APPLICATIONS OF DEEP CONVOLUTIONAL NEURAL NETWORKS TO PASSIVE ACOUSTIC MONITORING OF BALEEN WHALES

by

Mark Thomas

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
February 2024

For Elizabeth.

Table of Contents

List of Tables . . . . . **vi**

List of Figures . . . . . **ix**

Abstract . . . . . **xiv**

Acknowledgements . . . . . **xv**

Chapter 1 Introduction **1**

1.1 Contributions 4

1.2 Thesis Organization 6

Chapter 2 Background and Related Work **8**

2.1 Neural Networks 8

2.2 Convolutional Neural Networks 11
2.2.1 Deep Residual Learning 15
2.2.2 Additional Advancements 17

2.3 CNN-based Detection Models 19
2.3.1 Two-stage Detection Models: Region-based CNNs 20
2.3.2 Single-stage Detection Models 22

2.4 Performance Metrics for CNNs and R-CNNs 26
2.4.1 Precision, Recall, and F-1 Score 26
2.4.2 Using Average Precision to Describe Detection Performance . 27

2.5 Visual Representations of Acoustic Data 29
2.5.1 Linearly-scaled Spectrograms 30
2.5.2 Mel-scaled Spectrograms 31
2.5.3 Novel Representation: Stacked & Interpolated Spectrograms . 32

2.6 Applications of Machine Learning to PAM 34

2.7 Applications of Convolutional Neural Networks to PAM 35

Chapter 3 Deep Convolutional Learning for PAM **38**

3.1 Acoustic Data Sets 40

3.2 CNNs for Spectrogram Classification 40

3.2.1 Data set and Methods . . . 41
3.2.2 Experimental Results . . . 46

3.3 R-CNNs for Vocalization Detection . . . 48
3.3.1 Data set and Methods . . . 48
3.3.2 Experimental Results . . . 49
3.3.3 DCS Use and Adaptability . . . 50

Chapter 4 Learning from Unlabeled Passive Acoustic Data . . . 52

4.1 Background, Data Sets, and Methods . . . 52
4.1.1 Acoustic Data Sets . . . 53
4.1.2 Semi-Supervised Learning . . . 59

4.2 Experimental Results . . . 62
4.2.1 Baseline vs. Semi-supervised . . . 63
4.2.2 Out-of-Distribution Performance . . . 65

Chapter 5 Operational DCS . . . 67

5.1 Background and Requirements of Operational DCS . . . 67

5.2 Standardized methods for training and evaluation . . . 68
5.2.1 Data Set Creation . . . 69
5.2.2 Model Training . . . 73
5.2.3 Model Evaluation . . . 75
5.2.4 Inference on Edge Devices . . . 77

5.3 Automated DCS Results . . . 79
5.3.1 Blue, Fin, Right, and Sei Whale Detector . . . 79
5.3.2 Humpback and Infrasonic Fin Whale Detector . . . 85
5.3.3 Minke Whale Pulse Train Detector . . . 87

5.4 Model Updates via Transfer Learning . . . 88
5.4.1 Adapting to New Noise Sources . . . 89
5.4.2 Adapting to New Locations . . . 92
5.4.3 Discussion . . . 94

5.5 Updating Models Parameters Over-the-Air . . . 94
5.5.1 Model Quantization . . . 95
5.5.2 Transmitting Partial Networks . . . 97
5.5.3 Discussion . . . 100

Chapter 6 Conclusion and Future Work . . . 103

6.1 Keeping Up with the Evolution of Machine Learning . . . 104
6.1.1 Algorithms and Architectures . . . 104

6.1.2 Technological Tools and Resources 104
6.2 Future Work: Spatio-temporal Contextual Awareness 106
Bibliography **108**

List of Tables

3.1 Deployment locations and recording depths of the AMARs used in collecting the ESRF data. Note that a recorder was deployed at station 3 near Sable Island, however it was lost either due to burial in moving sand or fishing gear. 42

3.2 Number of files and the distribution of each acoustic source for the training, validation, and test sets. 43

3.3 Mean performance and 95% confidence intervals of ten training/testing runs using different random number generator seeds. 46

3.4 Median values of average precision (AP) evaluated over various IoU thresholds as described in the COCO Detection Challenge 50

4.1 Deployment locations and recording depths of the AMARs used in collecting Sets A and B. 56

4.2 Annotation distribution for Sets A and B separated at the acoustic source level. 58

4.3 Performance comparison of the baseline and semi-supervised CNN architectures in terms of precision, recall, and F-1 score, measured on the validation data of Set A: Bay of Fundy. The best performing models are highlighted in bold. 64

4.4 Performance comparison of the baseline and semi-supervised CNN architectures in terms of precision, recall, and F-1 score, measured using the data from Set B: Atlantic OCS. The best performing models are highlighted in bold. 65

5.1 Estimated number of parameters (measured in millions (M)) , floating-point operations per second (FlOPS, measured in billions (B)), and memory footprint (measured in megabytes (MB)) of the YOLO and EfficientDet families of models. 80

5.2 Overall performance of the YOLO models trained to detect the vocalizations of BW, FW, RW, and SW 82

5.3 Overall performance of the EfficientDet models trained to detect the vocalizations of BW, FW, RW, and SW 83

5.4 Per-file performance comparisons of the YOLO detector against JASCOs contour detector. No comparative metrics were available for detecting North Atlantic right whales (RW). 84

5.5 Measured runtimes of the YOLO detection models when evaluating 110 files each 10 minutes long at sampled at 8kHz. Inference was parallelized on a 24-thread CPU. Measurements include the the process of loading the models into memory, creating the TFLite interpreters, streaming the data, and performing the FFT. 85

5.6 Performance of the YOLO-s/l models trained to detect infrasonic FW vocalizations. 87

5.7 Performance of the YOLO-s/l models trained to detect HB vocalizations. 87

5.8 Performance of the YOLO-s/l models trained to detect both HB and infrasonic FW vocalizations. 87

5.9 Performance of the YOLO models trained to detect minke whale pulse train vocalizations. 88

5.10 Resulting performance of the YOLO models frozen at layer 9 and fine-tuned on an increasing number of glider self-noise examples. 90

5.11 Resulting performance of the YOLO models frozen at layer 24 and fine-tuned on an increasing number of glider self-noise examples. 91

5.12 Performance of the YOLO-s FW/HB detector fine-tuned on data from the Pacific Ocean compared to the performance of a YOLO-s model trained from scratch on that same data. 93

5.13 Performance of the BW/FW/RW/SW detector evaluated using two levels of model quantization (16-bit floating points and 8-bit integers). 96

5.14 Size of the full YOLO models quantized using 16-bit floating points and 8-bit integers.. 96

5.15 Run time and power consumption estimates of the YOLO models on the Tinker board. A * indicates that the active cooling fan on the SBC turned on and subsequently spiked power usage. . . 97

5.16 Model size measured in kilobytes (KB) and number of parameters measured in millions of the second TFLite models quantized at 16-bits. 98

5.17 Run-time and power consumption of the split models on the Tinker board. After 10 min, the experiments for the large and x-large models were stopped as they were no longer faster than real-time and deemed non-operational. A * indicates that the active cooling fan on the SBC turned on and subsequently spiked power usage. 98

5.18 Size of the serialized network weights measured in kilobytes (KB) after being split at the specified layer and quantized to 16-bits. 100

5.19 Time to recompile and peak power consumption of each YOLO model on the edge device. A * indicates that the active cooling fan on the SBC turned on and subsequently spiked power usage. 100

List of Figures

2.1 Three historically common activation functions: a) sigmoid (σ), b) hyperbolic-tangent (tanh), and c) rectified linear unit (ReLU). 9

2.2 Example figure of a fully-connected neural network with four inputs (x_i), three outputs ($\hat{y}_i$), and three hidden layers each with 5 neurons ($h_i^{(j)}$), where i indexes the nodes of the network for each layer and j indexes the layer number. 10

2.3 Example illustrations depicting the first two steps (top to bottom) of a convolution operation using a 3x3 filter (shaded) being applied over a 5x5 input (green) and the corresponding output (orange). The code used to create the example illustrations was downloaded from the technical report of Dumoulin et al. [27]. 13

2.4 Example network diagram of a convolutional neural network (CNN) with an input spectrogram (x), two convolutional layers displaying the size and number of filters of each layer, two 2x2 max pooling layers, and two fully connected layers (fc) displaying the number of nodes in the fully connected layer. The first fully connected layer is simply a flattened version of the output of the final max pooling layer. The final fully connected layer assumes the number of classes is 10 and a softmax activation function. 15

2.5 Example graphs of a residual building block (left) and bottleneck residual building block (right). 16

2.6 Diagram containing the three networks used in the Faster R-CNN model architecture. The first network consists of a CNN "backbone" used in extracting features. These features are sent to a region-proposal network (RPN) in order to determine where in the features there may be regions of interest (RoIs). Finally the CNN features and RoIs are combined using a pooling/alignment operation and passed to fully connected layers in the "head" network in order to optimize for classification (CE loss) and bounding box regression (L-1 loss). 22

2.7 Example diagram of the EfficientDet model architecture. The model is composed of an EfficientNet CNN backbone which produces features at various scales. These features are passed to a feature pyramid network (BiPFN), of which each layer in the BiFPN is fully connected to two convolutional layers used in producing bounding box predictions and class probabilities. The convolutional blocks within the backbone network that are passed to the BiFPN are denoted in blue. 24

2.8 Example diagram of the YOLOv5 model architecture. The model is composed of a CNN backbone with convolutional blocks and a SPP block. Three scales of learned features are passed as input to a PANet referred to as the "neck". Finally, three additional convolutional blocks in the head network are combined via non-max suppression. The convolutional/SPP blocks within the backbone network that are passed to the PANet are denoted in blue and purple. 25

2.9 Example depiction of the intersection (left) and union (right) between a ground truth bounding box (blue) and predicted bounding box (orange). The intersection-over-union (IoU) of these two bounding boxes is computed as the division of these two areas, respectively. 28

3.1 An example spectrogram containing two consecutive sei whale vocalizations, the first from times 2-4 seconds and the second from roughly 6-8 seconds. The spectrogram is duplicated in order to demonstrate two different systems: (a) the predicted output of system trained for spectrogram classification, and (b) the predicted output of a system trained for vocalization detection. 39

3.2 Individual deployment locations for each of the 19 AMARs used in collecting the ESRF data set. The recording devices were deployed over 12 months along the Scotian Shelf off the coast of Atlantic Canada. Note that a recorder was deployed at station 3 missing from this figure near Sable Island, however it was lost either due to burial in moving sand or fishing gear. 41

3.3 Example spectrograms for each of the three whale species: a) blue whales, b) fin whales, c) sei whales, and d) non-biological noise. Dashed vertical lines depict the start and end times of the expert annotations centered within the 30s window used in the data creation sampling routine. For visualization purposes filtered versions of the spectrograms are being shown on a log-frequency scale so that the reader can more easily identify the vocalizations. 44

3.4 Normalized confusion matrices of the two best performing classifiers in terms of F-1 score. 47

3.5 Example annotations (top row) and corresponding predictions made by the R-CNN (bottom row) for several example vocalizations produced by the three species of interest. As previously mentioned vocalizations have only been partially labelled, for example: in row 1 column 5, there appears to be several sei whale vocalizations occurring consecutively, however, only one has been annotated. Interestingly, the R-CNN has detected two vocalizations and therefore the reported metrics for this test instance would be artificially low. 49

3.6 Example predictions (green) and expert annotations (yellow) presented in PAMlab. The file used in this example was randomly selected from a set of data not used during training, validation, or testing. The acoustic file was recorded in June 2015 off the coast of Nova Scotia in the Gully Marine Protected Area. 51

4.1 Deployment Locations: (a) Set A: three AMARs were deployed in the Bay of Fundy between August and November, 2015. (b) Set B: six AMARs were deployed along the Atlantic Outer Continental Shelf (OCS) from late November 2017 to June 2018. . 55

4.2 The data distribution of all files from the training and validation data set (Set A) plotted over time, factored by the deployment location (station) and annotation level (fully, partially, or non-annotated). Files that contain at least one vocalization made by minke whales known as a "pulse train" are plotted as blue squares. Files that either explicitly do not (fully-annotated files) or possibly contain a pulse train (partially/non-annotated) are plotted using red X's (i.e., possible sources of false alarm). The plotted figure only contains data until November 1 of 2015 as there were no minke whale annotations during the final two months of the deployment. A slight *jitter* was added to each point in order to distinguish files from the same date. 57

4.3 A spectrogram containing a minke whale pulse train annotated with a long blue bounding box along with several overlapping humpback whale vocalizations individually annotated using yellow bounding boxes. 58

4.4 Two example data instances before and after being passed through the SpecAugment data augmentation routine. The top row of spectrograms contains consecutive minke whale pulse trains below 400Hz as well as several humpback whale vocalizations bounded between 400 and 600Hz. The second row of spectrograms contains a possible source of false alarm pertaining to humpback whale vocalizations bounded between 200 and 500Hz coupled with several fin whale 20Hz pulse vocalizations. Column one (starting from the left) contains the original data instance. Column two contains the same data instance after being "time-warped" using SpecAugment ($W = 200$). Column three contains two example time and frequency masks per instance using maximum mask widths of $T = 50$ and $F = 50$. 61

5.1 Outline of the pipeline used in creating a DCS development data set. The requirements to the pipeline include a configuration file that may specify several parameters used when running the FFT and a SQL query for selecting the annotations from the database. 71

5.2 Image of the Asus Tinker Edge T SBC. Image source: product information page from ASUS. 78

5.3 Example AP@.5 of the YOLO-l model evaluated after each epoch. The model with 9 frozen layer is able to make major adjustments to the weights of the neural network due to fewer weights being frozen, but over time the difference in performance is negligible. The figure is faceted by the number of new instances supplied to the model for training. 92

5.4 AP@0.5 measured over 300 epochs of training a model from scratch versus using transfer learning. 93

Abstract

Research into automated detection and classification systems (DCS) of marine mammal vocalizations in acoustic recordings is expanding internationally due to the necessity to analyze large collections of data for conservation purposes. This work discusses historical implementations of marine mammal DCS and introduces more recent developments using deep learning and convolutional neural networks (CNNs). A novel application of region-based CNNs (R-CNNs) is used for the development of a DCS that is capable of detecting the vocalizations of endangered baleen whales in both time and frequency. A state-of-the-art approach to semi-supervised learning is adapted for the task of spectrogram classification in order to address the issue of data scarcity commonly found in passive acoustic monitoring (PAM) and increase the performance of the aforementioned systems by as much as ten percent. Finally, an operational marine mammal DCS is developed to address real-world constraints like data set variance, computational limitations, and over-the-air model updates. This work contributes to advancing marine mammal vocalization detection and supports conservation efforts through PAM.

Acknowledgements

Throughout the duration of my doctoral studies, I have had the privilege of working with many incredibly talented, thoughtful, and kind individuals in industry, government, and academia. To name them all here would exceed the page limit of this document.

Much of this work was completed in collaboration between myself and researchers at JASCO Applied Sciences and was made possible through a number of grants spanning several years: NSERC Engage, Mitacs Accelerate, and Department of Defence IDEaS. I would like to sincerely thank several members of the JASCO Applied Sciences team, notably: Katie and Julien for their receptivity and allowing me to pick their brain with any questions I may have related to marine biology; Briand for his help in data collection, data pre-processing, and Java development; and Nisarg for his assistance in operationalizing the research described in this work. I was also fortunate to work at Google X through their AI Residency program. I consider many of the individuals that I was able to work with during this time to be some of the smartest people I have ever met and owe a immense amount of gratitude.

I am overwhelmingly grateful to my co-supervisors Bruce and Stan. First, to Bruce I simply cannot express how thankful I am for the invitation to work on the research outlined in this document and introducing me to a field in which I was at first oblivious and now hope to continue working on for many years to come. I thank you for your guidance and suggestions relating to passive acoustic monitoring and digital signal processing in which your years of expertise and ability to foster community in this space are overtly apparent. To Stan, I thank you for being a calm and passionate researcher who is both a fervent learner and inspiring to so many others. I am astounded by your continued grace, encouragement, and support of my studies especially during several very challenging years.

Finally, to my family. There are no words that I could write on this page that are capable of characterizing how grateful I am for shaping me into the person I am today and the happiness you have brought me. I love you all.

Chapter 1

Introduction

The majority of the Earth's surface (roughly 70.8%) is covered by the ocean [45]. It is therefore not surprising that the ocean plays a critical role in our planets climate and its many ecosystems. However, the health of the Earth's ocean has increasingly been threatened by a wide range of human activity, including: climate change, over-extraction of resources, and pollution [34, 40]. Many of us are well aware of–if not deeply connected to–some of the anthropogenic threats just listed. Particularly those whose striking imagery has been captured and communicated broadly (e.g., receding sea ice, coral reef bleaching, and waterways filled with plastic or industrial waste). Another significant issue that plagues the ocean is that of *noise pollution*. Anthropogenic activities like off-shore drilling and pile-driving, seismic surveys, vessel traffic, and military exercises have altered the acoustic landscape of the ocean [52]. These activities can generate sounds at frequencies that disrupt the communication and navigation of marine life.

For all of these reasons, it is becoming increasingly important to study marine environments and the species that inhabit them. One such group of species are the subset of marine mammals known as cetaceans. Increases in the number of ship strikes in common breeding and feeding grounds have lead to the deaths of hundreds of cetaceans [48, 62]. Entanglement in fishing gear is another source of concern in less travelled marine traffic environments and has also been responsible for a sizable number of cetacean casualties [35]. Many of the species belonging to this taxonomy produce vocalizations for the purposes of navigation, feeding, breeding, and intraspecies communication [135]. Marine mammals must now compete with the aforementioned anthropogenic acoustic sources in order to be heard by their co-inhabitants [78]: a concept known as acoustic masking. Making matters worse, research has shown that noise created by human activity causes temporary to permanent hearing loss in marine mammals; severely altering their way of life [109] or in some cases leading to

strandings [125].

Policy makers, government officials, and environmental groups, among others, have expressed their concern for the well being of marine mammals and have presented solutions to mitigate the risk that human activity poses to these animals, such as: tightening speed restrictions on vessels, removing fishing gear, and pausing military exercises or construction when marine mammals are detected in a given area. However, often times our limited understanding of cetacean behaviour and/or inability to detect species presence in real-time has resulted in sub-optimal enactment of these mitigation strategies [23].

Ones ability to make informed decisions and mitigate further harm to these species hinges on a robust and accurate method of determining species presence. Fortunately, marine mammal vocalizations are predominantly distinct on a per-species level, allowing one to determine which species–if any–are present in a given area by detecting the sounds (i.e., vocalizations) they emit [135]. Using the total number of detected vocalizations, it is possible to estimate the abundance of a given species present in a recording area [76, 82] and potentially improve our understanding of cetacean behaviour. The process of analyzing acoustic data for the purpose of estimating species presence and abundance is more generally known as Passive Acoustic Monitoring (PAM). As one may expect, PAM is not unique to marine environments [30, 110, 118], however for the case of marine mammals, PAM offers several advantages. First, by passively monitoring for marine mammal vocalizations–as opposed to actively monitoring (e.g., using active SONAR)–we reduce the risk of altering the species' behaviour [86]. A similar argument can be made when comparing passive acoustics to visual surveys on-board vessels. Second, visual surveys at sea, using aircraft, or through satellite imagery are limited to the short periods of time when the species of interest is in close proximity to the ocean surface, acclimate weather conditions not withstanding. Finally, as a function of time, the financial cost of passive acoustics out-weighs visual surveys, active SONAR, and other means of determining presence/absence [82]. For the remainder of this work, the use of the acronym PAM implies passive acoustic monitoring of cetaceans only.

As alluded to above, PAM offers a practical approach to measuring marine mammal presence and make subsequent abundance estimates. In terms of threat mitigation, one is predominantly concerned with detecting species presence. The primary challenge in determining species presence via PAM, is developing a robust detection algorithm. Research and development of detection algorithms is accomplished using recorded audio and accompanying ground truth data in the form of annotations. Once a detection system has been developed and proven to be reliable, it may be used in real-time applications and assist in threat mitigation. Ideally, the ground truth annotations would be created by a trained expert (e.g., marine biologist) who would analyze all of the available PAM data to provide an accurate count of the individual marine mammal vocalizations: start/end times of each vocalization and corresponding frequency bounds. However, the volume that PAM data is regularly collected has long-since surpassed the speed at which it can be analyzed by human experts. In many cases, PAM data is collected continuously over several months via moored acoustic recording devices. A single acoustic recorder typically collects multiple terabytes of PAM data per deployment and collections of PAM data are starting to be described on a petabyte-scale [30, 127]. In reality, when the acoustic recording devices are retrieved, often as little as three percent of the entirety of the data is manually annotated due to time and/or resource restrictions [59]. As a result, machine learning (ML) has been a useful tool in developing automated Detection and Classification Systems (DCS) of marine mammal vocalizations for many years [5, 31, 81, 100]. More recently, researchers have started to develop neural network based DCS under the ML sub-field known as deep learning (DL) [56, 105, 117].

The scope of this document is to briefly overview the past, present, and possible future of ML/DL-based DCS for PAM. This research is limited to the cetacean sub-order known as Mysticeti (i.e., baleen whales), however, the applications of this research may be applied to species outside of this taxanomy. Related work corresponding to other species of marine mammals (e.g., Odontoceti; or toothed whales) is included due to the relatively small amount of related work that exists.

This document describes several working applications using original data collected for scientific research that could have substantial implications towards environmental policy and conservation efforts. The data was manually selected based on the target

species of interest, however, it has not been cleaned and manipulated unlike many research areas in ML that use common sets of image data or pre-processed acoustic recordings.

1.1 Contributions

The main contributions of this research are as follows:

1. A convolutional neural network (CNN) is trained to classify spectrograms containing the vocalizations of endangered baleen whales.

2. Region-based CNNs (R-CNNs) are developed to detect individual baleen whale vocalizations in time and frequency.

3. State-of-the-art (SOTA) approaches to semi-supervised learning for computer vision are adapted to the task of spectrogram classification and the classifiers trained through semi-supervised learning are demonstrated to be more robust to out-of-distribution examples.

4. The research above is packaged for operational use in an industry PAM setting, running DCS on edge devices, and performing over-the-air updates when networking bandwidth is limited.

This research has resulted in five publications (one journal article and four conference proceedings), and a number of contributed/invited talks:

Publications

- A. Theissler, **M. Thomas**, M. Burch, F. Gerschner (2022) "ConfusionVis: Comparative evaluation and selection of multi-class classifiers based on confusion matrices". Knowledge-Based Systems

- **M. Thomas**, B. Martin, and S. Matwin, (2021) "Leveraging Unlabelled Data through Semi-supervised Learning to Improve the Performance of a Marine Mammal Classification System". From Shallow to Deep: Overcoming Limited and Adverse Data Workshop, International Conference on Learning Representations (ICLR)

- **M. Thomas**, B. Martin, K. Kowarski, B. Gaudet, and S. Matwin, (2019) "Detecting endangered baleen whales within acoustic recordings using R-CNNs". Joint Workshop on AI for Social Good, Neural Information Processing Systems (NeurIPS)

- **M. Thomas**, B. Martin, K. Kowarski, B. Gaudet, and S. Matwin, (2019) "Marine mammal species classification using convolutional neural networks and a novel data representation". European Conference on Machine Learning (ECML)

- **M. Thomas**, (2019) "Towards a novel data representation for classifying acoustic signals". Canadian Conference on Artificial Intelligence

Technical Reports

- **M. Thomas**, N. Patel, E. Maxner, B. Martin (2023) "Operational Neural Networks for Marine Mammal Detection and Classification" Technical report for Defence Research and Development Canada (DRDC).

Invited Talks

- "Letting marine mammal Vocalization neural networks swim on their own", Machine Learning Advances for Marine Acoustics and Imagery Data, MERIDIAN and Ocean Networks Canada, Halifax, NS, Nov 2022

- "Potential and novelty of deep learning (panel discussion)", Detection and Classification in Marine Bioacoustics with Deep Learning, MERIDIAN and Ocean Networks Canada, Victoria, BC, Nov 2019

- "Detecting endangered baleen whales within acoustic recordings using R-CNNs", Detection and Classification in Marine Bioacoustics with Deep Learning, MERIDIAN and Ocean Networks Canada, Victoria, BC, Nov 2019

- "Marine mammal species classification using convolutional neural networks", Workshop on the Analysis of Acoustic Landscapes, University of São Paulo, São Paulo, Brazil, Sep 2018

Contributed Talks

- "An end-to-end approach for true detection of low frequency marine mammal vocalizations", 178th Meeting of the Acoustical Society of America, San Diego, California, Dec 2019

- "Deep learning detection and classification of baleen whale vocalizations using a novel data representation", CIFAR Deep Learning and Reinforcement Learning Summer School (DLRLSS), University of Alberta, Edmonton, AB, Jul 2019

- "Marine mammal species classification using convolutional neural networks", 8th International DCLDE Workshop, Paris, France, Jun 2018

1.2 Thesis Organization

The remainder of this document is outlined as follows.

In Chapter 2, two subsets of models frequently used for the tasks of image classification and object detection are reviewed: CNNs and single/two-stage detections models, respectively. These models make up the basis of the DL-based DCS developed in this work. An overview of visual representations of acoustic data, namely spectrograms, is presented and a justification is made for why such a representation is useful in PAM. Metrics for measuring the performance of DL-based DCS are introduced. Finally, related work that makes use of feature engineering coupled with ML algorithms and more recent work applying DL to PAM is discussed.

The majority of the related work outlined above can be categorized under the ML paradigm of "supervised learning". In Chapter 3 supervised learning strategies to DCS development are presented. First a CNN is developed to classify spectrograms possibly containing the vocalizations of endangered baleen whales. Next, a novel approach to detecting individual vocalizations within a spectrogram is put forward that uses R-CNNs as a model architecture. In doing so, a necessary distinction between spectrogram classification and vocalization detection is made.

Chapter 4 outlines the current limitations of supervised learning for DCS development and presents an alternative approach to updating the weights of a neural

network under the ML paradigm of "semi-supervised learning", whereby, unlabeled data coupled with annotated examples can be used to produce more generalizable classification systems.

In Chapter 5 an operational marine mammal DCS is developed that addresses several of the constraints found in real-world applications that are often overlooked in a purely research environment, such as: the variance and bias of PAM data sets, computational limitations, power efficiencies, and the ability to update models on autonomous platforms.

Lastly, Chapter 6 contains concluding remarks and proposes further research to include spatio-temporal context in the development of marine mammal DCS.

Chapter 2

Background and Related Work

The accelerated pace at which the field of machine learning continues to evolve unquestionably makes for a difficult to write introductory chapter. This chapter therefore hopes to describe several concepts at a level of granularity that both informs the reader and encourages further investigation, while at the same time, avoids becoming out-dated within a matter of months–if not weeks–of the time of writing.

In this chapter, the fundamental concepts of neural networks (NNs) are outlined, setting the stage for an introduction to CNNs used for the task of classification. Following which, CNN-based model architectures used in performing detection are introduced. To monitor/benchmark the NNs throughout this work, several metrics are mathematically defined. Finally, related work pertaining to more traditional ML algorithms and those using DL in some capacity are reviewed.

2.1 Neural Networks

Neural networks are computational models that consist of interconnected layers of *artificial neurons*. They are designed to learn representations of functions or complex patterns using input data. Representations are learned through the process of training and optimization. A basic NN is comprised of an input layer, one or more hidden layers, followed by an output layer, and each layer generally consists of multiple neurons. A single neuron can be defined as a non-linear mapping of some input x to some output $h(x)$. This mapping is typically split into two steps. The first step being a linear combination of the neurons inputs and model parameters. This step is often referred to as the "pre-activation" and defined as:

$$a(x) = \mathbf{W}^T x + b, \tag{2.1}$$

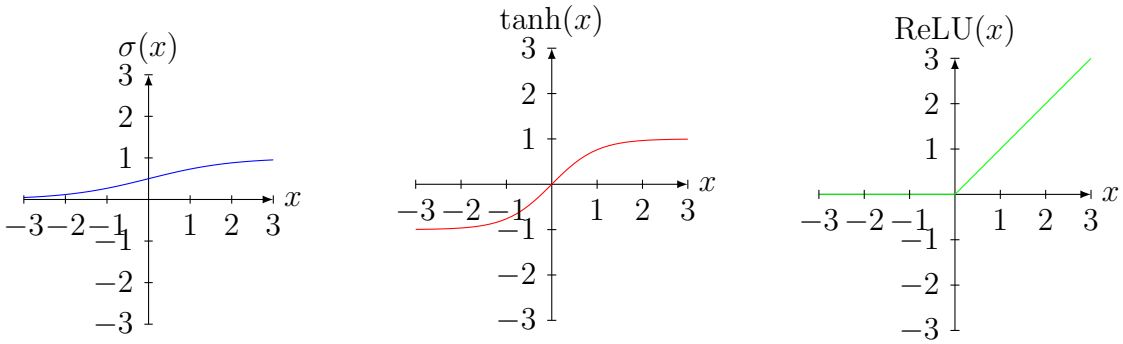
where x is the input to the neuron (i.e., the input data or the output of a previous neuron), $\mathbf{W}$ is a vector of weights that quantifies the connection between neurons, and

b is a bias term specifying the intercept of the affine transformation. The values of the weights and biases are determined using an optimization procedure that is described provided below. The second step is often referred to as the "post-activation" and simply applies a non-linear function to the output of Equation 2.1:

$$h(x) = f(a(x)), \tag{2.2}$$

where $a(x)$ is the output of the pre-activation defined above and the non-linear function f is referred to as an "activation" function.

As alluded to above, the activation function introduces non-linearity into the network which enables a networks ability to learn complex representations of the input data. Moreover, as the number of layers in the model increases and subsequent non-linearities are stacked, more complex non-linear relations can be learned. Historically, several common activation functions include the sigmoid function (σ), hyperbolic tangent (tanh), and rectified linear unit (ReLU) [84], each of which are defined and visualized in Figure 2.1. There is a great deal of existing research into alternative activation functions [3], some of which are adaptations to those listed above. For the purposes of this work, it is generally assumed that each NN employs ReLU activation functions in the hidden layers.



(a) Sigmoid (b) tanh (c) ReLU

$$\sigma(x) = \frac{1}{1+e^{-x}} \qquad \tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \qquad \text{ReLU}(x) = \max(0, x)$$

Figure 2.1: Three historically common activation functions: a) sigmoid (σ), b) hyperbolic-tangent (tanh), and c) rectified linear unit (ReLU).

Neural networks are often presented in the form of an acyclic graph like that of Figure 2.2. The graph depicted in this figure displays a neural network with four

inputs (x_i), three outputs ($\hat{y}_i$), and three hidden layers, each containing five neurons ($h_i^{(j)}$). In the notation above, i is used to index the node of the graph, and j indexes each node's corresponding layer. The provided graph is a common example of a fully-connected neural network (e.g., multi-layer perceptron (MLP)) whose namesake comes from the fact that each node is connected to every other node in the layers immediately before and after.

In keeping with the notation used so far, the mathematical operation performed at node $h_1^{(1)}$ is equivalent to:

$$h_1^{(1)} = f(w_1^{(0)}x_1 + w_2^{(0)}x_2 + w_3^{(0)}x_3 + w_4^{(0)}x_4 + b^{(0)}), \tag{2.3}$$

where the weights $w_i^{(0)}$ are depicted in the graph as the edges connecting nodes x_i and $h_i^{(1)}$.

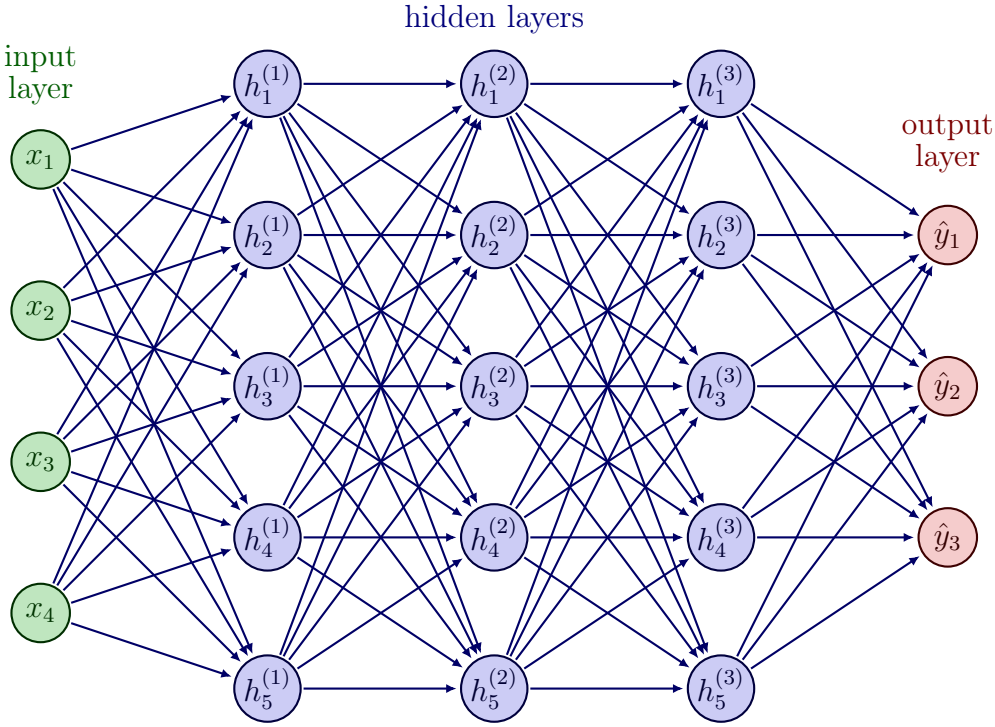


Figure 2.2: Example figure of a fully-connected neural network with four inputs (x_i), three outputs ($\hat{y}_i$), and three hidden layers each with 5 neurons ($h_i^{(j)}$), where i indexes the nodes of the network for each layer and j indexes the layer number.

The "training" or "learning" procedure of a neural network is conceptually straightforward and equivalent to finding values for each weight ($w_i^{(j)}$) and bias ($b^{(j)}$) belonging to a parameter set θ that minimizes a pre-defined loss function between the network's

predictions ($\hat{y}_i$) and the target values (y_i). When performing classification, a prediction $\hat{y}_i$ is interpreted as a score representing how confident the model is that the input pertains to class i. Together, the confidence scores can each be interpreted as un-calibrated probabilities of classifying the input correctly, for each class respectively [85]. The process of updating the weights and biases to minimize the loss function is generally achieved through *backpropagation* [101]. In backpropagation, the gradient of the loss function with respect to each weight and bias in the network is computed via the chain rule of calculus and these gradients are used to update the parameters following some optimization algorithm (e.g., gradient descent). The choice of loss function used in training a network is an area of research in its own right. The work presented throughout this document generally makes use of the cross-entropy loss function defined in Equation 2.4.

$$\mathcal{L}_{CE}(y, \hat{y}) = -\sum_{i=1}^{N} y_i \cdot \log(\hat{y}_i), \tag{2.4}$$

where N is the number of classes (e.g., $N = 3$ in the example network of Figure 2.2). Assuming the weights and biases of the neural network belong to the set θ, the update rule for the network parameters using gradient descent can be written as:

$$\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L} \tag{2.5}$$

where $\nabla_\theta \mathcal{L}$ is the gradient vector for each parameter in the network and α is a scalar value specifying the magnitude of the gradient update (also known as the "learning-rate"). The process of calculating gradients and updating the weights and biases is iteratively performed until the loss function converges to a minimum value, a specified number of passes over the entire data set (i.e., epochs) has occurred, or the training process has been halted to avoid over-fitting (i.e., early-stopping).

2.2 Convolutional Neural Networks

Convolutional neural networks were first introduced in the late 1980s [66] and later used to achieve state-of-the-art results in computer vision on the MNIST benchmark data set in 1998 [65]. However, CNNs were long thought to be too computationally inefficient until researchers from the University of Toronto found that by using graphics processing units (GPUs) they could handle many matrix multiplications in parallel

and accelerate the training process. Subsequently, this allowed for them to increase the number of hidden layers in the network and achieve significant improvements in accuracy for image classification tasks [60]. Since that time, CNNs have been known to perform well in cases where the training data of the network is structural (e.g., vectors containing time series data, tensors containing RGB image data). In contrast to the fully-connected neural network described above, the benefit of CNNs–specifically their ability to learn complex representations from structural data–is due to a technique known as parameter sharing, which involves using the same set of weights and biases on different regions of the input data. Moreover, parameter sharing in a CNN has an added benefit of reducing the number of parameters in the set θ.

In general, a CNN is simply a neural network that makes use of the mathematical *convolution* operation in lieu of the matrix multiplication in at least one layer of the network. In reality, most CNNs correspond to neural networks which are almost entirely made up of convolutional layers. A typical convolutional layer replaces the pre-activation defined in Equation 2.1 with a discrete convolution operation between the input (I) and a matrix or tensor of weights referred to as a kernel or *filter* (F). Assuming the input I is a tensor with height H, width W, and number of channels C (e.g., an RGB image) and the convolutional filter F has height h, width w, and channels c, the convolution operation can be written as:

$$O(i,j) = \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} \sum_{k=0}^{c-1} I(i+m, j+n, k) \cdot F(m,n,k) + b, \tag{2.6}$$

where b is an additional learnable bias term. Note that convolution as described above is computed "channel-wise" and therefore c and C must be equal. In much of this work, the input to the CNN is a spectrogram which can be interpreted as a gray-scale image and thus the initial number of channels in the first convolutional layer is equal to one.

The degree to which the filter is passed over the input in terms of horizontal and vertical units is known as the *stride* of the convolution operation. Typically, a stride of one or two units is used. Additional *padding* may be added to the input I prior to the convolution operation in order to convolve over each value of the input an equal number of times and maintain the size of the output (O). This degree of padding is referred to as "same" padding. If no padding is added the convolution operation

may be said to have "valid" padding [27]. The choice of filter size, stride length, and the amount of padding used in a convolution operation impacts the size of the corresponding output. Assuming the input has size $H \times W$, the filter has size $h \times w$, the stride of the operation across height and width are s_h and s_w, respectively, and similarly, the padding in height/width is p_h and p_w, then the dimension of the output can be calculated as:

$$\left\lfloor \frac{H - h + 2p_h}{s_h} + 1 \right\rfloor \times \left\lfloor \frac{W - w + 2p_w}{s_w} + 1 \right\rfloor \tag{2.7}$$

Figure 2.3 contains three examples depicting a convolution using a 3×3 filter, various levels of padding, and different stride values.

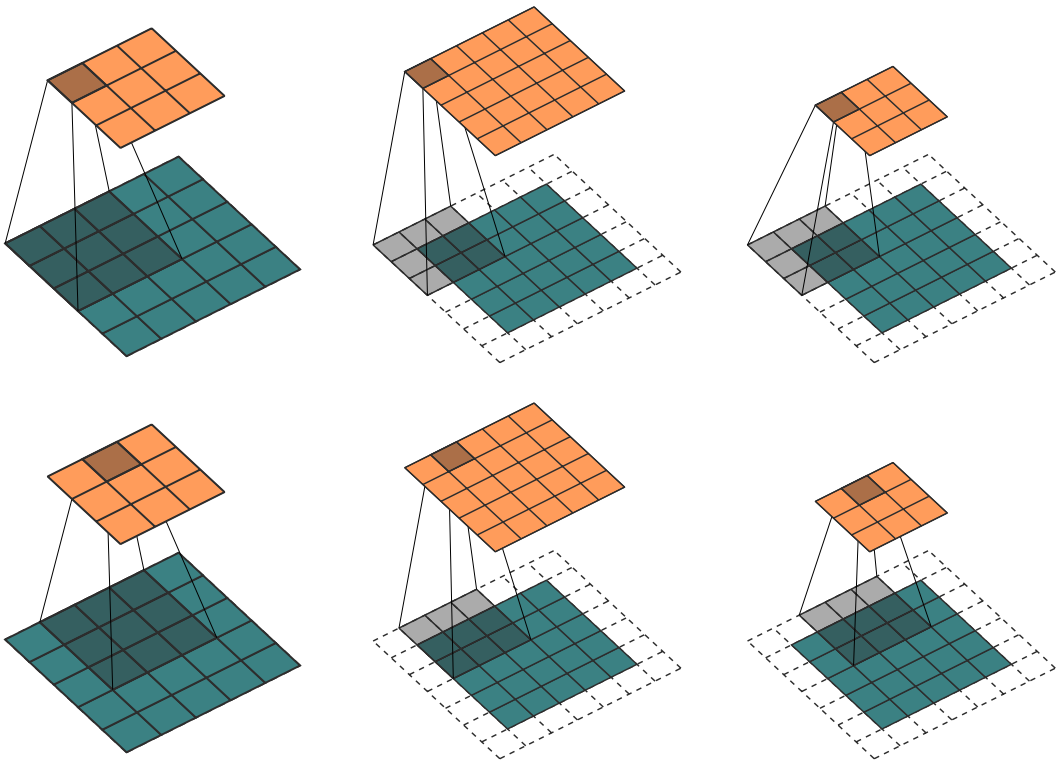


Figure 2.3: Example illustrations depicting the first two steps (top to bottom) of a convolution operation using a 3x3 filter (shaded) being applied over a 5x5 input (green) and the corresponding output (orange). The code used to create the example illustrations was downloaded from the technical report of Dumoulin et al. [27].

Parameter sharing is achieved through passing the same filter or kernel over multiple locations of the input. This reduces the total number of parameters in the network

and allows the CNN to learn interesting features that are invariant to translation. For example, an edge in the top left of a two-dimensional input may be equivalent to an edge in the bottom right of a similar input in that they are both edges and their location does not matter. As the parameters of the filter (F) are shared over the entire input, multiple filters are used, each with their own learnable weights. The notation for expressing the size and number of convolutional filters in a layer is typically denoted as "$h \times w \times n$", where as before, h and w refer to the height and width of the filter, and n is the number of filters used in that layer. The number of channels in each filter is inferred from the number of channels in the input or the output of the previous layer. In other words, the number of filters n used in one convolutional layer is equivalent to the number of channels C in the next layer.

In practice, implementations of convolution in many DL libraries–and in fact, that of Equation 2.6–actually implement *cross-correlation*, the only difference being the filter is not flipped relative to the input. Mathematically, true convolution is equivalent to moving the indices i and j in Equation 2.6 from the filter (F) to the input (I). As the parameters of the filter are learned from the data, the difference in notation is negligible. In order to stay consistent with the bulk of the deep learning literature, the use of the term convolution even when referring to cross-correlation is maintained throughout this document.

The output of the convolution operation is then passed through an activation function (as in Equation 2.2), applied element-wise to each value in O. Many convolutional layers are followed by a *pooling* operation which assists in making the learned model parameters–often referred to as *features* or *feature maps*–less susceptible to small variations. Common pooling operations include *max pooling* in which the maximum value within a specified grid is passed to the next layer, or *average pooling* where the mean value is used. Pooling can also be interpreted as down-sampling the feature-maps. Much like convolution, pooling arithmetic is dependent on the size of the sampling grid and the stride of the operation. Typically a 2×2 grid and stride of one is used, in which case, the dimensions of the feature maps after pooling are reduced by half. In some of the literature, it is assumed that pooling is included in a convolutional layer, however, the two operations are distinguished between in this work for the simple reason that not all layers in the networks described below adhere

to this assumption.

The model *architecture* of a CNN is often visualized as a network diagram displaying the size of the input at each layer, the size of the convolutional filters, the number of filters, pooling operations, and the output of the network (Figure 2.4).

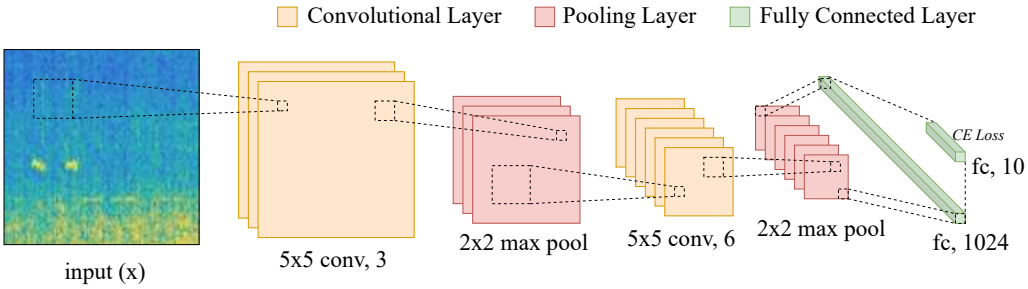


Figure 2.4: Example network diagram of a convolutional neural network (CNN) with an input spectrogram (x), two convolutional layers displaying the size and number of filters of each layer, two 2x2 max pooling layers, and two fully connected layers (fc) displaying the number of nodes in the fully connected layer. The first fully connected layer is simply a flattened version of the output of the final max pooling layer. The final fully connected layer assumes the number of classes is 10 and a softmax activation function.

2.2.1 Deep Residual Learning

Apart from the utilization of GPUs for the purposes of training deep learning models, perhaps one of the most important advances to the field of CNNs was the introduction of residual learning [38]. Intrigued by the results reported by other researchers that using deeper CNN architectures drastically improves performance [46, 107, 113], He et al. posed the question:

> "*Is learning better networks as easy as stacking more layers?*"

The authors demonstrate that this is not the case and that after a certain depth, including additional layers in the network actually increases training error. One would assume that if indeed there was nothing left to be learned from the data, the additional layers would simply learn an identity mapping and thus the network would yield the

same results as its shallower version. However, due to optimization complexities, an identity mapping is not learned and the training accuracy begins to degrade.

To this end, the authors proposed a novel solution to the above issue by learning a function $\mathcal{H}(\mathbf{x})$ through a residual mapping $\mathcal{F}(\mathbf{x})$, such that $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$. In this way, learning an identity mapping is as simple as learning $\mathcal{F}(\mathbf{x}) = \mathbf{0}$. Figure 2.5 contains two examples of what the authors call "residual building blocks": one standard block comprised of two convolutional layers and a second "bottleneck" block containing a stack of three layers.

As depicted in Figure 2.5, the bottleneck version of the residual mapping contains two 1-dimensional convolutional layers that are used to speed up the training process of deeper networks (e.g., 50 layers or more) by first reducing the dimension of the input that is passed to the 3×3 convolution, then subsequently increasing the dimension back to a desired size. An added benefit of the first 1-D convolutional layer is that it increases the non-linearity of the residual mapping via the inclusion of an additional ReLU activation.

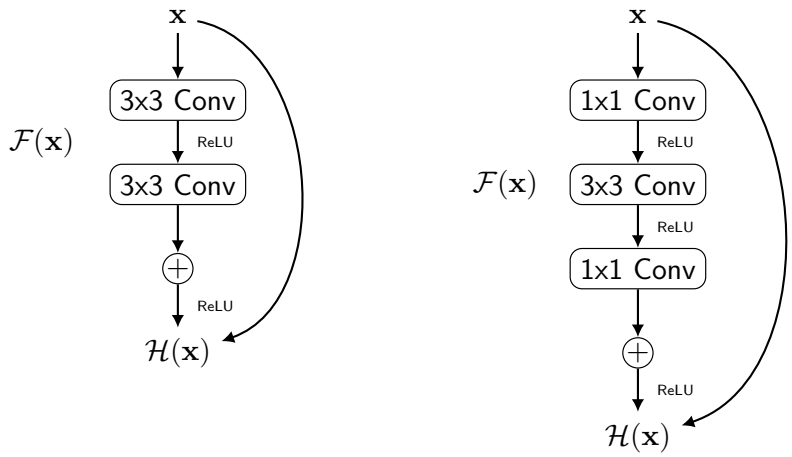


Figure 2.5: Example graphs of a residual building block (left) and bottleneck residual building block (right).

The building blocks described above make up a class of CNN architectures known in the literature as Residual Networks or ResNets [38]. Since their introduction in 2015, Residual building blocks have been adapted and used in many other CNN architectures [41, 43, 112, 115]. In the following Chapters, several commonly used

ResNets with bottleneck residual building blocks are implemented. The numeric value in the name of the model, for example "ResNet-50", indicates that the network has a total of fifty layers.

2.2.2 Additional Advancements

Machine learning–in particular deep learning–is a fast paced area of research. While it is futile to list all of the many advancements, algorithms, and techniques for improving the performance of deep learning based approaches to computer vision, a few techniques that were particularly effective in this work are described below.

SGD with Momentum & Adam

The optimization algorithm used in updating the parameters of a neural network plays an important role in determining the convergence speed and generalization performance of a model. Equation 2.5 outlines how the weights and biases (θ) are updated using gradient descent, where it is assumed that all of the available input data is used for each update step. In other words, the gradient vector $\nabla_\theta \mathcal{L}$ is calculated using every instance in the training set. Full gradient descent is often infeasible in a DL setting due to the size of the training data sets and the computational complexity of the models. As such, Stochastic Gradient Descent (SGD) is a preferred optimization strategy whereby sampled *batches* of data are used to compute $\nabla_\theta \mathcal{L}$. The number of instances used in computing the gradient update vector is referred to as the training "batch size" and the update step of standard (or vanilla) SGD is equivalent to that of Equation 2.5. In this work, two additional optimization routines performed favourably to vanilla SGD. The first optimization algorithm is a slight modification to SGD known as SGD with momentum [101]. SGD with momentum aims to accelerate convergence by maintaining a moving average of previous gradient vectors in order to determine the direction of the next gradient update step in the event that the gradient vectors may be noisy. The gradient update rule of the parameters θ using SGD with momentum can be expressed as:

$$\begin{aligned} \nu_t &= \beta \nu_{t-1} + (1-\beta)\nabla_\theta \mathcal{L} \\ \theta &= \theta - \alpha \nu_t, \end{aligned} \tag{2.8}$$

where ν_t is the "velocity" at step t, β is the momentum coefficient, and as before: $\nabla_\theta \mathcal{L}$ is the gradient vector of the loss function with respect to the model parameters (θ) and α is the learning rate. Common values for SGD with momentum are $\alpha = 0.01$ and $\beta = 0.9$

The second optimization algorithm is Adaptive Moment Estimation (Adam) [55], which aims to achieve the same goal as SGD with momentum (i.e., improving convergence speed and generalization) but does so by maintaining moving average estimates of the first and second moments of the computed gradient vector. The update rule when using Adam can be expressed as:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1)\nabla_\theta \mathcal{L} \\ \nu_t &= \beta_2 \nu_{t-1} + (1 - \beta_2)\nabla_\theta \mathcal{L}^2 \\ \theta &= \theta - \alpha \left(\frac{m_t}{\sqrt{\nu_t} + \epsilon} \right), \end{aligned} \tag{2.9}$$

where m_t and ν_t are the estimates of the first and second moments of the computed gradient vector, respectively, β_1 and β_2 are "decay rates" for the two moments, $\nabla_\theta \mathcal{L}^2$ is the element-wise square of the gradient vector $\nabla_\theta \mathcal{L}$, ϵ is a small constant to avoid division by zero, and α is the learning rate. In practice, m_t and ν_t are bias-adjusted by dividing by $(1 - \beta_1^t)$ and $(1 - \beta_2^t)$, respectively. The mathematics of Equation 2.9 can be interpreted as adapting the learning rate for each parameter in θ according the computed moments. Similarly to SGD, updating θ with Adam is performed using batches of data. Common hyper-parameter values used in Adam are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$.

Depthwise Separable Convolutions

Lastly, when model efficiency is of concern–as discussed in Chapter 5–a variant of the standard convolution operation may be beneficial. One such variant that has proved useful for computationally lightweight applications is known as "depthwise separable convolution" [41, 19, 115]. Depthwise separable convolution consists of separating the convolution operation into two stages: a depthwise convolution and a pointwise (or 1×1) convolution.

In a depthwise convolution, each channel of the input has a unique filter. In comparison to a standard convolution, a depthwise convolution is not accumulated

across the input channels. A depthwise convolution can be expressed as:

$$O_d(i,j,k) = \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} I(i+m, j+n, k) \cdot F_d(m,n,k), \tag{2.10}$$

where, as before, I is the input to the convolutional layer, however, the output O_d and filter F_d are unique on a per-channel level.

A pointwise convolution is then applied to the outputs of the depthwise convolution to achieve a channel-wise sum:

$$O(i,j) = \sum_{k=0}^{C-1} O_d(i,j,k) \cdot F_p(1,1,k) + b. \tag{2.11}$$

Multiple pointwise convolutions are performed using various learnable filters (F_p) and the outputs are stacked along the channel axis.

In contrast to a standard convolution operation, depthwise separable convolutions decouple spatial and channel-wise features. As a result, the computational overhead and number of parameters is significantly reduced. Intuitively, this can be shown with the following example. Suppose the input to the first convolutional layer is an RGB image with dimension $10 \times 10 \times 3$. Applying 32 filters of size 3×3 to the input using a standard convolution would require $32 \times 3 \times 3 \times 3 = 864$ individual weights, and assuming a stride of one and valid padding, would consist of $32 \times 3 \times 3 \times 3 \times 8 \times 8 = 55,296$ multiply–accumulate operations (MACs). In comparison, the depthwise convolution would require $3 \times 3 \times 3 = 27$ weights with $3 \times 3 \times 3 \times 8 \times 8 = 1,728$ MACs, the pointwise operation consists of $1 \times 1 \times 3 \times 32 = 96$ weights with $1 \times 1 \times 3 \times 10 \times 10 \times 32 = 9,600$ MACs, for a combined 123 weights and $11,328$ MACs.

While beneficial in terms of computational efficiency, depthwise separable convolutions may present several drawbacks including limiting the expressiveness of the network in terms of learning complex representations and as a result may lead to underfitting and lower model accuracy.

2.3 CNN-based Detection Models

Convolutional neural networks, as they have been described above, are particularly useful for the task of classification, notably on images. In other words, CNNs are

quite efficient at interpreting the contents of an input x and assigning a confidence score that the contents of the input pertain to some class y. However, a traditional CNN generally performs less favourably when it comes to developing systems that are capable of specifying where in the input the content of interest lies or specifying whether the input to the network contains multiple informative regions of interest of the same or different classes. The most prominent example of the task just described is generally referred to in ML as "object detection".

While it is certainly possible to detect objects to some degree using a standard CNN, for example via a sliding window [104], systems designed specifically with detection in mind are preferred. These systems can be classified into two categories: single-stage and two-stage detection models. As the name suggests, single-stage models aim to directly predict bounding boxes and class labels for targets (e.g., objects in an image) in a single pass over the input. Likewise, two-stage approaches first propose potential regions of interest (RoI) in the input using one neural network and subsequently refine and classify these proposals using a secondary network. The choice of which algorithm to use generally reduces to the intended application of the model. Single-stage detection models are known for their ability to perform inference at or close to real-time making them well-suited for applications where latency is of concern (see Chapter 5), while two-stage detectors have historically outperformed their single-stage counterparts in terms of precision and thus are preferred in scenarios where detection accuracy supersedes run-time performance.

2.3.1 Two-stage Detection Models: Region-based CNNs

The most well-known and commonly used two-stage detection models are region-based convolutional neural networks (R-CNNs). R-CNNs were first introduced in 2014 as an alternative to ensemble systems used for the task of object detection [33]. The original R-CNN model architecture first generated RoIs within an image using a selective search algorithm [119]. These RoIs respresent possible bounding boxes and are then passed to a CNN for classification. In some sense, one can interpret this model as simply a more efficient means of the sliding window approach. While the results of the original architecture were promising, the computational overhead of running each RoI through the CNN was undesirable. The following year, Fast R-CNN

was proposed that simply swapped the order of the selective search algorithm and the CNN [32]. Instead, Fast R-CNN projects the RoIs generated via selective search onto the shared feature maps of the CNN. In comparison, if the selective search algorithms generated 1000 RoIs for a single input, the Fast R-CNN architecture would perform only a single pass through the CNN while the original R-CNN architecture would require 1000 passes.

In subsequent works, Faster R-CNN [95] and Mask R-CNN [36] were proposed which replaced the selective search algorithm all-together with a secondary neural network known as a "region proposal network" (RPN). The RPN used in these architectures carries the benefit of reducing the total number of RoIs necessary for precise detection that is typically required when using selective search. For the purpose of this work, slight differences between the Faster/Mask R-CNN architectures are neglected, notably the use of object masks for the task of instance segmentation [111]. For the remainder of this document the use of the term R-CNN simply implies the model architecture described by Faster R-CNN.

The R-CNN architecture used in this work can be described as follows. First, the model input is passed through a CNN or similar feature extraction network (referred to as the *backbone* network). The feature maps of the backbone network are then temporarily stored as well as passed to the RPN which generates N RoIs (typically N=256). The RoIs of the RPN are then projected onto the stored feature maps and handed to a third network referred to as the *head* network. Projection of the RoIs onto the feature maps is accomplished via either a pooling operation known as RoIPool [95] or using a similar yet more effective operation known as RoIAlign [36]. The head network is comprised of several fully-connected layers and is used to maintain two loss functions: a cross-entropy loss ($\mathcal{L}_{cls} = \mathcal{L}_{CE}$; Equation 2.4) for performing classification on the predicted bounding boxes, and a smoothed L-1 loss function for performing bounding box regression:

$$\mathcal{L}_{reg}(t, v) = \sum_{i \in x,y,w,h} \text{smooth}(t_i - v_i), \tag{2.12}$$

where t represents the predicted bounding box indexed at x, y, w, and h, v is the

corresponding ground truth annotation, and the "smooth" function can be defined as:

$$\text{smooth}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{2.13}$$

The final loss function is a weighted combination of the classification and regression losses $\mathcal{L} = \mathcal{L}_{cls} + \lambda \mathcal{L}_{reg}$. An example network diagram of R-CNN is depicted in Figure 2.6.

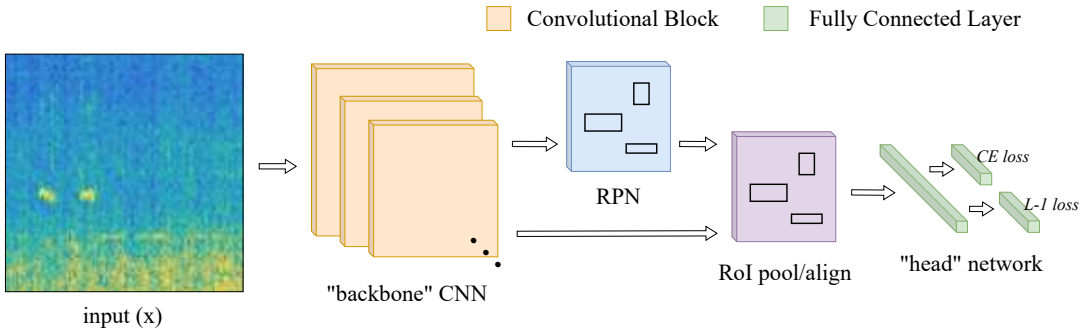


Figure 2.6: Diagram containing the three networks used in the Faster R-CNN model architecture. The first network consists of a CNN "backbone" used in extracting features. These features are sent to a region-proposal network (RPN) in order to determine where in the features there may be regions of interest (RoIs). Finally the CNN features and RoIs are combined using a pooling/alignment operation and passed to fully connected layers in the "head" network in order to optimize for classification (CE loss) and bounding box regression (L-1 loss).

2.3.2 Single-stage Detection Models

As discussed previously, single-stage detectors generate bounding boxes and class predictions of the targets contained in an input using a single neural network. Like the CNNs and indeed the backbone network of an R-CNN, single-stage detectors first start by learning features of interest using convolutional layers. In many cases, following the feature extraction layers of the network, *anchor boxes* are generated at various locations, sizes, and aspect-ratios and superimposed on the learned feature maps. Each generated anchor box represents a potential candidate bounding box for the detection task, the coordinates of which are adjusted during optimization.

Depending on how the task of detection is framed the model architecture of a single-stage detector may vary in its construction. Two variations of single-stage detectors are used in this work: EfficientDet and You Only Look Once (YOLO).

EfficientDet

The EfficientDet architecture [116] is a family of object detection models that combines a similarly named image classification CNN called EfficientNet [115] with a feature pyramid network (FPN; used for learning object scale [69]) and additional convolutional layers to generate class and bounding boxes predictions. Both EfficientNet and EfficientDet are grounded on the concept of "compound-scaling", which suggests that as the input to a model increases (e.g., the size of an image), the receptive-field–and by proxy, model size–should also increase. Traditionally, CNN model scaling is done in a single dimension by increasing either the number of filters (width) or the number of layers (depth) in the network. Compound scaling suggests that both dimensions are dependent on the input of the model. An EfficientDet model is denoted with a suffix indicating the compound scaling coefficient used to re-scale width and depth (e.g., EfficientDet-"D1", uses a scaling coefficient of 1). The term "efficient" in the model name is largely due to its use of depth-wise separable convolutions outlined in Section 2.2.2 that are more efficient than standard convolution operations while maintaining their effectiveness in terms of feature extraction.

In EfficientDet, features from various levels of the EfficientNet backbone are combined via an improved variation of FPN known as Weighted Bi-directional FPN (BiFPN) such that a balance between high-level features (i.e., those learned in the earlier convolutional layers) which have higher spatial resolution but lower semantic information, and low-level features (i.e., those of the latter convolutional layers) that inversely have poor spatial resolution but improved semantic information, can be fused to improve the models ability to learn object scale. These fused features are passed to two distinct groups of convolutional layers tasked with generating class probabilities and bounding box predictions, respectively (Figure 2.7).

EfficientDet replaces the cross-entropy loss function introduced earlier with a focal

loss designed to address the problem of class imbalance [70]:

$$\mathcal{L}_{cls}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \cdot (1 - \hat{y})^{\gamma} \cdot \log(\hat{y}_i), \tag{2.14}$$

where γ is a focusing parameter used to weight hard examples.

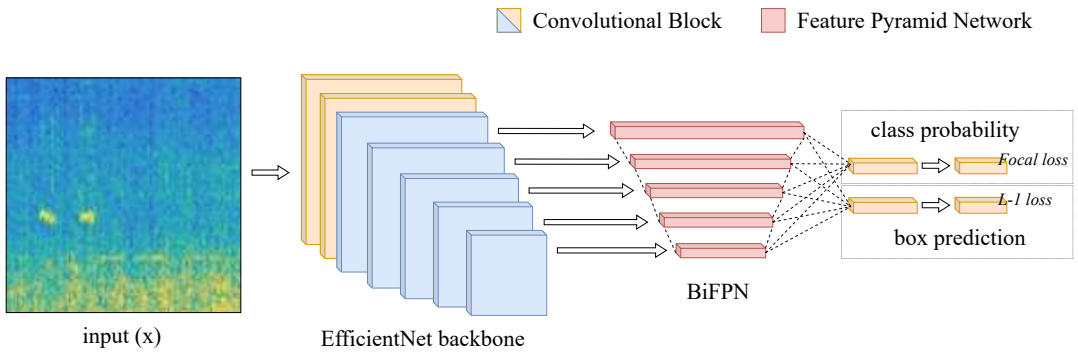


Figure 2.7: Example diagram of the EfficientDet model architecture. The model is composed of an EfficientNet CNN backbone which produces features at various scales. These features are passed to a feature pyramid network (BiPFN), of which each layer in the BiFPN is fully connected to two convolutional layers used in producing bounding box predictions and class probabilities. The convolutional blocks within the backbone network that are passed to the BiFPN are denoted in blue.

You Only Look Once

You Only Look Once (YOLO) was introduced in 2016 [92] and is likely the first example of a single-stage detector. The primary insight of YOLO was framing object detection as purely a regression problem. Detection is achieved by applying an $S \times S$ (default 7×7) grid cell to the input and tasks grid cells which contain the center of various targets/objects with producing bounding box predictions, a box confidence score, and conditional class probabilities. The resulting model was able to achieve performance close to several state-of-the-art (SOTA) two-stage detectors, but with significantly faster inference run-times.

Following the success of the first version of YOLO, a number of improved versions have been proposed by both the original authors and other unaffiliated researchers and engineers. Several of the improvements to the YOLO model over the years include

using batch-normalization in the network and k-means clustering to better initialize anchor boxes (YOLOv2 [93]); an improved backbone network with multi-scale predictions (YOLOv3 [94]); as well as self-attention and improved data augmentation strategies (YOLOv4 [13]).

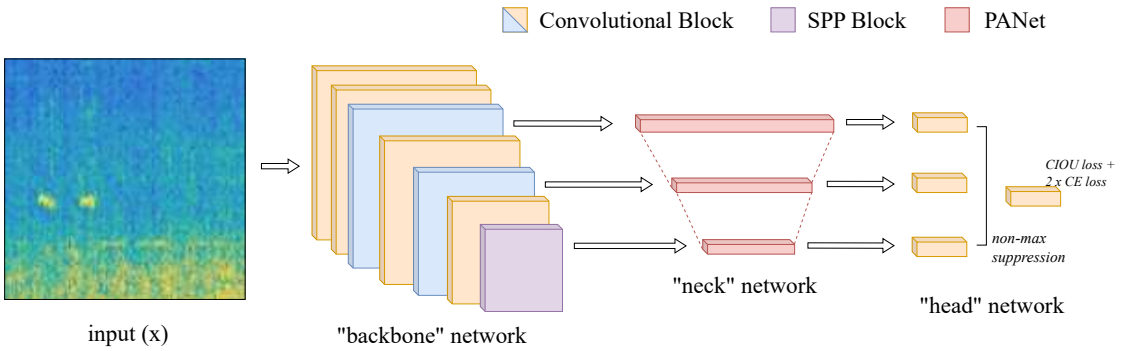


Figure 2.8: Example diagram of the YOLOv5 model architecture. The model is composed of a CNN backbone with convolutional blocks and a SPP block. Three scales of learned features are passed as input to a PANet referred to as the "neck". Finally, three additional convolutional blocks in the head network are combined via non-max suppression. The convolutional/SPP blocks within the backbone network that are passed to the PANet are denoted in blue and purple.

After version four, discrete updates to the YOLO model are less traceable and have largely become iterative open-source repositories that frequently update the model architecture based on improvements found in the deep learning and computer vision literature. In this work, one such repository (*YOLOv5* [51]) is used that also implements a variation of compound scaling to create various sized networks. The YOLOv5 family of models with compound scaling are also named using a suffix indicating the size of the model: YOLO-n (nano), -s (small), -m (medium), -l (large), and -x (extra-large). YOLOv5 uses a combination of three loss values: two binary cross-entropy losses on the box confidence score and conditional class probabilties, respectively, and a Complete Intersection-over-Union (CIOU) loss [131]. YOLOv5 also implements Spatial Pyramid Pooling (SPP) [37] in the final block of the backbone network. Similar to the BiFPN used in EfficientDet, YOLOv5 computes features at several scales and passes these features to a secondary network known as a Path Aggregation Network (PANet) [71] used in combining the multi-scale features efficiently.

A simplified network diagram of YOLOv5 is presented in Figure 2.8.

2.4 Performance Metrics for CNNs and R-CNNs

In this section several statistical metrics commonly used to measure the performance of detection and classification systems are introduced. However, for the time being one should consider these metrics only from the point of view of a researcher coming from machine learning and not specifically PAM. Additional comments on these metrics from a PAM perspective are brought forward in Chapter 5 when discussing operational DCS.

2.4.1 Precision, Recall, and F-1 Score

The most commonly computed metrics for evaluating the performance of a classifier are precision (P) and recall (R). Precision and recall are calculated using the total number of true positive (TP), false positive (FP), and false negative (FN) predictions. In the case of a CNN trained for the task of binary classification (i.e., the class labels are either 0 or 1), a true positive is equivalent to the scenario where both the class label and predicted class of the CNN are equal to 1. Adversely, a false positive is equivalent to the case where the predicted class is 1, however, the true class label is 0. Finally, a false negative is equivalent to the case where the predicted class is 0 and the true class label is 1.

Precision and recall are computed using Equations 2.15 and 2.16, respectively.

$$\mathrm{P} = \frac{TP}{TP + FP} \tag{2.15}$$

$$\mathrm{R} = \frac{TP}{TP + FN} \tag{2.16}$$

Precision defines the proportion of TP predictions among all positive predictions (i.e., it measures the accuracy of positive predictions), while recall defines the proportion of TP predictions among all actual positive classes (i.e., measuring the ability of the model to identify positive classes).

For the case of binary classification, precision and recall is straightforward. In the case of multi-class classification, the metrics are computed using either a *micro* or

macro average. For the purposes of this work, precision and recall are expressed using as a macro average, whereby, the final precision and recall metrics are computed as the mean of the individual precision and recall for each possible class. Consider a classification problem with N possible classes, computing a macro average of precision can be expressed as:

$$P_{macro} = \frac{1}{N} \sum_{i=1}^{N} P_i, \tag{2.17}$$

where P_i is the precision of class i. Macro-averaged recall is computed analogously. Using a macro average, the performance of the classification model is treated equally across all possible classes, and therefore better represents overall performance of class-unbalanced data sets.

At a certain point, the precision and recall metrics described above become inversely proportional, i.e., as precision increases, recall tends to decrease. As such, a balance between high precision and high recall is desired. A third metric known as F-score (F_β) is often used in order to express such a balance and is computed as the harmonic mean of precision and recall:

$$\mathrm{F}_\beta = (1 + \beta^2) \cdot \frac{\mathrm{P} \cdot \mathrm{R}}{(\beta \cdot \mathrm{P}) + \mathrm{R}}, \tag{2.18}$$

where β is a positive real value that weights the importance of recall and precision. Three commonly used values for β include 1 (balanced importance of precision and recall), 2 (increased importance of recall), and 0.5 (increased importance of precision).

Notably, in this work, model "accuracy" is never used, as such a metric is misleading for class-imbalanced data sets. Later sections of this work will demonstrate the unbalanced nature of PAM annotations due to differences in species population and vocalization rates.

2.4.2 Using Average Precision to Describe Detection Performance

The performance metrics outlined in Section 2.4.1 are appropriate when the entire input to a model can be considered as belonging to a single class. When individual regions of a model's input are described differently, as in the case of object detection, these metrics must be elaborated upon. In this work, the performance of a detection

model (single-stage or two-stage) is measured using Average Precision (AP) and computed following the formulas described by the COCO Detection Challenge benchmark data set.[1]

AP is dependent on an additional threshold function known as Intersection over Union (IoU). IoU measures the similarity between the ground truth bounding box and a predicted bounding box and is computed by dividing the area of the boxes' intersection by the area of their union. A visual depiction of the intersection and union of two bounding boxes can be seen in Figure 2.9.

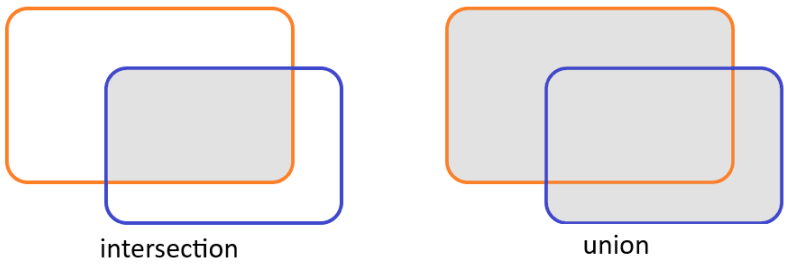


Figure 2.9: Example depiction of the intersection (left) and union (right) between a ground truth bounding box (blue) and predicted bounding box (orange). The intersection-over-union (IoU) of these two bounding boxes is computed as the division of these two areas, respectively.

Somewhat un-intuitively, computing AP is not as simple as taking the "average of precision", and is instead computed by taking the area under the precision-recall curve.

Computing AP can be outlined as follows:

- Sort the models' predictions in descending order based on confidence scores.
- For each prediction, determine the values of FP and TP using a threshold on IoU. If the IoU between a ground truth bounding box and a predicted bounding box exceeds this threshold, the prediction is considered a TP, otherwise a FP.
- Compute P and R for the current prediction (where the denominator used in computing in R is simply the total number of ground truth boxes). Upon iterating over all the predictions, the result is a precision-recall curve ordered by the confidence scores of the model.

[1]Details of the COCO Detection Challenge: `https://cocodataset.org/#detection-eval`

- Compute the area under the P/R-curve.

Typically, the IoU thresholds used in computing AP range from 0.5 to 0.95 by increments of 0.05 (sometimes denoted AP@0.5:0.95). By varying the IoU threshold, we can evaluate the performance of a detection model at different levels of difficulty (e.g., different target sizes and aspect ratios). AP may also be computed at specific IoU thresholds, for example AP@0.5.

For a multi-class detection problem AP is computed for each class and a second mean over all classes is taken. In some cases, this may be denoted as mean-Average Precision (mAP); however, the "m" is often inferred from the data set. While quite convoluted, the metrics defined above offer a very informative description of detection performance in scenarios where the number, size, and scale of ground truth annotations in a single input may vary significantly.

Finally, in addition to (m)AP, in later sections of this work, the precision and recall that produce the highest F-1 score on the P/R-curve used in computing AP are also reported. These P and R metrics are sometimes referred to as "box precision" and "box recall".

2.5 Visual Representations of Acoustic Data

The models introduced in Sections 2.2 and 2.3 are primarily used in the visual domain on tasks such as image classification and object detection. In this section a commonly used visual representation of acoustic data often found in the DCS literature is described, namely, the spectrogram. However, this topic often begs the question:

"*Why use a visual representation of an auditory signal?*".

The Fourier transform necessary to produce a spectrogram will inevitably lead to some loss of information (i.e., that pertaining to phase), and as such, provide the automated DCS with an imperfect representation of the ground truth data. However, the benefits that come with a visual representation typically out-weigh the trade-offs created from this imperfection.

For one, more reliable results have been observed when using models that operate in a visual domain versus models originally developed to operate on vectors of

sequential data such as recurrent neural networks (RNNs) [12]. Anecdotally, these findings seem to suggest that spectrograms simply provide DL-based systems more signal and less noise.

Finally, and perhaps the most obvious response to this question, is simply that spectrograms correspond to the domain which experts–namely marine biologists–use during acoustic analysis [59]. It is quite likely that the ground truth annotations were created using spectrograms in software like JASCO Applied Sciences PAMLab[2], Cornell University's Raven[3], or Audacity[4]. Additionally, the species of marine mammals that is of concern in this work vocalize across relatively consistent frequency bands and therefore, visual representations such as spectrograms allow for DCS predictions to have temporal and frequency bounds. As a result, the predictions of an automated DCS trained on spectrograms can be easily and quickly interpreted by experts and more informative in down-stream tasks.

There are several other data representations used in the development of PAM DCS beyond the spectral representations described below, some of which are are briefly discussed in the related work (Section 2.6).

2.5.1 Linearly-scaled Spectrograms

As outlined above, human analysis of acoustic recordings is often performed aurally by listening to the recording as well as visually using spectrograms.

A popular approach for generating spectrograms is through a Short-time Fourier Transform (STFT). The STFT procedure calculates the sinusoidal frequency and phase content of an acoustic signal over time and is most commonly visualized in two dimensions with time on the x-axis, frequency on the y-axis, and intensity expressed by varying colour.

The equation of the discrete-time STFT of a signal $x[n]$ can be expressed as:

$$X(n, \omega) = \sum_{m=-\infty}^{\infty} x[m]w[m-n]e^{-i\omega m} \ , \tag{2.19}$$

where w is a windowing function with a pre-specified length centred at time n. In

[2]JASCO Applied Sciences: `https://www.jasco.com`
[3]Raven Sound Analysis: `https://ravensoundsoftware.com`
[4]Audacity: `https://www.audacityteam.org`

the equation expressed above, time (n) is discrete and frequency (ω) is continuous, however, in practice both units are discretized and each successive STFT is computed using an implementation of the Fast Fourier Transform (FFT) algorithm (e.g., the Cooley-Tukey algorithm [22]). Equation 2.19 describes a complex function, therefore, we take the square of the absolute value of $X(n, \omega)$ yielding a spectrogram of the power spectral density. Finally, we convert the intensity to a logarithmic scale (i.e., decibels (dB)), as is commonly the case in underwater acoustics.

2.5.2 Mel-scaled Spectrograms

A spectrogram computed using the approach formulated above is linear in frequency and therefore does not contain information related to how human beings perceive sound. For example, while the difference between two signals occurring at 1000Hz and 1500Hz and two other signals occurring at 10kHz and 10.5kHz are numerically equivalent (i.e., their difference is equal to 500Hz), the difference of the lower frequency signals is perceptually much larger to a human listener. In some scenarios, including low frequency applications, such an imperception is negligible, however at higher frequencies a logarithmic scale may be preferred. The most common logarithmic scale used in the literature is the mel-spectrogram, whereby frequency is transformed from hertz to mels (from the word melody) using the following equation:

$$\omega_{mel} = 2595 \log_{10} \left(1 + \frac{\omega_{Hz}}{700}\right) \ . \tag{2.20}$$

Using this transformation, the resulting frequency scale more closely aligns with the log-like human-auditory perception known as pitch. The audio processing Python library *librosa*[5] used throughout this work converts a spectrogram to mels by further discretizing the frequencies into n bins, computing the mel-scale of each bin and transforming the spectrogram from hertz to mels using a triangular filter. In effect, the mel-spectrograms have effectively been compressed in the frequency axis to the value n.

[5]Librosa

2.5.3 Novel Representation: Stacked & Interpolated Spectrograms

During the creation process of a linear or mel-scale spectrogram, a decision must be made on the appropriate combination of parameters to pass to the FFT. In practice, when marine biologists analyze acoustic recordings, they will often generate multiple spectrograms using different FFT parameters, for example: changing the length of the FFT window and/or the window overlap. By changing the parameters of the FFT, the time and frequency resolutions of the spectrogram are altered. Using multiple spectrograms with varying resolutions is particularly helpful when annotating underwater acoustic recordings containing marine mammal vocalizations because some species tend to make prolonged low-frequency vocalizations with a small bandwidth (e.g.: blue whale moans), while other species make shorter vocalizations with a larger bandwidth (e.g.: humpback songs). Depending on the set of parameters used to generate the spectrogram, one can easily mis-classify a vocalization as a different species or miss the vocalization entirely.

A novel representation of an acoustic signal is proposed that attempts to exploit the strategy used by human experts during the annotation process. First, following Equation 2.19, several spectrograms are generated using different FFT parameters. Because each of the spectrograms vary in resolution across time and frequency, they are interpolated using a simple linear interpolation spline over a grid proportionate to the smallest time and frequency resolution. The equation of a linear interpolation spline for some point (n, ω) between (n_i, ω_i) and (n_{i+1}, ω_{i+1}), where n is known, can be expressed as:

$$\omega = \omega_i + \frac{\omega_{i+1} - \omega_i}{n_{i+1} - n_i}(n - n_i) \ . \tag{2.21}$$

After interpolation, the dimensions of the matrices corresponding to each spectrogram are the same. The interpolated spectrograms are then stacked to form a multi-channel tensor; imitating the concept of RGB channels in a digital colour image. The details of the algorithm used to produce a single instance of the novel representation described above are outlined in Algorithm 1.

Algorithm 1: Generating an instance of the novel representation

Input: The waveform x, function w, and parameters $\mathbf{\Theta} = [\theta_1, \theta_2, \ldots, \theta_k]$

Output: A tensor $\mathbf{Z}$ with k channels

Initialize the interpolation resolutions ω_0 and n_0 to ∞
for $i = 1$ *to* k **do**
    Generate a spectrogram $\mathbf{D}_i = \text{STFT}(x; w, \theta_i)$ (Eqn 2.19)
    Maintain a running minimum of ω_0 and n_0
    if $\Delta\omega_i < \omega_0$ **then**
        $\omega_0 = \Delta\omega_i$
    end
    if $\Delta n_i < n_0$ **then**
        $n_0 = \Delta n_i$
    end
end
for $i = 1$ *to* k **do**
    Interpolate each spectrogram $\mathbf{S}_i = \text{INTERPOLATE}(\mathbf{D}_i; \omega_0, n_0)$ (Eqn 2.21)
end
Stack the interpolated spectrograms $\mathbf{Z} = [\mathbf{S}_1, \mathbf{S}_2, \ldots, \mathbf{S}_k]$
Return $\mathbf{Z}$

2.6 Applications of Machine Learning to PAM

In developing an automated DCS using machine learning, one hopes to accurately detect and assign a label to an instance of an audio recording containing a species of interest. However, developing a generalizable DCS presents several distinct challenges. For one, underwater recordings often have a low signal-to-noise ratio (SNR) making feature extraction difficult. Another challenge is that ground truth labelled data is difficult to obtain due to the required expertise and training of the labeller. As a result, only a very small fraction of the large collections of acoustic data is suitable for supervised learning [29]. Furthermore, the small numbers of some species coupled with the low rate of occurrence of their vocalizations make for highly unbalanced data sets. Many of these challenges, especially as they relate to real-world DCS use, are elaborated upon in Chapter 5.

Traditionally, many of the algorithms used to detect and classify marine mammal vocalizations are derived from the properties of a signal of interest. In general, these approaches can be divided into two categories. The first category involves comparing unlabelled data to templates of certain vocalizations. Examples of this approach include matched filtering, where a template corresponding to the vocalization of interest is convolved with the signal to produce a detection function. The detection function is evaluated using a pre-determined threshold parameter to determine whether a detection has been made [21]. Another example is *spectrogram correlation*–not to be confused with spectrogram *cross-correlation*, which is equivalent to matched filtering [79]. Spectrogram correlation first computes a correlation kernel using segments of template spectrograms. The correlation kernel is then convolved over the spectrogram of the unlabelled data producing a vector representing the similarity between the spectrogram and the kernel over time. Large similarity values correspond to possible detections. These methods largely do not involve machine learning.

The second category of algorithms is made up of three stages: detection, feature extraction, and classification. An example being detecting regions of interest in a spectrogram, following which, features (e.g.: the duration of the detection or the absolute change in frequency) are extracted and used as input vectors for classification. Various detection algorithms are used in the first step of this approach including: neighbourhood search algorithms (e.g., pixel connectivity) in spectrograms that have

been filtered, smoothed, and cast to binary representations [5] and contour detectors that operate by continually searching for local maxima within pre-specified frequency bands of normalized spectrograms over time [81]. These detection algorithms are heavily dependent on the filtering, normalization, and smoothing operations that are performed on each spectrogram. Once the regions of interest are determined, feature vectors are then handed to commonly used classification algorithms such as: linear or quadratic discriminant analysis (LDA/QDA) [5, 31], support vector machines (SVM) [26], shallow artificial neural networks [4, 26], Gaussian mixture models (GMMs), and hidden Markov models (HMMs) [99, 100, 108]. Outside of using spectrograms, other data representations including continuous and discrete wavelet transformations and mel-scale frequency cepstral coefficients (MFCCs) have been reported to varying degrees of success [120].

The algorithms described above involve a significant amount of human input–often from experts–which is a limitation to the development of future classifiers for several reasons. In the former category the templates used for detection and classification are largely specific to not only certain species, but also different types of vocalizations produced by the same species. At the same time, the detection threshold may require fine-tuning depending on the noise characteristics of the data set. For the latter category of algorithms, many of the hyperparameters provided to the smoothing and noise-removal routines are dependent on the data set. Subsequently, the hand-engineered features are contaminated by these specifications as well as human bias. These limitations yield systems which are not as easily generalizable to a broad category of species using data collected at different sampling rates, geographic locations, or using different recording devices.

2.7 Applications of Convolutional Neural Networks to PAM

More recently, researchers have started to use DL-based alternatives in place of traditional machine learning algorithms for the purpose of DCS development. In part, the application of deep learning to PAM has been brought on by its success in other fields, namely image classification [38, 61]. The attraction of deep learning also stems from the ability to train neural networks that can more easily generalize to new tasks [57, 115]. But more importantly, deep learning algorithms typically rely on zero

feature engineering and learn *latent representations* directly from the training data. Much like the traditional machine learning approaches mentioned above, applications of deep learning typically operate in the frequency domain (i.e., using spectrograms). Most notably, DL-based DCS development has been likened to image classification through the use of CNNs.

Much of DL-based PAM research reported in the literature is focused on higher frequency vocalizations (e.g., whistles, clicks, and pulsed calls) made predominantly by toothed whales (*odontocetes*). Bergler at el. [9] train several ResNet models to classify spectrograms containing various orca (Orcinus orca) vocalizations against background noise examples and in a separate work report improved performance by pre-training the ResNet classifier using an auto-encoder [8]. Jiang et al. [50] train a CNN to classify spectrograms containing the vocalizations of orcas and long-finned pilot whales (*Globicephala melas*). Liu at al. [72] apply CNNs to classify spectrograms containing various vocalization types as opposed to the species that produced them, and Luo et al. [74] train a CNN to detect the vocalizations of odontocetes using a combination of real audio recordings and synthetic data. Zhong et al. [132] employ an ensemble model of CNNs to classify Beluga detections as true or false positives. Finally, Best et al. [11] acknowledge that many of the reported findings in the literature may be susceptible to data drift (a topic that will be touched upon in this work later on), and pose better separations to the data sets used in training/testing the orca classification models previously mentioned.

Slightly less research has been reported applying CNNs to the lower frequency ($<$ 1000Hz) vocalizations of baleen whales where acoustic masking due to vessel traffic can be especially challenging. One species of baleen whales that is of particular concern for conservation purposes is the North Atlantic right whale (*Eubalaena glacialis*). As such, several papers have been published which attempt to classify spectrograms containing their vocalizations. The most distinguishing right whale vocalization is known as the upcall. Kirsebom et al. [56] as well as Shiu et al. [105] both develop spectrogram classification models using ResNets to identify upcalls. In subsequent work, Ibrahim at el. [44] develop a multi-model approach to determining right whale presence/absence by passing a spectrogram through a CNN, a scaleogram through a stacked auto-encoder and fusing the outputs of both systems to create a

final predictive result. Vickers et al. [123] propose the use of denoising CNNs to improve model accuracy using both upcalls and a second right whale vocalization known as a gunshot. In additional work, the same authors [122] suggest methods to improve model robustness in a similar fashion to that reported above by Best et al.

Beyond right whales, additional research has been reported that uses CNNs to classify spectrograms possibly containing the vocalizations of blue whales (*Balaenoptera musculus*) [133] and humpback whales (*Megaptera novaeangliae*) [2, 83]. Lastly, the work of Madhusudhana et al. [75] presents a significant advancement through its incorporation of an RNN that enables a CNN to incorporate temporal context in its predictions of fin whale (*Balaenoptera physalus*) vocalizations and subsequently improve performance. Moreover, the recurrent nature of the model allows the output of the model to closer align with the notion of "vocalization detection" (as opposed to "spectrogram classification") used in this thesis.

Chapter 3

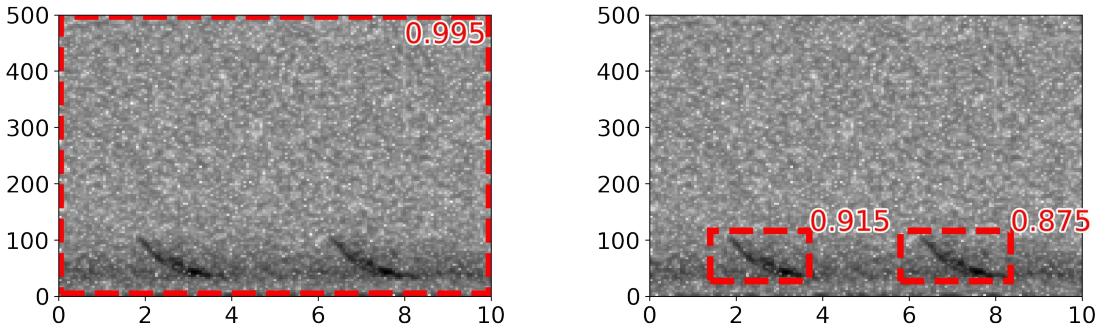
Deep Convolutional Learning for PAM

So far the concepts of classification and detection have often been used interchangeably, however, a distinction between these two tasks should be made and elaborated upon. For the case of a DL-based DCS, particularly one that employs a CNN, it is argued that true detection of marine mammal vocalizations should contain precise frequency components in the predicted output (e.g., bounding boxes). From this, much of the related work cited in Section 2.7 often confuses detection with what is referred to in this work as spectrogram classification. This distinction is necessary when one is concerned with estimating the total number of individual vocalizations within a specified time frame or when detecting multiple species whose vocalizations are visible in the same spectrogram.

Consider Figure 3.1 that depicts a spectrogram containing two consecutive sei whale vocalizations known as "down sweeps". On the left (a) is the scenario of the predicted output of a system trained to do spectrogram classification. The output of such a system is simply a confidence score (i.e.: probability) that the entire spectrogram contains at least one sei whale vocalization. On the right (b) is the same spectrogram as well as the predicted output of a similar system trained to detect individual vocalizations. As we can see, individual detections are presented as bounding boxes with an attached score representing the probability that said bounding box contains a single sei whale vocalization. Clearly the system designed to produce predictions at the spectrogram level is incapable of distinguishing between individual vocalizations and are therefore unreliable for estimating species abundance. Similarly, the classification system is unable to confidently determine whether a spectrogram contains multiple vocalizations from different species.

In some cases, spectrogram classification may be sufficient to detect individual vocalizations, at least temporally [56]. When the length of the vocalization of interest is highly consistent (e.g., always three seconds in length), one can simply ensure the

length of the spectrogram that is being classified matches that of the vocalization of interest. Detection is accomplished by sliding the classifier over a spectrogram containing the entire recording, provided the classifier was trained on ambient or false positive examples. Spectrogram classification is less adaptable when one is attempting to detect call types of varying lengths (e.g. songs produced by humpback whales) or when developing automated DCS for multiple call types and/or multiple species. In the latter two scenarios, it is very unlikely that the collective call types of interest share the same average duration.



(a) *spectrogram classification* (b) *vocalization detection*

Figure 3.1: An example spectrogram containing two consecutive sei whale vocalizations, the first from times 2-4 seconds and the second from roughly 6-8 seconds. The spectrogram is duplicated in order to demonstrate two different systems: (a) the predicted output of system trained for spectrogram classification, and (b) the predicted output of a system trained for vocalization detection.

For the remainder of this work, it is maintained that the use of the phrase "an automated DCS of marine mammal vocalizations" adheres to the following definition, as demonstrated in Figure 3.1b.:

Definition 1 (Detection and Classification System (DCS))**.** *An automated system that can accomplish both detection (e.g., in the form of bounding boxes or contour masks) and classification of individual calls represented as a classification score or probability.*

The use of the term "classification systems" or "spectrogram classification" is preferred when referring to simpler systems that only classify the contents of a spectrogram to a particular species or call type, as in Figure 3.1a.

Based on the research presented below, the system described in Section 3.3 was, at the time of publication, novel to PAM in producing detections of individual vocalizations in the frequency domain via end-to-end deep learning.

3.1 Acoustic Data Sets

The remainder of this Chapter relies on an acoustic data set collected by JASCO Applied Sciences under a contribution agreement with the Environmental Studies Research Fund (ESRF) and his here-by referred to simply as the ESRF data set. This data set was collected in two deployments: the first starting in August 2015 and ending in July 2017. Twenty Autonomous Multi-channel Acoustic Recorders (AMARs) were deployed off the coast of Atlantic Canada along an area of particular biological interest known as the Scotian Shelf (Figure 3.2) at depths ranging from 44 meters to 2002 meters. In 2015-2016, the AMARs were fitted with HTI-99 omnidirectional hydrophones from HTI Inc. (-165 ± 3dB re 1V/μPa sensitivity) and in 2016-2017, with M36-V35dB omnidirectional hydrophones from GeoSpectrum Technologies Inc. (-165 ± 3dB re 1V/μPa sensitivity). The AMARs operated on a 20 minute duty cycle. The AMARs recorded for 11 minutes 18 seconds at 8 kilo-samples per second (ksps) bounded between 10Hz and 4kHz, followed by 64 seconds at a much higher 250ksps. For the remainder of this Chapter, as this work is only concerned with low-frequency baleen whale vocalizations, the data is restricted to the 8ksps data. Further details of the locations and depths of each recorder are provided in Table 3.1

3.2 CNNs for Spectrogram Classification

The contents of this Section are largely taken from the conference proceedings:
Thomas, Mark, et al. *"Marine mammal species classification using convolutional neural networks and a novel acoustic representation."* Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Cham, 2019.

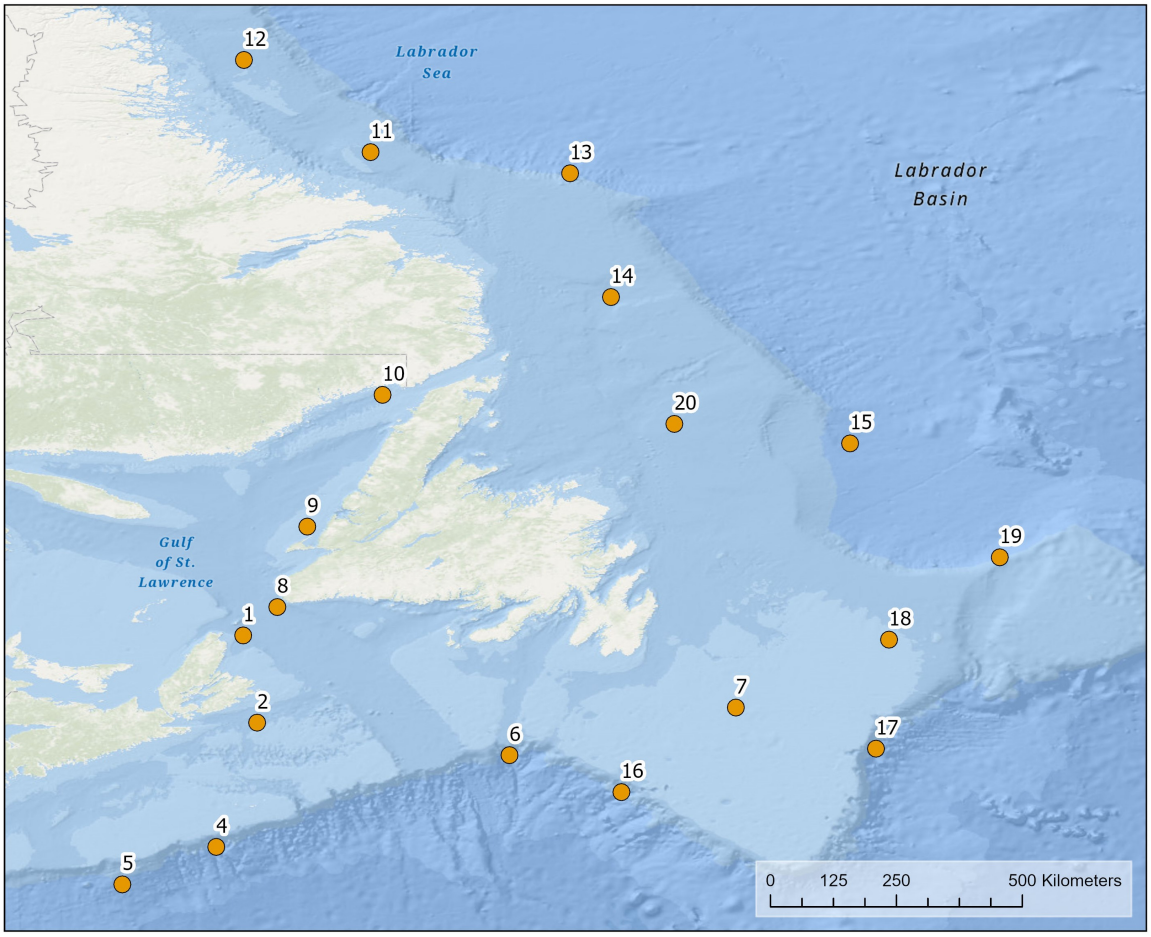


Figure 3.2: Individual deployment locations for each of the 19 AMARs used in collecting the ESRF data set. The recording devices were deployed over 12 months along the Scotian Shelf off the coast of Atlantic Canada. Note that a recorder was deployed at station 3 missing from this figure near Sable Island, however it was lost either due to burial in moving sand or fishing gear.

3.2.1 Data set and Methods

Over the past decade, CNNs have continued to improve upon the state-of-the-art for many computer vision tasks [38, 61]. Recently, a growing collection of research has been brought forward applying CNNs to tasks which are auditory in nature, including: speech recognition [1, 24, 67], musical information retrieval [17, 18, 42], and acoustic scene classification [89, 90, 102]. Inspired by the compelling results obtained in the previously mentioned domains, researchers in oceanography and marine biology have started to investigate similar solutions for PAM.

Table 3.1: Deployment locations and recording depths of the AMARs used in collecting the ESRF data. Note that a recorder was deployed at station 3 near Sable Island, however it was lost either due to burial in moving sand or fishing gear.

Set A: Bay of Fundy (model training/validation)			
station	longitude	latitude	depth (m)
1	46°59′28.8240″ N	60°1′26.5080″ W	186
2	45°25′33.5634″ N	59°45′50.3274″ W	125
4	43°13′1.2714″ N	60°29′57.9474″ W	1830
5	42°32′51.3600″ N	62°10′34.4634″ W	2002
6	44°51′11.1240″ N	55°16′15.8874″ W	1802
7	45°42′2.9520″ N	51°13′59.3400″ W	78
8	47°29′35.0520″ N	59°24′47.6994″ W	428
9	48°55′38.3874″ N	58°52′40.2954″ W	44
10	51°16′8.8320″ N	57°32′15.3240″ W	121
11	55°36′10.8000″ N	57°45′1.4394″ W	158
12	57°15′9.8274″ N	60°0′6.3000″ W	143
13	55°13′40.6914″ N	54°11′25.6914″ W	1750
14	53°0′56.4120″ N	53°27′36.7914″ W	582
15	50°24′47.7714″ N	49°11′46.9674″ W	2000
16	44°11′32.2800″ N	53°16′27.8760″ W	1602
17	44°58′17.0754″ N	48°44′1.4280″ W	1282
18	46°54′31.5714″ N	48°30′15.0474″ W	111
19	48°43′43.4274″ N	49°22′51.1320″ W	1282
20	50°45′8.3514″ N	52°20′9.6714″ W	237

In this section, a classification system is presented that is capable of classifying spectrograms containing the vocalizations of three species of endangered baleen whales: blue whales (BW; *Balaenoptera musculus*), fin whales (FW; *Balaenoptera physalus*), and sei whales (SW; *Balaenoptera borealis*). The vocalizations of these species can be particularly challenging to distinguish as all three species are capable of making a similar vocalization known as a down sweep during the summer months and often inhabit the same region during overlapping time frames.

The ESRF acoustic recordings introduced in Section 3.1 were analyzed by marine biology experts at JASCO Applied Sciences producing over 30,000 annotations in the

form of bounding boxes around signals pertaining to the three species of whales and other acoustic sources labelled as "non-biological". Other species of whales present in the recording area were also annotated, however, they are not included in this work. The distribution of annotations is heavily unbalanced in favour of the more vocal fin whales at roughly a 6:1 ratio, as depicted in Table 3.2.

Table 3.2: Number of files and the distribution of each acoustic source for the training, validation, and test sets.

Acoustic source	label	training		validation		test	
Blue Whale	BW	2692	(6.23%)	601	(6.49%)	574	(6.20%)
Fin Whale	FW	15,118	(35.01%)	3244	(35.06%)	3272	(35.36%)
Sei Whale	SW	1701	(3.94%)	332	(3.59%)	383	(4.14%)
Non-biological	NN	2078	(4.81%)	449	(4.85%)	398	(4.30%)
Ambient noise	AB	21,589	(50.00%)	4626	(50.00%)	4627	(50.00%)

The data used for training, validating, and testing the classifiers were created in the following fashion. First, 30 second long excerpts centered on human annotations were extracted from the full set of acoustic files. Four spectrograms depicting typical examples of the 30 second excerpts are provided in Figure 3.3; one for each of the possible acoustic sources. Examples of the start/end times of the annotated bounding boxes are drawn using dashed vertical lines. As we can see, not every vocalization that appeared in a spectrogram was labelled. In Figure 3.3a for example, there appears to be three blue whale vocalizations occurring consecutively, however, only the second has been annotated. Partial annotations are an unfortunate by-product of the way in which marine biologists annotate PAM data if, for a certain data set, stakeholders are more concerned with species presence/absence versus abundance. Ways of mitigating some of the issues that may arise when using partial annotations are discussed later on in Chapter 4.

For each 30 second excerpt, a smaller ten second long sample (hereby referred to as simply a "sample") containing the entire annotation is randomly selected. Due to the partial labelling of the recordings, it is possible that a sample may include more than one vocalization. For example, a sample from time 10 to 20 seconds in the file used to produce Figure 3.3c would in fact contain three sei whale vocalizations. This

means that it is also possible that a sample contains an unlabelled vocalization of another species. The data containing only ambient noise were produced in a similar fashion, however, they were produced from a large set of files that were known to not contain baleen whale vocalizations. As such, the sampling routine simply selected a ten second sample randomly from the entire file.

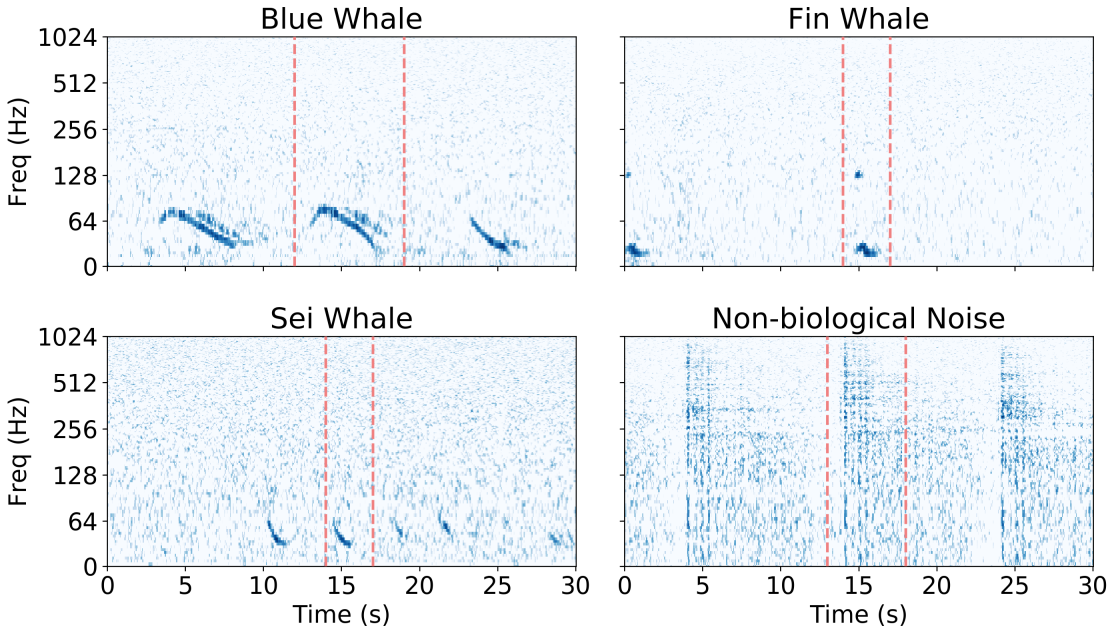


Figure 3.3: Example spectrograms for each of the three whale species: a) blue whales, b) fin whales, c) sei whales, and d) non-biological noise. Dashed vertical lines depict the start and end times of the expert annotations centered within the 30s window used in the data creation sampling routine. For visualization purposes filtered versions of the spectrograms are being shown on a log-frequency scale so that the reader can more easily identify the vocalizations.

The samples were used to produced three spectrograms using varying FFT window lengths (256, 2048, and 16,384 samples) in order to train three different CNNs. Each FFT used an overlap of 25 percent and a Hann windowing function. The spectrograms were truncated to have an upper frequency bound of 1000Hz. Additionally, a mel-scaled copy of the spectrogram obtained using a FFT length equal to 2048 was created with 128 mels and used to train a fourth CNN. Finally, a combination of the three original spectrograms was used to form a three-channel version of the novel representation outlined in Section 2.5.3. Each spectrogram was scaled to decibels

(dBs) before being normalized between [0, 1] for optimization purposes. No additional filtering, smoothing, or noise removal was applied to the spectrograms. An individual 10 second, decibel-scaled, and normalized spectrogram is analogous to a single "instance" used for training, validation, and testing.

In practice, the ten second sampling routine and all subsequent steps including spectrogram generation, re-scaling, etc., were executed in parallel on the CPU while the CNNs were trained on the GPU. In this way, the sampling routine acted as a quasi-data-augmentation strategy for each training batch.

Separate training, validation, and test data sets were produced using a random split ratio of 70/15/15, respectively. Model performance on the validation set was measured after each epoch, while the model performance on the test set was only measured after the training had completed. The random sample is a notable flaw and discussed in detail later on in Chapter 5 when developing systems to be used in the real-world.

Multiple ResNet-50 CNNs were trained to classify the spectrograms described above. Each CNN was implemented in Python using the PyTorch open source deep learning platform [88]. Training was distributed over four NVIDIA P100 Pascal GPUs each equipped with 16GB of memory. Computing resources were provided by Compute Canada[1] The sampling routine and subsequent data processing was performed in parallel on two 12-core Intel E5-2650 CPUs. The initial learning rate was set to 0.001 and decayed by a factor of 10 using a step schedule of 30 epochs. The batch size of each training step was set to 128. Stochastic Gradient Descent (SGD) with momentum ($\beta = 0.9$) and weight decay (equal to $1e^{-4}$), was used to optimize a cross-entropy loss function. The CNN was trained for a total of 100 epochs. After each epoch, model performance was measured using the validation set and a checkpoint of the model was saved if the measured F-1 score increased. As such, an early stopping criteria was not used, however, if the model began to overfit to the training set, the best model with respect to the validation set was still maintained. Finally, the training process was repeated ten times using different random number generator seeds.

[1]Compute Canada: `https://alliancecan.ca/en`

3.2.2 Experimental Results

Performance metrics for each ResNet-50 CNN were compiled as the mean and 95% confidence interval over the ten training runs. Performance is measured in terms of precision, recall, and F-1 score and was evaluated on the test set listed in Table 3.2.

The classifier trained on the novel representation outperforms the remaining classifiers trained on single-channel inputs. However, two-sample t-tests indicate that the improvement in performance between the classifier trained on the novel representation is not quite statistically significant (maximum $p = 0.0881$). Figure 3.4 contains two confusion matrices corresponding to two of the better performing classifiers: those trained on the novel representation and the single-channel linearly scaled spectrogram produced using a window length of 2048 samples.

Table 3.3: Mean performance and 95% confidence intervals of ten training/testing runs using different random number generator seeds.

	FFT length	precision	recall	F-1 score
Freq. in Hz	256	0.714 ($\pm$0.060)	0.641 ($\pm$0.037)	0.675 ($\pm$0.046)
Freq. in Hz	2048	0.863 ($\pm$0.036)	0.838 ($\pm$0.039)	0.850 ($\pm$0.023)
Freq. in Hz	16384	0.860 ($\pm$0.032)	0.847 ($\pm$0.058)	0.853 ($\pm$0.031)
Freq. in mels	2048	0.762 ($\pm$0.067)	0.723 ($\pm$0.048)	0.742 ($\pm$0.044)
Novel representation	-	**0.887** ($\pm$0.045)	**0.871** ($\pm$0.036)	**0.878** ($\pm$0.031)

The performance of the CNN trained on linearly scaled spectrograms using an FFT window of 2048 frames is compared to those trained on linear spectrogram generated with window lengths of 256 and 16,384 frames. The comparison is accomplished via two-sample t-tests of statistical significance. In comparison to the CNN trained on spectrograms with window lengths equal to 256, the model trained using an FFT length of 2048 is preferable ($p = 3.375e^{-14}$). This finding is not necessarily surprising as the smaller FFT length is more suited for short sweeping vocalizations such as whistles, which the species of interest in this section do not make. The CNN trained on spectrograms with an FFT length equal to 16,384 performed equally well in terms of statistical significance ($p = 0.8086$). This is likely due to the fact that the longer FFT window is well suited to the display low infrasonic moans and pulse vocalizations

made by blue and fin whales, respectively. However, due to the smaller variance in performance presented by the CNN trained using FFT lengths of 2048 samples, it is likely the preferable model; offering a balance between low-frequency vocalizations and the down-sweeps. The performance of the CNN trained on mel-spectrograms did not exceed that of the linearly scaled spectrograms ($p = 6.451e^{-11}$) due to the fact that the logarithmic scaling of spectrograms below 1000Hz is somewhat negligible.

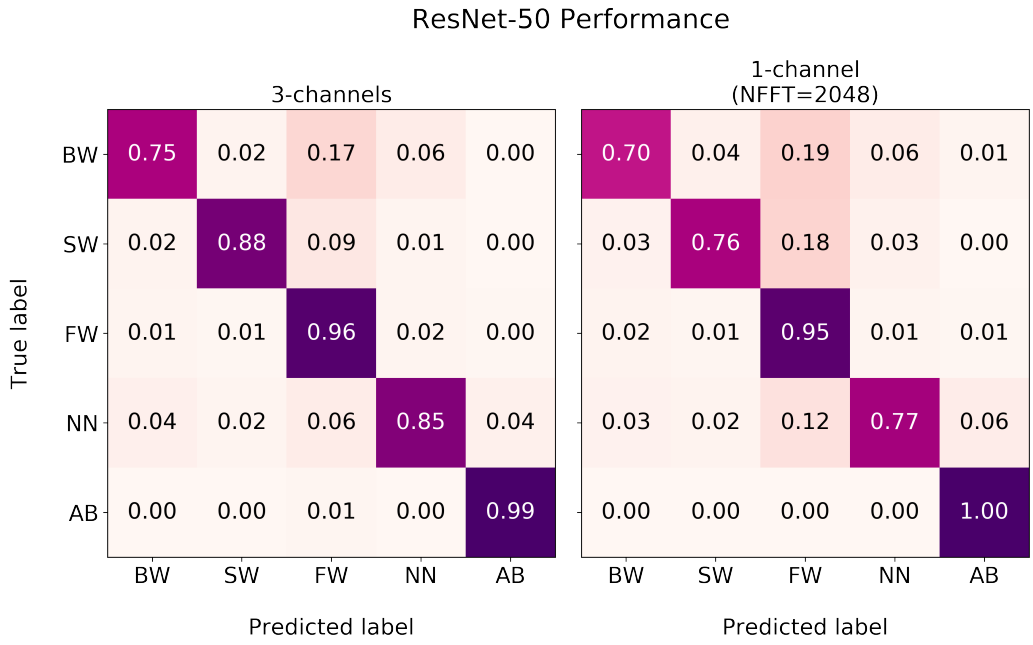


Figure 3.4: Normalized confusion matrices of the two best performing classifiers in terms of F-1 score.

While the reported performance metrics provided above are significant, spectrogram classification of the species of interest in this section (i.e., blue, fin, and sei whales) is only a partial accomplishment. Moreover, it is difficult to directly compare the performance of our trained CNN against the traditional approaches presented in Section 2.6 as most traditional methods are aimed at detecting individual vocalizations. In the following section, a full DCS is introduced that builds upon the ResNet-50 model presented here in order to detect individual vocalizations.

3.3 R-CNNs for Vocalization Detection

The contents of this section are largely taken from the workshop proceedings: Thomas, Mark, et al. *"Detecting Endangered Baleen Whales within Acoustic Recordings using Region-based Convolutional Neural Networks."* Joint Workshop on AI for Social Good at Neural Information Processing Systems (NeurIPS), 2019.

3.3.1 Data set and Methods

The same ESRF acoustic recordings used in training the CNNs of Section 3.2 were used to train a R-CNN for vocalization detection. There were however several slight differences in making the training, validation, and testing data sets. First the R-CNN makes use of the entire bounding box annotation as opposed to only the start/end times used in the sampling procedure of Section 3.2.1. Second, the ambient and non-biological noise classes are omitted as the detection of these acoustic sources is baked into the background class of the R-CNN. Third, the model was trained on five second spectrograms instead of the ten second spectrograms used earlier. Reducing the length of the spectrograms by half was necessary in order to deal with GPU memory constraints. In other words, due to the way the RPN of an R-CNN inflates the number of training instances in each mini-batch, the size of the original spectrograms had to be reduced in order to fit in memory. The spectrograms used to train the R-CNN used a FFT window of length 2048 samples, overlapped by 25 percent, and a Hann windowing function. The novel representation of section 3.2 is not employed here as the computational overhead of generating three spectrograms instead of one coupled with the more computationally intense R-CNN model made for slow experimentation. Again, the spectrograms were truncated using an upper frequency bound of 1000Hz and scaled to dBs before being normalized between [0, 1].

The R-CNN architecture used in this section was proposed by Ren et al. [95], known as Faster R-CNN and introduced in Section 2.3.1. The R-CNN was implemented in Python using PyTorch [88]. The same ResNet-50 architecture trained for spectrogram classification was used as a feature extraction layer, coupled with a Feature Pyramid Network (FPN) [69]. The 256 output features of the FPN are then handed to the RPN producing 1000 RoI proposals per training instance. The 1000

RoIs are then passed through the RoIAlign procedure and the head network composed of fully connected layers for classification and bounding box regression.

The R-CNN was trained for 100 epochs with early stopping being evaluated on the loss of the validation set. Four NVIDIA P100 Pascal GPUs each with 16GB of memory and a batch size of 4 were used for training. The initial learning rate of SGD with momentum ($\beta = 0.9$) was set to 0.003 and decayed by a factor of 10 when the loss of the training set plateaued.

3.3.2 Experimental Results

The results detailed in this section depict the median of ten training runs using different random number generator seed values.

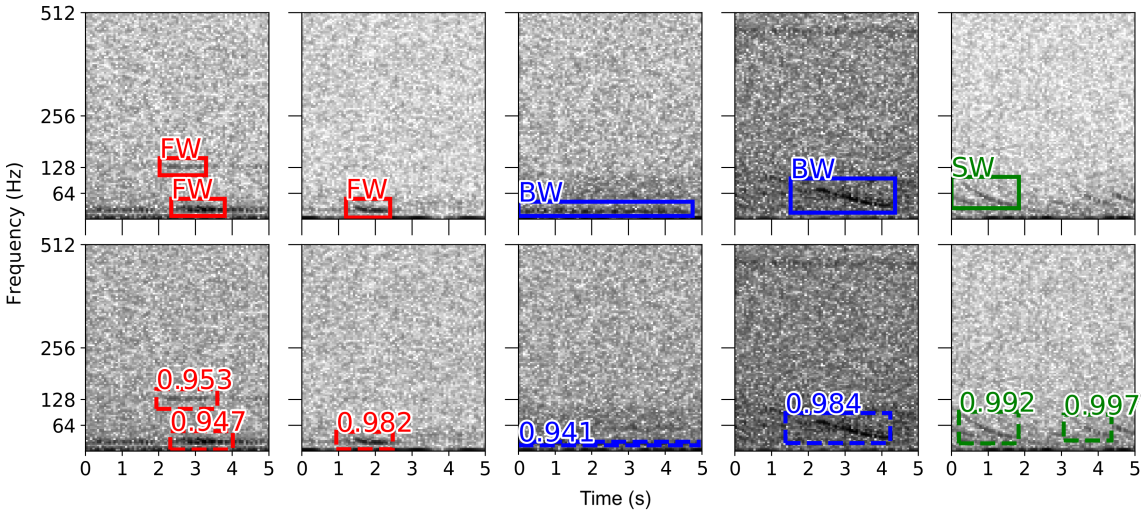


Figure 3.5: Example annotations (top row) and corresponding predictions made by the R-CNN (bottom row) for several example vocalizations produced by the three species of interest. As previously mentioned vocalizations have only been partially labelled, for example: in row 1 column 5, there appears to be several sei whale vocalizations occurring consecutively, however, only one has been annotated. Interestingly, the R-CNN has detected two vocalizations and therefore the reported metrics for this test instance would be artificially low.

The R-CNN performs well when considering a low Intersect over Union (IoU) threshold (e.g., AP@.5) between the ground truth bounding boxes and the R-CNN predictions, as reported in Table 3.4. The performance drops for IoU values larger than 0.7, which is reflected through the mAP column of the same table. This is

an acceptable result especially as the area of the bounding boxes being predicted is relatively small in comparison to those found in the training data of natural images for which these metrics were first created.

Table 3.4: Median values of average precision (AP) evaluated over various IoU thresholds as described in the COCO Detection Challenge

Species	Label	AP@.5	mAP@.5:.95
Overall	-	82.1	41.8
Blue whale	BW	85.7	52.8
Fin whale	FW	75.3	30.8
Sei whale	SW	85.4	41.9

The R-CNN described in this section represents a true detection and classification system of endangered baleen whale vocalizations. Such a system obtained via end-to-end deep learning is novel to the area of PAM of cetaceans. Using this approach, one can detect the vocalizations of blue, fin, and sei whales at an individual vocalization level and these detections can be used to estimate call distribution, species abundance, and subsequently better understand the behaviour of these species.

3.3.3 DCS Use and Adaptability

The R-CNN developed in this section represents the first iteration of a system known internally at JASCO Applied Sciences as MammalNet. The primary goal in developing MammalNet was to assist in the detection and classification of blue, fin, and sei whale down-sweep vocalizations, particularly as these vocalizations are difficult to distinguish using traditional methods like spectrogram correlation. A secondary goal was to implement this model directly into the manual analysis software PAMlab used by marine biologists at JASCO, the Department of Fisheries and Oceans (DFO) and Defence Research and Development Canada (DRDC). As PAMlab is developed in Java, the model trained above was converted using the Open Neural Network Exchange[2] compatibility standard and implemented into PAMlab via a Java runtime environment. The result is a fully working blue, fin, and sei whale DCS that can be used by marine biologists and other stakeholders on data collected in various parts

[2]Open Neural Network Exchange (ONNX): `https://onnx.ai`

of the world. Furthermore, the underlying code for running MammalNet in PAMlab is adaptable and can be used to easily implement CNNs and/or R-CNNs trained to classify or detect new species that vocalize at similar or dissimilar frequencies; provided they have been converted to a compatible ONNX format. Figure 3.6 contains a screenshot of the MammalNet DCS running in PAMlab.

While this effort was successful in implementing MammalNet into PAMlab, the run-time performance was deemed to slow for operational use. This fact is largely due to the relatively large computation cost of the R-CNN model. These issues as well as more direct comparisons between DL-based DCS and traditional methods used in industry are addressed in detail in Chapter 5.

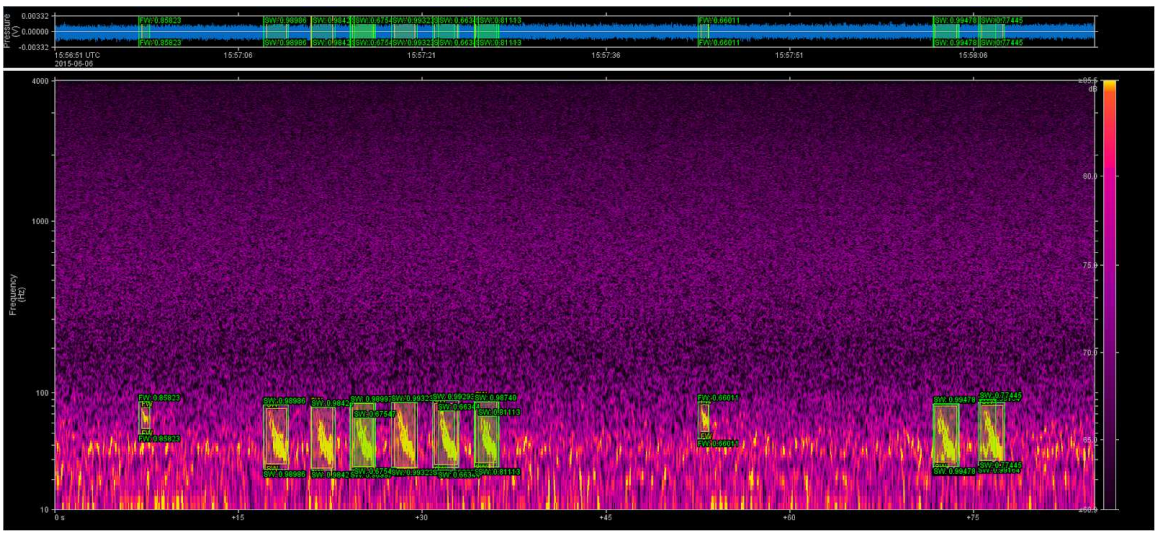


Figure 3.6: Example predictions (green) and expert annotations (yellow) presented in PAMlab. The file used in this example was randomly selected from a set of data not used during training, validation, or testing. The acoustic file was recorded in June 2015 off the coast of Nova Scotia in the Gully Marine Protected Area.

Chapter 4

Learning from Unlabeled Passive Acoustic Data

The contents of this chapter are largely taken from the workshop proceeding:
Thomas, Mark, et al. *"Leveraging Unlabelled Data through Semi-supervised Learning to Improve the Performance of a Marine Mammal Classification System."* From Shallow to Deep: Overcoming Limited and Adverse Data Workshop at International Conference on Learning Representations (ICLR), 2021.

4.1 Background, Data Sets, and Methods

The advantages of deep learning applied to PAM DCS development are contrasted with a few disadvantages. For one, fully supervised frameworks for learning deep neural networks have been notoriously branded as "data-hungry" [63, 64], i.e., requiring enough labeled training data to ensure that the correct patterns or latent features are learned instead of simply exploiting data memorization. The necessity to acquire large amounts of labeled training data is a by-product of the number of parameters typically found in a deep neural network, often measured in millions–or more recently, even billions–of model weights. Secondly, while deep learning has been presented as a means of learning abstract and adaptable features, in some cases, the same algorithms can behave poorly on instances that are deemed *out-of-distribution* (OOD) [7].

As it relates to PAM, deep learning faces two challenges. Due to the speed/scale at which PAM data is collected and the percentage of this data that is then manually annotated, labeled training data is inordinately scarce. The result of which is usually observed through model bias, sometimes referred to as *over-fitting* [103], in favour of the training data set. Second, depending on the sampling strategy used to determine which individual acoustic recordings should be analyzed, manual annotations may lack enough variation such that the developed algorithm is robust to OOD instances. For example, the acoustic sources during the first month of a several month deployment may vary dramatically to that of the following month depending

on weather conditions, the presence of different species of marine mammals, and additional sources of non-biological noise. From a machine learning perspective this is not unlike the phenomenon known as *data drift* [126]. Additionally, something should be said regarding ones interest in making use of the vast amounts of data that have been collected purely out of consideration to the cost of data acquisition.

Sparsely annotated training data sets are not exclusive to PAM. Many scientific applications that employ deep learning solutions are faced with data scarcity, especially those applications for which manual annotations are expensive to collect or require expert knowledge (e.g.: medical annotations of MRI scans [47]). To this end, a great deal of research has been conducted that aims to develop deep learning algorithms that learn from both labeled and unlabeled examples. This area of research is typically categorized as semi-supervised learning. In this work, a SOTA semi-supervised learning algorithm known as MixMatch [10] is adapted to improve the performance of a deep neural network used to classify spectrograms. The spectrograms in question contain minke whale (MW; *Balaenoptera acutorostrata*) vocalizations for which there were relatively few labeled examples available during the time of training. In particular, it is demonstrated that:

i. Deep learning remains an appropriate solution to spectrogram classification even in cases where only a small number of labeled examples is available for training.

ii. Deep neural networks trained using the semi-supervised learning framework outlined in this work are more robust to OOD examples.

4.1.1 Acoustic Data Sets

There are two distinct data sets used in this work. The first data set, hereby referred to as "Set A", consists of acoustic recordings spanning roughly three months starting from late August to December of 2015. During this time, three Autonomous Multichannel Acoustic Recorders (AMARs) were strategically deployed in the Bay of Fundy in order to measure the sound pressure level (SPL) of vessels travelling in the area as well as detect marine mammal vocalizations. Of the three recorders, two were positioned along the inbound shipping lane at depths of 151 and 140 meters,

respectively, and one along the outbound shipping lane at a depth of 123 meters. Figure 4.1 contains the locations of the three deployed acoustic recorders. Each AMAR used in collecting Set A was fitted with an M36-V35dB omnidirectional hydrophone from GeoSpectrum Technologies Inc. (-165 ± 3dB re 1V/μPa sensitivity). The hydrophones were protected by a shroud-covered cage, which doubled as a means of reducing unwanted acoustic artifacts corresponding to flow noise. The AMARs operated on a 15 minute duty cycle. For each 15 minute cycle, the AMARs recorded for 10 minutes 34 seconds at 16 kilo-samples per second (ksps) bounded between 10Hz and 8kHz, followed by 64 seconds at 375ksps (10Hz to 187.5kHz recording bandwidth). The acoustic recordings corresponding to each cycle were each stored on internal solid-state flash memory. Once again, as this work is primarily concerned with low-frequency baleen whale vocalizations, the data is restricted to the 24-bit 16ksps recording channel.

The second data set ("Set B") consists of a selection of acoustic recordings taken from a large scale deployment along the Atlantic Outer Continental Shelf (OCS). Six AMARs were deployed along the Atlantic OCS from late November 2017 to June 2018. The locations of the six AMARs are also contained in Figure 4.1. Similarly to Set A, each AMAR used in the OCS deployment was fitted with an M36-V35dB omnidirectional hydrophone (GeoSpectrum Technologies Inc., -165 ± 3dB re 1V/μPa sensitivity) and used hydrophone cages covered with cloth shrouds. The AMARs used in this deployment used a duty cycle schedule targeted at minimizing recording time when echo sounders were in use, as well as allowing for a mixture of single/multi-channel recordings to occur. In this work, only the single-channel data was used. The sampling rate of the single channel data varied from 8kHz and 16kHz and the recording duration also varied from 60 seconds to 10 minutes.

The two data sets serve distinct purposes. Set A is used for both model training and model validation, during which, the classifier is simply tasked with distinguishing between two classes: "contains a minke whale vocalization" vs. "does not" (i.e., binary classification). In other words the training set is limited to the explicit annotations corresponding to minke whale vocalizations. As such, instances pertaining to false alarms, either ambient noise, non-biological noise, or the vocalizations of other species, were taken from 227 *fully-annotated* files. The minke whale annotations were

taken from 160 of these files, plus an additional 512 *partially-annotated* files. Finally, roughly 10 percent (725) of the remaining non-annotated files of the deployment were processed to be used for semi-supervised learning. Figure 4.2 contains the entire data distribution with respect to time of the instances making up Set A, separated by their annotation level (fully, partially, or non-annotated). As you can see, the training data is sparse in terms of the number of minke whale annotations and highly unbalanced in favour of the possible false alarms. Moreover, from a deep learning perspective, the total number of minke whale annotations available for training is significantly smaller than is likely necessary for neural network trained using only supervised learning as a binary classification problem.

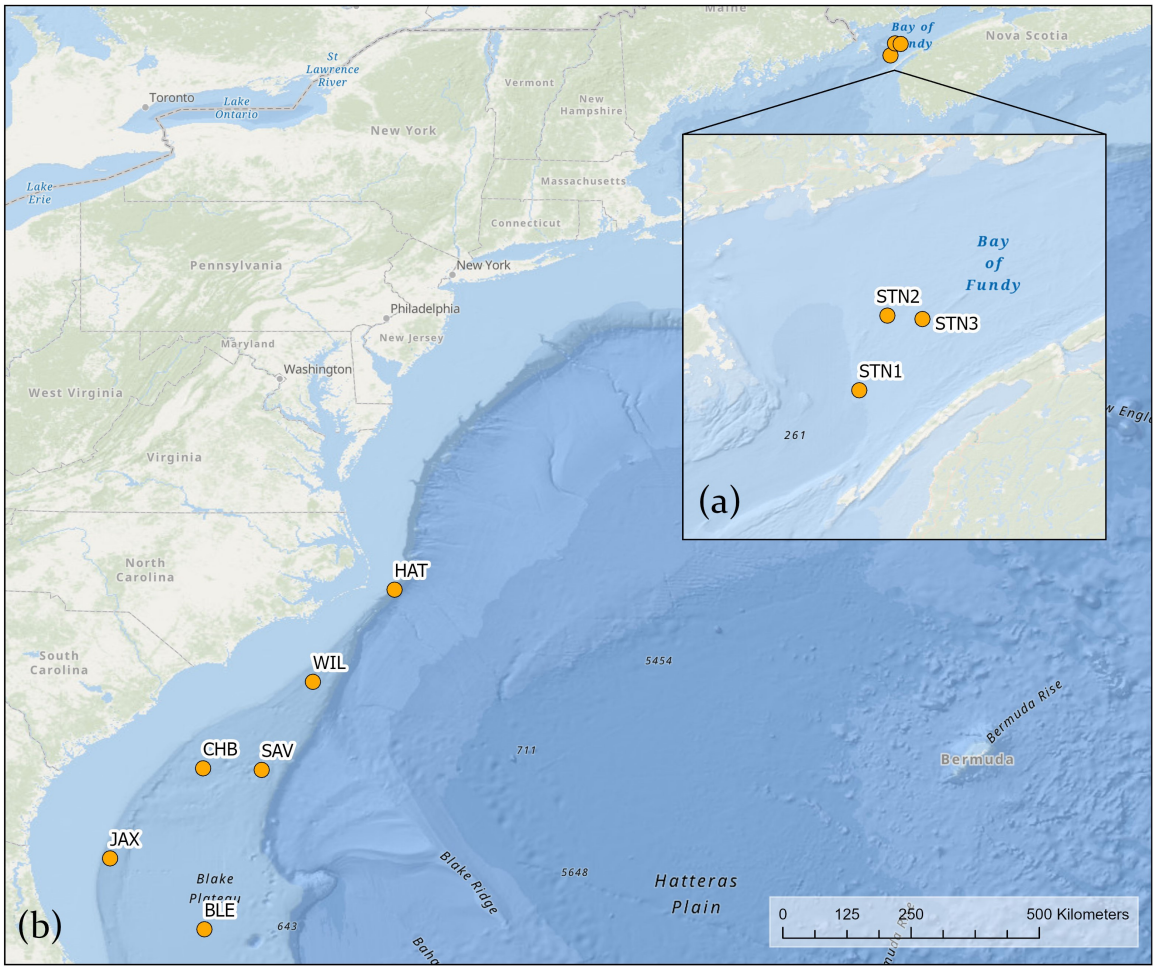


Figure 4.1: Deployment Locations: (a) Set A: three AMARs were deployed in the Bay of Fundy between August and November, 2015. (b) Set B: six AMARs were deployed along the Atlantic Outer Continental Shelf (OCS) from late November 2017 to June 2018.

Table 4.1: Deployment locations and recording depths of the AMARs used in collecting Sets A and B.

Set A: Bay of Fundy (model training/validation)			
station	longitude	latitude	depth (m)
STN1	44°33‘29.4600″ N	66°20′1.4400″ W	151
STN2	44°46′7.9200″ N	66°15′15.7800″ W	140
STN3	44°45′32.9400″ N	66°9′19.7400″ W	123
Set B: Atlantic OCS (model testing)			
station	longitude	latitude	depth (m)
BLE	29°15′3.5274″ N	78°21′2.7000″ W	872
JAX	30°29′33.8640″ N	80°0′11.2314″ W	317
CHB	32°4′12.9000″ N	78°22′26.5794″ W	404
HAT	35°11′58.3800″ N	75°1′13.3680″ W	296
SAV	32°2′31.8480″ N	77°20′52.4394″ W	790
WIL	33°35′6.8634″ N	76°27′2.0154″ W	461

Set B represents a proxy for OOD examples and therefore is only used during model testing. Table 4.2 contains the annotation distribution of both Set A and Set B at the source level.

Unlike many other species of large baleen whales, MWs are currently not listed as endangered species under Canada's Species at Risk Act (SARA) or the Marine Mammal Protection Act (MMPA) of the United States. However, researchers believe that the species is still being threatened by various sources of anthropogenic activity, including: climate change, entanglement in fishing gear, ship strikes, increased underwater noise, and whaling [98]. For this reason, the development of automated detection and classification systems has continued to expand beyond the scope of only endangered species.

In this work, a model capable of classifying a vocalization distinct to the north Atlantic minke whale known as a "pulse train" (PT) is developed. Minke pulse trains recorded in the Caribbean have been characterized according two different forms: the "speed-up" pulse train typically occupies the 200–400 Hz band and has a duration of roughly 45 seconds and the "slow-down" pulse train which occupies a slightly tighter

250-350Hz band and lasts roughly 60 seconds [80]. A third form known as the "constant" pulse train was characterized for minke whales recorded off the Gulf of Maine [97]. More generally, the difference in minke pulse train forms is typically characterized by the inter-pulse interval (IPI), i.e., the period of time between consecutive pulses in a train. For the purpose of this work, the various forms of pulse trains are treated equally and grouped under a single class.

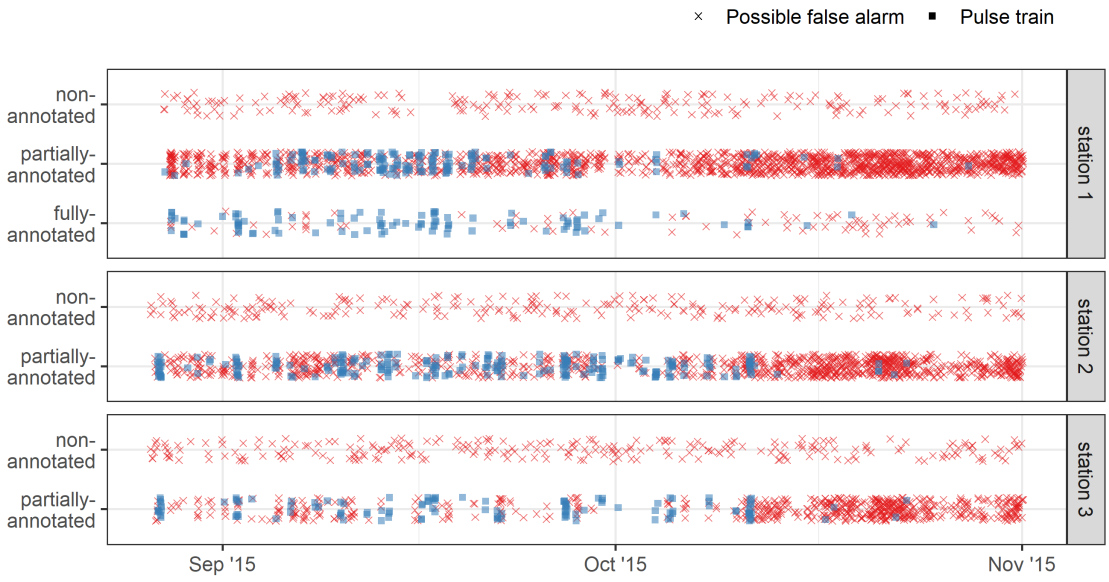


Figure 4.2: The data distribution of all files from the training and validation data set (Set A) plotted over time, factored by the deployment location (station) and annotation level (fully, partially, or non-annotated). Files that contain at least one vocalization made by minke whales known as a "pulse train" are plotted as blue squares. Files that either explicitly do not (fully-annotated files) or possibly contain a pulse train (partially/non-annotated) are plotted using red X's (i.e., possible sources of false alarm). The plotted figure only contains data until November 1 of 2015 as there were no minke whale annotations during the final two months of the deployment. A slight *jitter* was added to each point in order to distinguish files from the same date.

Due to the relatively long signal length of the minke whale pulse train, it is not uncommon to see overlap between pulse trains and the vocalizations of other baleen whales. For the case of the training data (Set A), pulse trains often overlapped with the much shorter low-frequency vocalizations of fin whales (FW; *Balaenoptera physalus*), humpback whales (HB; *Megaptera novaeangliae*), and North Atlantic right

whales (RW; Eubalaena glacialis) recorded in the area. In the case of the OCS deployment (Set B), the sources of false alarm included the vocalizations of sei whales (SW; *Balaenoptera borealis*), plus additional non-biological noise corresponding to vessels and distant seismic activity off the coast of West Africa.

Table 4.2: Annotation distribution for Sets A and B separated at the acoustic source level.

Set A: Bay of Fundy (model training/validation)			
Acoustic source	label	training	validation
Minke whale pulse train	MW	556	56
Ambient noise	AB	5560	620
Fin whale	FW	3383	422
Humpback whale	HB	5773	597
North Atlantic right whale	RW	462	49

Set B: Atlantic OCS (model testing)		
Acoustic source	label	test set
Minke whale pulse train	MW	336
Non-biological noise	NN	266
Sei whale	SW	62

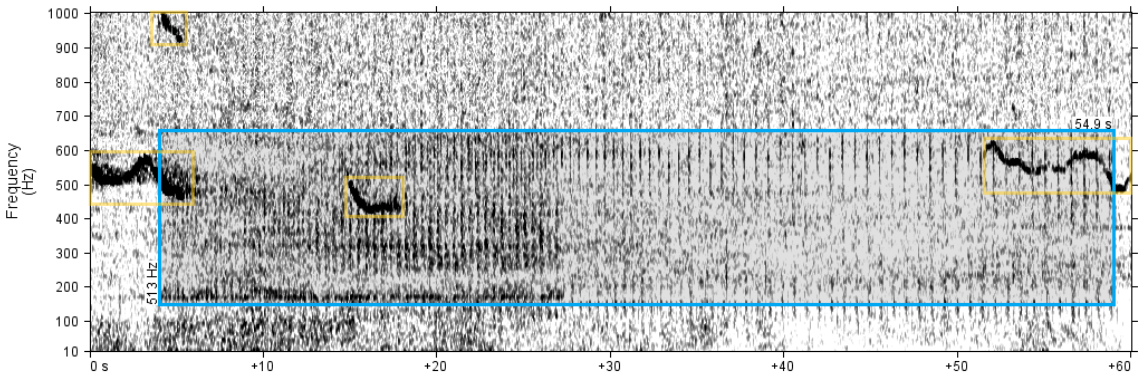


Figure 4.3: A spectrogram containing a minke whale pulse train annotated with a long blue bounding box along with several overlapping humpback whale vocalizations individually annotated using yellow bounding boxes.

Each time series corresponding to the acoustic recordings (i.e., WAV files) described above are split into 45-second segments overlapped by three seconds to ensure full coverage of the marine mammal vocalizations. Much like the previous work in Chapter 3, each 45 second segment is passed through a FFT with a window size equal to 2048 frames and is overlapped by 512 frames using a Hann windowing function. The magnitude of the spectrogram is scaled to dBs, truncated using an upper frequency bound of 1000Hz, and normalized between [0, 1]. The resulting scaled, truncated, and normalized spectrogram is equivalent to a single data instance used as input for training, validating, and testing the neural network.

4.1.2 Semi-Supervised Learning

Several variations of the commonly used ResNet CNN architecture introduced in Section 2.2.1 are trained to to classify spectrograms to one of two classes: "ambient/non-biological/other" vs. "minke whale pulse train".

The semi-supervised learning algorithm used in this work, MixMatch, was presented as a SOTA approach to handling label scarcity across a variety of image classification tasks [10]. MixMatch was chosen due to its performance on image classification tasks on data sets similar in size to that described above. MixMatch is grounded upon the use of two types of data-augmentation [106]. First MixMatch makes use of the similarly named *mixup* data-augmentation strategy such that the neural network learns a *vicinal* distribution [16] rather than an *empirical* distribution of the data. In other words, *mixup* constructs a single training instance with features $\tilde{x}$ and labels $\tilde{y}$ as a linear combination of two instances;

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j \end{aligned} \tag{4.1}$$

where (x_i, y_i) and (x_j, y_j) are chosen at random from the entire data set and the random variable $\lambda \sim Beta(\alpha, \alpha)$ controls the intensity of the vicinal distribution. Models trained using vicinal risk minimization (VRM) have been shown to be less susceptible to data memorization than those trained using the standard paradigm often used in machine learning–and that which has been discussed thus far–known as *empirical risk minimization* (ERM) [128, 129]. Such models trained via VRM have also been shown to be more robust to OOD examples [114].

The second data-augmentation strategy is used to produce predictions of unlabeled instances under the assumption that a robust classifier should predict the same class label for a transformed instance $a = g(x_i)$ as it would for a transformed instance $b = g(x_i)$ if the stochastic function g corresponds to a commonly used image transformation such as cropping, rotation, warping, or adding noise. A single prediction for x_i is obtained following a "sharpening" routine:

$$p_c = \frac{p_c^2}{\sum_j p_j^2}, \tag{4.2}$$

where p_c is the average probability over two stochastic data augmentations that x_i belongs to class c.

Briefly, the entire MixMatch algorithm can be described as follows. At each iteration of the model training process, MixMatch maintains two equally sized sets of data, X and U, one for which labels are available and the other whose labels are sharpened predictions from the model currently being trained. Finally, two vicinal data sets $\tilde{X}$ and $\tilde{U}$ are created using *mixup*.

The semi-supervised loss function used for training a classifier via MixMatch is the combination of the supervised cross-entropy loss ($\mathcal{L}_X = \mathcal{L}_{CE}$) and an unsupervised loss $\mathcal{L}_U$ weighted by a hyper-parameter λ_U, i.e., $\mathcal{L} = \mathcal{L}_X + \lambda_U \mathcal{L}_U$. The unsupervised loss is calculated as the squared Euclidean norm between the sharpened label guesses and predicted model output.

$$\mathcal{L}_U = \frac{1}{N|\tilde{U}|} \sum_{u,q \in \tilde{U}} ||q - f(y|u;\theta)||^2, \tag{4.3}$$

where $f(y|u;\theta)$ is the predicted output of the classifier provided input u and model parameters θ, and N is the number of possible classes.

MixMatch is explicitly dependent on several hyper-parameters. Suggestions from the literature [10] narrow the scope of this work to two of these hyper-parameters: the weighting parameter of the combined loss function (λ_U) which controls how much emphasis should be placed on optimizing $\mathcal{L}$ with respect to the unlabeled data, and the VRM distribution parameter (α) which determines how dominant one instance should be over the other during *mixup*.

The second data augmentation strategy of the MixMatch algorithm is necessary for sharpening the guessed labels of the classifier and in turn the stability of the loss

function. However, as one may surmise, most of the common data transformation techniques used in traditional image classification do not transfer directly to the spectrogram domain. For example, one common strategy is to randomly crop the input image before passing it to the model. However, in the case of a spectrogram such a transformation will alter the frequency representation of the input. Similarly, rotating or flipping the input horizontally are not safe transformations. Instead, a set of transformations known as SpecAugment [87] is relied upon, whereby, random masks are added vertically (in time) or horizontally (in frequency) to the spectrogram. In the time domain, assuming a spectrogram with τ time steps, SpecAugment randomly masks the time steps $[t_0, t_0 + t)$ where $t_0 \sim \mathit{Unif}(0, \tau)$ and $t \sim \mathit{Unif}(0, T)$. Similarly, for a spectrogram with ν frequency bins, SpecAugment masks bins $[f_0, f_0 + f)$ where $f_0 \sim \mathit{Unif}(0, \nu)$ and $f \sim \mathit{Unif}(0, F)$. The hyper-parameters T and F correspond to the maximum number of time steps and frequency bins, respectively, that can be masked.

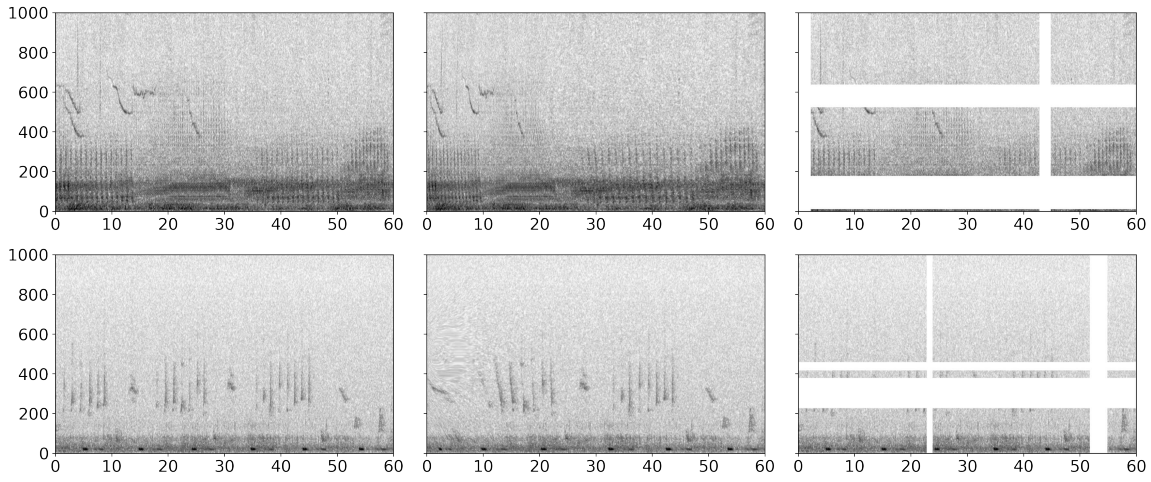


Figure 4.4: Two example data instances before and after being passed through the SpecAugment data augmentation routine. The top row of spectrograms contains consecutive minke whale pulse trains below 400Hz as well as several humpback whale vocalizations bounded between 400 and 600Hz. The second row of spectrograms contains a possible source of false alarm pertaining to humpback whale vocalizations bounded between 200 and 500Hz coupled with several fin whale 20Hz pulse vocalizations. Column one (starting from the left) contains the original data instance. Column two contains the same data instance after being "time-warped" using SpecAugment ($W = 200$). Column three contains two example time and frequency masks per instance using maximum mask widths of $T = 50$ and $F = 50$.

In addition to masking, SpecAugment includes a hyper-parameter W used to warp the spectrogram in the time domain using a polyharmonic spline between time steps $(W, \tau - W)$ either to the left or right of the center of the spectrogram by some distance $w \sim \mathit{Unif}(0, W)$.

Figure 4.4 contains two data instances before and after being transformed through SpecAugment using various hyper-parameters.

4.2 Experimental Results

Two subsets of models were trained for the classification task of identifying marine mammal vocalizations within spectrograms. First, three baseline models (ResNet-18, 50, and 101) were trained using the fully-annotated and partially-annotated data listed in Table 4.2. The baseline model was trained by minimizing the supervised loss function $\mathcal{L}_X$ of Equation 1, this is analogous to the concept of ERM mentioned previously. Second, multiple ResNet models were trained via semi-supervised learning, following the MixMatch algorithm described in Section 4.1.2, using various MixMatch and SpecAugment hyper-parameters. These models made use of both the fully/partially-annotated data as well as the available non-annotated data.

Both the baseline and semi-supervised models were implemented using the Python deep learning library PyTorch [88]. Model training was distributed over two NVIDIA V100 GPUs each equipped with 16GB of memory. Both models were trained for roughly 30 epochs (i.e., complete iterations over the data) or until early stopping was deemed necessary as the performance of the model on the validation set started to decrease. The ResNet-18 and ResNet-50 baseline models were trained on mini-batches of 64 instances, while the ResNet-101 baseline was trained on slightly smaller batches of 32 instances due to GPU memory restrictions. Using two data augmentations of the unlabeled set, the semi-supervised model was trained using mini-batches of size 48 (16 labeled examples plus 2×16 unlabeled examples). Initial attempts at training the CNNs using the unbalanced class distributions described in Set A did not perform favourably. Therefore, the use of a balanced sampling routine is maintained during training. In other words, the labeled examples of each mini-batch were distributed evenly across the two classes. For example, during the training of the baseline ResNet-50 model, 32 of the 64 instances making up a mini-batch belonged

to the minke whale class and the remainder was taken from the collection of other sources listed in Table 4.2. Similarly, while training the models via semi-supervised learning, 8 of the 16 labeled examples were taken from the minke whale class. The initial learning rate of both the baseline model and semi-supervised models was set to 0.001 and decayed by a factor of ten after the training loss plateaued. The Adam [55] algorithm introduced in Section 2.2.2 is used to optimize the semi-supervised loss function defined above. Per the suggestions of the authors [10], the hyper-parameter λ_U was increased linearly from 0 to the provided value over the first five epochs. After each epoch, the performance of each model was evaluated on the validation data set. Errors were not back-propagated through the network using the validation set. The highest performing model in terms of F-1 score on the validation set was maintained after each epoch.

4.2.1 Baseline vs. Semi-supervised

After training various combinations of models and hyper-parameters, it was observed that the VRM parameter (α) did not significantly impact performance. Also of note is that the performance of the classifiers on the validation set decreased when using the more complex ResNet-50/101 architectures. This phenomenon can be attributed to the relatively small data set used during training which allowed the models with more parameters to over-fit to the training data. Due to this decrease in performance the results obtained from training ResNet-50 are omitted and the performance of the 18-layer architecture with the much deeper 101-layer version are presented instead.

These experiments demonstrated the SpecAugment hyper-parameters were more important compared to those used for MixMatch, with the exception of the time warping parameter (W) which did not appear to impact performance. This finding aligns with comments made by the authors of the original SpecAugment paper [87]. In general, it was found that a balanced number of frequency and time masks performs favourably. The best performing models in terms of F-1 score were observed using two time and frequency masks with $T = 69$ and $F = 26$; equal to roughly 10 percent of the dimensions of the spectrogram (691×257).

Table 4.3 contains the direct comparison of overall performance between the models trained using semi-supervised learning versus those trained using only labeled

data. The first row of each sub-table in Table 4.3 represents the performance of the baseline (fully-supervised) model, while the subsequent rows correspond to models trained via semi-supervised learning. Classifier performance was evaluated on the validation data of Set A and is presented in terms of precision, recall, and F-1 score. The values depicted represent the median training run in terms of F-1 score after training each model five times using different random number generator seeds. The performance metrics contained in Table 4.3 use a VRM hyper-parameter of $\alpha = 0.5$ and several values of the unsupervised loss hyper-parameter (λ_U).

Table 4.3: Performance comparison of the baseline and semi-supervised CNN architectures in terms of precision, recall, and F-1 score, measured on the validation data of Set A: Bay of Fundy. The best performing models are highlighted in bold.

ResNet-18 CNN Architecture				
Training paradigm	λ_U	precision	recall	F-1 score
Supervised (ERM)	-	0.79700	0.85807	0.82641
Semi-supervised (VRM)	5	0.77950	0.92624	0.84656
	10	**0.81645**	**0.90301**	**0.85755**
	25	0.79570	0.90153	0.84532
	50	0.80782	0.90242	0.85250
	100	0.73856	0.93043	0.82346
ResNet-101 CNN Architecture				
Training paradigm	λ_U	precision	recall	F-1 score
Supervised (ERM)	-	0.73344	0.80124	0.76585
Semi-supervised (VRM)	**10**	**0.78889**	**0.93577**	**0.85607**
	25	0.74697	0.89709	0.81538
	50	0.78212	0.88308	0.82954

Regardless of the choice of model architecture, the performance of the models trained using semi-supervised learning appear to outperform those trained strictly using labeled data. This can be seen by comparing rows one and three of the performance metrics for the ResNet-18 CNN and comparing rows one and two for the

ResNet-101 CNN. Using a paired sign test of population medians, the increase in performance of the ResNet-101 architecture is statistically significant ($p = 0.00364$). This is a substantial finding as it implies the features learned by the CNN were positively influenced using unlabeled data. Moreover, the fact that models trained with lower values of the VRM hyper-parameter (α) performed on par with those where $\alpha > 0.5$, means the increase in model performance is not strictly due to the use of VRM. In general, it was observed that setting λ_U to a value between 10 and 50 lead to a well performing model in terms of F-1 score. The hypothesis is that values in this range provide a better balance between $\mathcal{L}_X$ and $\mathcal{L}_U$ instead of putting too much emphasis on either one of these terms during training.

4.2.2 Out-of-Distribution Performance

The two baseline models trained using fully-supervised learning from Section 4.2.1 as well as the best performing models trained using semi-supervised learning were used to classify the data from Set B. Table 4.4 contains the overall performance metrics of the semi-supervised classifiers versus the baseline models.

Table 4.4: Performance comparison of the baseline and semi-supervised CNN architectures in terms of precision, recall, and F-1 score, measured using the data from Set B: Atlantic OCS. The best performing models are highlighted in bold.

ResNet-18 CNN Architecture			
Training paradigm	precision	recall	F-1 score
Supervised (ERM)	0.75213	0.75178	0.75195
Semi-supervised ($\lambda_U = 10$)	**0.89658**	**0.88799**	**0.89226**
ResNet-101 CNN Architecture			
Training paradigm	precision	recall	F-1 score
Supervised (ERM)	0.72769	0.69944	0.71329
Semi-supervised ($\lambda_U = 10$)	**0.85406**	**0.85282**	**0.85344**

The models trained using MixMatch outperform the models trained using only labeled data. The observed increase in performance is even larger for Set B, clearly demonstrating that the baseline models are more susceptible to training bias and poor OOD performance when the number of available training examples is small.

Again, using a paired sign test of population medians, the increase in performance is statistically significant ($p = 0.00364$). Moreover, the performance of the semi-supervised models on Set B actually exceed that of Set A, demonstrating that the semi-supervised CNNs are more robust and can generalize to unseen acoustic data collected in distinct locations, at varying depths, and are less susceptible to unknown acoustic sources.

Chapter 5

Operational DCS

The contents of this chapter are largely taken from the report:
Thomas, Mark, et al. *"Operational Neural Networks for Marine Mammal Detection and Classification"* Technical report for Defence Research and Development Canada (DRDC), 2023.

5.1 Background and Requirements of Operational DCS

The research reported up to this point–both within this document and related work conducted by others–has largely been effective at demonstrating that CNN-based automated DCS are viable candidates for detecting marine mammal vocalizations post hoc, i.e., using data that has already been collected via moored recording devices. Far less time and attention has been spent on analyzing the functionality of the developed DCS in terms of its use in real-world and real-time applications. Largely, this is due to the various operational constraints that are not necessarily a problem when conducting research. In fact, one could argue that much of the related work outlined in Section 2.7, while certainly beneficial to the bio-acoustics community, also fails at demonstrating operational deep learning-based DCS.

Developing a DCS presents several technical challenges that have hindered the operational utility of the research discussed so far. For one, passive acoustic monitoring is a multidisciplinary research topic involving acousticians, marine biologists, oceanographers, physicists, engineers, and computer scientists. The process of training, evaluating, and deploying an automated DCS is challenging for all, but the level of difficulty may vary dramatically depending on the expertise of the researcher. Second, of the constraints not faced when purely conducting research, perhaps the most presumptuous is computational efficiency. Many of the automated DCS reported in the literature fail to mention how well they perform on lightweight/low-power devices. Such devices are effectively a requirement if the end-goal of the system is to determine

species presence/absence in real-time. And finally, while deep learning models have shown to perform well on acoustic data from the same environment that they were trained on, their performance begins to degrade on new data collected in different locations for a variety of reasons, including: seasonality, weather, bathymetry, the type or configuration of the recording device, and local noise conditions.

This chapter addresses the above challenges in the following ways:

1. An end-to-end pipeline, with very little required human input, is developed in order to democratize the training and evaluation process involved in DCS development. Additionally, guidelines on how PAM data sets should be selected and separated into train/test splits are provided, such that, the resulting systems are more robust and generalizable.

2. High-performing and lightweight alternatives to the two-stage detection models used in Section 3.3 are developed, and measures of the computational/energy requirements to run these models in real-time (or faster) on edge devices are compiled.

3. Methods for updating neural networks via transfer-learning and sending the updated network parameters over-the-air are presented. These methods are particularly effective when the automated DCS is running on an edge device or autonomous platform and network bandwidth is extremely limited.

The following work was completed through a Department of National Defence (DND) IDEaS Grant and conducted at JASCO Applied Sciences (JASCO).

5.2 Standardized methods for training and evaluation

The task of producing an automated DCS (operational or otherwise) can be divided into three stages: data set selection/creation, model training, and model evaluation. In this section, each of these stages is outlined in turn, with the caveat that ML research, especially deep learning, is a highly iterative and experimental process, whereby one stage may hinge on another, and stages may be explored simultaneously.

Paramount to this work is the fact that deep learning has sometimes been presented as more of an art than a science. The intention is to divide the stages above

into digestible and tractable steps such that this art is closer to "paint-by-numbers". A great deal of consideration went into the selection of software, libraries, and tools used to create the pipelines described below. In some sense one intends on removing the "gate-keeper" such that more efficient experimentation can be carried out by researchers with varying levels of DL expertise.

5.2.1 Data Set Creation

Perhaps the most critical component of any ML/DL algorithm is the quality of the data set on which the algorithm is trained. The neural network architectures that have proved successful so far in this work are variations of CNNs and often used on structural data in the form of images. The most suitable visual analogue for the task of marine mammal detection and classification is the spectrogram [56, 105, 117]. The steps of combining annotations stored in a relational database with WAV files stored in a data-warehouse to create a large corpus of spectrogram/label pairs used in training and testing detection models are summarised below.

JASCO maintains a large database containing over 1.4 million annotations (as of October 2022) in the form of bounding boxes (start/end times, low/high frequencies) drawn around a variety of sounds in spectrograms. Such sources include marine mammals, fish, and non-biological sources. A pipeline was developed that takes several configuration parameters and outputs a directory containing binary records. Each record contains one-or-more spectrograms, corresponding annotations when available, and additional metadata. The data set creation pipeline was developed using Python in a flexible way such that it could be run on a single machine using built in Python multiprocessing or distributed across a larger computing cluster using Apache Beam[1], Flink[2], or Spark[3]. The pipeline operates using the following steps:

1. Query JASCO's annotation database to select the annotations that make up the data set.

2. Loop over the unique WAV files from the returned set of annotations and stream the contents of these files in blocks equal to the requested length (seconds) of

[1]Apache Beam: `https://beam.apache.org`
[2]Apache Flink: `https://spark.apache.org`
[3]Apache Spark: `https://spark.apache.org`

the spectrogram.

3. Perform the FFT, creating a spectrogram from the streamed block and apply any requested transformations (e.g., scaling, normalization, etc.)

4. Format and carry-forward any annotations/metadata pertaining to the data block that was just processed.

5. Byte-encode the output of the previous step and save the results to a binary file.

For those working in the field of machine learning, the steps outlined above are all-to-familiar. What makes the pipeline accessible is the limited input required by the user to create data sets quickly and efficiently. The inputs to the pipeline are two-fold: a SQL query specifying which annotations in the database the user would like to select and a configuration file containing parameters used during the creation of the spectrogram (e.g., length in seconds, frequency range, and normalization). The SQL query can be as detailed or as general as necessary. Alternatively, if the user cannot access the database at that time, a CSV of annotations that have been previously exported from the database can be provided and the SQL query will not be executed. An example SQL query that matches the schema of the JASCO annotation database is contained in Listing 5.1. Similarly, the configuration file can include as much or as little detail as the user requires to develop their algorithm. The contents of an example configuration file are presented below (Listing 5.2).

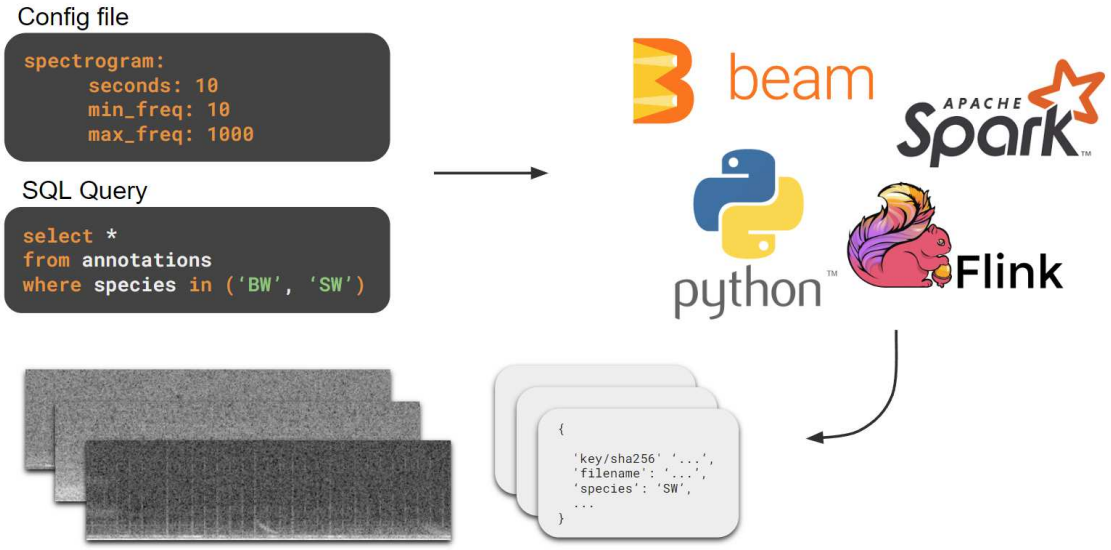


Figure 5.1: Outline of the pipeline used in creating a DCS development data set. The requirements to the pipeline include a configuration file that may specify several parameters used when running the FFT and a SQL query for selecting the annotations from the database.

Listing 5.1: Example SQL query used in selecting the annotations from JASCO's annotation database to be used in the dataset creation pipeline.

```
-- Example Query:
-- Select all blue, fin, and sei whale annotations from the ESRF
-- deployment between June 2015 and July 2017, where the vocalization
-- falls in the 25-1000Hz frequency band.

select * from annotations
where (
    client = 'ESRF'
    and species in ('BW', 'SW', 'FW')
    and starttime >= '2015-06-01'
    and endtime < '2017-08-01'
    and highfreq <= 1000
    and lowfreq >= 25
);
```

Listing 5.2: Example configuration file for the data set creation pipeline. Optional parameters are prefixed as such.

```
record:
    directory: /home  # Where to put the data set
    name: my_dataset  # The name of the data set
    prefix: train  # dataset prefix (e.g., train or test)
annotations:
    data: query.sql  # Path to SQL query or CSV
    agnostic: false  # (Optional) train a class-agnostic model
    balance: false  # (Optional) balance the dataset
    true_negatives: false  # (Optional) include true-negatives examples
spectrogram:
    seconds: 10  # Length of the spectrogram
    min_freq: 10  # (Optional) lower frequency bound
    max_freq: 1000  # (Optional) upper frequency bound
    db_scale: true  # (Optional) dB scale the spectrogram
    log_scale: true  # (Optional) log-scale y-axis
    pcen: false  # (Optional) apply per-channel energy normalization
```

Processing of the pipeline is done in parallel on a per-file basis. For example, if there are 8 unique WAV files returned in step 1 above, on a 4-core/8-thread CPU, each thread would process a single file concurrently and the entire batch of files would be processed in one phase. On a workstation with many cores (e.g., the 12-core/24-thread CPU used for much of this work) processing time is achieved quickly even for large data sets in excess of 50,000 examples. Additionally, it was discovered through experimentation that streaming the contents of a file in step 2, as opposed to loading the entire file into memory, led to a 10x improvement in run-time.

The most challenging aspect in creating a well-suited data set using the pipeline developed in this work is defining the problem statement which subsequently characterizes the SQL query or optional CSV. And while the quality of the annotated acoustic data at JASCO is notable, there is no perfect data set as far as ML or DL is concerned. Furthermore, the stochastic nature of acoustic data exacerbates several challenges one often faces when training and testing neural networks: model generalization and robustness to out-of-distribution data. It is likely the case that many of the DL-based PAM algorithms reported in the literature have been developed under sub-optimal conditions. The standard approach to dividing data into training and test sets is through simple random sampling. This approach does not provide sufficient evidence that a system designed for PAM generalizes in any way. During this

work, a set of guidelines and heuristics was formulated that outlines how to select training and test data sets. These guidelines were presented to internal and external stakeholders to garner feedback. It was argued that the importance of the test data set exceeds that of the training set if one intends on using the developed detector for operational PAM. Several suggestions in creating data sets for PAM development include:

1. Separate the training and test sets at
   (a) bare minimum: hold out individual WAV files
   (b) better: hold out entire stations or time frames from the full data set
   (c) best: hold out time frames, stations, and even full acoustic deployments
2. Visualize the time/geographical overlap of your data sets to ensure the same data is not trained and tested on by accident (e.g., two recording stations very close in proximity picking up the same low-frequency sound)
3. Pick good FP/FN/TN examples (e.g., low signal-to-noise, new soundscapes, overlapping sources, etc.)

By creating the data sets used throughout this work with these considerations in mind, the hypothesis is that the results presented in this chapter fairly represent model generalizability, while maintaining a balance between model bias and model variance. Further ablation studies to demonstrate this hypothesis are required and discussed in the proposed future work (Chapter 6).

5.2.2 Model Training

Much like the pipeline developed for creating data sets, a great deal of effort went into creating a model training code base with very little configuration requirements. There are an abundance of open-source tools and libraries available within the deep learning community, all of which have benefits and drawbacks, however, as one of the goals in this research is to make DCS development more accessible, making use of these resources, which have existing levels of community engagement and support, seemed like the best approach going forward. A suitable application programming interface

(API) known as the Tensorflow (TF) Object Detection API was identified. The TF API consists of several pre-existing workflows for training SOTA detection algorithms with little user input, apart from standard configuration parameters like batch size, learning rate, and data augmentations. Later on, while continuing to work on the additional goals of this work, notably those pertaining to transfer learning (Section 5.4) and model quantization (Section 5.5.1), a second open-source Python library "YOLOv5" was identified as being suitable for the task of model training.

As will be discussed later on, committing to a single framework may prove costly in a fast-paced field like machine learning. It was decided to use both libraries in tandem with the benefit of being able to compare results of varying models. This resulted in the development of two code repositories for training detection models using different ML frameworks: Tensorflow and PyTorch. Both of which can be used to compile a trained model for inference on edge devices.

Using either of the developed code bases, the training procedure is quite similar and hinges on a configuration file that is used by wrapper functions to the TF Object Detection API and YOLOv5. The configuration file includes a few important pieces of information:

i. The name of the detection architecture one would like to use.

ii. The location of the training and test data sets.

iii. A list of the classes and their labels

iv. Optional hyper-parameters such as the initial learning rate, batch-size, and how often to save model checkpoints.

An example configuration file for the model training procedure is available in 5.3.

Listing 5.3: Example configuration file for the model training pipeline.

```
# Base directory of the dataset
dataset: /home/my_dataset

# Location of train/test under the base directory
train: train
test: test

# The classes in the dataset
classes:
  0: BW
  1: FW
  2: RW
  3: SW

# The name of the model to train
model: yolov5n

# Optional hyper-parameters
batch_size: 16
learning_rate: 0.001
optimizer: adam
...
```

Both code repositories implement various optimization routines, loss functions, and data augmentation strategies. For the remainder of this chapter, one can assume that model training in both PyTorch and Tensorflow was performed using the NVIDIA CUDA Toolkit, a parallel computing framework and API that uses graphical processing units (GPUs) to perform the many multiply-accumulate (MAC) operations required in a single pass of a neural network. Two varieties of GPUs were used for training the neural networks: a NVIDIA RTX 3090 GPU with 24GB of VRAM was used to train models locally at JASCO Applied Sciences and computation resources using a NVIDIA Tesla V100 with 16GB of VRAM were provided by Compute Canada.

5.2.3 Model Evaluation

The performance metrics introduced in Section 2.4.2 (i.e., Average Precision and mean-Average Precision) are used in evaluating the DCS detectors developed throughout this chapter. However, these metrics were derived for computer vision tasks and not widely used in industry PAM applications [39]. To benchmark the deep learning-based DCS against methods currently used in industry, notably the contour

detectors at JASCO [77], performance is evaluated on a per-file basis rather than per-vocalization.

Per-file Performance Metrics

The contour detectors currently in use at JASCO are deterministic and therefore do not yield interpretable confidence scores. Additionally, the contour detectors were developed with the goal of species presence/absence detection in mind, as opposed to counting individual vocalizations. To measure performance of the contour detectors in terms of presence/absence, JASCO compiles the number of TP, TN, FP, and FN on a per-file basis. The procedure for computing P, R, and F-score on a per-file basis is as follows:

1. Select a small sample of files to be annotated by a manual analyst. At JASCO this is performed using "Automatic data selection for validation" (ADSV [58]) such that the distribution of the sample is as closely aligned as possible to the full acoustic data set.

2. Set an integer threshold value (≥ 1) to be used in Step 3

3. For each annotated file from the sample of Step 1, check whether the number of detections from that file exceeds the threshold set in Step 2. If so, this file is labeled as a 1, otherwise a 0. The result is a large table of binary presence/absence predictions.

4. Again, iterating over each file, compare the presence/absence ground truth to the output of Step 3 to determine if the contour detector ran on that file is a TP, TN, FP, or FN.

5. Sum the TP, TN, FP, and FN counts from Step 4 and compute P, R, and F-score using the standard equations 2.15, 2.16, and 2.18.

6. Increase the threshold set in Step 2, and iterate through this process, optimizing for one of P, R, or F-score. Notably at JASCO this process is generally optimized to improve an additional metric known as Matthew's Correlation Co-efficient (MCC).

7. Repeat this process for each detector/species to be evaluated.

To directly compare the output of the automated DCS to JASCO's contour detectors, a thresholding and optimization system was implemented. As the CNN-based detection models contain confidence scores, two thresholds are used in Step 2 of the procedure described above: one threshold for the number of detections and a second threshold on the confidence score. As the number of detectable species in the detector increases, so does the number of threshold parameters. The system for determining threshold values was implemented such that if the number of species is small (such as the examples in this chapter), a brute-force approach to iterating over a grid of possible values can be used. If the detection model can detect many species, the choice of thresholds is treated as a multi-dimensional optimization problem and optimized using a Genetic Algorithm (GA).

The topic of model evaluation was subject to several productive conversations throughout the duration of this work, again with both internal and external stakeholders. What level of granularity performance is measured should depend on the goal of the operational system. For the purpose of presence/absence detection, per-file computations are informative and likely sufficient. For additional down-stream tasks such as density estimation, more refined metrics such as mAP may prove useful. Also of note is the ability to optimize the predictions of a model for precision over recall (or vice versa). In some cases, for example when attempting to detect highly endangered species, one may be willing to sacrifice lower precision for higher recall. In other cases precision may outweigh recall, such as a desire to reduce unnecessary engagement by a human operator.

5.2.4 Inference on Edge Devices

Several neural network architectures were considered for this work, all of which can be described as CNN-based object detection algorithms. While the two-stage R-CNN detector developed in Section 3.3 performed well, the additional proposal stage required by these networks generally leads to less efficient models in both size and runtime. In Section 3.3.3, we alluded to the limitations of using R-CNNs for performing inference (i.e., using a trained model to make predictions on unseen data) at or close to real-time. Because of this, emphasis was placed on training less computationally

expensive single-stage detectors: EfficientDet and YOLO (Section 2.3.2).

These models and others are part of a growing trend in the development of lightweight networks for use on mobile and edge devices. A by-product of the increased interest in lighter neural network architectures is the increased availability of hardware to run these networks efficiently. One such piece of hardware is the "Edge TPU" from Google; an Application-Specific Integrated Circuit (ASIC) that is purposefully designed to perform neural network inference.

The Asus Tinker Edge T (hereby referred to as "Tinker board"; Figure 5.2) is a commercially available single board computer (SBC) similar in design to a Raspberry Pi, with the additional benefit of an on-board Edge Tensor-processing unit (TPU). Briefly, TPUs are machine learning accelerators that speed up the processing of neural network inputs in a similar fashion to the CUDA Toolkit described above. Edge TPUs are computationally efficient and consume less power than a conventional CPU or GPU. As the Edge TPU is a Google product it has been optimized to run TensorFlow Lite (TFLite) models, and thus TFLite was adapted as the compiled network format for this work.

Figure 5.2: Image of the Asus Tinker Edge T SBC. Image source: product information page from ASUS.

There are multiple ways to create a TFLite model. The EfficientDet model was implemented using the Tensorflow Object Detection API and converting these models to TFLite is straightforward using the Tensorflow Lite Converter. For the case of YOLO, the models were first transformed from PyTorch to a standard ONNX format and subsequently from ONNX to TFLite. In both cases, the conversion process allows for some additional modifications to the model to reduce latency and size if required.

The output in both cases (EfficientDet and YOLO) is a compiled ".tflite" file can that be run via an interpreter on the edge device (i.e., Tinker board). The results of running the YOLO model on the Tinker board are well documented in the latter sections of this chapter. These results indicate the power draw of the boards while running model inference. When the Tinker board is idle its power draw is roughly 2.5 watts (W). No attempt to shutdown peripherals, remove unnecessary bloat, or sleep the boards between operation was investigated during this work.

5.3 Automated DCS Results

Experiments with models of various sizes and complexities are carried out for several different data sets and species. In total, there were five model architectures used from each of the EfficientDet and YOLO families of models, respectively. The average footprints of the model in megabytes (MBs), number of parameters, and necessary floating-point operations per second (FLOPs) required to complete a single pass through each network are compiled in detail in Table 5.1. The values presented in the table are estimates based on the models reported in Section 5.3.1. Depending on the size of the input (i.e., spectrogram), the number of FLOPs and parameters may differ slightly from the values listed below.

5.3.1 Blue, Fin, Right, and Sei Whale Detector

In the first experiment, the models listed above were trained to detect the audible vocalizations of blue whales (BW, *Balaenoptera musculus*), fin whales (FW, *Balaenoptera physalus*), north Atlantic right whales (RW, *Eubalaena glacialis*), and sei whales (SW, *Balaenoptera borealis*). The data set used to train the BW/FW/RW/SW detector is comprised of acoustic data from three Atlantic Ocean deployments. The training data set consisted of acoustic data recorded by the Bedford Institute of Oceanography (BIO) at the Gully Marine Protected Area (MPA; May 2015 to April 2016) and the Environmental Studies Research Fund (ESRF) project on the Scotian-shelf (August 2015 to July 2017). The test data set from BIO was recorded in the Emerald Basin over multiple years (May 2015 to April 2016; September 2016 to November 2017). The spectrograms used for training and testing were 10 seconds in length and bounded between 10 and 500 Hz. Each spectrogram was scaled

logarithmically in the frequency axis and the magnitude of the STFT was scaled in decibels (dBs). Notably, the two data sets were taken from separate deployments and contain no geographical or temporal overlap. The data were selected in such a way to ensure a good measure of generalizability. Infrasonic vocalizations made by blue and fin whales were not used in training or testing these models.

Table 5.1: Estimated number of parameters (measured in millions (M)) , floating-point operations per second (FlOPS, measured in billions (B)), and memory footprint (measured in megabytes (MB)) of the YOLO and EfficientDet families of models.

EfficientDet			
Model Size	Parameters (M)	FLOPS (B)	Size (MB)
D0	3.9	2.5	22
D1	6.6	6.1	34
D3	12	25	79
D5	34	135	197
D7	52	325	235

YOLO			
Model Size	Parameters (M)	FLOPS (B)	Size (MB)
nano (n)	1.8	4.1	7
small (s)	7.0	15	28
medium (m)	21	48	80
large (l)	46	107	178
extra-large (x)	86	204	330

All models were trained on a GPU (RTX 3090 or V100) for a total of 300 epochs (i.e., iterations through the entire data set). The batch size was set to 16 for the EfficientDet models and 32 for the YOLO models. The neural networks were optimized via gradient descent using the Adam optimizer with an initial learning rate of 0.001 that decayed to 0.00001 following a cosine learning rate schedule [73]. The YOLO models were trained using 16-bit floating point precision, as is standard for this model, while the EfficientDet models used 32-bit floating points; theoretically giving an upper hand to EfficientDet. This training routine is maintained throughout the remainder of the report with the only exception being altering the number of

epochs on occasion.

The performance of the YOLO family of models significantly outperforms that of the EfficientDet family, as demonstrated in Tables 5.2 and 5.3. Much of the detriment of the EfficientDet models appears to be its poor performance on FW vocalizations whose bounding boxes are particularly small. Interestingly, the smallest YOLO model, YOLO-n, also seems to struggle with detecting FWs, but the second smallest YOLO model (YOLO-s), does not. Also of note is that the performance of the YOLO models seems to peak at the "large" (YOLO-l) model size. This is likely due to the "extra-large" model having more parameters and potentially over-fitting to the training data.

The data used to train the BW/FW/RW/SW detector is by-and-large partially labeled. Meaning that in a single spectrogram with multiple vocalizations, there is a high probability that only one of the vocalizations has been annotated. Unfortunately, partially labeled data will negatively impact the performance of the model. This research has successfully demonstrated that partially labelled data can be used for the task of spectrogram classification (Chapter 4), and the intention is to conduct future work to adapt these approaches for the task of detection. One hypothesis is that the vocalizations of FWs are particularly challenging when partially labeled.

Deep learning research is an iterative and experimental process. As a result, the remaining work focuses on the YOLO family of models due to its increased performance, resulting in faster and more effective experimentation.

Comparison to JASCO's Contour Detector

To compare the results of these models against the contour detectors currently in use at JASCO, the results are restricted to two of the better performing YOLO models at different ends of the size-spectrum (YOLO-s and YOLO-l). The neural network and contour detectors are compared on a per-file basis, of which, the method for converting per-vocalization predictions to per-file predictions is described in Section 5.2.3.

A perfect one-to-one comparison of the detectors is still slightly out of reach due to the way the contour detectors and neural networks were developed. The contour detectors are mostly separated by call-class or call-type whereas the neural networks

Table 5.2: Overall performance of the YOLO models trained to detect the vocalizations of BW, FW, RW, and SW

YOLO Model Size	**Species**	**P**	**R**	**AP@.5**	**mAP@.5:.95**
nano (n)	All	0.554	0.453	0.441	0.270
	BW	0.666	0.667	0.671	0.394
	FW	0.115	0.225	0.047	0.027
	RW	0.660	0.450	0.480	0.273
	SW	0.775	0.460	0.567	0.385
small (s)	All	0.858	0.546	0.607	0.498
	BW	0.844	0.651	0.746	0.569
	FW	0.836	0.477	0.497	0.429
	RW	0.816	0.555	0.610	0.479
	SW	0.935	0.499	0.577	0.514
medium (m)	All	0.884	0.525	0.607	0.479
	BW	0.811	0.590	0.687	0.511
	FW	0.857	0.467	0.501	0.408
	RW	0.947	0.538	0.635	0.496
	SW	0.920	0.504	0.605	0.499
large (l)	All	0.919	0.526	0.620	0.515
	BW	0.817	0.656	0.701	0.560
	FW	0.942	0.513	0.537	0.456
	RW	0.960	0.532	0.624	0.506
	SW	0.959	0.494	0.620	0.539
x-large (x)	All	0.880	0.551	0.623	0.488
	BW	0.808	0.669	0.742	0.555
	FW	0.811	0.497	0.515	0.432
	RW	0.966	0.541	0.632	0.470
	SW	0.936	0.497	0.603	0.494

Table 5.3: Overall performance of the EfficientDet models trained to detect the vocalizations of BW, FW, RW, and SW

EfficientDet Model Size	**Species**	**P**	**R**	**AP@.5**	**mAP@.5:.95**
D0	All	0.529	0.423	0.417	0.238
	BW	0.588	0.623	0.617	0.352
	FW	0.301	0.234	0.164	0.075
	RW	0.694	0.507	0.550	0.380
	SW	0.533	0.326	0.337	0.144
D1	All	0.559	0.437	0.447	0.278
	BW	0.680	0.621	0.656	0.432
	FW	0.320	0.247	0.176	0.078
	RW	0.684	0.574	0.606	0.436
	SW	0.554	0.307	0.351	0.169
D3	All	0.561	0.436	0.445	0.284
	BW	0.659	0.631	0.665	0.435
	FW	0.285	0.230	0.166	0.084
	RW	0.742	0.580	0.595	0.447
	SW	0.560	0.302	0.353	0.171
D5	All	0.561	0.457	0.463	0.306
	BW	0.678	0.656	0.687	0.477
	FW	0.311	0.239	0.177	0.087
	RW	0.692	0.583	0.614	0.475
	SW	0.562	0.350	0.373	0.186
D7	All	0.547	0.462	0.458	0.286
	BW	0.664	0.663	0.683	0.452
	FW	0.294	0.248	0.167	0.080
	RW	0.683	0.588	0.601	0.428
	SW	0.546	0.349	0.381	0.185

operate at the species level. The results of Table 5.4 represent a best approximation of a one-to-one comparison.

In general, the neural network performs as well, if not slightly better for most species with the exception of BWs. The results of the per-file detector on BWs suggest that additional data containing false positive examples should be used during training. Specifically, it was observed that a large degree of over-prediction (i.e., many false positives) for BWs when using the YOLO-l model. The performance of the contour detector for this species is far superior.

Table 5.4: Per-file performance comparisons of the YOLO detector against JASCOs contour detector. No comparative metrics were available for detecting North Atlantic right whales (RW).

Blue Whale (BW)			
	YOLO-s	YOLO-l	JASCO
Precision	0.41	0.33	**0.93**
Recall	0.52	0.94	**0.96**
F-1 Score	0.46	0.49	**0.94**

Fin Whale (FW)			
	YOLO-s	YOLO-l	JASCO
Precision	**0.97**	0.96	0.95
Recall	**0.80**	0.80	0.79
F-1 Score	**0.88**	0.87	0.86

Sei Whale (SW)			
	YOLO-s	YOLO-l	JASCO
Precision	**0.84**	0.84	0.61
Recall	**0.92**	0.91	0.73
F-1 Score	**0.88**	0.87	0.66

Inference Run-times

As a side result, when running the YOLO models to compile per-file performance comparisons, the run-time required to evaluate 110 WAV files was also logged. Each WAV file is roughly 10 minutes in length and sampled at 8kHz. Evaluation was performed in parallel on a 24-thread CPU, i.e., 24 instances of the model were initialized and ran concurrently. The evaluation run-times described below include the process of loading the models into memory, creating the TFLite interpreters, data streaming, and FFT computations (Table 5.5). Finally, this analysis was performed using two levels of model quantization. The values listed below are total run-times as opposed to per-file run-times and present a significant promise on running these models on entire acoustic deployments in a very short amount of time. Notably, a significant decrease in run-time is observed when models are quantized at 8-bits. As will be discussed in Section 5.5.1, this advantage is also present when performing model inference on edge devices.

Table 5.5: Measured runtimes of the YOLO detection models when evaluating 110 files each 10 minutes long at sampled at 8kHz. Inference was parallelized on a 24-thread CPU. Measurements include the the process of loading the models into memory, creating the TFLite interpreters, streaming the data, and performing the FFT.

Model Size	**16-bit Runtime**	**8-bit Runtime**
nano (n)	12s	11s
small (s)	21s	16s
medium (m)	58s	29s
large (l)	2m 16s	54s
x-large (x)	4m 47s	1m 35s

5.3.2 Humpback and Infrasonic Fin Whale Detector

Several research projects at JASCO with an emphasis on manual annotation have resulted in acoustic deployments that include many fully annotated files. Multiple deployments containing fully annotated files were identified for training detection models for humpback (HB, *Megaptera novaeangliae*) vocalizations and infrasonic calls

made by FWs. These vocalizations have proven difficult for the neural networks when the data sets used for training are only partially annotated. Because these species generally vocalize in sequence, if only the first of a series of vocalizations is annotated, the detection model in unintentionally penalized for correctly predicting the remaining calls in the partially annotated sequence and learning stagnates.

The data set used to train the HB and infrasonic FW detector is comprised of acoustic data from four Atlantic Ocean deployments. The training data set was recorded by Stantec in the Bay of Fundy (August 2015 to December 2015). The test data sets consisted of acoustic data from BIO that was obtained in the Gully MPA (May 2015 to April 2016), ESRF project on the Scotian-shelf (August 2015 to July 2017), and the University of New Hampshire (UNH) on the United States (US) Outer Continental Shelf (OCS; December 2017 to June 2018). The spectrograms used for training and testing were 30 seconds in length and bounded between 10 and 2000 Hz. Each spectrogram was scaled logarithmically in the frequency axis and the magnitude of the STFT was scaled in dBs. The train and test data sets share a small degree of temporal overlap in terms of their collection (fall 2015), however, they are in completely different locations (the closest being the Gully found along the Scotian Shelf and the Bay of Fundy). Therefore, these data sets should present very little risk of bias and remain good measures of generalizability.

Several experiments were performed to test the efficacy of the detection models for these species/call-types. First, a detector was trained for each species separately, following which, a multi-class detector was trained on both species' annotations. While the species' vocalizations used in training the models above don't overlap, these experiments are informative in measuring any possible trade off between running multiple models independently or combining their detection into a single model. In this case, the model trained on both species at once outperformed the two models trained separately. This behaviour is-—in all likelihood-—simply a matter of entropy. Not only are we providing the model with more data to learn generalizable features, but the instances pertaining to the other class can act as false positive signals to train on.

Table 5.6: Performance of the YOLO-s/l models trained to detect infrasonic FW vocalizations.

Model Size	P	R	AP@.5	mAP@.5:.95
small (s)	0.717	0.719	0.747	0.344
large (l)	0.706	0.688	0.711	0.320

Table 5.7: Performance of the YOLO-s/l models trained to detect HB vocalizations.

Model Size	P	R	AP@.5	mAP@.5:.95
small (s)	0.717	0.775	0.800	0.567
large (l)	0.902	0.875	0.932	0.675

Table 5.8: Performance of the YOLO-s/l models trained to detect both HB and infrasonic FW vocalizations.

Model Size	Species	P	R	AP@.5	mAP@.5:.95
small (s)	All	0.930	0.903	0.962	0.760
	FW	0.969	0.944	0.988	0.808
	HB	0.892	0.863	0.936	0.712
large (l)	All	0.874	0.867	0.926	0.691
	FW	0.939	0.925	0.975	0.763
	HB	0.809	0.808	0.878	0.619

5.3.3 Minke Whale Pulse Train Detector

Finally, a third set of models were trained to detect minke whale (MW, *Balaenoptera acutorostrata*) pulse trains (PTs). The detection models trained for identifying the vocalizations listed thus far have all been relatively short in duration (e.g., a few seconds) and as a result, the input spectrogram to these models has been short (maximum of 30s). The PT emitted by MWs is much longer in duration: lasting up to 45-60s per vocalization [97]. Two data sets containing annotated MW PTs were used for training and testing YOLO models. The data set used to train the MW detector is comprised of acoustic data recorded by UNH on the US OCS (December 2017 – December 2020) while the test data set was obtained from Stantec in the Bay of Fundy

(August 2015 – April 2016). The spectrograms used for training and testing were 45 seconds in length and bounded between 10 and 2000Hz. Again, each spectrogram was scaled logarithmically in the frequency axis and the magnitude of the FFT was scaled in dBs.

Much like the x-large model trained to detect BW, FW, RW, and SW, the larger models trained to detect MW appear to overfit to the training set and the best performance is observed when using the smallest (YOLO-n) model.

Table 5.9: Performance of the YOLO models trained to detect minke whale pulse train vocalizations.

Model Size	**P**	**R**	**AP@.5**	**mAP@.5:.95**
nano (n)	0.776	0.644	0.733	0.444
small (s)	0.755	0.620	0.698	0.391
large (l)	0.776	0.563	0.643	0.326

5.4 Model Updates via Transfer Learning

Transfer learning is a machine learning technique in which the trained weights of a neural network are used as a starting point for a new learning task. Conceptually, transfer learning assumes that many of the features learned by the previous network are useful towards understanding the inputs corresponding to the new task. As such, the time to train the new network and the number of required training examples is theoretically lower [134]. Transfer learning was identified as a suitable technique for passive acoustic data for a few primary reasons:

- The distribution of the data at a species-level can vary broadly between PAM deployments.
- Various factors including weather, bathymetry, season, and anthropogenic sources can have a significant impact on acoustic landscape and the data collected.
- The probability of encountering new and/or unknown acoustic sources and events is high.

In a similar vein to the reasons listed above, is the concept of data drift which refers to the phenomenon where the distribution of the input data changes over time and the performance of a model trained on said data begins to degrade (sometimes referred to as model drift). Such a scenario is not uncommon in passive acoustics, where the time of a recording, the recording device or configuration, and biases of the manual analysts can result in distributional variance of the training data sets. Transfer learning has also been shown to be effective at mitigating data drift [130].

When performing transfer learning for a task with very few data, it is often preferred to "freeze" many of the initial layers of the detection model (e.g., the backbone). The model weights within the frozen layers are not updated via gradient descent. This procedure is also referred to as model "fine-tuning". Several levels of model freezing were experimented with, using various model sizes and these experiments have a direct impact on the findings of Section 5.5.

Both traditional transfer learning for new tasks and transfer learning for adjusting to data/model drift were recognized as candidates for experimentation in this work. The transfer learning experiments use the YOLO family of models. Information pertaining to what models were used as baselines for fine-tuning, how many layers were frozen, and the performance of each are laid out in the following sub-sections.

5.4.1 Adapting to New Noise Sources

A goal of this work was to demonstrate that transfer learning could be used to update a trained network to reject a new class of noise signals perhaps not seen during training. A primary example of unseen/unexpected distribution shift in an operational PAM setting is self-noise made by an ocean glider. Acoustic data collected by a Teledyne glider on the Scotian Slope was analyzed for self-noise and categorized into several annotation types. The most common example of glider self-noise overlaps with frequency bands typically associated with baleen whale vocalizations. In particular, one instance of glider self-noise lasting roughly 1 second in duration in the frequency range $\leq$ 500Hz may be confused with an impulsive vocalization made by FWs.

The self-noise glider data is a good example of possible data drift as an unseen and possibly conflicting source is present in the data and the method of data collection is

entirely different (hydrophone attached to an ocean glider versus a moored recording device). This experiment tests the efficacy of transfer learning to fine-tune a detection model to ignore glider self-noise. The experiment is broken down with the following two variables:

i. Test the effectiveness of transfer learning when freezing the YOLO model at two different levels: layer 9 and layer 24. Layers 0–9 in the YOLO model pertain to the model backbone, while Layer 24 is the classification/bounding box head.

ii. Test the effectiveness of transfer learning depending on how many new annotations are available for fine-tuning. The new number of new training examples provided to the model during fine-tuning is varied by $n = 50,\ 100,\ 500,$ and 1000.

The resulting experiment consists of 8 configurations for each YOLO model (or 40 configurations total). The baseline models used for fine-tuning on glider self-noise were those trained to detect BW, FW, RW, and SW (Section 5.3.1). The experimental results are presented in Tables 5.10 and 5.11.

Table 5.10: Resulting performance of the YOLO models frozen at layer 9 and fine-tuned on an increasing number of glider self-noise examples.

		P		**R**		**AP@.5**		**mAP@.5:.95**	
Model Size	**# Examples**	**Value**	**% Inc.**	**Value**	**% Inc.**	**Value**	**% Inc.**	**Value**	**% Inc.**
nano (n)	50	0.708	6%	0.462	1%	0.516	3%	0.346	4%
	100	0.734	10%	0.458	0%	0.520	4%	0.348	5%
	500	0.753	13%	0.457	0%	0.524	5%	0.350	6%
	1000	0.745	12%	0.466	2%	0.523	5%	0.350	6%
small (s)	50	0.794	2%	0.486	0%	0.574	0%	0.430	2%
	100	0.815	5%	0.488	1%	0.580	1%	0.435	3%
	500	0.808	4%	0.493	2%	0.579	1%	0.433	3%
	1000	0.813	5%	0.496	2%	0.579	1%	0.435	3%
medium (m)	50	0.800	4%	0.475	4%	0.582	4%	0.449	6%
	100	0.801	4%	0.480	5%	0.584	4%	0.446	6%
	500	0.796	4%	0.497	9%	0.591	5%	0.449	6%
	1000	0.814	6%	0.489	8%	0.588	5%	0.449	6%
large (l)	50	0.819	4%	0.534	4%	0.625	3%	0.487	4%
	100	0.813	3%	0.539	5%	0.626	3%	0.487	4%
	500	0.806	2%	0.539	6%	0.624	3%	0.486	4%
	1000	0.797	1%	0.545	7%	0.625	3%	0.486	4%
x-large (x)	50	0.771	1%	0.523	7%	0.589	2%	0.427	3%
	100	0.768	1%	0.522	6%	0.585	2%	0.428	3%
	500	0.796	4%	0.517	5%	0.590	2%	0.431	4%
	1000	0.782	3%	0.522	6%	0.590	2%	0.430	4%

Table 5.11: Resulting performance of the YOLO models frozen at layer 24 and fine-tuned on an increasing number of glider self-noise examples.

		P		R		AP@.5		mAP@.5:.95	
Model Size	**# Examples**	**Value**	**% Inc.**	**Value**	**% Inc.**	**Value**	**% Inc.**	**Value**	**% Inc.**
nano (n)	50	0.715	7%	0.449	-2%	0.510	2%	0.345	4%
	100	0.697	5%	0.455	0%	0.509	2%	0.346	5%
	500	0.718	8%	0.458	0%	0.516	3%	0.348	5%
	1000	0.707	6%	0.463	1%	0.519	4%	0.349	6%
small (s)	50	0.807	4%	0.479	-1%	0.579	1%	0.434	3%
	100	0.815	5%	0.479	-1%	0.582	2%	0.435	3%
	500	0.813	5%	0.490	1%	0.585	2%	0.436	4%
	1000	0.815	5%	0.488	1%	0.584	2%	0.435	3%
medium (m)	50	0.793	3%	0.462	2%	0.574	2%	0.437	4%
	100	0.796	4%	0.457	1%	0.573	2%	0.438	4%
	500	0.790	3%	0.476	5%	0.578	3%	0.440	4%
	1000	0.790	3%	0.478	5%	0.577	3%	0.439	4%
large (l)	50	0.802	2%	0.511	0%	0.617	2%	0.483	3%
	100	0.813	3%	0.517	1%	0.621	3%	0.484	3%
	500	0.814	3%	0.524	3%	0.621	2%	0.485	4%
	1000	0.800	1%	0.528	3%	0.622	3%	0.483	3%
x-large (x)	50	0.780	2%	0.500	2%	0.586	2%	0.426	3%
	100	0.775	2%	0.501	2%	0.585	2%	0.425	3%
	500	0.770	1%	0.514	5%	0.590	2%	0.428	3%
	1000	0.761	0%	0.521	6%	0.589	2%	0.426	3%

The effectiveness of transfer learning was somewhat dependent on the size of the network, the number of layers frozen, and the number of new annotations introduced to the model during fine-tuning. However—-and interestingly–this effect seems to largely disappear as the model is fine-tuned for longer (Figure 5.3). The percentage increases (% inc.) listed in the table represent an increase in performance for each metric after transfer learning. In other words, the performance of the model before transfer learning was that amount worse, and likely predicting the glider self-noise as a marine mammal vocalization.

Regardless of where the model was frozen, providing more data for fine-tuning had a very slight advantage (e.g., 4 % increase in mAP n=50 versus 5% for n=500 at layer 9). However, this increase is negligible and demonstrates that, especially for the smaller transfer learning task at layer 24, only a small amount additional examples may be required. Additional experiments pertaining to various possible sources of false alarm may be beneficial.

The glider self-noise problem was specifically selected as it presents a challenge to the deterministic contour detectors. When testing the automated DCS on data

containing glider noise, the performance of the models was relatively good to begin with, however, through transfer learning, the average precision of the model increased by an average 4–5%. This experiment demonstrates the generalizability of the neural network quite well, however, it does not demonstrate an "order of magnitude"-type increase in performance that one may hope for.

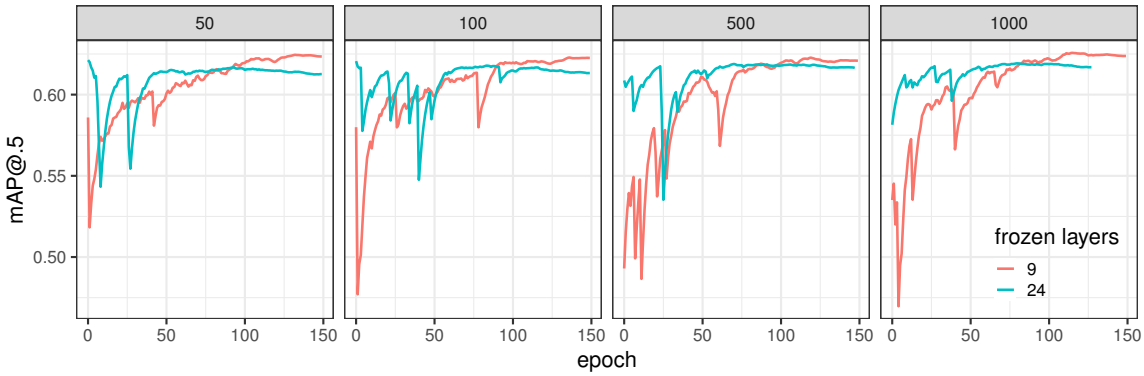


Figure 5.3: Example AP@.5 of the YOLO-l model evaluated after each epoch. The model with 9 frozen layer is able to make major adjustments to the weights of the neural network due to fewer weights being frozen, but over time the difference in performance is negligible. The figure is faceted by the number of new instances supplied to the model for training.

5.4.2 Adapting to New Locations

A data set collected on the West Coast of Canada was identified as a good example for transfer learning where the method of recording and the species one is attempting to detect remains the same (FW and HB), but the location and possible variety of vocalizations differs from the baseline model. The data set used during transfer learning in this experiment is comprised of acoustic data from one deployment recorded by Parks Canada and the Department of Fisheries and Oceans Canada (DFO) off Gowgaia Shelf (July 2017 to July 2021). The test data set came from the same location but was recorded from July 2021 to June 2022. The spectrograms used for training and testing were 30 seconds in length and bounded between 10 and 2000 Hz. Each spectrogram was scaled logarithmically in the frequency axis and the magnitude of the spectrogram was scaled in dBs.

The prominent interest in this experiment is determining whether the training process of a detection model for a new task can be expedited simply by using the

weights of a baseline model as initialization. Unlike the glider self-noise experiment above, the layers of the baseline model are not frozen and the data for fine-tuning is not restricted to some number n. This experiment aims to determine whether the features learned on data collected in the Atlantic Ocean can be used to expedite the training process for a new detection task using data from the Pacific Ocean. The baseline model used in this experiment is the YOLO-s model trained to detect FW and HB vocalizations (Section 5.3.2) and the layers of these models are not frozen during transfer learning.

Table 5.12: Performance of the YOLO-s FW/HB detector fine-tuned on data from the Pacific Ocean compared to the performance of a YOLO-s model trained from scratch on that same data.

	Species	**P**	**R**	**AP@.5**	**mAP@.5:.95**
Trained from scratch	All	0.361	0.387	0.295	0.168
	FW	0.336	0.461	0.338	0.201
	HB	0.386	0.314	0.251	0.136
Transfer learning	All	0.572	0.504	0.525	0.381
	FW	0.589	0.582	0.601	0.460
	HB	0.555	0.425	0.449	0.301

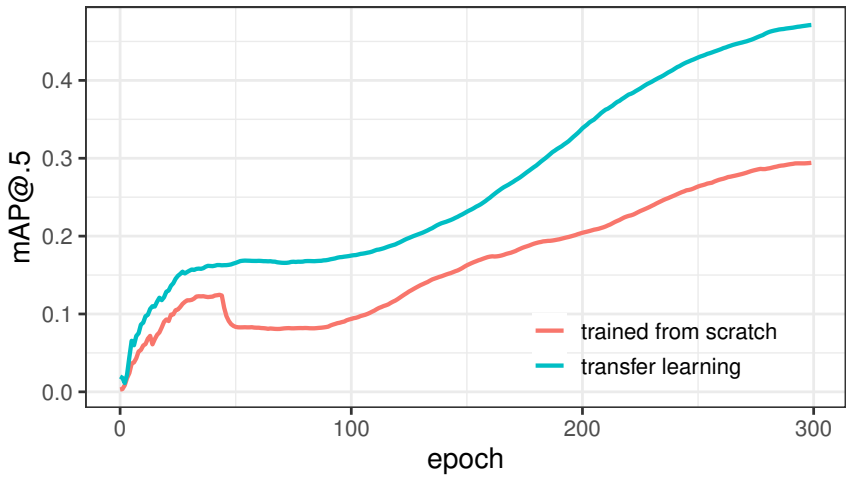


Figure 5.4: AP@0.5 measured over 300 epochs of training a model from scratch versus using transfer learning.

As anticipated, the training process of the YOLO-s model fine-tuned from a baseline was much faster than the model trained from scratch (Figure 5.4). While the model was not able to be trained for as long as perhaps desired, due to resource restrictions during experimentation, after 300 epochs, the transfer-learned model manages to outperform the model trained from scratch by roughly 55% (Table 5.12).

5.4.3 Discussion

Through these experiments, it was demonstrated that transfer learning is effective in two different scenarios. In the first scenario, the data that was used for transfer learning contained unseen and possibly overlapping energy produced by an autonomous glider. In addition, the method of data collection varied significantly from that of the acoustic data used for training the baseline model. It was shown that neural networks trained to detect baleen whale vocalizations can adapt to data drift using very few examples. The glider self-noise transfer learning experiment was successful in demonstrating that a large proportion of the models' layers may be frozen, and this finding is especially impactful when one aims to update the weights of a neural network remotely (Section 5.5). This experiment was also successful in demonstrating the generalizability of the neural network detectors. In the second scenario, the acoustic data used in transfer learning was collected in a similar fashion to the data set used to train the baseline model and also targeted to detect the same species. This experiment demonstrated that using pre-existing weights as initialization significantly decreases the time it takes to train a new model. Moreover, while further training of both models would likely be beneficial, after 300 epochs of training, the performance of the transfer learning model is significantly higher than that of the model trained from scratch.

5.5 Updating Models Parameters Over-the-Air

Updating detection models to adapt to unseen or unexpected noise is a feasible task and relatively straight-forward between deployments or periods of data collection. However, in the scenario described above pertaining to glider self-noise (Section 5.4.1), the automated DCS has been deployed to an autonomous platform with extremely limited networking connectivity, that is both slow and expensive. Even the smallest

model found in Table 5.1 (YOLO-n), with a footprint of only 7MB, far exceeds the networking limitations imposed on autonomous ocean gliders. The glider used in this work is assumed to have Iridium Certus 100 communications that supports TCP/IP connections at 3 kilobits/second and an operating service plan that permits 20 MB/-month of total data transfer at a cost of $300/month. Updating the YOLO-n model over-the-air (OTA) would eat up 35% of the monthly data allowance and require more than 2 hours of uninterrupted network connectivity. Several approaches are explored to make updating the NNs feasible in an operational setting.

5.5.1 Model Quantization

A particularly effective strategy in minimizing the footprint of a neural network is model quantization [68]. The idea behind model quantization is to represent the weights and/or activation functions of a neural network using fewer bits than the original floating-point representation. The standard floating-point representation in neural networks developed using PyTorch and Tensorflow is 32-bits. Common representations used in model quantization include 16-bit floating-points and 8-bit integers. The reduced footprint of the model not only leads to lower memory usage but also improves inference time since lower precision computations can be performed more quickly (as shown in Table 5.5). With the reduction in precision of the model weights and/or activations, comes the possibility that performance may decrease. The degree to which model quantization impacts performance is dependent on the model and the type of quantization being performed. In some cases, quantization may have little to no effect, while in other cases, the performance of the neural network may drop significantly. Several experiments were conducted, using different sized YOLO models and different model quantization levels, to determine the efficicacy of using model quantization for low-bandwidth OTA updates.

As an exploratory result, the overall performance for each of the YOLO models using 16-bit and 8-bit quantization was compiled (Table 5.13). There is a measurable drop in performance when the YOLO-n model is quantized at 8-bits. A much smaller decrease was observed for YOLO-l/x and no change in performance for the s/m variants of the YOLO model.

The estimated footprint in kilobytes (KBs) of the various YOLO models quantized

at 16 and 8-bits is presented in Table 5.14. In the best-case, the size of the model after quantization is reduced by 50-75%, however, at roughly 2MB the smallest model (YOLO-n quantized at 8-bits) is still too large for effective transmission over Iridium.

Table 5.13: Performance of the BW/FW/RW/SW detector evaluated using two levels of model quantization (16-bit floating points and 8-bit integers).

Model Size	**16-bit overall mAP**	**8-bit overall mAP**
nano (n)	0.270	0.155
small (s)	0.498	0.498
medium (m)	0.479	0.477
large (l)	0.515	0.502
extra-large (x)	0.488	0.463

Table 5.14: Size of the full YOLO models quantized using 16-bit floating points and 8-bit integers..

Model Size	**16-bit Size (KB)**	**8-bit Size (KB)**
nano (n)	3,541	1,937
small (s)	13,807	7,196
medium (m)	40,877	20,976
large (l)	90,243	45,977
x-large (x)	168,534	85,513

As a benchmark for the following sections pertaining to model separation, the YOLO detectors were run on the Tinker board producing run-time and power-consumption estimates (Table 5.15). Each size of YOLO model was run using 16-bit and 8-bit precision. The estimated run-time and power consumption includes the process of loading the models, setting up the TFLite interpreter, streaming the acoustic data, and performing the Fourier transforms. An important note is that the power draw measurements outlined in this section were collected visually by monitoring a bench power supply while each model was run or compiled. As such, only the peak power draw was collected, that is: the maximum amount of power drawn at any one point in time. Anecdotally, the average power draw was observed to be at a minimum 10% lower than peak. The power supply operated on 12V and was the not current limited.

In addition, the longer a model ran or took to compile, the more likely it was for the active cooling fan on the TPU to spin up; significantly increasing peak power draw by another 5-10%. Experiments during which the active cooling on the Tinker Board was engaged are denoted with an asterisk next to the power consumption estimate.

Table 5.15: Run time and power consumption estimates of the YOLO models on the Tinker board. A * indicates that the active cooling fan on the SBC turned on and subsequently spiked power usage.

	16-bit Float		**8-bit Integer**	
Model Size	**Run-time**	**Peak Power Usage (W)**	**Run-time**	**Peak Power Usage (W)**
nano (n)	0m 19s	4.4	0m 18s	4.3
small (s)	0m 25s	4.8	0m 23s	4.5
medium (m)	0m 43s	5.4 *	0m 36s	4.4
large (l)	1m 23s	5.5 *	1m 0s	4.4
extra-large (x)	2m 43s	5.6 *	1m 37s	4.7

5.5.2 Transmitting Partial Networks

The most straight forward option for updating the weights of a neural network on an edge device is simply transmitting an entire pre-compiled model OTA. In this scenario, no loss in performance will be observed apart from those potentially due to quantization. However, this may not be feasible provided the edge device has limited networking capabilities. In such a case, two additional strategies present themselves.

Divide the network into two sections

The first possible strategy to updating the weights of a neural network may be effective if the network was fine-tuned via transfer learning using frozen layers. In this scenario, the model may be divided into two pre-compiled sections, where the output of the first network is used as the input to the second network and only one network (likely the latter) is updated OTA. In this scenario, differences in evaluation performance are explicitly tied to the results obtained during transfer learning or model quantization. The downside of this approach is that splitting the models into two sections may

come with computational overhead and additional configuration requirements by the user.

Additional experiments were conducted that divided each model at different "freezing" points using layers 9 and 24 of the YOLO network architecture. After the models were split, they were each re-compiled into two separate TFLite models. For this work, one can assume that only the second models (i.e., those updated via transfer learning) are likely to be sent OTA and the sizes of the second models are provided in Table 5.16. The footprints of the models in Table 5.16 correspond to those after quantization at half-precision (i.e., 16-bit floating point representations).

Table 5.16: Model size measured in kilobytes (KB) and number of parameters measured in millions of the second TFLite models quantized at 16-bits.

	Split at layer 9		**Split at layer 24**	
Model Size	**16-bit Size (KB)**	**Parameters (M)**	**16-bit Size (KB)**	**Parameters (M)**
nano (n)	1,796	0.9	52	0.01
small (s)	6,925	3.5	75	0.02
medium (m)	19,968	10.2	99	0.03
large (l)	43,414	22.2	123	0.05
extra-large (x)	80,262	41.0	146	0.06

Table 5.17: Run-time and power consumption of the split models on the Tinker board. After 10 min, the experiments for the large and x-large models were stopped as they were no longer faster than real-time and deemed non-operational. A * indicates that the active cooling fan on the SBC turned on and subsequently spiked power usage.

	Split at layer 9		**Split at layer 24**	
Model Size	**Run-time**	**Peak Power Usage (W)**	**Run-time**	**Peak Power Usage (W)**
nano (n)	0m 58s	5.6 *	0m 58s	5.7 *
small (s)	2m 30s	5.8 *	2m 30s	5.8 *
medium (m)	5m 51s	5.9 *	5m 52s	5.8 *

As depicted above, freezing the model at the backbone versus the head will have a drastically different effect on the size of the second model. The smallest model update: YOLO-n, quantized using 16-bit floating points, and split at layer 24, has a transferable size of 52 KB; roughly 1.5% of the full model. Splitting the model in two parts presents slightly higher overhead on the edge device as two TFLite interpreters

must be loaded simultaneously. Run-times and power consumption estimates were measured using the Tinker board when both models are ran in sequence. These measurements are presented in Table 5.17.

Fine-tune layers and recompile the network

In a similar fashion to splitting the network in two, if the network weights were updated via transfer learning, only those weights updated during gradient descent (i.e., the unfrozen layers) may be packaged, compressed, and sent to the autonomous platform. Following which, the network is re-compiled on the edge device. It is possible that the packaged weights are quantized beyond 16-bits prior to being sent to the autonomous platform, however, this remains a technical limitation in this work and quantization beyond 16-bits is handled on-edge during re-compilation when deemed necessary. Similar to the previous method, any noticeable change in evaluation performance is due to transfer learning and should be anticipated.

The weights of the unfrozen layers after transfer learning are extracted and packaged using the Python (de)serialization package "pickle". Following which, the new weights are sent OTA to the old model and the model is recompiled to TFLite. This approach to model updating comes with the benefit of less overhead on the edge device at run-time, however, with the downside of having to recompile. Table 5.18 contains the approximate sizes (in kilobytes) of the serialized weights that would need to be transferred after splitting the model at layers 9 and 24.

As one can see, there is practically no difference in the transferrable size of the compiled model and model weights when the neural network is split at layer 9 (i.e., the backbone). This anomaly can be explained by the fact that there is a small, static amount of data required to represent the computational graph of each model. When the number of parameters being updated is large (e.g., layer 9), this static portion of the model is relatively small in comparison to the model's weights. When the number of parameters being updated is much smaller (e.g., layer 24) the impact is far more noticeable.

In the best-case scenario; transferring only the serialized weights of the neural network as opposed to the TFLite compiled second model, the amount of bandwidth required to perform OTA updates is reduced by almost 50% from 52 KB to 27 KB.

After re-compilation, the run-times for each of these models matches those reported in Table 5.15.

Table 5.18: Size of the serialized network weights measured in kilobytes (KB) after being split at the specified layer and quantized to 16-bits.

Model Size	Split at layer 9	Split at layer 24
	Size (KB)	Size (KB)
nano (n)	1,749	27
small (s)	6,878	50
medium (m)	19,915	74
large (l)	43,355	98
extra-large (x)	80,197	121

Table 5.19: Time to recompile and peak power consumption of each YOLO model on the edge device. A * indicates that the active cooling fan on the SBC turned on and subsequently spiked power usage.

Model Size	Recompilation time	Peak Power Usage (W)
nano (n)	7m 6s	4.8
small (s)	8m 48s	5.3 *
medium (m)	13m 54s	4.9 *
large (l)	23m 32s	5.6 *
extra-large (x)	30m 14s	6.1 *

5.5.3 Discussion

It is likely the case that updating an entire neural network is not feasible under the connectivity assumptions for autonomous gliders, for example over Iridium where bandwidth, down-time, and cost is a limiting factor. The smallest quantized model explored in this work is roughly 1.9 MB in size, exceeding the amount one could feasibly send over Iridium by several orders of magnitude. By dividing the network in two and/or updating only a small portion of the network via transfer learning,

one is able to minimize the amount of bandwidth required to update the model from 1.9 MB (YOLO-n 8-bit quantized full model) to 52 KB (YOLO-n 16-bit split model) or 27 KB (YOLO-n 16-bit updated weights). Moreover, the performance of the split/weight updated models would likely exceed that of the full models due to the level of quantization used. With additional experimentation, its is likely that one may overcome issues of quantizing the updated model weights to 8-bits and likely reduce the size of the Iridiam transfer by another 50%, or roughly 14 KB.

This research demonstrates that there are advantages and disadvantages to the various methods of updating models OTA. A few additional factors that go into selecting a method of OTA updating should also be considered:

- When the model can be updated in its entirety, one is not subject to how many or which layers to freeze during transfer learning or fine-tuning. Given our current satellite technology, updating the entire network is likely infeasible and such an approach to OTA updates is more suitable to situations when the autonomous system has a direct connection to shore.

- One fact not mentioned above pertaining to model splitting, is that one may prefer to split the model into many pieces (>2) and perform transfer learning using multiple approaches. Moreover, it is possible to swap or stack an entirely new architecture to/on the previous models so long as the shape and datatypes of the inputs/outputs remain the same.

- In a similar fashion to splitting the models more than once, when sending serialized weights OTA, one can also choose which weights to update, and this selection can be in no particular order. However, the architecture of the model remains static.

Based on the experiments of this chapter, transferring the serialized weights of a model seems to be the most effective way to perform updates OTA. The bandwidth required to send the model weights is measurably smaller than when the model is split into multiple sections and the performance of the larger split models is too slow for operational use (i.e., slower than real-time). While there is additional down-time required to recompile the models on the edge, this amount of time seems feasible based on the application and use-case. Finally, it should be noted that while the

experiments conducted here are promising, it is possible that a new task may not adapt as easily and with as few examples as those of the glider self-noise experiments. Brainstorming and initial research and development of an optional solution to such a problem has been considered, whereby, a network trained on the predictions being sent home from the autonomous system is used as a filter. On an ocean glider where bandwidth is expensive, such a solution does not necessarily minimize the financial cost accrued though sending home incorrect predictions, however, it may lessen the workload required by a manual analyst.

Chapter 6

Conclusion and Future Work

This thesis has outlined the process of research and development for DL-based passive acoustic monitoring (PAM) detection and classification systems (DCS). A review of deep learning models focused on two subsets: convolutional neural networks (CNNs) commonly used for image classification and single/two-stage detection models used in object detection. These models formed the foundation of the DL-based DCS implemented in this work. The significance of using spectrograms as a visual representation of acoustic data in PAM was justified, showcasing their utility in the context of the developed DCS. Moreover, performance metrics for evaluating DL-based DCS were introduced and provided a means to assess the efficacy of the aforementioned systems.

This work explored related research, spanning from traditional feature engineering coupled with machine learning (ML) algorithms as well as more recent applications using deep learning for PAM. In Chapter 3, details were presented surrounding the development process of supervised learning strategies, beginning with a CNN designed to classify spectrograms potentially containing vocalizations of endangered baleen whales. Subsequently, a novel approach was proposed, using R-CNNs to detect individual vocalizations within a spectrogram, emphasizing the distinction between spectrogram classification and vocalization detection. The limitations of supervised learning in DCS development were outlined in Chapter 4, leading to the exploration of an alternative approach under the ML paradigm known as "semi-supervised learning". This method involved utilizing unlabeled data alongside annotated examples to build more generalizable classification systems. Finally, Chapter 5 presented the development of an operational marine mammal DCS, addressing real-world constraints often overlooked in research environments. These constraints included handling variance and bias in PAM data sets, considering computational limitations, power efficiency, and enabling model updates on autonomous platforms.

6.1 Keeping Up with the Evolution of Machine Learning

Keeping up with the current state-of-the-art (SOTA) in machine learning can be an exciting yet somewhat arduous task, as new technologies and resources continue to be introduced and others improved upon each day. This fact is especially true for neural networks and deep learning. Staying at the forefront of research and development in DL can be broken down into several categories: the algorithms and/or architectures used in reaching SOTA performance, the hardware and software used during development, and the training and evaluation data sets. For an operational system, additional considerations must be made, as new technologies and algorithms are often unproven in their reliability beyond an academic paper.

6.1.1 Algorithms and Architectures

Throughout the latter portions of this work, R&D of an operational DCS was approached by considering both the current SOTA as well as the proven reliability of past methods. Specifically, the use of CNN-based detection models was used due to their proven ability across various domains. In recent years, the SOTA methods for tasks such as image classification have made use of the attention mechanism [121] by way of Transformer architectures [25]; but the necessity of lightweight computation inhibited additional research in this area. The use of attention mechanisms may prove especially useful in the suggested future work described later in this chapter related to including contextual information to improve DCS reliability.

6.1.2 Technological Tools and Resources

The tools and resources for developing neural networks can be separated into software and hardware. On the hardware side, SOTA algorithms and deep learning architectures often originate from large technology companies and AI research labs with seemingly no budgetary limitations on computing resources. The act of simply training many SOTA methods introduced more recently, such as large Transformer models, is likely infeasible at smaller research groups with limited computational power. Due to the highly iterative and experimental nature of deep learning research,

an inability to run hundreds of experiments simultaneously can be a bottleneck towards adopting the current SOTA methods. The neural network architectures chosen for this work, while not on the bleeding edge, are well-proven and have associated best practices for training and testing which allowed for more impactful experiments using the hardware available.

On the software side, while there are many frameworks and/or libraries available for ML research, not committing to a single library may be beneficial in order to keep up with the pace of innovation and maximize productivity. For example, much of the research in this work was developed using the Python ML framework PyTorch and while experimenting with porting neural networks to Java via an ONNX runtime, it was observed that the models produced would not run sufficiently fast enough to be considered operational. During this time, a Java runtime for models developed in Tensorflow was indeed available and it is possible that the inference speeds experienced using ONNX could have been improved, however, the process of converting and/or retraining these models was out of scope. In a similar experience, mid-way through developing the operational detection models of Chapter 5, the Tensorflow Object Detection API was deprecated in favour of another tool also being maintained by Google. While the API can still be used, updated models and compatibility with newer versions of Tensorflow will not continue.

Beyond the risk of deprecation, another reason to avoid committing to a single ML framework or software tool is that much of the heavy lifting in terms of model implementation has already been done. Many ML academic papers include links to implementations of their algorithms on GitHub, Huggingface, or similar websites. By becoming proficient in multiple libraries/frameworks, it is easier to stay adaptable to the latest developments, iterate, and runs experiments. Despite these advantages, it can be challenging to learn and use multiple libraries simultaneously. Therefore, it is also advised to design stages of a ML pipeline such that they are adaptable and separable (e.g., the pipeline outlined in Section 5.2).

6.2 Future Work: Spatio-temporal Contextual Awareness

With all of their successes, both machine and deep learning approaches to DCS development–as they currently stand–largely do not make use of important contextual data that can further inform the system. The most widely used deep learning approach to DCS development is to train models on slices of spectrograms possibly containing one or more marine mammal vocalization [56, 105, 117]. When applied to an entire acoustic recording in a sliding window fashion, such models disregard the previous predictions in the immediate vicinity. For some species, such as sei whales which often vocalize in doubles or triples, short-term temporal context such as whether-or-not the previous prediction indicated that a sei whale vocalization was present can be useful in sharpening the current prediction. An additional example of when contextual information may prove useful is when particularly vocal species (e.g., humpback whales) are present in an area of interest, and whose vocalizations may at times resemble those of other less-vocal species. Not only can these types of temporal context be used to inform the models predictions, but also assist in estimating the number of individual whales present in the recorded area. Such an improvement would lend itself to more robust density estimations. Additionally, many species of marine mammals are capable of making a variety of different vocalizations that vary in bandwidth and duration. Developing a DCS that classifies vocalizations at the call type level (as opposed to species level) may enhance individual detection performance as well as better inform a contextually aware system.

Longer-term temporal context, such as the time of year of the recordings, can also inform the model as most species of baleen whales follow predictable migratory patterns. In addition to temporal data, spatial context such as the location of the deployment, the depth of the recording device, and the estimated detection range are important factors that are largely ignored by current DL-based DCS.

Naively introducing contextual awareness through the application of binary rules (e.g., only one sei whale vocalization was detected so it is a false positive) may lead to decreased false positives, however, would likely negatively impact the true positive rate. One possible solution, would be to couple the predictions of a DCS similar to that in this work with a model that is more accustomed to sequential data (e.g., a recurrent neural network (RNN)) as demonstrated by Madhusudhana et al [75].

Unfortunately, introducing a second deep learning algorithm would likely negate any progress being made in terms of model efficiency. The same can be said regarding model boosting or ensembles of models. Outside of the realm of deep learning, one approach would be to use the predictions of the DCS as a prior distribution to a lightweight Bayesian model, for example a Gaussian process [91].

Research targeted at the tasks of video classification [14] and human action recognition within videos [54] may be easily adapted for contextual awareness. Researchers have suggested the use of 3-dimensional convolutions [49, 53]–as opposed to the 2-dimensional operation used in this work–in order to learn temporal context. Non-local neural networks [124] have been proposed in order to capture both short-term and long-term temporal dependencies between individual frames of videos [15, 28, 124]. However, while these approaches are useful for encoding temporal context, they lack support for the type spatial information we are also interested in.

Perhaps the most likely candidate for introducing contextual information directly into a DCS is by way of the attention mechanism. In doing so, a detection model may include contextual information both at a high (e.g., long-term temporal) and low (e.g., pixel-space) levels. Attention mechanisms have been reported for the task of speech recognition as far back as 2015 [20] and subsequently used for acoustic scene classification in 2018 [96]. Perhaps the most promising example in relation to this work is Context R-CNN [6] which has proven to be successful at improving the performance of detection models used to predict the presence of species in natural environments using camera traps.

The PAM community has expressed a great deal of interest in the development of a system described above that effectively incorporates contextual information, as it holds great potential for significantly improving the accuracy and robustness of marine mammal detection and classification. Further research in this area is a highly encouraged and and would be well-received.

Bibliography

[1] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[2] Ann N Allen, Matt Harvey, Lauren Harrell, Aren Jansen, Karlina P Merkens, Carrie C Wall, Julie Cattiau, and Erin M Oleson. A convolutional neural network for automated detection of humpback whale song in a diverse, long-term passive acoustic dataset. *Frontiers in Marine Science*, 8:607321, 2021.

[3] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *Neural Networks*, 138:14–32, 2021.

[4] Mohammed Bahoura and Yvan Simard. Blue whale calls classification using short-time fourier and wavelet packet transforms and artificial neural network. *Digital Signal Processing*, 20(4):1256–1263, 2010.

[5] Mark F Baumgartner and Sarah E Mussoline. A generalized baleen whale call detection and classification system. *The Journal of the Acoustical Society of America*, 129(5):2889–2902, 2011.

[6] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Context r-cnn: Long term temporal context for per-camera object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13075–13085, 2020.

[7] Yoshua Bengio, Frédéric Bastien, Arnaud Bergeron, Nicolas Boulanger-Lewandowski, Thomas Breuel, Youssouf Chherawala, Moustapha Cisse, Myriam Côté, Dumitru Erhan, Jeremy Eustache, et al. Deep learners benefit more from out-of-distribution examples. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 164–172, 2011.

[8] Christian Bergler, Manuel Schmitt, Rachael Xi Cheng, Hendrik Schröter, Andreas Maier, Volker Barth, Michael Weber, and Elmar Nöth. Deep representation learning for orca call type classification. In *Text, Speech, and Dialogue: 22nd International Conference, TSD 2019, Ljubljana, Slovenia, September 11–13, 2019, Proceedings 22*, pages 274–286. Springer, 2019.

[9] Christian Bergler, Hendrik Schröter, Rachael Xi Cheng, Volker Barth, Michael Weber, Elmar Nöth, Heribert Hofer, and Andreas Maier. Orca-spot: An automatic killer whale sound detection toolkit using deep learning. *Scientific reports*, 9(1):10997, 2019.

[10] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *Advances in Neural Information Processing Systems*, pages 5049–5059, 2019.

[11] Paul Best, Maxence Ferrari, Marion Poupard, Sébastien Paris, Ricard Marxer, Helena Symonds, Paul Spong, and Hervé Glotin. Deep learning and domain transfer for orca vocalization detection. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2020.

[12] Michael J Bianco, Peter Gerstoft, James Traer, Emma Ozanich, Marie A Roch, Sharon Gannot, and Charles-Alban Deledalle. Machine learning in acoustics: Theory and applications. *The Journal of the Acoustical Society of America*, 146(5):3590–3628, 2019.

[13] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[14] Darin Brezeale and Diane J Cook. Automatic video classification: A survey of the literature. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(3):416–430, 2008.

[15] Yue Cao, Jiarui Xu, Stephen Lin, Fangyun Wei, and Han Hu. Gcnet: Non-local networks meet squeeze-excitation networks and beyond. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019.

[16] Olivier Chapelle, Jason Weston, Léon Bottou, and Vladimir Vapnik. Vicinal risk minimization. In *Advances in neural information processing systems*, pages 416–422, 2001.

[17] Keunwoo Choi, George Fazekas, and Mark Sandler. Automatic tagging using deep convolutional neural networks. *arXiv preprint arXiv:1606.00298*, 2016.

[18] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2392–2396. IEEE, 2017.

[19] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[20] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. *Advances in neural information processing systems*, 28, 2015.

[21] Christopher W Clark, Peter Marler, and Kim Beeman. Quantitative analysis of animal vocal phonology: an application to swamp sparrow song. *Ethology*, 76(2):101–115, 1987.

[22] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[23] Kimberley TA Davies and Sean W Brillant. Mass human-caused mortality spurs federal action to protect endangered north atlantic right whales in canada. *Marine Policy*, 104:157–162, 2019.

[24] Ltsc Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael L Seltzer, Geoffrey Zweig, Xiaodong He, Jason D Williams, et al. Recent advances in deep learning for speech research at microsoft. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 26, page 64. IEEE, 2013.

[25] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[26] Peter J Dugan, Aaron N Rice, Ildar R Urazghildiiev, and Christopher W Clark. North atlantic right whale acoustic signal processing: Part i. comparison of machine learning recognition algorithms. In *2010 IEEE Long Island Systems, Applications and Technology Conference*, pages 1–6. IEEE, 2010.

[27] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

[28] Christoph Feichtenhofer, Haoqi Fan, Jitendra Malik, and Kaiming He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6202–6211, 2019.

[29] Fabio Frazao, Bruno Padovese, and Oliver S Kirsebom. Workshop report: Detection and classification in marine bioacoustics with deep learning. *arXiv preprint arXiv:2002.08249*, 2020.

[30] Rory Gibb, Ella Browning, Paul Glover-Kapfer, and Kate E Jones. Emerging opportunities and challenges for passive acoustics in ecological assessment and monitoring. *Methods in Ecology and Evolution*, 10(2):169–185, 2019.

[31] Douglas Gillespie, Marjolaine Caillat, Jonathan Gordon, and Paul White. Automatic detection and classification of odontocete whistles. *The Journal of the Acoustical Society of America*, 134(3):2427–2437, 2013.

[32] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[33] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[34] Benjamin S Halpern, Shaun Walbridge, Kimberly A Selkoe, Carrie V Kappel, Fiorenza Micheli, Caterina D'Agrosa, John F Bruno, Kenneth S Casey, Colin Ebert, Helen E Fox, et al. A global map of human impact on marine ecosystems. *science*, 319(5865):948–952, 2008.

[35] Sheryl Hamilton and G Barry Baker. Technical mitigation to reduce marine mammal bycatch and entanglement in commercial fishing gear: lessons learnt and future directions. *Reviews in Fish Biology and Fisheries*, 29(2):223–247, 2019.

[36] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015.

[38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[39] John A Hildebrand, Kaitlin E Frasier, Tyler A Helble, and Marie A Roch. Performance metrics for marine mammal signal detection and classification. *The Journal of the Acoustical Society of America*, 151(1):414–427, 2022.

[40] Ove Hoegh-Guldberg and John F Bruno. The impact of climate change on the world's marine ecosystems. *Science*, 328(5985):1523–1528, 2010.

[41] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[42] Eric J Humphrey and Juan Pablo Bello. Rethinking automatic chord recognition with convolutional neural networks. In *11th International Conference on Machine Learning and Applications (ICMLA)*, volume 2, pages 357–362. IEEE, 2012.

[43] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[44] Ali K Ibrahim, Hanqi Zhuang, Laurent M Chérubin, Nurgun Erdol, Gregory O'Corry-Crowe, and Ali Muhamed Ali. A multimodel deep learning algorithm to detect north atlantic right whale up-calls. *The Journal of the Acoustical Society of America*, 150(2):1264–1272, 2021.

[45] Shiklomanov Igor. World fresh water resources. *Water in crisis: a guide to the world's. Oxford University Press, Inc, Oxford*, 1993.

[46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[47] Ryo Ito, Ken Nakae, Junichi Hata, Hideyuki Okano, and Shin Ishii. Semi-supervised deep learning of brain tissue segmentation. *Neural Networks*, 116:25–34, 2019.

[48] Aleria S Jensen, Gregory K Silber, Connie Ewald Akamine, Dave Flannagan, John Ford, Pat Gerrior, Joseph Green, Frances Gulland, Diana Gutierrez, Michael Henshaw, et al. Large whale ship strike database. *NOAA Technical Memorandum NMFS-OPR*, 2004.

[49] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[50] Jia-jia Jiang, Ling-ran Bu, Fa-jie Duan, Xian-quan Wang, Wei Liu, Zhong-bo Sun, and Chun-yue Li. Whistle detection and classification for whales based on convolutional neural networks. *Applied Acoustics*, 150:169–178, 2019.

[51] G Jocher. Yolov5 by ultralytics (version 7.0)[computer software], 2020.

[52] Nicola Jones. Ocean uproar: saving marine life from a barrage of noise. *Nature*, 568:158–161, 04 2019.

[53] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[54] Eunju Kim, Sumi Helal, and Diane Cook. Human activity recognition and pattern discovery. *IEEE pervasive computing*, 9(1):48–53, 2009.

[55] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] Oliver S Kirsebom, Fabio Frazao, Yvan Simard, Nathalie Roy, Stan Matwin, and Samuel Giard. Performance of a deep neural network at detecting north atlantic right whale upcalls. *The Journal of the Acoustical Society of America*, 147(4):2636–2646, 2020.

[57] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better imagenet models transfer better? In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2661–2671, 2019.

[58] Katie A Kowarski, Julien J-Y Delarue, Briand J Gaudet, and S Bruce Martin. Automatic data selection for validation: A method to determine cetacean occurrence in large acoustic data sets. *JASA Express Letters*, 1(5):051201, 2021.

[59] Katie A Kowarski and Hilary Moors-Murphy. A review of big data analysis methods for baleen whale passive acoustic monitoring. *Marine Mammal Science*, 2020.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, page 1097–1105, Red Hook, NY, USA, 2012. Curran Associates Inc.

[61] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[62] David W Laist, Amy R Knowlton, James G Mead, Anne S Collet, and Michela Podesta. Collisions between ships and whales. *Marine Mammal Science*, 17(1):35–75, 2001.

[63] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[64] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[65] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[66] Yann LeCun et al. Generalization and network design strategies. In *Connectionism in perspective*, volume 19. Citeseer, 1989.

[67] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[68] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, 2021.

[69] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[70] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[71] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.

[72] Songzuo Liu, Meng Liu, Mengjia Wang, Tianlong Ma, and Xin Qing. Classification of cetacean whistles based on convolutional neural network. In *10th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–5. IEEE, 2018.

[73] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[74] Wenyu Luo, Wuyi Yang, and Yu Zhang. Convolutional neural network for detecting odontocete echolocation clicks. *The Journal of the Acoustical Society of America*, 145(1):EL7–EL12, 2019.

[75] Shyam Madhusudhana, Yu Shiu, Holger Klinck, Erica Fleishman, Xiaobai Liu, Eva-Marie Nosal, Tyler Helble, Danielle Cholewiak, Douglas Gillespie, Ana Širović, et al. Improve automatic detection of animal call sequences with temporal context. *Journal of the Royal Society Interface*, 18(180):20210297, 2021.

[76] Tiago A Marques, Len Thomas, Stephen W Martin, David K Mellinger, Jessica A Ward, David J Moretti, Danielle Harris, and Peter L Tyack. Estimating animal population density using passive acoustics. *Biological Reviews*, 88(2):287–309, 2013.

[77] Bruce Martin, Katie Kowarski, Xavier Mouy, and Hilary Moors-Murphy. Recording and identification of marine mammal vocalizations on the scotian shelf and slope. In *2014 Oceans-St. John's*, pages 1–6. IEEE, 2014.

[78] S Bruce Martin, Corey Morris, Koen Bröker, and Caitlin O'Neill. Sound exposure level as a metric for analyzing and managing underwater soundscapes. *The Journal of the Acoustical Society of America*, 146(1):135–149, 2019.

[79] David K Mellinger. A comparison of methods for detecting right whale calls. *Canadian Acoustics*, 32(2):55–65, 2004.

[80] David K Mellinger, Carol D Carson, and Christopher W Clark. Characteristics of minke whale (balaenoptera acutorostrata) pulse trains recorded near puerto rico. *Marine Mammal Science*, 16(4):739–756, 2000.

[81] David K Mellinger, Stephen W Martin, Ronald P Morrissey, Len Thomas, and James J Yosco. A method for detecting whistles, moans, and other frequency contour sounds. *The Journal of the Acoustical Society of America*, 129(6):4055–4061, 2011.

[82] David K Mellinger, Kathleen M Stafford, Sue E Moore, Robert P Dziak, and Haru Matsumoto. An overview of fixed passive acoustic observation methods for cetaceans. *Oceanography*, 20(4):36–45, 2007.

[83] October 29 Monday and Deep LearningEnvironmentMachine HearingMachine Perception. Acoustic detection of humpback whales using a convolutional neural network.

[84] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[85] Radford M Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[86] Douglas P Nowacek, Lesley H Thorne, David W Johnston, and Peter L Tyack. Responses of cetaceans to anthropogenic noise. *Mammal Review*, 37(2):81–115, 2007.

[87] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[88] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[89] Sai Samarth R Phaye, Emmanouil Benetos, and Ye Wang. Subspectralnet-using sub-spectrogram based convolutional neural networks for acoustic scene classification. *arXiv preprint arXiv:1810.12642*, 2018.

[90] Karol J Piczak. Environmental sound classification with convolutional neural networks. In *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2015.

[91] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer school on machine learning*, pages 63–71. Springer, 2003.

[92] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[93] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[94] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[95] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE transactions on pattern analysis and machine intelligence*, 39(6):1137–1149, 2016.

[96] Zhao Ren, Qiuqiang Kong, Kun Qian, Mark D Plumbley, and Björn W Schuller. Attention-based convolutional neural networks for acoustic scene classification. In *Scenes and Events 2018 Workshop (DCASE2018)*, page 39, 2018.

[97] Denise Risch, Christopher W Clark, Peter J Dugan, Marian Popescu, Ursula Siebert, and Sofie M Van Parijs. Minke whale acoustic behavior and multi-year seasonal and diel vocalization patterns in massachusetts bay, usa. *Marine Ecology Progress Series*, 489:279–295, 2013.

[98] Denise Risch, Thomas Norris, Matthew Curnock, and Ari Friedlaender. Common and antarctic minke whales: Conservation status and future research directions. *Frontiers in Marine Science*, 6:247, 2019.

[99] Marie A Roch, Holger Klinck, Simone Baumann-Pickering, David K Mellinger, Simon Qui, Melissa S Soldevilla, and John A Hildebrand. Classification of echolocation clicks from odontocetes in the southern california bight. *The Journal of the Acoustical Society of America*, 129(1):467–475, 2011.

[100] Marie A Roch, Melissa S Soldevilla, Jessica C Burtenshaw, E Elizabeth Henderson, and John A Hildebrand. Gaussian mixture model classification of odontocetes in the southern california bight and the gulf of california. *The Journal of the Acoustical Society of America*, 121(3):1737–1748, 2007.

[101] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[102] Justin Salamon and Juan Pablo Bello. Deep convolutional neural networks and data augmentation for environmental sound classification. *IEEE Signal Processing Letters*, 24(3):279–283, 8 2016.

[103] Cullen Schaffer. Overfitting avoidance as bias. *Machine learning*, 10(2):153–178, 1993.

[104] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

[105] Yu Shiu, KJ Palmer, Marie A Roch, Erica Fleishman, Xiaobai Liu, Eva-Marie Nosal, Tyler Helble, Danielle Cholewiak, Douglas Gillespie, and Holger Klinck. Deep neural networks for automated detection of marine mammal species. *Scientific Reports*, 10(1):1–12, 2020.

[106] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[107] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[108] Mark D Skowronski and John G Harris. Acoustic detection and classification of microchiroptera using machine learning: lessons learned from automatic speech recognition. *The Journal of the Acoustical Society of America*, 119(3):1817–1833, 2006.

[109] Brandon L Southall, James J Finneran, Colleen Reichmuth, Paul E Nachtigall, Darlene R Ketten, Ann E Bowles, William T Ellison, Douglas P Nowacek, and Peter L Tyack. Marine mammal noise exposure criteria: Updated scientific recommendations for residual hearing effects. *Aquatic Mammals*, 45(2):125–232, 2019.

[110] Dan Stowell. Computational bioacoustics with deep learning: a review and roadmap. *PeerJ*, 10:e13152, 2022.

[111] Farhana Sultana, Abu Sufian, and Paramartha Dutta. Evolution of image segmentation using deep convolutional neural network: a survey. *Knowledge-Based Systems*, 201:106062, 2020.

[112] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[113] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[114] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[115] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114, 2019.

[116] Mingxing Tan, Ruoming Pang, and Quoc V Le. Efficientdet: Scalable and efficient object detection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10781–10790, 2020.

[117] Mark Thomas, Bruce Martin, Katie Kowarski, Briand Gaudet, and Stan Matwin. Marine mammal species classification using convolutional neural networks and a novel acoustic representation. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 290–305. Springer, 2019.

[118] Devis Tuia, Benjamin Kellenberger, Sara Beery, Blair R Costelloe, Silvia Zuffi, Benjamin Risse, Alexander Mathis, Mackenzie W Mathis, Frank van Langevelde, Tilo Burghardt, et al. Perspectives in machine learning for wildlife conservation. *Nature communications*, 13(1):792, 2022.

[119] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[120] Ayinde M Usman, Olayinka O Ogundile, and Daniel JJ Versfeld. Review of automatic detection and classification techniques for cetacean vocalization. *IEEE Access*, 8:105181–105206, 2020.

[121] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[122] William Vickers, Ben Milner, Artjoms Gorpincenko, and R Lee. Methods to improve the robustness of right whale detection using cnns in changing conditions. In *2020 28th European Signal Processing Conference (EUSIPCO)*, pages 106–110. IEEE, 2021.

[123] William Vickers, Ben Milner, Denise Risch, and Robert Lee. Robust north atlantic right whale detection using deep learning models for denoising. *The Journal of the Acoustical Society of America*, 149(6):3797–3812, 2021.

[124] Xiaolong Wang, Ross Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7794–7803, 2018.

[125] Lindy S Weilgart. The impacts of anthropogenic ocean noise on cetaceans and implications for management. *Canadian journal of zoology*, 85(11):1091–1116, 2007.

[126] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine learning*, 23(1):69–101, 1996.

[127] Sean M Wiggins and John A Hildebrand. Long-term monitoring of cetaceans using autonomous acoustic recording packages. In *Listening in the Ocean*, pages 35–59. Springer, 2016.

[128] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[129] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

[130] Peilin Zhao, Steven CH Hoi, Jialei Wang, and Bin Li. Online transfer learning. *Artificial intelligence*, 216:76–102, 2014.

[131] Zhaohui Zheng, Ping Wang, Dongwei Ren, Wei Liu, Rongguang Ye, Qinghua Hu, and Wangmeng Zuo. Enhancing geometric factors in model learning and inference for object detection and instance segmentation. *IEEE Transactions on Cybernetics*, 52(8):8574–8586, 2021.

[132] Ming Zhong, Manuel Castellote, Rahul Dodhia, Juan Lavista Ferres, Mandy Keogh, and Arial Brewer. Beluga whale acoustic signal classification using deep learning neural network models. *The Journal of the Acoustical Society of America*, 147(3):1834–1841, 2020.

[133] Ming Zhong, Maelle Torterotot, Trevor A Branch, Kathleen M Stafford, Jean-Yves Royer, Rahul Dodhia, and Juan Lavista Ferres. Detecting, classifying, and counting blue whale calls with siamese neural networks. *The Journal of the Acoustical Society of America*, 149(5):3086–3094, 2021.

[134] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

[135] Walter MX Zimmer. *Passive acoustic monitoring of cetaceans*. Cambridge University Press, 2011.