Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Fredrik Almås

Bottom-detection in Doppler Velocity Logs using Recurrent Neural Networks on an embedded platform

Master's thesis in Engineering Cybernetics
Supervisor: Jo Arve Alfredsen
Co-supervisor: Waseem Hassan
June 2022

NTNU
Norwegian University of
Science and Technology

Fredrik Almås

Bottom-detection in Doppler Velocity Logs using Recurrent Neural Networks on an embedded platform

Master's thesis in Engineering Cybernetics
Supervisor: Jo Arve Alfredsen
Co-supervisor: Waseem Hassan
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

Increased use of autonomous underwater vehicles (AUVs) creates a growing demand for underwater tracking solutions. Hence the importance of providing robust and accurate long-term subsea navigation systems has never been greater. One of the most common navigation methods employed by underwater vehicles consists of estimating position from Doppler Velocity Log (DVL) and inertial navigation (IN) system data. The DVL sonar measures the relative velocity between an instrument and the bottom of a body of water by estimating the Doppler shift of multiple acoustic beams that point in different directions. In order for DVLs to measure the velocity accurately it requires advances bottom tracking technology. Current bottom tracking algorithms are heuristic algorithm that depend on history, previously known features and requires time-consuming tuning for each instrument.

A recurrent neural network (RNN) approach to the bottom tracking task was proposed. Seven different models were designed, tuned, optimised and tested on amplitude time series data recorded by Nortek on various Nortek DVL instruments. The models performed binary classification to find time samples corresponding to bottom echoes. The trained RNN models display competitive results as they correctly detect bottom samples with an average accuracy of 98.22%, across all models. The best performing model achieved an accuracy of 98.78%, tested on 247 352 unseen examples. These models successfully detect bottom echoes in random sequences of samples stemming from seven different Nortek DVLs without tuning for any specific instrument.

Additionally an improved metric for interval detection was proposed. Due to only small parts of each signal corresponding to the bottom echo a lot of imbalance was present in the data. The overlap metric is proposed as a solution, as it only considers how much of the detected bottom echo overlaps with the true labels. The average overlap across all models was 82.66% with a maximum of 87.74%.

All seven RNN models were converted into tensorflow lite (tflite) models, and quantised with float 16 conversion, resulting in a total of 14 tflite models. These models were shown to have similar detection results to the original models, with some fall in performance of the two best performing models. The tflite models were transferred to a DIGI-CC8MNDVK board and tested for inference times and memory footprints. They were tested on 5000 examples from the test dataset, using CPU computations with the XNNPACK delegate. All models performed inference faster than the maximal ping frequency for Nortek instruments of 8 Hz. The fastest model used $5.968ms$ per example and the slowest used $51.45ms$. Inference times were invariant to the float 16 quantisation and largely depended on RNN model complexity.

Sammendrag

Økt bruk av autonome undervannsfarkoster (Autonomous Underwater Vehicle, AUV) skaper en økende etterspørsel etter systemer for undervannsnavigasjon. Derfor har viktigheten av å tilby robuste og nøyaktige langsiktige systemer for undervannsnavigasjon aldri vært større. En av de vanligste navigasjonsmetodene som brukes av undervannsfarkoster består av å estimere posisjon ved bruk av en Doppler Velocity Log (DVL) kombinert med treghetsnavigasjon (Inertial Navigation, IN). DVL-ekkoloddet måler den relative hastigheten mellom instrumentet og bunnen av en vannmasse ved å estimere Dopplerskiftet mellom flere akustiske stråler som peker i forskjellige retninger. For at DVL-er skal måle hastigheten nøyaktig, krever det avansert bunndeteksjon. Vanlige bunndeteksjonsalgoritmer er heuristiske algoritmer som avhenger av historikk, kjente terskelverdier og krever tidkrevende kalibrering for hvert instrument.

En tilnærming som bruker tilbakevendende nevrale nettverk (Recurrent Neural Networks, RNN) til bunndeteksjonsoppgaven ble foreslått. Syv forskjellige modeller ble designet, optimert og testet på tidsseriedata av akustiske amplitudesignal, tatt opp med diverse Nortek DVL-instrumenter. Modellene utførte binær klassifisering for å finne bunnekko i akustiske signaler. De trente RNN-modellene viser konkurransedyktige resultater, ettersom de oppdager bunnekko med en gjennomsnittlig nøyaktighet på $98,22\%$, på tvers av alle modeller. Modellen som presterte best oppnådde en nøyaktighet på $98,78\%$, testet på 247 352 usette eksempler. RNN modellene oppdaget bunnekko i tilfeldige sekvenser av signaler som stammer fra syv forskjellige Nortek DVL-er uten tuning for et spesifikt instrument.

I tillegg ble det foreslått et forbedret måltall for intervalldeteksjon. På grunn av at kun små deler av hvert signal tilsvarer bunnekkoet var det mye ubalanse i dataene. Overlapp er foreslått som en løsning. Overlapp vurderer kun hvor mye av det detekterte bunnekkoet som overlapper med det sanne ekkoet. Gjennomsnittlig overlapp på tvers av alle modeller var $82,66\%$ med et maksimum på $87,74\%$.

Alle de syv RNN-modellene ble konvertert til tensorflow lite (tflite)-modeller, og kvantisert med float 16-konvertering, noe som resulterte i totalt 14 tflite-modeller. Det ble vist at disse modellene hadde tilsvarende deteksjonsresultater som de originale modellene, med noe fall i ytelsen til de to modellene som i utgangspunktet hadde best ytelse. Tflite-modellene ble overført til et DIGI-CC8MNDVK-kort og testet for deteksjonstid og minneavtrykk. De ble testet på 5000 eksempler fra testdatasettet, ved å bruke CPU-beregninger med XNNPACK-delegatet. Alle modellene utførte deteksjon raskere enn den maksimale ping-frekvensen for Norteks instrumenter, på 8 Hz. Den raskeste modellen brukte $5,968ms$ per eksempel og den tregeste brukte $51,45ms$. Deteksjonstidene var invariante for float 16-kvantiseringen og kom i stor grad an på RNN-modellens kompleksitet.

Preface

This master thesis was conducted at the Norwegian University of Science and Technology (NTNU) and submitted to the Department of Engineering Cybernetics in the spring of 2022. An external company, Nortek AS, provided the project description and necessary data and guidance throughout the process. Nortek specialize in underwater acoustic instruments, applying the Doppler principle to measure underwater properties and provide sensor data for use in subsea navigation.

Throughout the research, implementation and writing of this thesis I have received massive assistance from my advisors at Nortek. I would like to thank Waseem Hassan and Audun Ramstad. They have provided necessary data, external tools, introductory domain knowledge and ideas, in addition to thoroughly answering any questions at short notice, even on weekends and during holidays. I am especially grateful to Waseem for always being available to help. I also would like to thank my supervisor at NTNU, Jo Arve Alfredsen for help and guidance with access to NTNU tools used for the experiments.

Contents

List of Figures

List of Tables

Abbreviations

Adam - Adaptive moment estimation

AI - Artificial Intelligence

ANN - Artificial Neural Network

AP - Acoustic Positioning

AUV - Autonomous Underwater Vehicle

BA - Binary Accuracy

BCL - Binary Crossentropy Loss

DL - Deep Learning

DNN - Deep Neural Network

DR - Dead Reckoning

DVL - Doppler Velocity Log

ES - Early Stopping

FC - Fully Connected

GPS - Global Positioning System

GPU - Graphics Processing Unit

IN - Inertial Navigation

LBL - Long Baseline

ML - Machine Learning

MSE - Mean Squared Error

NPU - Neural Processing Unit

ReLU - Rectified Linear Unit

RL - Reinforcement Learning

RNN - Recurrent Neural Network

SBL - Short Baseline

SDK - Software Development Toolkit

SL - Supervised Learning

SNR - Signal to Noise Ratio

Tanh - Hyperbolic Tangent

Tflite - Tensorflow lite

UL - Unsupervised Learning

USBL - Ultrashort Baseline

VM - Virtual Machine

Chapter 1

Introduction

In recent years AUVs have seen an increase in popularity. As such the demand of providing accurate positional estimates underwater has increased. The DVL is commonly used as a velocity-sensor in this context [1]. DVLs are highly dependent on their bottom tracking to provide accurate velocity measurements. This thesis aims to provide recurrent neural networks for robust sea bottom-detection in signals stemming from a Nortek DVL1000-4000 m., DVL1000-300 m., DVL500-6000 m., DVL500-300m., Speed Log, Compact500, and DVL333. A reccurent neural network model that successfully performs bottom-detection on various instruments will avoid time-consuming work of tuning new instruments, while possibly providing increased detection in signals that are ignored in current algorithms. The sequential nature of bottom tracking makes recurrent networks highly relevant. The models will function independently of external measurements, and only look at short sequences of acoustic data. In addition these models could run on embedded devices taken underwater. This possibility of running on-device inference has the potential to result in reduced latency, reduced energy costs, as well as increased reliability and security.

1.1 Motivation

The demand for accurate, reliable, and long-term subsea navigation are ever increasing. Use of marine environments for agriculture, energy and harvesting of resources all require tools that depend on accurate navigation in areas where the ordinary global positioning system (GPS) cannot be used. A common solution to determine the position of an underwater vehicle below the surface is acoustic positioning (AP) or inertial navigation (IN) systems. However to enable individual operation of subsea vehicles modeling of the environment and necessary sensor measurements are required. One of the tools used for velocity measurements underwater is the DVL. DVLs heavily rely on accurate bottom tracking algorithms in order to provide reliable measurements. This is performed using complex heuristic algorithms which require time-consuming tuning for each DVL instrument. Shifting to a neural network assisted bottom detection model enables automatic feature-extraction to observe patterns and relations, currently not regarded in the heuristic algorithm. Ad-

ditionally a method that avoids tuning of parameters will reduce the workload and provide generic performance over a set of different instruments. Such models could also run on GPUs or NPUs on underwater appliances to provide efficient detections with low latency and low energy costs.

1.2 Literature review

Existing bottom-detection solutions were identified to observe common practice in this research field. In addition modern attempts on using DL within the field of subsea navigation was explored. As the models presented are recurrent neural networks, an investigation into general time series classification (TSC) with RNNs was conducted.

1.2.1 Bottom detection algorithms

Skatvedt [2] performed bottom tracking on DVL signals by using neural networks for binary classification and regression. Skatvedt proposes two models, a one-dimensional convolutional neural network and a fully connected neural network. Both models were trained for both tasks, i.e. binary classification and regression. The bottom tracking was performed in conjunction with Nortek, on Nortek DVL data. On test data these models achieved a 98.03% and 99.35% detection rate respectively.

Nortek DVLs obtain state-of-the-art velocity measurements, by utilizing an industry developed bottom tracking algorithm. In 2016 Hegrenæs et.al. [3] presented trial results obtained by running tests using the Kongsberg Maritime AUV, HUGIN with Norteks DVL 500 kHz. However the bottom tracking algorithm used is referenced to unpublished work.

Taudien [4] proposed a bottom detection algorithm related to the bottom tracking technology used by Nortek. This is technology based upon correlation and covariance coefficients. These methods require transmitted signals consisting of repeated code units with specified bandwidth. In addition a hybrid algorithm was presented, composed of bottom detection using the covariance and correlation coefficients, combined with an adaptive narrowband filter to maximize signal to noise ratio (SNR). The detection rate of the hybrid algorithm was obtained using a 614 kHz Pathfinder DVL, detecting bottom echoes with a 90% detection rate at about 155 m. altitude, which additionally was a 74% increase in altitude compared to a conventional bottom-mode algorithm on the same instrument [4]. Taudien also provided a summary of previously published boundary detection technologies. He covers filtering following peak-detection [5], separate optimisation of matched filter and transmit waveform [6], beamforming and amplitude-and interferometric-detection in multibeam sonar using multiple subarrays [7], as well as logarithmic power detection [8].

1.2.2 Deep Learning for subsea navigation

Machine Learning (ML) is a familiar concept within the research field of subsea navigation. The company WaterLinked offers a DVL that utilizes Artificial Intelligence (AI) technology to estimate velocity directly [9]. Neural networks have also been applied to provide outlier detection [10], and on forecasting velocity at time-steps to ensure reliable velocity predictions for short periods in cases of DVL malfunction [11]. Additionally neural networks have been proposed to provide assistance in sensor-fusion [12]. Furthermore [13] proposed a modified fully convolutional neural net have as a fault-detection method in multi-source navigation systems.

1.2.3 Time series classification with recurrent neural nets

RNNs have seen some use in classification of time series data. Smirnov et.al. [14] experimented with 10 RNNs in the UCR Time Series Classification Archive [15], which is a time series data collection for research. They used both LSTM as well as simple RNN layers in their models. An earlier example of RNNs for time series classification is detailed in [16], where the dynamic behaviour of RNNs is utilized to categorize input sequences into different specified classes.

1.3 Research objectives

Previous works have performed bottom detection using neural networks, however no published work on utilizing recurrent neural networks to perform bottom-detection in acoustic signals was found. We attempt to use sequential data from time series DVL data to achieve bottom detection to a high level on all signals, without filtering out difficult samples. It is hypothesised that sequential information could help discern bottom echoes in previously misdetected echo signals. Furthermore a new metric for evaluating bottom detection is presented, in order to address problems of using traditional deep learning metrics on unbalanced data. Additionally the possibility of running on-device inference should be explored, with a comparison of bottom detection performance as well as an evaluation of time and memory constraints. The guiding questions governing the research can be stated as:

- Will recurrent neural networks perform bottom detection exceeding or matching the performance of previous neural network assisted methods in processed amplitude signals?
- Is the proposed Overlap metric a better alternative than the traditional binary accuracy and binary cross-entropy for evaluating detection of bottom echoes in DVL time series data?
- Can recurrent neural networks be converted and used for inference on an embedded device without significant drops in performance?

- Are recurrent neural networks a viable alternative for on-device bottom detection inference during real-time subsea navigation operations?

1.4 Outline

This thesis is comprised of 5 chapters. Chapter 1 covers some background and motivation for the creation of new bottom-detection methods for DVLs. Moreover it reviews various published works within themes relevant to the thesis, and establishes the objectives and research questions addressed. Relevant background theory on subsea navigation, the DVL instrument, deep learning, recurrent neural networks as well as embedded inference are covered in chapter 2. Chapter 3 describes the bottom detection task, what kind of data and data pre-processing was used, and implementation and design of neural networks based on two different types of recurrent layers, which performed bottom-detection. In addition the chapter goes over execution of inference on an embedded device, as well as the tools used for all experiments. Results and discussion of the results are provided in chapter 4, together with a proposed metric for correct bottom interval detection. Finally conclusions and suggestions for future work are presented in chapter 5. The appendices contain model descriptions, loss plots, quantitative results, examples for qualitative analysis and finally embedded inference results.

Chapter 2

Theory

The theory chapter consists of an introduction on subsea navigation, followed by an overview of the DVL-instrument and the DVLs bottom detection algorithm. In addition basic theory on deep learning and neural networks are presented, including a description of RNNs and embedded ML. The chapter aims to present background information needed to understand the experiments, as well as the reasoning behind the experiments detailed in this thesis.

2.1 Subsea Navigation

Subsea navigation technology includes various methods of tracking the position of objects underwater. It differs from land-based navigation technology which mainly relies on GPS, with highly accurate satellite references to provide positional estimates. The damping of electromagnetic signals in liquids results in rapid attenuation of high frequency signals, preventing the signals from propagating long distances underwater [17]. GPS transmits and receives in microwave-frequencies, leading to highly attenuated signals underwater. Thus positional data in subsea navigation must be provided by physical model-based methods, for modelling of physical environments and estimation of new positions based on measured displacements.

Traditionally subsea localization and navigation technologies have been based on AP systems or IN systems [18]. Recent research has explored 'hybrid' navigation systems that utilize sensor-fusion to improve state estimation further, in order to avoid some of the shortcomings of traditional navigation solutions [19].

2.1.1 LBL, SBL and USBL acoustic positioning systems

AP systems use sound waves to estimate positions in reference to stationary transponders or transponders on the surface, located by GPS. The physical properties of sound allows fast propagation and slow attenuation in water, enabling sound waves to travel long distances underwater [17]. These properties encourage underwater

vehicle navigation to use acoustic transponders. The AP systems can be categorized as Long Baseline (LBL), Short Baseline (SBL) or Ultrashort Baseline systems (USBL) [19].

LBLs use networks of transponders on the sea bottom in fixed, known positions. These are often used at locations where repeated operations require navigation. The network of transponders typically surrounds the area of interest, and can be used to determine positions within the deployed network by triangulation. These systems generally have high accuracy. However they are generally expensive and operationally demanding, due to the setup needed as well as requiring periodical replacement of transponder batteries [1].

USBL and SBL systems offer greater flexibility, by using a following vehicle on the water surface. The reference vehicle provides a mobile GPS surface reference. Based on the GPS reference the position of a subsea vehicle is estimated by measuring the phase-shift between the elements in USBL systems, and the position between the transceiver and the transponder array in SBL systems. These systems lack the same level of accuracy as LBL systems due to refraction in the water [1]. Sound transmission over long distances travels along a path determined by the speed-of-sound profile of the water column and its boundaries. Non-uniform speed-of-sound profiles are always present, and when sound travels through mediums with different speeds of sound it will refract.

2.1.2 Inertial Navigation systems

Subsea navigation is commonly based on Dead Reckoning (DR). With DR position is estimated from an initial starting-point by adding the distance traveled in a measured direction. IN systems are commonly used as advanced DR-devices, where motion and direction is measured using integrated sensor-packages [19]. By integrating measured velocity the displacement can be found, shown in equation 2.1.1. Adding the displacement to the current position results in a new position estimate, which can be used for the next estimation.

$$d(t) = \int v(t)dt = v * t \tag{2.1.1}$$

An IN system is commonly used as an advanced DR device, consisting of accelerometers paired with rotational- and heading sensors to measure angular acceleration and velocity. Using DR with IN systems was one of the earliest approaches for autonomous underwater vehicle (AUV) navigation [20], where integration of accelerometer output yields velocity and displacement measures, as shown in equation 2.1.2.

$$d(t) = \int v(t)dt = \int\int a(t)d^2t \tag{2.1.2}$$

This method of integrating sensor output does not require transmission or receiving

of signals, and uses inexpensive sensors [21]. However IN systems are unable to detect drift caused by currents and internal sensor-inaccuracies, leading to an unspecified bias [19]. Double integration of this time-varying bias introduces an error in the position estimates, which grows over time. The error can be counteracted by combining the IN system with direct, external and accurate velocity measurements relative to the sea bottom. A Kalman Filter can then be used to fuse the different data sources in order to estimate position [1].

Figure 2.1.1 displays a sensor fusion module featuring the commonly used Kalman filter. Merging and processing the IN system and additional sensor data in with a Kalman filter minimizes the errors, by introducing increased reference comparison. The Kalman filter enhances strengths of the signals and attenuates weaknesses, providing optimal combinations of the sensor data.

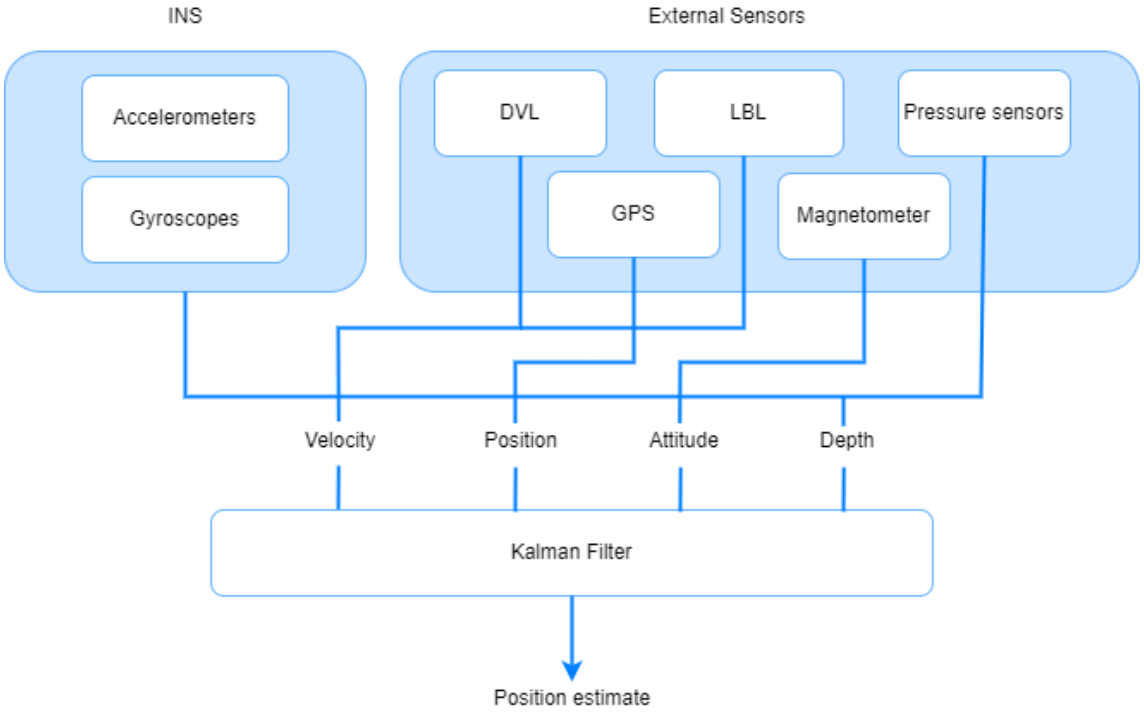


Figure 2.1.1: An overview of an IN system using a Kalman filter for sensor combination, providing positional estimates to an underwater vehicle. The illustration is based on Figure 2.1.2 in [2].

2.2 Doppler Velocity Log

DVLs are as mentioned commonly used in order to provide accurate data for subsea navigation. A DVL estimates velocity relative to the sea bottom. By sending a long pulse, a ping, along a minimum of three acoustic beams, each pointing in a different direction, accurate velocity measures are achieved [1]. The instrument considers the phase shift of the echo coming from the sea bottom, utilizing the Doppler shift from all beams to calculate the corresponding velocity. It should be noted that the DVL has to be within range of the bottom to maintain bottom tracking.

2.2.1 Norteks DVL bottom-tracking

The accuracy of DVL measurements is closely related to the quality of its bottom-tracking algorithm, as any error in the bottom tracking algorithm will result in a velocity measurement error. Hence bottom tracking is closely related to the entire navigation systems performance. Bottom-tracking technology typically detects the bottom by observing an amplitude derived from the correlation between the received echo and the transmitted signal. However the correlation functions, filtering, and general processing of the signals are generally considered trade secrets, as good solutions improve readability of the signals [2].

Nortek utilize a complex state-of-the-art detection of threshold-values on signals provided by a filtered and processed amplitude for bottom-detection. An example of Norteks amplitude-signal is displayed in Figure 3.1.1. Bottom-detection performed on amplitude signals is commonly based on threshold evaluations and properties of previously discovered features and relations. Signal strength, amplitude, and pattern features are utilized to generate a signal score, enabling detection of the bottom. If the depth fully exceeds the range of the DVL the instrument will measure velocity relative to water, known as water-tracking. However water-tracking is prone to significant errors, and the accuracy of the measurements is consequently highly dependent on the range of the acoustic beams [22].

2.3 Deep Learning

Deep learning (DL) is a subclass of ML, which in turn is a subset of AI. ML is based on the idea of creating algorithms that learn from and can make predictions on data. A common definition of ML comes from [23]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E"

DL uses interconnected artificial neurons to form complex structures, called artificial neural networks (ANNs). The ANNs mimic the way biological learning systems work. ANNs are built out of a interconnected set of simple units, where each unit takes a number of real-valued inputs (possibly the outputs of other units) and produces a single real-valued output (which may become the input to many other units) [23]. The structure of the ANNs allows them to approximate any function provided sufficiently many units [24], however some tasks are more suited for DL. According to [23] ANNs, and DLs, are appropriate for problems that fulfil the following:

- Instances are represented by many attribute-value pairs.
- The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.

- The training examples may contain errors.
- Long training times are acceptable.

Essentially DL excels at tasks where you have a lot of data examples, with real or discrete inputs and outputs and you have time to train complex models. In addition DL models are robust to noise in the training data. Long training times can be one of the worst drawbacks when using ANNs. Training can take anywhere from a few minutes to several days, weeks or months, depending on the complexity of the model and the amount of training data. Training times increase with the increase of parameters in an ANN. This increase in training time can be mitigated by the use of deep neural networks (DNNs). Deep networks can approximate the class of compositional functions as well as shallow networks but exponentially lower number of training parameters [25]. On the contrary to the long training time DNNs often have fast inference times, in other words application of the learned function is relatively fast. The inference times are usually considerably shorter than the required training time, which makes the continual use of a model significantly easier and faster than the creation and training of the model.

Furthermore DL does not require human understanding of the target function and patterns within the data. In more traditional ML algorithms human understanding and feature engineering are the grounds for the learning algorithms, however ANNs and DNNs have the potential to find functions and patterns humans do not comprehend. Feature engineering can still be used to aid the networks, but it is not needed. This point can be viewed as an advantage as well as a drawback. DL can find a better estimation of the target function, although the features used and the pattern found is often hard to understand. This often leads to a difficulty explaining why a certain result is obtained, in turn making it hard to realize how to change inputs in order to achieve a wanted result.

2.3.1 Learning paradigms

In ML there are three primary learning paradigms, supervised learning (SL), unsupervised learning (UL) and reinforcement learning (RL). These paradigms differ based on the goal, how the data is represented and how the learning process is conducted [26]. The most common form of ML, deep or not, is SL [27]. SL is also the only paradigm used for the experiments conducted in this thesis. Figure 2.3.1 illustrates the basic structure of each paradigm.

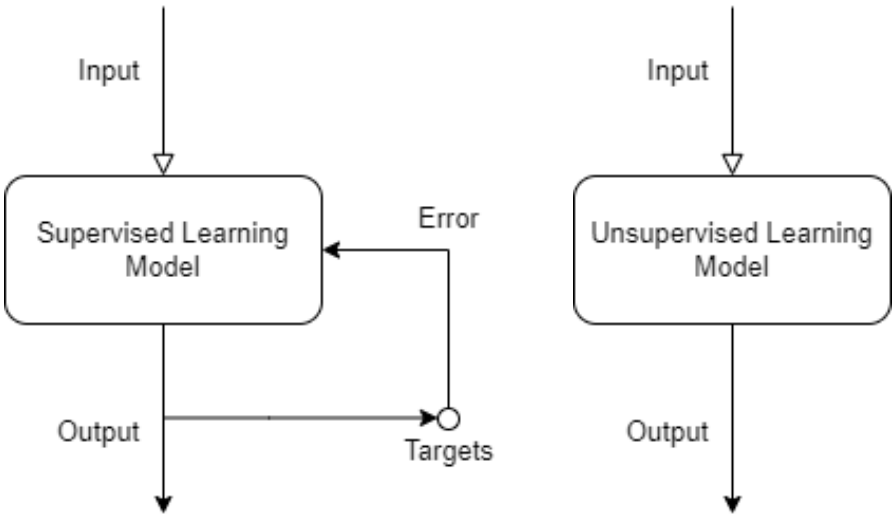
Two of the most important concepts within ML is over- and underfitting. Overfitting refers to a model learning mappings that correspond specifically to the data it is trained on, instead of general mappings that can be used for new, unseen data. Underfitting is the opposite, where a model is unable to learn crucial general patterns that are needed to make good predictions. In DL underfitting is easily prevented by having enough layers and wide enough layers, and is generally not a problem. Overfitting however can be troublesome and has to be considered and addressed.

As a general rule three datasets are needed when performing DL. One for training,

one for validation and one for testing. The training data usually contains most of the examples and is the data used for learning. Validation data is not used for training, instead it is used to assess changes made to the model and to compare how the model performs compared to on the training data. The test data is kept out of all training and model design, to only be used to evaluate how general the model is. As mentioned overfitting is when the model overly learn patterns in the training data. In order to monitor this the validation data can be used as a comparison. If results on the training and validation data do not match up there is a high likelihood of overfitting. In this case the model needs to be changed to correct the issues. The most common ways to combat overfitting is to reduce the model complexity, or to add regularisation to the model. Dropout is one of the regularisation techniques used widely in deep neural networks [28]. It essentially ignores some units randomly during the training phase of the network, which prevents complex co-adaptations among units.

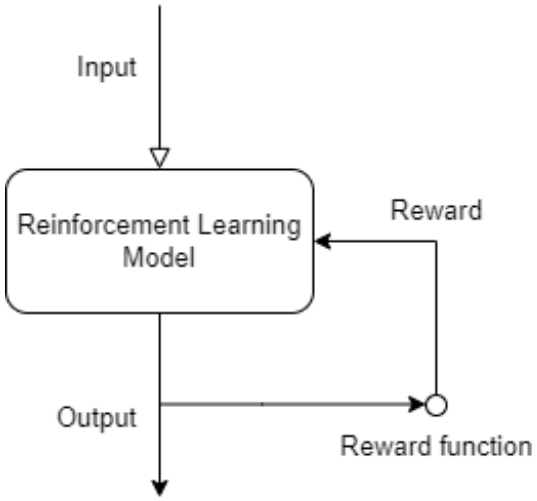
[29] defines SL as a ML paradigm for acquiring the input-output relationship information of a system based on a given set of paired input-output training samples. As the output is regarded as the label of the input data or the supervision, an input-output training sample is also called labelled training data, or supervised data. The labelled data is used to measure the error between a output produced for each category based on the input and the correctly labeled output values. As shown in subfigure 2.3.1a the machine then modifies its internal adjustable parameters, its weights, to reduce this error. To properly adjust the weights an optimisation algorithm is used to minimize the error [27]. After training and adjusting the weights optimally the model has learned a mapping between inputs and outputs, which lets it infer new outputs from new unseen inputs. In order to learn this mapping a dataset of pre-labelled data is required. Hence the main problem of SL is the time and cost of labelling large datasets before training can start.

UL eliminates the labelling problem, as it uses no labelled data. With UL the machine looks for intrinsic patterns within the data in order to create groups and clusters of data points that appear similar. The clusters are found by using algorithms that reward high internal similarity within and high diversity between clusters [26]. RL relies on a feedback-loop provided by rewards or penalties on actions performed by an agent in an environment. The agent creates a mapping from previous feedback and applies it to the current observation of the environment to resolve further action, thus learning how to maximize rewards and minimize penalties [26]. UL and RL are displayed in subfigures 2.3.1b and 2.3.1b respectively.



(a) Supervised learning

(b) Unsupervised learning



(c) Reinforcement learning

Figure 2.3.1: Illustration of the three main learning paradigms of ML. Subfigure 2.3.1a displays the basic structure of SL, subfigure 2.3.1b displays UL and RF is shown in subfigure 2.3.1c. The illustration is based on Figure 1 in [30]

2.4 Neural network fundamentals

As mentioned ANNs were initially constructed to resemble an overly simplified structure of biological neural networks. They work by communicating signals between layered processing units over a large number of weighted connections [31]. The processing units are artificial neurons and constitute the building blocks of an ANN. Each unit has an associated input weight, bias and an activation function. The relation between inputs and outputs from a single unit with N inputs $\mathbf{x}_i$, weight matrix $\mathbf{w}$, bias vector $\mathbf{b}$ and activation function F is described in equation 2.4.1. The same relation is illustrated in figure 2.4.1.

$$[H]\mathbf{y} = F\left(\sum_{i=1}^{N} \mathbf{x}_i * \mathbf{w}_i + b_i\right) \tag{2.4.1}$$

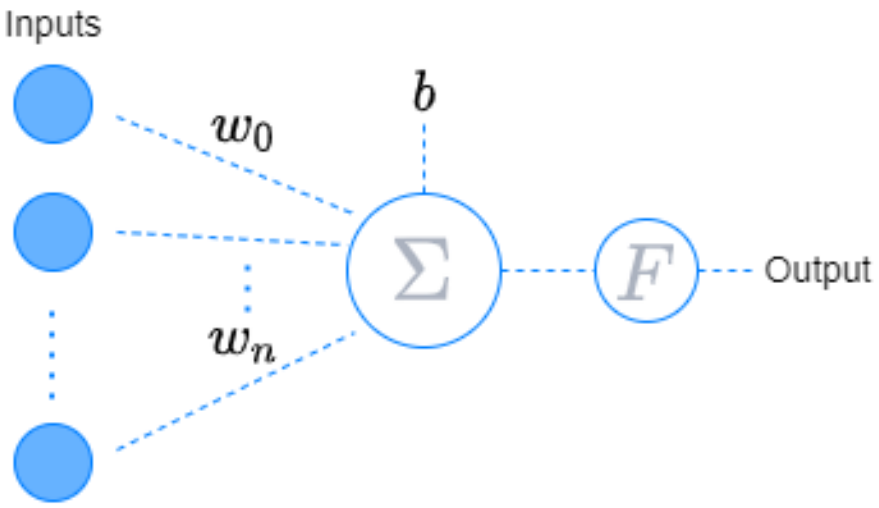


Figure 2.4.1: Structure of a single artificial neuron, displaying the transformation of inputs through weights $\mathbf{w}_i$, bias $\mathbf{b}$ and activation function F. The illustration is based on Figure 2 in [32].

These neurons can be layered into several layers, increasing the complexity of the ML model. The number of connections, nodes, and layers influences the transformation, and has to be determined by an engineer or by optimising over a parametric region of interest. The feed-forward neural network performs forward propagation. It is regarded as a nonlinear mathematical function, where the weight parameters influence the transformation [32]. Inputs are fed forward through the network and transformed by every layer. A standard feedforward network is shown in figure 2.4.2. The weight parameters are adjusted by using gradient descent algorithms in order to minimize a predetermined loss function. Common loss functions are Mean Squared Error (MSE) and Binary Crossentropy Loss (BCL), detailed in equations 2.4.2 and 2.4.3 respectively. After propagating the input signals through the network a final output is traversed back using back-propagation. This allows calculation of gradients used to adjust the weights in order to reduce the output loss. The adjustment is done by using an algorithm based on steepest-descent minimization. Traditionally stochastic gradient descent (SGD) was used, however newer algorithms like the Adaptive Moment Estimator (Adam) [33] has surfaced. Adam combines adaptive learning rates and momentum on the gradient descent in order to speed up training significantly for most tasks.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{y}_i - \hat{\mathbf{y}})^2 \tag{2.4.2}$$

$$BCL = -\frac{1}{N}\sum_{i=1}^{N} y_i log(\hat{y}) + (1 + y_i) log(\hat{y} - 1) \text{ where } \begin{cases} y_i \in \{0, 1\} \\ \hat{y} = \frac{1}{1+e^{-x}} \end{cases} \tag{2.4.3}$$

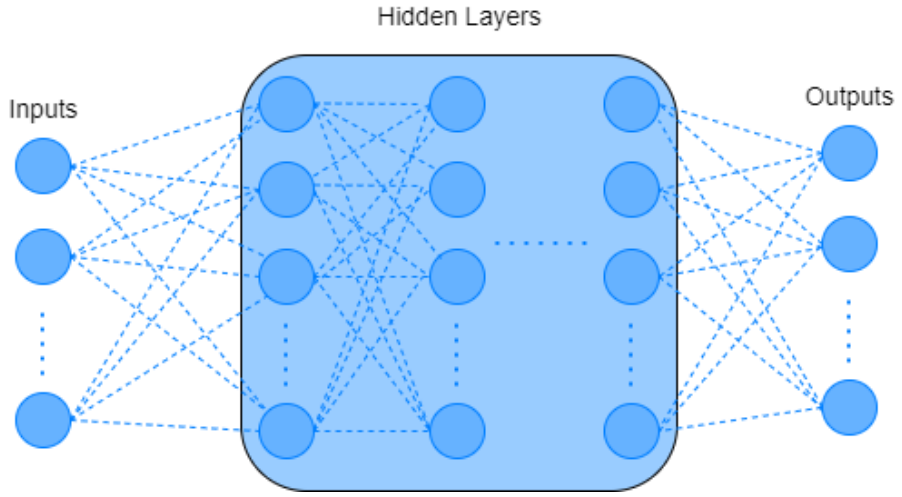
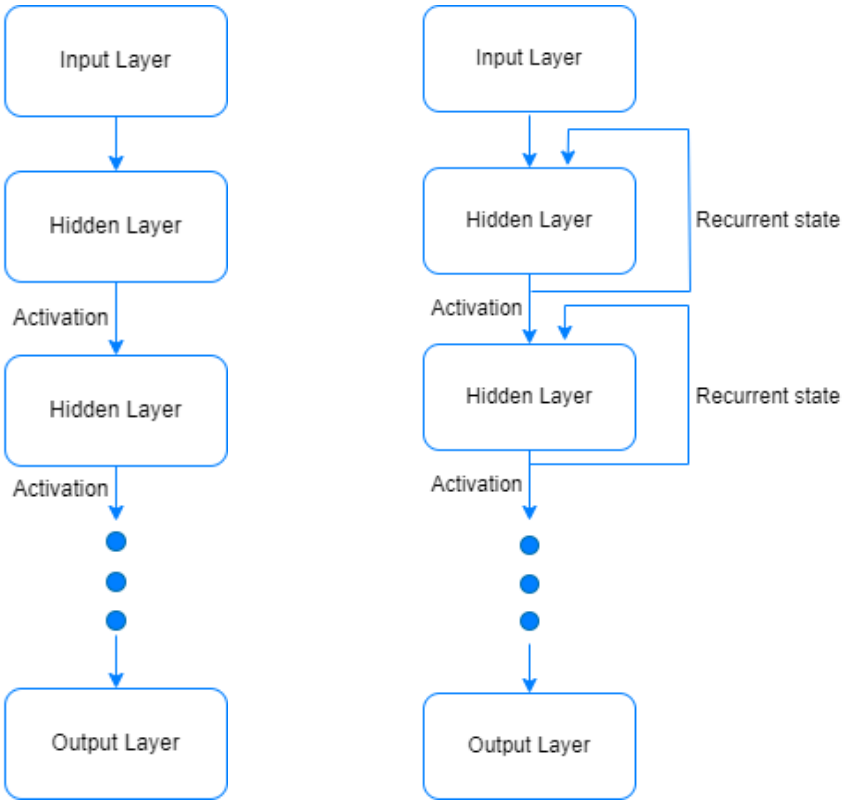


Figure 2.4.2: Structure of an ANN with inputs, outputs and several hidden layers. The illustration is based on Figure 2.3.3 in [2].

2.5 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are neural network architectures which are mainly used to detect patterns in sequences of data [34]. They are commonly used on data such as handwriting, genomes, text or numerical time series. By using cycles to pass information back to itself each layer in a RNN has some form of short-term memory. This enables them to extend the functionality of DNNs to also take into account previous inputs [34]. Furthermore the functionality allows the networks to handle sequences of variable length. The structural differences between DNNs and RNNs are illustrated in figure 2.5.1. However basic RNNs have been shown to struggle to capture long-term dependencies because the gradients tend to either vanish (most of the time) or explode [35].

One approach to combat vanishing and exploding gradients in RNNs has been to design more sophisticated activation functions [36]. Two main "improved" units resulted from this, the long short-term memory unit (LSTM) and the gated recurrent unit (GRU). RNNs employing either of these recurrent units have been shown to perform well in tasks that require capturing long-term dependencies see [37] and [38]. This thesis details experiments using GRU, while Nortek has explored some LSTM models.



(a) Feedforward network

(b) Recurrent network

Figure 2.5.1: Illustration of the structure and data flow in DNNs and RNNs. Subfigure 2.5.1a shows the structure of a standard feedforward DNN. RNNs differ from DNNs by the recurrent states shown in subfigure 2.5.1b. The illustration is based on Figure 1 in [34].

2.6 Embedded Machine Learning

On device inference with ML is becoming increasingly viable, due to faster embedded devices with more memory and computing power. In addition many devices have a graphical processing unit (GPU) or dedicated neural processing units (NPUs). These units specialize in parallel computations, making neural network computation significantly more efficient. Compared to traditional cloud computing, embedded computing has many unique advantages; including low latency, increased reliability, running with limited or no connectivity, reduced cost and privacy as well as security [39].

The most common framework for embedded ML is tflite [40]. Tflite allows users to deploy tensorflow models on embedded and edge devices, by creating tflite models or by converting existing models. This conversion can include optimisations such as quantisation, which reduces model size and improves latency with minimal loss of accuracy [41].

Chapter 3

Recurrent Neural Networks for bottom detection

As established in section 2.2 subsea navigation using a DVL is highly dependent on its bottom-tracking algorithm. [2] explored the possibility of bottom-tracking using a DNN and a convolutional neural network (CNN). Due to the sequential nature of the time series sensor data we assume that RNNs could be a suitable tool for the bottom-tracking task. The extra information in knowing where the sea bottom appeared in previous pings could improve upon the bottom-tracking achieved through DL. Furthermore converting models into tflite and running inference on an embedded device could lead to improved latency and eliminates the need for communication with surface computers.

This chapter aims to present the steps taken in order to achieve the results presented in the thesis. Including the methodology of designing, implementing and optimising RNNs for bottom-tracking on amplitude-data stemming from various Nortek DVL-instruments. The RNNs were created using two different recurrent units. In addition the chapter covers converting these models into tflite, as well as running inference on a DIGI-CC8MNDVK board [42].

3.1 Problem description

Norteks DVL collects data for bottom-detection consisting of time series amplitude signals stemming from various environments. The aim was to create and train generic and accurate recurrent models to find a bottom-interval in the amplitude signals where the echo from the sea bottom appeared. Moreover the models were to be converted into tflite, exported to an embedded device and tested for inference. Nortek provided large amounts of labelled DVL data from different DVL instruments, recorded in a wide range of environments. Hence the provided data allowed RNNs to be trained with SL. Figure 3.1.1 displays an example ping before transforming the labels into binary form. The datasets from Nortek consisted of such pings before processing the data.

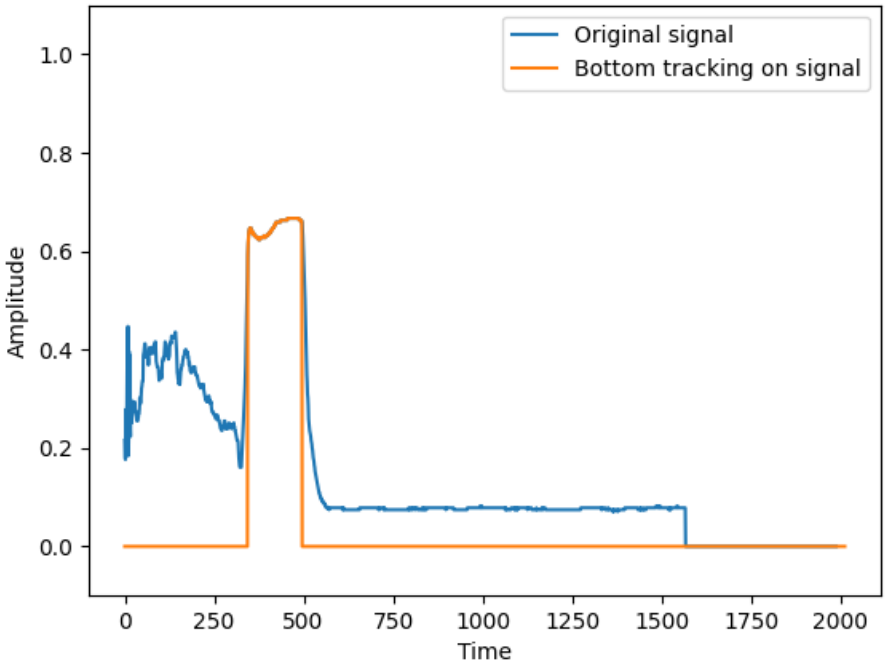


Figure 3.1.1: Example of a ping before labelling. The amplitude time series is constructed from echoes received by a Nortek DVL sensor. The bottom tracking comes from Norteks tracking algorithm.

Detecting the correct bottom-interval in the time series amplitude data was defined as a binary classification problem. Each time-sample of the signal was considered bottom or non-bottom for binary classification. The target values used to perform SL were previously constructed based on Nortek bottom-tracking algorithm results. For binary classification, binary vectors with the same length as the time series amplitude signals were created, consisting of class labels with values 0 or 1 for non-bottom and bottom pings respecively. The RNN models were trained using pre-processed time series amplitude signals as input, similar to the signal exhibited in figure 3.1.2. Outputs from the models were predictions matching the shape and form of the target labels, with predictions close to 0 are considered non-bottom and predictions closer to 1 are considered bottom samples. Figure 3.1.3 displays a labelled ping, with binary labels ready to be used for SL.

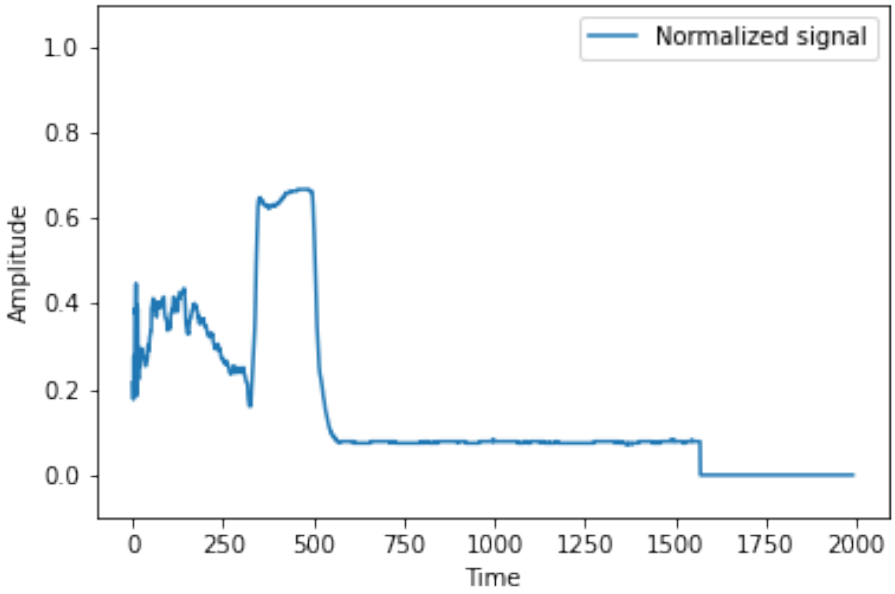


Figure 3.1.2: Example of a ping before labelling. The amplitude time series is constructed from echoes received by a Nortek DVL sensor. The bottom tracking comes from Norteks tracking algorithm.

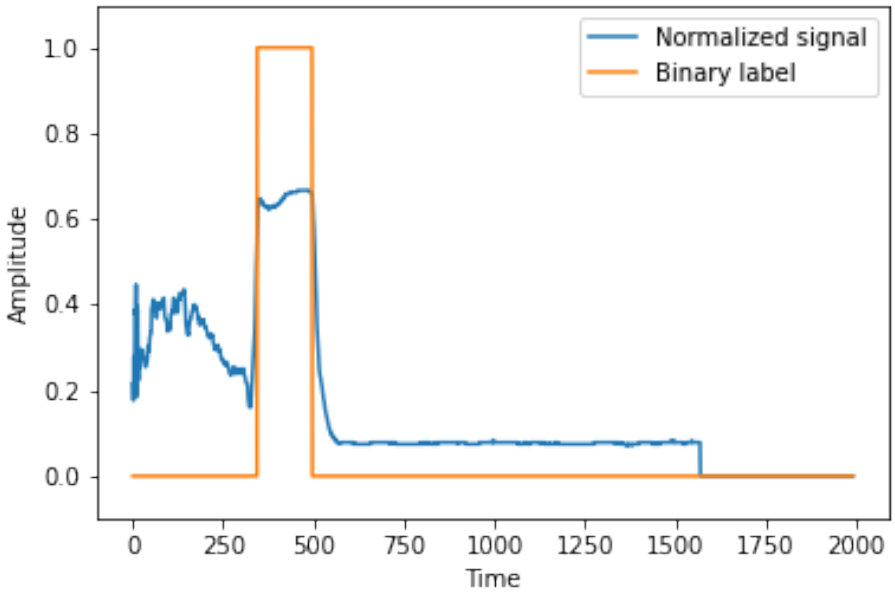


Figure 3.1.3: Example of a labelled ping, consisting of an amplitude time series from a received echo with the sample-window detected by the Nortek algorithm together with the constructed binary-and bottom-detection labels used to train the RNNs.

3.2 Data

The time series amplitude data were obtained during testing of different DVLs, conducted by Nortek. 23 datasets from different instruments and environments were provided, which consisted of a total of 1 158 639 pings. Out of these pings 91 973 had no bottom echo, making 7.94% of the signals in the data contain no bottom. Figure 3.2.1 illustrates one of these no bottom pings. With such a large part of the data being out of range from the bottom the models have the ability to learn to predict on both bottom pings as well as non-bottom pings. The datasets were collected using the Norteks DVLs; DVL1000-300 m. [43], DVL1000-4000 m. [44], DVL500-300m [45] and DVL500-6000 m. [46], Speed Log, Compact500, and DVL333. Technical specifications for each of the instruments are displayed in Table 3.2.1, as the different instruments operate within different ranges and environments. Because of this increased variety and noise within the collective dataset is introduced.

DVL instrument	Max. altitude	Min. altitude	Long-term accuracy	Max. operational depth
DVL 1000, 300 m.	75 m.	0.2 m.	±0.1% / ±0.1 cm/s	300 m.
DVL 500, 300 m.	200 m.	0.3 m.	±0.1% / ±0.1 cm/s	300 m.
DVL 1000, 4000 m.	75 m.	0.2 m.	±0.1% / ±0.1 cm/s	4000 m.
DVL 500, 6000 m.	200 m.	0.3 m.	±0.1% / ±0.1 cm/s	6000 m
DVL 333	330 m.	Unreported	Unreported	Unreported
Compact 500	175 m.	Unreported	Unreported	Unreported
Speed Log	200 m.	0.3 m.	±0.1% / ±0.1 cm/s	300 m.

Table 3.2.1: Technical specifications of the Nortek DVL instruments. The DVL 333 and the Compact 500 development, hence they are unreported.

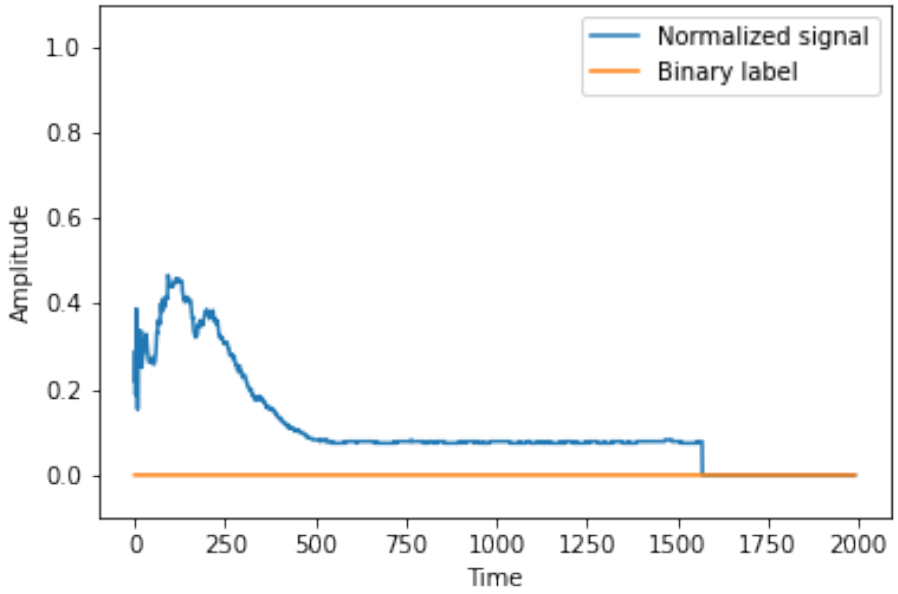


Figure 3.2.1: This is a labelled ping where the bottom is out of range for the DVL. Here the labels are 0 for the entire ping, and the model is supposed to predict values close to zero.

Any time-varying bias of the DVL can be detected by performing DR using the DVL to measure velocity and a heading sensor to measure direction, subsequently comparing the resulting positional estimates to an accurate GPS reference at certain time intervals. Norteks testing was described in [2]:

> "Testing is performed by mounting a DVL-instrument underneath the Moonpool boat, avoiding noise from the motors. A 24V power supply drives the instrument, and a computer manages the DVL via Ethernet cable, see Figure 3.2.1. The SD-card of the DVL logs the measured and processed data while the DVL transmits pings with specified bandwidth and repetition period. When receiving the echo, the transducers generate an analog signal to be amplified and filtered by an anti-aliasing filter before being digitalized at 20 MHz. Digital filters, decimating, and quadrature demodulation are further performed to filter the signal. The baseband signal is at the signal's center frequency at 500 kHz, 1000kHz, or 333kHz."

The Nortek DVLs are usually tested in the inner Oslo Fjord with depths ranging from 10 to 200 m, in Bunnfjorden, and in Horten. Speed Logs are mounted on larger vessels, the Hanna Kristina ship traveled along the Norwegian coast, and River Orashi operated in international waters.

Each time series, or ping, consists of 1990 samples. The processing, filtering, and smoothing technique used by Nortek introduce a loss of detail and a time lag in the signals. The time lag is incorporated into Norteks bottom-tracking algorithm,

implying that the true bottom window is slightly shifted to the left of where the leading edge appears in the processed signal. Generally the shift follows a pattern that varies based on the instrument. As a general rule the bottom-tracking window is expanded on the leading edge with ten samples after sample 18, and 40 samples after sample number 450 [2]. It is worth noting that the loss of detail may remove crucial information in the peaks corresponding to the bottom echo.

3.2.1 Data pre-pocessing

Preprocessing of the data was done using many of the same methods as in [2], and some datasets were used here as well as in [2]. The provided datasets contained normalized amplitude time series of different lengths. All signals were in chronological order. As a result the data had to be pre-processed to provide generalized lengths and values despite coming from separate datasets. The target values also had to be adjusted to represent target values for binary classification instead of continuous values.

First the target values were constructed by creating binary vectors. The start and end of the bottom detections reported by the Nortek bottom-tracking algorithm were detected by finding indexes of the first and final non-zero values in the time series. The indexes between the determined start and end-indexes were marked as bottom (1), while the indexes outside the interval were marked non-bottom (0). If no bottom was detected in the signal, the entire label vector was filled with zeros.

The range of the DVL impacts amplitude-signal length, the range is determined by the instrument type or pre-defined by the user. Consequently the shorter signals had to be padded with 0s to create signals with an equal length of 1990. The resulting datasets consisted of labeled matrices, where each row represented a new ping, and each column was a new sample in the time series. An example of labeling the same ping with a binary vector is shown in figure 3.2.2.

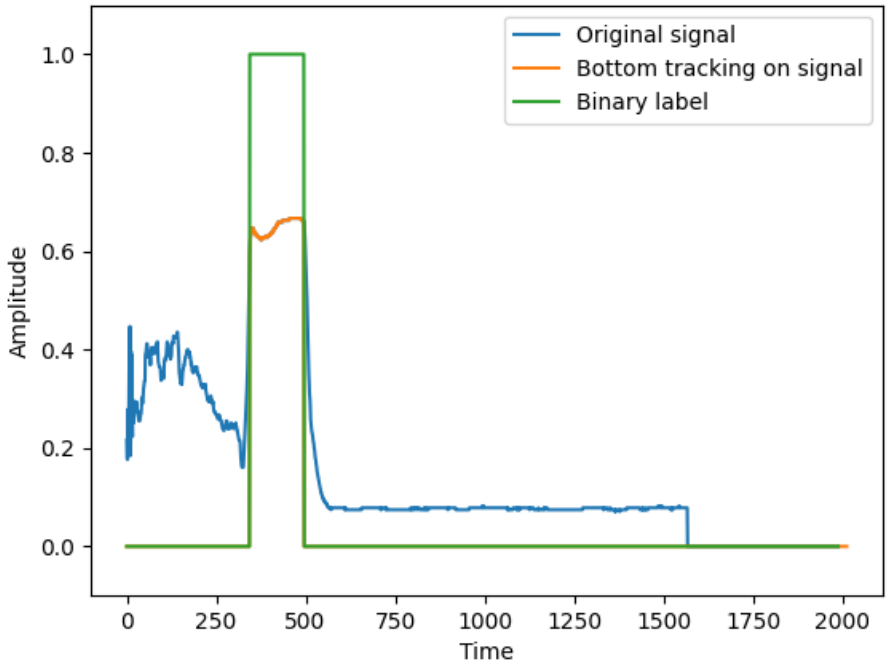


Figure 3.2.2: An example ping to illustrate the creation of binary labels from original bottom tracking on an amplitude signal.

The chronological ordering of the data was kept, in order to keep sequentiality in the data for learning recurrent patterns. However some attempts were made to create randomized input order, which promotes pattern learning instead of memorization in the RNN models. The data was split into batches and sequences, which were shuffled on batch level. The batch and sequence lengths varied between models as optimisation parameters. This approach kept the sequential properties of the data, whilst introducing randomness.

20 of the datasets were used to create training and validation sets, while 3 datasets were kept completely separate as a test set. The test set was extracted from separate datasets in order to simulate a real world application where new unseen data comes in the form of new datasets. Hence the results achieved in this thesis are more likely to represent the true generalization capabilities of presented models. This split kept 928 727 pings for training and validation, and 247 352 pings for testing. Each of the 20 datasets used for training were split into 80% for training and 20% for validation, such that the validation set is representative of the entire training data distribution.

3.3 Evaluation Matrix

Each model was evaluated using the evaluation metrics listed in Table 3.3.1. These metrics were used on the validation data for design and tuning, and on the test data for evaluation of the models.

	Binary bottom detection
Loss function	BCL
Performance metrics	Average BA Overlap

Table 3.3.1: An evaluation matrix displaying the loss-function used to train the neural network models, as well as the metrics used to evaluate their performance on the validation and test data. BCL and average BA are traditional metrics used in DL. The formulas for these can be found in equations 2.4.3 and 3.3.1 respectively. Overlap is a custom metric detailed in equation 3.3.2.

For binary classification the most popular loss function and evaluation metric are BCL and BA. They are well established and efficiently implemented in the Keras package. BCL is detailed in equation 2.4.3, and BA is displayed in equation 3.3.1. For this task they were assumed to be useful, however they come with some drawbacks. Each time series was of length 1990 and contained approximately 130 bottom samples and 1860 non-bottom samples. Thus the dataset may be imbalanced and yield deflated cross entropy and inflated accuracy results on incorrect detections.

$$BA = \frac{1}{N}\sum_{i=1}^{N} \hat{y}_i \text{ where } \begin{cases} \hat{y}_i = y_i \text{ and} \\ y_i,\ \hat{y}_i \in 0, 1 \end{cases} \tag{3.3.1}$$

Due to the imbalanced data a custom evaluation metric was proposed. The Overlap metric compares the intervals of bottom detection and bottom labels, to calculate how much of the detected bottom echo overlaps with the true labels. The size of the overlapping interval is then normalized with respect to the longest interval out of the detections and the labels. In the case of a ping being labelled as non-bottom the metric is set to 0 if any bottom interval is detected by the model, and to 1 if no bottom is detected. Equation 3.3.2 details the formula for Overlap in the case where there is a labelled bottom, and figure 3.3.1 illustrates the interval used in overlap calculations.

$$Overlap = \frac{\min\limits_{\substack{i=0,\dots,N \\ j=0,\dots,N}} (\max(i), \max(j)) - \max\limits_{\substack{i=0,\dots,N \\ j=0,\dots,N}} (\min(i), \min(j))}{\max\limits_{\substack{i=0,\dots,N \\ j=0,\dots,N}} (\max(i) - \min(i), \max(j) - \min(j))} \text{ where } \begin{cases} y_i = 1 \\ \hat{y}_j = 1 \\ Overlap \in [0, 1] \end{cases} \tag{3.3.2}$$

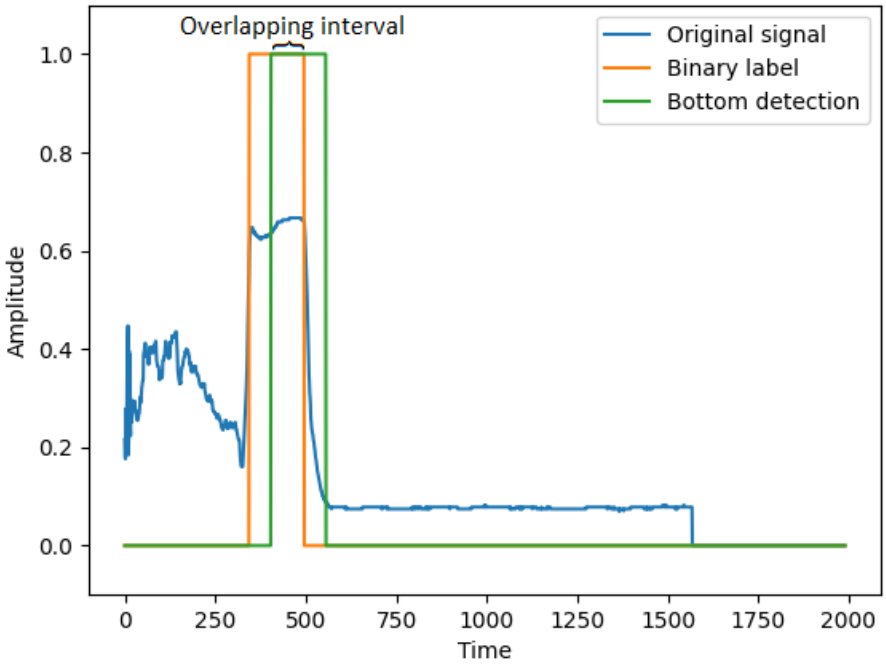


Figure 3.3.1: Illustration of the overlapping interval used when calculating the Overlap metric. This is a crafted example, not an actual model output.

An attempt to use $1 - Overlap$ as a custom loss function was attempted, in order to promote more precise bottom detections. However due to technical difficulties with custom loss functions in model tuning this was not achieved, and is instead suggested as future work.

3.4 Models

The RNN architectures used to perform bottom-detection were models using simple RNN layers [47] and models using GRU layers [48]. All models used a fully connected (FC) layer [49] with 1990 nodes as the output layer. The models were designed, tuned, and trained to to classify each time sample as bottom or non bottom by outputting a value between 0 and 1, where values closer to 1 are more likely bottom echoes. All models took inputs in the shape $(Batch\ size, Sequence\ length, 1990)$, and returned outputs of length 1990. Each hidden layer used a rectified linear unit (ReLU) for activation, while the output layers used sigmoid. Sigmoid was used in output layers in order to output values between 0 and 1, representing the estimated probability that the corresponding time sample is from a bottom echo. Any output value above 0.5 is considered a bottom prediction. Higher threshold values were tested on validation and test data, but 0.5 gave the best validation results. On the test data higher threshold values resulted in higher Overlap, however as the test data should not influence model parameters this was not taken into account. Otherwise the model performance could be overestimated and not true for general

purposes. Adam was used for backpropagation, with a parameter selection that commonly obtains best performance: $lr = 0.001$, $\beta 1 = 0.9$, $\beta 2 = 0.999$, $\epsilon = 1e{-}07$ and $decay = 0.0$. The models were trained for a maximum of 15 epochs.

On all models early stopping (ES) was implemented, as well as checkpointing in order to keep the model with the lowest validation loss or the highest validation overlap, depending on the model. ES makes training stop if the chosen metric stops improving, keeping training times lower in addition to reducing the risk of overfitting. Other measures taken to avoid overfitting was comparing training and validation loss and overlap, in addition to having one model of each type with dropout between all hidden layers.

3.4.1 Simple RNN

The first set of models experimented upon were based on simple RNN layers [47]. Models were made with $1 - 5$ simple RNN layers followed by an output layer or a FC layer in addition to the output layer. The width of each non output layer was varied during tuning, while the output layer had a width of 1990. In the end three different simple RNN models were trained and tested. These will be referred to as simple RNN model 1, simple RNN model 2 and simple RNN model 3.

Simple RNN model 1 was created by using hyperband tuning with the tuning scheme displayed in Table 3.4.1. Tuning methods are further explained in section 3.4.3. The final model consisted of two RNN layers, with a width of 800 and 200, as well as one FC layer for outputs. Figure A.0.1 illustrates the model architecture.

The second simple RNN model was created by using Random search tuning with the tuning scheme displayed in Table 3.4.1. The final model consisted of one RNN layers, with a width of 200, as well as two FC layers, one with width 400 and one for outputs. Figure A.0.2 illustrates the model architecture.

Simple RNN model 3 uses the same architecture and parameters as simple RNN model 2, with the addition of dropout layers [50]. Dropout layers with a dropout rate of 0.3 was placed between the simple RNN layer and the first FC layer, as well as between the two FC layers. The dropout rate was chosen for validation results after experimenting with values between 0.2 and 0.5 Figure A.0.3 illustrates the model architecture.

3.4.2 GRU

Gru layers were used for the second set of models [48]. These were assumed to perform better than the simple RNN models, due to the problems explained in section 2.5. The model structures match the structures used for simple RNN models, and they were tuned in the same fashion. In addition two GRU models were tuned for maximizing overlap, as the overlap metric was assumed to be a better indicator of good bottom detections than the standard BCL. This assumption was based on problems explained in section 3.3. In total four GRU based models were created.

They will be referred to as GRU model 1, GRU model 2, GRU model 3 and GRU model 4.

GRU model 1 was created by using hyperband tuning with the tuning scheme displayed in Table 3.4.2. This model was tuned for maximized validation overlap instead of minimized loss. GRU model 1 consisted of one GRU layer, with a width of 200, as well as two FC layers of width 600 and 1990. Figure A.0.4 illustrates the model architecture.

The second GRU model used random search tuning for maximized overlap. Table 3.4.2 shows the tuning scheme. In the end it consisted of two GRU layers, both with a width of 200. Following the GRU layers were two FC layers of width 800 and 1990. Figure A.0.5 illustrates the model architecture.

GRU model 3 was made with hyperband tuning for minimized validation loss. Table 3.4.2 shows the tuning scheme. The model includes two GRU layers, of width 800 and 200 and two FC layers of width 400 and 1990. Figure A.0.6 illustrates the model architecture.

The final GRU model is similar to GRU model 1, but with additional dropout layers like in simple RNN model 3. The same dropout rates were used, however for the GRU models the parameter was not varied. The main reason for adding dropout was to examine if the dropout layers would influence the validation results, not to create fully optimised models, hence the focus was not on tuning the dropout rate. Figure A.0.7 illustrates the model architecture.

3.4.3 Tuning and optimisation

Two different hyperparameter tuning methods were explored in the experiments. The random search tuner [51] and the hyperband tuner [52]. These tuners allow training many models for a few epochs each, to see which hyperparameters give the best results. Random search tuning chooses parameters at random, within the tuning scheme given. It always chooses a set of parameters that have not been used yet in the tuning run. Random search was used to train 80 models for 5 epochs each. The hyperband tuner also samples parameters randomly, however it uses Successive Halving [53] in order to choose which models to stop training early. By doing this it can cover the search space more efficiently than the random search. Li et.al. [54] observed an order-of-magnitude speedup over competing tuning algorithms on a variety of deep-learning problems.

Tuning parameter	Potential values	Final value Simple RNN model 1	Final value Simple RNN model 2
Number of RNN layers	1-5	2	1
RNN width	200-1000	Layer 1: 800 Layer 2: 200	Layer 1: 200
Number of FC layers	1-2	1	2
FC width	200-1000	Layer 1: 1990	Layer 1: 400 Layer 2: 1990
Batch size	16-96	80	48
Sequence length	2-10	4	6

Table 3.4.1: Table displaying the tuning scheme and the final parameters used for the simple RNN based models. FC layers with a width of 1990 are output layers. The final values for the two models are listed. These models were chosen based on having the lowest validation loss.

Tuning parameter	Potential values	Final value GRU model 1	Final value GRU model 2	Final value GRU model 3
Number of GRU layers	1-5	1	2	2
GRU width	200-1000	Layer 1: 200	Layer 1: 200 Layer 2: 200	Layer 1: 800 Layer 2: 200
Number of FC layers	1-2	2	2	2
FC width	200-1000	Layer 1: 600 Layer 2: 1990	Layer 1: 800 Layer 2: 1990	Layer 1: 400 Layer 2: 1990
Batch size	16-96	64	80	96
Sequence length	2-10	8	8	4

Table 3.4.2: Table displaying the tuning scheme and final parameters used for the GRU based models. FC layers with a width of 1990 are output layers. The final values for the three models are listed. Model 1 and 2 were chosen based on having the highest validation overlap, while model 3 was chosen for having the lowest validation loss.

3.5 Inference on embedded device

The seven models covered in section 3.4 were all converted into tflite. In addition post-training float 16 quantisation was performed on the same set of models, resulting in a total of 14 tflite models, seven of which used float 32 and seven using float 16.

As a result the quantised models were halved in storage size, in comparison to the unquantised models. The models were then transferred to the DIGI-CC8MNDVK board together with 5000 pings from the test dataset. Using these 5000 pings inference was run on each model, testing the average inference time. This was done with python on CPU, using the XNNPACK delegate as in [55]. Attempts were made to run inference with C++ on both CPU and GPU, using the NNAPI delegate [56]. However due to technical issues with the C++ library version on the DIGI board and the software development toolkit (SDK) used this was not achieved. C++ code and a compiled binary for this was made, based on the example code provided from tensorflow [57]. Completing this task is left as further work. Another test was ran on each of the tflite models, which assessed the memory usage when running inference with the models. This test came with the CC8MNDVK board and approximates memory footprint from running a model.

3.6 Implementation

Python [58] was used for all programming tasks regarding data processing, visualization and model engineering. The data processing was done by using the Numpy package [59]. Visualization of data utilized the packages Matplotlib [60] and Pandas [61], whilst visualization of models was performed with the tool Netron [62]. All work of designing, implementing and optimising models was done using the packages Tensorflow [63], Keras [64] and KerasTuner [65]. Converting and quantisation of models was done with the package tensorflow lite [40]. Training, tuning and testing of models was done on the NTNU Idun GPU cluster [66].

In order to run on-device inference a google VM [67] with Linux was used, for building executable binaries. The binaries were created based on C++ [68] code and CMake [69] build tools.

Chapter 4

Results and Discussion

This chapter aims to present and discuss the results obtained from training, validation and testing of the seven models detailed in section 3.4. The models were trained to solve the binary classification bottom-detection task on Norteks processed amplitude signals. Both a quantitative and a qualitative approach to analysing the results was performed. Both analysis and discussion of the evaluation metrics focus on the Keras version of each model, while the tflite results are used for a comparison to see if conversion and quantisation impacts model performance. All training and validation loss results are displayed in Appendix B, while the quantitative results from testing are displayed in Appendix C.

4.1 RNN bottom-detection

All models were evaluated on the metrics listed in the evaluation matrix in Table 3.3.1. The quantitative results presented are the BCL, average BA and Overlap on three different datasets, training, validation and test. The main points for analysis and discussion were comparing training, validation and test results to assess whether or not the models are overfitting, as well as evaluating the test results that stem from inference on completely unseen data. In addition some randomly selected pings, both correctly and incorrectly detected, will be qualitatively analysed in section 4.1.2.

4.1.1 Quantitative analysis

The training and validation BCL obtained during training of the binary classification models are displayed in Appendix B. For all models with no dropout there is a significant gap between training loss and validation loss. Meanwhile for the two models using dropout the losses are far more similar, and for GRU model 4 the validation loss is lower than the training loss. This indicates that there could have been some overfitting on the training data for models without dropout. All models have a very low BCL, implying that the classifications on training and validation data were successful. However due to the imbalance in data discussed in section 3.3

this could be caused by the nature of the data format.

Further quantitative results are presented in the tables of Appendix C. It is worth noting that all models performed significantly better on the training and validation data compared to the test data, in all evaluation metrics. This further indicates that there has been overfitting to the training data. Some difference was expected, however the differences here were significant enough to assume there has been some overfitting. In addition simple RNN model 3 performed notably better than simple RNN model 2, even though they shared architectures except for the dropout layers. For GRU model 1 and 4, which also share the same connection, the dropout layers did not cause any improvement on average BA and caused a decrease in Overlap. This implies that simple RNN model 2 was overfitting, while GRU model 1 is not overfitting or overfitting less. Moreover GRU model 3 performed similarly to simple RNN model 3 without having dropout, while also being the model with the most model parameters. This means the model should be more prone to overfitting. Thus GRU models might be less prone to overfitting than the simple RNN models. Further experimentation on dropout or other regularisation methods is suggested for further work, as the improvements from dropout, especially on the simple RNN models, is promising.

The best performing models were simple RNN model 3 and GRU model 3. There are no significant differences between the performance of these two models. Hence the assumption raised in section 3.4 seems to be wrong for this task. One possible reason for this is the short sequence lengths used. As mentioned in section 2.5 simple RNN layers struggle to capture long-term dependencies. With short sequence lengths there are no long-term dependencies to capture, there are instead short-term dependencies. There might be some more information to gain by having longer sequence lengths for GRU layers. However short sequences are desirable in order to be able to predict bottom echoes quickly after starting DVLs, and to quickly detect the DVL going outside or back into its operating range.

Overlap was assumed to be a more valuable metric than BA. The highest test Overlap achieved was for GRU model 3. This is also the metric with the most variance between models, with an average overlap of 0.8266 and a standard deviation of 0.03524 on the test set. In comparison average BA across all models was 0.9822 with a standard deviation of 0.00463. Because of the large difference in variances the assumption is considered valid, and we conclude that overlap gives more information about model performance than average BA for the bottom detection task.

4.1.2 Qualitative analysis

Based on the quantitative results, all the binary classification models and the correct bottom-intervals in most signals from the test set. However the overlap values indicate that some detections are totally missed or shifted. To qualitatively evaluate the predictions performed by the models, the original amplitude signal, the annotated bottom-interval, and the detected bottom were plotted for a random selection of six pings. Two pings were completely random, two were no-bottom pings and two pings were chosen from the faulty detections of the best performing model, GRU model

3. 200 000 pings from the test dataset were tested and the sample pings were picked at random from those.

Appendix D displays the selected pings with predictions from each model. All the models perform really well on the randomly selected pings, which have clear bottom echoes with amplitude peaks. The models also all predict the selected non bottom pings correctly. Some models have a slight increase in the bottom echo probability around time samples 1450 - 1500, most notably simple RNN model 1 and GRU model 1. The examples with faulty predictions differ a lot more between models. Simple RNN model 1 continues over predicting and predicts a bottom echo in both samples. Simple RNN model 2 underpredicts, and also predicts the bottom echoes a bit earlier than the others. Other underpredicting models, on these samples, are simple RNN model 3, GRU model 2 and GRU model 4. GRU model 1 predicts a bottom for both examples, similarly to simple RNN model 1. Finally GRU model 3 detects wrong for both, which was the criteria for how these pings were chosen.

From the qualitative results it appears that all seven models are good at detecting samples with a strong amplitude peak at the bottom echo. They are also capable of detecting non-bottom pings with relatively high certainty, unless the pings have tiny amplitude peaks around time sample 1450 - 1500. For the badly detected pings no model correctly detected both samples. The models either overpredicted or underpredicted these pings. However the bottom echo in sample ping 5 is not distinguishable to the human eye, and as such it seems likely that the models struggle on this ping. There might be some sequential info which is not taken into account here, although that would most probably make some of the models be more certain of a bottom echo.

The difficulties of the two selected bad detects could be due to the DVL being at the very edge or outside its operating range. Seeing as the difficult part of the signal is so far to the right, it is an echo that has traveled far. Thus the bottom, if it is a bottom echo, must be far away. In addition the echo gets attenuated more by the surrounding water after more time passes, making it harder to distinguish. Another possibility is the chance of faulty labelling. Labelling was done by Norteks heuristic algorithm. Since there are too many pings to manually label them this currently the best available option. In addition manual labelling could likely be worse, as it is very hard to distinguish if there is an amplitude peak or not. Even so it is not perfect, and can make mistakes just like the models experimented upon here.

4.1.3 Tflite models

The main points of interest regarding the tflite models is how they compare to the original Keras models, in addition to evaluating the effects of float 16 quantisation. Five of the models experience no significant change in performance on conversion to tflite, however the two models with the best results have significant drops in both average BA and Overlap. This makes GRU model 1 the best performing tflite model, with an average BA of 0.9795 and Overlap of 0.8225. Overlap values for the two best keras models, simple RNN model 3 and GRU model 3, are very similar for both the Keras version as well as the tflite versions of the models. Thus an explanation

for the drop in performance could be that there are pings where both models barely predict correct bottom echoes. With a small change in outputs these pings could be incorrect predictions, hurting the performance. These pings are likely the same set of pings for both models, due to the similar results and performance drop.

Float 16 quantisation is not observed to have any effect on performance. All models performed on the same level for both the unquantised and quantised tflite models.

4.2 Inference on CC8MNDVK

For inference on the DIGI-CC8MNDVK board inference times and memory usage was tested. All models exceed their test results when inferring on 5000 pings, implying that the models perform as well on the DIGI board as on the Idun cluster. Appendix E includes results from testing embedded inference. Inference times are presented in Table E.0.1, while Table E.0.2 display memory footprints. These times are from running inference on the CPU on the DIGI board, using a python script. It is assumed that running inference with C++ code on the GPU would result in inference times faster by magnitudes, however this was not tested due to challenges mentioned in section 3.5.

All models run inference relatively fast, with little to no difference between quantised or unquantised models. The smaller models run the fastest, with simple RNN model 3 running at 5.975 ms per ping. GRU model 3 is the largest model, and as such the slowest with 51.35 ms per ping. However the nortek DVLs used for data collections have a maximum ping rate of 8 Hz [43], [44], [45], [46], or once per 125 ms. Hence all the models are fast enough to continuously detect bottom echoes, even when running on the CPU. In a real scenario there might be several DVLs used together, in which case the slower models would not be able to keep up. Even so they should be able to run faster on GPU, which would make them viable for multi DVL inference. Memory usage also reflects upon the model size, with quantised models having a slightly higher memory footprint. This increased memory usage came during the initialization of the quantised models in all cases, with unquantised models usually having a larger footprint after initializing.

Chapter 5

Conclusion and further work

In this thesis, seven recurrent neural networks were designed to perform bottom detection in acoustic signals provided by the Nortek DVL1000-4000 m. [45], DVL1000-300 m. [44], DVL500-6000 m. [47], DVL500-300m. [46], Speed Log, Compact500, and DVL333. They propose alternatives to the current state-ofthe-art heuristic bottom-tracking algorithm, as well as the models presented in [2]. The network architectures were based on simple RNN layers [47] and GRU layers [48]. All models performed binary classification on amplitude-data, tasked with finding the interval of the signal corresponding to sea bottom echos. Models were trained and validated on a total of 928 727 acoustic signals, and tested on 247 352 signals. For the binary classification task all models performed with high accuracy and a high Overlap rate. The best model achieved a test accuracy of 98.78% and an overlap of 87.74%.

All models were tested on a DIGI CC8MNDVK board using CPU with the XNNPACK delegate. Inference times and memory footprints were tested, and all models achieved inference times below maximum ping rate of Nortek DVLs.

5.1 Conclusions

Following the results and discussions presented in chapter 4, this thesis concludes that the recurrent neural networks trained on amplitude signals successfully detect bottom intervals. The models performed accurate detections on data from Nortek DVL1000-4000 m., DVL1000-300 m., DVL500-6000 m., DVL500-300m., Speed Log, Compact500, and DVL333 without any instrument specific tailoring. The highest average test accuracy achieved was 98.78%. Claims that the presented overlap metric was a better indicator for precise bottom detection were supported by the achieved results. The highest test overlap achieved was 87.74%, which is deemed sufficient after the qualitative analysis, in section 4.1.2, detailing the miniscule differences between some non-bottom and bottom pings.

Due to the imbalanced nature of the data, as well as differences in datasets used, it is difficult to conclude whether the presented models outperform previous bottom detection algorithms and current state-of-the-art methods. [2] reported a maximum

test accuracy of 99.3% on binary classification. The maximum accuracy achieved here is comparable, while testing on a larger dataset, with a higher ratio of non-bottom signals and signals where a bottom echo was barely distinguishable. Results from the RNN models also exceed those of Taudien [4]. However as both the instrument and environment used for testing differed this is considered an observation and a reference instead of a conclusion, like mentioned by Skatvedt [2].

We also conclude that recurrent neural networks can be converted and quantised and still keep a high level of performance. The best performing models presented did experience a drop in performance, however they still performed on a high level. The maximal test accuracy for tflite converted models was 97.95% and the highest overlap was 82.25%. Float 16 quantisation showed no drop in performance when compared to tflite models using float 32 data. As such model conversion and quantisation using tflite is considered to be a valid solution in order to obtain models for embedded inference.

Finally the testing of embedded inference showed that it is possible to use RNNs for on device inference in subsea bottom tracking applications. All models reported inference times faster than the ping frequency of Norteks DVL instruments, while running on CPU. Such embedded inference could yield reduced latency, reduced energy usage as well as increased reliability and security in underwater operations. It is hypothesised that the inference times can be reduced by magnitudes by running inference on GPUs or NPUs, due to the parallel nature of computations needed. This hypothesis further supports the possibility for embedded inference. As such pre trained RNNs are considered a viable option for on-device bottom detection inference during real-time subsea navigation operations.

5.2 Further work

5.2.1 Inference on embedded GPU

GPU inference on the DIGI board was not achieved in these experiments. It was attempted, however it was not succesful due to technical difficulties with versioning as well as time constraints. It is believed to be a relatively small task to achieve this after managing CPU inference with python, however a SDK that builds binaries working with Glibc version 2.32 or older is required. We assume that GPU usage leads to faster and more efficient inference, having the potential to both reducing latency and energy costs. Both of which are crucial for efficient underwater operations.

5.2.2 Experiments with Hyperbolic Tangent activation

All presented models used ReLU as the activation function in all recurrent layers. Previous research suggests that the Hyperbolic Tangent (tanh) function works well with RNNs [70]. Most RNNs use either the ReLU or the tanh for activation and the

ReLU was chosen in these experiments. Further experimentation with activation functions or different model architectures could lead to interesting results.

5.2.3 Neural networks with overlap loss function

The overlap metric proposed in the thesis proved to be a promising evaluation metric for the bottom detection models. This was due to the imbalance in the data, causing traditional binary classification metrics to be less informative. Imbalanced data also influences BCL, which was used as the loss function for the presented models. As such using an overlap based loss function, $(1 - Overlap)$, could encourage learning new patterns, that help distinguish difficult bottom echoes that the presented models struggle with.

5.2.4 Improving regularisation

As discussed in section 4.1.1 there is a high likelihood that some of the models were overfitting the training data. Some experimentation with dropout for regularisation was performed, however there seems to be potential for great increase in results from improving the regularisation used on the models. All models saw significantly better validation results compared to test results, suggesting that there is a lot to gain from imrpoved regularisation, with dropout or other regularisation methods. Better regularisation would lead to a greater capability of creating general models that work for new, unseen data.

Bibliography

[1] N. AS. 'New to subsea navigation?' (2021), [Online]. Available: https://www.nortekgroup.com/knowledge-center/wiki/new-to-subsea-navigation (visited on 25/04/2022).

[2] M. Skatvedt, 'Deep neural network assisted sea bottom- and stable phase-detection using data from doppler velocity logs', M.S. thesis, NTNU, 2021. [Online]. Available: https://hdl.handle.net/11250/2828953.

[3] Ø. Hegrenæs, A. Ramstad, T. Pedersen and D. Velasco, 'Validation of a new generation dvl for underwater vehicle navigation', in *2016 IEEE/OES Autonomous Underwater Vehicles (AUV)*, 2016, pp. 342–348. DOI: 10.1109/AUV.2016.7778694.

[4] J. Y. Taudien, 'Doppler velocity log algorithms: Detection, estimation, and accuracy', The Pennsylvania State University, Jun. 2018. [Online]. Available: https://etda.libraries.psu.edu/catalog/15530jyt106.

[5] A. D. W. Robert N. McDonough, *Detection of Signals in Noise*. May 1995.

[6] T. G. Kincaid, 'Optimum waveforms for correlation detection in the sonar environment: Noise-limited conditions', 258, vol. 43, 1968. DOI: 10.1121/1.1910775.

[7] L. Yang and T. Taxt, 'Multibeam sonar bottom detection using multiple subarrays', in *Oceans '97. MTS/IEEE Conference Proceedings*, vol. 2, 1997, 932–938 vol.2. DOI: 10.1109/OCEANS.1997.624116.

[8] K. L. Deines and S. J. Maier, 'United states patent nr. 5,122,990', Jun. 1992.

[9] Waterlinked. 'Waterlinked, dvl a50'. (2021), [Online]. Available: https://waterlinked.com/dvl/ (visited on 13/06/2022).

[10] N. Davari and A. P. Aguiar, 'Real-time outlier detection applied to a doppler velocity log sensor based on hybrid autoencoder and recurrent neural network', *IEEE Journal of Oceanic Engineering*, vol. 46, no. 4, pp. 1288–1301, 2021. DOI: 10.1109/JOE.2021.3057909.

[11] W. Li, M. Chen, C. Zhang, Z. Lundong and R. Chen, 'A novel neural network-based sins/dvl integrated navigation approach to deal with dvl malfunction for underwater vehicles', *Mathematical Problems in Engineering*, vol. 2020, pp. 1–14, Jul. 2020. DOI: 10.1155/2020/2891572.

[12] P. Li, X. Zhang and X. Xu, 'Novel terrain integrated navigation system using neural network aided kalman filter', vol. 1, Aug. 2010, pp. 445–448. DOI: 10.1109/ICNC.2010.5583345.

[13] H. Xu and B. Lian, 'Fault detection for multi source integrated navigation system using fully convolutional neural network', *IET Radar, Sonar & Navigation*, vol. 12, Mar. 2018. DOI: 10.1049/iet-rsn.2017.0424.

[14] D. Smirnov and E. Mephu Nguifo, 'Time series classification with recurrent neural networks', Sep. 2018.

[15] H. A. Dau, E. Keogh, K. Kamgar *et al.*, *The ucr time series classification archive*, https://www.cs.ucr.edu/~eamonn/time_series_data_2018/, Oct. 2018.

[16] M. Hüsken and P. Stagge, 'Recurrent neural networks for time series classification', *Neurocomputing*, vol. 50, pp. 223–235, Jan. 2003. DOI: 10.1016/S0925-2312(01)00706-8.

[17] G. Taraldsen, T. A. Reinen and T. Berg, 'The underwater gps problem', in *OCEANS 2011 IEEE - Spain*, 2011, pp. 1–8. DOI: 10.1109/Oceans-Spain.2011.6003649.

[18] K. Gade, 'Inertial navigation — theory and applications', NTNU, 2018. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2491714/Kenneth%5C%20Gade_fulltext.pdf?sequence=7.

[19] J. González-García, A. Gómez-Espinosa, E. Cuan-Urquizo, L. G. García-Valdovinos, T. Salgado-Jiménez and J. A. E. Cabello, 'Autonomous underwater vehicles: Localization, navigation, and communication for collaborative missions', *Applied Sciences*, vol. 10, no. 4, 2020, ISSN: 2076-3417. DOI: 10.3390/app10041256. [Online]. Available: https://www.mdpi.com/2076-3417/10/4/1256.

[20] J. J. Leonard, A. A. Bennett, C. M. Smith, H. Jacob and S. Feder, 'Autonomous underwater vehicle navigation', in *MIT Marine Robotics Laboratory Technical Memorandum*, 1998.

[21] T. Nicosevici, R. Garcia, M. Carreras and M. Villanueva, 'A review of sensor fusion techniques for underwater vehicle navigation', vol. 3, Dec. 2004, 1600–1605 Vol.3, ISBN: 0-7803-8669-8. DOI: 10.1109/OCEANS.2004.1406361.

[22] Y. R. Petillot, G. Antonelli, G. Casalino and F. Ferreira, 'Underwater robots: From remotely operated vehicles to intervention-autonomous underwater vehicles', *IEEE Robotics & Automation Magazine*, vol. 26, no. 2, pp. 94–101, 2019. DOI: 10.1109/MRA.2019.2908063.

[23] T. M. Mitchell, *Machine Learning*. McGraw Hill, 1997.

[24] K. Hornik, 'Approximation capabilities of multilayer feedforward networks', *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. [Online]. Available: https://arxiv.org/pdf/2001.03329.

[25] H. Mhaskar, Q. Liao and T. Poggio, 'When and why are deep networks better than shallow ones?', *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, Feb. 2017.

[26] J. D. Kelleher, *Machine Learning*. MIT Press, 2019.

[27] Y. LeCun, Y. Bengio and G. Hinton, 'Deep learning', *Nature*, no. 521, pp. 436–444, 2015. DOI: 10.1038/nature14539.

[28] S. Salman and X. Liu, 'Overfitting mechanism and avoidance in deep neural networks', *CoRR*, vol. abs/1901.06566, 2019. arXiv: 1901.06566. [Online]. Available: http://arxiv.org/abs/1901.06566.

[29] Q. Liu and Y. Wu, 'Supervised learning', Jan. 2012. DOI: 10.1007/978-1-4419-1428-6_451.

[30] S. Wang, W. Chaovalitwongse and R. Babuska, 'Machine learning algorithms in bipedal robot control', *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 5, pp. 728–743, 2012. DOI: 10.1109/TSMCC.2012.2186565.

[31] B. Kröse and P. van der Smagt, 'An introduction to neural networks', *J Comput Sci*, vol. 48, Jan. 1993.

[32] C. M. Bishop, 'Neural networks and their applications', *Review of Scientific Instruments*, vol. 65, Mar. 1994. DOI: 10.1063/1.1144830.

[33] D. P. Kingma and J. Ba, 'Adam: A method for stochastic optimization', 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.

[34] R. M. Schmidt, 'Recurrent neural networks (rnns): A gentle introduction and overview', *CoRR*, vol. abs/1912.05911, 2019. arXiv: 1912.05911.

[35] Y. Bengio, P. Simard and P. Frasconi, 'Learning long-term dependencies with gradient descent is difficult', *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994. DOI: 10.1109/72.279181.

[36] J. Chung, Ç. Gülçehre, K. Cho and Y. Bengio, 'Empirical evaluation of gated recurrent neural networks on sequence modeling', *CoRR*, vol. abs/1412.3555, 2014. arXiv: 1412.3555. [Online]. Available: http://arxiv.org/abs/1412.3555.

[37] A. Graves, 'Supervised sequence labelling with recurrent neural networks', *Stud Comput Intell*, vol. 385, Jan. 2012. DOI: 10.1007/978-3-642-24797-2.

[38] I. Sutskever, O. Vinyals and Q. V. Le, 'Sequence to sequence learning with neural networks', 2014. DOI: 10.48550/ARXIV.1409.3215. [Online]. Available: https://arxiv.org/abs/1409.3215.

[39] M.-A. A'râbi and V. Schwarz, 'General constraints in embedded machine learning and how to overcome them — a survey paper', Jul. 2019. DOI: 10.13140/RG.2.2.14747.21280.

[40] Tensorflow. 'Tensorflow lite'. (2022), [Online]. Available: https://www.tensorflow.org/lite (visited on 25/04/2022).

[41] ——, 'Tensorflow lite guide'. (2022), [Online]. Available: https://www.tensorflow.org/lite/guide (visited on 25/04/2022).

[42] DIGI. 'Digi embedded documentation portal'. (2022), [Online]. Available: https://www.digi.com/resources/documentation/digidocs/embedded/index.html (visited on 21/05/2022).

[43] Nortek. 'Dvl1000 - 300 m'. (2021), [Online]. Available: https://www.nortekgroup.com/products/dvl-1000-300m (visited on 19/05/2022).

[44] ——, 'Dvl1000 - 4000 m'. (2021), [Online]. Available: https://www.nortekgroup.com/products/dvl1000-4000m (visited on 19/05/2022).

[45] ——, 'Dvl1000 - 300 m'. (2021), [Online]. Available: https://www.nortekgroup.com/products/dvl500-300-m (visited on 19/05/2022).

[46] ——, 'Dvl500 - 6000 m'. (2021), [Online]. Available: https://www.nortekgroup.com/products/dvl500-6000-m (visited on 19/05/2022).

[47] Keras. 'Simplernn layer'. (2022), [Online]. Available: https://keras.io/api/layers/recurrent_layers/simple_rnn/ (visited on 21/05/2022).

[48] ——, 'Gru layer'. (2022), [Online]. Available: https://keras.io/api/layers/recurrent_layers/gru/ (visited on 21/05/2022).

[49] ——, 'Dense layer'. (2022), [Online]. Available: https://keras.io/api/layers/core_layers/dense/ (visited on 02/06/2022).

[50] ——, 'Dropout layer'. (2022), [Online]. Available: https://keras.io/api/layers/regularization_layers/dropout/ (visited on 02/06/2022).

[51] ——, 'Randomsearch tuner'. (2022), [Online]. Available: https://keras.io/api/keras_tuner/tuners/random/#randomsearch-class (visited on 02/06/2022).

[52] ——, 'Hyperband tuner'. (2022), [Online]. Available: https://keras.io/api/keras_tuner/tuners/hyperband/#hyperband-class (visited on 02/06/2022).

[53] K. G. Jamieson and A. Talwalkar, 'Non-stochastic best arm identification and hyperparameter optimization', *CoRR*, vol. abs/1502.07943, 2015. arXiv: 1502.07943. [Online]. Available: http://arxiv.org/abs/1502.07943.

[54] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh and A. Talwalkar, 'Hyperband: A novel bandit-based approach to hyperparameter optimization', *CoRR*, vol. abs/1603.06560, 2016. arXiv: 1603.06560. [Online]. Available: http://arxiv.org/abs/1603.06560.

[55] N. Semiconductors, 'I.mx machine learning user's guide', Dec. 2021. [Online]. Available: https://www.nxp.com/docs/en/user-guide/IMX-MACHINE-LEARNING-UG.pdf.

[56] Tensorflow. 'Tensorflow lite nnapi delegate'. (2022), [Online]. Available: https://www.tensorflow.org/lite/android/delegates/nnapi (visited on 05/06/2022).

[57] ——, 'Tensorflow lite examples'. (2022), [Online]. Available: https://github.com/fredralm/tensorflow/tree/master/tensorflow/lite/examples (visited on 05/06/2022).

[58] Python. 'Python'. (2022), [Online]. Available: https://www.python.org/about/ (visited on 25/04/2022).

[59] Numpy. 'Numpy'. (2022), [Online]. Available: https://numpy.org/ (visited on 21/05/2022).

[60] Matplotlib. 'Matplotlib: Visualization with python'. (2022), [Online]. Available: https://matplotlib.org/ (visited on 21/05/2022).

[61] Pandas. 'Pandas'. (2022), [Online]. Available: https://pandas.pydata.org/ (visited on 21/05/2022).

[62] L. Roeder. 'Netron, visualizer for neural network, deep learning, and machine learning models'. (2022), [Online]. Available: https://netron.app/ (visited on 21/05/2022).

[63] Tensorflow. 'Tensorflow, an end-to-end open source machine learning platform'. (2022), [Online]. Available: https://www.tensorflow.org/ (visited on 21/05/2022).

[64] Keras. 'Keras, the python deep learning api'. (2022), [Online]. Available: https://keras.io/ (visited on 21/05/2022).

[65] ——, 'Kerastuner'. (2022), [Online]. Available: https://keras.io/keras_tuner/ (visited on 21/05/2022).

[66] M. Själander, M. Jahre, G. Tufte and N. Reissmann, 'EPIC: An energy-efficient, high-performance GPGPU computing research infrastructure', 2019. arXiv: 1912.05848 [cs.DC].

[67] Google. 'Google cloud platform'. (2022), [Online]. Available: https://console.cloud.google.com/ (visited on 21/05/2022).

[68] C++. 'Cplusplus'. (2022), [Online]. Available: https://cplusplus.com/ (visited on 21/05/2022).

[69] CMake. 'Cmake'. (2022), [Online]. Available: https://cmake.org/ (visited on 21/05/2022).

[70] T. Szandala, 'Review and comparison of commonly used activation functions for deep neural networks', *CoRR*, vol. abs/2010.09458, 2020. arXiv: 2010.09458. [Online]. Available: https://arxiv.org/abs/2010.09458.

Appendix A

Final model architectures

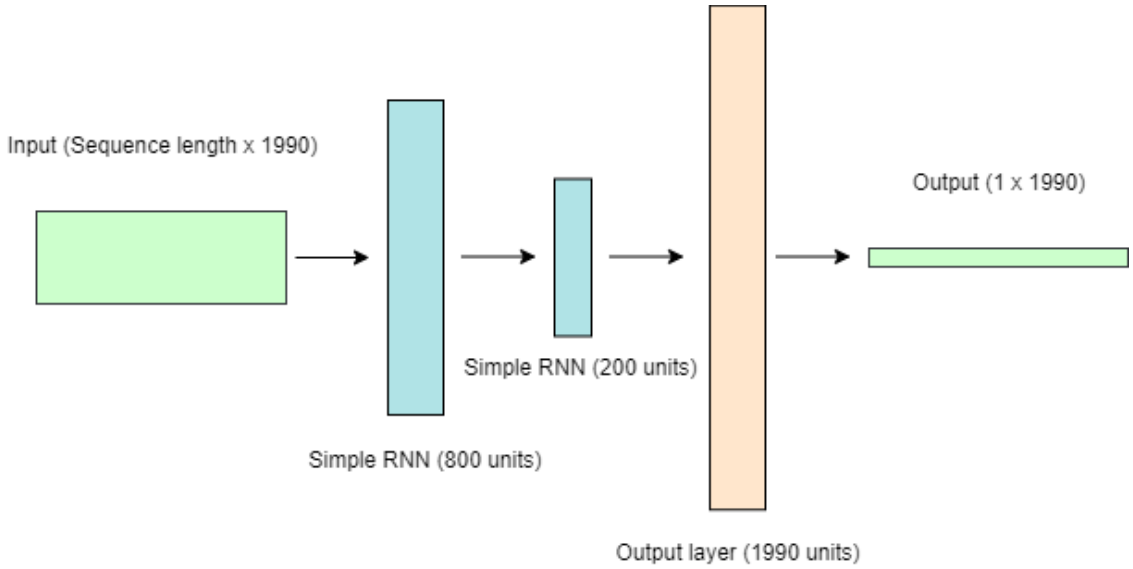


Figure A.0.1: Illustration of the final architecture of simple RNN model 1, displaying inputs, layers, widths and outputs.

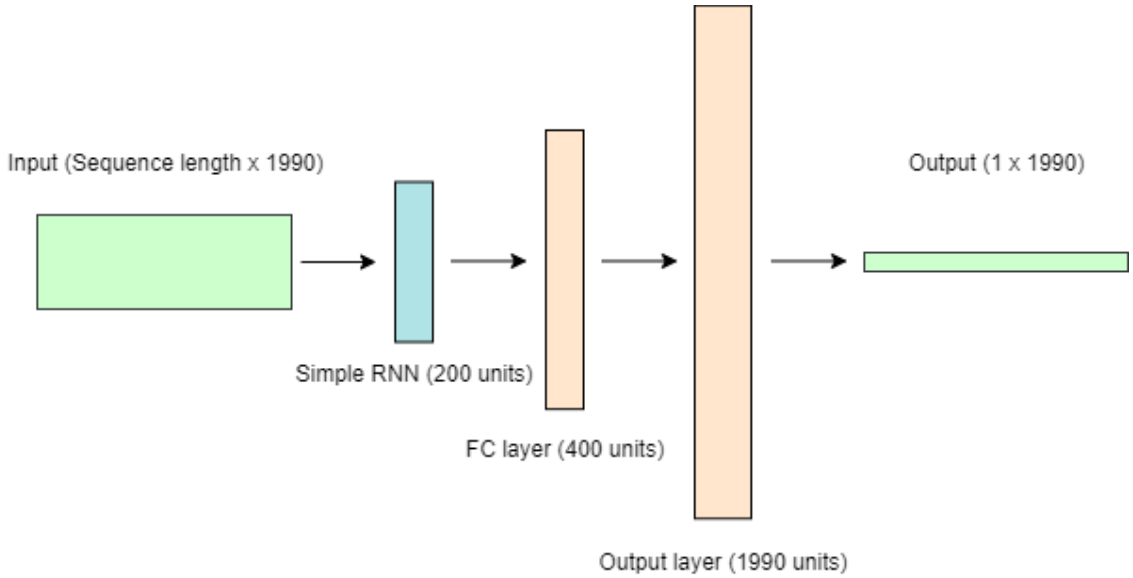


Figure A.0.2: Illustration of the final architecture of simple RNN model 2, displaying inputs, layers, widths and outputs.

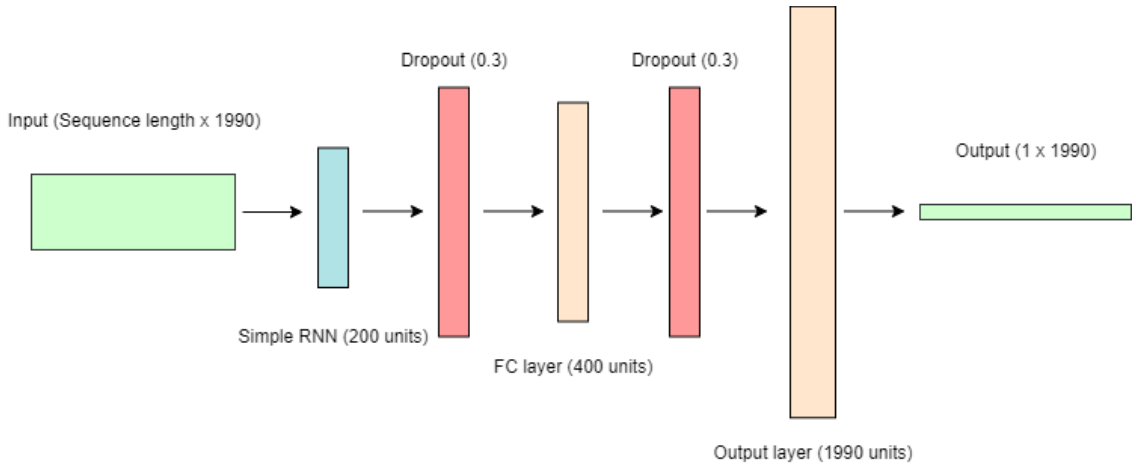


Figure A.0.3: Illustration of the final architecture of simple RNN model 3, displaying inputs, layers, widths and outputs.

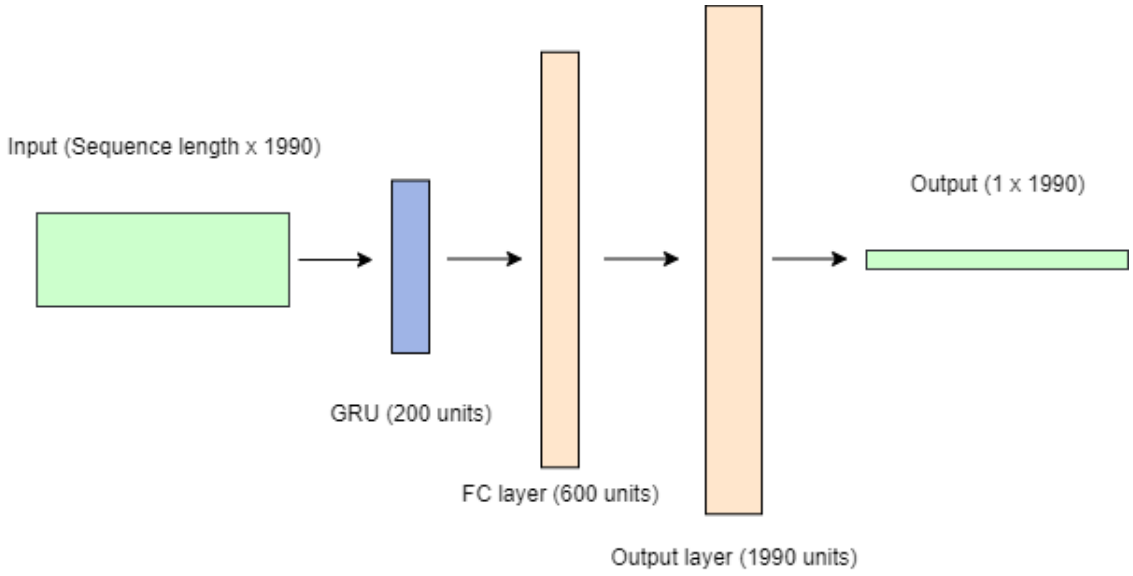


Figure A.0.4: Illustration of the final architecture of GRU model 1, displaying inputs, layers, widths and outputs.

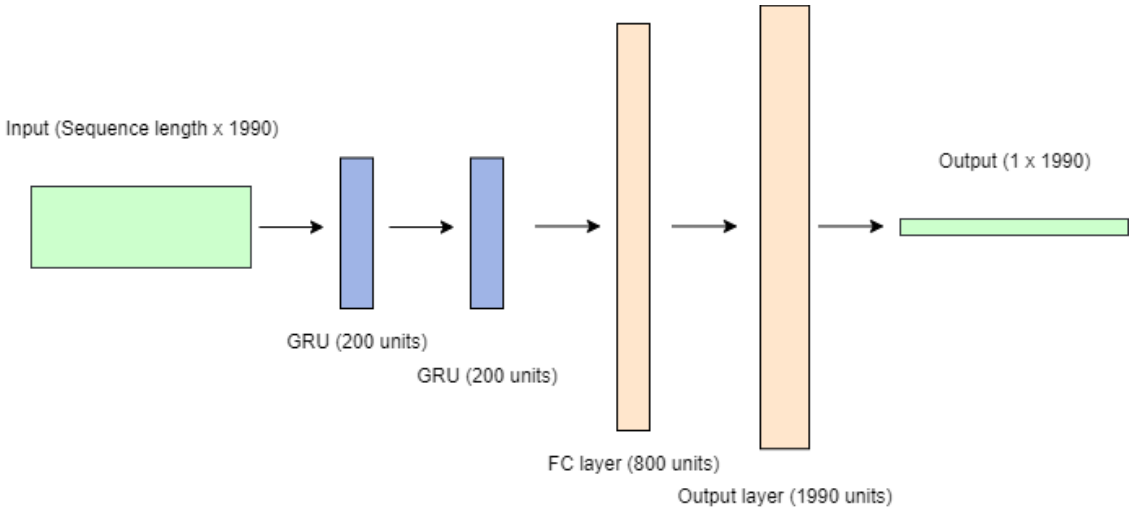


Figure A.0.5: Illustration of the final architecture of GRU model 2, displaying inputs, layers, widths and outputs.

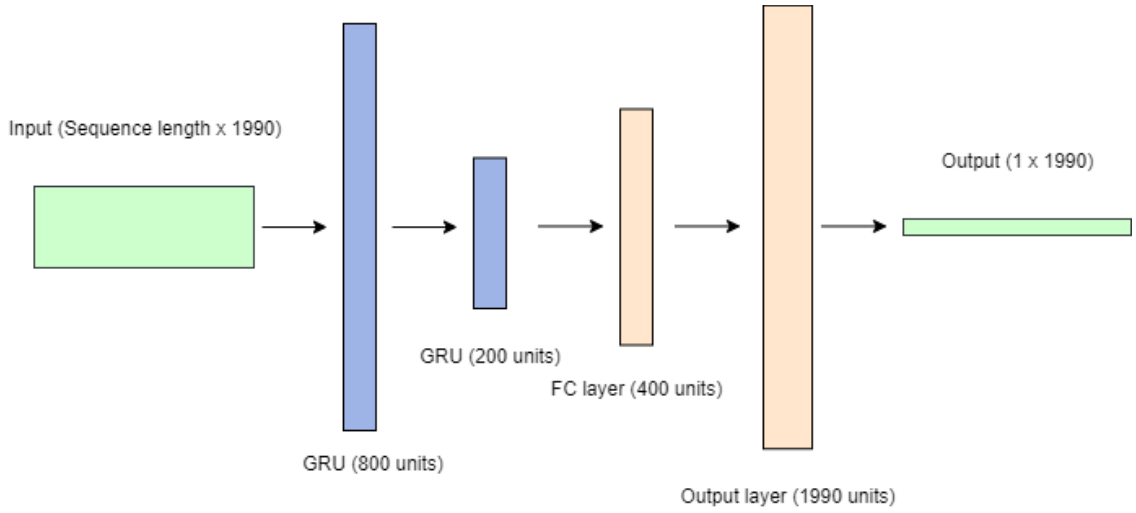


Figure A.0.6: Illustration of the final architecture of GRU model 3, displaying inputs, layers, widths and outputs.

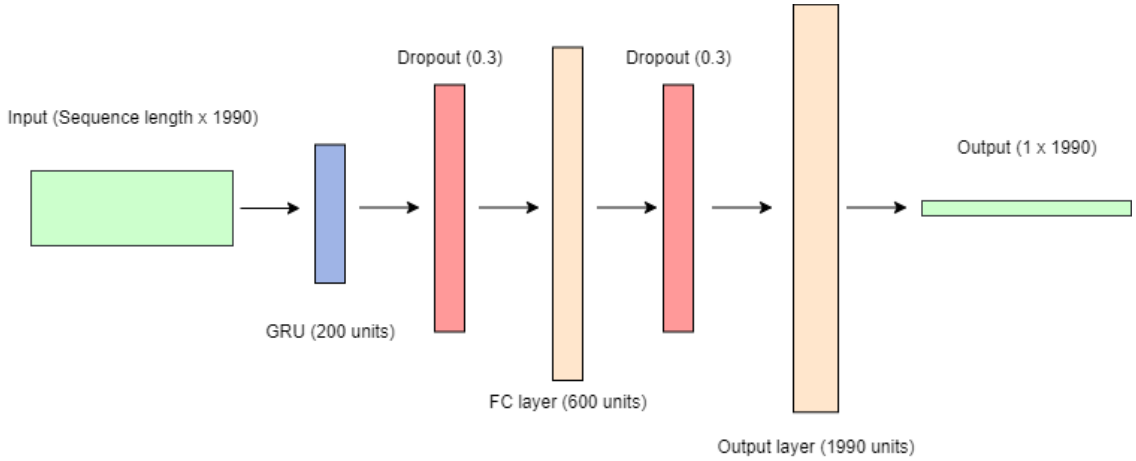


Figure A.0.7: Illustration of the final architecture of GRU model 4, displaying inputs, layers, widths and outputs.

Appendix B

Loss plots

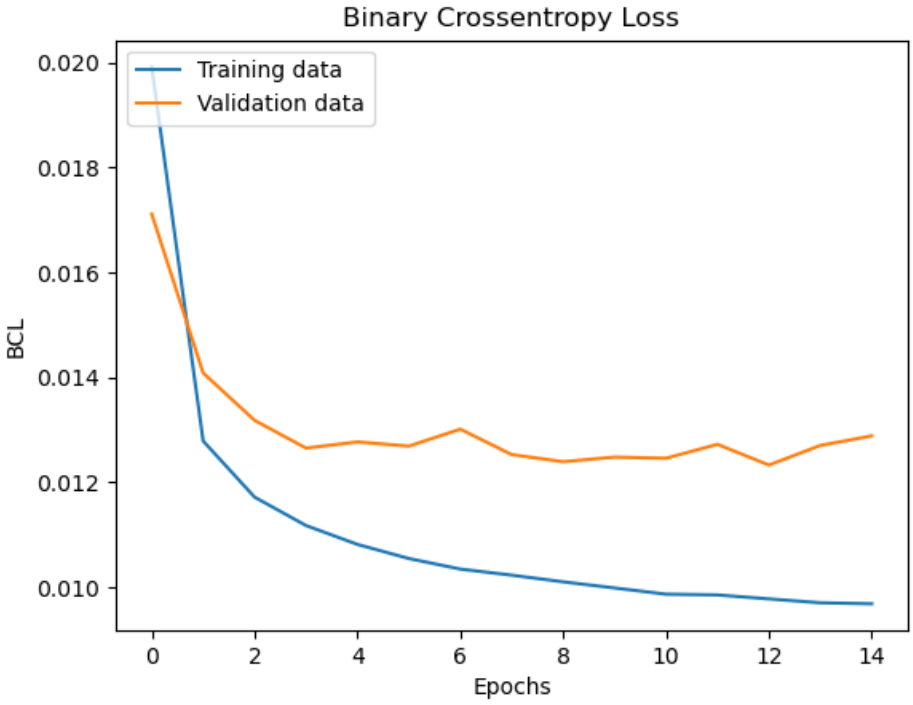


Figure B.0.1: Loss plot from training simple RNN model 1, comparing training and validation loss

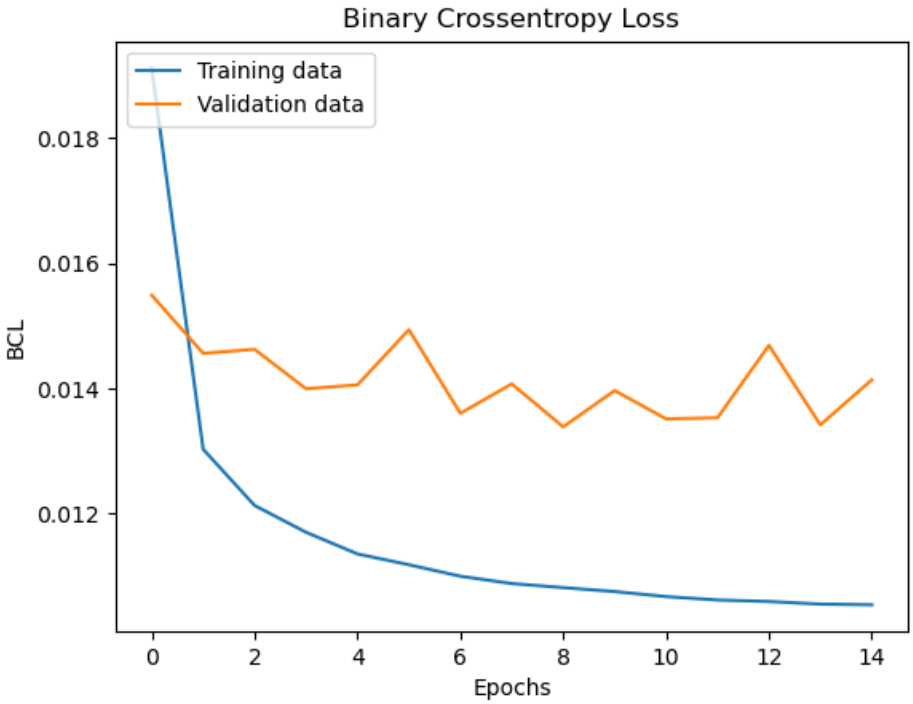


Figure B.0.2: Loss plot from training simple RNN model 2, comparing training and validation loss

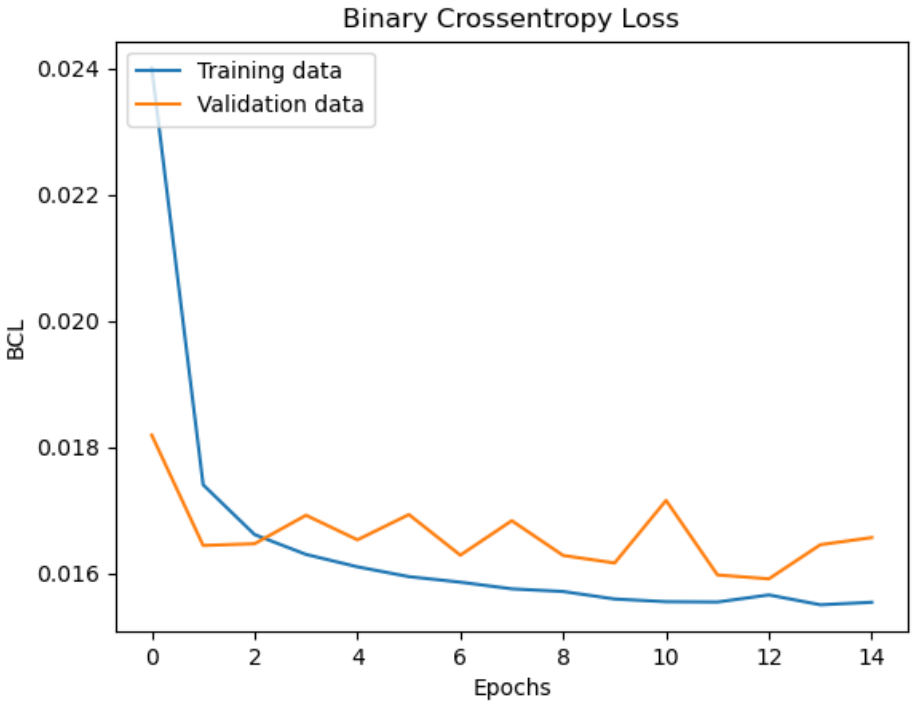


Figure B.0.3: Loss plot from training simple RNN model 3, comparing training and validation loss

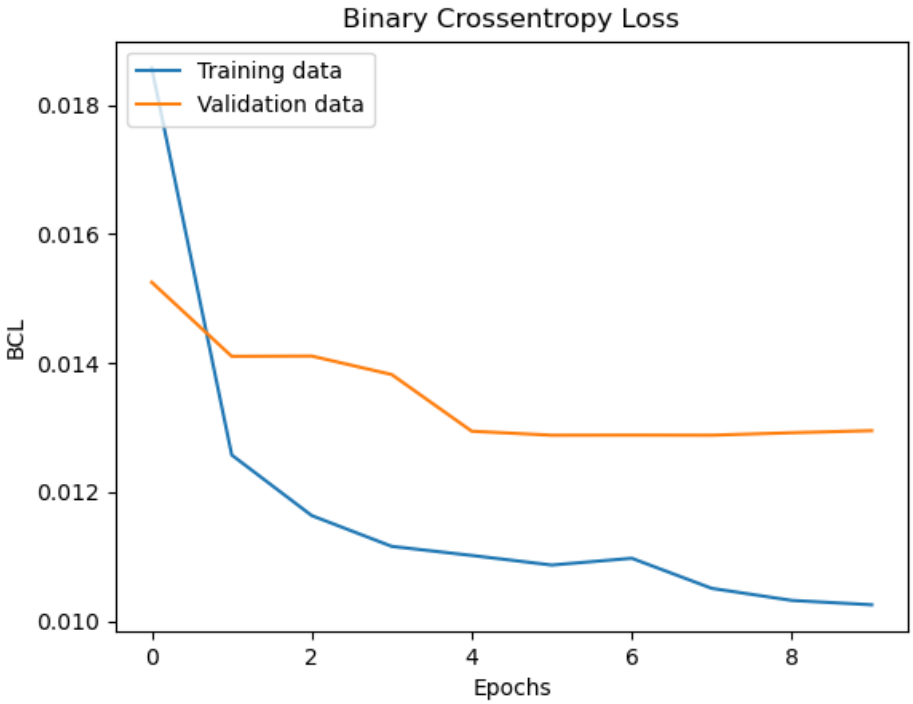


Figure B.0.4: Loss plot from training GRU model 1, comparing training and validation loss

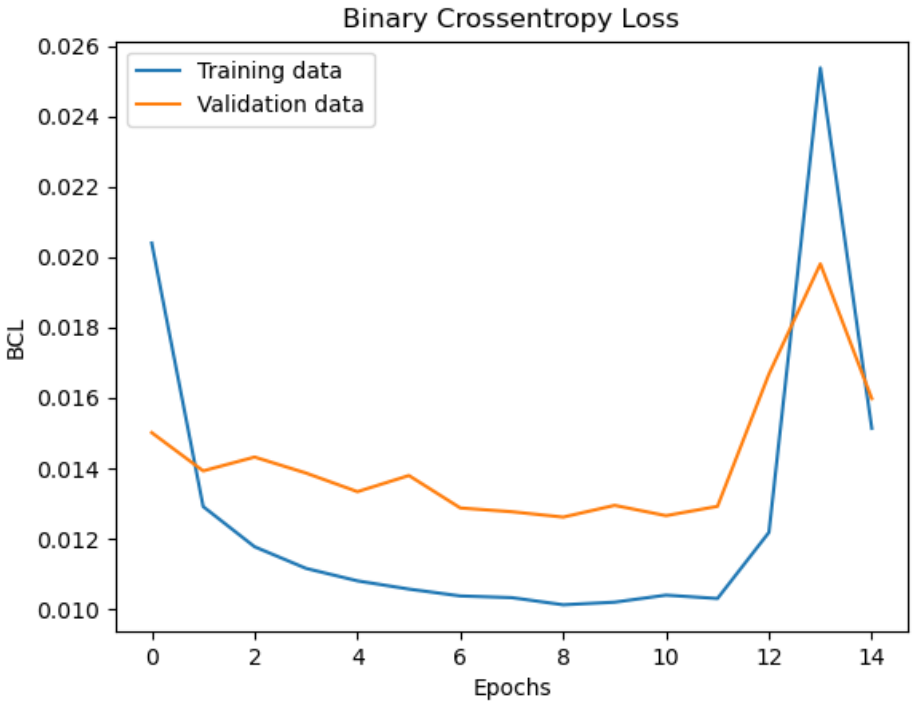


Figure B.0.5: Loss plot from training GRU model 2, comparing training and validation loss

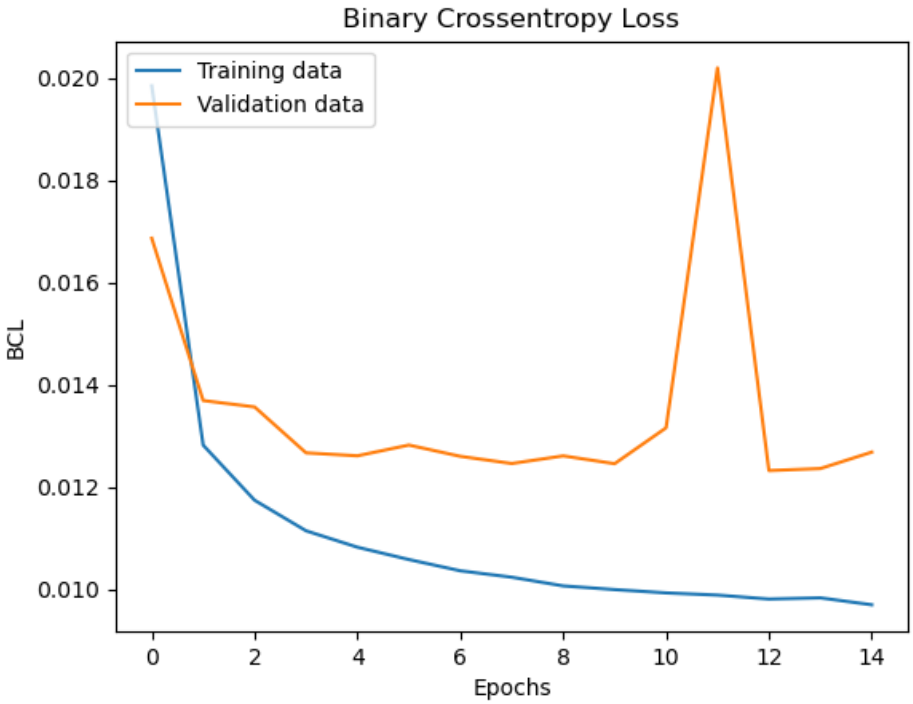


Figure B.0.6: Loss plot from training GRU model 3, comparing training and validation loss

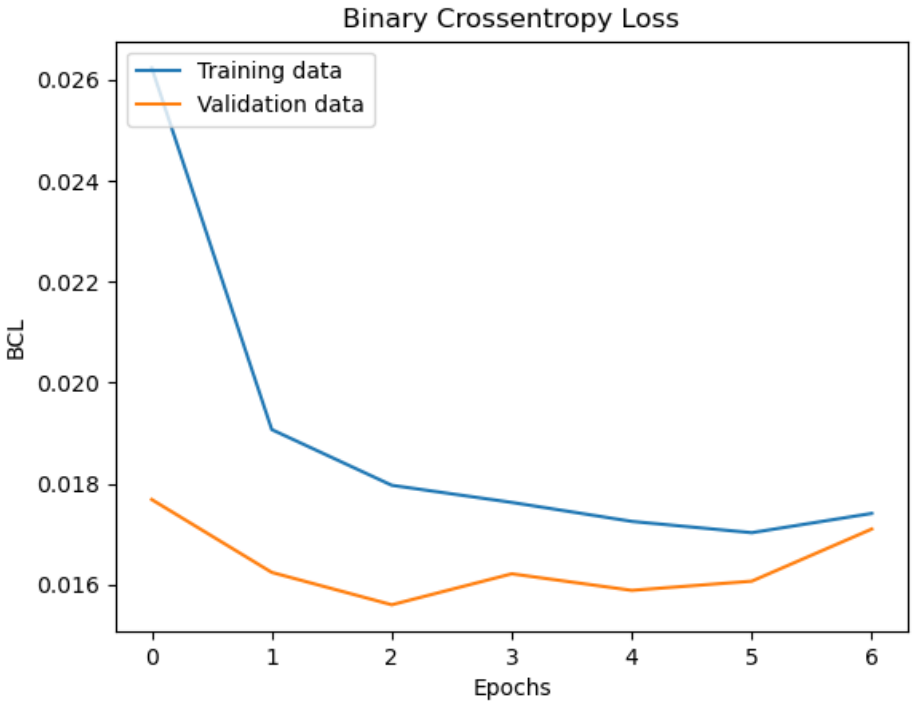


Figure B.0.7: Loss plot from training GRU model 4, comparing training and validation loss

Appendix C

Quantitative results

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0098	Not tested	Not tested
BCL validation data	0.0123	Not tested	Not tested
BCL test data	0.1026	Not tested	Not tested
Average BA training data	0.9960	Not tested	Not tested
Average BA validation data	0.9955	0.9955	0.9955
Average BA test data	0.9770	0.9770	0.9770
Overlap training data	0.9533	Not tested	Not tested
Overlap validation data	0.9507	0.9508	0.9508
Overlap test data	0.8018	0.8023	0.8023

Table C.0.1: Table displaying the quantitative results for simple RNN model 1.

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0108	Not tested	Not tested
BCL validation data	0.0134	Not tested	Not tested
BCL test data	0.1284	Not tested	Not tested
Average BA training data	0.9957	Not tested	Not tested
Average BA validation data	0.9952	0.9950	0.9950
Average BA test data	0.9743	0.9743	0.9743
Overlap training data	0.9496	Not tested	Not tested
Overlap validation data	0.9478	0.9462	0.9462
Overlap test data	0.7831	0.7837	0.7837

Table C.0.2: Table displaying the quantitative results for simple RNN model 2.

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0157	Not tested	Not tested
BCL validation data	0.0159	Not tested	Not tested
BCL test data	0.0428	Not tested	Not tested
Average BA training data	0.9940	Not tested	Not tested
Average BA validation data	0.9947	0.9946	0.9946
Average BA test data	0.9863	0.9769	0.9769
Overlap training data	0.9287	Not tested	Not tested
Overlap validation data	0.9403	0.9394	0.9394
Overlap test data	0.8764	0.8102	0.8102

Table C.0.3: Table displaying the quantitative results for simple RNN model 3.

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0109	Not tested	Not tested
BCL validation data	0.0159	Not tested	Not tested
BCL test data	0.1056	Not tested	Not tested
Average BA training data	0.9930	Not tested	Not tested
Average BA validation data	0.9926	0.9953	0.9953
Average BA test data	0.9820	0.9795	0.9795
Overlap training data	0.9496	Not tested	Not tested
Overlap validation data	0.9490	0.9482	0.9482
Overlap test data	0.8222	0.8225	0.8225

Table C.0.4: Table displaying the quantitative results for GRU model 1.

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0104	Not tested	Not tested
BCL validation data	0.0127	Not tested	Not tested
BCL test data	0.0672	Not tested	Not tested
Average BA training data	0.9931	Not tested	Not tested
Average BA validation data	0.9929	0.9943	0.9943
Average BA test data	0.9857	0.9795	0.9795
Overlap training data	0.9507	Not tested	Not tested
Overlap validation data	0.9500	0.9354	0.9354
Overlap test data	0.8134	0.8139	0.8139

Table C.0.5: Table displaying the quantitative results for GRU model 2.

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0098	Not tested	Not tested
BCL validation data	0.0123	Not tested	Not tested
BCL test data	0.0573	Not tested	Not tested
Average BA training data	0.9934	Not tested	Not tested
Average BA validation data	0.9930	0.9953	0.9953
Average BA test data	0.9878	0.9782	0.9782
Overlap training data	0.9534	Not tested	Not tested
Overlap validation data	0.9508	0.9492	0.9492
Overlap test data	0.8774	0.8101	0.8101

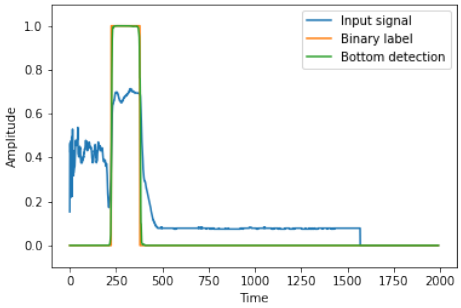
Table C.0.6: Table displaying the quantitative results for GRU model 3.

Performance metric	Keras model	Tflite model	Quantized tflite model
BCL training data	0.0180	Not tested	Not tested
BCL validation data	0.0156	Not tested	Not tested
BCL test data	0.0734	Not tested	Not tested
Average BA training data	0.9887	Not tested	Not tested
Average BA validation data	0.9882	0.9942	0.9942
Average BA test data	0.9827	0.9762	0.9762
Overlap training data	0.9207	Not tested	Not tested
Overlap validation data	0.9375	0.9360	0.9360
Overlap test data	0.7954	0.7958	0.7958

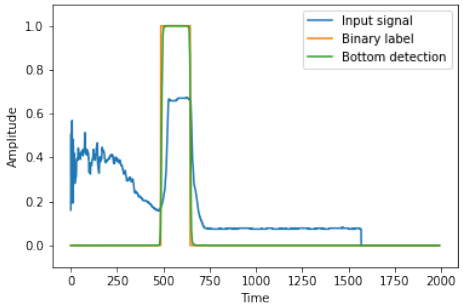
Table C.0.7: Table displaying the quantitative results for GRU model 4.

Appendix D

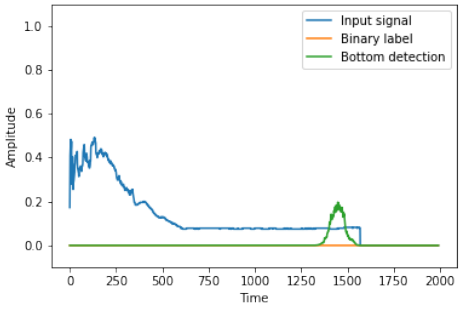
Qualitative results



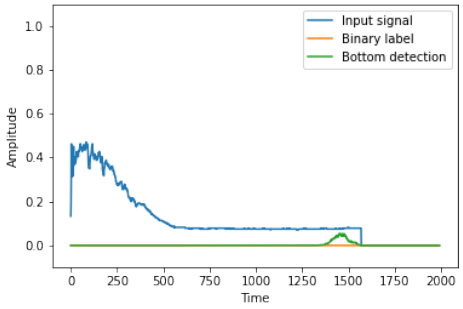
(a) Sample ping 1, simple RNN model 1



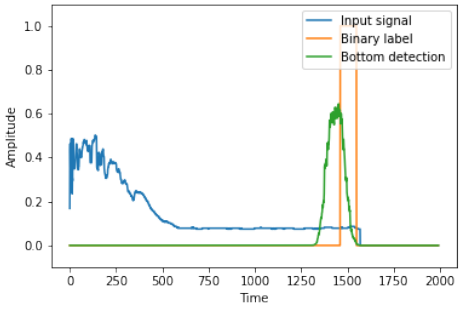
(b) Sample ping 2, simple RNN model 1



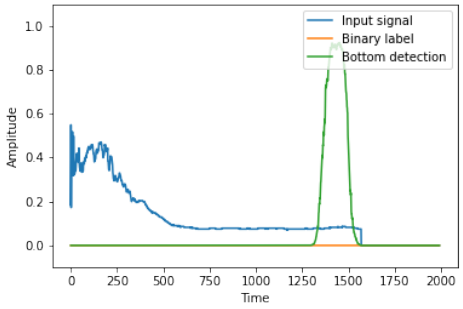
(c) Sample ping 3, simple RNN model 1



(d) Sample ping 4, simple RNN model 1

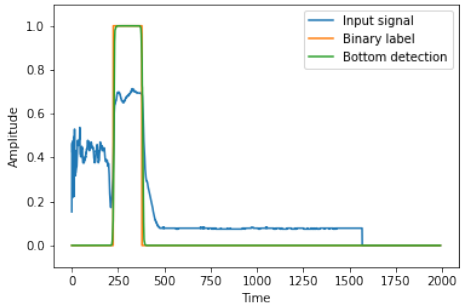


(e) Sample ping 5, simple RNN model 1

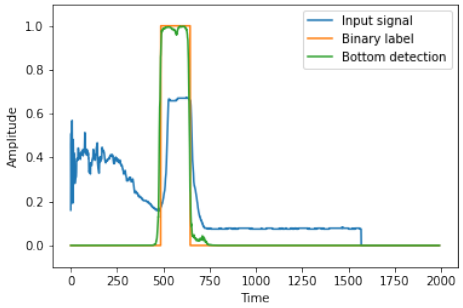


(f) Sample ping 6, simple RNN model 1

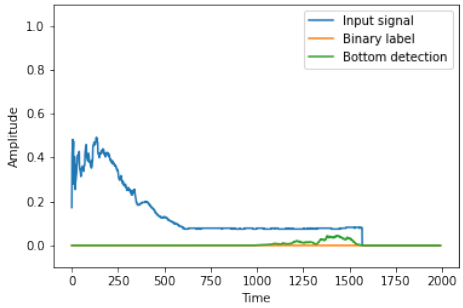
Figure D.0.1: Sample pings for qualitative analysis of simple RNN model 1.



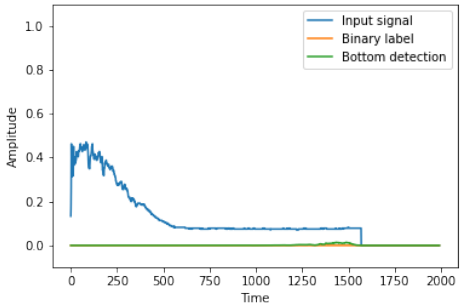
(a) Sample ping 1, simple RNN model 2



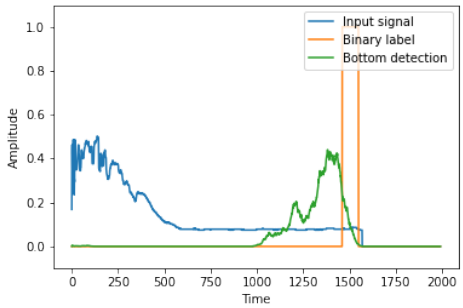
(b) Sample ping 2, simple RNN model 2



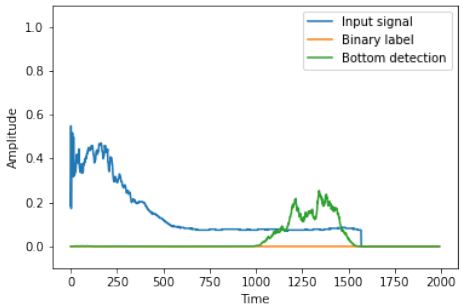
(c) Sample ping 3, simple RNN model 2



(d) Sample ping 4, simple RNN model 2

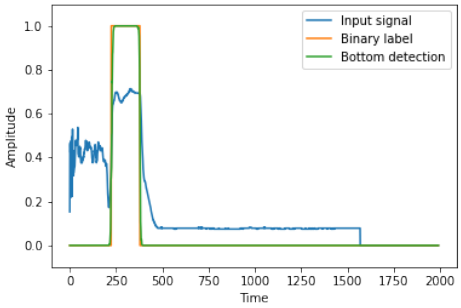


(e) Sample ping 5, simple RNN model 2

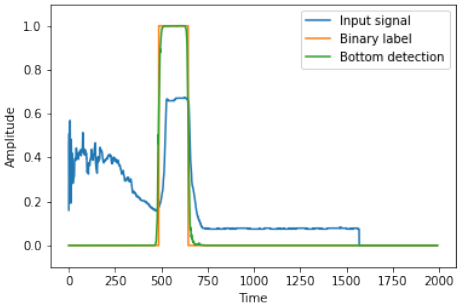


(f) Sample ping 6, simple RNN model 2

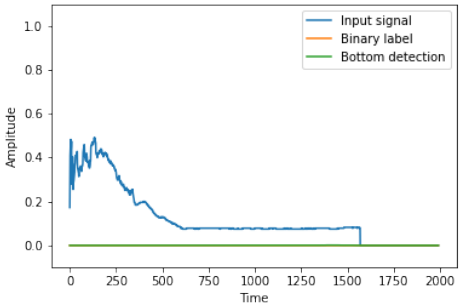
Figure D.0.2: Sample pings for qualitative analysis of simple RNN model 2.



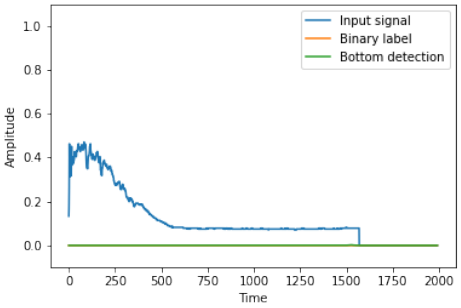
(a) Sample ping 1, simple RNN model 3



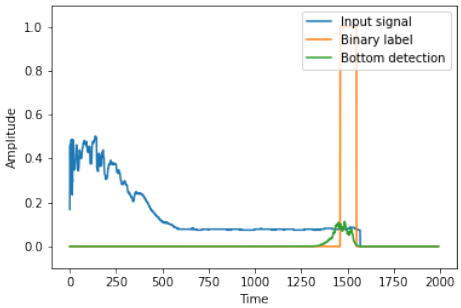
(b) Sample ping 2, simple RNN model 3



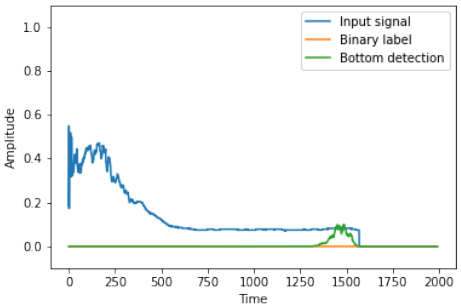
(c) Sample ping 3, simple RNN model 3



(d) Sample ping 4, simple RNN model 3

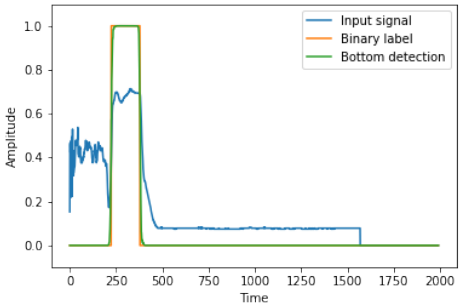


(e) Sample ping 5, simple RNN model 3

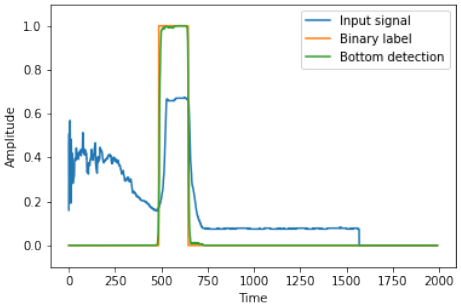


(f) Sample ping 6, simple RNN model 3

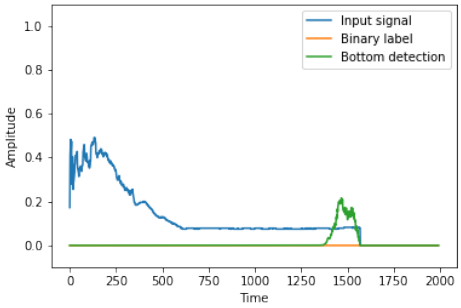
Figure D.0.3: Sample pings for qualitative analysis of simple RNN model 3.



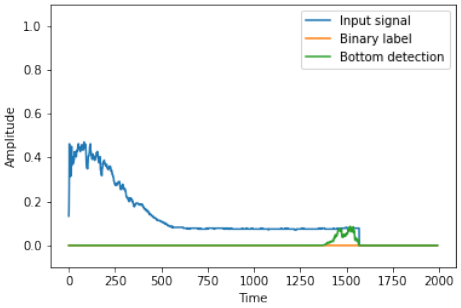
(a) Sample ping 1, GRU model 1



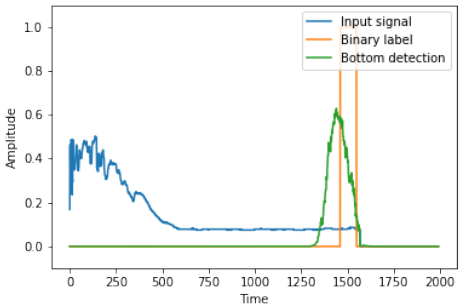
(b) Sample ping 2, GRU model 1



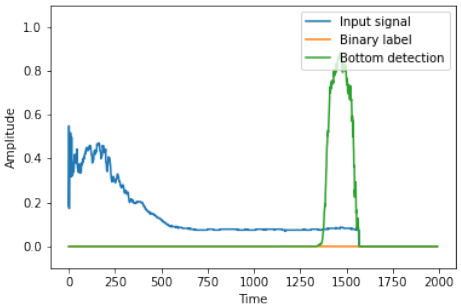
(c) Sample ping 3, GRU model 1



(d) Sample ping 4, GRU model 1

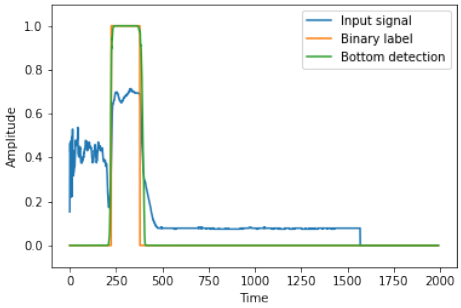


(e) Sample ping 5, GRU model 1

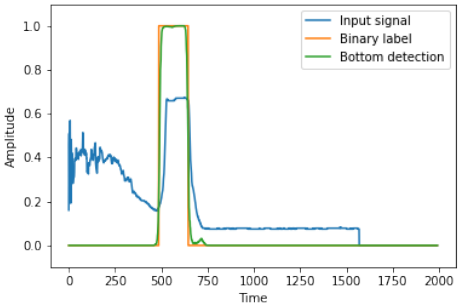


(f) Sample ping 6, GRU model 1

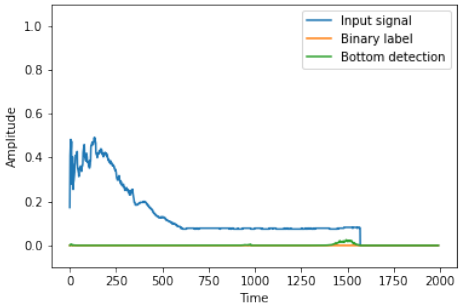
Figure D.0.4: Sample pings for qualitative analysis of GRU model 1.



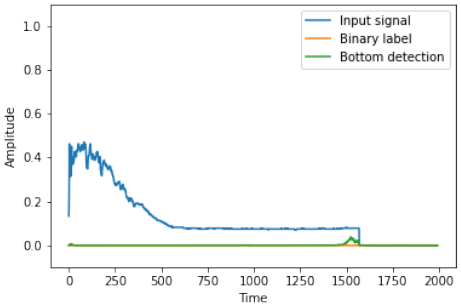
(a) Sample ping 1, GRU model 2



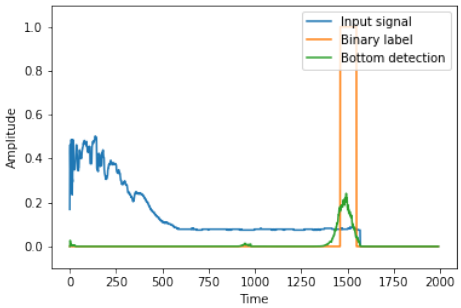
(b) Sample ping 2, GRU model 2



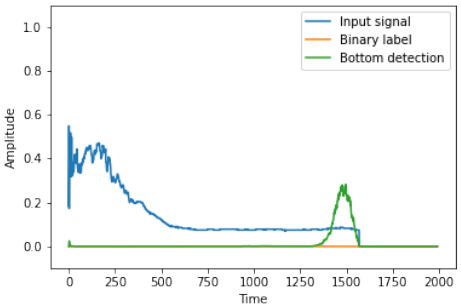
(c) Sample ping 3, GRU model 2



(d) Sample ping 4, GRU model 2

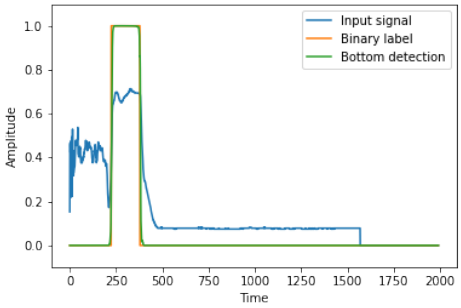


(e) Sample ping 5, GRU model 2

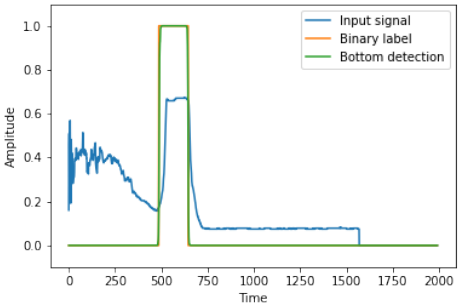


(f) Sample ping 6, GRU model 2

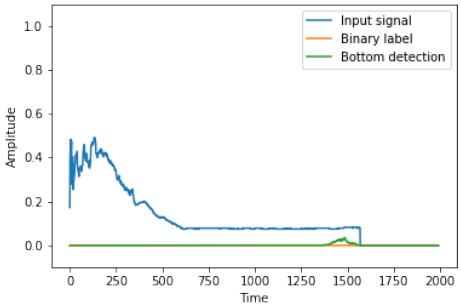
Figure D.0.5: Sample pings for qualitative analysis of GRU model 2.



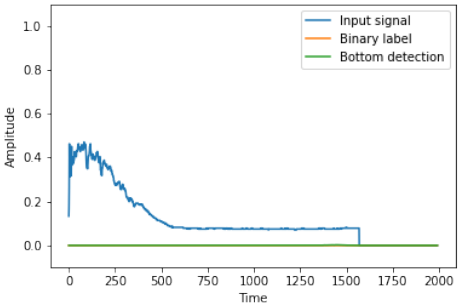
(a) Sample ping 1, GRU model 3



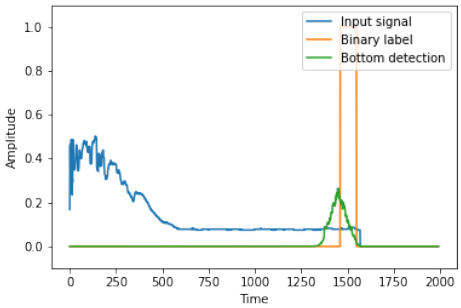
(b) Sample ping 2, GRU model 3



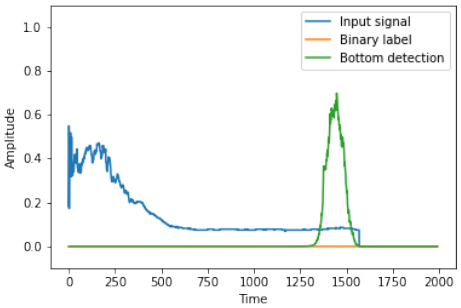
(c) Sample ping 3, GRU model 3



(d) Sample ping 4, GRU model 3

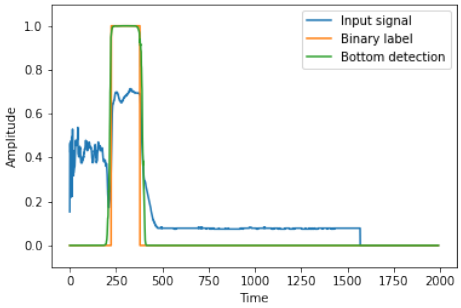


(e) Sample ping 5, GRU model 3

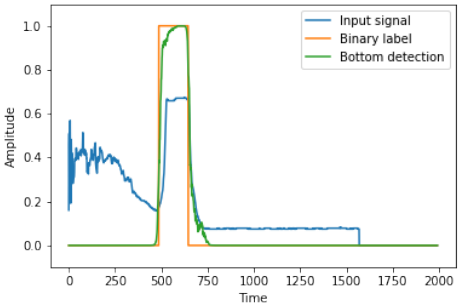


(f) Sample ping 6, GRU model 3

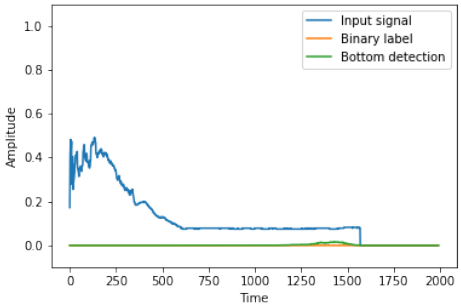
Figure D.0.6: Sample pings for qualitative analysis of GRU model 3.



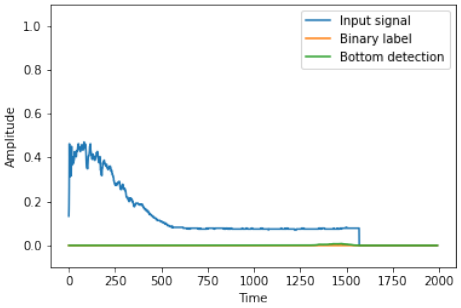
(a) Sample ping 1, GRU model 4



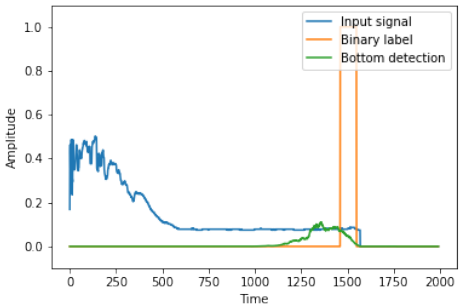
(b) Sample ping 2, GRU model 4



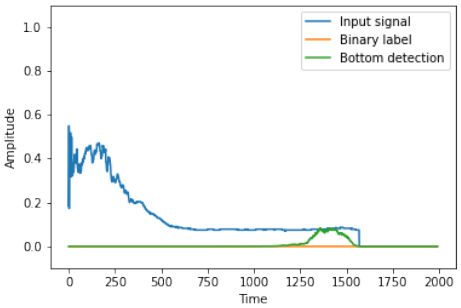
(c) Sample ping 3, GRU model 4



(d) Sample ping 4, GRU model 4

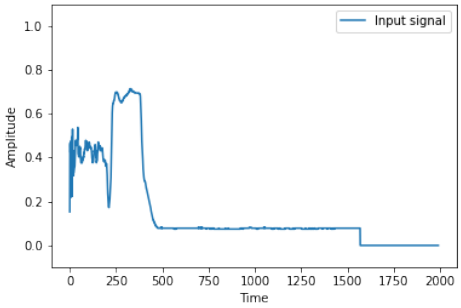


(e) Sample ping 5, GRU model 4

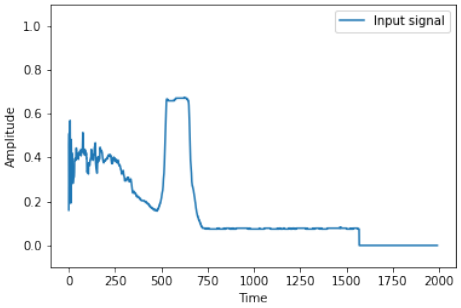


(f) Sample ping 6, GRU model 4

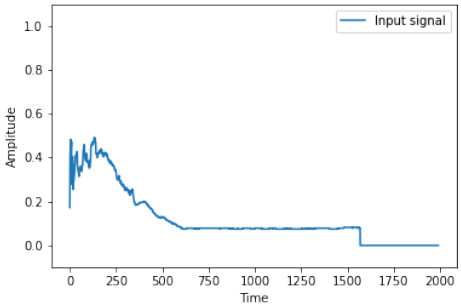
Figure D.0.7: Sample pings for qualitative analysis of GRU model 4.



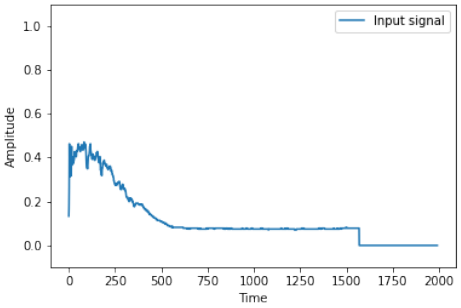
(a) Sample ping 1, input



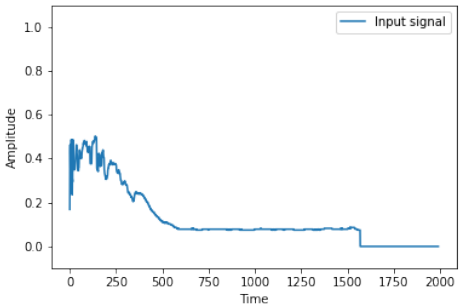
(b) Sample ping 2, input



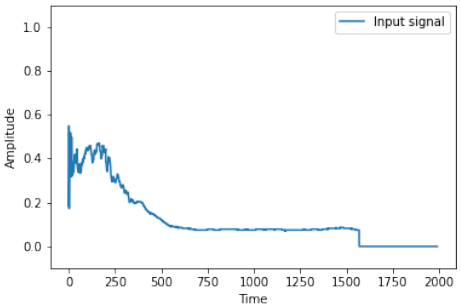
(c) Sample ping 3, input



(d) Sample ping 4, input



(e) Sample ping 5, input



(f) Sample ping 6, input

Figure D.0.8: Input of the pings used for qualitative analysis.

Appendix E

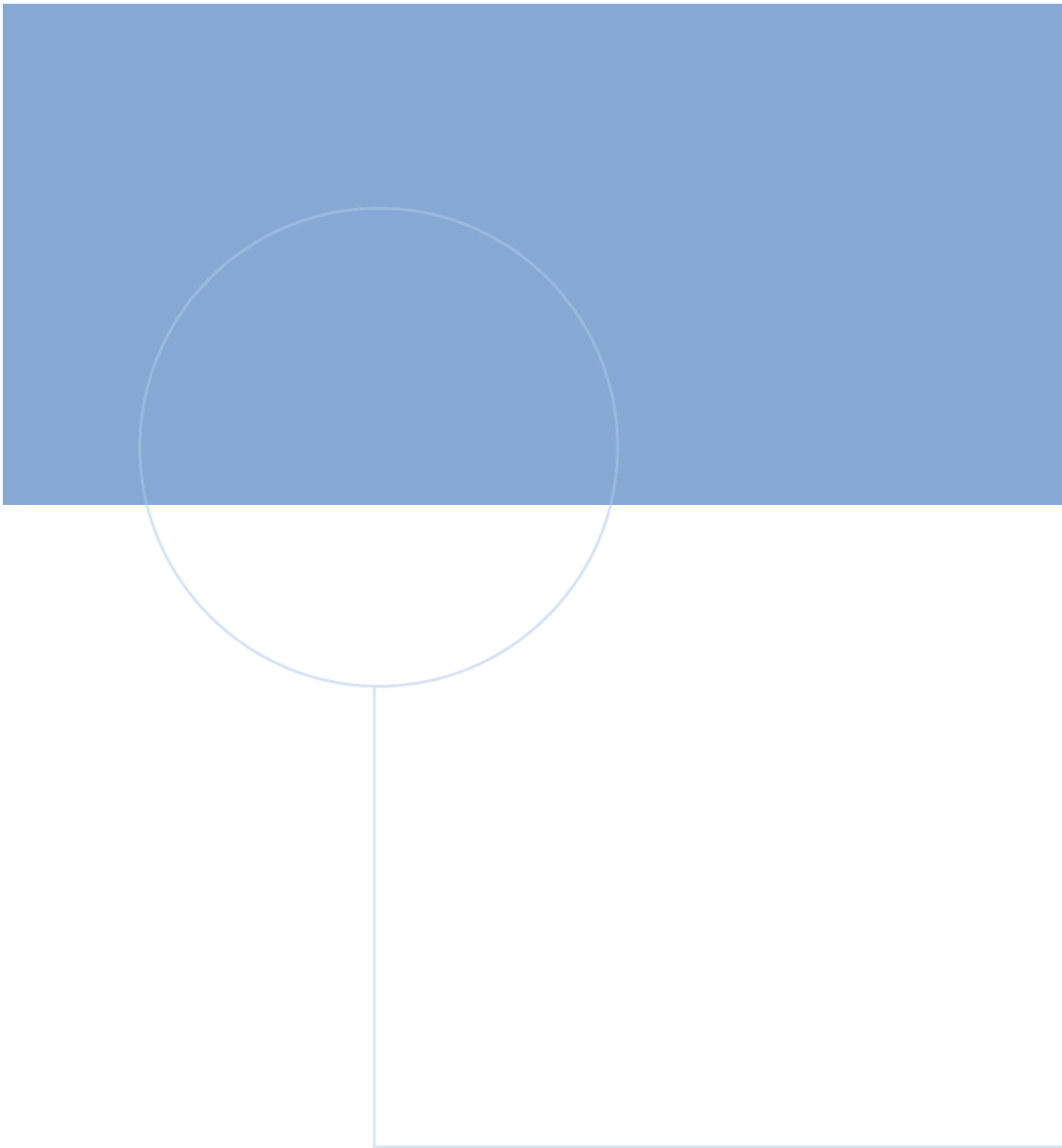
Inference on DIGI board

	Inference times	
Model name	Average (ms)	Total (s)
Simple RNN Model 1	17.71	88.5331
Simple RNN Model 1 quantized	17.67	88.3420
Simple RNN Model 2	6.829	34.1470
Simple RNN Model 2 quantized	6.845	34.2273
Simple RNN Model 3	5.975	29.8748
Simple RNN Model 3 quantized	5.968	29.8407
GRU model 1	21.91	109.5638
GRU model 1 quantized	22.05	110.2580
GRU model 2	26.40	132.0006
GRU model 2 quantized	26.41	132.0514
GRU model 3	51.35	256.7456
GRU model 3 quantized	51.45	257.2596
GRU model 4	21.97	109.8654
GRU model 4 quantized	21.92	109.6121

Table E.0.1: Table displaying the average and total inference times for each model. All inference times are from testing on 5000 pings from the test dataset using python code running on CPU.

Model name	Initialization footprint (MB)	Total footprint (MB)
Simple RNN Model 1	50.78	50.78
Simple RNN Model 1 quantized	56.81	56.81
Simple RNN Model 2	19.83	22.86
Simple RNN Model 2 quantized	25.29	25.54
Simple RNN Model 3	19.04	19.04
Simple RNN Model 3 quantized	20.74	21.00
GRU model 1	59.57	59.94
GRU model 1 quantized	64.53	64.79
GRU model 2	67.99	71.18
GRU model 2 quantized	79.03	79.29
GRU model 3	157.39	157.71
GRU model 3 quantized	141.33	143.23
GRU model 4	70.68	71.53
GRU model 4 quantized	78.22	78.49

Table E.0.2: Table displaying approximate memory footprints for each model.



NTNU

Norwegian University of
Science and Technology