available to the Underwater Acoustics Research Group at BYU through the Physics and Astronomy department GitLab installation. This documentation was generated by the Sphinx Python package.

A.1 ORCA Interface Documentation

Documentation for the ORCA Class, a simple way to handle interfacing with ORCA.

A sample SVP TOML file is presented in Sec A.1.4 and a sample OPT TOML file is presented in Sec. A.1.5.

class `orca_interface.ORCA`(***file_prefix*, *base_svp=None*, *base_opt=None*, *temp_dir=True***)

Bases: `object`

A class that can handle all of the ORCA functionality.

The ORCA class can be used for many things. It is intended to be a catch-all for using the ORCA Fortran script in an easy-to-use class. It has methods for loading in ORCA options (OPT) and sound velocity profiles (SVP) from default files or from custom files. ORCA can then be run, and the generated transmission loss or mode functions can be found directly in the attributes.

Example usage:

```
>>> from orca_interface import ORCA orca = ORCA() orca.run()
```

To pull any of the variables, you can do the following

```
>>> tl = orca.tl
```

Parameters

- **file_prefix** (*string, optional*) – The prefix for ORCA to use in all of its output files.
- **base_svp** (*string, optional*) – An SVP file to be used to overwrite all SVP defaults.
- **base_opt** (*string, optional*) – An OPT file to be used to overwrite all SVP defaults.
- **temp_dir** (*bool, optional*) – Whether or not you want to run ORCA in a temporary directory. True is recommended to keep the working directory clean.

Variables

- **file_prefix** (*string*) – The selected file prefix that ORCA will use for many things, including building the output ASCII files and also for saving the finished data.
- **temp_dir** (*bool*) – This is a basic switch for if a temporary directory will be used when ORCA is called.
- **opt** (`OPT_Configs`) – The opt attribute contains all of the necessary data to build the ORCA options files. There shouldn't be much need to change things directly inside here, so please use some of the methods.
- **svp** (`SVP_Data`) – The svp attribute contains all of the *sound velocity profile* information. It is what characterizes the water and the seabed.
- **nmode** (*int*) – The number of calculated modes.
- **eig_re** (`np.ndarray`) – The real eigenvalues.
- **eig_im** (`np.ndarray`) – The imaginary eigenvalues.
- **vg** (`np.ndarray`) – The group velocity.
- **freq** (*float*) – The frequency ORCA calculated at. This is set by the set_frequency() method.
- **mf_re** (`np.ndarray`) – The real part of the mode functions.

- **mf_im** (`np.ndarray`) – The imaginary part of the mode functions.
- **mfz** (`np.ndarray`) – The z grid corresponding to where the mode functions were evaulated at.
- **tl_z** (`np.ndarray`) – The z grid that corresponds to the transmission loss map.
- **tl_r** (`np.ndarray`) – The r grid corresponding to the transmission loss map.
- **tl** (`np.ndarray`) – The transmission loss map calculated by ORCA.

Note
The variables nmode, eig_re, eig_im, vg, freq, mf_re, mf_im, mfz, z, r, and tl do *not* exist until the run() method is called. Also, some might not exist based on the many of the different options ORCA supports. However, by default ORCA is configured to return *all* of these variables.

`add_sediment_layer`(***layer_***, ***position=None***) A method to add a sediment layer.

This method can add an entire sediment layer at any position. This is useful to have a script replace a sediment layer to calculate some form of change in transmission loss or mode function calculations. Posision needs to be done with *Python indexing*. if not set, it will be inserted at the end.

Note: ap has the following units: dB/m/kHz if positive, dB/wavelength if negative

The new layer needs to follow any of the following forms depending on the type you wish to use:

linear : [1 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2]

blug1 : [2 h cp1 g cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 beta ctol]

blug2 : [3 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 beta ctol]

blug3 : [4 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 g ctol]

These arrays can also be in the form of dictionaries with those specific keys. Just also include a type key set to the first number in the list.

Usage:

```
>>> # a linear layer
>>> # :1  h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2
>>> new_layer = [1, 20.15, 1650.0, 1670.0, 0.0, 0.0, 1.85, 1.85,
>>>     0.1, 0.15, 0.0, 0.0]
>>> # insert the new layer *after* the top sediment layer
>>> orca.add_sediment_layer(new_layer, 1)
```

Parameters

- **layer_** (*list or dict*) – The pre-defined input layer to be added. Can be either a dictionary or a list, see the description above for array and dictionary construction.

- **position** (*int, optional*) – The position to put the new layer. By default it places it at the end of the set of layers. *Use Python indexing for this parameter.*

`build_mfz_grid()` Builds the mode function grid

This returns the grid where the mode functions are defined at.

`dump_bottom_layers()` Return the bottom layers as a 2D Numpy array

This undoes the add_sediment_layer method and directly outputs the 2D array that makes up the bottom layers.

`max_modes(`***num_modes***`)` Sets the maximum number of modes to look for.

Use this method to set the maximum number of modes that ORCA should find. It does not guarantee this many, but it will absolutely limit the calculations.

Usage:

```
>>> # only find a maximum of 10 modes
>>> orca.max_modes(10)
```

Parameters **num_modes** (*int*) – The maximum number of modes. This must be an integer.

Note
Default is 50. Set to zero to force no limit.

`new_opt_from_file(`***opt_file***`)` Create the OPT configs object from a specific file.

This method was developed to be used in constructing the class if a particular OPT file was desired *after* class construction. It is recommended to set the OPT configs with the ORCA class constructor directly.

Parameters **opt_file** (*string*) – The filename/path of the OPT TOML/YAML file to be used.

`new_svp_from_file(`***svp_file***`)` Create the SVP configs object from a specific file.

This method was developed to be used in constructing the class if a particular SVP file was desired *after* class construction. It is recommended to set the SVP with the ORCA class constructor directly.

Parameters **svp_file** (*string*) – The filename/path of the OPT TOML/YAML file to be used.

`remove_sediment_layer(`***index=None***`)` A method to remove a sediment layer.

This method will remove a sediment layer at any position. By default, it removes the last listed sediment layer.

Usage:

```
>>> # remove the second layer
>>> orca.remove_sediment_layer(1)
```

Parameters **index** (*int, optional*) – Which sediment layer you wish to remove. *Use Python indexing for this parameter.*

`replace_water_profile`(***water_data***, ***constant_rho=None***, ***constant_ap=None***)

A method to replace the water profile.

This method will remove the *entire* water profile with the new set input. It accepts an array of arrays with [depth, speed] pairs, if the constant terms are both set. If they are not, it expects an array of arrays with [depth, speed, rho, ap] sets.

Usage:

```
>>> # a new ocean profile with 2 points and constant rho and ap
>>> new_water = [[0.0, 1480], [80, 1530]]
>>> const_rho = 1.0
>>> const_ap = 0.0
>>> orca.replace_water_profile(new_water, const_rho, const_ap)
```

Parameters

- **water_data** (*list of lists*) – This parameter must be a list of lists. Either of [depth, speed] pairs or [depth, speed, rho, ap] sets.
- **constant_rho** (*float, optional*) – The rho value that you wish to set as the constant value. This value *must* be set with constant_ap
- **constant_ap** (*float, optional*) – The ap value that you wish to set as the constant value. This value *must* be set with constant_rho.

Note

If constant_rho and constant_ap is set, if water_data contains a list of [depth, speed, rho, ap] sets, the rho and ap values will just be ignored.

Also, please be sure to set *both* constant_rho and constant_ap together or not at all. Errors will be thrown if not.

`run`() Run the ORCA binary based on current configs

This method takes all of the current configs and then runs ORCA. Without this method, **nothing will be calculated**. This method also sets all of the mode function variables and the transmission loss variables, so be sure to call this one!

Syntax for this method is simple:

```
>>> orca.run()
```

`run_single_frequency`(***freq***, ***alpha=None***) Run the ORCA binary based on current configs but specify one frequency

This method is particularly useful when you are looping through multiple frequencies and want parallel processing. It allows you to make sure that you are not running into race conditions when using it.

This function also *ignores* the temp_dir input in the class construction. The ORCA calculation will automatically be run in a temporary directory. This is to facilitate parallelization to not need a separate context manager.

> **Note**
> Please also be advised that this function actually *returns* the values that ORCA processes from the sub_orca() function in the base module. **It does not set the internal values at all.**

Parameters **freq** (*float*) – Frequency for which to run the single frequency.

Returns

- **nmode** (*int*) – The number of calculated modes.
- **eig_re** (`np.ndarray`) – The real eigenvalues.
- **eig_im** (`np.ndarray`) – The imaginary eigenvalues.
- **vg** (`np.ndarray`) – The group velocity.
- **freq** (*float*) – The frequency ORCA calculated at. This is set through the set_frequency() method.
- **mf_re** (`np.ndarray`) – The real part of the mode functions.
- **mf_im** (`np.ndarray`) – The imaginary part of the mode functions.
- **mfz** (`np.ndarray`) – The z grid corresponding to where the mode functions were evaulated at.

`save`(***save_dir=None***, ***new_save_name=None***) Save the innards of the ORCA class

Use this method to create a .npy file of all calculated results as well as metadata about the current internal state of ORCA. By default, it saves as [orca_pref]_[freq]_saved.npy new_save_name overwrites that default entirely.

Usage:

```
>>> # save with the default name in the existing `output` dir
>>> orca.save("output")
```

Loading in saved data:

```
>>> # load in the data
>>> loaded_data = np.load("filename.npy").item()
>>> # access the frequency:
>>> loaded+data['freq']
```

> **Note**
> Please note that when Numpy saves a dictionary, you will need to load it in with .item() to get the dictionary. See usage above/

Parameters

- **save_dir** (*string, optional*) – A string for a folder or path to save the data.
- **new_save_name** (*string, optional*) – A string for a new save name. Can be a path with a name, or just a string for the new name.

`set_frequency`(***freq***) Sets the frequency to run the calculations at.

This method overwrites any frequency that is currently within memory. It is important to use this method to make sure that the correct frequency is set. By default, it uses the frequency set in the OPT file (whether it be the default or the input file).

Usage is simple:

```
>>> new_frequency = 250 orca.set_frequency(new_frequency)
```

Parameters **freq** (*float*) – The new *single* frequency for ORCA to be run at. It *must* be a single value.

`set_mode_grid`(***mode_grid*, *force_list=True***) Set up the grid for the mode functions to be evaluated at.

ORCA can calculate the mode functions loss based on an input grid. This method sets up the grid for where the mode functions will be calculated. The grid is the depths in meters.

Usage:

```
>>> # set the mode grid to be from 0-70 m deep with 100 points
>>> orca.set_mode_grid([0, 70, 100])
```

Parameters

- **z_grid** (*array*) – This is the depth grid positioning for calculating modes.
- **force_list** (*bool, optional*) – Set to false to allow ORCA to have 3 values to set it's own grid

Note
If mode_grid is of length three and force_list is false, then it will assume the order of *min, *max*, *number* for ORCA to create its own grid for computation.

`set_ram_disk`(***ram_disk_path***) Set the root to a temporary directory

This method let's us set a temporary directory with a RAM disk path - used in run and run_single_frequency

`set_sources`(***source_grid*, *force_list=False***) Set up the depths of simulated sources for transmission loss

ORCA allows for transmission loss calculations by simulating source positions. Use this method to set up the grid for sources. They are placed at r=0 at the depths specified.

Usage:

```
>>> # move the source to 10 m deep
>>> orca.set_sources([10])
```

Parameters

- **source_grid** (*list*) – The list defining the depth points of the sources.
- **force_list** (*bool, optional*) – If placing 3 sources at specific locations, use this parameter to force them instead of being *min*, *max*, *num*.

Note
If source_grid is of length three, then it will assume the order of *min*, *max*, *number* for ORCA to create its own grid for computation. Set the optional force_list parameter to True to force 3 points.
Currently ORCA interface does not support multiple sources. In the meantime, please use one source.

`set_tl_grid`(***z_grid=None*, *r_grid=None***) Set up the transmission loss calculation grid

ORCA can calculate the transmission loss based on an input grid. It also relies on a source grid (which can be set with set_sources()) to calculate the transmission loss at depths and ranges. This method sets up the grid for where the transmission loss will be calculated. The depth grid is in meters, and the range grid is in kilometers.

Either z_grid or r_grid can be set beyond the default, but it is recommended to set both together at the same time. The best method is to use sets of 3 for each, where you go [min, max, num] for each. Self-generated arrays are supported.

Usage:

```
>>> # set the tl grid to be from 0-70 m deep with 100 points
>>> # set the tl grid to be from 0.01-2 km in range with 200 points
>>> orca.set_tl_grid([0, 70, 100], [0.01, 2, 200])
```

Parameters **z_grid** (*array, optional*) – This is the depth grid positioning. r_grid : array, optional This is the range grid positioning.

Note
If either are of length three, then it will assume the order of *min*, *max*, *number* for ORCA to create its own grid for computation. If the *number* value is directly casted to an int no matter what.
z_grid is the depths, and each depth is in meters. r_grid is the ranges, and each range is in kilometers.

static `water_attenuation`(***freq***) Get the water attentuation value corresponding to the frequency

Water attenuation in the watercolumn is dependent on frequency. This function takes that into account and calculates the new attenuation.

From MATLAB: alpha_f: attenuation in water column:
[pos~db/m/kHz, neg~dB/wavelength]

Parameters **freq** (*float*) – The frequency (in Hz) of the current case.

Returns The modified ORCA object

Return type float

A.1.1 ORCA Options and Configurations

Documentation for handling the OPT class. This class contains all of the options necessary to run ORCA properly.

class `orca_interface.orca_class.OPT_Configs`**(*input_file*)**

Bases: `object`

A class that can handle all of the OPT options

This class isn't loadable by the orca_interface package, by default. **It is instead built into the ORCA class and can be accessed as a nested object.**

Anyway, this class contains all of the various OPT options. More information about many of them can be found within the opt.sample.toml file included in the package repository.

To pull any of the variables, you can do the following:

```
>>> # access the rmax information, for example
>>> orca.opt.rmax
```

Parameters **input_file** (*string*) – The input file for setting up the OPT dataclass. By default, it pulls the default options distributed with the package.

Variables **rmax** (*float*) – The selected file prefix that ORCA will use for many things, including building the output ASCII files and also for saving the finished data.

> **Note**
> Please visit the opt.sample.toml file to view more information about these options. Generally, **you should not be needing to modify any of these** as the default options have been set for maximum functionality.

`dict_dump`**()** Creates a dictionary dump of all of the attributes in the class

Returns **out_dict** – OPT data.

Return type dict A dictionary containing *all* options contained in the

A.1.2 ORCA Sound Velocity Profile Options

Documentation for handling the SVP class. This class contains all of the water and sediment layer properties that ORCA uses.

class `orca_interface.orca_class.SVP_Data`(***default_file***) Bases: `object`

A class that can handle all of the SVP parameters

This class isn't loadable by the orca_interface package, by default. **It is instead built into the ORCA class and can be accessed as a nested object.**

Anyway, this class contains all of the various SVP parameters. More information about many of them can be found within the svp.sample.toml file included in the package repository.

To pull any of the variables, you can do the following:

```
>>> # access the rmax information, for example
>>> orca.svp.svp_title
```

Parameters **input_file** (*string*) – The input file for setting up the OPT dataclass. By default, it pulls the default options distributed with the package.

Variables

- **svp_title** (*string*) – The title given to this specific SVP.
- **sediments** (`Sediment_Layer`) – The sediments are directly attached to another class caleed Sediment_Layer. This particular attribute is actually a plain Python list of sediment layers. To access a particular layer and its attributes, use something like opt.svp.sediments[0].h to access the height value of layer 1. You can also see the documentation for the Sediment_Layer class for information on its attributes.
- **upper_cp** (*float*) – The compressional sound speed at the upper halfspace.
- **upper_cs** (*float*) – The shear sound speed at the upper halfspace.
- **upper_rho** (*float*) – The density of the upper halfspace.
- **upper_ap** (*float*) – The compressional attenuation of the upper halfspace.
- **upper_as** (*float*) – The shear attenuation of the upper halfspace.
- **ctol** (*float*) – The tolerance for the speed of sound. Not recommended to change.
- **ocean_constant_rho_ap** (*bool*) – Whether or not to use a constant rho and ap when defining the ocean. *NOTE:* it is not recommended to update this parameter. Use ORCA's built in replace_water_profile() to modify this parameter.
- **ocean_constant_rho** (*float*) – The constant value set for the water's density. *NOTE:* it is not recommended to update this parameter. Use ORCA's built in replace_water_profile() to modify this parameter.

- **ocean_constant_ap** (*float*) – The constant value set for the water's attenuation. *NOTE:* it is not recommended to update this parameter. Use ORCA's built in replace_water_profile() to modify this parameter.
- **ocean_layers** (*list of lists*) – The list of lists that defines the water's sound speed and depth. *NOTE:* it is not recommended to update this parameter. Use ORCA's built in replace_water_profile() to modify this parameter.
- **lower_cp** (*float*) – The lower compressional sound speed at the lower halfspace.
- **lower_cs** (*float*) – The shear sound speed at the lower halfspace.
- **lower_rho** (*float*) – The density of the lower halfspace.
- **lower_ap** (*float*) – The compressional attenuation of the lower halfspace.
- **lower_as** (*float*) – The shear attenuation of the lower halfspace.

Note
Please visit the svp.sample.toml file to view more information about these options.

`add_sediment_layer`(***layer_, position=None***) A method to add another sediment layer

This method will add a sediment layer to the ocean enviornment. If the optional position argument is given, it will place the sediment layer at different positions.

The new layer needs to follow any of the following forms depending on the type you wish to use:

linear : [1 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2]

blug1 : [2 h cp1 g cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 beta ctol]

blug2 : [3 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 beta ctol]

blug3 : [4 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 g ctol]

These arrays can also be in the form of dictionaries with those specific keys. Just also include a type key set to the first number in the list.

Parameters **layer_** (*list or dict*) – The pre-defined input layer to be added. Can be either a dictionary or a list, see the description above for array and dictionary construction. position : int, optional The position to put the new layer. By default it places it at the end of the set of layers. *Use Python indexing for this parameter.*

Warning
Please do not use this method directly. The ORCA class has an identical method and passes all information directly to this function. **Please use the ORCA class method instead.***

`dict_dump`() Creates a dictionary dump of all of the attributes in the class

Returns **out_dict** – A dictionary containing *all* options contained in the OPT data.

Return type dict

`remove_sediment_layer`(***index=None***) Remove an entire sediment layer

This method will remove a sediment layer from the set of sediments. By default, it will remove the bottom-most layer. Use the optional index parameter to select which layer index you want to remove.

Parameters **index** (*int, optional*) – Which sediment layer you wish to remove. *Use Python indexing for this parameter.*

Warning
Please do not use this method directly. The ORCA class has an identical method and passes all information directly to this function. **Please use the ORCA class method instead.***

`replace_water_profile`(***water_data, constant_rho=None, constant_ap=None***)

A method to replace the water sound speed profile

This method will remove the *entire* water profile with the new set input. It accepts an array of arrays with [depth, speed] pairs, if the constant terms are both set. If they are not, it expects an array of arrays with [depth, speed, rho, ap] sets.

Parameters

- **water_data** (*list of lists*) – This parameter must be a list of lists. Either of [depth, speed] pairs or [depth, speed, rho, ap] sets.
- **constant_rho** (*float, optional*) – The rho value that you wish to set as the constant value. This value *must* be set with constant_ap
- **constant_ap** (*float, optional*) – The ap value that you wish to set as the constant value. This value *must* be set with constant_rho.

Note
If constant_rho and constant_ap is set, if water_data contains a list of [depth, speed, rho, ap] sets, the rho and ap values will just be ignored.
Also, please be sure to set *both* constant_rho and constant_ap together or not at all. Errors will be thrown if not.

Warning
Please do not use this method directly. The ORCA class has an identical method and passes all information directly to this function. **Please use the ORCA class method instead.***

A.1.3 ORCA Sediment Layer Options

Documentation for handling the Sediment Layers and its attributes. This is what defines a single layer contained in the SVP “sediments” attribute.

class `orca_interface.orca_class.Sediment_Layer`**(*input_data*)** Bases: `object`

A class to hold information about a sediment layer.

This class makes it easy to manage a single sediment layer. The SVP class has the attribute sediments that is a list of these classes. To use a particular type of layer, make sure you have the right set of inputs. For this reason it is **highly recommended** to use ORCA's add_sediment_layer() method to handle creating a sediment layer. However, once it is built, internal variables can be accessed and modified.

Accessing one of these attributes from ORCA can be done like so:

```
>>> # change the height of the first layer
>>> orca.svp.sediments[0].h = 30.5
>>> # change the compressional speed at top of the third layer
>>> orca.svp.sediments[2].cp1 = 1500
```

Also, the variables corresponding to the different layers are:

linear : [1 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2]

blug1 : [2 h cp1 g cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 beta ctol]

blug2 : [3 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 beta ctol]

blug3 : [4 h cp1 cp2 cs1 cs2 rho1 rho2 ap1 ap2 as1 as2 g ctol]

Variables

- **type** (*string*) – Which type of layer you are using. Accepts "linear", "blug1" "blug2", or "blug3".
- **h** (*float*) – The height of the layer in meters.
- **cp1** (*float*) – Compressional sound speed at the top of the layer.
- **cp2** (*float*) – Compressional sound speed at the bottom of the layer.
- **cs1** (*float*) – Shear sound speed at the top of the layer.
- **cs2** (*float*) – Shear sound speed at the bottom of the layer.
- **rho1** (*float*) – Density at the top of the layer.
- **rho2** (*float*) – Density at the bottom of the layer.
- **ap1** (*float*) – Compressional attenuation at the top of the layer.
- **ap2** (*float*) – Compressional attenuation at the bottom of the layer.
- **as1** (*float*) – Shear attenuation at the top of the layer.
- **as2** (*float*) – Shear attenuation at the bottom of the layer.
- **g** (*float*) – Gradient value. (For use in a blug type.)
- **beta** (*float*) – A beta value. (For use in a blug type.)
- **ctol** (*float*) – A sound speed tolerance value. (For use in a blug type.)

Note
Not all of the variables exist above, and some may never be used based on the type of sediment layer selected (linear, blug1, blug2, blug3). Please refer to each layer type for more information.
For reference, commonalities accross all layer types include h, cp1, cs1, cs2, rho1, rho2, ap1, ap2, as1, as2.

`dict_dump()` Creates a dictionary dump of all of the attributes in the class

A.1.4 Example SSP TOML File

```
# ORCA NORMAL MODE MODEL SETTINGS
#
# This YAML File is an updated configuration file to interface with ORCA
# Follow the comments for directions, as all comments are proceeded with
    '#'
#
# This YAML File assumes version 2.0 of the ORCA opt files

# -------
# SECTION 1: General Set-up
# -------
# Insert the name of the file here. Not necessarily filename.
svp_title = "svp_mud_over_sand_sbc"


# -------
# SECTION 2: Upper Halfspace
# -------
[upper_halfspace]
# speed of sound
cp = 343.0
cs = 0.0
# density
rho = 0.00121
# attenuation
ap = 0.0
as = 0.0


# -------
# SECTION 3: SVP Points and Tolerance
# -------
[svp_ctol]
# tolerance used in fitting SVP to eliminate layers
# use 0 to keep all layers
ctol = 0
```

```


# -------
# SECTION 4: Ocean SVP Profile
# -------
# Make sure that section 3 includes the correct number of SVP points
[ocean_svp]
# will you be using constant rho and ap?
constant_rho_ap = true
# constant rho (will be added on the first line)
rho = 1
# constant ap
ap = 0

# The ocean SVP points. If constant rho and ap was not selected, include
    them
# in each layer
# layers can be either a list of lists (the current example) or a
    dictionary of pairs
# seen below.
# list version MUST follow pattern of [z, cp] (if constant is false)
# or [z, cp, rho, ap]  (if constant is true)

# if constant_rho_ap is set to false, the list and dictionary *must*
    include rho and ap
# values for each defined layer
# sidenote, ALL values in a TOML array must be of the same type, so just
    add a .0
layers = [
    [0.0, 1468.5],
    [74.4, 1469.5]
]


# -------
# SECTION 6: Bottom Layer Profiles
# -------
# Make sure that section 5 includes the correct number of bottom layers
[[layer_profiles]]
# There are three different types of profiles that can be entered.
# Use "linear", "blug1", "blug2", "blug3" and include the following:
#   type: linear
#     h  cp1 cp2  cs1 cs2  rho1 rho2  ap1 ap2  as1 as2
#   type: blug1
#     h  cp1   g  cs1 cs2  rho1 rho2  ap1 ap2  as1 as2  beta ctol
#   type: blug2
#     h  cp1 cp2  cs1 cs2  rho1 rho2  ap1 ap2  as1 as2  beta ctol
#   type: blug3
#     h  cp1 cp2  cs1 cs2  rho1 rho2  ap1 ap2  as1 as2     g ctol
# just make sure that each object in the list contains the elements
    requested
# by the type.
type = "linear"
h = 11.1
cp1 = 1468.785
cp2 = 1568.685
cs1 = 0.0
cs2 = 0.0
rho1 = 1.6
rho2 = 1.6
ap1 = 0.035
ap2 = 0.045
as1 = 0
as2 = 0

[[layer_profiles]]
type = "linear"
h =  5.0
cp1 =  1650.0
cp2 =  1650.0
cs1 =  0.0
cs2 =  0.0
rho1 =  1.8
rho2 =  1.8
ap1 =  0.1
ap2 =  0.15
as1 =  0.0
as2 =  0.0


# -------
# SECTION 7: Lower Halfspace
# -------
[lower_halfspace]
cp = 1800.0
cs = 0.0
rho = 2.20
ap = 0.25
as = 0.0

# Section 8: Top Layers
# This is unsupported since we have no use for it. It could be useful for
# oil in the future.
```

A.1.5 Example ORCA OPT File

```
# ORCA NORMAL MODE MODEL SETTINGS
#
# This TOML File is an updated configuration file to interface with ORCA
# Follow the comments for directions, as all comments are proceeded with
    '#'
#
# This TOML File assumes version 2.0 of the ORCA opt files

# -------
# SECTION 1: Types of Computation
# -------
[computation_config]
# Mode Computations
#   - 0 for no computations
#   - 1 for CW
#   - 2 for broadband
iicw = 1
# Complex k-plane images
#   - 0 for none
#   - 1 for ln(R1*R2)
#   - 2 for ln(1-R1*R2)
iikpl = 0
# PW Reflection Coefficient
#   - 0 for none
#   - 1 for R vs angle, f
#   - 2 for FFT file vs. angle
iirc = 0
# Parameter Study
#   Parameter study uses iicw, iikpl, iirce
#   - 0 for no
#   - 1 for yes
iiparm = 0
# Geoacoustic Profile
#   Outputs a geoacoustics profile file (_prof)
#   - 0 for no
#   - number of depth pts (an integer)
n_env = 0
# Output Format
#   Select output format
#   - 1 for HDF
#   - 2 for MAT
#   - 3 for ASCII (RECOMMENDED)
#   - 4 for all types
iifmt = 3


# -------
# SECTION 2: Mode Computation General Parameters
# -------
# Note: Set all to 0 to find modes automatically
```

```

[mode_config]
# Real Axis Version
#   - 0 for no
#   - 1 for yes
iirx = 0
# Min Phase Speed
#   - 0 for p-wave modes only
#   - number for the minimum phase speed
#   - -1 for seismic modes also
cphmin = 0
# Max Phase Speed
#   - 0 to just use the minimum
#   - number for the maximum phase speed
#   - negative number for maximum angle in degrees
cphmax = 0
# Min Range of Interest in km
#   - >999 to use cphmax
#   - number for range
#   - 0 to use S/R geometry
rmin = 0.01
# Max range of Interest
#   - 0 to use S/R geometry
#   - number for range
rmax = 100
# Phase Step Parm
#   Step by 2*pi/phfac
#   - Set to 4-8
#   - 0 for default (4)
phfac = 4
# Modes Weaker by db_cut Ignored
#   Removes the modes weaker than specified dB value
#   - Set between 0 and 120
#   - 0 for default (50)
db_cut = 200
# Gradient Lower h-space
#   - 0 for default
#   - -1 for homogeneous
#   - greater than 0 to set to da_bar
Aih_l = 135.001
# Aih_l = 0
# Gradient Upper h-space
#   - 0 for default
#   - -1 for homogeneous
#   - greater than 0 to set to da_bar
Aih_u = -1
# Gaussian Beam Source
#   - 0 for no
#   - 1 for yes
#   If set to yes, add the beam angle and beamwidth for each source depth
#   in sections 5 or 8
iigbs = 0
# Print Diagnostic Messages
#   - 0 for no
#   - 1 for yes
iidiag = 0

# -------
# SECTION 3: CW Node Frequencies
# -------
# Note: this section is only applied if iicw is set to 1 in Section 1

[cw_frequencies]
# Will you list the frequencies?
list = true
# Number of Frequencies
nf = 1
# List of Frequencies (in Hz)
#   This must be a TOML list of frequencies
#   The length of this list must also be equal to nf set above
items = [
  200
]
min = 200
max = 400


# -------
# SECTION 4: CW Output Options
# -------

[cw_out_config]
# TL
#   - 0 for no
#   - 1 for zs, zr, and r (see Section 5)
#   - 2 for source track/rec array in Section 10
iitl = 0
# Mode Functions
#   - 0 for no
#   - 1 for p-wave
#   - 2 for s-wave
#   - 3 for both
#   Enter the depths in Section 6
iimf = 1
# Display uz, ux, and Stress sigzz, sigzx
#   - 0 for no
#   - 1 for yes
iisig = 0
# Mode Trajectory in k-plane
#   Associated ASCII file
#   - 0 for no
#   - 1 for yes
iimt = 0
# Disp Curves
#   - 0 for no
#   - 1 for vg
#   - 2 for vph
#   - 3 for both
iidc = 3
# Mode Eigenvalues
#   Re(kn), Im(kn)
#   TODO: Figure this one out
iikn = 1
# List of Eigenvalue Characteristics
#   TODO: Yeah, what do the values mean?
iieig = 0
# Output Various Files to Use in Other Programs
#   - iikrak: Kraken
#   - iioas: OASES
#   - iifepe: FEPE
#   - iimlab: MODELAB
#   - 0 for no
#   - 1 for yes
iikrak = 0
iioas = 0
iifepe = 0
iimlab = 0


# -------
# SECTION 5: Source Depths, Reciever Depths, and Ranges
# -------
# source_depths creates an object with various bits of information.
#
# If you wish to specify individual source depths, set source_depths.list
# to true and then within source_depths.items, list the individual depths
# in meters.
# TODO: clean this section - I have to double check how this all works
[source_depths]
list = true
# list the source depths themselves
# 6 is generally a default for a surface ship
items = [
  6
]
# values for if list is set to False
num = 50
min = 1
max = 99

```

```
# receiver_depths works the same way as source_depths
[receiver_depths]
list = false
items = [
  60,
  100
]
num = 200
min = 1
max = 120

# reciever_ranges works the same way as source_depths and reciever_depths.
# However, these values are all in kilometers.
[receiver_ranges]
list = false
items = [
  5,
  20,
  25
]
num = 250
min = 0.1
max = 10


# -------
# SECTION 6: Mag/Phase Options and Depths
# -------
# Use these options if iimf is greater than 0 or iisig = 1
[mag_phase_config]
#
iiri = 3
#
iimp = 0
#

  [mag_phase_config.mag_phase]
  list = false
  items = [
    5,
    10,
    15
  ]
  num = 74
  min = 1
  max = 74


# -------
# SECTION 7: BB Mode Computation Parameters
# -------
[bb_config]
# Sample Frequency
fs = 512
# NFFT or Time Window
#   - positive number: NFFT
#   - negative number: time window in S
nfft_Tw = 256
# Minimum Frequency
fmin = 90
# Maximum Frequency
fmax = 250
# Output FFT File
#   - 0 for no
#   - 1 for yes, input zs, zr, and r in Section 8
#   - 2 for yes, read file in Section 10
iifft = 1
# Output BB Eigenvalues and Functions
#   - 0 for no
#   - 1 for yes, input zs, zr, and r in Section 8
#   - 2 for yes, read file in Section 10
iiout = 0
# Frequency Trajectory (ASCII)
#   - 0 for no
#   - 1 for yes
iift = 0
# Mode Trajectory (ASCII)
#   - 0 for no
#   - 1 for yes
iimt = 0
# Display Curves
#   - 0 for no
#   - 1 for yes
iidc = 0
# Mode Functions at Receiver Depth
#   Returns mode functions at the depths defined in Section 8
#   - 0 for no
#   - 1 for 2D HDF
#   - 2 for 3D HDF
iimf = 0


# -------
# SECTION 8: Depths and Ranges for IIFFT, IIOUT or IIMF
# -------
# Source information to use with IIFFT and IIOUT
#   If list is set to true, the script looks at items. If false, it uses
    num,
#   min, and max
[ii_source]
```

```
list = true
items = [
  36
]
num = 10
min = 5
max = 20

# Reciever depths information to use with IIFFT and IIOUT
#   If list is set to true, the script looks at items. If false, it uses
    num,
#   min, and max
[ii_depths]
list = true
items = [
  70
]
num = 10
min = 5
max = 40

# Reciever range information to use with IIFFT and IIOUT (km)
#   If list is set to true, the script looks at items. If false, it uses
    num,
#   min, and max
#   This uses km instead of meters
[ii_ranges]
list = false
items = [
  2,
  10
]
num = 5
min = 0.5
max = 0.8


# -------
# SECTION 9: Parameter Study
# -------
# TODO: adjust this to accept multiple parameter variation studies
[iiparm_opts]
# Number of CW mode Runs
nrun = 10
# Number of parameters to vary
nparm = 1
# Random # seed (Integer)
#   - 0 for none (will vary linearly from val1 to val2)
rseed = 0
# Medium
```

```
obt = 0
nlay = 1
ktb = 2
pc = 0
val1 = 200
val2 = 100


# -------
# SECTION 10: S/R Geometry for Source Track and Rec Array
# -------
# this is if iitl=2 in section 4 or iifft=2 in section 8
[sr_geom]
# zs spacing
zs = 36
# number of source segments
n_src_seg = 1
# filename to run
file_name = "hla_array"


# -------
# SECTION 11: Source Track
# -------
# For specifying a source track
[source_track]
# a list of objects
x_y_form = true
iic = 0
v = 5
t1 = 0
t2 = 1
# + for dt, - for nt
dtnt = 50
# if we're using rect form, these will be used
x1 = 0.5
y1 = 90
x2 = 0
y2 = 0
# if we're using polar form, these will be used
cpa = 0
phi = 10


# -------
# SECTION 12: k-plane Images
# -------
# if iikpl is greater than 0 in section 1
# List of k-plane images
[[kplane_imgs]]
```

```
freq = 100
iivar = 1
iikf = 3
kr1 = 0.8
kr2 = 1.0
nkr = 101
ki1 = 200
ki2 = 0
nki = 101
nduct = 1
iiph = 2
iishp = 1
iishs = 1


# -------
# SECTION 13: Reflection Coefficient vs f (Hz) and Grazing Angle (deg)
# -------
# Used if iirce is set to 1 in Section 1
[r_vs_f]
freq1 = 10
freq2 = 50
nfreq = 41
iilog = 0
theta1 = 90
theta2 = 0
ntheta = 91


# -------
# SECTION 14: F T File of R vs. Frequency and Angle
# -------
# Used if iirc is set to 2 in Section 1
[rtr_vs_fa]
freq1 = 10
freq2 = 50
fs = 100
nfft = 512
theta1 = 90
theta2 = 0
ntheta = 91
```

Appendix B

Data Simulation Program

What follows is a configuration file used for the data generation script. This configuration file (written with the TOML syntax) shows the capabilities of the data generation script for building simulated datasets. There are two main options for the source type, SUS charge and SOO spectrogram simulation (see Sec. 2.2 for more information). The full source code for the data simulation program is available to the Underwater Acoustics Research Group at BYU on the Physics and Astronomy department GitLab installation.

```
# This document contains the configurations for the main file TOML was
   chosen
# because its harder to actually mess up indentation isn't necessary

# some general setup first:
dataset_name = "test_dataset"

# SAVETYPE - how to save the final array
#   - "numpy" will save it as a NumPy file - ONLY IN SUS
#   - "matlab" will save it as a MATLAB file - ONLY IN SUS
#   - "hdf5" will save it as an hdf5 file - ONLY IN SOO
save_type = "hdf5"

# base output directory: blank defaults to "output"
output_directory = "output"

## extraction information *FOR SUS CHARGES ONLY*
extract_signal = true
# how long (in seconds) the extraction should be
```

```
extract_length = 1.0
# where direct arrival should occur
extract_direct = 0.158
# if you want to save the true times from extraction, only triggers if
    extract_signal is /also/ true
extract_time_save = false

# ONE ORCA OPTION
# how many modes to get from ORCA
# increasing this number may provide more "precise" results, but will also
     increase
# runtime, so be careful adjusting this.
# TODO: remove this option and allow the number of modes be dynamically
    chosen
max_modes = 50

# PARALLELIZE - whether or not you wish to parallelize the calculations (
    highly recommended)
parallelize = true
# how many cores you wish to use, recommended one minus the number of
    cores available (leave as 0 for this case)
num_cores = 0

# CHECKPOINTS - whether or not you wish to save checkpoints
#    - True will save a checkpoint that can easily be reloaded if there's
    a system failure
#    - False will not save a checkpoint offering a speed-up
# THIS ONLY APPLIES TO SUS CHARGES
checkpoint = true

# Maximum File Size - in Megabytes, only applies to the SUS charges
max_file_size = 500



### SOURCE BLOCK ###
# this block lets us set up more information about our source
[source_info]

# Which source type you're going to use, currently only supports "sus", "
    ship" and "impulse"
source_type = "ship"

# maximum frequency: half of the sampling rate, keep 2500 for IVAR tests
# if using "soo" spectrograms, this is the highest frequency in the
    spectrogram
fmax = 500

    [source_info.sus_charges]
    ### SUS CHARGE INFORMATION - THIS IS ONLY USED IF THE SOURCE TYPE IS
```

```
SET
 # TO SUS ABOVE

 # source depth 24.8 is the Mk46 detonation depth (18.3) + a correction
 factor
 # depth of the source
 zs = [18]

 # w constant for the SUS generation - keep at 0.031
 w = 0.031

 # frequency windowing - set the upper and lower bounds (Hz) of the
window applied to
 # the SUS spectra
 bandpass_window = [20, 2500]

 [source_info.ship_noise]
 ### SHIP NOISE INFORMATION - THIS IS ONLY USED IF THE SOURCE TYPE IS
SET TO
 # SHIP ABOVE

 # the minimum frequency to save the spectrogram, only if spectrogram!
 # useful for reducing the amount of data needed to load
 # set to 0 to have the minimum be df
 fmin = 30

 # total length of time to use (in hours) for the half track sims
 spec_length = 1 # other good examples: 3.3333333333333 #0.5 #1.229

 # interval between the "measurements" (in seconds)
 spec_interval = 10

 # how long the "extracted" signal should be at each interval
 # this is what determines the frequency spacing: df = 1 / spec_length
 # e.g. 1 for 1 Hz spacing
 ship_time_chunk = 1 # 3.125 is the value used in the original MATLAB
script

 # ship depth used for calculations in m, generally about 6
 ship_depth = [6.0]
 # randomize ship depths, if this is selected it'll choose a random
value between
 # the FIRST and SECOND values in the ship_depth above. It will ignore
other values!
 ship_depth_random = false
 # number of random ship depths to choose
 ship_depth_random_num = 1

 # if the CPA should be centered in the grid
 # THIS IS ALWAYS TRUE BECAUSE THE STARTING PLACES BELOW ISN'T YET
```

```
    IMPLEMENTED
    ship_cpa_centered = true

    # ship speeds in knots, these become the bins like the ranges (see below)
    ship_speeds = [18, 20, 22]
    # add randomized ship speeds
    ship_speeds_per_bin = 1
    # min, max ship speeds when doing random values
    ship_speeds_min_max = [16, 24]
    # type of distribution for the random selection, uniform or normal (u/n)
    ship_speeds_random_dist = 'u'
    # randomize the ship speeds every SSP calculation
    ship_speeds_random_each_ssp = true
    # how far to stay from bin boundaries
    ship_speeds_random_epsilon = 0.01

    # ship starting distances (in y). The ship's CPA is always at Y=0, but you can use this to have
    # different starting places
    # NOT YET IMPLEMENTED
    ship_starting_places = []

    # spectra output type
    # true - converts the values to levels
    # false - returns the complex "pressure"
    spec_output_level = false
    # spectral output density
    # if you just want to save the levels, (previous must be true)
    # you can choose to have spectral levels or spectral density levels. True does the latter
    spec_output_density = true

    # whether or not to use random phase in the spectrum
    # as per SIM 1.1, this should be false, but the option remains
    rand_phase = true


### ENVIRONMENT INFORMATION
# What follows are the environments you wish to use.
[environments]
# inside this block, you can list the config file names that you want to loop
# through. By default, it will search for the files within the `/data` folder.
# use directory

# if directory is blank, it will just search inside data. this can be relative
# to this program or an absolute directlry.
env_directory = "data"

env_file_list = [
    "svp_deepmud_sbc.toml",
    "svp_mud_over_sand_sbc.toml",
    "svp_sandysilt_sbc.toml",
    "svp_sandy_sbc.toml"
]

# Now, give us some information about the files that will be loaded to change
# the ocean SVP. Same rules apply to these files as above. NOTE: the OCEAN parts
# of the SVP config files will be ignored and replaced with those found here. So
# MAKE SURE all of your oceans are expressed in this list.
ocean_svp_directory = "data"
ocean_svp_file = "SSPs_SCB2017_sample.mat"



### RANGE INFORMATION
## This governs how the ranges are set. All arrays of microphones
# are based off of this.
# NOTE:
# THESE RANGES BECOME CLOSEST POINT OF ARRIVAL WHEN USING SHIP OF OPPORTUNITY
[range_classes]
# number of ranges per bin. It will *always* include the bin centers
# in the list of ranges. If it is just one, it will just be the bin centers.
# if set to more than one, it'll randomly select between the bounds of the bin
# in a uniform or normal distribution as defined below
ranges_per_bin = 1

# SET THE ABSOLUTE BIN LIMITS minimum range value
min_range = 1.5

# max range value - not included, just an extra bound for the final
# distribution - doesn't really affect gaussian but could affect
# normal distribution
max_range = 15

# where the centers of the binds should be, or a full list of the ranges
# of interest - this is all in km
# minimum for first bin will always be 0
bin_centers = [2.0, 2.5, 3.0, 3.5, 4.0,
    4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5, 9.0,
    9.5, 10.0, 10.5, 11.0, 11.5, 12.0, 12.5, 13.0, 13.5, 14.0, 14.5
]

# whether or not you want the ranges to be randomly drawn each ssp
# meaing it will be drawn for every env + ssp combo.
# this is only really used if ranges_per_bin is set to greater than one
random_each_ssp = true

# how far to stay from boundaries
epsilon = 0.01

# sampling type - uniform or normal - u or n
type = "n"



### RECEIVER BLOCK ###
# this block governs how the receivers are built
[receiver_info]
# the grid - currently supports independent ranges and
# depths, meaning that it'll make a grid given input and
# output

# if you want the depths to be a number of meters off of the *bottom* of
    the ocean, do so here
relative_depth = true

# positions of receivers - *in m*
# this is a list-of-lists where each inner list requires three values
# those are [z, x, y]
# z is the depth of the receiver (this can be relative thanks to
    relative_depth)
# x is the "horizontal" point of the receiver, whether it's closer or
    farther from the source from 0,0 (negative is closer)
# y is the "vertical" point of the receiver, whether it's offset from the
    center of the array
receiver_positions = [
    [0.5, 0.0, 0.0],
    # [0.5, 0.0, 0.0],
    # [1.0, 0.0, 0.0],
    # [1.5, 0.0, 0.0],
]
# [0.5, 0.0, 0.0] indicates 0.5 meters from the bottom, centered in x and
    y with range calculations

#### LEGACY for SUS CHARGES
# depth of receivers - *in m*
zra = [0.5] # half a meter off the bottom of the ocean
# sub ranges of receivers
ra = [0.0] # the different ranges relative to the main range - *in km*
```

Appendix C

Machine Learning Code Information

What follows is a configuration file for the script that runs the machine learning framework. After the study presented in Chapter 5, an effort was made to standardize the machine learning code used in the research group. The reworked code contains all of the pieces used to train the models in all of the studies used, but now can be configured by a single TOML file. A copy of this file is presented. All code used for the machine learning aspects of the studies in this thesis are available to the Acoustics Research Group on the Physics and Astronomy GitLab installation.

Machine learning framework configuration file:

```
##### TRAINING SETTINGS as a TOML
# To ease the use of the training script, a method of setting up
# various settings in a TOML is provided. Just go through these
# settings and edit them as you please.

### GENERAL TRAINING SETTINGS
#   What follows are some general training settings.

## Run Name
#   A simple name used to uniquely identify this particular run. It
#   is used when saving the model and statistics about training and
#   testing
run_name = "TEST_run"

## Run Save Path
#   Where you'd like the model and stats saved to your machine.
run_save_path = "default_output"
```

```

## Run Description
#   A longer-form description if you need to express what you were
#   trying in words.
run_description = """
This is just a test run of the various training options
available through the "base" script.
"""

## Number of Models to Train
#   This option lets you set how many times you want to run the
#   script with the same configurations. This is useful for
#   determining how well a specific model does on a dataset by
#   accounting for random weight initialization and dataset splits.
#   If you just need one model, just set this option to one. Please
#   note, if using k-fold cross validation (an option later in this
#   file), this will be ignored and it will be performed
num_runs = 1

## GPU Device
#   This option lets you select a GPU to run this on if you have
#   more than one. If you only have one, leave it as 1
gpu_id = 0

## Model Name
#   This is one of the cool features of this code base. You can
#   specify what model you want to train on just by providing a
#   name. Make *your own* Python file containing your deep learning
#   model and save it to the `models` directory. Then, the script
#   will find the model's class name based on what you put in the
#   option below. That way you don't even have to edit the training
#   script to actually load your model! An example model is found in
#   `models/example_cnn.py`.
model_name = "Example2DModel"

## Number of Epochs used for Training
#   Number of times a dataset is passed through a network through
#   training.
num_epochs = 20

## Starting Learning Rate
#   Could be used as a *starting* learning rate if a scheduler is
#   defined and used below.
learning_rate = 0.001

## Lable Names to Train On
#   This is a list of names that you want to train on. Please note
#   that the dataloader *has* to support these for them to work.
labels_to_learn = ["geo", "cpa", "speed"]

```

```
## Scheduler
#   When an epoch is finished, the learning rate can be adjusted via
#   a scheduler. A cosine annealing rate is probably our favorite.
#   OPTIONS: cosine, exponential, step, plateau, none
scheduler = "cosine"

## Epochs Per Test Pass
#   The number of epochs to pass before geting metrics on testing.
#   For long runs, 5 is sufficient, but a more accurate view of the
#   trends may require smaller numbers.
epochs_per_test = 5

## LEARN MODE
#   What mode you want the labels to be learned. The options are:
#   classification, regression, and multitask.
#   * classification:
#       * the "unique" values are split up into "classes" and the
#         network learns a probability distribution In the case of
#         more than one label to learn, it employs
#         "multiclassification" where all unique *combinations* are
#         devided into their own class.
#   * regression:
#       * The network learns to return a number. If the correct
#         value is 10, it will learn to return 10. In the case of
#         more than one label to learn, it employs "multiregression"
#         where all of the labels are given as numbers.
#   * multitask:
#       * The network learns to do combinations of regression and
#         classification on the multiple labels learned. If this is
#         selected, please be sure to modify multitask_settings
#         below!
learn_mode = "regression"

### Training and Testing Options
#   These options govern how many times the network will be run and
#   how the train and test dataset will be used.

## Number of Times to Train the Network
#   This is how many times the network should be trained. It will
#   produce this number of networks. This option is *ignored* if
#   K-fold cross-validation is selected.

## Test on Secondary Dataset
#   If this option is set to true, the script will load a secondary
#   dataset. If set to false, it will rely on the train-test split
#   defined below
use_test_dataset = false

## K-FOLD Cross-validation
#   If this is true, it will enable k-fold cross validation and
#   ignore the train_test_split. However, this will be ignored if
#   the use_test_dataset option is set to true
k_fold = false
k_num_folds = 5

## Train-test Split
#   If there is no k_fold enabled, nor a secondary dataset, the
#   script will split the dataset by this amount. The number entered
#   refers to how much of the data goes to the *training* dataset.
train_test_split = 0.90

## Early Stopping
#   If you want to train the network with early stopping. This takes
#   the loss specified ees if it is improving between epochs.
#   **NOTE** if this is enabled, the `epochs_per_test` option will
#   effectively become **one**.
early_stopping_enable = true
#   Which metric to use to determine
early_stopping_metric = "validation"
#   How many epochs to wait for improvement before ending, lowest
#   patience is 1
early_stopping_patience = 5
#   Minimum change to be observed before an improvement is noted
early_stopping_delta = 0.001

### DATASET OPTIONS
#   These options are required for loading in the training and
#   testing datasets.
## Dataset Type
#   This option specifies the dataset options that you will be
#   learning from. It currently supports "sus" and "soo"/"ship". To
#   add more types of datasets, please edit the
#   `tools/dataset_selection.py` and add your dataset class to the
#   dataloaders repository. Please note that your __item__ method
#   must return (features, labels) as tuples for these scripts to
#   work. For testing purposes (on a small dataset), "mnist" is also
#   an option.
dataset_type = "soo"

[dataset_settings]
#   This covers the settings required to load in a dataset from the
#   soo or sus categories.

# The name of the train dataset. See the csv file located with the
# dataloaders for the names you can choose.
train_dataset_name = "example_train_dataset"
# the name of the test dataset. If k-fold crossvalidation was
# selected, this will be ignored. This will also be ignored if
# use_test_dataset was set to true
test_dataset_name = "example_test_dataset"
```

```
# The total amount of memory that can be allocated to these datasets
# (WIP feature)
hard_memory_limit = 32

# whether or not to normalize the data
data_norm = false
# the method to normalize the data if data_norm is true
data_norm_method = "individual"

# whether or not to normalize the labels (useful for regression,
# saved to the model so that real predictions can be made)
label_norm = true
# label norm method (ignored if label_norm is false): max,
# normalize, standard
label_norm_method = "standard"
# which labels to normalize, *must* be from the list above, leave
# empty for all
label_norm_use = []

# FOR SOO TYPE ONLY
# How you want the channels to be loaded: real_imaginary, abs,
# levels real_imaginary -> one channel is the real part, the other
# channel is the imaginary part abs -> only one channel and it's the
# absolute pressure (take the abs of the real and imaginary part)
# levels -> converts the absolute pressure to decibels re 1 muPa
spectrogram_channel_type = "real_imaginary"

train_ssps = []
train_rs = []
train_geo = []


### MISCELANEOUS OPTIONS:
#   Some of these settings are necessary for other things to work.

### MODEL KEYWORD ARGUMENTS
#   If the model you've programmed has different keyword arguments,
#   you can add them here
[model_kwargs]
batch_norm = true

### SCHEDULER KEYWORD ARGUMENTS
[scheduler_kwargs]
#   This will need to be edited based on the documentation for your
#   scheduler of choice. Please visit
#   https://pytorch.org/docs/stable/optim.html and find your
#   scheduler and see what other information it requires to run.
#   e.g.
#   the "cosine" annealing learning rate requires a "T_max"
#   parameter and an optional 'eta_min' parameter, so it would be
#   input as the following. If the cosine annealing one is selected,
#   if T_max is 0, then it uses the number of epochs:
T_max = 0
eta_min = 0

### MULTITASK SETTINGS
#   If you set MULTITASK to true above, you *need* to make sure that
#   this section is built *IF* you want fine control over the
#   individual labels you want to learn. If these are empty, then it
#   will default to options set inside the tools/__init__.py file.
[multitask_settings]
# Part of multitask requires "scaling" so that the network learns
# the importance of different labels on it's own. The "learnable"
# value here is "eta" in the multitask equation. Enter in X numbers
# in a list to be the starting eta values. put them in order
# applying to `labels_to_learn` so that it is properly
# added.
eta = [1.0, 0.5, 0.8]

[[multitask_settings.individual]]
# Here's an example for the 'geo' label. MAKE SURE the label name
# matches exactly
label = "geo"
mode = "classification"
criterion = "crossentropy"
[multitask_settings.individual.criterion_args]
# and then includind this criterion_args, you can specify individual
# loss parameters according to
# https://pytorch.org/docs/stable/nn.html
reduction = "mean"

[[multitask_settings.individual]]
label = "cpa"
mode = "regression"
criterion = "mse"
[multitask_settings.individual.criterion_args]
# this section can be blank if you have no arguments to add (that's
# the usual case)


[other_options]
# Don't mess with these unless you know what you're doing.

# if you need to extend the available options, please feel free to
default_classification_criterion = "crossentropy"

# default regression criterion:
default_regression_criterion = "mse"

# use these fields to add your kwargs for the loss values.
```

```
[other_options.regression_kwargs]
# This section exists for regression criterion kwargs if necessary

[other_options.classification_kwargs]
# This section exists for classification criterion kwargs if
# necessary
```

Bibliography

[1] P. H. Dahl and D. R. Dall'Osto, "Vector acoustic analysis of time-separated modal arrivals from explosive sound sources during the 2017 seabed characterization experiment," IEEE Journal of Oceanic Engineering (2019).

[2] E. K. Westwood, C. T. Tindle, and N. R. Chapman, "A normal mode model for acousto-elastic ocean environments," J. Acoust. Soc. Am. **100**(6), 3631–3645 (1996).

[3] M. B. Porter, "The kraken normal mode program," , SACLANT Undersea Research Centre Memorandum (SM-245) and Naval Research Laboratory Memorandum Report No. 6920 (1991).

[4] M. D. Collins, "A split-step padé solution for the parabolic equation method," J. Acoust. Soc. Am. **93**(4), 1736–1742 (1993).

[5] M. D. Collins, "An energy-conserving parabolic equation for elastic media," J. Acoust. Soc. Am. **94**(2), 975–982 (1993).

[6] A. Tolstoy, *Matched field processing for underwater acoustics* (World Scientific, 1993).

[7] M. D. Collins, W. A. Kuperman, and H. Schmidt, "Nonlinear inversion for ocean-bottom properties," J. Acoust. Soc. Am. **92**(5), 2770–2783 (1992).

[8] P. Gerstoft, "Inversion of seismoacoustic data using genetic algorithms and a posteriori probability distributions," J. Acoust. Soc. Am. **95**(2), 770–782 (1994).

[9] J. A. Shorey, L. W. Nolte, and J. L. Krolik, "Computationally efficient monte carlo estimation algorithms for matched field processing in uncertain ocean environments," Journal of Computational Acoustics **02**(03), 285–314 (1994).

[10] T. B. Neilsen and E. K. Westwood, "Extraction of acoustic normal mode depth functions using vertical line array data," J. Acoust. Soc. Am. **111**(2), 748–756 (2002).

[11] D. P. Knobles, R. A. Koch, L. A. Thompson, K. C. Focke, and P. E. Eisman, "Broadband sound propagation in shallow water and geoacoustic inversion," J. Acoust. Soc. Am. **113**(1), 205–222 (2003).

[12] S. E. Dosso, "Quantifying uncertainty in geoacoustic inversion. i. a fast gibbs sampler approach," J. Acoust. Soc. Am. **111**(1), 129–142 (2002).

[13] D. P. Knobles, P. S. Wilson, J. A. Goff, L. Wan, M. J. Buckingham, J. D. Chaytor, and M. Badiey, "Maximum entropy derived statistics of sound-speed structure in a fine-grained sediment inferred from sparse broadband acoustic measurements on the new england continental shelf," IEEE Journal of Oceanic Engineering 1–13 (2019).

[14] S. E. Dosso and M. J. Wilmut, "Bayesian multiple-source localization in an uncertain ocean environment," J. Acoust. Soc. Am. **129**(6), 3577–3589 (2011).

[15] S. Dosso, M. Yeremy, J. Ozard, and N. Chapman, "Estimation of ocean-bottom properties by matched-field inversion of acoustic field data," IEEE Journal of Oceanic Engineering **18**(3), 232–239 (1993).

[16] Z.-H. Michalopoulou and U. Ghosh-Dastidar, "Tabu for matched-field source localization and geoacoustic inversion," J. Acoust. Soc. Am. **115**(1), 135–145 (2004).

[17] T. B. Neilsen, "An iterative implementation of rotated coordinates for inverse problems," J. Acoust. Soc. Am. **113**(5), 2574–2586 (2003).

[18] B. Z. Steinberg, M. J. Beran, S. H. Chin, and J. H. Howard Jr, "A neural network approach to source localization," J. Acoust. Soc. Am. **90**(4), 2081–2090 (1991).

[19] R. Lefort, G. Real, and A. Drémeau, "Direct regressions for underwater acoustic source localization in fluctuating oceans," J. Appl. Acoust. **116**, 303–310 (2017).

[20] H. Niu, E. Ozanich, and P. Gerstoft, "Ship localization in santa barbara channel using machine learning classifiers," J. Acoust. Soc. Am. **142**(5), EL455–EL460 (2017).

[21] H. Niu, E. Reeves, and P. Gerstoft, "Source localization in an ocean waveguide using supervised machine learning," J. Acous. Soc. Am. **142**(3), 1176–1188 (2017).

[22] Z. Huang, J. Xu, Z. Gong, H. Wang, and Y. Yan, "Source localization using deep neural networks in a shallow water environment," J. Acoust. Soc. Am. **143**(5), 2922–2932 (2018).

[23] Y. Wang and H. Peng, "Underwater acoustic source localization using generalized regression neural network," J. Acoust. Soc. Am. **143**(4), 2321–2331 (2018).

[24] W. Wang, H. Ni, L. Su, T. Hu, Q. Ren, P. Gerstoft, and L. Ma, "Deep transfer learning for source ranging: Deep-sea experiment results," J. Acoust. Soc. Am. **146**(4), EL317–EL322 (2019).

[25] E. Ozanich, P. Gerstoft, and H. Niu, "A feedforward neural network for direction-of-arrival estimation," J. Acoust. Soc. Am. **147**(3), 2035–2048 (2020).

[26] E. M. Fischell and H. Schmidt, "Classification of underwater targets from autonomous underwater vehicle sampled bistatic acoustic scattered fields," J. Acoust. Soc. Am. **138**(6), 3773–3784 (2015).

[27] E. L. Ferguson, R. Ramakrishnan, S. B. Williams, and C. T. Jin, "Convolutional neural networks for passive monitoring of a shallow water environment using a single sensor," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE (2017), pp. 2657–2661.

[28] W. Liu, Y. Yang, M. Xu, L. Lü, Z. Liu, and Y. Shi, "Source localization in the deep ocean using a convolutional neural network," J. Acoust. Soc. Am. **147**(4), EL314–EL319 (2020).

[29] Z. Michalopoulou, D. Alexandrou, and C. d. Moustier, "Application of neural and statistical classifiers to the problem of seafloor characterization," IEEE Journal of Oceanic Engineering **20**(3), 190–197 (1995).

[30] J. Benson, N. R. Chapman, and A. Antoniou, "Geoacoustic model inversion using artificial neural networks," Inverse Problems **16**(6), 1627 (2000).

[31] Y. Stephan, X. Demoulin, and O. Sarzeaud, "Neural direct approaches for geoacoustic inversion," Journal of Computational Acoustics **6**(01n02), 151–166 (1998).

[32] J. Piccolo, G. Haramuniz, and Z.-H. Michalopoulou, "Geoacoustic inversion with generalized additive models," J. Acoust. Soc. Am. **145**(6), EL463–EL468 (2019).

[33] C. Frederick, S. Villar, and Z.-H. Michalopoulou, "Seabed classification using physics-based modeling and machine learning," arXiv preprint arXiv:2003.11156 (2020).

[34] M. J. Bianco, P. Gerstoft, J. Traer, E. Ozanich, M. A. Roch, S. Gannot, and C.-A. Deledalle, "Machine learning in acoustics: Theory and applications," J. Acoust. Soc. Am. **146**(5), 3590–3628 (2019).

[35] P. S. Wilson, D. P. Knobles, P. H. Dahl, A. R. McNeese, and M. C. Zeh, "Short-range signatures of explosive sounds in shallow water used for seabed characterization," IEEE Journal of Oceanic Engineering (2019).

[36] N. R. Chapman, "Source levels of shallow explosive charges," J. Acoust. Soc. Am. **84**(2), 697–702 (1988).

[37] K. D. Heaney, “Rapid geoacoustic characterization using a surface ship of opportunity,” IEEE Journal of Oceanic Engineering **29**(1), 88–99 (2004).

[38] C. Park, W. Seong, and P. Gerstoft, “Geoacoustic inversion in time domain using ship of opportunity noise recorded on a horizontal towed array,” J. Acoust. Soc. Am. **117**(4), 1933–1941 (2005).

[39] L. Muzi, M. Siderius, and C. M. Verlinden, “Passive bottom reflection-loss estimation using ship noise and a vertical line array,” J. Acoust. Soc. Am. **141**(6), 4372–4379 (2017).

[40] L. Xu, K. Yang, and Q. Yang, “Joint time-frequency inversion for seabed properties of ship noise on a vertical line array in south china sea,” IEEE Access **6**, 62856–62864 (2018).

[41] S. C. Wales and R. M. Heitmeyer, “An ensemble source spectra model for merchant ship-radiated noise,” J. Acoust. Soc. Am. **111**(3), 1211–1231 (2002).

[42] D. F. Van Komen, T. B. Neilsen, D. P. Knobles, and M. Badiey, “A convolutional neural network for source range and ocean seabed classification using pressure time-series,” Proc. Meet. Acoust. **36**(1), 070004 (2019).

[43] D. F. Van Komen, T. B. Neilsen, K. Howarth, D. P. Knobles, and P. H. Dahl, “Seabed and range estimation of impulsive time series using a convolutional neural network,” J. Acoust. Soc. Am. **147**(5), EL403–EL408 (2020).

[44] D. Svozil, V. Kvasnicka, and J. Pospichal, “Introduction to multi-layer feed-forward neural networks,” Chemometrics and intelligent laboratory systems **39**(1), 43–62 (1997).

[45] T. D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” Neural Netw. **2**(6), 459–473 (1989).

[46] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 807–814.

[47] D. F. Van Komen, T. B. Neilsen, D. P. Knobles, and M. Badiey, "A feedforward neural network for source range and ocean seabed classification using time-domain features," Proc. Meet. Acoust. **36**(1), 070003 (2019).

[48] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series," The handbook of brain theory and neural networks **3361**(10), 1995 (1995).

[49] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems* (2012), pp. 1097–1105.

[50] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556 (2014).

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

[52] G. R. Potty, J. H. Miller, and J. F. Lynch, "Inversion for sediment geoacoustic properties at the new england bight," J. Acoust. Soc. Am. **114**(4), 1874–1887 (2003).

[53] J.-X. Zhou, X.-Z. Zhang, and D. P. Knobles, "Low-frequency geoacoustic model for the effective properties of sandy seabottoms," J. Acoust. Soc. Am. **125**(5), 2847–2866 (2009).

[54] P. S. Wilson, D. P. Knobles, and T. B. Neilsen, "Guest editorial an overview of the seabed characterization experiment," IEEE Journal of Oceanic Engineering **45**(1), 1–13 (2020).

[55] C. Gervaise, B. G. Kinda, J. Bonnel, Y. Stéphan, and S. Vallez, “Passive geoacoustic inversion with a single hydrophone using broadband ship noise,” J. Acoust. Soc. Am. **131**(3), 1999–2010 (2012).

[56] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *NIPS-W* (2017).

[57] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” arXiv preprint arXiv:1412.6980 (2014).

[58] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning* (MIT Press, 2016).

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

[60] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” Neural computation **9**(8), 1735–1780 (1997).

[61] J. A. Goff, A. H. Reed, G. Gawarkiewicz, P. S. Wilson, and D. P. Knobles, “Stratigraphic analysis of a sediment pond within the new england mud patch: New constraints from high-resolution chirp acoustic reflection data,” Marine Geology **412**, 81–94 (2019).

[62] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, and L. Antiga, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems* (2019), pp. 8024–8035.

[63] I. Loshchilov and F. Hutter, “Sgdr: Stochastic gradient descent with warm restarts,” arXiv preprint arXiv:1608.03983 (2016).

[64] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Ijcai*, Montreal, Canada (1995), Vol. 14, pp. 1137–1145.

[65] S. Mao, E. Tu, G. Zhang, L. Rachmawati, E. Rajabally, and G.-B. Huang, “An automatic identification system (ais) database for maritime trajectory prediction and data mining,” in *Proceedings of ELM-2016* (Springer, 2018), pp. 241–257.

[66] D. J. Battle, P. Gerstoft, W. A. Kuperman, W. S. Hodgkiss, and M. Siderius, “Geoacoustic inversion of tow-ship noise via near-field-matched-field processing,” IEEE journal of oceanic engineering **28**(3), 454–467 (2003).

[67] M. Nicholas, J. S. Perkins, G. J. Orris, L. T. Fialkowski, and G. J. Heard, “Environmental inversion and matched-field tracking with a surface ship and an l-shaped receiver array,” J. Acoust. Soc. Am. **116**(5), 2891–2901 (2004).

[68] S. E. Crocker, P. L. Nielsen, J. H. Miller, and M. Siderius, “Geoacoustic inversion of ship radiated noise in shallow water using data from a single hydrophone,” J. Acoust. Soc. Am. **136**(5), EL362–EL368 (2014).

[69] S.-H. Byun, C. M. Verlinden, and K. G. Sabra, “Blind deconvolution of shipping sources in an ocean waveguide,” J. Acoust. Soc. Am. **141**(2), 797–807 (2017).

[70] R. A. Koch and D. P. Knobles, “Geoacoustic inversion with ships as sources,” J. Acoust. Soc. Am. **117**(2), 626–637 (2005).

[71] S. A. Stotts, R. A. Koch, S. M. Joshi, V. T. Nguyen, V. W. Ferreri, and D. P. Knobles, “Geoacoustic inversions of horizontal and vertical line array acoustic data from a surface ship source of opportunity,” IEEE Journal of Oceanic Engineering **35**(1), 79–102 (2010).

[72] K. L. Gemba, J. Sarkar, B. Cornuelle, W. S. Hodgkiss, and W. Kuperman, “Estimating relative channel impulse responses from ships of opportunity in a shallow water environment,” J. Acoust. Soc. Am. **144**(3), 1231–1244 (2018).

[73] H. Niu, Z. Gong, E. Ozanich, P. Gerstoft, H. Wang, and Z. Li, “Deep-learning source localization using multi-frequency magnitude-only data,” J. Acoust. Soc. Am. **146**(1), 211–222 (2019).

[74] C. M. Bishop, *Neural networks for pattern recognition* (Oxford university press, 1995).

[75] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*, Ieee (2009), pp. 248–255.

[76] A. Kendall, Y. Gal, and R. Cipolla, “Multi-task learning using uncertainty to weigh losses for scene geometry and semantics,” in *Proceedings of the IEEE conference on computer vision and pattern recognition* (2018), pp. 7482–7491.

Index