Neural Networks in Application to Dolphin Whistle Detection and Generation

by

Xi Lu

BEng, Royal Military College of Canada, 2021

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Applied Science

in

THE FACULTY OF GRADUATE AND POSTDOCTORAL STUDIES
(Electrical and Computer Engineering)

The University of British Columbia
(Vancouver)

December 2023

The following individuals certify that they have read, and recommend to the Faculty of Graduate and Postdoctoral Studies for acceptance, the thesis entitled:

Neural Networks in Application to Dolphin Whistle Detection and Generation

submitted by **Xi Lu** in partial fulfillment of the requirements for the degree of **Master of Applied Science** in **Electrical and Computer Engineering**.

Examining Committee:

Lutz Lampe, Professor, Electrical and Computer Engineering, UBC
Supervisor

Matthew Yedlin, Associate Professor, Electrical and Computer Engineering, UBC
Supervisory Committee Member

Purang Abolmaesumi, Professor, Electrical and Computer Engineering, UBC
Supervisory Committee Member

Abstract

In the underwater realm, marine mammals rely heavily on acoustic signals for their communications. Being able to accurately and easily detect these signals can aid in studying factors such as creature presence and migratory habits. Additionally, recent research in the field of covert Underwater Acoustic Communications (UWAC) has begun to incorporate marine mammal signals to utilise naturally-occurring sounds. Compared to traditional methods, marine mammal signals allow transmissions to occur at higher power levels, increasing the range of covert communications. Commonly, there are two categories of acoustic marine mammal signals: clicks and whistles. Both can be detected by converting audio waveforms to spectrogram images, and the detection process is often done visually by human experts. This image data type is particularly important for whistle detection, as these tend to be lower power and have a narrow-band, time-varying frequency profile. Given the potential uses of these signals, the creation of accurate, consistent automated detection methods has been an active area of research.

This thesis investigates the utility of Neural Networks (NNS) in application to dolphin whistle detection and generation. We seek to provide a detection pipeline which is robust to changing environments and requires no context-specific work to be done. This is accomplished by performing minimal preprocessing on data and utilising transfer learning from a large dataset into a newer, smaller one. Using these techniques, we are able to achieve detection accuracy greater than 95% for our tested models. For whistle generation, we investigate two methods known as Generative Adversarial Networks (GANS) and Denoising Diffusion Probabilistic Models (DDPMS), the latter of which is found to be more effective. We separate the task of generating synthetic realistic whistles into two steps: contour and varia-

tions. The end result is a cascaded DDPM system which generates whistles following these two steps. We demonstrate an iterative detection application to assess the efficacy of this generative method, integrating our synthetic samples into the task of improving automated signal detection.

Lay Summary

Dolphin whistles are one example of the diverse acoustic signals that can be used to track presence and population of marine mammals. The detection of these signals is vital to research in the marine field, and the ability to automate this otherwise time-consuming process can be a boon to anybody wishing to develop a comprehensive database. Whistles tend to be assessed visually in their spectrogram form, and thus image classification neural networks can be levied to perform this task. We propose a data preprocessing method which allows for higher adaptability to different oceanic environments, enabling wider and easier automated detection. Additionally, we develop image generation networks to create original, realistic whistle samples. These generated whistles are used in an example application to train a detector in a novel oceanic environment with no previously tagged whistles.

Preface

This thesis is an original, independent work by the author Xi Lu. A version of chapter 2 was presented at OCEANS 2023, June 2023: "Transfer Learning of Image Classification Networks in Application to Dolphin Whistle Detection" [37] with co-authors Lutz Lampe, Burla Nur Korkmaz, Alberto Testolin, and Roee Diamant. I developed the research idea, performed the analysis and numerical tests, and wrote the manuscript. Dr. Lutz Lampe helped with research direction and manuscript preparation. Dr. Alberto Testolin and Burla Nur Korkmaz guided the research inspiration and provided their own results for comparison. Dr. Roee Diamant provided experimental data and research guidance.

All research was done under the supervision of Professor Lutz Lampe, who helped with developing concepts, formalising methods, and revising manuscripts.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

The following abbreviations and acronyms are used in this thesis:

Acc	Accuracy, percent describing proportion of samples classified correctly
CNN	Convolutional Neural Network
DCGAN	Deep Convolutional Generative Adversarial Network
DDPM	Denoising Diffusion Probabilistic Model
DFT	Discrete Fourier Transform
GAN	Generative Adversarial Network
FA	False Alarm, percent describing proportion of negative samples wrongly classified as positive
FN	False Negative
FP	False Positive
MD	Missed Detection, percent describing proportion of positive samples wrongly classified as negative
NN	Neural Network
PDF	Probability Density Function
SDetA	Synthetic Detection Accuracy, defined in Chapter 4
SDetPA	Synthetic Detection Positive Accuracy, defined in Chapter 4
SDisA	Synthetic Discrimination Accuracy, defined in Chapter 4
SDisE	Synthetic Discrimination Epochs, defined in Chapter 4
STFT	Short Time Fourier Transform

TN	True Negative
TP	True Positive
UWAC	Underwater Acoustic Communications

Acknowledgments

I would first offer my profound appreciation to my supervisor, Dr. Lutz Lampe, for his unrelenting encouragement and support in this endeavour. His assistance has bolstered me in my many times of confusion and made this work possible.

This work was also completed under the NATO Science for Peace and Security Programme grant G5293. A warm thanks to the project members for their suggestions and feedback throughout the process. Special recognition goes to Dr. Roee Diamant for provision of the datasets that are used throughout this thesis.

Many individuals were consulted through the writing of this thesis, providing invaluable suggestions or their own methods of performing related and relevant tasks. This includes but is but not limited to: Dr. Alberto Testolin and Burla Nur Korkmaz at the University of Padova (whistle detection), Dr. Renjie Liao at the University of British Columbia (generative models), Itamar Davidesco at the University of Haifa (whistle trace fitting), and Dr. Paolo Casari (Bellhop modelling) at the University of Trento.

Finally, I greatly appreciate my friends and family for their continuous support over my years of study. Even when the end was nowhere in sight, their belief in its existence kept me going.

Chapter 1

Introduction

The underwater world shows a breadth and depth of diversity that has yet to be fully explored and understood by humankind. In particular, marine mammals are known to communicate in a wide variety of conditions and manners, conveying complex information through their acoustic signals. These sounds can differ greatly depending on the species and context.

In the marine biology field, these signals indicate presence of creatures and can convey information about the population, which can be indicators of general health and migration patterns. Additionally, some scientists seek to understand the semantic and contextual meanings being conveyed, the why or what of these signals. Marine mammal acoustic signals can be also used in biomimicry schemes for covert Underwater Acoustic Communications (UWAC), which will be discussed in more depth in Section 3.1. The varied, multiple uses of these communications leads us to exploring methods with which they can be detected and generated.

1.1 Dolphin Signals

In this thesis, while odontocetes or marine mammals may be referenced in general, we focus on dolphins. Although the terminology for the types of signals they emit can differ, [31] reviews multiple sources and broadly classifies these many types into two categories that are also used in other literature: clicks and whistles. A visual depiction of some dolphin whistles can be found in Figure 1.1.

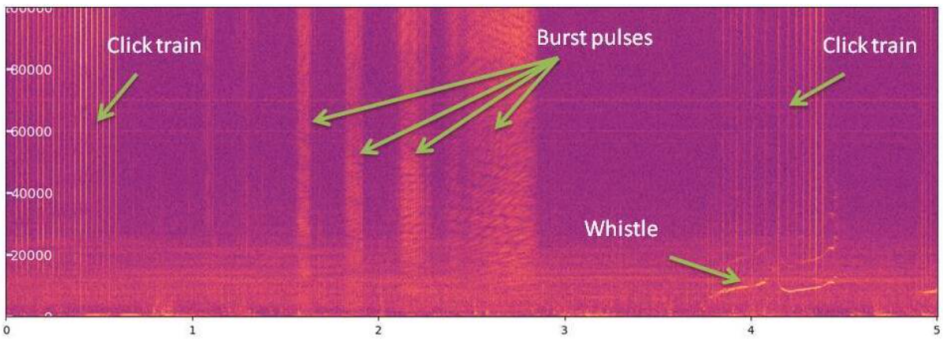


Figure 1.1: Labelled spectrogram with various types of signals from dolphins. Source: [12].

Clicks are higher energy and thus more distinct or obviously noticed in recordings. They can occur in a series of individual clicks, which is as a whole called a "click train". This type of signal is commonly used for echolocation, which aids in navigation under varying environmental conditions. [31] describes clicks as being broadband with "a sharp instantaneous front" as well as being short-duration, typically lasting between 0.1 s and 3.0 s. The assumed frequency range of clicks varies depending on the source. This range has widened over the years, and it is now known to extend past 200 kHz which is outside a dolphin's hearing range [31]. Even if it can be included in research, [31] states that a low-pass filter at 200 kHz will often be used. Visually, clicks occupy a short time-span but large frequency-span, appearing on spectrograms as vertical lines.

[31] states that whistles are created by pulses with high enough repetition rates to be heard as continuous sounds. Whistles are composed of a fundamental frequency which is concentrated in a narrow bandwidth, and this is transmitted alongside harmonics that occur at integer multiples. In the early decades, it was thought that the fundamental frequency was limited to a range of 7 kHz to 15 kHz. Sources such as [5] indicate that this frequency will rarely exceed 20 kHz. The upper range of this was expanded gradually as recording hardware improved and more research was conducted. [31] states that whistles have been seen to exceed 40 kHz. However, works in dolphin whistle detection rarely consider signals beyond 25 kHz, and they are sometimes more restrictive than that. This is often a limit imposed by recording quality, particularly when a mix of older and newer samples are used

and consistency is maintained between the datasets. The duration of whistles varies greatly but is typically approximately 0.5 s and less than 1.0 s [31]. While they are lower-energy signals than clicks, whistles have distinct time-frequency variations that create shapes known as "contours". These contours contain inflection points over their duration, and a single whistle can have as few as 0 or as many as 26 inflection points [31]. In spectrograms, whistles appear similar to chirp signals, with differing degrees of variability and shape. A subtype of whistle observed in bottlenose dolphins is known as a "signature whistle". These are understood to be used to convey identity and location information, and they are unique contours that dolphins develop in the early months of their lives [31]. In [11], over a hundred bottlenose dolphins were studied and it was found that signature whistles accounted for the majority of their whistle emissions.

1.2 Application of Neural Networks

To use or analyse marine mammal signals, they must first be detected. This can be a time-consuming process which requires manually sifting through hours upon hours of recordings that contain noise or confounding sounds. This is especially true for the lower-volume dolphin whistles that we focus on in this thesis. Although there are several different automated alternatives, as we will cover in Section 2.1, modern technology makes the application of Neural Networks (NNs) a very natural progression. Artificial intelligence can accomplish tasks previously considered to be inaccessible by computer programs or mathematical models, and dolphin whistle detection can be difficult using non-learning methods. Image classification networks are capable of handling extremely complicated tasks with great variance in the data, indicating that these may apply well to this detection task. Additionally, a well-trained NN has a degree of consistency, unlike multiple humans operators of differing skill levels working across different environments. This can be a huge benefit in creating a versatile tool for whistle detection that can be widely used. In effect, we seek to minimise the human effort needed in dolphin whistle detection. We will also be investigating the way that data creation and composition may affect this goal, elaborated further in Chapter 2.

Application of NNs to whistle detection may be hindered by insufficient data.

Image classification tasks tend to fare better when there are tens of thousands of varied samples, but most whistle databases have merely a few hundred or thousand signal samples. In the realm of NNs, transfer learning is a technique where previously learned knowledge is used for another task. It is particularly common in image classification tasks, as most models will learn basic patterns at initial layers that are later composed into more complex shapes. Inspired by [32], we utilise transfer learning from popular image classification models trained on the ImageNet dataset [16]. Since the model was trained on a large and diverse dataset previously, we use that as a starting point for the smaller and more similar data within our whistle spectrograms.

In addition to detection, the application of NNs to dolphin whistle generation is explored. The generated whistles are left in the image domain, and the focus is on generating synthetic samples that can pass for real dolphin whistles. Using a generative model can allow creation of contours not based on existing recordings or parameterised by equations. This allows generation of "original" data that can be used in a variety of ways. One application that we explore is in data augmentation, which can expedite the detection process when labelled data is sparsely available.

1.3 Contributions

In this thesis, we first demonstrate the efficacy of NNs in dolphin whistle detection on two separate datasets. Several models are used throughout, the majority of which are from popular image classification families. We also utilise a basic Convolutional Neural Network (CNN) [33] to provide demonstrate baseline NN capability. In line with the overarching goal to minimise human effort and involvement in the detection process, we develop a data preprocessing method that is bare and push the burden of detection as much onto the model's ability to learn as possible. This is compared to another preprocessing method adapted from [32], and its efficacy is proven in detection performance in a different environment. We utilise transfer learning from models trained on an ImageNet classification task as well as models trained on one dataset and then re-trained on the second. In conjunction with experiments on freezing model parameters and reducing the size of the training set, we demonstrate that this minimal preprocessing and transfer learning can

allow models to learn with sufficient accuracy and perform well across different environments.

Additionally, we investigate generative models and their ability to create original, realistic samples in the image domain. We divide this generation task into two steps: contour and variations. We start by training two styles of models – Generative Adversarial Networks (GANs) [44] and Denoising Diffusion Probabilistic Models (DDPMs) [25] – on whistle contours generated by polynomial fitting. DDPM was found to be a more effective model type for this task. Then, we develop an optimal set of variation parameters to create realistic-looking whistles. NN-based metrics for evaluating generative model performance are utilised to determine suitable parameters, and these are used to guide a second DDPM with the task of modifying a given contour to mimic what the parameters would do. The two DDPMs are sequentially used in a "cascaded" fashion to generate our synthetic samples. These are applied to the task of enhancing detection capability when in a new oceanic environment with no tagged samples.

1.4 Organisation

Chapter 2 outlines the application of NNs to detection of dolphin whistles. We start by exploring how the data is pre-processed and converted into a format usable for the NNs before reviewing the results. Additional results related to generalisability and parameter freezing are also explored. Chapter 3 describes the generation of dolphin whistle contours. The various models attempted are described and reviewed for effectiveness when trained to generate whistle contours. Chapter 4 demonstrates how generated whistle contours can be made to look realistic through variations in their appearance. We develop a set of parameters to control this, which are optimised using a set of metrics to assess the synthetic whistles' resemblance to real whistles. Additionally, we examine how this can be integrated into a cascaded DDPM system and pose an example for the utility of this feature. Finally Chapter 5 presents conclusions and outlines possible future work.

Chapter 2

Automated Detection

Any work investigating or utilising dolphin whistles first requires that a collection of these signals exists. While this can be done manually by human eyes and often is, reliably automating this process can be a boon, particularly in the day and age of massive amounts of data. This chapter covers the process through which NNs – in particular image classification networks – can be used to detect dolphin whistles. We start by reviewing existing methods of detection before discussing the datasets we use and the preprocessing conducted to create input data for the models. Then, we briefly outline how image classification networks function. The specific models used in this thesis are explained, and the results are presented. We then examine how the models are able to perform across different datasets to prove utility in the real world. Finally, we include expansions on these basic concepts with parameter freezing, dataset size decreasing, and a combination of the two.

2.1 Existing Methods

Attempts at automating marine mammal acoustic detection have been years in the making, targeted at a variety of animals and sound types. Gillespie [19] develops a method for detecting and classifying right whale calls based on their spectrograms. This starts by smoothing with a Gaussian kernel before an edge detection algorithm is used to detect the presence of these signals, and this method was able to detect 90% and 60% of calls identified by human operators in two different envi-

ronments with low False Alarm (FA) rates. Both [39] and [2] use a spectrogram correlation method applied to different whale species, which applies a base "truth" kernel to novel sounds and achieves a closeness score used to detect signals. These were applied to longer-duration signals – whale songs in this case – and achieved success rates of 97.5% and 74% with real-time functionality. In the realm of dolphin signals, [48] uses entropy as a detection measure for both clicks and whistles, theorising that the presence of these signals among the general ocean noise would increase the similarity between successive time samples. They achieve 96.6% and 97% accuracy; however, these reported results are skewed towards samples with non-presence of signals. [20] uses several algorithms to reduce noise in signals and applies amplitude thresholding before applying a region search for odontocete whistle detection and classification. They are able to achieve a 76% detection rate for human-labelled sounds, with 88% of all detections being valid.

There are also many detection methods that include NNs. In particular, image classification networks are often seen since spectrograms are particularly useful for lower-strength signals (i.e. whistles). Bergler et al. [6] develop a toolkit called ORCA-SPOT with a positive predictive value of 93.2% using state-of-the art CNNs and large amounts of data. [51] involves use of the chimp optimisation algorithm on dolphin whistle detection, achieving accuracy up to 95.45%. Utilising one of the same datasets as in this thesis, [32] tested a well-known image classification network on its ability to identify dolphin whistle signals with the additional benefit of transfer learning, achieving 98.9% accuracy.

Oftentimes, the work goes beyond the task of detection. A typical task involves whistle contour extraction, as the time-frequency shape of the signals carry information. Sermani et al. [47] apply independent component analysis followed by wavelet denoising to separate bottlenose dolphin emissions from other underwater noise in an effort to extract the signals. In [22], the authors utilise a Monte-Carlo probabilistic view and require a set of measurements for every moment in time to reduce background noise. [45] demonstrates two methods, one using Bayesian filtering to estimate contours from spectral peaks while the other uses adaptive polynomial prediction to connect peaks in graphs.

Our work in this chapter – particularly Section 2.2.2 – is motivated primarily by that found in [32], as well as providing an extension on and reproduction of

their methods. Additionally, we seek to provide an alternative to the preprocessing and denoising processes discussed above. The aforementioned detection and classification methods found in literature near-universally include several preprocessing steps, typically aimed at denoising and thus enhancing the relevant signal. We theorise that this limits a detection method's ability to easily be used in different environments as the process is developed with one dataset in mind and could potentially result in lower-quality samples elsewhere. Thus, we intend to use as little preprocessing as possible, performing only what is required to convert our data from audio to image.

2.2 Image Classification Networks

This section starts by reviewing the theory behind image classification networks, including the types of layers utilised within them. Then, we provide an outline of the model architectures used in this thesis.

2.2.1 Overview

NNs began as models with few layers, but they have long since grown into complicated structures. This is commonly indicated with the word "deep" [24] nowadays, a reference to the many-layered architectures. In this thesis, we will continue to use the term NN. Initial NN nodes were what is now termed "linear" or "dense". Every dense node receives all of the previous layer's outputs as input, and these are combined in a linear equation where the weights for each input are the learnt parameters for each node. This linear combination is passed through a non-linear activation function, which theoretically allows a model to approximate any function. The expression for a single dense node is given by

$$y = \sigma\left(\mathbf{w}^T \mathbf{x}\right), \tag{2.1}$$

where y is the node's output, $\mathbf{x}$ is the node's inputs from the previous layer as a vector, $\mathbf{w}$ represents the learned weights as a vector, and σ is the activation function. A constant term is often included within the linear combination as well but is not explicitly notated.

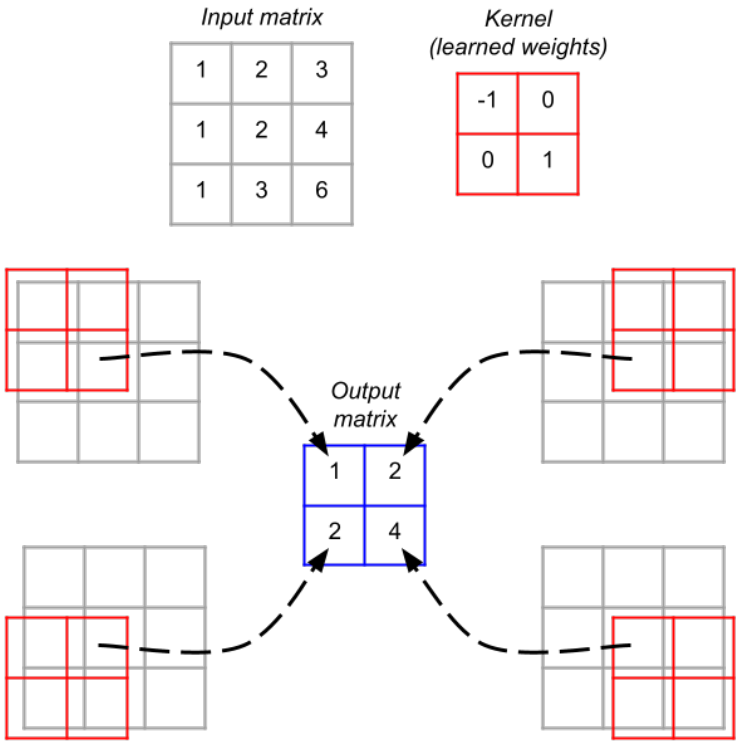


Figure 2.1: Convolution result of a 3×3 input matrix with a 2×2 kernel resulting in a 2×2 output matrix. This example uses a stride of 1 and padding of 0.

The archetypal and generally successful formula for networks utilising image data relies on the use of convolution operations. An example of this process can be seen in Figure 2.1.

Rather than producing a single value, each node produces a matrix whose size will depend on the convolutional parameters. The weights in a node represent a kernel which uses a spatially-correlated subset of the inputs, and this kernel "slides" over the entire input matrix to obtain the outputs. It is also possible to add constant terms after the convolution is performed. Elements of the output matrix are individually passed through a non-linear activation function, and this results in the final output of a convolutional node. A convolution operation allows information to be collated over a local area, thereby maintaining the spatial relationships be-

tween input data points that are inherent to an image. Factors such as kernel size (i.e. number of weights in the node), stride (i.e. how many pixels the kernel moves over), and padding (i.e. pixels used to increase the input image size) affect the output shape.

Typically, a CNN will begin with a certain number of convolutional layers. After, the output matrix will be flattened and dense layers will complete the model's architecture. While this is not necessary since a model can be entirely composed of convolutional layers, that is the style for NNs used in this thesis. One way to view this structure is that the convolutional layers are feature extractors while the dense layers learn how these features correspond to the desired model output. The final output of the CNN depends on its purpose. Often, the task at hand is image classification and thus the model outputs a vector of probabilities corresponding to the possible classes. For our purpose, we have created a single value output for a detector that indicates the probability of the input image containing a whistle.

Another layer seen in many image classification networks is a "pooling" layer. Rather than using a learned kernel, a fixed operation will be performed; this is commonly "max pooling", which outputs the maximum value within the kernel's reach. This is generally done to reduce spatial dimensions while emphasizing the important values within the inputted matrix.

Finally, dropout layers are also used. During training, they mask out certain nodes with a chosen probability, which prevents models from learning to rely on only specific elements in their architecture. When deployed, this probability becomes a multiplier across the entire set of inputs.

2.2.2 Neural Networks Utilised

As a benchmark of NN capability, we developed a basic CNN composed of the standard layers mentioned in Section 2.2.1. The architecture was selected over a series of experiments to determine a suitable structure that could sufficiently perform the task at hand while using relatively few parameters. This model (Simple) was created with the goal of being lightweight and to test how well a basic CNN could perform at the detection task. It does not have the benefit of any pre-training. The model first utilises five sets of convolution-dropout-pooling layers followed by

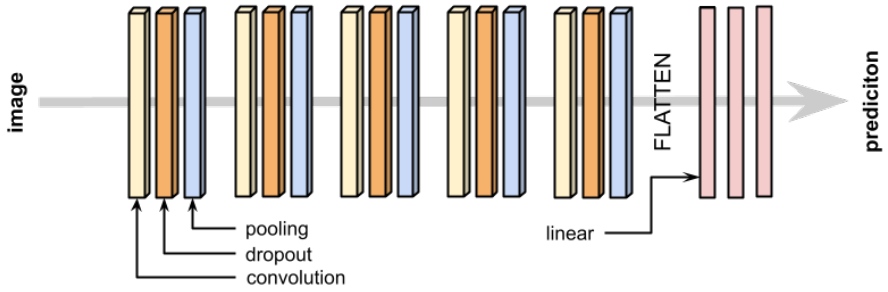


Figure 2.2: Architecture of Simple network.

three dense layers. This is visualised in Figure 2.2.

Pre-trained models are taken from one of three well-known series: VGGNet [49] [35] (VGG), ResNet [23] (Res), and DenseNet [27] (Dense). The VGGNet series began with a desire to create deeper convolution networks. These models were of comparable size to other existing, non-convolutional architectures, made possible by using small kernel sizes [49]. Later, in [35], they explored the benefits of techniques such as batch normalisation and increased dropout to reduce overfitting. While a deeper network has more capacity, it is increasingly vulnerable to problems such as vanishing gradients which cause difficulties in training and poor performance. In [23], the ResNet series sought to address this primarily through the use of residual connections. This allows "shortcuts" to be made, bringing earlier parts of a network directly to later nodes without passing through intermediate layers. Similarly, DenseNet accomplishes the same through even more connections, allowing every layer to be directly connected to each subsequent one.

As inspired by [32], using the pre-trained models is motivated by a desire to apply transfer learning to the detection task. The weights are taken from PyTorch [42] and are trained on the ImageNet-1k dataset. These pre-trained models are structured to produce 1000 outputs as class likelihoods, so each model's final layer is replaced to output only one value. In addition to choosing our own models, which will be explained in Section 2.4.1, we replicate the one produced in [32] using their preprocessing method. Their VGG16 model replaces the final dense layers with those of size 50, 20, and 1. Two versions were created, one with (VGG16d**) and one without (VGG16**) dropout layers between these dense layers since the

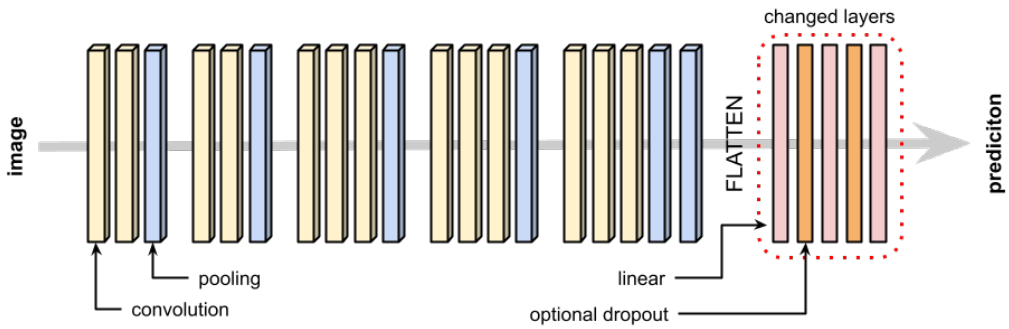


Figure 2.3: Architecture of VGG16**/VGG16d** network.

original architecture has these while [32] does not. The overall structure of these models can be found in Figure 2.3.

2.3 Data

This section describes the data that is used in this thesis. We first discuss the sources and amount of data available from each. Then, we cover how the audio data is converted into a usable form for our NNs. Finally, we explain the specific composition of data samples. We refer to samples as "positive" to indicate presence of a dolphin whistle signal and "negative" elsewise.

2.3.1 Sources

In this thesis, there are two datasets of dolphin whistles available. Dataset A contains recordings made at a depth of 50 m and a distance of 200 m away from bottlenose dolphin (tursiops truncatus) reefs in Eilat, Israel. They were recorded over 27 days in the summer of 2021 using two Geospectrum M18 hydrophones suspended 1.5 m above the seabed. Dataset A consists of 2-channel audio data which is tagged in an ongoing process by a human expert; the tagging information used in this thesis was updated in late 2022. In total, there are 5247 whistles, which also have time-frequency points along the contour notated. This dataset is also used in [32] and is available at [1].

Our second dataset (B) is from the 8th DCLDE workshop [15]. These files were recorded in the Western Atlantic Ocean and involve an unknown mixture of

dolphin species. Dataset B is single-channel with unknown recording conditions and hardware. The dataset consists of 1598 manually tagged whistles.

In both cases, far more non-whistle audio data exists, and a random subset was chosen to serve as negative samples. The negative subset has the same number of samples as the positive subset, which was done to avoid class imbalance. Both subsets were individually split into three portions: 80% training set, 10% test set, and 10% validation set. The training sets were seen and used by the models for backpropagation, and the validation sets were used to select the best model parameters to reduce overfitting. Unless indicated otherwise, loss and accuracy values in this section are expressed for the relevant test set.

2.3.2 Preprocessing

Most work with marine mammal communications is done through the spectrogram version of acoustic signals, particularly if non-click sounds are being investigated. As image data, the signals tend to be more evident, especially to a human operator; the distinctive contours of whistles are more easily identifiable by eye in spectrograms than by ear in audio clips. Spectrograms are a method of representing audio data that allows a viewer to visually interpret the change in signal frequency over time. The presentation of spectrograms may vary, but in this thesis we use x-axis as "time", y-axis as "frequency", and pixel intensity as power. This waveform-to-spectrogram conversion is done by decomposing the audio signal into its composite frequencies using the Discrete Fourier Transform (DFT). N data points are taken from the T_s-sampled acoustic signals. This segment is transformed via

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-i2\pi kn}{N}}, \quad k = 0, 1, \ldots, N-1. \tag{2.2}$$

$X(k)$ is the transformed signal at the k-th frequency bin corresponding to the frequency $f = \frac{k}{NT_s}$.

While performing a DFT of an entire waveform will reveal the composite frequencies, doing this operation over small snippets of the waveform will show the progression over time. This is known as a Short Time Fourier Transform (STFT), which provides temporally-localised information about the frequency components

in a signal. The full waveform is thus broken into consecutive and often overlapping segments of data, each of which individually undergoes an STFT. These segments are often combined with a windowing function $w(k)$ of length L, which can help to emphasize the most relevant moment in time (i.e. the "centre" of the segment). The STFT can be expressed as

$$X(m,k) = \sum_{l=m-(L-1)}^{m} x(l)w(m-l)e^{\frac{-i2\pi kl}{N}}, \quad k = 0,1,\ldots,N-1,\ m \in \mathbb{Z}\,. \tag{2.3}$$

The STFT is a function of time and frequency, where the m identifies the location of the time window and k is the frequency bin. Once this transform is created, the absolute value or square of the complex result is taken to provide the amplitude/power. These consecutive transforms are the columns of the spectogram. The amplitude/power is represented through the visual "brightness" of the image pixel. While this can be shown in a colour (RGB) image, there is no tangible benefit to this since it is only the relative intensity that matters. Therefore, we present spectrograms in grayscale.

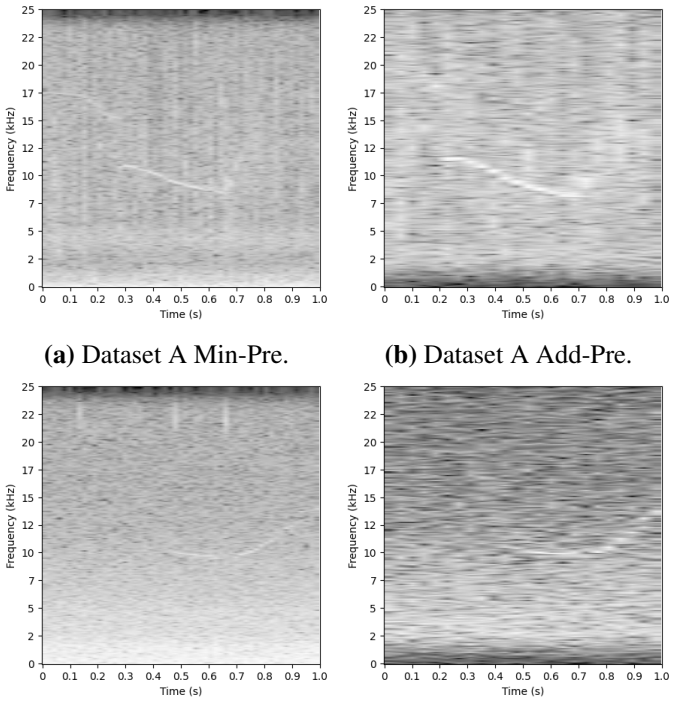
In our datasets, the time frame chosen for this conversion was audio clips of 1.0 s in length, which we review in Section 1.1 to be sufficient. Based on the data available, this encapsulates the vast majority of whistles without being overly large as to make the shorter-duration samples insignificant. The frequency range for spectrograms was 0 kHz to 25 kHz, which matches existing works as discussed in Section 1.1. Additionally, this encompasses the frequencies that a human operator noted the whistles in database A to be within. Due to the Nyquist limit, we therefore require that the audio data is sampled at a minimum of 50 kHz, twice that of our highest relevant frequency. Each audio clip thus consists of 50,000 data points. The spectrogram conversion is done using Hamming windows of length 1024 and overlap of 128, and these parameters were chosen based on visual assessment of different options. The resulting images are scaled to the appropriate size (224×224) for the NNs. Each spectrogram is min-max normalised between 0 and 1 to ensure consistency between samples.

The vast majority of the aforementioned works in Section 2.1 utilise a degree of data preprocessing aimed at enhancing the signal quality. This is commonly

a filter applied to the audio signal (i.e. bandpass or low/high pass) to isolate the frequency band that appears to be most relevant. Some authors perform additional steps, such as attempting to remove undesired noise in the background (e.g. click signals) or augmenting the appearance/strength of the desired signal. However, we aim to put as much of this work onto the NN as possible, thus minimising the amount of preprocessing that must be performed. We theorise that the changing environments across datasets would require some degree of customisation to obtain the most effective denoising schemes, whether this is simply changing certain threshold values or accounting for new noise. For example, this could be due to a different composition of creatures resulting in different noises or weather patterns dictating how the ocean itself may move and sound. To obtain consistent data samples in differing environments, the preprocessing scheme would likely need to be altered. If we eliminate these steps, we are able to leverage the huge learning capacity of NNs to adapt to novel conditions. This method of preprocessing is referred to as "Min-Pre" in following sections and only involves the specific windowing and spectrogram conversion method as discussed, and min-max normalisation for each sample. This represents, in our opinion, the minimal amount of work necessary to provide usable inputs to a NN if we are to use spectrograms.

Additionally, we replicate the work done in [32] by reproducing their preprocesing pipeline, referred to as "Add-Pre". Both [32] and our work utilises dataset A, but we remove less/different samples compared to their selection process. For consistency, we use our selected samples; this allows us to best assess performance using different models and preprocessing methods. In [32], audio clips are constrained to a duration of 0.8 s and frequency range of 3 kHz to 20 kHz. [32] uses different parameters for their spectrogram conversion, which we copy. As well, the audio is first fed through a bandpass filter from 5 kHz to 20 kHz. At the end, the spectrogram image is also resized and normalised as in Min-Pre. Any model trained on data preprocessed using Add-Pre is notated with "*" appended to their name in this thesis.

In Figure 2.4, examples of the images produced by these two preprocessing methods are shown. Since only one channel is used for a grayscale image, any additional input channels can be used to convey other information. Specifics about channel composition will be discussed in Section 2.3.3.



(a) Dataset A Min-Pre. **(b)** Dataset A Add-Pre.

(c) Dataset B Min-Pre. **(d)** Dataset B Add-Pre.

Figure 2.4: Spectrograms produced by data preprocessing methods on a sample whistle from dataset A and a sample whistle from dataset B.

2.3.3 Data Sample Composition

The pre-trained models were trained to classify colour images, which involves three input channels. In a pre-trained model's first layer, a single RGB channel's weights (arbitrarily chosen to be R) was copied across all three channels. This was done in hopes of encouraging a more equal assessment of the three channels and to break the typical dependence/relationship implied by the RGB structure.

We utilised multiple channels to enhance the data available for the model. Since dataset A has two-channel audio data, it is a sensical conclusion to use both. Additionally, we utilised the average of both channels as a third channel. This is a technique borrowed from [32], and they theorise that an averaged image may be able to moderate the background noise. This resulted in a 3-channel input for each data sample. In addition to this method, we also copy the data composition found

in [32] which stacks the average across all three channels as well as utilising Add-Pre. These models are noted with "**" appended to their name in this thesis. A single "*" only indicates that Add-Pre was used in the place of Min-Pre, and the input is composed of channel 1, channel 2, and averaged channel.

Dataset B only has a single channel of audio information. To maintain the easiest cross-usage of NNs between datasets, we triplicated this single channel to obtain a 3-channel image input.

2.4 Results

In this section, we discuss the results obtained using datasets A and B. We started by reviewing how model selection was conducted among the series of pre-trained models mentioned in Section 2.2.2. Then, we discuss results obtained in three training cases. First, models trained fully on dataset A. Second, models trained fully on dataset B. Third, models using hyperparameters and parameters developed from dataset A but trained on dataset B. The section concludes by briefly outlining experiments where only 1-channel inputs were used.

2.4.1 *k*-Fold Cross Validation

To start, we conducted k-fold cross validation on the three series of pre-trained models to determine the most effective model in each case. Each series considers several models which had the highest metrics as provided by PyTorch [42]. The entirety of dataset A was split into $k = 5$ segments, one of which was held out every time to serve as the test set. The resulting output of these models is in the range of $[0,1]$ as a probability, where 1 indicates presence of whistle. In our models, whenever the output exceeded 0.5, we consider this to be prediction of a whistle. Adam optimisation is used to train the models, and the relevant hyperparameters include learning rate, β_1, β_2, and decay. The models are trained using binary cross-entropy loss with a tuned hyperparameter that allowed the class weighting to be shifted. Typically, the binary cross-entropy loss of a single sample is calculated as

$$L_{bce} = y \log(\hat{y}) + (1-y) \log(1-\hat{y}). \tag{2.4}$$

y represents the true value of the sample (either 0 or 1) and $\hat{y}$ the predicted value as given by the model (a probability in range $[0,1]$). The class weight parameter modifies Equation 2.4 to become

$$L_{bce} = p_w y \log(\hat{y}) + (1 - y_{true}) \log(1 - \hat{y}). \tag{2.5}$$

When $p_w > 1$, the model is biased in favour of positive predictions (increased recall). When $p_w < 1$, this is in favour of negative predictions (increased precision).

The model was trained k times to utilise each segment as the test set while training with the rest. The models were trained on dataset A until they showed no improvement on training loss for a consecutive number of epochs. In total, we performed this three times for each model to obtain the results found in Table 2.1. Multiple trials were used to obtain a better sense of which models would be able to consistently perform better. This is true in the following sections as well, since we attempt to verify the benefits or drawbacks of the pre-processing methods.

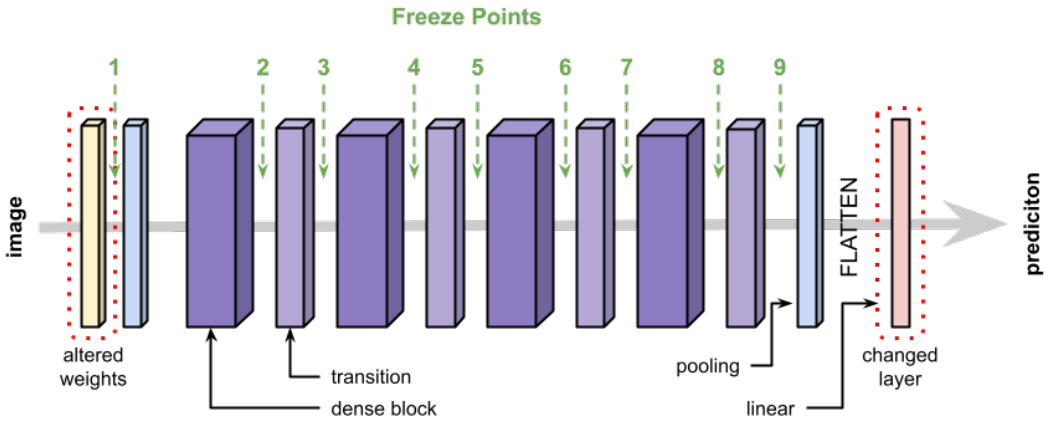
Based on these results, we selected the three following pre-trained models: Dense161, Res152, and VGG19bn. The architectures for these models are shown in Figure 2.5, which also indicates which layers are modified from their original incarnations. There are indications of where model freezing checkpoints are located, which will be explained in Section 2.5.1.

Model	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	Trial Avg	Total Avg
Dense161	0.2306	0.3200	0.3634	0.2586	0.2735	0.2892	0.2921
	0.2957	0.3035	0.2911	0.2587	0.2454	0.2789	
	0.3372	0.3024	0.3197	0.2127	0.3687	0.3082	
Dense169	0.2837	0.4600	0.4012	0.3987	0.2188	0.3525	0.3271
	0.3946	0.3035	0.2709	0.3713	0.2332	0.3147	
	0.3464	0.3292	0.3173	0.2942	0.2834	0.3141	
Dense201	0.4309	0.2118	0.3336	0.2828	0.3792	0.3276	0.3245
	0.2888	0.2588	0.3167	0.3766	0.3743	0.3231	
	0.3669	0.3251	0.2171	0.4055	0.2992	0.3228	
Res50	0.4384	0.3588	0.3032	0.3582	0.3254	0.3568	0.3729
	0.4403	0.3688	0.2742	0.4043	0.3196	0.3614	
	0.4349	0.3705	0.4172	0.5107	0.2691	0.4005	
Res101	0.4379	0.3224	0.3634	0.3534	0.4983	0.3951	0.3882
	0.4471	0.2830	0.4879	0.4265	0.5741	0.4437	
	0.4526	0.3608	0.2353	0.2996	0.2802	0.3257	
Res152	0.5279	0.3858	0.4005	0.4395	0.2607	0.4029	0.3639
	0.3917	0.4202	0.2007	0.3881	0.2307	0.3263	
	0.3876	0.2837	0.3494	0.4022	0.3905	0.3627	
VGG16	0.9557	0.2551	0.5278	0.9563	0.5301	0.6450	0.6169
	0.4787	0.6725	0.9557	0.3057	0.6513	0.6128	
	0.3006	0.3301	0.4187	0.9599	0.9557	0.5930	
VGG16bn	0.5601	0.3573	0.2480	0.6942	0.3795	0.4478	0.4133
	0.5240	0.3113	0.2024	0.5910	0.3305	0.3918	
	0.4610	0.3107	0.4159	0.4188	0.3950	0.4003	
VGG19	0.9555	0.2736	0.9562	0.9596	0.9583	0.8207	0.8609
	0.9559	0.9574	0.6502	0.9566	0.5055	0.8051	
	0.9581	0.9588	0.9558	0.9561	0.9557	0.9569	
VGG19bn	0.5520	0.3269	0.3589	0.4095	0.2898	0.3874	0.3713
	0.4821	0.3925	0.2524	0.4652	0.2861	0.3757	
	0.2087	0.3079	0.3359	0.4113	0.4905	0.3509	

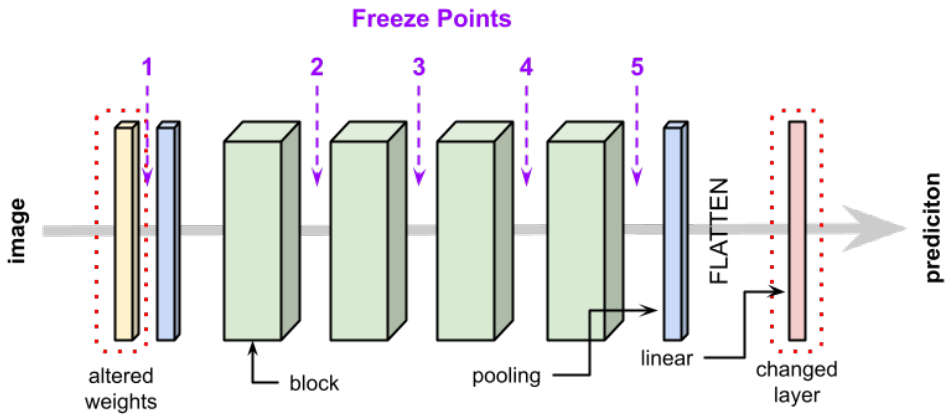
"Avg" indicates "average" as calculated by mean.

"bn" appended to a model name indicates that it is the batch normalised version.

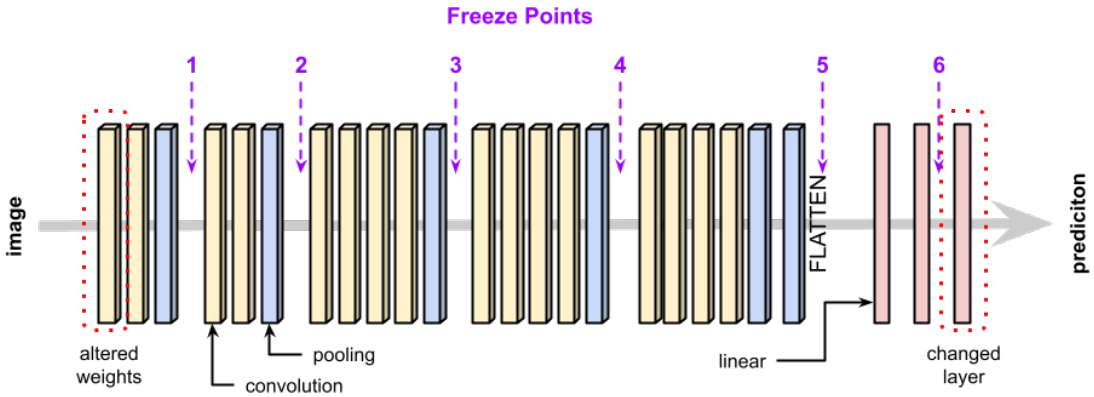
Table 2.1: k-cross accuracy results for all tested pre-trained models, done using dataset A with a set of standard hyperparameters and $k = 5$.



(a) Dense161.



(b) Res152.



(c) VGG19bn.

Figure 2.5: Architecture of selected pre-trained models.

2.4.2 Detection in Dataset A

We first train the models using both preprocessing methods. We start with dataset A and conducted hyperparameter tuning using optuna [4] for each of the following: Simple, Simple*, Dense161, Dense161*, Res152, Res152*, VGG19bn, VGG19bn*, VGG16**, and VGG16d**. A set of optimal hyperparameters was reached for each model, referred to as hyperparameter set A. Since the class weight hyperparameter varies between models, losses are no longer shown. Instead, Table 2.2 displays three trial results for each model reporting Accuracy (Acc), Missed Detection (MD), and False Alarm (FA). These are each calculated as follows:

$$Acc = \frac{TP+TN}{TP+TN+FP+FN} \tag{2.6}$$

$$MD = \frac{FN}{FN+TP} \tag{2.7}$$

$$FA = \frac{FP}{FP+TN}. \tag{2.8}$$

True Positive (TP) refers to the number of positive samples labelled (correctly) as 1. True Negative (TN) refers to the number of negative samples labelled (correctly) as 0. False Positive (FP) refers to the number of negative samples labelled (incorrectly) as 1. False Negative (FN) refers to the number of positive samples labelled (incorrectly) as 0.

Overall, the NNs achieve detection accuracy values ranging from approximately 93% to 96%. This is comparable to or better than existing detection methods, and this dataset does not reflect a class imbalance that may bias the reported accuracy. VGG16** and VGG16d** do not reflect the same 98.89% performance as mentioned in [32]. A contributing factor in this is likely the slight difference in data selection; although the same dataset is used, [32] is more selective with the samples specifically included. As well, they use an earlier iteration of tagged whistles different from those that we utilise. In general, the models are slightly biased towards producing MDs as opposed to FAs. This is in spite of the class weight hyperparameter, which is greater than 1 in all cases. For the vast majority of models, it is between 1 and 1.5.

For Dense161, the Min-Pre version shows an average accuracy improvement

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	89	92.95	4.00	10.10	93.84
	174	94.10	4.00	7.81	
	169	94.48	1.52	9.52	
Simple*	221	94.10	2.29	9.52	93.56
	195	92.48	2.67	12.38	
	268	94.10	1.90	9.90	
Dense161	10	96.86	0.95	5.33	96.06
	21	95.33	1.90	7.43	
	18	96.00	1.33	6.67	
Dense161*	19	95.24	0.95	8.57	94.67
	8	94.19	2.48	9.14	
	26	94.57	2.86	8.00	
Res152	8	94.48	1.33	9.71	94.29
	35	94.29	4.19	7.24	
	11	94.10	1.90	9.90	
Res152*	9	94.76	1.33	9.14	94.73
	7	94.76	1.71	8.76	
	5	94.67	1.52	9.14	
VGG19bn	28	95.90	3.24	4.95	95.46
	19	94.67	5.14	5.52	
	15	95.81	2.10	6.29	
VGG19bn*	7	96.10	1.14	6.67	95.65
	7	95.24	4.38	5.14	
	9	95.62	1.52	7.24	
VGG16**	6	95.24	3.81	5.71	94.98
	4	94.95	5.52	4.57	
	2	94.76	2.67	7.81	
VGG16d**	6	94.95	2.10	8.00	95.14
	6	95.33	2.10	7.24	
	7	95.14	3.24	6.48	

"Ep." indicates the number of epochs required to reach these results.

Table 2.2: Performance of models on dataset A using hyperparameter set A.

larger than 1%; all other models show very little difference between Min-Pre and Add-Pre. While the values do not vary greatly across the board, it can be seen through these results that the additional bandpass filter and closer cropping of the spectrogram used in Add-Pre do not tangibly benefit the final accuracy. The number of epochs required to train these models is generally the same for Simple and Dense161. However, in the VGGNet series and Res152, it seems to be lower when using Add-Pre, although this is only particularly noticeable in a few cases. This may be the result of the additional data preprocessing creating inputs that have less variability, as well as the cropping process exaggerating the whistle relative to the overall image.

The results from this section prove the effectiveness of NNs in application to dolphin whistle detection. We are able to achieve high accuracy with these models, and this appears to be a consistent result. As well, we show that a basic CNN is able to achieve reasonable performance but is generally outperformed by the other models. It also takes longer to converge, which is largely due to the lack of transfer learning. Additionally, it is a simpler model with fewer parameters, which typically indicates a lower capacity to learn. However, due to its size, it is also quick to train and does not occupy much storage space which may be useful if many different models need to be trained.

2.4.3 Generalisability in Dataset B

At this stage, there is no clear difference in the preprocessing methods. To further assess the potential benefit of using Min-Pre, we move to dataset B. The same hyperparameter tuning was conducted using this second dataset to obtain hyperparameter set B, and results can be found in Table 2.3. Comparisons between Table 2.2 and Table 2.3 are also graphically represented in Figure 2.6 for simpler viewing. This first provides a baseline, theoretically "optimal" performance in the second dataset since the hyperparameters are tuned for this dataset.

All models using Min-Pre experience a change in accuracy within 2% compared to the results from Table 2.2, with all models except for Res152 and VGG19bn showing a slight decrease in performance. However, the models using Add-Pre uniformly perform worse, ranging from 2% to 7% accuracy decrease. In addition,

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	21	93.44	4.38	8.75	93.13
	33	93.75	8.75	3.75	
	41	92.19	11.88	3.75	
Simple*	65	90.94	4.38	13.75	91.25
	38	90.63	3.75	15.00	
	31	92.19	1.25	14.38	
Dense161	36	95.31	5.00	4.38	95.63
	22	95.63	6.25	2.50	
	50	95.94	3.13	5.00	
Dense161*	32	90.00	4.38	15.63	91.46
	63	92.50	2.50	12.50	
	59	91.88	5.00	11.25	
Res152	7	96.25	2.50	5.00	95.94
	1	95.31	3.75	5.63	
	13	96.25	4.38	3.13	
Res152*	18	92.19	4.38	11.25	90.31
	13	91.56	1.88	15.00	
	36	87.19	10.00	15.63	
VGG19bn	2	95.94	3.75	4.38	96.04
	2	95.63	3.13	5.63	
	5	96.56	2.50	4.38	
VGG19bn*	6	86.25	0.00	27.50	86.98
	12	88.13	10.63	13.13	
	15	86.56	18.13	8.75	
VGG16**	2	90.00	2.50	17.50	90.52
	3	90.31	1.88	17.50	
	3	91.25	1.88	15.63	
VGG16d**	3	92.19	4.38	11.25	92.29
	5	92.19	3.13	12.50	
	3	92.50	3.75	11.25	

Table 2.3: Performance of models on dataset B using hyperparameter set B.

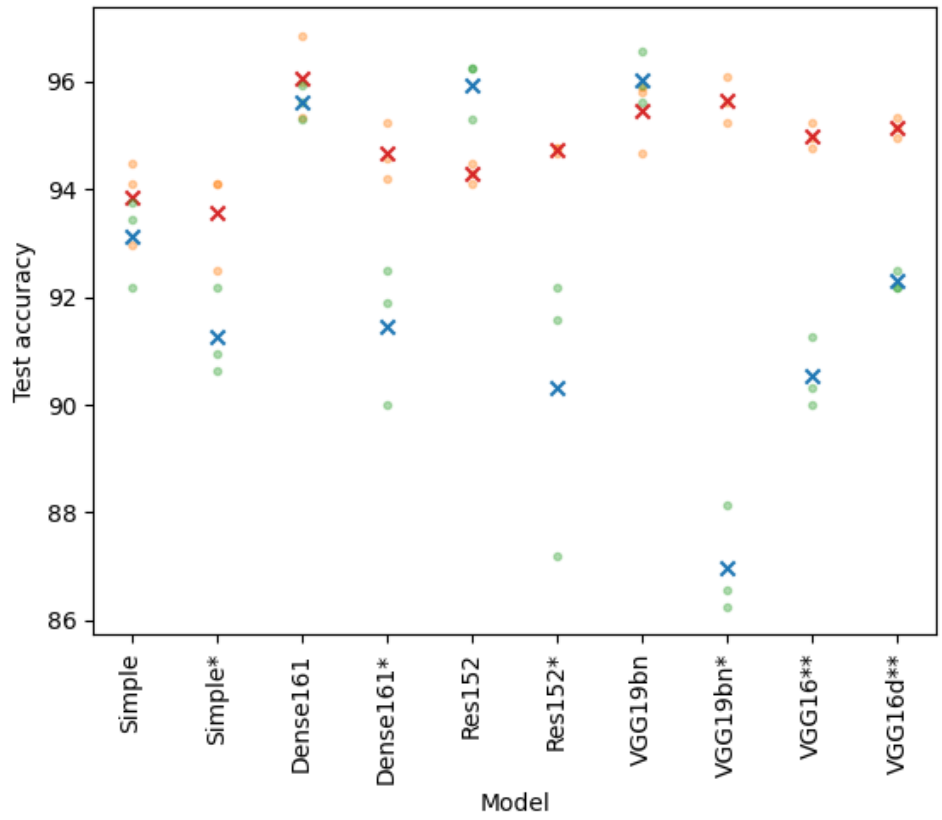


Figure 2.6: Visual depiction of results from Table 2.2 (orange per trial, red average) and Table 2.3 (green per trial, blue average).

Add-Pre models all suffer from MD rates of over 10% and as high as 27.50%, with the exception of VGG19bn* trial 3 being 8.75%. This lends credence to the idea that additional preprocessing elements may be detrimental to the model's learning process. Oftentimes, the end product of such preprocessing is assessed by eye or by training on a single dataset, which does not reflect how it may affect data from other environments. While it is true that this process may be changed, our results indicate that it can skipped altogether without a notable negative impact on detection performance. This shifts the burden of work nearly entirely onto the model and requires less contextual adjustments to be made by human operators.

The main benefit of this is in practical deployments of detection models. In the real world, we would like to have a single process that is as streamlined and widely

applicable as possible. We can expect that recordings from different environments will have a diverse range of background noises and conditions which may regularly require differing denoising algorithms to result in similar-looking spectrograms. In addition to the background differences, the dolphins whistles themselves may be variable. Literature such as [17] report that some characteristics of dolphin whistles, specifically duration and contour shape, tend to vary greatly between different populations. Specific groups of dolphins may have unique characteristics to their whistles as well. The geography and isolation of dolphin groups seem to impact the character of their communications, and it is suggested that migration between different regions may result in shared similarities. Additionally, [38] shows that anthropogenic noise such as ship engine sounds correlates with different dolphin whistles. Populations exposed to more engine noise tend to increase their frequency, duration, and modulation of communications. Altogether, while dolphin whistles may share a general structure and likeness, an individual spectrogram may vary significantly in both the background and the signal itself, particularly when taken from a different geographical context. Rather than accounting for these variations each time, we should allow the models to learn them.

2.4.4 Transferability Between Datasets

Additionally, we wanted to test if models pre-trained on spectrogram data would reach better results compared to those pre-trained on ImageNet-1k with the default weights provided by PyTorch. We also want to eliminate the need for hyperparameter tuning in every new dataset. Therefore, while the models are trained on dataset B, they start from the best iteration obtained in Table 2.2 and train using hyperparameter set A. The results can be found in Table 2.4. Comparisons between Table 2.3 and Table 2.4 are also graphically represented in Figure 2.7 for simpler viewing.

Overall, the performance of models from Table 2.4 does not differ greatly from Table 2.3. Almost all differences in average test accuracy remain with 1.5% between the two sets of results; VGG19bn* is an exception, as it sees an improvement of nearly 5%. As well, there does not seem to be a notable change in the distribution of errors belonging to FA or MD. For most models, the number of training

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	58	93.75	5.63	6.88	94.38
	62	94.69	3.13	7.50	
	42	94.69	3.13	7.50	
Simple*	15	90.63	0.63	18.13	90.21
	19	90.00	0.63	19.38	
	16	90.00	1.25	18.75	
Dense161	2	95.31	3.75	5.63	95.63
	13	96.25	2.50	5.00	
	3	95.31	3.13	6.25	
Dense161*	5	92.19	5.63	10.00	92.19
	2	91.88	1.88	14.38	
	4	92.50	2.50	12.50	
Res152	12	95.00	5.63	4.38	95.73
	20	95.63	2.50	6.25	
	5	96.56	3.75	3.13	
Res152*	3	91.25	4.38	13.13	91.56
	2	92.50	6.25	8.75	
	1	90.94	3.75	14.38	
VGG19bn	9	96.25	0.63	6.88	96.98
	3	97.81	0.63	3.75	
	6	96.88	1.25	5.00	
VGG19bn*	2	90.94	1.88	16.25	91.88
	2	93.44	3.75	9.38	
	3	91.25	3.75	13.75	
VGG16**	2	91.56	10.00	6.88	91.88
	2	92.81	5.63	8.75	
	3	91.25	3.75	13.75	
VGG16d**	5	90.63	3.75	15.00	91.15
	3	91.88	0.63	15.63	
	2	90.94	3.75	14.38	

Table 2.4: Performance of models on dataset B using hyperparameter set A and starting from their respective best model in Table 2.2.

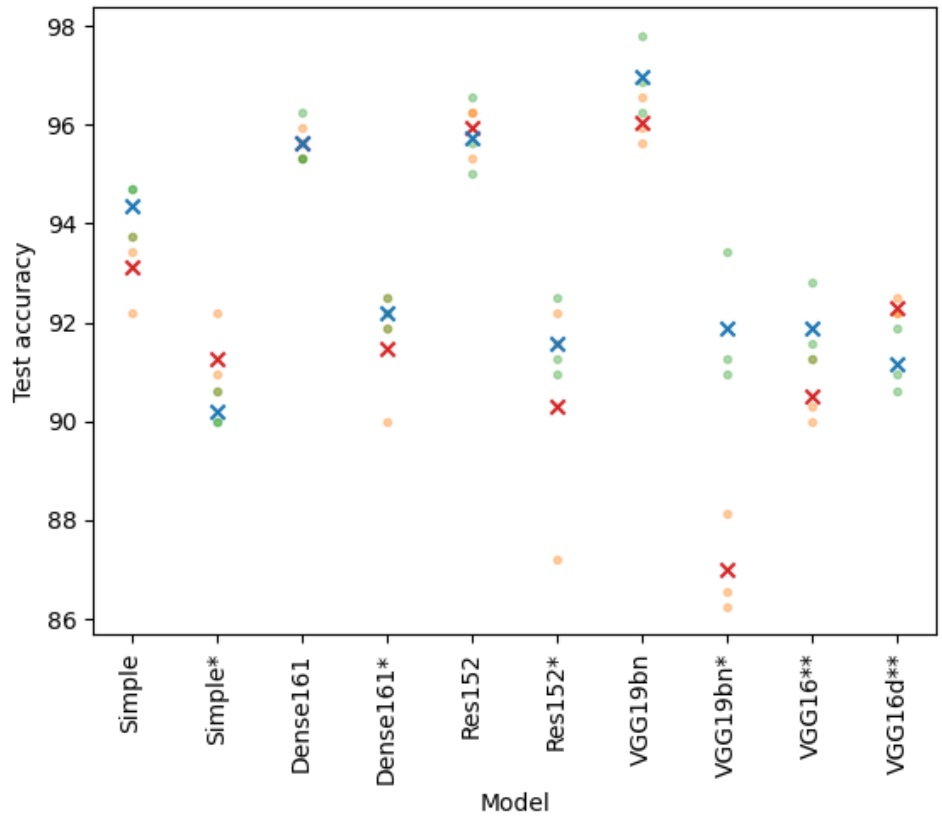


Figure 2.7: Visual depiction of results from Table 2.3 (orange per trial, red average) and Table 2.4 (green per trial, blue average).

epochs required is not significantly different. However, Dense161 and Dense161* appear to converge at a noticeably lower epoch number here as opposed to in Table 2.3. In general, using a starting point of more similar data than that taken from ImageNet-1k does not have a large impact in this section. This seems to indicate that when enough data is available, the starting point of the model may not matter for the final performance. This will be explored further in Section 2.5.3.

2.4.5 Single Channel

A verification on findings reported in this section are conducted using a different data input style. Since dataset A comes in the form of two-channel audio data, we

look at three different cases: channel 1, channel 2, and averaged channels. Rather than having a 3-channel image data where these are stacked, we utilise them individually. This more closely allows dataset A and dataset B to resemble one another – since dataset B is one-channel – and eliminates any potentially confounding factors in investigating the utility of Min-Pre as opposed to Add-Pre. Results from these experiments can be found in Appendices A, B, C, and D displayed in the same tables/figures as found in this chapter.

Overall, these results confirm that Min-Pre is sufficient in dataset A compared to Add-Pre and actually outperforms the latter in dataset B. Additionally, utilising the 3-channel data input method does appear to represent a performance increase of several percent in dataset A. Based on training accuracy values that we obtained, this does not seem to be a case of overfitting in dataset A's training set as opposed to its test set. Instead, we conclude that it is like a characteristic of the dataset itself. It is likely that there are some cases where the spectrogram whistle is particularly faint in one channel as opposed to the other, which results in a better performance when the stacked data method is used. As well, the tagged whistles in dataset B were manually reviewed by us rather than an expert in the marine biology field; as such, the positive samples are likely biased towards stronger signals.

2.5 Additional Experiments

In this section, we return to using the 3-channel input style as described in Section 2.3.3. Since previous results have shown that Add-Pre does not provide benefits to the overall detection scheme, we only investigate models using Min-Pre hereon out. We consider the effects of parameter freezing and training set size, as well as combining the two.

2.5.1 Parameter Freezing

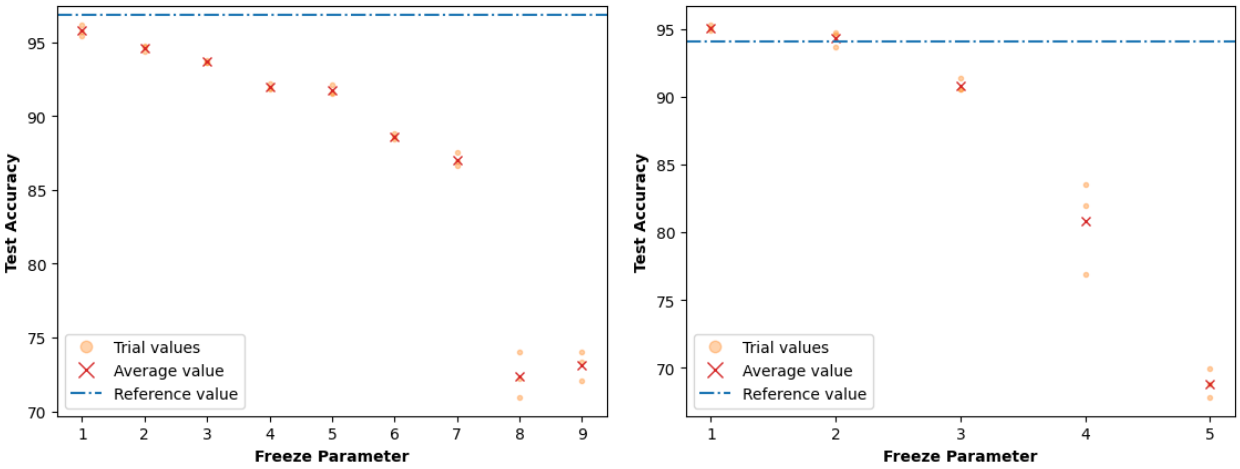
For models from the VGGNet, ResNet, or DenseNet series, we have thus far utilised a pre-trained model as a starting point for training and allowed the entire model to train. This is intuitively sensible since we expect many differences between the ImageNet-1k samples and our spectrogram samples, particularly in the way that the 3-channel inputs are laid out. However, some applications of transfer

learning will use a fine-tuning approach where only part of the model is able to update; the remaining layers will be "frozen". This is particularly used in image classification models since it is widely understood that early layers in the models detect patterns (e.g. vertical or horizontal lines). Therefore, these patterns can be building blocks for nearly all image classification tasks. We would want to fix the learned patterns in place when a much larger dataset was used rather than risk overfitting on a newer task with a smaller dataset. Especially for our dolphin whistles, the diversity of data is much less than for image ImageNet-1k classification samples and retraining could potentially lose some of that diversity.

[53] explores how well freezing model parameters can be applied to different tasks. They test this on two classification tasks made from discrete subsets of the ImageNet dataset. CNNs are trained and cross-trained while being frozen at various points throughout their architecture. Overall, [53] finds that utilising a trained model as a starting point for a new task and allowing the entire system to relearn is more successful than performing any parameter freezing. The authors theorise that this is due to connections which are learned between layers and likely need to be refined together. When a layer is frozen, it loses this ability to adapt.

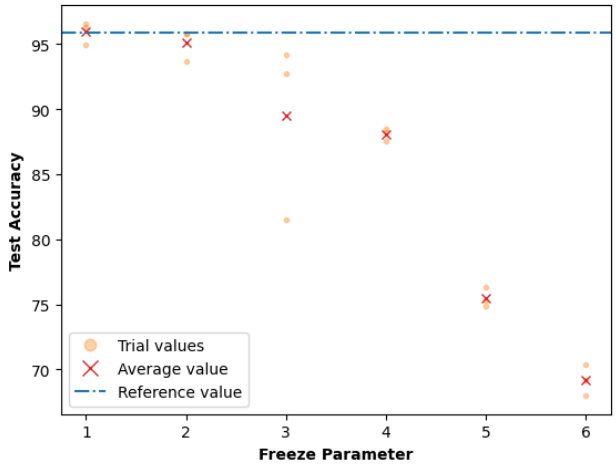
In this section, we attempt a similar experiment with our detection task. As seen in Section 2.4.1, the three model architectures in Figure 2.5 have various freezing checkpoints indicated. These "Freeze Points" denote the point up to where a model is frozen. For example, model Dense161 with $f = 1$ freezes only the first convolution layer while the rest of the model is able to train; by contrast, Dense161 with $f = 9$ only allows the final dense layer to train while the rest of the model is frozen. Models Dense161 and Res152 are composed of a series of distinct, repeated blocks that serve as logical breakpoints for freezing parameters. VGG19bn has repeating sets of convolutional layers followed by a pooling layer, so freezing checkpoints were inserted throughout after pooling layers. For all models, the first checkpoint is always after the first convolutional layer and the last only excludes the final dense layer(s). The number of trainable parameters remaining in each model based on the freeze point can be found in Table 2.5. We train the frozen models on dataset A and using hyperparameter set A, the results of which are shown in Figure 2.8.

In general, our findings align with [53]. A model's performance decreases as



(a) Dense161.

(b) Res152.



(c) VGG19bn.

Figure 2.8: Three trials of parameter freezing conducted for each NN using dataset A and hyperparameter set A. The results at various partially-frozen conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.2.

Model	Freeze	Parameters	% Parameters
Dense161	-	26,474,209	100.00
	1	26,459,905	99.95
	2	25,708,513	97.11
	3	25,634,017	96.83
	4	23,572,513	89.04
	5	23,276,065	87.92
	6	11,727,841	44.30
	7	9,493,345	35.86
	8	6,625	0.025
	9	2209	0.008
Res152	-	58,145,857	100.00
	1	58,136,321	99.98
	2	57,930,049	99.63
	3	55,590,209	95.60
	4	14,976,321	25.76
	5	11,585	0.020
VGG19bn	-	139,585,345	100.00
	1	139,546,369	99.97
	2	139,324,417	99.81
	3	137,256,961	98.33
	4	128,993,281	92.41
	5	119,549,953	85.65
	6	4097	0.003

Table 2.5: Number of parameters in each of the freeze-able models. "% Parameters" indicates the percentage utilised of total trainable parameters.

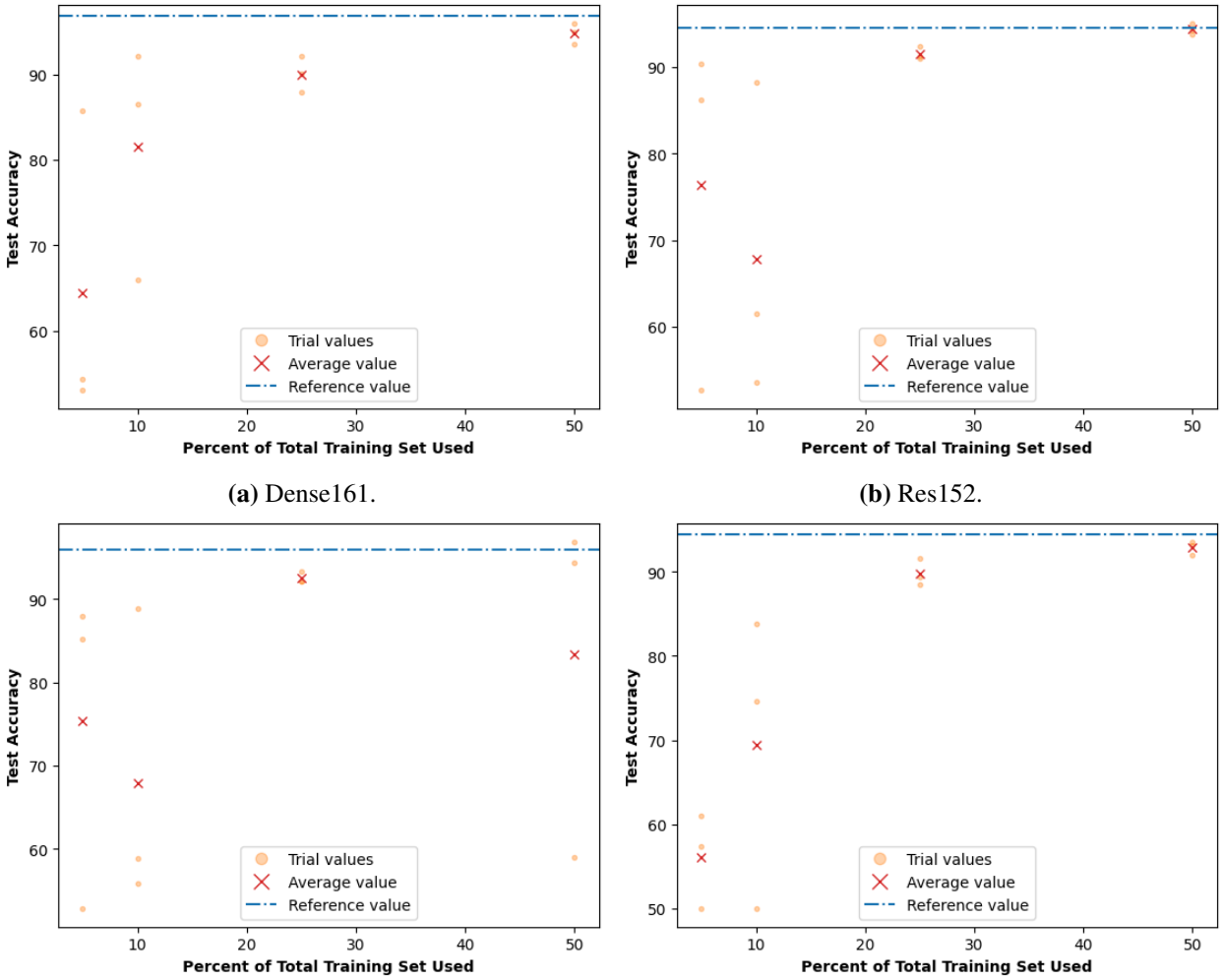
more of its parameters are frozen, with accuracy suffering an approximate 20% decrease in the worst case. The only exception is $f = 1$ in models Res152 and VGG19bn, where the performance is approximately equal to and occasionally better than the best trial with no freezing. This trend is also reflected in the loss values, which are not depicted. In most cases, the MD rate is higher than the FA rate, as it is in the baseline. While parameter freezing alone appears to be harmful to model performance, we attempt this under different circumstances in Section 2.5.3.

2.5.2 Dataset Size Reduction

While the ideal image classification task would involve tens of thousands of data samples, this is not always realistic, particularly for a set of dolphin whistles. Manual whistle tagging can be time-consuming and slow, and oftentimes there are only several hundred samples available. Our datasets number in the low thousands. The quality of these samples can also vary, which makes consistency difficult. As such, we investigate the ability of these models to perform at smaller training set sizes. For the sake of comparability, the training, validation, and test sets are split in the same manner as previously. While the validation and test sets are untouched, the training set is reduced since this represents what the model is actually seeing. While a real life situation would likely also reflect in a reduced validation set, this was the best way to directly examine the desired impact of fewer training samples.

We first use dataset A's training set of originally 4197 whistles (80% of the total whistle count). In conjunction with the negative samples, there are a total of 8397 samples in the training set. This was reduced to 50% (4196 samples), 25% (2098 samples), 10% (838 samples), and 5% (418 samples). Both validation and tests remain at 1050 samples throughout. Models are trained until they reach a maximum number of epochs or until they show no improvement for a certain number of consecutive epochs. This stopping condition was increased inversely proportionally to the training set size reduction which would result in an approximately equal training time at the end. We train the frozen models on dataset A and using hyperparameter set A, the results of which are shown in Figure 2.9.

In general, we obtain the expected result that a smaller training set obtains significantly poorer performance. Simple seems to suffer the most consistent downgrade, with the smallest training set size (5%) obtaining on average barely above 50% accuracy, effectively a random guess. Interestingly, the pre-trained models are occasionally able to obtain results with semi-reasonable performances even at the smallest training set size, each obtaining at least one version of the three trials that exceeds 85% test accuracy in every case. However, this is an inconsistent accomplishment, as can be witnessed by the large spread of results in the 10% and 5% cases. Models Res152 and VGG19bn also show an unexpected case of the 10% training set having a worse average accuracy than its 5% counterpart, which rein-



(a) Dense161.

(b) Res152.

(c) VGG19bn.

(d) Simple.

Figure 2.9: Three trials of reduced training set sizes conducted for each NN using dataset A and hyperparameter set A. Results at various partially-frozen conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.2.

forces the degree of inconsistency in using a small number of samples. Overall, a small dataset severely hampers the reliability and consistency of model results.

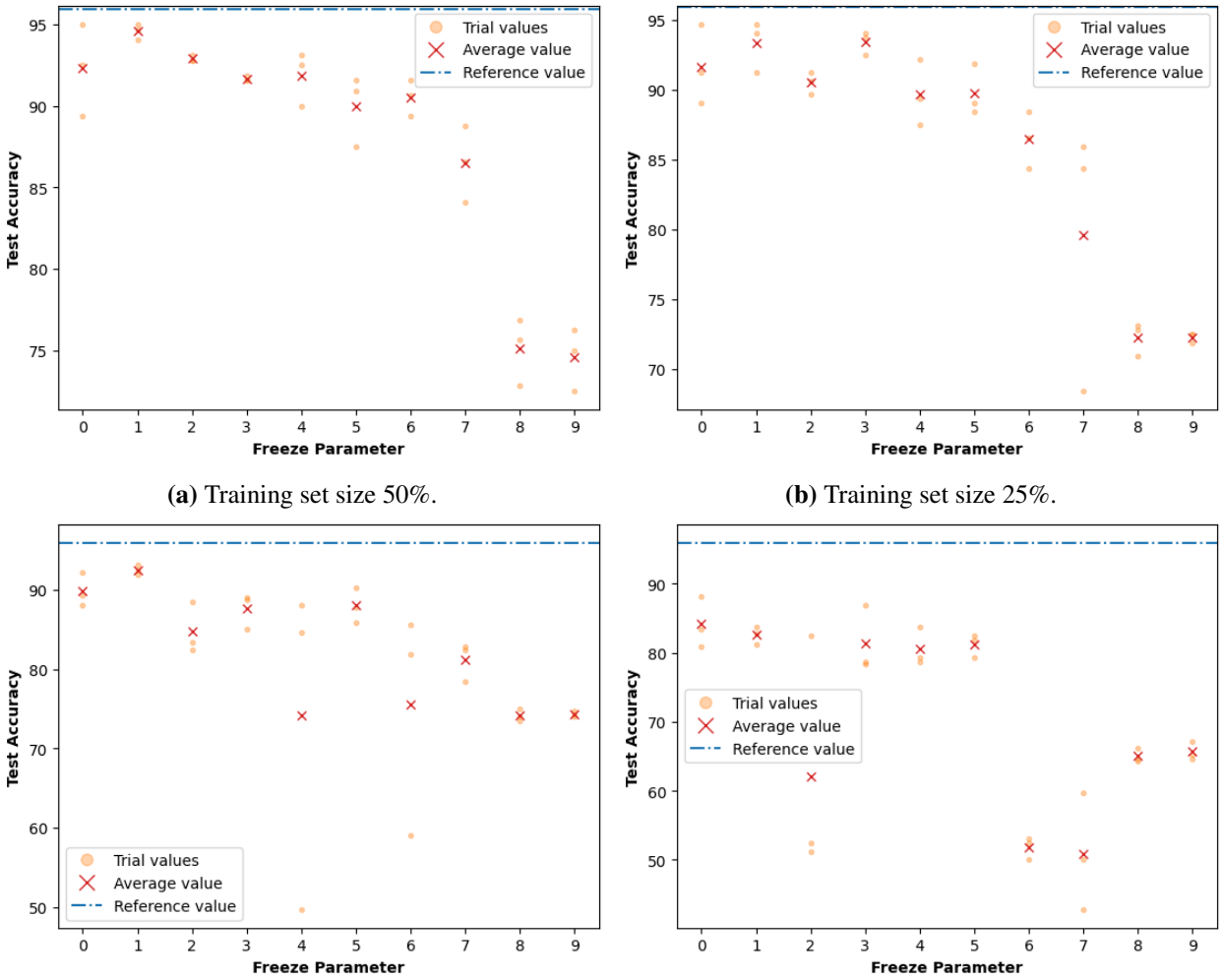
2.5.3 Combined Freezing and Size Reduction

A final experiment was done to assess if parameter freezing could help in the case of a smaller dataset size, combining the techniques in Section 2.5.1 and 2.5.2. This is an intuitive application since we are re-using parameters obtained from a large dataset into a smaller one. This is comparable to the benefits obtained by starting on pre-trained ImageNet-1k models to the dolphin whistle detection task, and potentially better since this transference is more directly comparable as the datasets are more alike. Freezing the early parameters may help prevent the models from overfitting, which a task with a small sample size can be very prone to, and it could potentially yield more consistent results. Dataset B consists of 5114 training samples, which are reduced as follows: 50% for 2557 samples, 25% for 1279 samples, 10% for 512 samples, and 5% for 256 samples. The same freezing checkpoints are used as in Section 2.5.1, and thus we only consider models Dense161, Res152, and VGG19bn. In each case, examples for $f = 0$ are included as references for what the training results may be in the absence of any parameter freezing.

In order to assess the potential benefit of freezing and size reduction in a realistic environment, we use dataset B in two situations: trained from scratched (i.e. ImageNet-1k starting parameters, similar to Section 2.4.3) and re-trained from the best on dataset A (i.e. the best from Table 2.2, similar to Section 2.4.4). In the the first case, we are providing optimal conditions for models to learn the whistles of our dataset but without learned whistle-detection parameters. Hyperparameter set B is used in this scenario. In the second case, we mimic entering a new environment with previously learned information which includes the hyperparameters.

Figures 2.10, 2.11, and 2.12 present the first case aforementioned, where the models are trained from scratch.

For every freeze parameter/partial training set possibility, no average test accuracy beats the baseline comparison, and in fact only one trial overall manages to beat the baseline. However, we can also see that every model and partial training set greater than 5% is able to come within approximately 5% of the best baseline



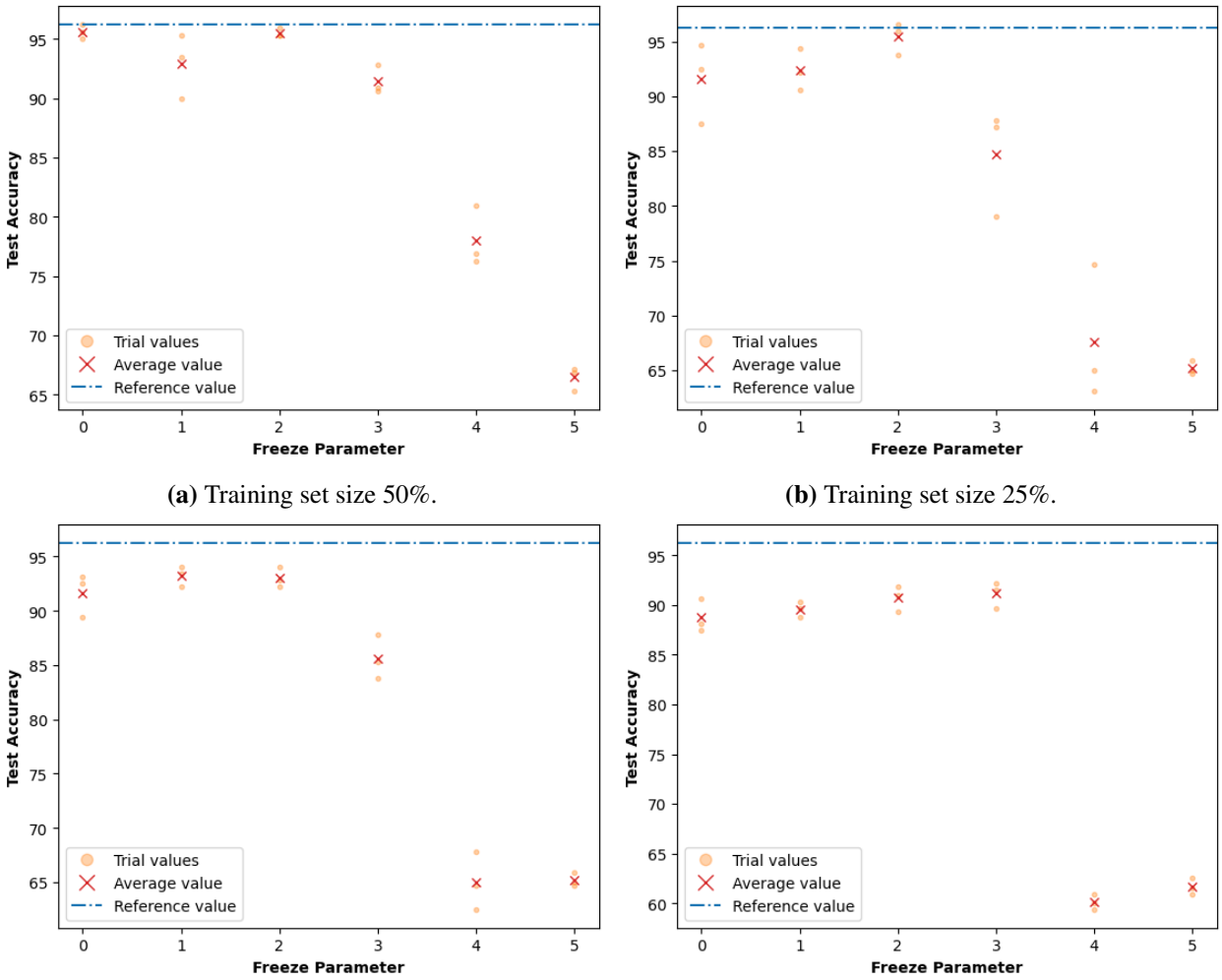
(a) Training set size 50%.

(b) Training set size 25%.

(c) Training set size 10%.

(d) Training set size 5%.

Figure 2.10: Three trials of frozen and partial training set size conducted for Dense161 using dataset B and hyperparameter set B. Each graph represents a different training set size reduction, with freezing parameters indicated on the horizontal axis. $f = 0$ is provided as a baseline for when no freezing is conducted. Results at various conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.3.



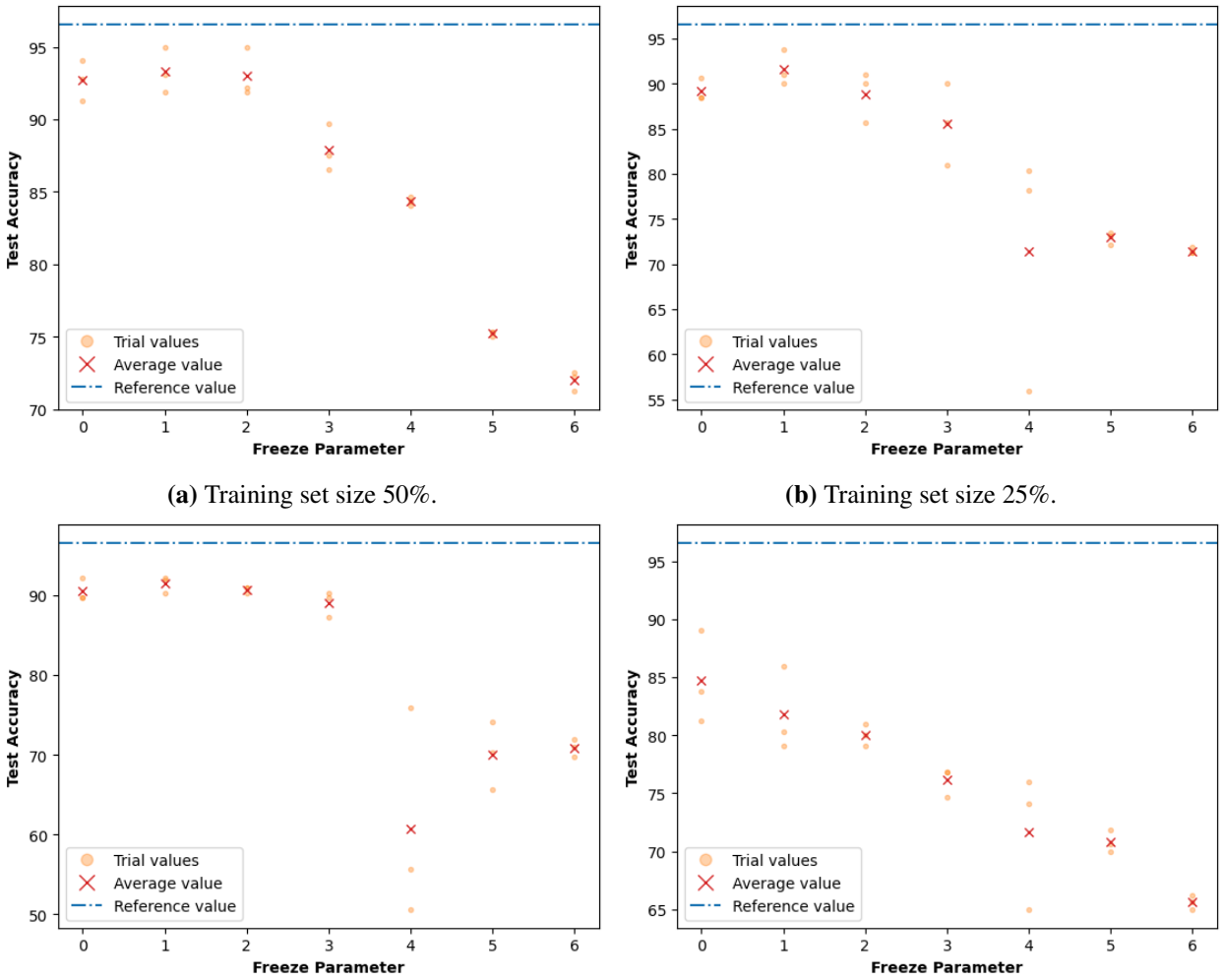
(a) Training set size 50%.

(b) Training set size 25%.

(c) Training set size 10%.

(d) Training set size 5%.

Figure 2.11: Three trials of frozen and partial training set size conducted for Res152 using dataset B and hyperparameter set B. Each graph represents a different training set size reduction, with freezing parameters indicated on the horizontal axis. $f = 0$ is provided as a baseline for when no freezing is conducted. Results at various conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.3.



(a) Training set size 50%.

(b) Training set size 25%.

(c) Training set size 10%.

(d) Training set size 5%.

Figure 2.12: Three trials of frozen and partial training set size conducted for VGG19bn using dataset B and hyperparameter set B. Each graph represents a different training set size reduction, with freezing parameters indicated on the horizontal axis. $f = 0$ is provided as a baseline for when no freezing is conducted. Results at various conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.3.

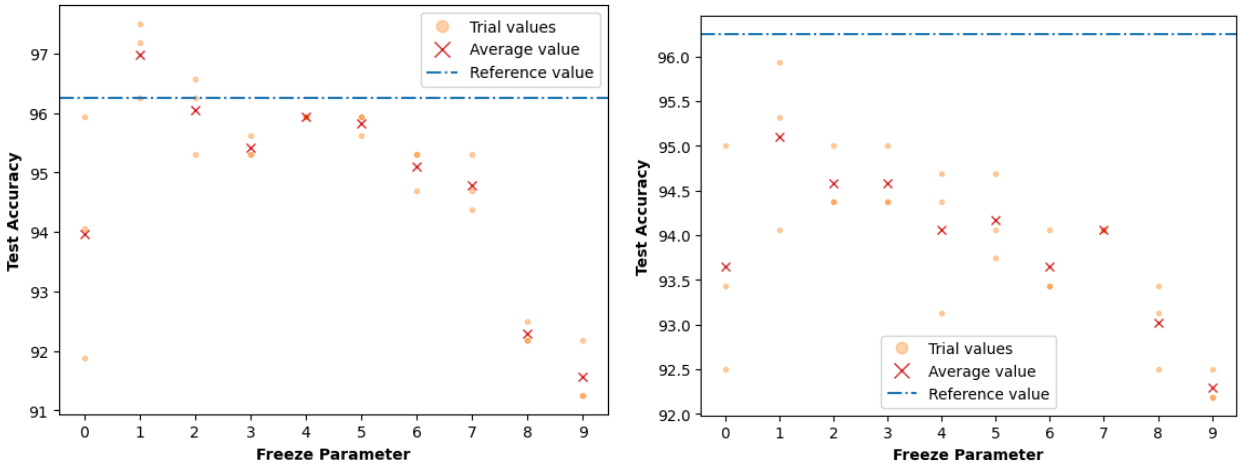
performance. When using a smaller dataset, parameter freezing is an advantage.

There are several instances where a small amount of parameter freezing ($f = 1$ or $f = 2$) can result in a better performance than none. However, none of the models achieve a good performance when the freezing parameter is at a maximum. This represents the case where only the dense layers at the end of a CNN are allowed to train, which is often the case for transfer learning. These can suffer a test accuracy decrease as high as 30%. Potentially due to the differences in data type between ImageNet-1k and spectrograms, this method is non-ideal. A final notable impact is that the training results are more consistent when parameter freezing is involved. In Figure 2.9, it can be seen that the 10% and 5% cases generally show a spread of test accuracy values within approximately 10%. Although there are some instances in which the three trials of figures 2.10, 2.11, and 2.12 achieve widely different test accuracy values (differing up to approximately 30%), this is rare.

Figures 2.13, 2.14, and 2.15 present the second case, where the models are re-trained from a starting point. Hyperparameter set A is used, and this simulates a "realistic case" scenario since the hyperparameters and model were taken from the large, known dataset.

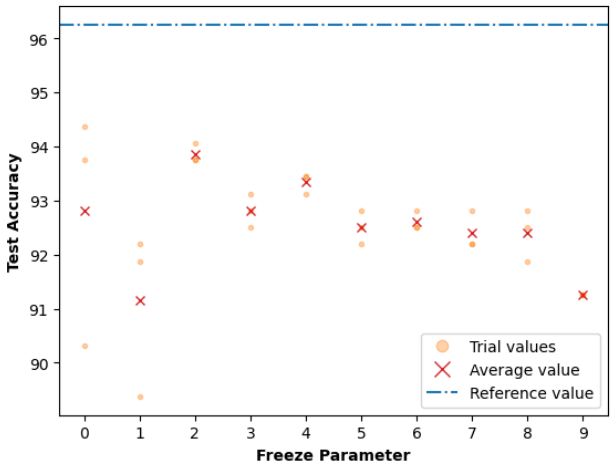
Overall, it can be seen that this yields much more useful results when parameter freezing is utilised. The difference between a non-frozen model's as opposed to partially-frozen model's performances is considerably less stark than previously, indicating that useful patterns have been preserved. This is in spite of the data coming from different environments and being composed differently, since dataset B has only one channel being replicated three times as opposed to a combination of multi-channel and averaged data. While a maximum f parameter still does not yield an ideal training process, every graph shows that $f = 0$ also does not, indicating that early freezing will definitely improve the model's performance.

Finally, we want to compare the results from these two cases more directly. In Figure 2.16, we contrast these directly, and a positive value shows an improvement in performance when using the second case. Nearly all combination of freezing parameter and partial training set size are improved when using a better starting point, and this improvement is very large (approximately 40%) in a few cases. It is also generally more significant the smaller the training set is, which can be particularly useful in real life when a new dataset is very sparsely tagged.

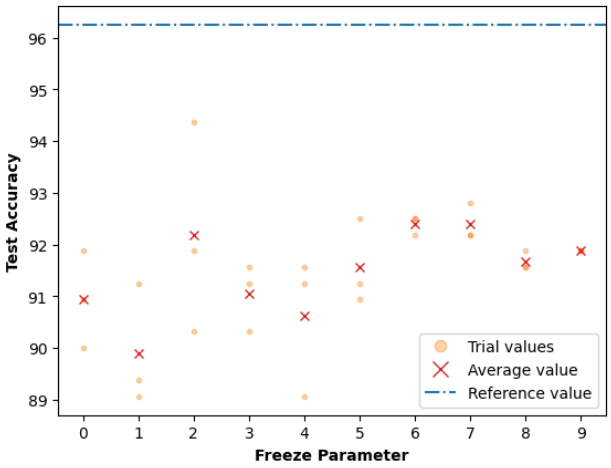


(a) Training set size 50%.

(b) Training set size 25%.

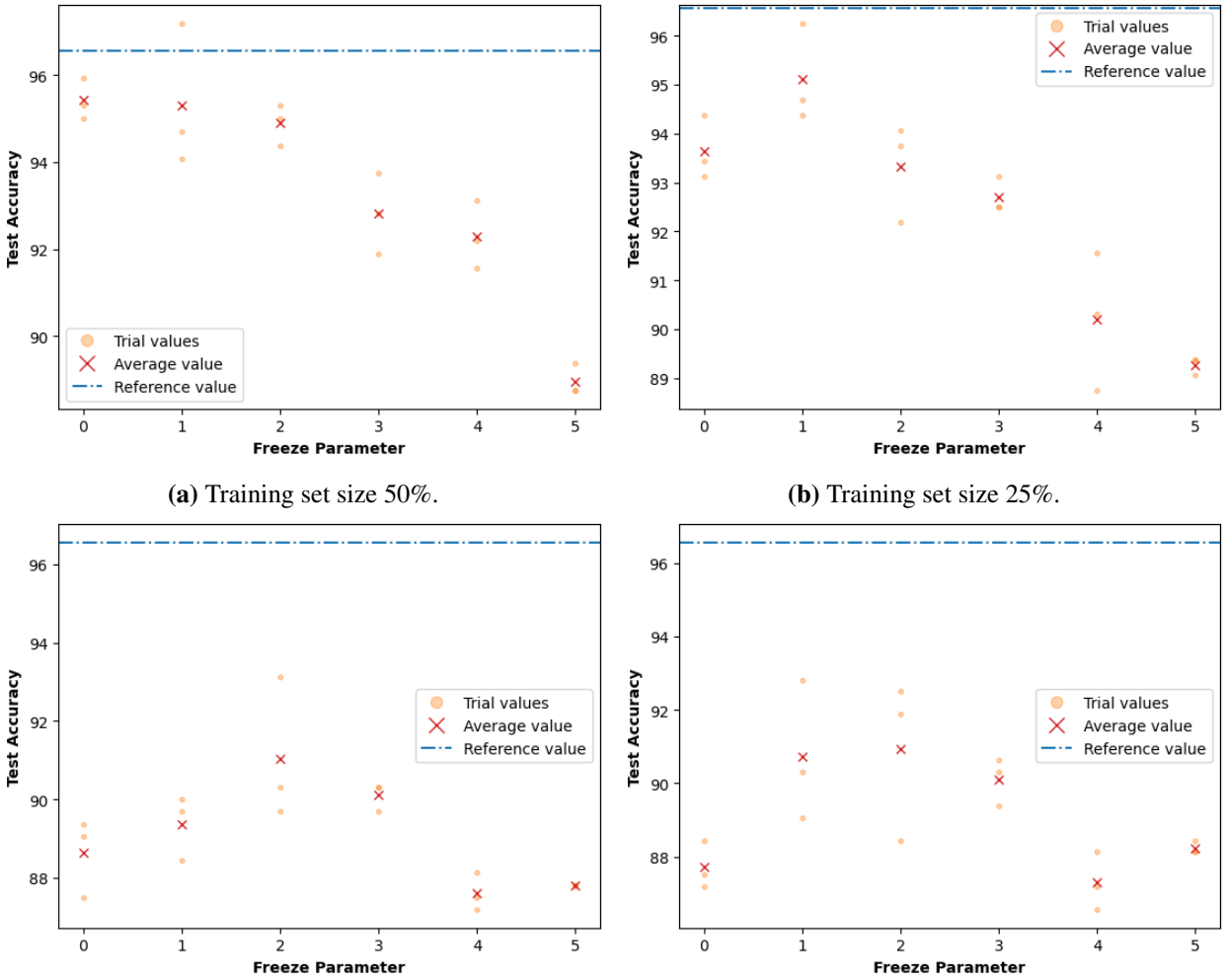


(c) Training set size 10%.



(d) Training set size 5%.

Figure 2.13: Three trials of frozen and partial training set size conducted for Dense161 using dataset B and hyperparameter set A and model starting point from best version obtained in Table 2.2 Each graph represents a different training set size reduction, with freezing parameters indicated on the horizontal axis. $f = 0$ is provided as a baseline for when no freezing is conducted. Results at various conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.4.



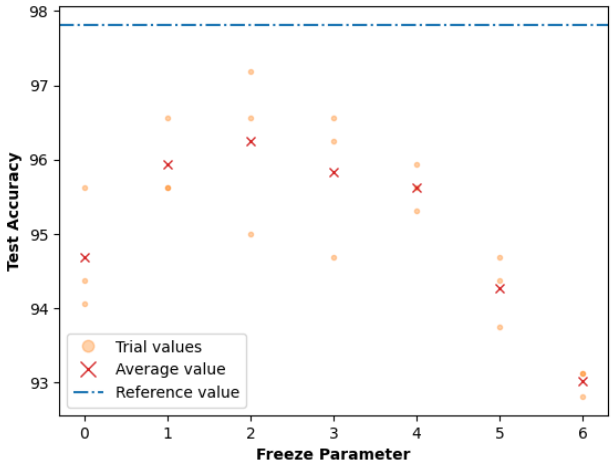
(a) Training set size 50%.

(b) Training set size 25%.

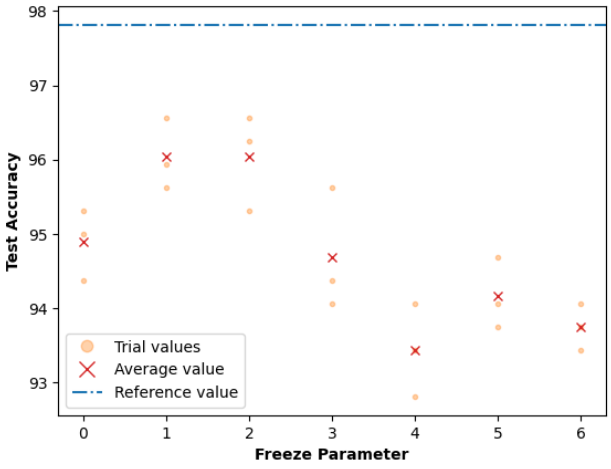
(c) Training set size 10%.

(d) Training set size 5%.

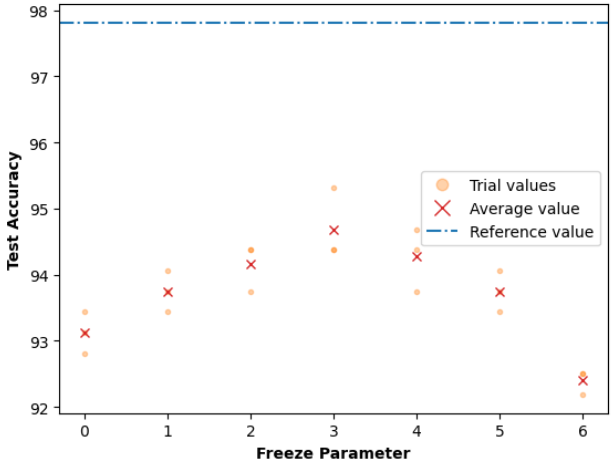
Figure 2.14: Three trials of frozen and partial training set size conducted for Res152 using dataset B and hyperparameter set A and model starting point from best version obtained in Table 2.2 Each graph represents a different training set size reduction, with freezing parameters indicated on the horizontal axis. $f = 0$ is provided as a baseline for when no freezing is conducted. Results at various conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.4.



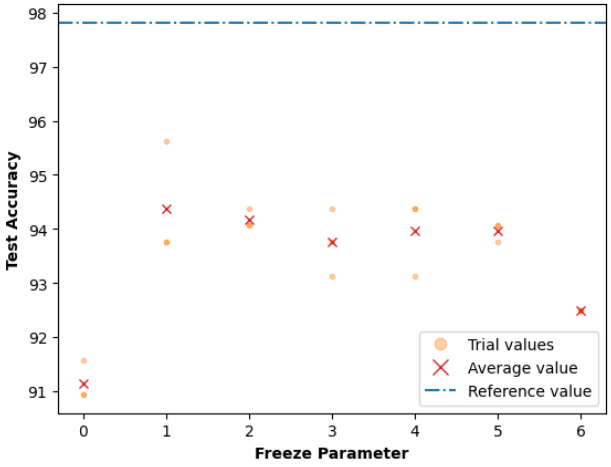
(a) Training set size 50%.



(b) Training set size 25%.

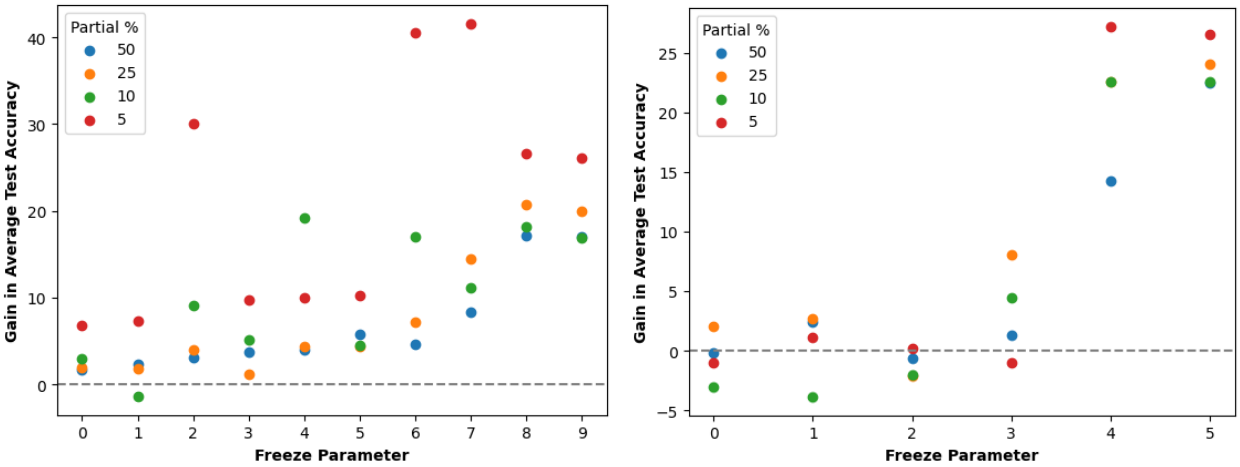


(c) Training set size 10%.



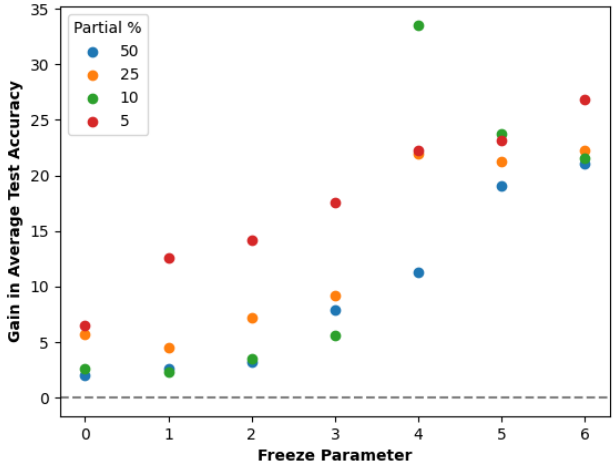
(d) Training set size 5%.

Figure 2.15: Three trials of frozen and partial training set size conducted for VGG19bn using dataset B and hyperparameter set A and model starting point from best version obtained in Table 2.2 Each graph represents a different training set size reduction, with freezing parameters indicated on the horizontal axis. $f = 0$ is provided as a baseline for when no freezing is conducted. Results at various conditions are shown in orange and red, while the blue line indicates the best test accuracy from Table 2.4.



(a) Dense161.

(b) Res152.



(c) VGG19bn.

Figure 2.16: Comparison of average results from results in figures 2.10, 2.11, and 2.12 against those found in figures 2.13, 2.14, and 2.15. The grey line represents 0% difference, and positive shows improvement in the latter set's favour.

Chapter 3

Contour Generation

Generative neural networks have managed in recent years to progress towards producing extremely high-quality samples, particularly in the realm of image generation. By applying these models to the task of dolphin whistle contour generation, we aim to to obtain "original" whistles that can be used in a variety of circumstances. We consider a full dolphin whistle to consist of its general contour shape and the variations that affect it; visually, these variations appear in spectrograms through the "brightness" of pixel values and the "thickness" of the signal. In this chapter, we focus on generating only the guiding shape (i.e. the contour) of the whistle. Synthetic whistles are those produced through any means that do not originate from a marine mammal. Whistles which are produced by dolphins are interchangeably referred to as "true", "real", or "biological".

This chapter begins with an outline of the existing whistle contour generation schemes. Then, we explain the process through which data is obtained. Finally, we outline the two types of image generation networks attempted in this thesis: Generative Adversarial Networks (GANs) and Denoising Diffusion Probabilistic Models (DDPMs). The results of these network types are shown, and a best model is chosen. Overall, this chapter demonstrates the basics of generating whistle contours, with further expansion on generating a realistic, usable whistle in Chapter 4.

3.1 Existing Methods

Whistle generation has been far less explored than the realm of whistle detection. However, a recently emerging use for marine mammal signals is to enable covert UWAC. Covert communications refer to the transmission and receipt of information in a manner that is hidden from observers, and in the underwater domain, this is typically done through acoustic signals. A common technique to accomplish this is to use spread spectrum, where a signal is spread over a large bandwidth and thus each frequency band has a very low power output. However, these methods are not fool-proof; in particular, they result in reduced communication performance due to the lowered power output [36]. As well, they can be detected by integrating over a sufficiently long detection period [36]. The output of this type of covert UWAC is emission of non-natural noise in an oceanic environment. As such, there has been interest in using biologically-inspired covert UWAC, wherein a signal such as a dolphin whistle can be used to convey hidden communications. Because these produced signals would appear as naturally occurring, they do not need to be weakly transmitted and can have the benefit of larger ranges. With this as a final goal, it becomes obvious that there is need to produce signals which are absent of recorded background noises and can be modified in some way to carry this information. In general, existing methods incorporate varying degrees of what we call "model-based" and "recreation-based" techniques.

We refer to model-based generative methods as those which produce whistles using mathematical models with little reference to recorded signals other than as a baseline for general comparison. While these whistles can be regarded as more original, they may also be very simple and thus not realistic. Some simpler cases involve generating a signal that is merely whistle-like. For instance, in [8], dolphin whistles are studied for their mixed-directional properties as a method of gaining insight on their behaviours. Synthetic whistles are generated to recreate recordings to circumvent the additive background noises. This paper uses MATLAB to generate pure tones with terminal up-sweeps, adding harmonics as integer multiples of the fundamental frequency to mimic the harmonic profile of the original recordings. The use case for these generated signals is not as heavily reliant on their realism and resemblance to true dolphin whistles, so the generative method is less

attached to existing recordings and ensuring realism.

On the other end, there are certain situations where the "generated" signal is merely a replay or recreation of a recorded one in what we would call a recreation-based technique. These whistles tend to be more complex and have contours that visually resemble biological signals, but this results in a lack of originality in the generation process. If used for biocovert communications, an individual, distinct unit of sound will typically stand in as a symbol that can be decoded in conjunction with others to reveal the intended message. Generating rather than only replaying, however, involves either modifying or recreating a whistle. A simplistic technique of achieving this can be found in [14]. Tagged time-frequency points on real whistle contours are used to fit a polynomial function that is recreated in the audio domain to produce a synthetic whistle waveform. In [36] and [52], covert UWAC is conducted using more than one signal, in a format that is not uncommon for this field. Firstly, there is a synchronisation signal in the form of a specific replayed dolphin whistle, which can be used to inform the listener of channel characteristics or simply that there is oncoming communications. Secondly, the authors use different, existing signal's contours to carry information. In [36], the following signals are either up-sweep or down-sweep to represent a bit of information. In [52], they segment the whistle and use frequency offsets or time delays from the known baseline to encode data. Similarly, [3] modifies an existing scheme of bit encoding into dolphin whistles wherein the contour is segmented and each piece is formulated as an up-chirp or down-chirp based on the desired bit output.

Existing works also use a combination of model-based and recreation-based techniques, using a model that less strictly recreates a signal but more follows the general trend of its contour for guidance. [30] groups conventional signal models for marine mammal sounds into two categories, either consisting of weighted superposition of sinusoidal frequencies or a simple frequency-modulated system. An example of the former is found in [9], where the generated signal is of the form $s[n] = \sum_{r=1}^{R} a_r[n] \sin(2\pi\theta_r[n])$. This directly formulates the generated dolphin whistle as a sum of R harmonics, with amplitudes and phases represented by $a_r[n]$ and $\theta_r[n]$ respectively. The authors in [30] generate signals based on piecewise construction of short segments drawn from the two mentioned categories, choosing either sinusoidal or power frequency modulation depending on which better

matches the segment of whistle they are trying to mimic. Motivated by the desire for a novel form of covert UWAC, [18] aims to model a form of dolphin whistles where the parameters can be used to encode the desired information. They use a generalized frequency-modulated signal formula $s(t;\mathbf{b}) = A\alpha(t)\exp^{j2\pi(c\zeta(t/t_r)+f_0 t)}$, wherein the relevant parameters of amplitude, carrier frequency, chirp rate, and duration are used to encode covert communications. While the generated signals may be judged on their ability to resemble real whistles, the models developed are able to organically create these signals with varying levels of realism.

Evidenced by these generative methods, it is clear that the structure of dolphin whistles is not well-understood. While observations about the typical parameters of these whistles can be made (e.g. frequency range, duration, inflection points), there is little understanding about the correlation of these with any behaviours or environmental conditions. The mentioned works judge the ability of their generative methods either based on recreation of the desired whistle contour or a general assessment on appearance/sound. Additionally, there has been little integration of NNs into this topic of research. We speculate that this is at least partly due to the current gap in generative networks when using audio data; while many models are now able to create images that pass as real, audio data is still lacking. Furthermore, most audio work is centred on human voices, which have extremely different characteristics and requirements than dolphin whistles.

In this thesis, we focus on generating whistle contours in the image domain. This allows us to integrate NNs into the realm of whistle generation by creating the time-frequency shapes that would characterise the signal. This builds the first half of generating usable synthetic dolphin whistles, which is generating the contour; this generative process will be expanded and applied in Chapter 4. The models in this chapter are assessed through a combination of their respective loss values and visual quality of the produced samples.

3.2 Data

For expediency of the training process, the dimensionality of synthetic data as compared to what was using in Chapter 2 is halved; thus the whistle contours have spatial dimensions of 112×112. We are also only concerned with generating a single

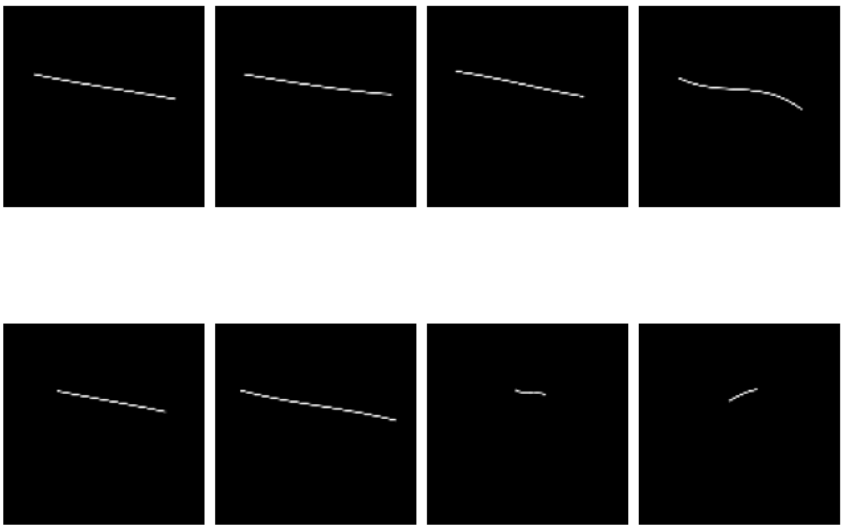


Figure 3.1: Eight examples of clean contours used for inputs in generative models.

channel of image data since using RGB spectrograms adds no discernible value.

The desired output of our generative NNs is a synthetic whistle free from any background noise. This firstly simplifies the training process since models are not required to generate signal and background; as well, this allows for generation of usable contours, regardless of what environment we are in. Since we want the generative models to create clean contours, we need training data of the same format. Therefore, the data from Chapter 2 cannot be used directly. Dataset A includes manually labelled time-frequency points for the positive samples, which were previously unused. This information can be used to recreate the whistle contour shapes; we follow the technique in [14] and use polynomial fitting (degree 3) on these points. This is considered a clean synthetic whistle, and images of the same likeness are the desired output of our generative models. While not a fully accurate reproduction of a whistle contour, this method offers an approximation that can be used as inputs for these generative models. Examples of these clean contours can be found in Figure 3.1.

3.3 Generative Adversarial Networks

This section covers the use of GANs to our contour generation task. We start by providing an overview of this model structure and the concepts behind its training process. Then, we outline the model architecture used. Finally, we apply this architecture to contour generation and assess its suitability.

3.3.1 Background

The overarching framework of a GAN system pits two models against one another: the generator (G) and the discriminator (D) [21]. While G attempts to create realistic samples from random noise inputs, D attempts to distinguish between true and fake samples. In the ideal situation, G advances to the point where D is unable to separate true from fake. D outputs predictions of a given sample belonging to the true data distribution, and the ideal GAN ends at the point where G is able to produce realistic data while D is constrained to 50% accuracy since it cannot distinguish between real and generated samples. Both G and D are constructed as neural networks, so the entire system can be trained using backpropagation.

In the GAN, we use random noise $\mathbf{z}$ of arbitrarily dimensionality sampled from $p_{\mathbf{z}}$ as the input to G. Generally, $\mathbf{z}$ will be a vector of empirically determined length n of values drawn from a normal Gaussian distribution, and so $\mathbf{z} \sim N(0, \mathbf{I}_n)$. On the other end, we have true data samples $\mathbf{x}$ which come from an unknown distribution $p_{\mathbf{x}}$. In image generation, each data sample is of shape $H \times W$, where often H is equal to W for simplicity. True data $\mathbf{x}$ is pitted against the result of our G network, which is $G(\mathbf{z})$. A visualisation of this layout can be found in Figure 3.2

D and G are two players in a minimax game, and the value function as defined in [21] is

$$\min_G \max_D V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{\mathbf{x}}}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}}[\log(1 - D(G(\mathbf{z})))]. \tag{3.1}$$

D trains using both components of the value function in Equation 3.1, while G only needs to consider the second term. In a training epoch for a GAN, we first obtain a batch of fake samples $G(\mathbf{z})$ labelled 0 and a batch of true samples $\mathbf{x}$ labelled 1. D is trained using this data's loss and backpropagated accordingly. Then, a new

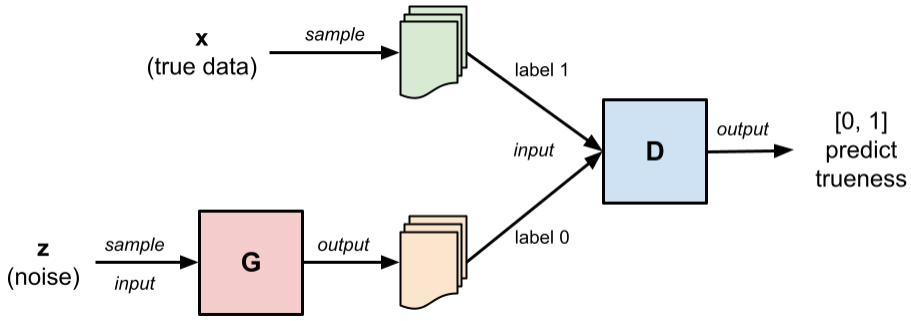


Figure 3.2: Illustrative flow of data through a GAN.

batch of fake samples is drawn and labelled as 1 in an attempt to assess G's ability to fool D (i.e. assess the realism of the generated samples). This loss is then used to train G.

3.3.2 Architecture

Our network architecture is founded in the Deep Convolutional Generative Adversarial Network (DCGAN) [44] architecture, with code based on PyTorch's tutorial at [28]. Prior to [44], there were difficulties encountered when attempting to generate larger-scale, high-resolution images using GANs. Often, the networks would be unable to produce realistic images and have non-informational loss values where a lower G loss did not necessarily correspond to better outputs. There are several architectural guidelines for creating DCGANs that are stable in training: replace pooling with strided convolutions, use of batch normalisation, elimination of most dense layers, using ReLU in the generator, and using Leaky ReLU in the discriminator [44].

In the DCGAN framework, D and G are made to mirror one another. D uses conventional convolution layers as described in Section 2.2.1. G, however, uses transposed convolutions, also known as fractionally-strided convolutions or deconvolutions. While referred to as a deconvolution, that is not the true operation since these layers are not the inverse of the convolution operations. Rather, the input is treated as a weight for the learned filter parameters, and this creates the resulting, typically upscaled output. A visual example of this transposed convolution operation can be found in Figure 3.3.

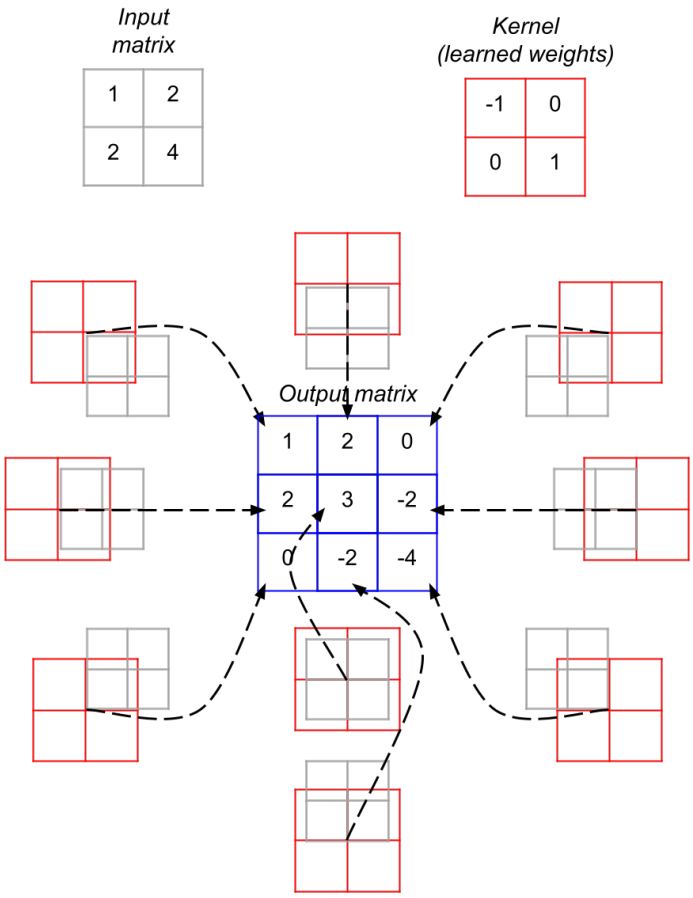


Figure 3.3: Transposed convolution result of a 2×2 input matrix with a 2×2 kernel resulting in a 3×3 output matrix. This example uses a stride of 1 and padding of 0.

Both models are primarily built on convolutional-type layers which change the data's spatial and channel dimensionailty as it progresses. For an input sample of size $(1 \times W \times H)$ being channels by width by height, D produces a single value output in the range $[0,1]$ indicating the probability of the input being a real sample. Our model G takes inputs of size $(n \times 1 \times 1)$ (i.e. there are n random variables drawn from $N(0,1)$). G's output is the same dimensionality as our discriminator's input, producing an image that is ideally as realistic as possible. Empirical testing showed that the most consistent results were produced when G had a final layer of

tanh rather than ReLU; as such, images used as input and produced as output in Section 3.3 were scaled to the range $[-1,1]$. The specific layers involved in both D and G can be found in Tables 3.1 and 3.2 respectively.

Layer Type	**Relevant Parameters**
Convolution	$ch_{in} = 1$ $ch_{out} = 16$ $k = 6$ $s = 2$
Leaky ReLU	$ns = 0.2$
Convolution	$ch_{in} = 16$ $ch_{out} = 32$ $k = 4$ $s = 2$
Batch Normalisation	$ch = 32$
Leaky ReLU	$ns = 0.2$
Convolution	$ch_{in} = 32$ $ch_{out} = 64$ $k = 4$ $s = 2$
Batch Normalisation	$ch = 64$
Leaky ReLU	$ns = 0.2$
Convolution	$ch_{in} = 64$ $ch_{out} = 128$ $k = 5$ $s = 1$
Batch Normalisation	$ch = 128$
Leaky ReLU	$ns = 0.2$
Convolution	$ch_{in} = 128$ $ch_{out} = 256$ $k = 5$ $s = 1$
Batch Normalisation	$ch = 256$
Leaky ReLU	$ns = 0.2$
Convolution	$ch_{in} = 256$ $ch_{out} = 1$ $k = 4$ $s = 1$
Sigmoid	–

Table 3.1: Architecture for D used in Section 3.3. Channels in for convolutional layer (ch_{in}), channels out for convolutional layer (ch_{out}), kernel for convolutional layer (k), stride for convolutional layer (s), negative slope for leaky ReLU (ns), channels in/out for batch normalisation layer (ch). Where not specified, default as provided by PyTorch is used.

Layer Type	Relevant Parameters
Transposed Convolution	$ch_{in} = Z$ $ch_{out} = 256$ $k = 4$ $s = 1$
ReLU	–
Transposed Convolution	$ch_{in} = 256$ $ch_{out} = 128$ $k = 5$ $s = 1$
Batch Normalisation	$ch = 128$
ReLU	–
Transposed Convolution	$ch_{in} = 128$ $ch_{out} = 64$ $k = 5$ $s = 1$
Batch Normalisation	$ch = 64$
ReLU	–
Transposed Convolution	$ch_{in} = 64$ $ch_{out} = 32$ $k = 4$ $s = 2$
Batch Normalisation	$ch = 32$
ReLU	–
Transposed Convolution	$ch_{in} = 32$ $ch_{out} = 16$ $k = 4$ $s = 2$
Batch Normalisation	$ch = 16$
ReLU	–
Transposed Convolution	$ch_{in} = 16$ $ch_{out} = 1$ $k = 6$ $s = 2$
Tanh	–

Table 3.2: Architecture for G used in Section 3.3. Number of random noise points drawn for input (Z), channels in for transposed convolutional layer (ch_{in}), channels out for transposed convolutional layer (ch_{out}), kernel for transposed convolutional layer (k), stride for transposed convolutional layer (s), channels in/out for batch normalisation layer (ch). Where not specified, default as provided by PyTorch is used.

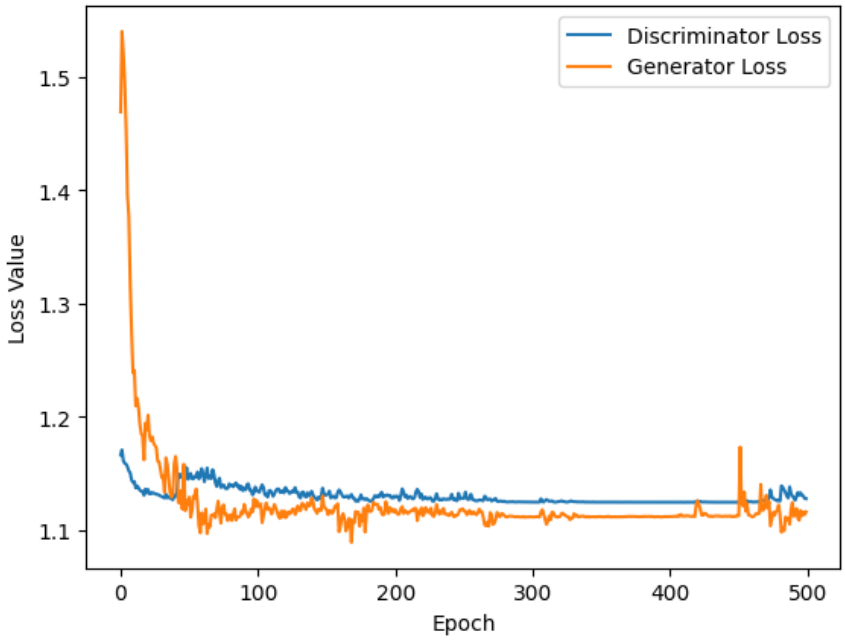


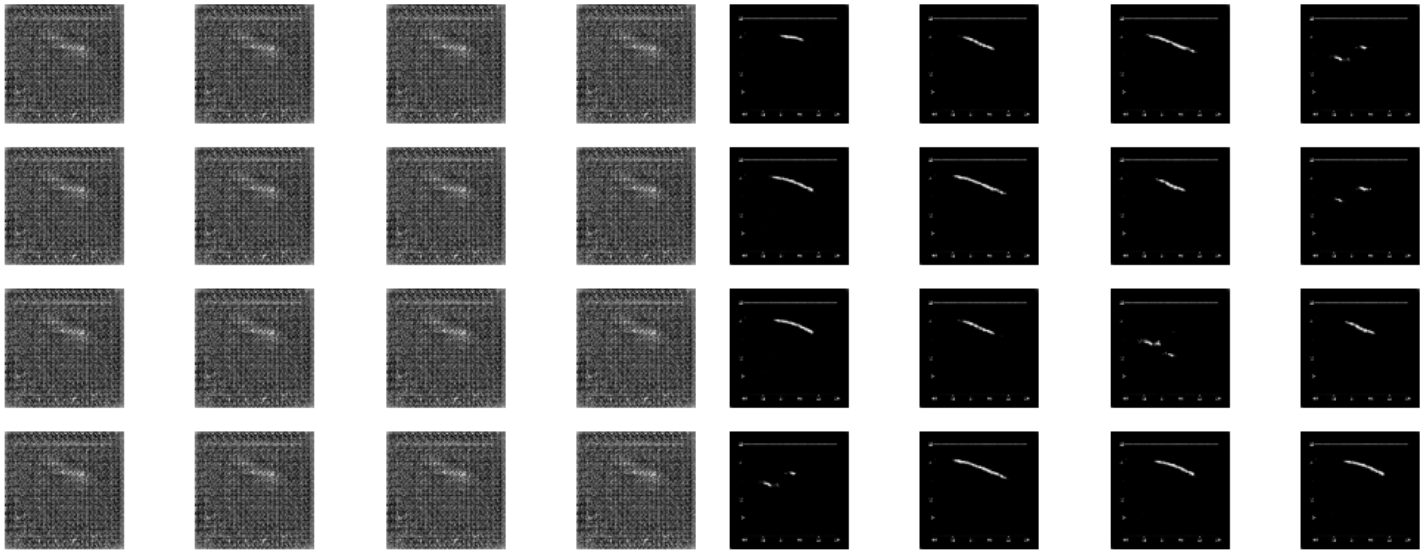
Figure 3.4: Learning curve for a GAN model with D and G architectures as indicated in Tables 3.1 and 3.2 respectively.

3.3.3 Results

Using the architecture as outlined in Section 3.3.2, hyperparameter tuning was conducted manually using grid search. The optimal results were based on the models' losses as well as visual assessment of produced images throughout, since the losses in a GAN system can be unstable and unclear. The system was allowed to train for 500 epochs, with checkpoints of the models taken every 25 epochs. For the best set of hyperparameters we were able to find, the learning curve with D and G losses can be found in Figure 3.4.

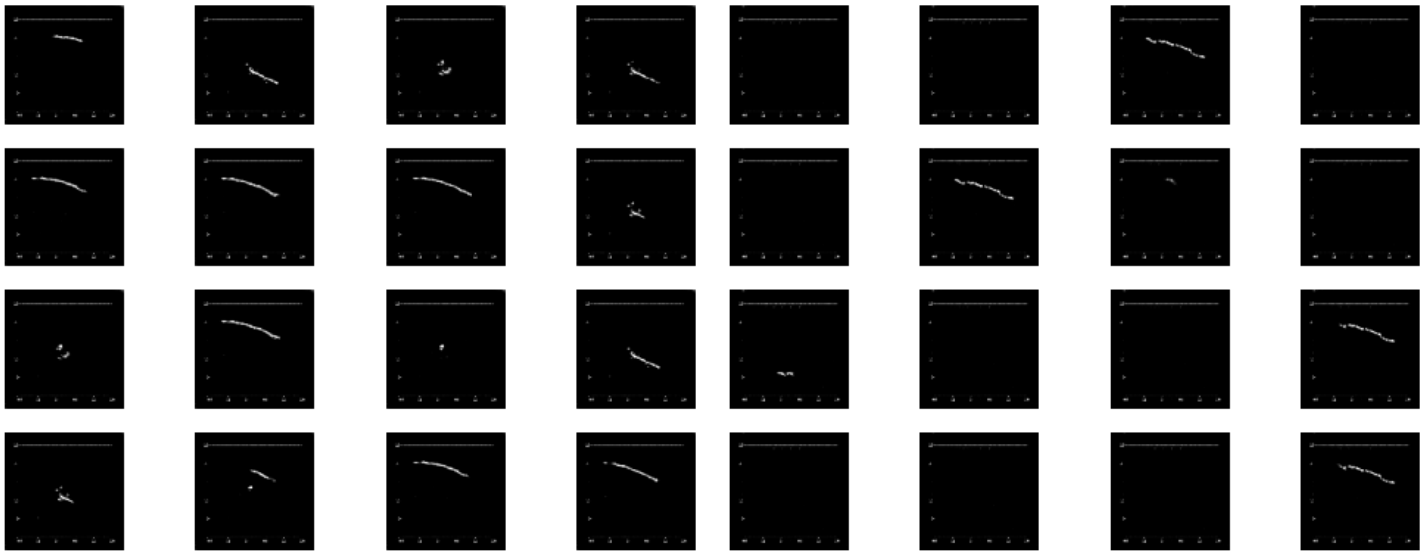
While the loss values seem to stabilise and indicate that the generator is performing well, this is not necessarily true when we visually assess the generated images. The outputs produced at select checkpoints corresponding to the learning curve in Figure 3.4 are shown in Figure 3.5. The images produced at all checkpoints can be found in Appendix E.

There are several issues with the images produced by the GAN models. First, it is difficult to directly correlate loss values with quality of image. In Figure 3.4,



(a) Epoch 1.

(b) Epoch 25.



(c) Epoch 125.

(d) Epoch 500.

Figure 3.5: Multiple samples of generated outputs of GAN at epochs 1, 25, 125, and 500.

it appears that the losses stabilise and become fairly consistent after approximately epoch 100. However, it is clear in Figure 3.5 that the results at epoch 500 are a downgrade from those at epochs 25 or 125. In Appendix E, it becomes even clearer that the later epochs do not necessarily have better results. As such, it can be difficult to save the appropriate best models. Second, there are artifacts in the perimeters of the generated images that do not correspond to the desired contour as was provided by inputs such as those in Figure 3.1. While these can be removed post-generation, this is a non-trivial adjustment. Third, the GAN system seems to be suffering from mode collapse. While there is some diversity in the images produced at epoch 125, this does not remain nor is it a consistent amount of diversity. This is well-known weakness of GANs, and a strong motivator for attempting a different style of generative model.

3.4 Denoising Diffusion Probabilistic Models

This section covers the use of DDPMs to our contour generation task. We start by providing an overview of this model structure and the concepts behind its training process. Then, we outline the model architecture used. Finally, we apply this architecture to contour generation and assess its suitability.

3.4.1 Background

DDPMs [25] use variational inference in a Markov chain to generate data samples. DDPMs define a forward process, where structured data becomes noise, and a backward process, where structured data is recovered from noise. GANs bridge the gap between noise input to structured output in one pass of the network; however, DDPMs achieve this gradually.

In the forward process of a DDPM, input data $\mathbf{x}$ is iteratively combined with Gaussian noise over T timesteps [25]. The strength of the noise is controlled over the timesteps with a series of parameters β_t. Thus, at each step, we have the conditional Probability Density Function (PDF)

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = N(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}). \tag{3.2}$$

This is a Markovian process, and so

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t-1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}). \tag{3.3}$$

The noise updates can become larger as time goes on, so $\beta_1 < \beta_2 < \ldots < \beta_T$. This variable can be learned but tends to be controlled by a schedule. Over the forward process, $\mathbf{x}_0$ loses its structure and becomes indistinguishable from random noise. Theoretically, the forward process would require that an initial image is distorted over T iterations with noise. Practically, reparameterisation allows $\mathbf{x}_t$ to be determined directly rather than necessitating a lengthy iterative calculation by

$$\mathbf{x}_t(\mathbf{x}_0, \varepsilon) = \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, \tag{3.4}$$

where $\varepsilon \sim N(\mathbf{0}, \mathbf{I})$, $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{s=1}^{t} \alpha_t$. Everything in the forward process is done without the use of a NN.

In the backward process, we attempt to recover $\mathbf{x}_{t-1}$ from $\mathbf{x}_t$. This step is done using a NN, which at every timestep is provided with the appropriate $\mathbf{x}_t$ and t, the latter of which can be used to find β_t. This NN is learning to undo the diffusion process. Similar to the forward process, this is a Markov chain with the Gaussian transitions

$$p(\mathbf{x}_{t-1}|\mathbf{x}_t) = N(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)), \tag{3.5}$$

where μ_θ and Σ_θ are learned and parameterised by some set of parameters θ. Both the forward and backward processes are depicted in Figure 3.6. Throughout the entire iterative process, $\mathbf{x}_t$ retains the same spatial dimensions.

While we are trying to obtain $\mathbf{x}_{t-1}$ from $\mathbf{x}_t$, the former is rarely predicted directly. Instead, [25] found it more effective to predict the noise that was added in that timestep. This would imply that both $\mu_\theta(\mathbf{x}_t, t)$ and $\Sigma_\theta(\mathbf{x}_t, t)$ from Equation 3.5 are model outputs. However, [25] also found that it was sufficient to only predict $\mu_\theta(\mathbf{x}_t, t)$. $\Sigma_\theta(\mathbf{x}_t, t)$ is treated as an untrained constant that can be produced given time t by setting it equal to

$$\Sigma_\theta(\mathbf{x}_t, t)) = \sigma_t^2 \mathbf{I}. \tag{3.6}$$

When using $\mathbf{x}_0 \sim N(\mathbf{0}, \mathbf{I})$, [25] finds that $\sigma_t^2 = \beta_t$ is optimal.

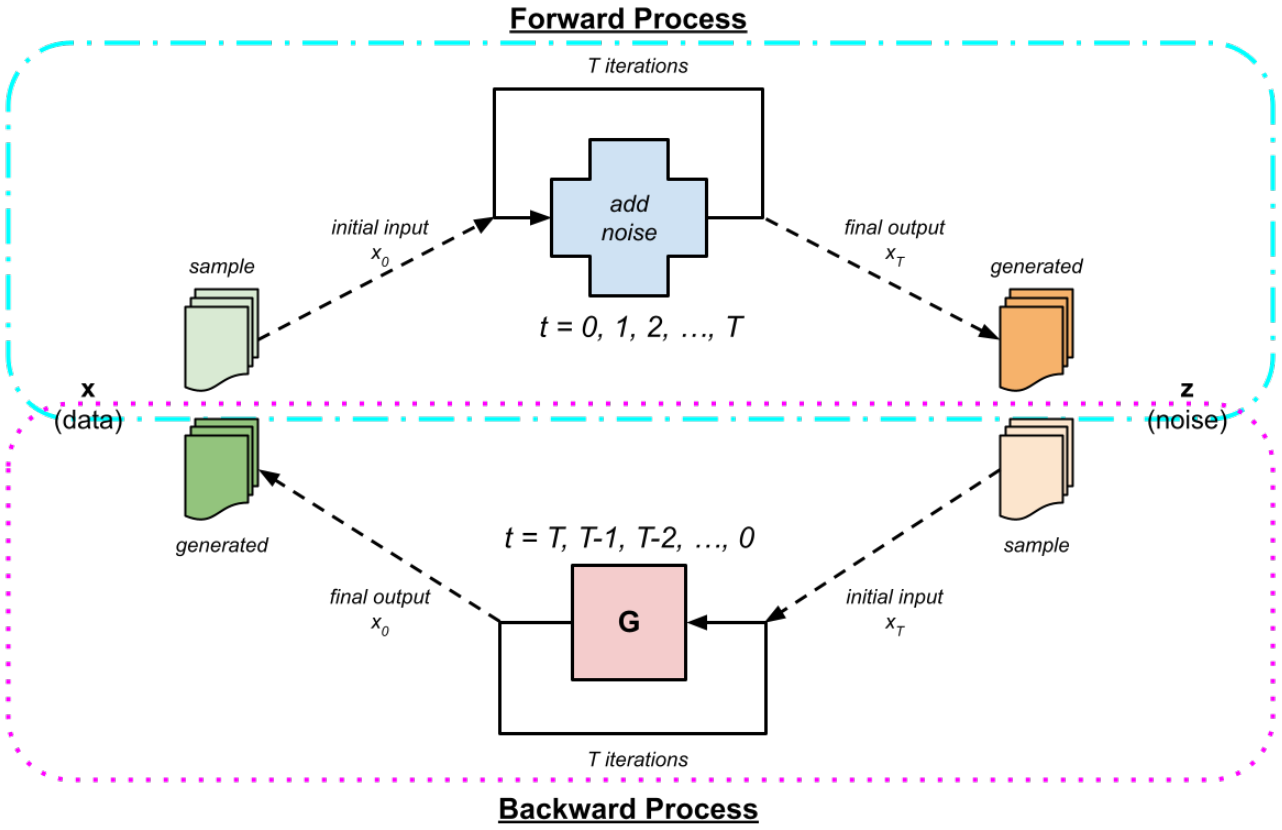


Figure 3.6: Illustrative flow of data through a DDPM. Both forward and backward processes are depicted. Only the red squared labelled G is a NN in this process.

DDPM training is done by optimising the variational bound on the negative log likelihood, the full derivation of which can be found in [25]. While the model can be made to predict μ_θ directly, this is often not the case. Reparameterising the relevant equations changes the target output of the model to predicting the noise ε from a given $\mathbf{x}_t$. This results in gradient descent being taken on

$$||\varepsilon - \varepsilon_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\varepsilon, t)||^2 . \tag{3.7}$$

When training, a random batch of real images is taken and a timestep t is drawn

randomly. These values are used for Equation 3.7, and this is repeated until the model converges.

To generate a sample (i.e. the backward process), $\mathbf{x}_T$ is drawn from $N(\mathbf{0}, \mathbf{I})$ at the same dimensionality of the desired image. This is the assumed final product of a forward diffusion process. Over T time steps, $\mathbf{x}_t$ is updated as

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \varepsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}, \tag{3.8}$$

where

$$\mathbf{z} \sim N(\mathbf{0}, \mathbf{I}) \quad \text{if } t > 1, \text{ else } \mathbf{z} = \mathbf{0}. \tag{3.9}$$

One notable benefit of DDPMs is that they do not suffer from mode collapse. In the case of GANs, the adversarial component can often drive the generative model to produce only one type of output, which can accomplish the goal of fooling the discriminator well. Additionally, the training process for a DDPM tends to be more stable than that of a GAN, which is hampered by the adversarial models working against one another.

3.4.2 Architecture

DDPMs have a semi-unique property that the model's input and output will have the same spatial dimension. The model architecture will need to account for this, and oftentimes U-Net [46] is utilised. U-Net expands on NNs that are fully convolutional, aiming to create a precise segmentation model which is able to work with a smaller training dataset that previous ones. A typical CNN downsizes the input image due to the nature of the convolution operation. This is retained in the U-Net structure, but [46] supplements this by a second half which performs upsampling. This results in a mirrored network structure which compresses the image before increasing its resolution back to the original dimensions. Additionally, U-Net makes use of skip connections which directly feed data from earlier layers into layer ones, bypassing the intermediate layers. The original U-Net architecture from [46] can be found in Figure 3.7.

Our DDPM model and training process is built on code found at [43]. This architecture incorporates time and context embeddings into the training process.

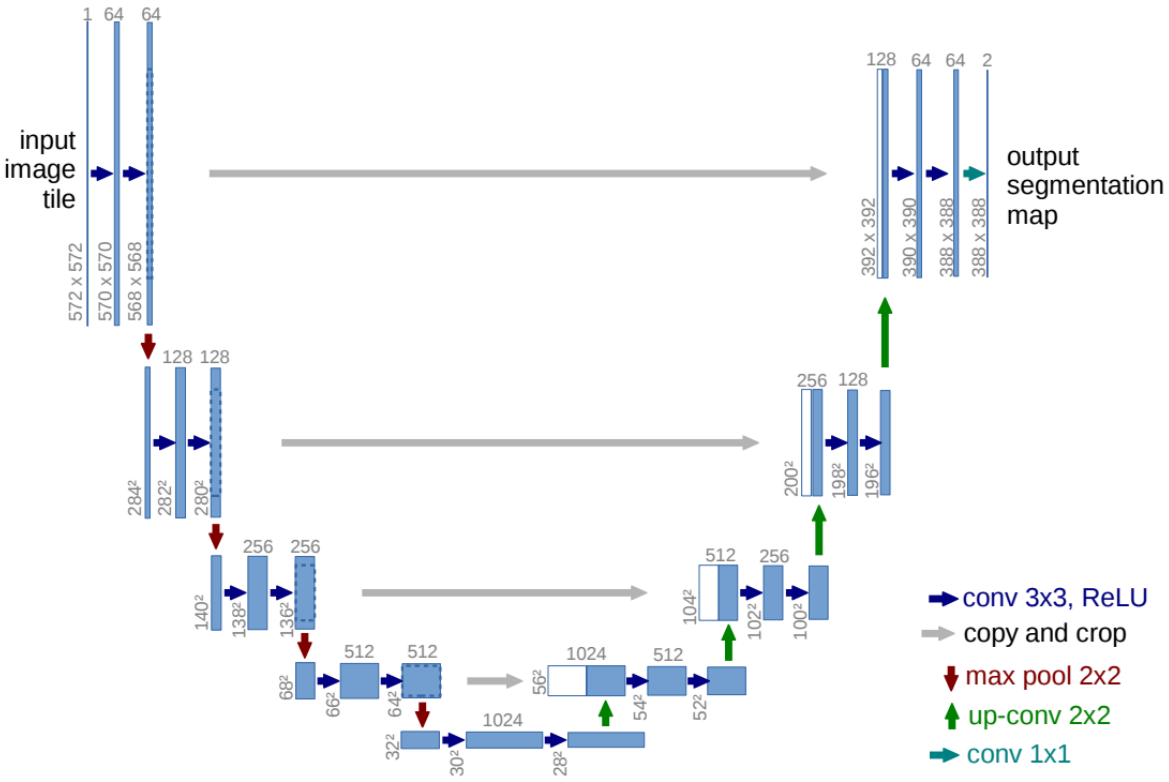


Figure 3.7: Architecture of U-Net model as illustrated in [46, Figure 1].

The former is used to make the model aware of how far along it is in the generation process, relative to the total number of time steps. The latter is used when classes are involved, guiding the model to conditional generation. Throughout the model, a parameter known as "number of features" (n_{feat}) or an integer multiple of this value is used to dictate the number of input/output channels. There are several components in the DDPM we use: residual convolution blocks (Table 3.3), U-Net downscaling (Table 3.4), U-Net upscaling (Table 3.5), and embedding block (Table 3.6). As a whole, the model is structured using these components as seen in Table 3.7.

In our model, the number of features n_{feat} is implemented as a hyperparameter. U-Net upscaling layers have increased input channel dimensions due to the addition of time and class embeddings. For our data in this chapter, $ch_{in} = ch_{out} = 1$ since we are using 1-channel clean whistle contours.

Sub-block	Layer Type	Relevant Parameters
1	Convolution	$ch_{in} = ch_{in}$ $ch_{out} = ch_{out}$ $k = 3$
	Batch Normalisation	$ch = ch_{out}$
	GeLU	–
1	Convolution	$ch_{in} = ch_{out}$ $ch_{out} = ch_{out}$ $k = 3$
	Batch Normalisation	$ch = ch_{out}$
	GeLU	–

Table 3.3: Composition of a residual convolution block. When residuals are activated, the output of this block is a scaled sum of either input and sub-block 2 output or sub-block 1 and sub-block 2 outputs. Otherwise, the model simply passes through these layers sequentially. Requires ch_{in} and ch_{out} parameters.

Layer Type	Relevant Parameters
Residual convolution block	$ch_{in} = ch_{in}$ $ch_{out} = ch_{out}$
Max pooling	$k = 2$

Table 3.4: Composition of a U-Net downscaling block. Does not use residuals for residual convolution block. Requires ch_{in} and ch_{out} parameters.

Layer Type	Relevant Parameters
Transposed Convolution	$ch_{in} = ch_{in}$ $ch_{out} = ch_{out}$ $k = 2$ $s = 2$
Residual convolution block	$ch_{in} = ch_{out}$ $ch_{out} = ch_{out}$
Residual convolution block	$ch_{in} = ch_{out}$ $ch_{out} = ch_{out}$

Table 3.5: Composition of a U-Net upscaling block. Does not use residuals for residual convolution block. Requires ch_{in} and ch_{out} parameters.

Layer Type	Relevant Parameters
Dense	$dim_{in} = dim_{in}$ $dim_{out} = dim_{out}$
GeLU	–
Dense	$dim_{in} = dim_{out}$ $dim_{out} = dim_{out}$

Table 3.6: Composition of an embedding block. Requires dim_{in} and dim_{out} parameters.

Component Type	Relevant Parameters
Residual convolution block	$ch_{in} = ch_{in}$ $ch_{out} = n_{feat}$
U-Net Down	$ch_{in} = n_{feat}$ $ch_{out} = n_{feat}$
U-Net Down	$ch_{in} = n_{feat}$ $ch_{out} = 2n_{feat}$
Average pooling	$k = mid$
GeLU	–
Transposed Convolution	$ch_{in} = 2n_{feat}$ $ch_{out} = 2n_{feat}$ $k = mid$ $s = mid$
Group normalisation	$group = 8$ $ch = n_{feat}$
ReLU	–
Embed (time)	$dim_{in} = 1$ $dim_{out} = 2n_{feat}$
Embed (class)	$dim_{in} = n_{classes}$ $dim_{out} = 2n_{feat}$
U-Net Up	$ch_{in} = 4n_{feat}$ $ch_{out} = n_{feat}$
Embed (time)	$dim_{in} = 1$ $dim_{out} = n_{feat}$
Embed (class)	$dim_{in} = n_{classes}$ $dim_{out} = n_{feat}$
U-Net Up	$ch_{in} = 2n_{feat}$ $ch_{out} = n_{feat}$
Convolution	$ch_{in} = 2n_{feat}$ $ch_{out} = n_{feat}$ $k = 3$
Group normalisation	$group = 8$ $ch = n_{feat}$
ReLU	–
Convolution	$ch_{in} = n_{feat}$ $ch_{out} = ch_{out}$ $k = 3$

Table 3.7: Architecture for DDPM used throughout Section 3.4. Parameter n_{feat} is a tuneable parameter. Parameter *mid* is set to the dimension of the data at that time, such that the average pooling results in a spatial dimension of 1×1. When the residual convolution block is used on its own, the residuals are activated; otherwise, in the U-Net upscaling/downscaling component, they are not.

3.4.3 Results

Hyperparameter tuning and model selection were conducted similarly to Section 3.3.3. While DDPM losses tend to be stable, it can be difficult to select a best model iteration or cross-compare between different training sessions; the lowest loss does not always correspond to the best output images. This is likely due to the sparsity of the data and thus how small the general differences between the generated and predicted noise are, resulting in a consistently low loss value even when the gener-

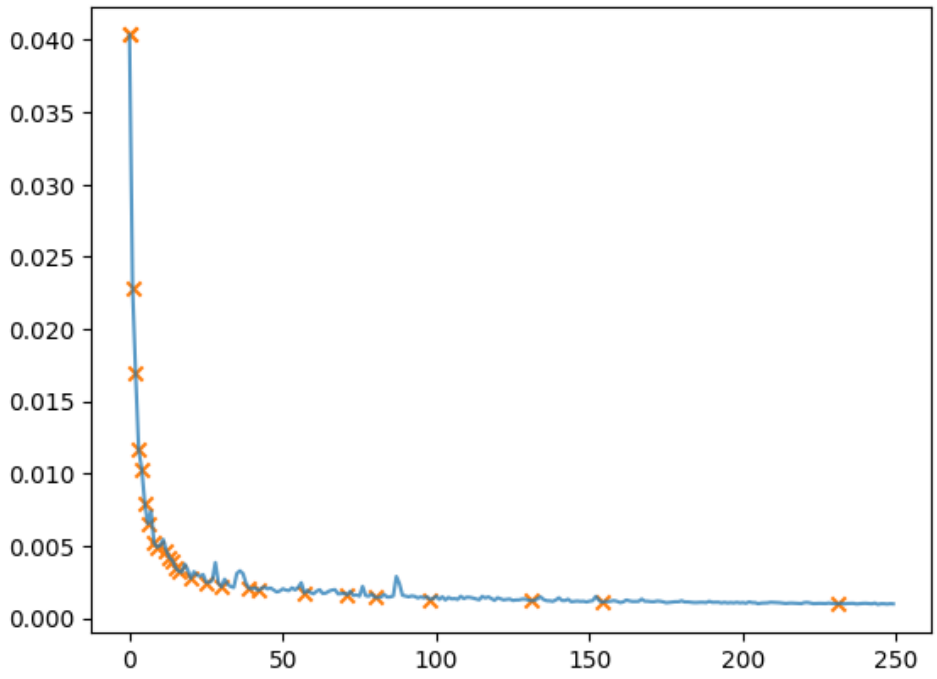
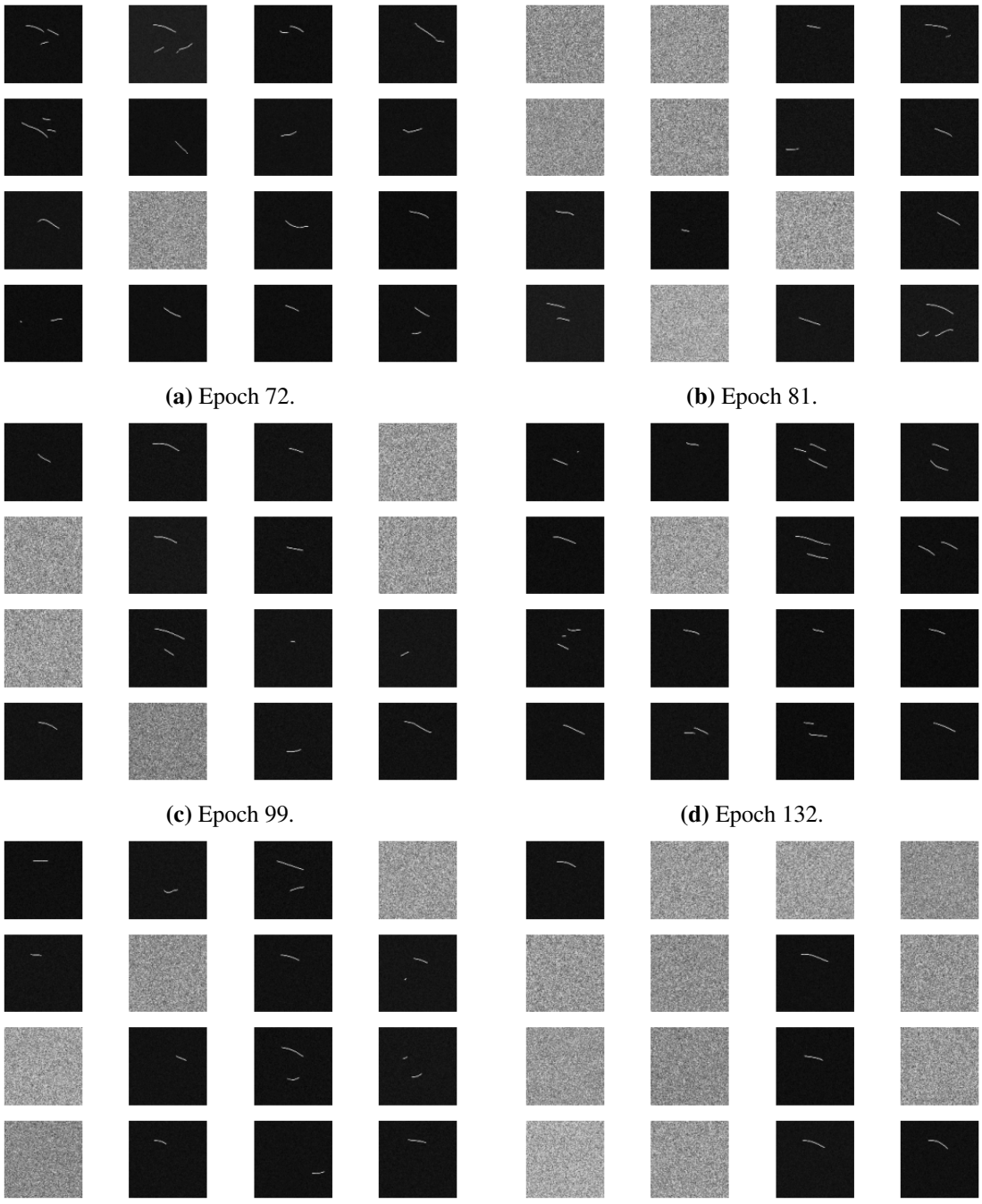


Figure 3.8: Learning curve for a DDPM. Orange "x" markers indicate epochs at which a new lowest loss value was found.

ated image is not ideal. However, because the loss values are stable, they can still be used to guide the selection process. The models were allowed to train for 250 epochs, and checkpoints of model parameters and generated images at the moment in time were taken every time the loss reached a new low. Loss values over the training session for our model can be found in Appendix F.

From these values, we utilise the last six markers (epochs 72, 81, 99, 132, 155, 232) as the desired checkpoints, since these loss values are all very low and seem to occur when the loss graph has effectively plateaued. Using these checkpoints, we generate samples as found in Figure 3.9. Samples produced by earlier epochs can be found in Figure F.1.

There are no noticeable artifacts in the background/edges of the images, other than a faint speckling of noise that can be easily removed with pixel-value thresholding unlike the corner traces left by GAN. Additionally, the origin of these is more understandable, as they result from the process of image generation; in the



(a) Epoch 72.

(b) Epoch 81.

(c) Epoch 99.

(d) Epoch 132.

(e) Epoch 155.

(f) Epoch 232.

Figure 3.9: Multiple samples of generated outputs for DDPM at epochs 72, 81, 99, 132, 155, and 232.

GAN, there is more unknown to the process that can be difficult to debug and remove. An outstanding issue with the DDPM that we notice is some samples not producing usable data and remaining as random noise. This is similar in principle to the cases in our GAN where the network would produce nothing but background artifacts. Both cases are easily detectable, however, and they can be discarded. The DDPM model is also prone to generating samples with multiple whistles rather than one. This is an interesting variation on the training data, since no provided samples have multiple distinct whistles. However, there are cases in the true positive samples – as well as in real life – when multiple whistles can overlap in time and produce something similar. As such, this does not provide as large of an issue as the mode collapse in our GAN. Based on these assessments and the general stability of training, we utilise this trained DDPM for Chapter 4 rather than the GAN. Specifically, we use the models saved at epoch 132, since this iteration produces the most consistent results with the least number of unusable samples.

Chapter 4

Variation Generation

In Chapter 3, the contours generated by our NNs differ from true whistles in several ways. The most visually obvious is that they do not vary over the time-frequency profile. Primarily, these variations appear through width and intensity. Contour width refers to the frequency band that the contour occupies (i.e. how many pixels in height the whistle occupies at each point in time). Contour intensity refers to the actual pixel values of the contour (i.e. how "bright" the line is which may vary along the frequency-axis even at the same point in time). While there are reasons that exist for these variations – physical and biological – these are not well-explored or understood, thus we are limited to mimicking existing data.

In this thesis, synthetic sample generation occurs in two steps. First, we generate a simple shape of constant width/intensity. These outputs are called "contours" and the results are as found in Chapter 3. Second, we generate variations for these contours that change the width/intensity at points throughout their duration. This is primarily controlled by a set of variation parameters, and we call these "synthetic whistles". Previously unseen negative samples are used as "backgrounds" to these synthetic whistles, and when combined in a manner to be explained in Section 4.1.3, we have the final product called a "synthetic sample".

This chapter continues with the second half of whistle generation in the image domain. We begin with details about the variation parameters and how they affect the synthetic whistle's appearance. Then, we assess several scoring metrics that can be used to quantitatively judge how closely the synthetic samples resemble

our positive samples. Finally, we show the optimised sets of parameters that were found and the resulting whisles of using these modifiers before applying generated synthetic samples to an example application.

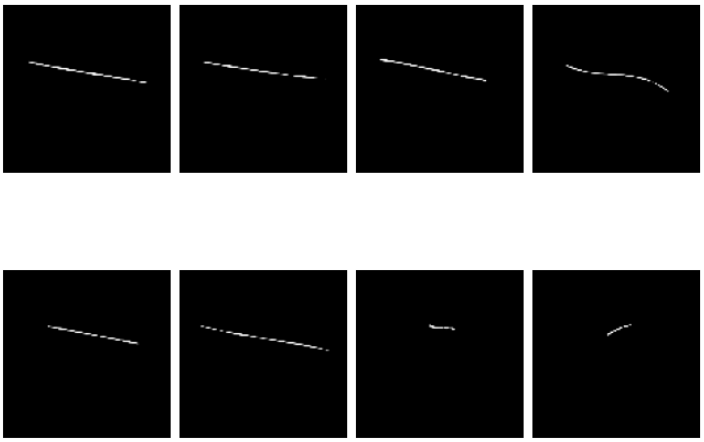
4.1 Variation Parameters

The methods we use to create these variations is primarily random with some guidance based on visual assessment compared to positive samples. This is mainly due to the fact that are no definitive answers on how the width/intensity variances of these whistles come about, thus we are limited to using comparisons between synthetic and real samples. The specifics of how we decide on the optimal parameters is outlined in Section 4.2. Throughout this section, we discuss the parameters used to alter the appearance of these contours in width and intensity. These are broken into four categories which will be elaborated on in further detail: width, intensity, relative strength, and softening.

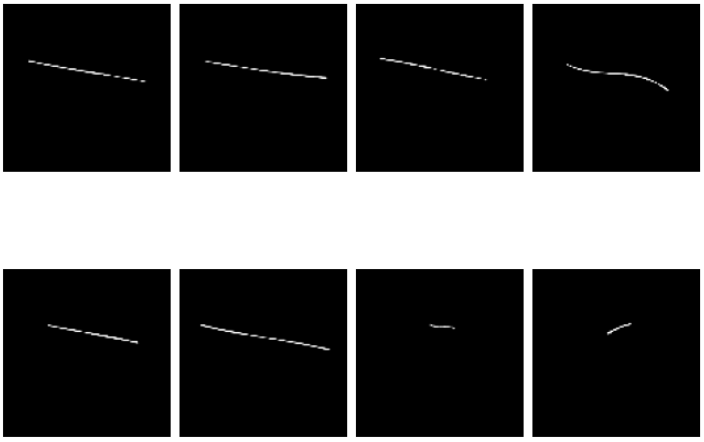
4.1.1 Width

The width of whistle contours can vary little or a moderate amount depending on the sample, but it is rarely consistent throughout the entire signal. In particular, the contours tend to taper off on either end (i.e. start and end). Whether this is due to the frequencies involved in the signal or the strength of its emission is unknown, but it is visually expressed in contour width.

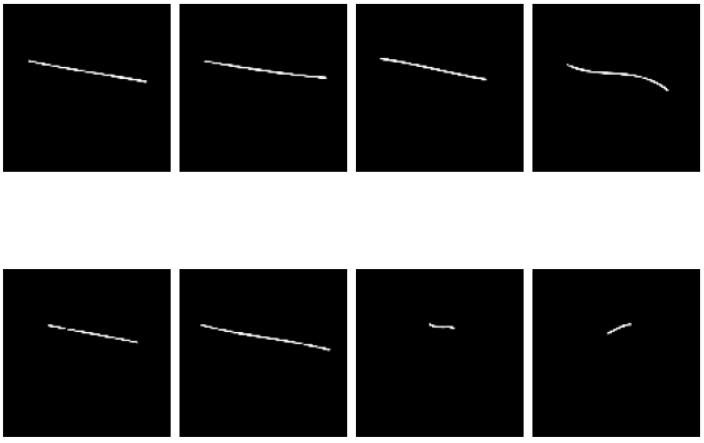
For our purposes, width of a whistle is represented as a random variable. A series of width values is produced over the timespan (image width) where the whistle is active; these control the frequency range (image height) that the contour will occupy at each moment in time. Width values are drawn from $N(\mu_{width}, \sigma_{width})$, with these two parameters controlled by the optimisation process. Additionally, mean filtering is used to smooth the widths so they are not as drastically different between consecutive values. This adds a third parameter k_{width} to represent the mean filter's kernel length. The width values are also constrained by a maximum value that is determined based on what seemed visually coherent with positive samples from dataset A. Examples of the same contour being varied using different sets of parameters are shown in Figure 4.1.



(a) Changed σ_{width}.



(b) Changed σ_{width} and k_{width}.



(c) Changed μ_{width}, σ_{width}, and k_{width}.

Figure 4.1: Different sets of width parameters resulting in different synthetic contours (8 each), all shown with no background.

4.1.2 Intensity

Over the course of the signal, its strength will also change. One factor would be in the signal itself, as the transmission volume may increase or decrease. This, however, is an unknown quantity and will not be considered. Another factor is the channel that the signal is received through, which in the case of dolphin whistles is an oceanic environment. Many factors can affect how sound propagates through the water, such as the physical surroundings (e.g. sea floor, depth, salinity, temperature) or non-constant conditions (e.g. weather, season) [10]. The end results can be signal scattering, absorption, and reflection which will change how the signal is received at an endpoint. As well, signals travelling underwater will be subject to multipath propagation, which can also contribute to signal fading [10]. For our work, we use Bellhop [41] to simulate the effects that an underwater channel would have on a whistle signal's intensity. Bellhop is a beam tracing model that can be used to simulate acoustic pressure in ocean environments, and it can generate a variety of outputs. As it may not be practical to obtain the oceanic environment for a location every time we wish to use this generation process, we instead use an example one. This provides the bathymetry for an environment that is realistic, if not exactly accurate to the conditions from our datasets.

We use Bellhop's ability to generate impulse responses to determine what distortions an underwater environment would impart upon a whistle contour. These responses are converted from time to frequency domain using a DFT. Multiple responses are used per contour to represent how the channel would affect the whistle signal at every point in time. When these are put together, we refer to this as an intensity "mask", which is overlaid with the baseline contour to allow for intensity variations. The original spectrograms have a dimensionality of 278×513, which is used for this portion. Thus, we create 278 arrival impulse responses for each mask, and each individual response is ensured to have a length of 513 when a DFT is applied to it. While this process does not fully encapsulate how the channel distortion would affect a dolphin whistle, it is able to modify the intensity in a manner that is visually satisfactory and based in a realistic foundation.

These channels are generated based on randomised locations, with certain limitations on potential positioning to be realistic. A receiver (i.e. hydrophone) and

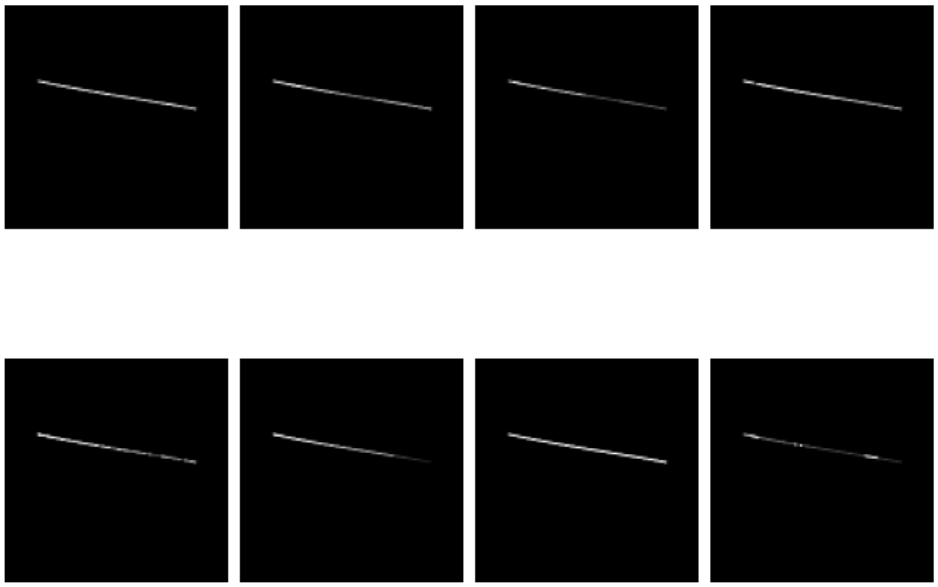
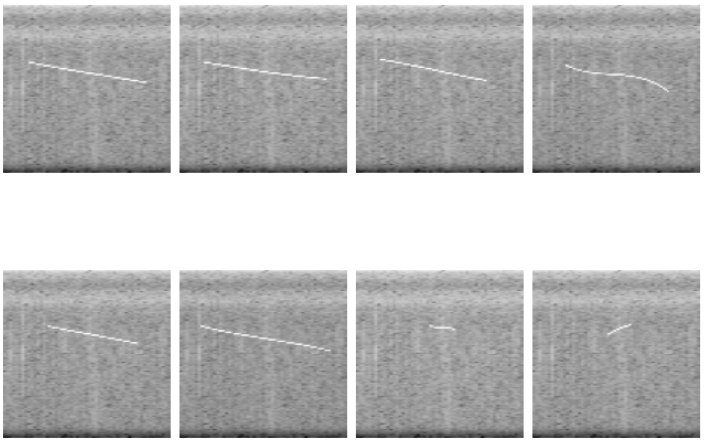


Figure 4.2: Same whistle contour overlaid with differing masks, shown with no background.

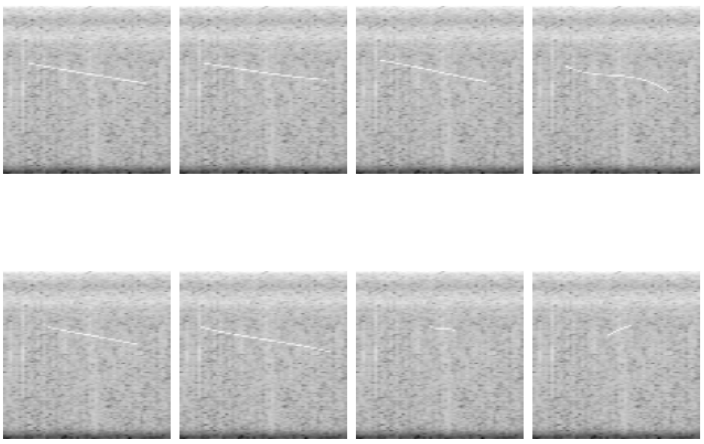
a sound source (i.e. dolphin) are initialised in Bellhop. The source has its position updated for every column in the mask that is generated, based on speed and direction. These position updates are based on viable speeds that a dolphin could move at, biased to continue traveling in the same direction as previous but with potential for change. This allows a simple but semi-realistic simulation of how the channel may vary due to changes in the positioning between the two. No adjustable parameters are included for the intensity factor; rather, hundreds of masks are generated and randomly multiplied with a contour. Examples of the same contour with varying intensity masks is shown in Figure 4.2.

4.1.3 Relative Strength

It is easiest to consider both the synthetic whistle and background as their own images normalised $[0,1]$. This allows the combination of the two to be controlled with a single parameter that represents the relative strength of the whistle. In essence, we have $\mathbf{X}_c$ for the synthetic whistle and $\mathbf{X}_b$ for the background. The variable s_w



(a) $s_w = 0.50$.



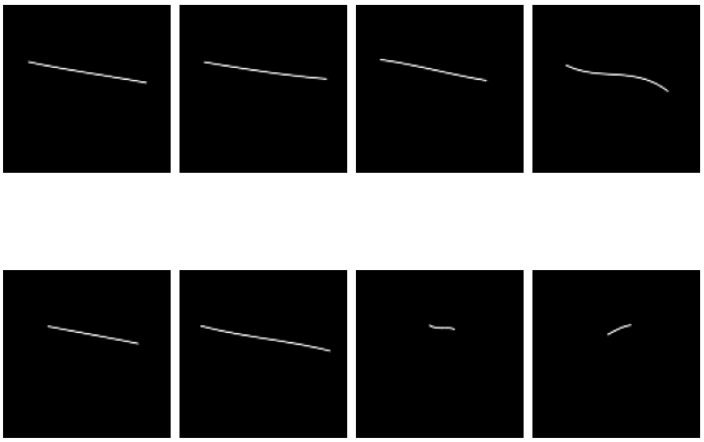
(b) $s_w = 0.25$.

Figure 4.3: Same whistle contours overlaid with differing relative strength values, shown on same background for all contours.

controls the strength of the whistle, and we have a final $\mathbf{X}_s$ synthetic sample as

$$\mathbf{X}_s = s_w \mathbf{X}_c + \mathbf{X}_b. \tag{4.1}$$

$\mathbf{X}_s$ must be normalised between $[0,1]$ to remain consistent with our positive samples. Examples of the same contour being generated using different sets of parameters are shown in Figure 4.3.



(a) α_{soft} True and small σ_{soft}.

(b) α_{soft} True and large σ_{soft}.

Figure 4.4: Different sets of softening parameters resulting in different synthetic contours (8 each), all shown with no background.

4.1.4 Softening

A final pair of parameters is included to "soften" the edges of the contour. The first parameter controls whether or not this process is utilised at all, implemented as a True/False switch we will refer to as α_{soft}. This softening is accomplished using a Gaussian filter, which replaces a target pixel with a linear combination if its neighbours. This is controlled by several factors, and we choose to vary the σ_{soft}, which dictates how much influence the neighouring pixels have on our target. Examples of the same contour being generated using different softening parameters is shown in Figure 4.4.

4.2 Optimisation of Parameters

In this section, we outline how the parameters mentioned in Section 4.1 are optimised to produce more realistic synthetic whistles. The end goal is to have synthetic samples which are indistinguishable from the positive samples we already have.

4.2.1 Scoring Metrics

A difficulty encountered in image generation NNs is the ability to quantitatively define how well the output images resemble the target dataset. Images are often easy for humans to visually judge, but this does not easily translate into a scoring system. There have been many proposed methods, all of which target different criteria and have varying benefits or drawbacks. [7] judges the efficacy of many existing evaluation metrics for GANs. The primary and most important quality for these metrics is that they should favour models that generate high fidelity and diverse samples, in a manner that is coherent with human judgement. Several specific metrics discussed in [7] will be mentioned in this section, as well as the ones we elected to use. When discussing scoring metrics, we may refer to "true" and "generated" samples as this is the terminology used in the context where these are most often seen. For our case, these correspond to "positive" and "synthetic" samples.

An intuitive standard is to utilise the probability densities of true and generated data. On the simplest level, the two probability densities calculated with finite numbers of samples can be directly compared using methods such as Kullback Leibler or Jensen Shannon divergence. [7] states that this technique has been questioned for suitability in GANs, and instead mentions the use of the generated samples as centroids in a Gaussian mixture. This can then be used in Parzen window density estimation to compute log-likelihood of true data from a test set belonging to this density. The largest drawback of this family of metrics is that they tend to be uninformative about the actual quality of generated images. They do not inform on diversity and learning capability of the model, and can potentially favour models which simply replicate training data.

Inception score and associated variants are some of the most commonly seen

metrics for GANs. These metrics are aptly named for the pre-trained NN they use in obtaining the score: Inception Net trained on ImageNet samples. The original inception score measured average divergence between conditional label distribution $p(y|\mathbf{X})$ and the marginal distribution $p(y)$. While this metric was better able to correlate high-quality and diverse images with better scores, they suffer similar drawbacks as using probability densities alone. Another popular, related metric is Frechet Inception Distance, which utilises intermediate layers of Inception Net although any CNN would suffice. The embedding layer is assumed to be a multivariate Gaussian distribution, and thus the mean/covariance are estimated for generated and real data samples. Frechet distance is calculated between these distributions, and in general [7] finds that this metric performs well. It also, however, makes the sometimes-false assumption that a Gaussian distribution can accurately represent the intermediate embeddings. While these metrics using Inception Net seem promising, works such as [13] establish flaws in their usage. When calculated over a finite set of samples, which is by necessity what must be done, the result does not represent a true or accurate assessment of the proper score. This difference also varies depending on the model used. In conjunction, this makes both scores flawed in their ability to properly assess generative performance across different models. Since we are attempting to find the most realistic set of parameters for data generation, this by necessity requires that we would be training models on different data and creating different models.

[7] also mentions the utilisation of classifier two-sample tests, which simply attempt to assess if two samples are drawn from the same distribution. A holdout set of true samples is pitted against generated samples by training a discriminator to differentiate between the two. While this is classically measured by the final model's discrimination accuracy, [7] also mentions normalised relative discriminative score which is focused on the idea that more epochs are required to distinguish between true and generated samples if the latter are realistic. Additionally, GANs can often be used to generated training samples to enhance a dataset, and this can be seen as a way to determine the diversity of generated samples. A model trained on generated samples – alone or mixed with true samples – can be assessed on a test set of true samples. These metrics are all dependent on the models at hand, but they are intuitively driven by the idea that a better generative model should result

in certain results for other, related tasks (i.e. poor discrimination with true samples, better classification when utilised as augmentation).

In our implementation and inspired by these metrics, we use NN-driven approaches. In particular, we consider two differing metrics scored each in two different ways, all with the goal of creating synthetic samples that are indistinguishable from – or at least similar to – positive samples.

First, we base performance on how well a NN is able to discriminate between positive and synthetic samples, with the ideal state being that the model is unable to converge. This is done in a similar fashion to the whistle detection conducted in Chapter 2, only we replace negative samples with synthetic ones. While we could theoretically create a 3-class classifier (positive, negative, synthetic), we instead chose to use only two classes (positive, synthetic) in hopes of eliminating potential confounding factors and creating the strictest test. We refer to this generally as the "discrimination" metric, and positive samples from dataset A are used. The first way this is evaluated is the intuitive metric of minimising the NN model's performance, which we choose to be the test accuracy. Therefore, we want the Synthetic Discrimination Accuracy (SDisA) score to be as close to 50% as possible. The second way this is evaluated is to measure how long the model requires for convergence. Thus, we want to maximise the Synthetic Discrimination Epochs (SDisE) score.

Second, we base performance on how well a model trained on synthetic and negative samples is able to label a real dataset. Ideally, if synthetic samples are realistic enough, a NN trained using no positive samples and only synthetic samples should still perform comparable to a model trained on positive samples. This is always assessed on samples that have not been seen in the training process; therefore, we use all the positive samples from dataset A and an equal number of unseen negative samples from dataset A. The first method labels only positive samples, and we wish to maximise the Synthetic Detection Positive Accuracy (SDetPA). The second method uses both positive and negative samples in the unseen set, and we wish to maximise the overall Synthetic Detection Accuracy (SDetA).

Opt. Metric	Nth Best	Score	SDisA	SDisE	SDetPA	SDetA
SDisA	1	56.45	96.56	146	23.43	61.57
	2	56.45	96.56	285	33.43	66.52
	3	56.45	96.67	197	100.0	50.00
SDisE	1	290	97.63	212	40.19	69.57
	2	290	97.10	164	14.00	56.86
	3	287	89.78	191	28.10	63.67
SDetPA	1	28.69	95.38	207	33.81	66.57
	2	26.83	89.25	231	35.14	67.33
	3	24.64	56.45	50	47.43	72.86
SDetA	1	57.27	99.57	63	2.29	51.14
	2	57.08	99.25	159	11.33	55.62
	3	57.06	99.46	92	6.86	53.38

"Opt. Metric" indicates the metric that was used to obtain the row's parameters.

"Nth Best" indicates the ranking in the metric of the row's parameters.

Table 4.1: Top three parameter set results based on each metric are shown, and their scores for all metrics are presented.

4.2.2 Results

As mentioned in Section 4.1.2, intensity masks are created and randomly combined with contours. The contours used in this section are the same as input data for our DDPM (i.e. polynomial-fitted). These contours are modified by variation parameters in the manners specified in Section 4.1. When using polynomial-fitted contours and variation parameters directly, we deem the resulting synthetic samples as "procedurally-generated". The optimisable parameters are μ_{width}, σ_{width}, k_{width}, s_w, α_{soft}, and σ_{soft}. Using these parameters assessed with the four metrics explained in Section 4.2.1, we obtain the results in Table 4.1.

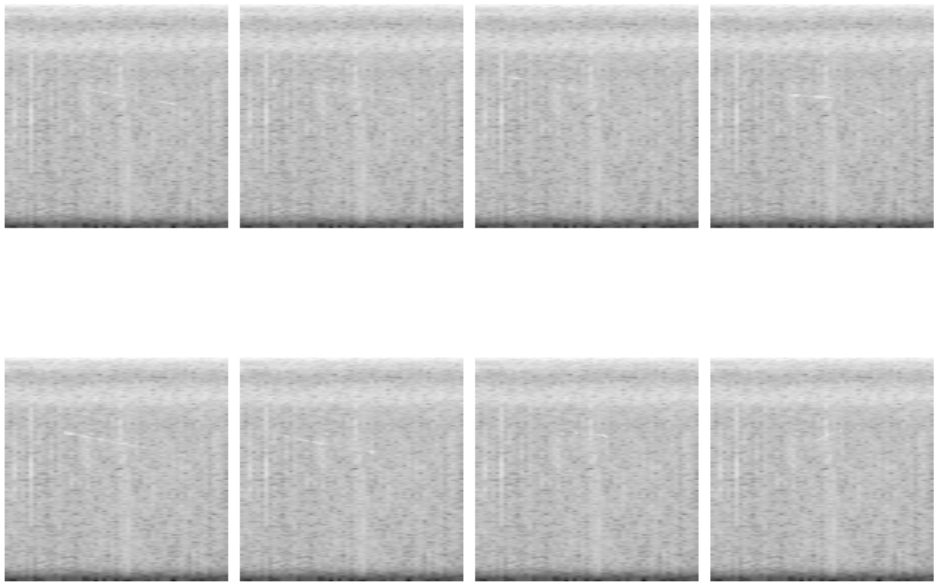
Optimisation is done using the optuna [4] library and all models are trained as was done in Chapter 2. The dataset is split into training (80%), testing (10%), and validation (10%) subsets, and the models are trained until they undergo a consecutive number of epochs without improvement on validation loss. Channel 1 of dataset A is used, so input data to the models is 1-channel as it was in Sec-

tion 2.4.5. The hyperparameters for these models are those found to be optimal in Appendix B for dataset A channel 1. The optuna process was allowed to run 50 sets of parameters for each metric, and the top three trials in each are shown. The last four columns are not obtained in the optimisation process; once the parameter set has been chosen, the relevant discrimination or detection process as indicated by the column and described in Section 4.2.1 is applied to the procedurally-generated synthetic samples. The column corresponding to the parameter set's own score is also calculated and this can function as a confirmation of consistent performance in that metric.

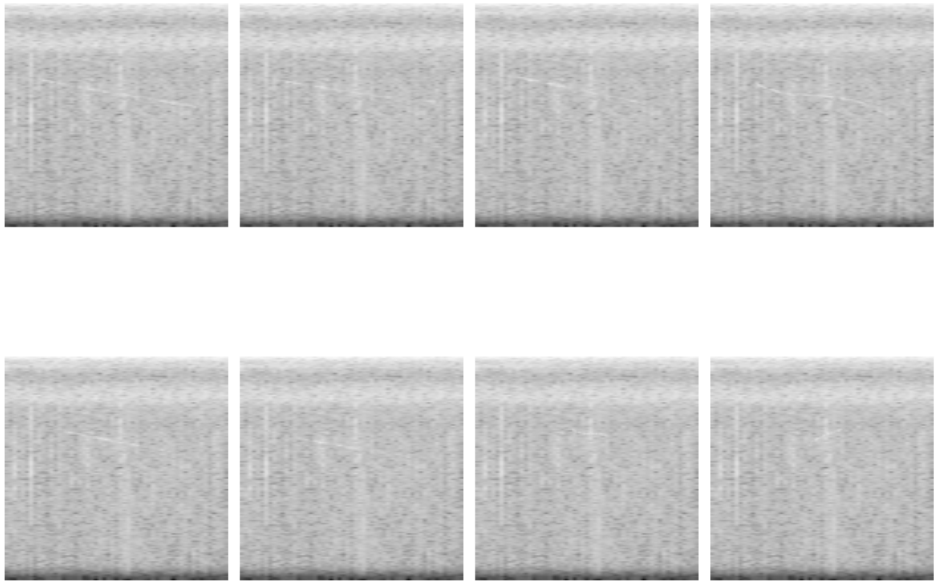
Overall, it can be seen that the top three trials in each metric did not have large differences between how they scored when being optimised. When comparing the metric's score (column "Score") to how it performs when re-assessed (one of the last four columns), there are some differences. Interesting to note is that all trials of SDisA initially showed that the models were unable to converge, but this was untrue when it was run again. This seems to indicate that these parameter sets were sometimes able to fool the Simple model but were not of high enough quality to consistently do so. When assessing a model's SDetA score, we are able to view the associated MD and FA. Although it is not depicted here, these parameters uniformly achieved very low FA rates with the except of SDisA (3). All FA values were less than 2%, with the majority being less than 1%. While their MD rates were higher, this is a reasonable trade-off since having falsely labelled positives could contaminate the true whistles dataset and result in extra work.

Based on these results, there are three sets of hyperparameters that seem to perform at relatively high levels in all four metrics: SDisE (Nth Best = 1), SDetPA (Nth Best = 2), and SDetPA (Nth Best = 3). All three showed the lowest classification accuracies and reasonably high numbers of epochs to reach that value, with the except of SDetPA (3) since it did not converge at all. Additionally, they achieve the highest SDetPA and SDetA values. The resulting images using the associated parameter sets for these three optuna trials are shown in Figure 4.5.

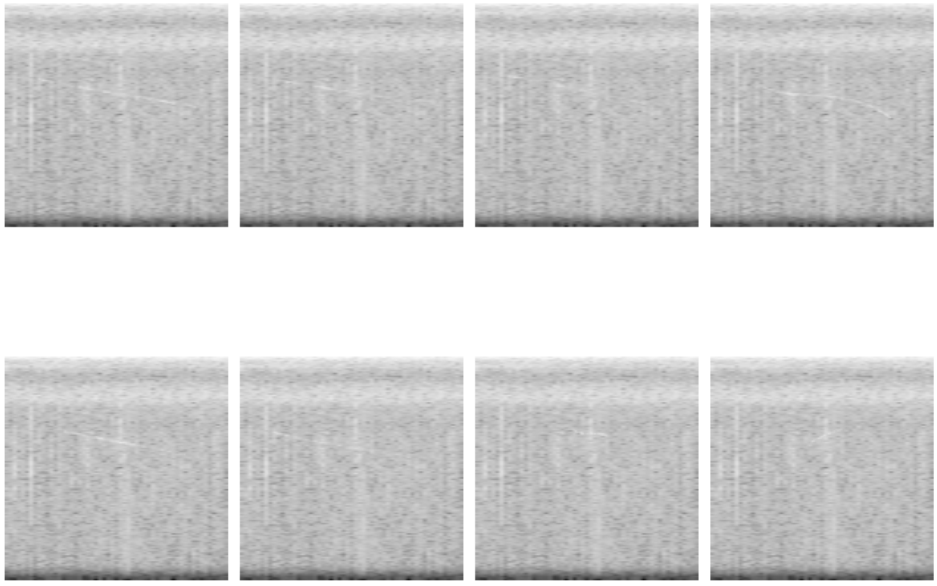
Visually, the samples generated from these parameter sets seem realistic if weak to the eye. At the same time, we can see from the results in Table 4.1 that quantitatively, they do not seem promising. Based on the lack of improvement in further trials and the sufficient visual appeal of these samples, however, we decided



(a) SDisE (1).



(b) SDetPA (2).



(c) SDetPA (3).

Figure 4.5: Different sets of width parameters as labelled resulting in different synthetic samples (8 each), all shown with the same background.

to perform some additional realism assessment.

4.2.3 Realism Assessment

While the Simple model seems to indicate that some sets of whistle parameters may produce semi-realistic samples – reducing the discrimination accuracy to 90% – we also assess the truth of this with more complicated image classification networks. We perform the process for determining SDisA with these models. As such, Table 4.2 shows the discrimination accuracy for models Simple, Dense161, Res152, and VGG19bn.

As can be seen, there is a strong difference between the synthetic and positive samples. The models near-universally achieve classification performance at approximately 99%, which indicates that they can almost always identify which samples are synthetic as opposed to positive. While the synthetic samples may be similar enough to be useful, as we will explore in Section 4.4, they cannot pass as

Opt. Metric	Nth Best	Model	Ep.	SDisA (%)
SDisE	1	Simple	290	97.63
		Dense161	45	99.68
		Res152	15	99.46
		VGG19bn	5	99.89
SDetPA	2	Simple	231	89.25
		Dense161	13	98.17
		Res152	3	99.68
		VGG19bn	1	99.68
SDetPA	3	Simple	50	56.45
		Dense161	9	96.24
		Res152	7	99.89
		VGG19bn	1	99.46

"Ep." indicates the number of epochs required to reach these results.

Table 4.2: Results from discrimination of positive/synthetic samples, where synthetic samples are procedurally-generated with specified parameters. Values for Simple model are the same as found in columns of Table 4.1.

positive samples.

Alternatively, however, we also examine if a model trained on positive samples would be able to detect these samples. This was motivated by the visual assessment of generated samples, which seemed to be sufficiently similar that we believed they should be able to interchangeably be used with positive samples in some situations. The models are trained on positive samples in channel 1 of dataset A, and the best models from Table B.1 are chosen for each model type. Then, the synthetic samples are labelled with this trained model and the accuracy is reported. These results are found in Table 4.3

These results indicate a strong similarity between positive and synthetic samples. Therefore, while they can be used in an interchangeable application to be seen in Section 4.4, this resemblance is not strong enough to merit trustworthiness for use in covert UWAC. This will be elaborated upon further in Section 5.2.

Opt. Metric	Nth Best	Model	Acc (%)
SDisE	1	Simple	92.33
		Dense161	89.37
		Res152	95.14
		VGG19bn	97.36
SDetPA	2	Simple	86.28
		Dense161	83.32
		Res152	91.78
		VGG19bn	95.02
SDetPA	3	Simple	85.84
		Dense161	83.47
		Res152	90.55
		VGG19bn	94.00

Table 4.3: Results from detection of synthetic samples from model trained on dataset A channel 1 real samples, where synthetic samples are procedurally-generated with specified parameters.

4.3 Cascaded DDPMs

In Section 4.2, we use a fully procedural method to generate synthetic whistles. Chapter 3 allowed us to use a model to produce the contour, and in this section, we use a model to add variation to the produced contours to create a fully model-based generative pipeline. We start with an overview of how a cascaded DDPM system functions. Then, we develop a second model to cascade with our existing contour-generator. Finally, we assess the realism of the results as done in Section 4.2.3. We call the synthetic samples created through usage of a cascaded DDPM system "model-generated".

4.3.1 Background

While it is possible to provide the DDPM with the synthetic whistles as inputs, thus skipping the intermediate step of generating a contour, there can be benefits to separating the two steps as follows. We can more easily change one without the other. The contour generator is able to create the shapes of whistles that we desire

while the variations can be customised to fit the dataset at hand. For instance, environmental context could affect how much the intensity of whistles changes over their duration, which can be adjusted by relearning the second component.

In [26], the authors use a series of cascaded DDPMs to increase the resolution of images. They begin by using a baseline DDPM to generate an original image of shape 32×32, based on the training dataset. Then, a DDPM model is used to spatially upscale the image without the artifacts that interpolation or other methods of resolution increase can result in. This can be done an arbitrary number of times, and [26] uses two DDPMs to change the dimensionality to 64×64 and then 256×256. These subsequent models differ from the first in a few ways, the most important being the input data. The resolution-increasing DDPMs still start from an input of noise which gradually becomes data, but they also receive an image as part of the input. For instance, the model which produces high-quality 64×64 images will take as input noise stacked with an algorithmically upsampled 32×32 image. In the training process, [26] has access to the original images of size 256×256. This allows them to create the required model inputs and target outputs for all three of the DDPMs.

For our situation, we would use $DDPM_c$ to generate the contour and $DDPM_v$ to generate the synthetic whistle. This has the same format as what is done in [26], but our cascaded models all use a dimensionality of 112×112. $DDPM_c$ receives only noise as an input and produces contours with constant width/intensity. $DDPM_v$ receives noise stacked with constant width/intensity contours as an input – thus having 2-channel data – and a synthetic whistle. This whistle will have the same general shape as its input contour, and its width/intensity will vary in a similar manner a desired set of variation parameters. The desired outputs of our $DDPM_v$ can be created through procedurally-generated whistles, which provides the target dataset. The step from synthetic whistle to sample is still done by multiplying our whistle with the appropriate s_w value.

Overall, the cascaded DDPMs system is used as follows to produce a single model-generated synthetic sample. This process is visually depicted in Figure 4.6, with the three labelled steps to be described. First, $DDPM_c$ is trained on the polynomial fitted time-frequency points to produce contours. A noise sample is provided as input to the model, and a contour is produced as output. Second, $DDPM_v$

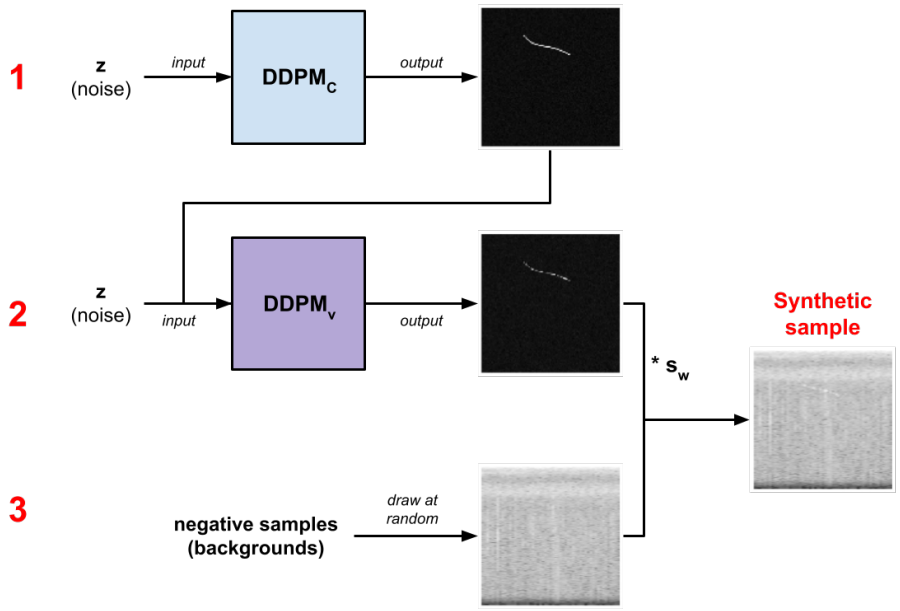


Figure 4.6: Illustrative process for creating a synthetic sample using the cascaded DDPM approach. A sample from SDisE (1) is shown here.

is trained using polynomial fitted time-frequency points which have been modified with the variation parameters and intensity masks. The training data includes a second channel of data which is a noise sample stacked with a contour from $DDPM_c$, and a whistle is produced as output. Third, using the s_w modifier, we stack this whistle onto a background to produce a synthetic sample.

4.3.2 Results

We use the process outlined in Section 4.3.1 to generate synthetic samples for dataset B, which will be shown and explained in Section 4.4.2, and thus we will generate at least 1598 samples to match the original dataset size. To begin, we generated 5000 contours and determined how many were usable. We first eliminate any samples which did not produce a contour and instead only look like noise; examples of this can be seen in Figure 3.9 and were discussed in Section 3.4.3. This occurred in 602 samples, which is 12.04% of the total number generated. Ad-

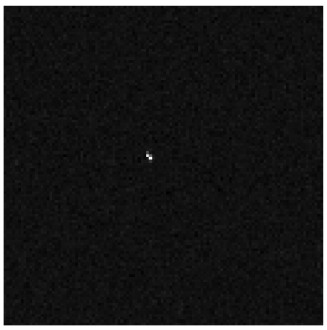


Figure 4.7: Example of a generated contour which could not be used because of its extremely short duration.

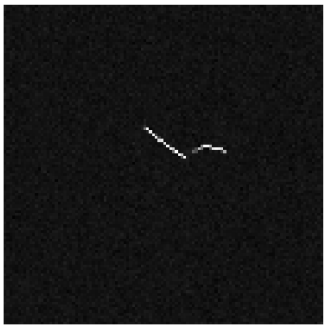
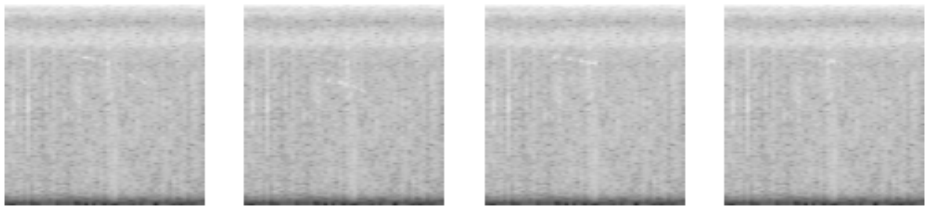


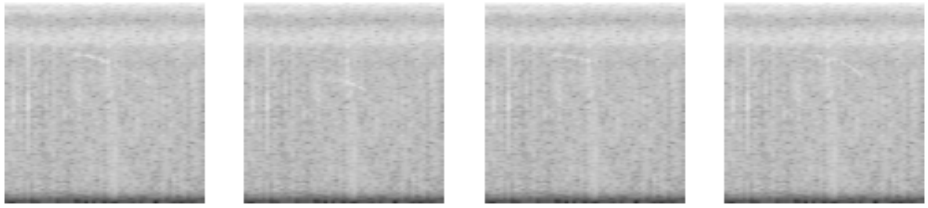
Figure 4.8: Example of a generated contour which was used despite being disconnected because the components could be from one signal.

ditionally, we eliminated a few cases where the produced contour was infeasibly small, such as Figure 4.7. This occurred in 72 samples, which is 1.44% of the total number generated. Finally, since we had a surplus of samples, we eliminated the samples which included more than one distinct contour that could not be interpreted as a single signal which was either interrupted or faded. An example of a $DDPM_c$ output that is composed of disjoint components but could be a single signal is found in Figure 4.8. There is a degree of subjectivity to this selection, and we eliminated 1901 samples (38.02%). Overall, we retain 2571 samples (51.42%) that could be used as synthetic contours.

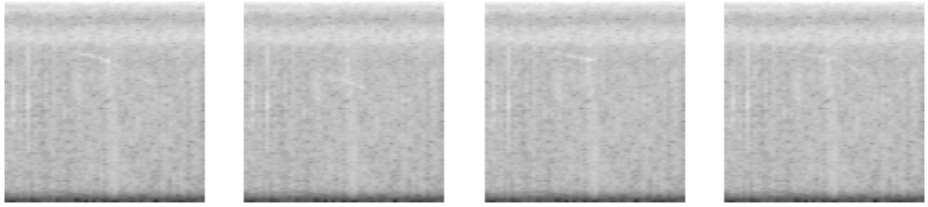
These contours were then stacked with noise for our $DDPM_v$ to create 2-channel input data. The target whistles were generated using variation parameter sets SDisE (1), SDetPA (2), and SDetPA (3). Examples of synthetic samples gen-



(a) SDisE (1).



(b) SDetPA (2).



(c) SDetPA (3).

Figure 4.9: Eight synthetic samples generated from the cascaded DDPM system. Each was produced using a different $DDPM_v$ model, trained using the labelled set of parameters.

erated from the cascaded DDPM system and overlaid onto the same background are shown in Figure 4.9. As mentioned in Section 3.4.3, $DDPM_c$ typically had faint traces of noise remaining in the generated image; the non-contour pixels were faint but not all were 0. The same phenomenon appears in outputs of $DDPM_v$, and it becomes less trivial to fix now that our relevant pixels are not all of the same intensity. Currently, we do not perform any cleaning on the synthetic whistles produced by $DDPM_c$ or $DDPM_v$.

Opt. Metric	Nth Best	Model	Ep.	SDisA (%)
SDisE	1	Simple	110	92.19
		Dense161	33	100.00
		Res152	26	99.69
		VGG19bn	37	99.69
SDetPA	2	Simple	145	92.81
		Dense161	29	99.69
		Res152	46	99.38
		VGG19bn	22	99.38
SDetPA	3	Simple	121	95.31
		Dense161	35	100.00
		Res152	57	99.69
		VGG19bn	29	100.00

Table 4.4: Results from discrimination of positive/synthetic samples, where synthetic samples are model-generated based on training data using specified parameters.

4.3.3 Realism Assessment

In Section 4.2.3, we used classification and detection to assess the synthetic samples' realism. This is done again, this time using model-generated synthetic samples using a cascaded DDPM. The results for classification of positive/synthetic samples and detection of positive samples are found in Tables 4.4 and 4.5 respectively.

The discrimination test performs approximately the same as in Section 4.2.3 (comparing Table 4.2 to Table 4.4). On average, it seems that there are more epochs required to reach the best result. The detection test performs marginally poorer in this section than in Section 4.2.3 (comparing Table 4.3 to Table 4.5). We suspect that this is due to the fact that dataset B – which we are using here – has been biased toward having only stronger positive samples because of the manual reviewing process as we stated in Section 2.4.5. Therefore, these model-generated samples likely appear weaker in the spectrogram, and it is more likely for a model trained on dataset B to not detect some synthetic samples. It is interesting to note that the detectors seem to be performing in the opposite direction as those previously; in

Opt. Metric	Nth Best	Model	Acc (%)
SDisE	1	Simple	93.87
		Dense161	90.99
		Res152	85.79
		VGG19bn	86.67
SDetPA	2	Simple	91.61
		Dense161	88.17
		Res152	86.92
		VGG19bn	85.67
SDetPA	3	Simple	92.37
		Dense161	90.30
		Res152	87.98
		VGG19bn	87.23

Table 4.5: Results from detection of synthetic samples, where synthetic samples are model-generated with $DDPM_v$ trained using whistles of specified parameters. The model is trained on channel 1 of dataset A, and the best models from Table B.1 are chosen for each model type.

other words, the models which achieved higher detection accuracy in Table 4.3 do worse in Table 4.5 and vice versa. The overall realism assessment is the same as in Section 4.2.3: while the synthetic samples can be used in an application that requires similarity to positive samples, they are not similar enough to fool NNs in a strict discrimination test.

4.4 Application to Iterative Detection

A scenario of where these realistically-generated synthetic samples may be useful is when whistle detection occurs in a novel environment. This is particularly true when we have very little or potentially no positive samples that have been labelled, but negative samples exist and there is a desire for automated detection based in this context. We can overlay generated contours with the backgrounds to create a synthetic/negative dataset, which can be used to drive future labelling in an automated fashion. This is similar in principle to the metric SDetA utilised in Section 4.2.

In this section, we demonstrate the extreme case where there is no positive labelled data available but sufficient negative data. Thus, we need three non-overlapping sets of negative samples: one for training, one for backgrounds, and one for "unseen" labelling. For simplicity, the sets will have the same number of samples. The iterative detection application is demonstrated twice using different datasets. We experiment with a naive technique that involves trusting the models and their results, incorporating any sample labelled as positive into the training set and replacing synthetic samples with these new ones.

4.4.1 Known Environment

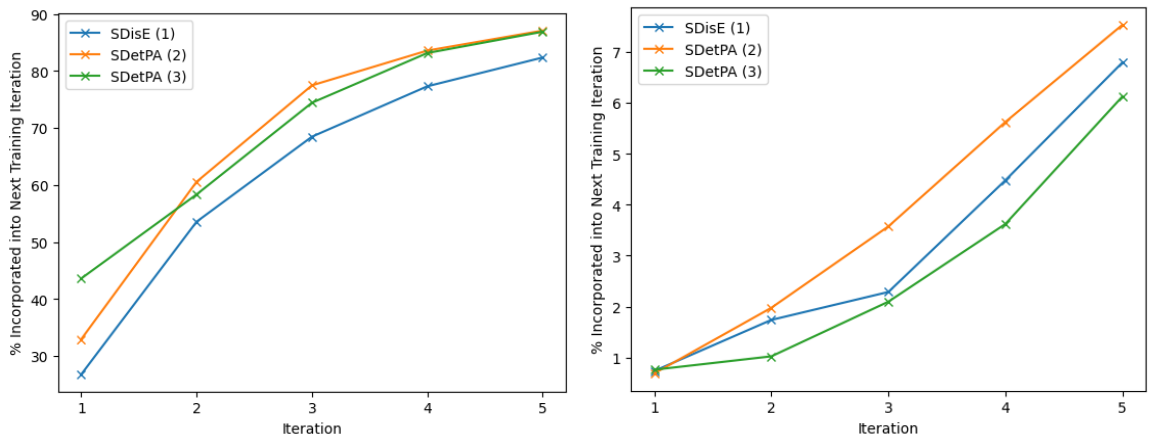
First, we start by using procedurally-generated synthetic samples. All real data is from dataset A. This example demonstrates iterative detection in a best-case scenario, since the polynomial-fitted contours and the positive samples used in parameter optimisation were both based on dataset A.

Our procedurally-generated synthetic samples are matched by an equal number of negative samples. Once a model is trained as a detector on the synthetic (output 1) and negative (output 0) samples, we test its capability on an unseen set of real data. This unseen set consists of 5247 positive samples (all labelled whistles in dataset A) and 5247 negative samples. 20% of these samples (1050 of each type) is held out of the training process, and these are used to judge detection accuracy on true samples at every iteration. We refer to this subset as the "Holdout" (HD). The remainder (4197 of each type) is labelled by the model and integrated as output 1 training/validation/testing samples for the next iteration of training. We refer to this subset as the "Integratable" (IG). The number of total samples used in these subsets is kept consistent, so synthetic samples are replaced by real samples which were previously labelled as positive. We refer to this method as a naive technique since we are trusting that any sample labelled "1" deserves to be integrated as a positive. This will reveal how much a model's abilities may degrade when the training/validation/testing samples become contaminated, and several iterations of detection using this process for the Simple model are shown in Table 4.6. The percentage of the available samples to be integrated into the training set which have been flagged at the end of every iteration is also shown visually in Figures 4.10.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	26.78	99.26	25.62	99.62
		2	53.56	98.26	52.57	98.76
		3	68.48	97.71	67.81	99.14
		4	77.32	95.52	73.62	95.62
		5	82.37	93.21	79.62	95.43
SDetPA	2	1	32.95	99.31	33.52	98.86
		2	60.57	98.02	59.90	98.19
		3	77.48	96.43	76.10	97.05
		4	83.58	94.38	80.19	96.67
		5	87.09	92.47	82.76	96.76
SDetPA	3	1	43.60	99.24	42.38	99.05
		2	58.35	98.98	55.62	99.14
		3	74.43	97.90	72.00	98.57
		4	83.15	96.38	80.48	97.62
		5	86.9	93.88	83.52	96.29

"Iter" indicates the iteration number used to achieve the row's results.

Table 4.6: Iterative detection results on procedurally-generated synthetic samples using Simple.



(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.10: Iteration IG accuracy values for Simple. Values are taken from Table 4.6.

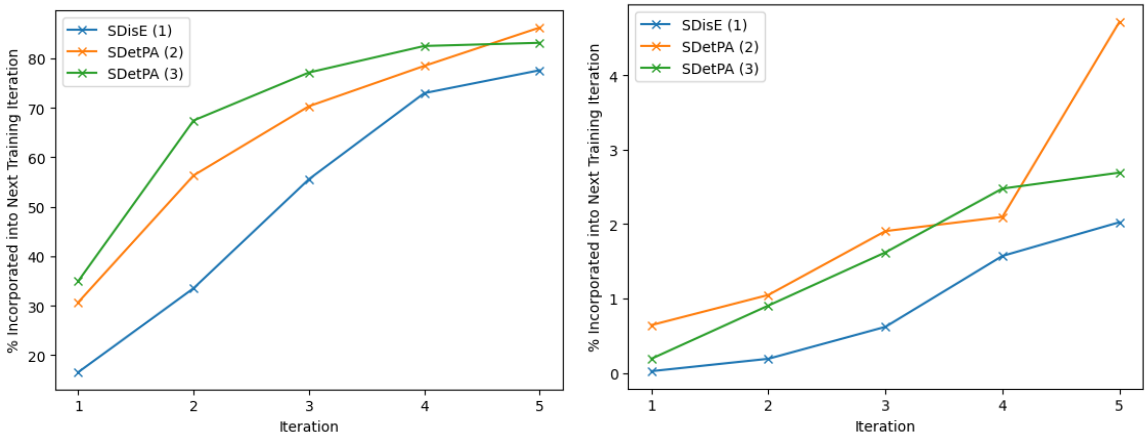
Overall, it appears that the chosen parameter sets are able to be used for iterative detection with fair success. The number of negative IG samples being accidentally integrated as positives is a low percentage of the total number available. Using Simple, we are able to detect the majority of true positive samples with only 5 iterations. A drawback of this process that is not depicted here is the occasional occurrence of model non-convergence. Table 4.6 only depicts sequences of events where the model was able to converge, but experimentally we occasionally found that the model would train for a consecutive number of epochs that we determined to be sufficient for early stopping without reaching a sufficiently low loss value. For the Simple model, this occurred only with parameter set SDetPA (3) in our experiments. This would typically be loss values greater than 0.5 with training/validation/test accuracy values lower than 70%. In terms of IG or HD accuracy, it would typically show 100% and 0% or values close to, indicating that the model was effectively becoming a "1" or "0" output in nearly all cases. This was, however, an inconsistent result – typically, a second run of the model was able to produce a result as found in Table 4.6. This would typically occur on the second or third iteration, indicating that it could potentially be a case of model confusion where the synthetic samples mixed with some positive samples made it more difficult for the model to produce clear results.

This can also be done with the other NN architectures used in Chapter 2: Dense161, Res152, and VGG19bn. The results are Tables 4.7, 4.8, and 4.9 respectively with their accompanying Figures 4.11, 4.12, and 4.13.

Dense161 suffers from the same occasional non-convergence issue as Simple. Since these two models are the smallest (less parameters), it appears that this non-convergence is mitigated by having a larger model with more parameters. Overall, these trials with more complicated models show the same results as with Simple. The performance does not vary greatly between the different type of model, and all are able to achieve high levels of accuracy for the IG and HD subsets.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	16.58	99.98	16.95	99.90
		2	33.55	99.81	31.14	99.62
		3	55.59	99.38	54.10	99.71
		4	72.98	98.43	70.48	99.05
		5	77.60	97.97	72.67	99.05
SDetPA	2	1	30.71	99.36	29.43	98.95
		2	56.35	98.95	53.81	99.33
		3	70.34	98.09	69.90	98.38
		4	78.48	97.90	74.86	99.62
		5	86.23	95.28	84.86	95.62
SDetPA	3	1	35.00	99.81	34.10	99.90
		2	67.45	99.09	67.52	98.48
		3	77.15	98.38	74.76	99.62
		4	82.51	97.52	78.57	98.67
		5	83.15	97.31	76.67	99.14

Table 4.7: Iterative detection results on procedurally-generated synthetic samples using Dense161.



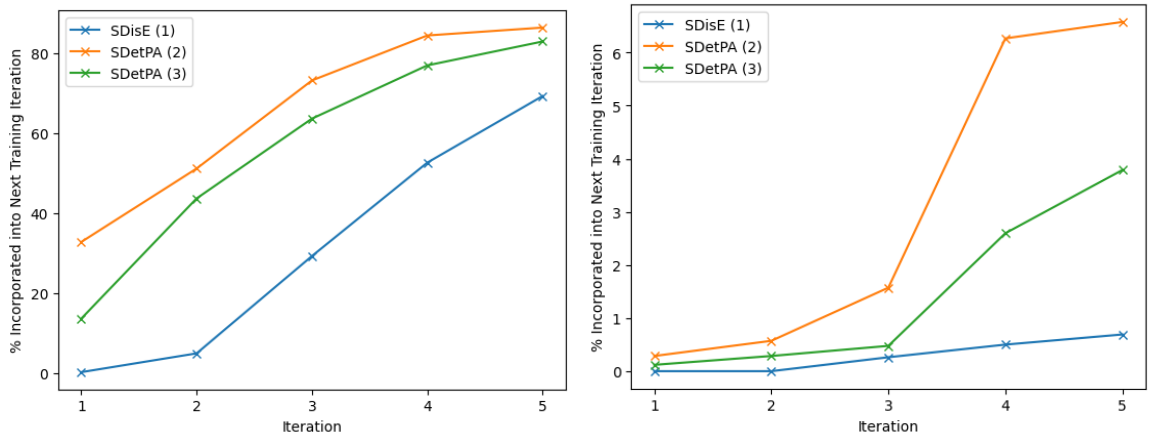
(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.11: Iteration IG accuracy values for Dense161. Values are taken from Table 4.7.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	0.21	100.00	0.29	100.00
		2	4.91	100.00	5.71	100.00
		3	29.26	99.74	28.00	99.90
		4	52.63	99.50	51.05	99.71
		5	69.29	99.31	69.05	99.05
SDetPA	2	1	32.79	99.71	32.19	99.90
		2	51.16	99.43	49.24	99.43
		3	73.20	98.43	71.14	98.76
		4	84.44	93.73	82.38	94.29
		5	86.42	93.42	75.43	99.52
SDetPA	3	1	13.60	99.88	13.81	99.71
		2	43.67	99.71	42.57	99.81
		3	63.64	99.52	63.14	100.00
		4	76.96	97.40	76.76	97.24
		5	82.94	96.21	78.10	98.48

Table 4.8: Iterative detection results on procedurally-generated synthetic samples using Res152.



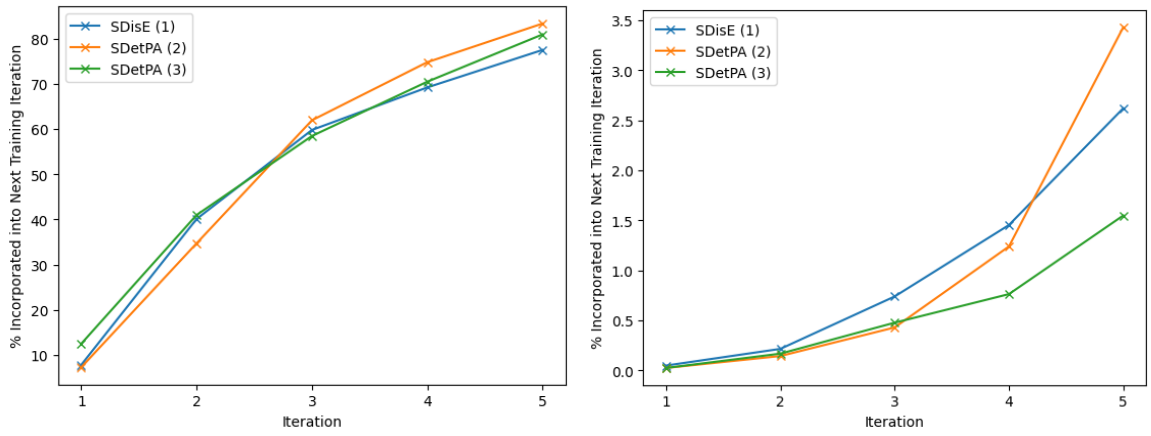
(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.12: Iteration IG accuracy values for Res152. Values are taken from Table 4.8.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	7.86	99.95	9.43	100.00
		2	40.12	99.79	39.24	99.81
		3	59.80	99.26	59.52	99.62
		4	69.22	98.55	68.00	98.95
		5	77.51	97.38	75.43	98.76
SDetPA	2	1	7.31	99.98	6.95	99.90
		2	34.74	99.86	33.14	99.71
		3	61.95	99.57	61.81	99.52
		4	74.79	98.76	74.00	98.86
		5	83.35	96.57	81.90	96.38
SDetPA	3	1	12.53	99.98	14.19	100.00
		2	41.01	99.83	40.29	99.71
		3	58.47	99.52	58.95	99.43
		4	70.48	99.24	67.81	99.33
		5	80.94	98.45	79.52	99.14

Table 4.9: Iterative detection results on procedurally-generated synthetic samples using VGG19bn.



(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.13: Iteration IG accuracy values for VGG19bn. Values are taken from Table 4.9.

4.4.2 Novel Environment

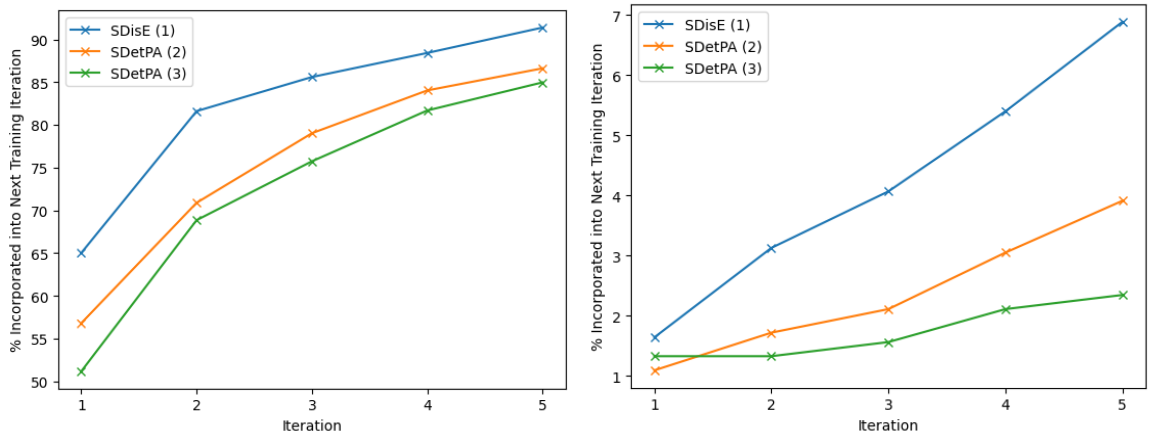
The application of this iterative detection to a new environment is the best assessment of its applicability. The time-frequency points were found based on positive samples from dataset A, so now we transition to using dataset B for all other aspects. This includes the negative backgrounds used to create the synthetic samples, the negative samples used for the model, the positive samples used to assess the model, and the unseen negative samples used to assess the model. The cascaded DDPM is used to generate our synthetic whistles, as explained in Section 4.3. We test it in the same manner as in Section 4.4.1 using Simple (Table 4.10 and Figure 4.14), Dense161 (Table 4.11 and Figure 4.15), Res152 (Table 4.12 and Figure 4.16), and VGG19bn (Table 4.13 and Figure 4.17).

Overall, we see that the results from this section are similar to and actually better than those in Section 4.4.1. Nearly all models are able to detect 50%+ of positive whistles on the first iteration, with the exception of VGG19bn which only manages this one. Additionally, the accuracy values after five iterations are higher than in Section 4.4.1. As stated previously, we believe this is due partly to the nature of how positive samples in dataset B were chosen and checked. However, it does indicate that although the time-frequency points we used to create training data for $DDPM_c$ were taken from dataset A, the model was able to learn diverse and general enough whistles that the generated outputs were usable for dataset B. Therefore, we can say that the cascaded DDPM system is effective at generating whistle samples which can be used across different oceanic environment.

This section has shown the efficacy of using generated synthetic whistles in training detectors to effectively detect real positive whistles with low rates of error. We suggest that it may be more effective to train with synthetic samples that are particularly weak so long as the detectors are able to sufficiently converge, as dataset A seems to be compared to dataset B. This allows the detectors to be "stronger" since they become accustomed to searching for the dolphin whistle time-frequency traces even when they are less distinguishable from background noise.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	65.02	98.36	67.50	99.38
		2	81.61	96.87	80.00	99.38
		3	85.60	95.93	83.13	99.06
		4	88.42	94.60	85.31	99.69
		5	91.39	93.11	86.56	97.50
SDetPA	2	1	56.81	98.90	59.38	99.69
		2	70.89	98.28	71.25	100.00
		3	79.03	97.89	78.44	99.38
		4	84.04	96.95	80.00	99.38
		5	86.62	96.09	83.44	98.44
SDetPA	3	1	51.17	98.67	53.44	99.38
		2	68.86	98.67	68.75	99.69
		3	75.74	98.44	74.06	100.00
		4	81.69	97.89	80.00	100.00
		5	84.98	97.65	81.88	99.69

Table 4.10: Iterative detection results on model-generated synthetic samples using Simple.



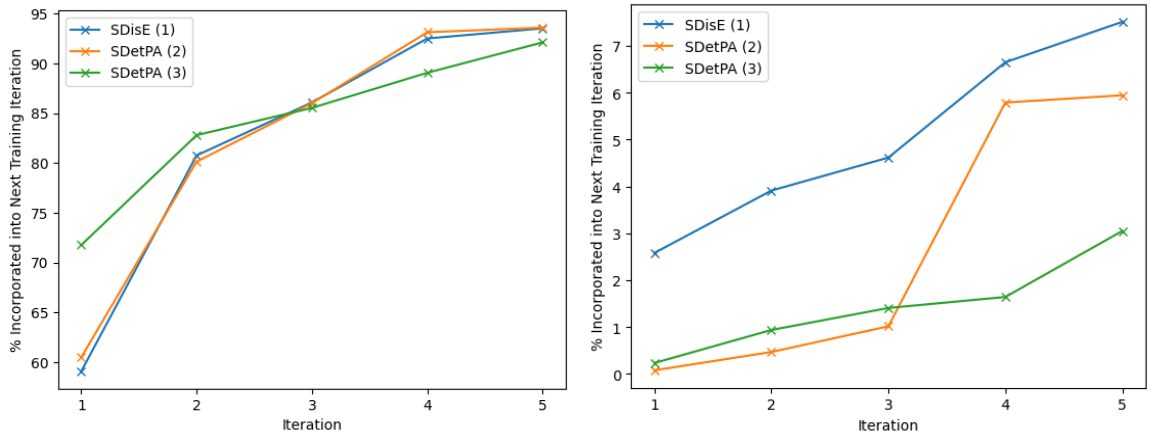
(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.14: Iteration IG accuracy values for Simple. Values are taken from Table 4.10.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	59.08	97.42	62.19	96.88
		2	80.75	96.09	78.44	99.06
		3	86.07	95.38	80.94	100.00
		4	92.49	93.35	88.44	96.56
		5	93.51	92.49	88.44	96.88
SDetPA	2	1	60.49	99.92	65.31	100.00
		2	80.13	99.53	79.06	100.00
		3	85.99	98.98	82.81	99.69
		4	93.11	94.21	93.75	94.69
		5	93.58	94.05	88.75	99.06
SDetPA	3	1	71.75	99.77	75.00	99.38
		2	82.79	99.06	83.44	99.69
		3	85.52	98.59	80.94	100.00
		4	89.05	98.36	86.25	99.69
		5	92.10	96.95	88.44	99.06

Table 4.11: Iterative detection results on model-generated synthetic samples using Dense161.



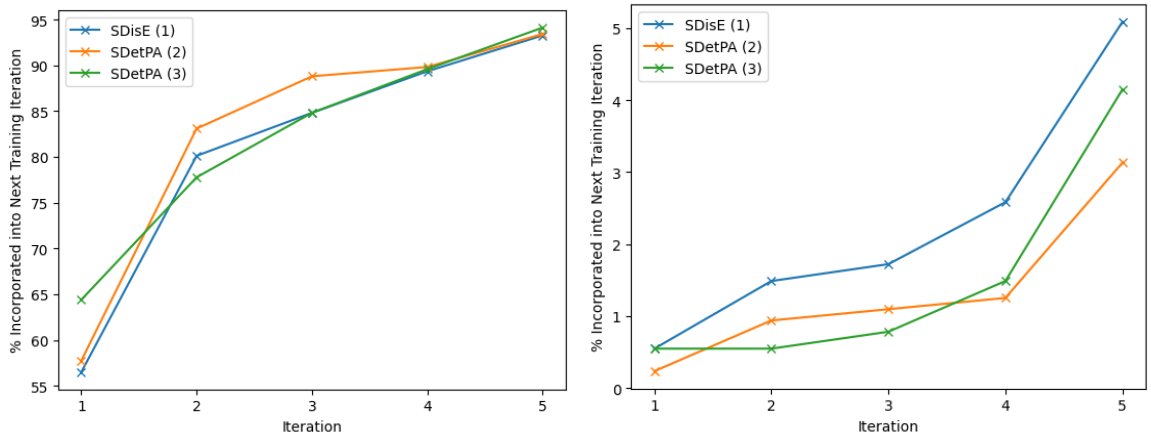
(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.15: Iteration IG accuracy values for Dense161. Values are taken from Table 4.11.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	56.49	99.45	59.69	99.69
		2	80.13	98.51	79.06	99.06
		3	84.82	98.28	79.38	100.00
		4	89.36	97.42	85.94	99.06
		5	93.27	94.91	89.69	97.19
SDetPA	2	1	57.75	99.77	59.69	99.69
		2	83.10	99.06	82.50	99.69
		3	88.81	98.90	86.25	99.69
		4	89.83	98.75	83.75	99.06
		5	93.43	96.87	90.63	95.94
SDetPA	3	1	64.40	99.45	67.19	99.69
		2	77.78	99.45	76.56	100.00
		3	84.82	99.22	83.13	100.00
		4	89.59	98.51	87.50	99.69
		5	94.13	95.85	90.31	98.13

Table 4.12: Iterative detection results on model-generated synthetic samples using Res152.



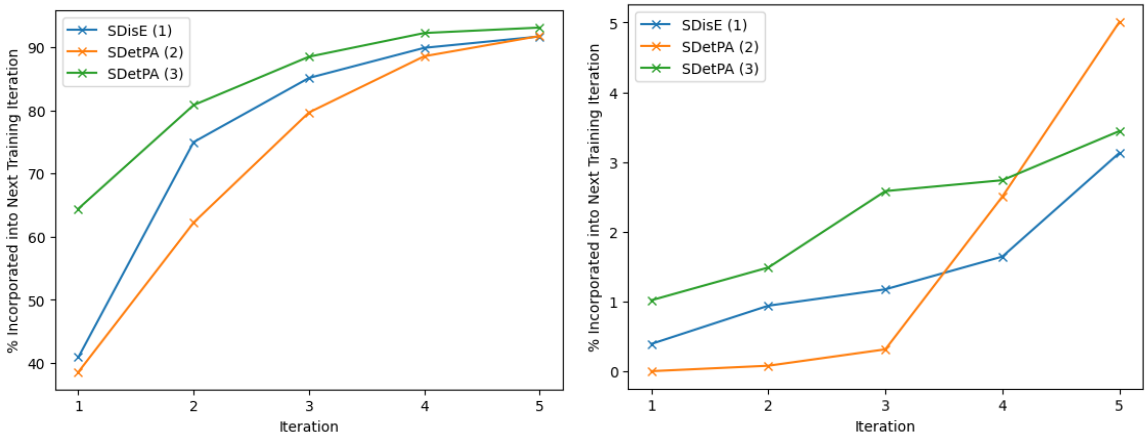
(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.16: Iteration IG accuracy values for Res152. Values are taken from Table 4.12.

Opt. Metric	Nth Best	Iter	IG Pos. Acc (%)	IG Neg. Acc (%)	HD Pos. Acc (%)	HD Neg. Acc (%)
SDisE	1	1	40.85	99.61	44.69	100.00
		2	74.96	99.06	75.31	99.69
		3	85.13	98.83	83.13	99.38
		4	89.91	98.36	87.19	99.06
		5	91.71	96.87	87.19	98.75
SDetPA	2	1	38.50	100.00	42.19	100.00
		2	62.21	99.92	60.94	100.00
		3	79.66	99.69	80.00	100.00
		4	88.58	97.50	86.88	98.75
		5	91.78	94.99	89.38	96.25
SDetPA	3	1	64.40	98.98	64.69	99.06
		2	80.83	98.51	79.69	99.69
		3	88.50	97.42	87.81	99.38
		4	92.25	97.26	90.00	99.69
		5	93.11	96.56	91.25	99.38

Table 4.13: Iterative detection results on model-generated synthetic samples using VGG19bn.



(a) Positive IG samples.

(b) Negative IG samples.

Figure 4.17: Iteration IG accuracy values for VGG19bn. Values are taken from Table 4.13.

Chapter 5

Conclusions and Future Work

In this chapter, we outline the main contributions of the thesis and propose directions for related future work.

5.1 Conclusions

This thesis demonstrated the utility of NNs for the application of dolphin whistle detection and generation. We started by developing an automated detection system for dolphin whistles using image classification networks, including one basic architecture and several which leveraged transfer learning from pre-trained classifiers. After k-cross validation and hyperparameter tuning, the trained networks were able to consistently achieve detection performance which matched or beat existing methods and in particular on a balanced dataset. This was tested on a second dataset in a different oceanic environment. We explored the impact of freezing layers of pre-trained models and reducing the training set size, both of which tended to degrade model performance. However, when combined, the tested networks were able to resist overfitting more effectively and achieved better performances than either effect alone. The key findings in this Chapter 2 are in the generalisability and transferability of our techniques. Generalisability describes the ability to perform across different datasets. When using Min-Pre (i.e. bare minimum preprocessing when converting to spectrogram data format), we are able to achieve comparable performance across different datasets without needing to adjust any noise filtering,

unlike the drop in performance that Add-Pre experiences. This was proven using multiple trials for each experiment, which verifies that the results are consistent rather than one-off happenstances. We showed that the closer cropping and additional filter in Add-Pre was detrimental to performance in dataset B, which is likely due to the fact that those procedures were chosen based on samples from dataset A. Thus, we can leverage the power of the NNs and minimise human effort in this detection process. Transferability refers to the utility of transfer learning. Initially, our non-Simple models retain parameters from an ImageNet [16] classification task and this allows the models to quickly converge. We tested the transference of parameters found from dataset A to training models on dataset B. The benefit of this is most obvious when parameter freezing and dataset size reduction occur; as such, we can say that if an additional, small dataset exists, it can be very beneficial to have a model which is pretrained on a sufficiently large set of similar spectrogram data. This can allow future models to converge faster, and overfitting can potentially be reduced by freezing part of the model parameters.

Chapters 3 and 4 outlined the process for generating dolphin whistles using NNs. This generation process is broken into the contour (i.e. whistle shape) and the variations (i.e. "width" and "pixel intensity" of the signal over time). Starting with the contour generation, we tested two popular generative model styles – GANs and DDPMs – before deciding that DDPM were more effective in producing varied samples without inexplicable background artifacts. To generate realistic whistles with the appropriate variations, we decided on six tuneable parameters and integration of randomly synthesized Bellhop impulse responses. These parameters were optimised using a series of NN-driven metrics aimed at creating synthetic samples which closely resemble positive samples. The top performing sets of parameters were used to train a cascaded pair of DDPMs which generated contours from noise and then synthetic whistles (i.e. with variations in width and intensity) from contours. These generated whistles were applied to the scenario of supplementing a dataset, which in our case started with 0 positive samples. A detector was trained on synthetic and negative samples and would gradually integrate positive/negative samples which it flagged as having a whistle, creating an iterative detection process which requires very little data to begin. This was tested on our second dataset, proving cross-environmental efficacy for our generative network and whistles. The

key findings in these chapters is the efficacy of using synthetic whistles. Chapter 3 is the foundation for the application in Section 4.4, first creating whistle contours; then, Sections 4.2 and 4.1 allow us to make these contours into synthetic samples which resemble true positive samples. When these synthetic samples were used in the place of positive samples during whistle detection training, we found that iterating even twice would allow us to detect over half of the positive samples with very few negatives accidentally labelled as positive. This was true even across datasets, indicating that the synthetic whistle samples we created were sufficiently realistic to guide detection in a novel environment. This can be particularly useful when we want to perform detection on large datasets that have few labelled samples; if a reasonable amount of negative samples and background noise can be obtained, we would be able to apply this detection technique to automatically find marine mammal signals without requiring that a dataset first be built manually. While it is inevitable that a human expert should be included in the process to verify the tagged data, that is a quicker and easier job than searching through all existing data for possible signals. The low FA rate is promising in that this system is unlikely to create much unnecessary/extra work; while the MD rate is initially fairly high, applying this detection process iteratively – until a sufficiently large number of positive samples are found – has been shown to be successful at lowering that.

5.2 Future Work

In this thesis, we utilised two datasets which are relatively clean; thus, our work does not cover environments where there are significant noise sources or other marine mammal signals. Additionally, the models and preprocessing we use are open to use in future work as classifiers rather than merely detectors. [29] utilises CNN to classify whistles from different whale species, and [34] applies a similar technique to dolphin clicks from different species. If Min-Pre is applied to this task, it may prove to be equally effective for automated classification.

Another avenue for future work is in developing better variation parameters for the contours. Ideally, this would be done with grounded justification in how dolphin whistles truly change over time, frequency, environments, etc. Regardless, as noted in Section 4.2.2, the scores obtained by our even our best metrics are far

from their ideal values. As well, the synthetic samples generated procedurally or by model fail the discrimination test against positive samples. An earlier iteration of produced samples resulted in extremely consistent high discrimination accuracy, and we were able to determine that this was a result of unnoticeable differences in how data was being normalised by the different libraries. Switching to the same library fixed this particular issue, but we speculate that there may be some artifacts in the data that may be interfere with achieving better discrimination.

For the developed DDPM, we utilise basic techniques which were sufficient for the task at hand to demonstrate the ability of these networks in generating whistles. However, there are areas for improvement for sample quality and generation speed. [40] finds several modifications which created competitive log-likelihood values while also decreasing the number of timesteps required. In [50], the diffusion process is reworked to be non-Markovian, which effectively allows a generation schedule which "skips" timesteps. As well, we noted that the generated samples were not always usable; for instance, the contour-DDPM would occasionally produce images which were merely noise without a noticeable contour. With further work and fine-tuning of the model parameters, it is possible that this is either eliminated or reduced. Alternatively, it would be possible to create a process which could assess samples and automatically discard those that could not be used. Finally, a next step for utilising these signals in biological covert UWAC is to incorporate the embedding of information into the generation process. This could be done, for instance, using conditional generation with an associated classifier at the receiver end. Finally, as mentioned, our DDPMs tended to produce images with faint, noise-like pixels still lingering. Removing these is left to future work and may aid in creation of better samples.

This thesis used spectrograms as the data format of choice. This was suitable for our application purpose and allowed us to leverage pre-existing work in image classification/generation networks, which is currently a more robustly explored field than audio generation. For real transmission of dolphin whistles, however, this would require that these signals are either converted into or generated as audio data. Thus, as next steps, spectrogram-to-audio methods and audio generation NNs should be assessed for their capabilities. This would create signals that could be used in biological covert UWAC.

Bibliography

[1] Link to the publicly available repository containing all our acoustic recordings, 2022. URL https://csms-acoustic.haifa.ac.il/index.php/s/2UmUoK80Izt0Roe. Accessed on Dec, 2022. → page 12

[2] T. A. Abbot, V. E. Premus, and P. A. Abbot. A real-time method for autonomous passive acoustic detection-classification of humpback whales. *The Journal of the Acoustical Society of America*, 127(5):2894–2903, 05 2010. ISSN 0001-4966. doi:10.1121/1.3365255. URL https://doi.org/10.1121/1.3365255. → page 7

[3] J. Ahn, H. Lee, Y. Kim, S. Lee, and J. Chung. Mimicking dolphin whistles with continuously varying carrier frequency modulation for covert underwater acoustic communication. *Japanese Journal of Applied Physics*, 58:SGGF05, 07 2019. doi:10.7567/1347-4065/ab14d2. → page 46

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019. → pages 21, 76

[5] C. Bazua. Differences in the whistle characteristics and repertoire of bottlenose and spinner dolphins. *Anais da Academia Brasileira de Ciências*, 76:386–92, 07 2004. doi:10.1590/S0001-37652004000200030. → page 2

[6] C. Bergler, H. Schröter, R. X. Cheng, V. Barth, M. Weber, E. Nöth, H. Hofer, and A. Maier. Orca-spot: An automatic killer whale sound detection toolkit using deep learning. *Scientific Reports*, 9, 2019. → page 7

[7] A. Borji. Pros and cons of gan evaluation measures. *Computer Vision and Image Understanding*, 179:41–65, 2019. ISSN 1077-3142. doi:https://doi.org/10.1016/j.cviu.2018.10.009. URL

https://www.sciencedirect.com/science/article/pii/S1077314218304272. → pages 73, 74

[8] B. Branstetter, A. Black, and K. Bakhtiari. Discrimination of mixed-directional whistles by a bottlenose dolphin (tursiops truncatus). *The Journal of the Acoustical Society of America*, 134:2274–85, 09 2013. doi:10.1121/1.4816404. → page 45

[9] J. R. Buck, H. B. Morgenbesser, and P. L. Tyack. Synthesis and modification of the whistles of the bottlenose dolphin, Tursiops truncatus. *The Journal of the Acoustical Society of America*, 108(1):407–416, 07 2000. ISSN 0001-4966. doi:10.1121/1.429474. URL https://doi.org/10.1121/1.429474. → page 46

[10] G. Burrowes and J. Khan. *Short-Range Underwater Acoustic Communication Networks*, chapter 8, pages 173–198. InTech, 10 2011. ISBN 978-953-307-432-0. doi:10.5772/24098. → page 69

[11] M. C. Caldwell, D. K. Caldwell, and P. L. Tyack. Review of the signature-whistle hypothesis for the atlantic bottlenose dolphin. In S. Leatherwood and R. R. Reeves, editors, *The Bottlenose Dolphin*, pages 199–234. Academic Press, San Diego, 1990. ISBN 978-0-12-440280-5. doi:https://doi.org/10.1016/B978-0-12-440280-5.50014-7. URL https://www.sciencedirect.com/science/article/pii/B9780124402805500147. → page 3

[12] Cetalingua. Dolphin chat project, 2023. URL https://cetalingua.com/dolphin-chat/. Accessed on Dec 15, 2023. → pages xv, 2

[13] M. J. Chong and D. Forsyth. Effectively unbiased fid and inception score and where to find them. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6069–6078, 2020. doi:10.1109/CVPR42600.2020.00611. → page 74

[14] I. Davidesco and R. Diamant. Detection of dolphin whistle-like biomimicking signals by phase analysis. *IEEE Access*, 10:36868–36876, 2022. doi:10.1109/ACCESS.2022.3165058. → pages 46, 48

[15] DCLDE. 8th DCLDE workshop – dataset challenge, 2018, http://sabiod.lis-lab.fr/dclde/challenge.html., 2018. → page 12

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. → pages 4, 100

[17] W. Ding, B. Wuersig, and W. E. Evans. Whistles of bottlenose dolphins: Comparisons among populations. In *Aquatic Mammalogy*, 1995. URL https://api.semanticscholar.org/CorpusID:89178403. → page 26

[18] A. ElMoslimany, M. Zhou, T. M. Duman, and A. Papandreou-Suppappola. A new signaling scheme for underwater acoustic communications. In *2013 OCEANS - San Diego*, pages 1–5, 2013. doi:10.23919/OCEANS.2013.6741333. → page 47

[19] D. Gillespie. Detection and classification of right whale calls using an 'edge' detector operating on a smoothed spectrogram. *Canadian Acoustics*, 32(2):39–47, Jun. 2004. URL https://jcaa.caa-aca.ca/index.php/jcaa/article/view/1586. → page 6

[20] D. Gillespie, M. Caillat, J. Gordon, and P. White. Automatic detection and classification of odontocete whistles. *The Journal of the Acoustical Society of America*, 134(3):2427–2437, 2013. → page 7

[21] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014. → page 49

[22] P. Gruden and P. White. Automated extraction of dolphin whistles-—a sequential Monte Carlo probability hypothesis density approach. *The Journal of the Acoustical Society of America*, 148(5):3014–3026, 2020. → page 7

[23] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi:10.1109/CVPR.2016.90. → page 11

[24] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006. doi:10.1162/neco.2006.18.7.1527. → page 8

[25] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *International Conference on Neural Information Processing Systems (NIPS)*,

2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf. → pages 5, 56, 57, 58

[26] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans. Cascaded diffusion models for high fidelity image generation. *J. Mach. Learn. Res.*, 23(1), Jan. 2022. ISSN 1532-4435. → page 82

[27] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. doi:10.1109/CVPR.2017.243. → page 11

[28] N. Inkawhich. DCGAN tutorial, 2022. URL https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html. → page 50

[29] J. jia Jiang, L. ran Bu, F. jie Duan, X. quan Wang, W. Liu, Z. bo Sun, and C. yue Li. Whistle detection and classification for whales based on convolutional neural networks. *Applied Acoustics*, 150:169–178, 2019. ISSN 0003-682X. doi:https://doi.org/10.1016/j.apacoust.2019.02.007. URL https://www.sciencedirect.com/science/article/pii/S0003682X18311241. → page 101

[30] J. Jiang, Z. Sun, F. Duan, X. Fu, X. Wang, C. Li, W. Liu, and L. Gan. Synthesis and modification of cetacean tonal sounds for underwater bionic covert detection and communication. *IEEE Access*, 8:119980–119994, 2020. doi:10.1109/ACCESS.2020.3004282. → page 46

[31] B. Jones, M. Zapetis, M. M. Samuelson, and S. Ridgway. Sounds produced by bottlenose dolphins (Tursiops): a review of the defining characteristics and acoustic criteria of the dolphin vocal repertoire. *The International Journal of Animal Sound and its Recording*, 29(4):399–440, 2019. → pages 1, 2, 3

[32] B. N. Korkmaz. *Deep learning techniques for biological signal processing: Automatic detection of dolphin sounds*. PhD thesis, University of Padova, 2021. → pages 4, 7, 11, 12, 15, 16, 17, 21

[33] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Neural Information Processing Systems*, 1989. URL https://api.semanticscholar.org/CorpusID:2542741. → page 4

[34] J. Liu, X. Yang, C. Wang, and Y. Tao. A convolution neural network for dolphin species identification using echolocation clicks signal. In *2018 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC)*, pages 1–4, 2018. doi:10.1109/ICSPCC.2018.8567796. → page 101

[35] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015. doi:10.1109/ACPR.2015.7486599. → page 11

[36] S. Liu, T. Ma, G. Qiao, L. Ma, and Y. Yin. Biologically inspired covert underwater acoustic communication by mimicking dolphin whistles. *Applied Acoustics*, 120:120–128, 05 2017. doi:10.1016/j.apacoust.2017.01.018. → pages 45, 46

[37] X. Lu, L. Lampe, B. N. Korkmaz, A. Testolin, and R. Diamant. Transfer learning of image classification networks in application to dolphin whistle detection. In *OCEANS 2023 - Limerick*, pages 1–9, 2023. doi:10.1109/OCEANSLimerick52467.2023.10244251. → page vi

[38] L. May-Collado and D. Wartzok. A comparison of bottlenose dolphin whistles in the atlantic ocean: Factors promoting whistle variation. *Journal of Mammalogy - J MAMMAL*, 89:1229–1240, 10 2008. doi:10.1644/07-MAMM-A-310.1. → page 26

[39] D. K. Mellinger and C. W. Clark. Recognizing transient low-frequency whale sounds by spectrogram correlation. *The Journal of the Acoustical Society of America*, 107(6):3518–3529, 06 2000. ISSN 0001-4966. doi:10.1121/1.429434. URL https://doi.org/10.1121/1.429434. → page 7

[40] A. Nichol and P. Dhariwal. Improved denoising diffusion probabilistic models. *CoRR*, abs/2102.09672, 2021. URL https://arxiv.org/abs/2102.09672. → page 102

[41] Oceans Acoustic Library. Acoustics toolbox, 2010. URL http://oalib.hlsresearch.com/AcousticsToolbox/. → page 69

[42] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In *International Conference on Neural Information*

Processing Systems (NIPS), pages 8024–8035. neurips, 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. → pages 11, 17

[43] T. Pearce. Conditional diffusion MNIST, 2022. URL https://github.com/TeaPearce/Conditional_Diffusion_MNIST/blob/main/script.py. → page 59

[44] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Y. Bengio and Y. LeCun, editors, *ICLR (Poster)*, 2016. URL http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#RadfordMC15. → pages 5, 50

[45] M. Roch, T. Brandes, B. Patel, Y. Barkley, S. Baumann-Pickering, and M. Soldevilla. Automated extraction of odontocete whistle contours. *The Journal of the Acoustical Society of America*, 130:2212–23, 10 2011. doi:10.1121/1.3624821. → page 7

[46] O. Ronneberger, P. Fischer, and T. Brox. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4. → pages xviii, 59, 60

[47] S. Seramani, E. A. Taylor, P. J. Seekings, and K. P. Yeo. Wavelet de-noising with independent component analysis for segmentation of dolphin whistles in a noisy underwater environment. In *OCEANS*, pages 1–7, 2006. doi:10.1109/OCEANSAP.2006.4393920. → page 7

[48] S. Siddagangaiah, C.-F. Chen, W.-C. Hu, T. Akamatsu, M. McElligott, M. Lammers, and N. Pieretti. Automatic detection of dolphin whistles and clicks based on entropy approach. *Ecological Indicators*, 117:106559, 10 2020. doi:10.1016/j.ecolind.2020.106559. → page 7

[49] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. → page 11

[50] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *CoRR*, abs/2010.02502, 2020. URL https://arxiv.org/abs/2010.02502. → page 102

[51] J. Wu, M. Khishe, M. Mohammadi, S. H. T. Karim, and M. Shams. Acoustic detection and recognition of dolphins using swarm intelligence neural networks. *Applied Ocean Research*, 115:102837, 2021. → page 7

[52] L. Xie, J. Zhu, Y. Jia, and H. Chen. Bionic covert underwater acoustic communication based on time–frequency contour of bottlenose dolphin whistle. *Entropy*, 24:720, 05 2022. doi:10.3390/e24050720. → page 46

[53] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *International Conference on Neural Information Processing Systems (NIPS)*, page 3320–3328, 2014. → page 30

Appendix A

1-channel detection using dataset B as training data

In this appendix, we show detection of dolphin whistle samples using 1-channel NNs. All results reported in this appendix are trained using a hyperparameter set developed from dataset B's only channel. The model takes 1-channel inputs, unlike the 3-channel versions in most of Chapter 2 which would take the single dataset B channel replicated three times. This exists as a baseline of performance on dataset B is and referenced in Section 2.4.5. Values shown in Table A.1 are based on the test subset (10% of total dataset).

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	111	95.00	3.75	6.25	93.65
	26	94.06	4.00	8.13	
	29	91.88	5.00	11.25	
Simple*	70	90.63	0.00	18.75	90.42
	27	91.25	1.25	16.25	
	53	89.38	3.13	18.13	
Dense161	33	96.25	2.50	5.00	96.25
	33	97.50	1.25	3.75	
	11	95.00	4.38	5.63	
Dense161*	24	91.25	1.88	15.63	90.31
	9	89.06	5.00	16.88	
	17	90.63	6.25	12.50	
Res152	2	95.00	7.50	2.50	93.75
	3	91.88	13.13	3.13	
	1	94.38	6.88	4.38	
Res152*	12	90.31	3.13	16.25	91.15
	19	91.56	1.25	15.63	
	14	91.56	5.63	11.25	
VGG19bn	6	95.31	5.00	4.38	93.96
	2	92.81	1.25	13.13	
	2	93.75	1.88	10.63	
VGG19bn*	8	90.63	3.75	15.00	91.15
	5	90.94	0.63	17.50	
	10	91.88	6.88	9.38	

"Ep." indicates the number of epochs required to reach these results.

"Avg" indicates "average" as calculated by mean.

Table A.1: Performance of models on dataset B. Comparable to results from Table 2.3.

Appendix B

1-channel detection using channel 1 of dataset A as training data

In this appendix, we show detection of dolphin whistle samples using 1-channel NNs. All results reported in this appendix are trained using a hyperparameter set developed from dataset A channel 1. The model takes 1-channel inputs, unlike the 3-channel versions in most of Chapter 2. This exists as an assessment of performance using dataset A channel 1 and its transferability to dataset B. The results are referenced in Section 2.4.5.

B.1 Results on dataset A

The models were trained using dataset A channel 1 and the test results are found in Table B.1.

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	159	89.52	7.24	13.71	90.00
	104	90.00	4.19	15.81	
	142	90.48	4.19	14.86	
Simple*	319	89.24	5.71	15.81	88.48
	392	86.95	5.33	20.76	
	439	89.24	5.14	16.38	
Dense161	3	92.38	2.10	13.14	92.10
	3	91.71	2.10	14.48	
	3	92.19	1.14	14.48	
Dense161*	1	89.43	1.52	19.62	90.16
	1	94.19	6.10	12.76	
	1	90.48	1.71	17.33	
Res152	9	91.43	5.33	11.81	91.02
	20	92.48	5.71	9.33	
	9	89.14	0.38	21.33	
Res152*	19	91.24	6.48	11.05	90.06
	24	89.05	7.24	14.67	
	25	89.90	2.10	18.10	
VGG19bn	28	95.90	3.24	4.95	90.38
	19	94.67	5.14	5.52	
	15	95.81	2.10	6.29	
VGG19bn*	2	90.67	2.29	16.38	92.00
	3	92.38	1.90	13.33	
	7	92.95	4.38	9.71	

Table B.1: Performance of models on dataset A. Comparable to results from Table 2.2.

B.2 Results on dataset B

The models started from the best iteration found in Table B.1. They were then were trained using dataset B's only channel and the test results are found in Table B.2.

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	30	92.50	5.00	10.00	93.44
	51	93.75	5.63	6.88	
	42	94.06	4.38	7.50	
Simple*	138	90.94	2.50	15.63	90.73
	97	90.31	3.13	16.25	
	135	90.94	2.50	15.63	
Dense161	12	96.88	4.38	1.88	96.98
	6	96.25	1.88	5.63	
	7	97.81	2.50	1.88	
Dense161*	1	91.56	9.38	7.50	90.31
	1	91.25	7.50	10.00	
	1	88.13	1.88	21.88	
Res152	2	94.69	3.13	7.50	95.42
	3	95.63	6.25	2.50	
	7	95.94	1.88	6.25	
Res152*	6	90.00	9.38	10.63	90.63
	7	90.63	4.38	14.38	
	2	91.25	1.88	15.63	
VGG19bn	2	94.38	0.63	10.63	95.63
	4	97.19	1.88	3.75	
	2	95.31	0.63	8.75	
VGG19bn*	2	91.88	6.88	9.38	91.46
	2	92.38	0.63	12.50	
	2	89.06	7.50	14.38	

Table B.2: Performance of models on dataset B. Comparable to results from Table 2.4.

Appendix C

1-channel detection using channel 2 of dataset A as training data

In this appendix, we show detection of dolphin whistle samples using 1-channel NNs. All results reported in this appendix are trained using a hyperparameter set developed from dataset A channel 2. The model takes 1-channel inputs, unlike the 3-channel versions in most of Chapter 2. This exists as an assessment of performance using dataset A channel 2 and its transferability to dataset B. The results are referenced in Section 2.4.5.

C.1 Results on dataset A

The models were trained using dataset A channel 2 and the test results are found in Table C.1.

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	65	91.71	8.95	7.62	91.33
	83	90.10	9.90	9.90	
	64	92.19	7.05	8.57	
Simple*	22	90.57	3.24	15.62	90.25
	18	90.57	5.90	12.95	
	20	89.62	4.19	16.57	
Dense161	3	94.76	2.10	8.38	93.78
	2	93.52	0.95	12.00	
	1	93.05	2.29	11.62	
Dense161*	9	93.24	3.05	10.48	92.79
	9	92.67	4.76	9.90	
	14	92.48	6.86	8.19	
Res152	24	93.62	2.29	10.48	93.62
	14	93.90	3.24	8.95	
	12	93.33	6.48	6.86	
Res152*	31	92.38	3.43	11.81	92.22
	16	92.67	3.43	11.24	
	20	91.62	3.24	13.52	
VGG19bn	5	93.81	2.48	9.90	93.71
	5	94.10	4.95	6.86	
	9	93.24	2.86	10.67	
VGG19bn*	11	92.76	3.43	11.05	92.48
	7	91.43	2.67	14.48	
	6	93.24	1.14	12.38	

Table C.1: Performance of models on dataset A. Comparable to results from Table 2.2.

C.2 Results on dataset B

The models started from the best iteration found in Table C.1. They were then were trained using dataset B's only channel and the test results are found in Table C.2.

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	17	90.31	9.38	10.00	89.48
	14	89.06	11.88	10.00	
	18	89.06	13.75	8.13	
Simple*	6	91.56	1.88	15.00	91.35
	5	91.88	2.50	13.75	
	7	90.63	3.13	15.63	
Dense161	1	96.88	1.88	4.38	96.88
	3	97.50	1.25	3.75	
	1	96.25	1.25	6.25	
Dense161*	5	92.81	5.63	8.75	92.29
	5	91.88	4.38	11.88	
	3	92.19	4.38	11.25	
Res152	6	94.06	2.50	9.38	94.48
	15	92.50	9.38	5.63	
	15	96.88	1.25	5.00	
Res152*	5	91.88	6.25	10.00	91.98
	4	92.19	6.88	8.75	
	6	91.88	5.63	10.63	
VGG19bn	1	96.25	0.63	6.88	96.25
	5	96.25	0.63	6.88	
	3	96.25	1.88	5.63	
VGG19bn*	5	92.50	4.38	10.63	91.46
	3	91.56	3.75	13.13	
	4	90.31	5.00	14.38	

Table C.2: Performance of models on dataset B. Comparable to results from Table 2.4.

Appendix D

1-channel detection using averaged channels of dataset A as training data

In this appendix, we show detection of dolphin whistle samples using 1-channel NNs. All results reported in this appendix are trained using a hyperparameter set developed from dataset A by averaging its two channels. The model takes 1-channel inputs, unlike the 3-channel versions in most of Chapter 2. This exists as an assessment of performance using dataset A averaged channels and its transferability to dataset B. The results are referenced in Section 2.4.5.

D.1 Results on dataset A

The models were trained using dataset A averaged channels and the test results are found in Table D.1.

Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	193	91.52	6.67	10.29	93.84
	180	95.33	2.67	6.67	
	123	94.67	2.10	8.57	
Simple*	119	92.38	4.95	10.29	92.06
	100	92.10	3.62	12.19	
	120	91.71	2.48	14.10	
Dense161	2	96.29	1.14	6.29	96.10
	3	95.43	2.86	6.29	
	2	96.57	0.76	6.10	
Dense161*	3	95.81	2.86	5.52	95.24
	2	94.29	1.33	10.10	
	2	95.62	1.33	7.43	
Res152	3	94.57	0.95	9.90	94.67
	5	94.76	2.48	8.00	
	8	94.67	4.57	6.10	
Res152*	18	93.43	0.95	12.19	94.13
	14	94.00	0.76	11.24	
	17	94.95	1.90	8.19	
VGG19bn	1	96.10	3.62	4.19	96.48
	1	96.76	2.48	4.00	
	1	96.57	3.05	3.81	
VGG19bn*	6	94.48	0.19	10.86	95.14
	4	95.81	3.05	5.33	
	4	95.14	1.52	8.19	

Table D.1: Performance of models on dataset A. Comparable to results from Table 2.2.

D.2 Results on dataset B

The models started from the best iteration found in Table D.1. They were then were trained using dataset B's only channel and the test results are found in Table D.2.

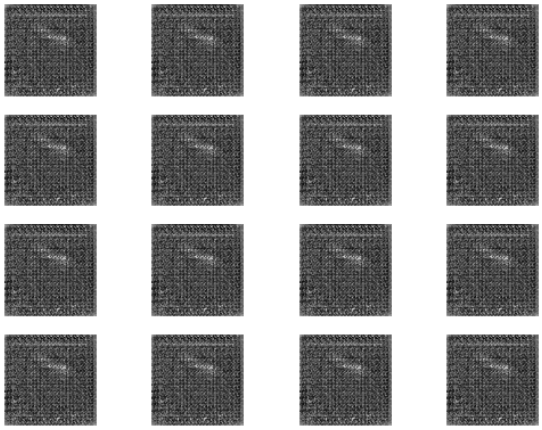
Model	Ep.	Acc (%)	FA (%)	MD (%)	Avg Acc (%)
Simple	53	94.06	6.25	5.63	93.85
	78	94.38	4.38	6.88	
	44	93.13	5.00	8.75	
Simple*	62	89.06	5.63	16.25	88.44
	48	89.69	5.00	15.63	
	75	86.56	11.25	15.63	
Dense161	2	95.63	2.50	6.25	96.15
	7	95.94	2.50	5.63	
	5	96.88	3.13	3.13	
Dense161*	2	93.13	3.13	10.63	92.92
	1	93.13	2.50	11.25	
	1	92.50	2.50	12.50	
Res152	3	95.00	4.38	5.63	94.48
	1	93.75	0.63	11.88	
	8	94.69	6.88	3.75	
Res152*	1	91.25	2.50	15.00	90.63
	2	88.75	4.38	18.13	
	3	91.88	0.00	16.25	
VGG19bn	2	95.94	3.13	5.00	96.04
	1	95.94	3.75	4.38	
	1	96.25	3.13	4.38	
VGG19bn*	3	92.19	3.13	12.50	92.08
	3	91.25	4.38	13.13	
	2	92.81	1.25	13.13	

Table D.2: Performance of models on dataset B. Comparable to results from Table 2.4.

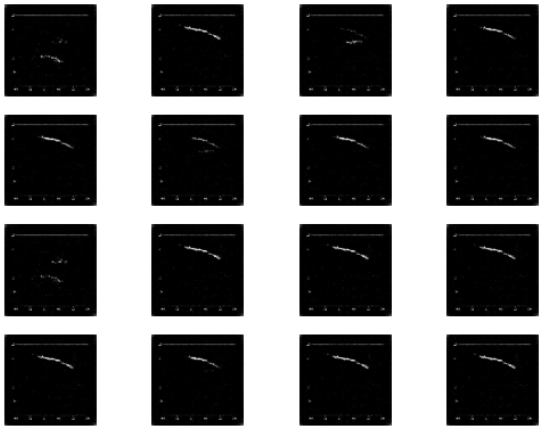
Appendix E

Checkpointed outputs by GAN

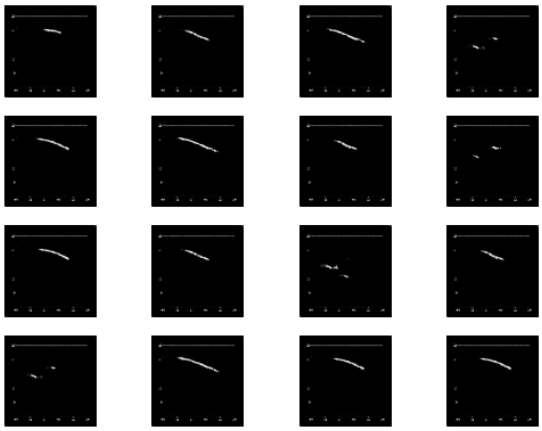
A GAN model was hypertuned and trained as specified in Section 3.3. This was used to create whistle contours that serve as the basis for generation of realistic synthetic samples. The model was allowed to train for 250 epochs and a checkpoint was taken every 25 epochs. At every step, the model state was saved and sixteen samples were randomly generated. These can be seen in Figure E.1.



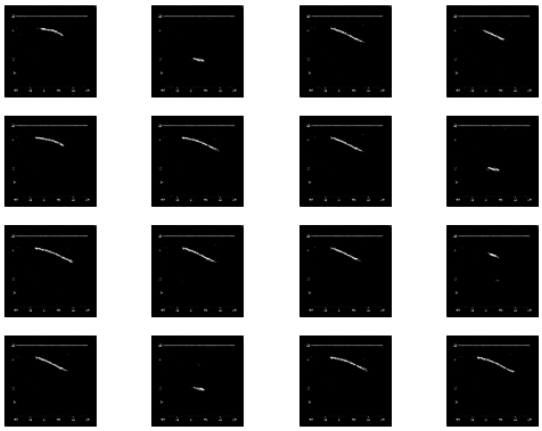
(a) Epoch 25.



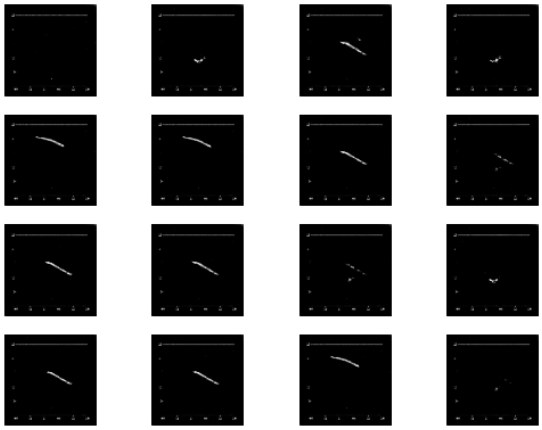
(b) Epoch 50.



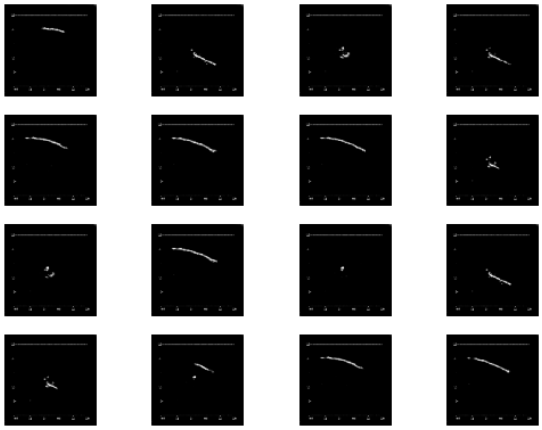
(c) Epoch 75.



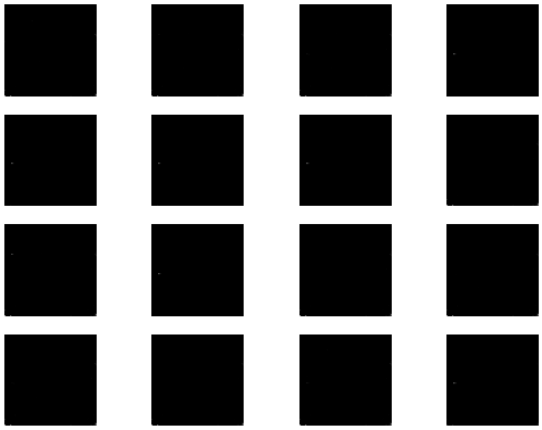
(d) Epoch 100.



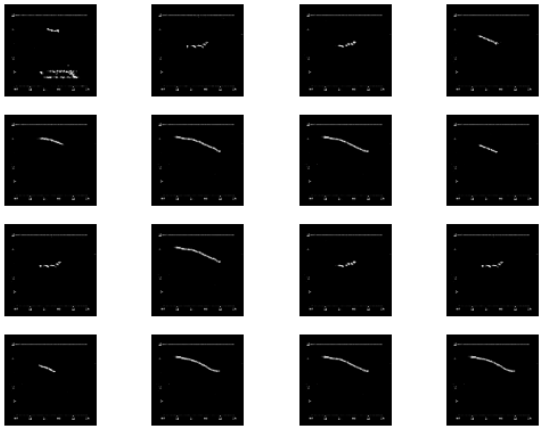
(e) Epoch 125.



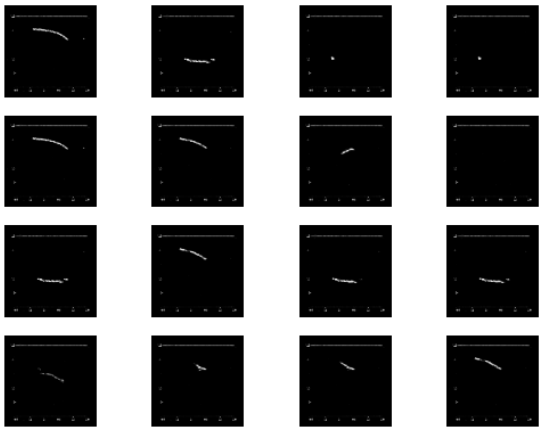
(f) Epoch 150.



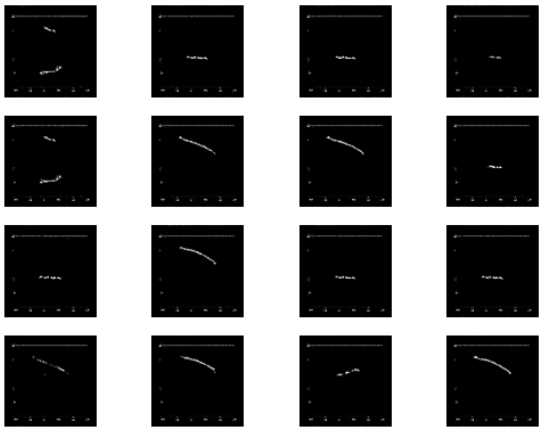
(g) Epoch 175.



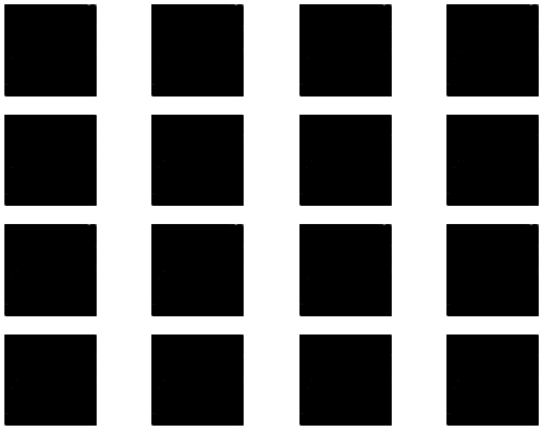
(h) Epoch 200.



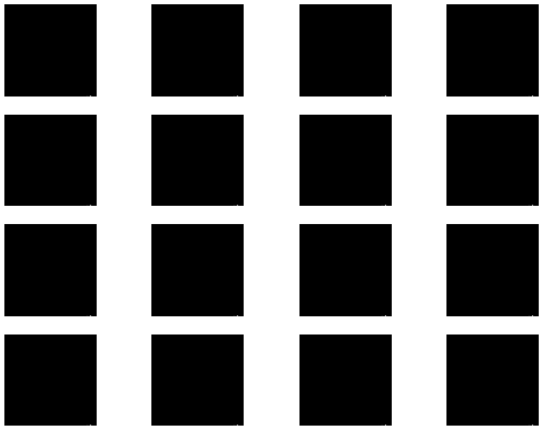
(i) Epoch 225.



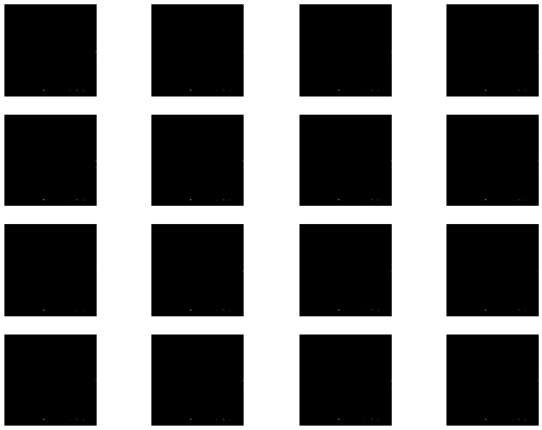
(j) Epoch 250.



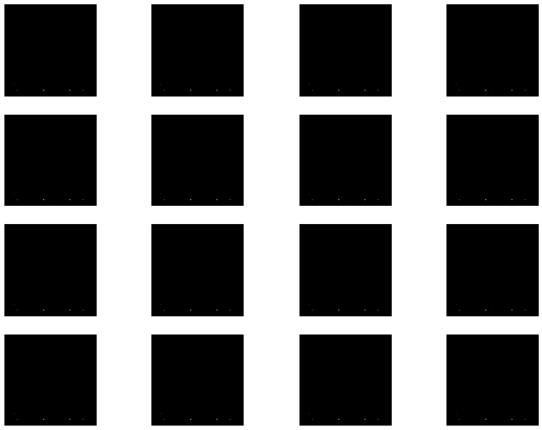
(k) Epoch 275.



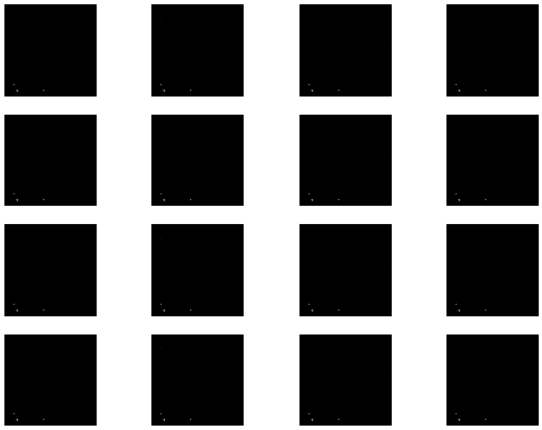
(l) Epoch 300.



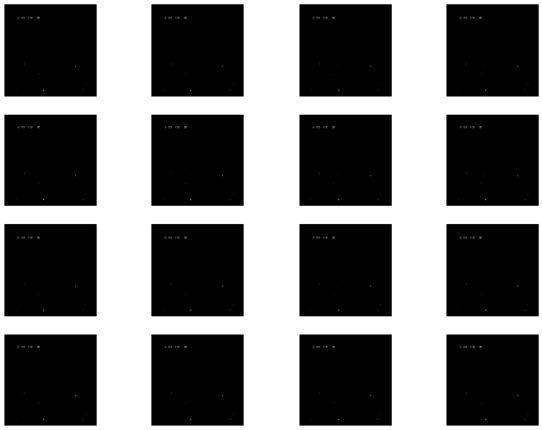
(m) Epoch 325.



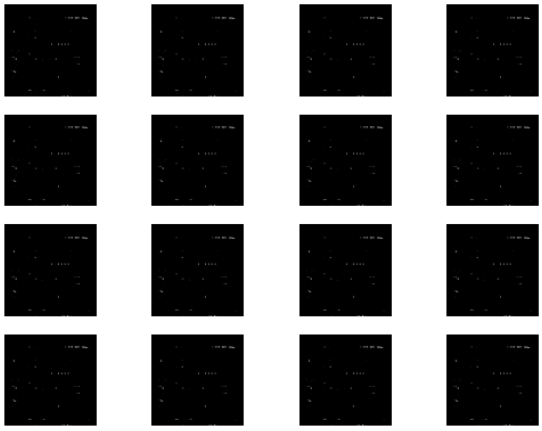
(n) Epoch 350.



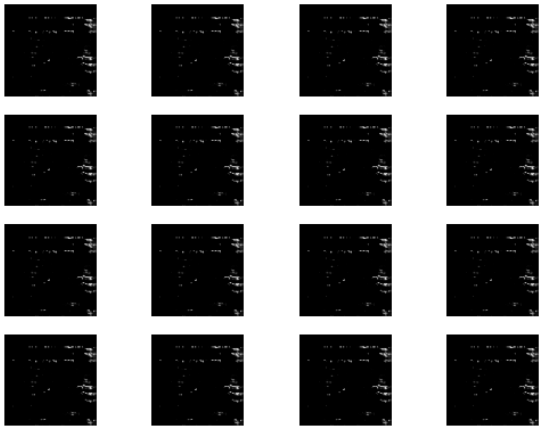
(o) Epoch 375.



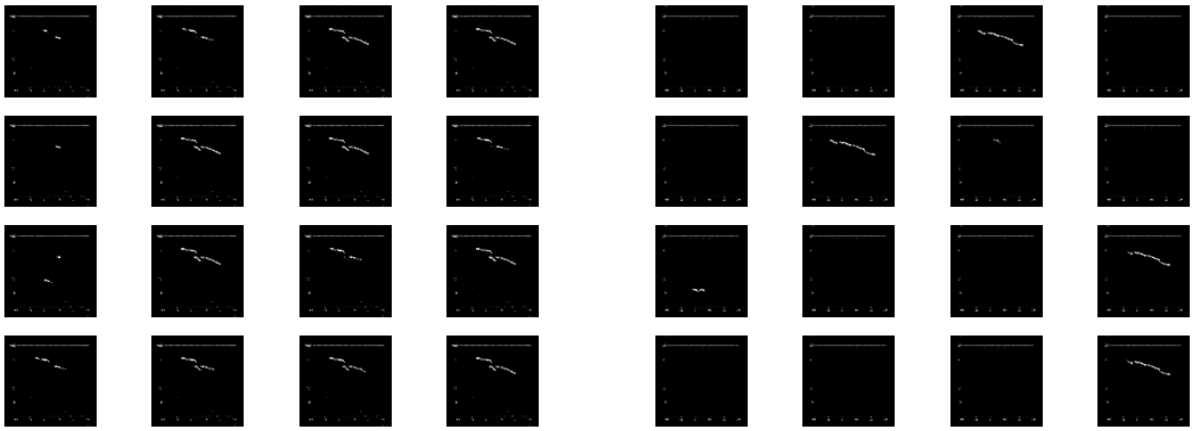
(p) Epoch 400.



(q) Epoch 425.



(r) Epoch 450.



(s) Epoch 475.

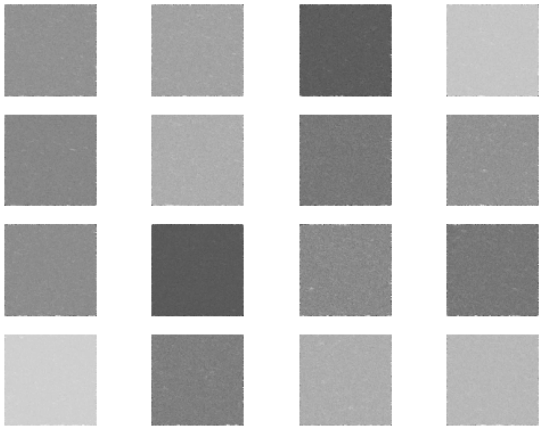
(t) Epoch 500.

Figure E.1: Multiple samples of generated outputs of GAN at epochs 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, 300, 325, 350, 375, 400, 425, 450, 475, and 500.

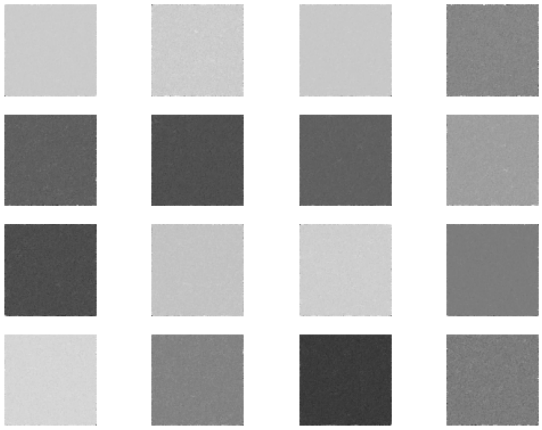
Appendix F

Checkpointed outputs by DDPM

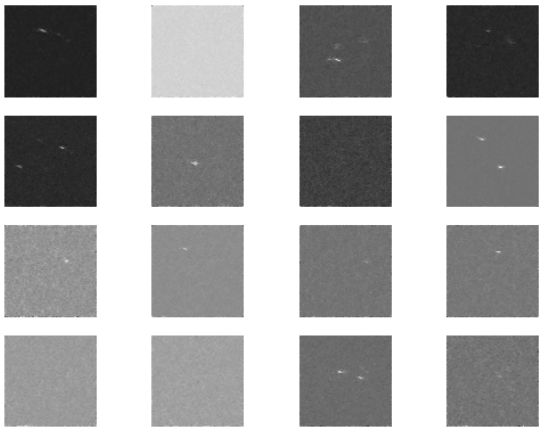
A DDPM model was hypertuned and trained as specified in Section 3.4. This was used to create whistle contours that serve as the basis for generation of realistic synthetic samples. The model was trained for 250 epochs and a checkpoint was saved every time the loss value reached a new low. At these checkpoints, the model state was saved and sixteen samples were randomly generated. Epochs aside from the last six (which can be found in Figure 3.9) can be seen in Figure F.1.



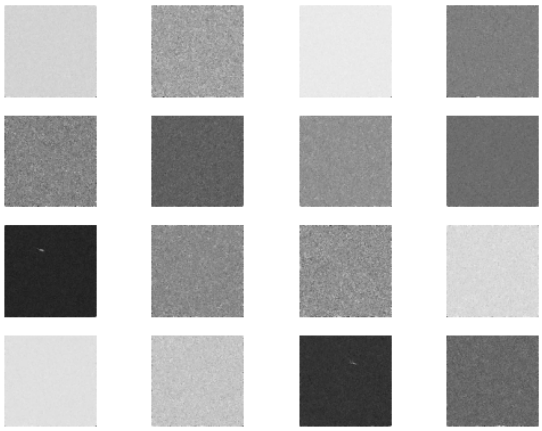
(a) Epoch 1.



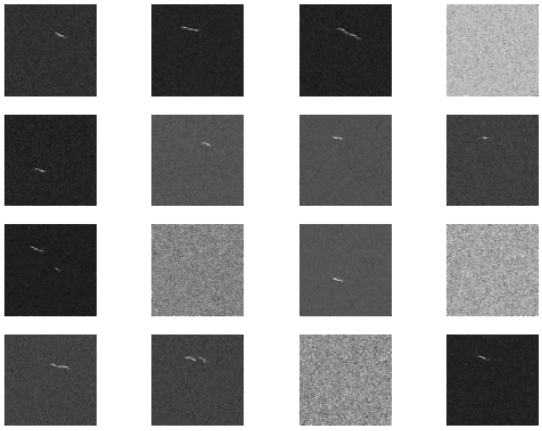
(b) Epoch 2.



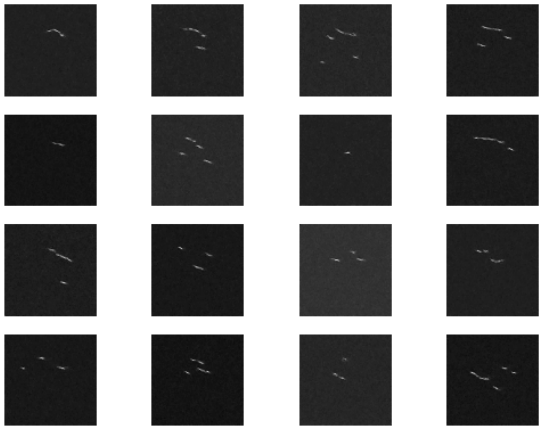
(c) Epoch 3.



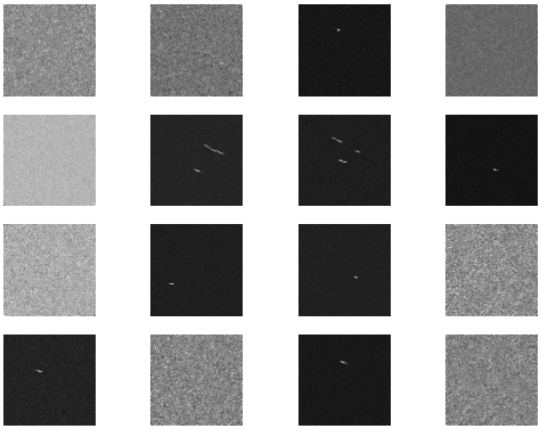
(d) Epoch 4.



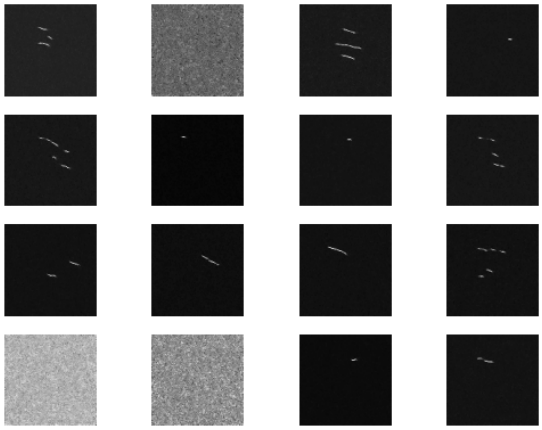
(e) Epoch 5.



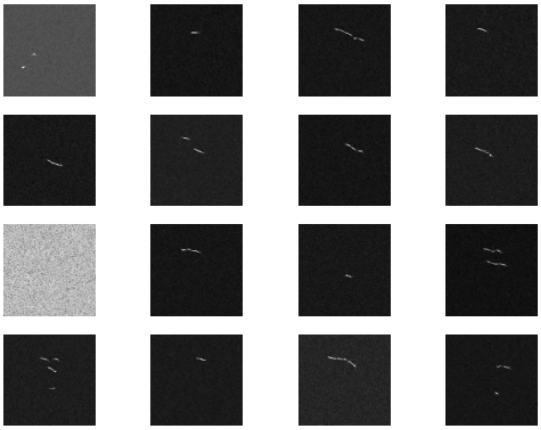
(f) Epoch 6.



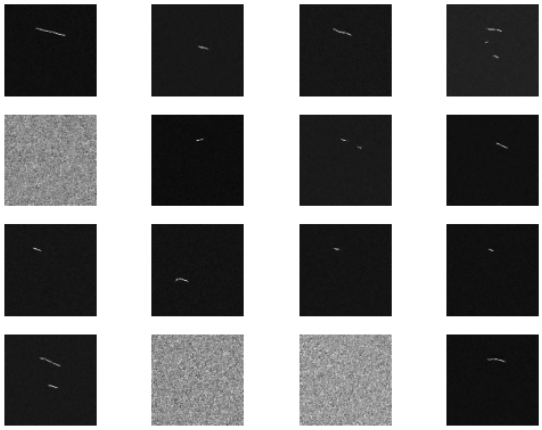
(g) Epoch 7.



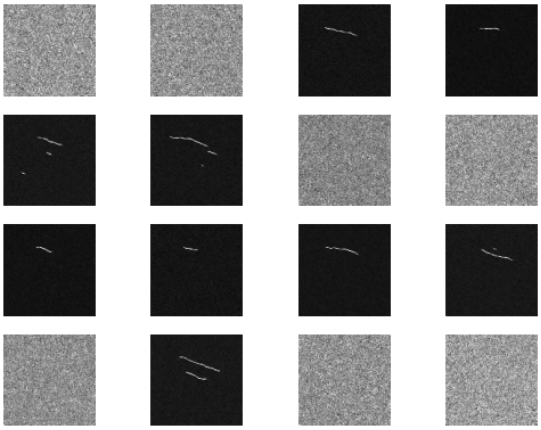
(h) Epoch 9.



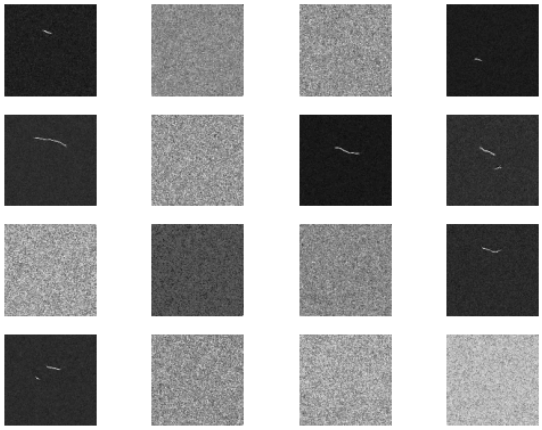
(i) Epoch 10.



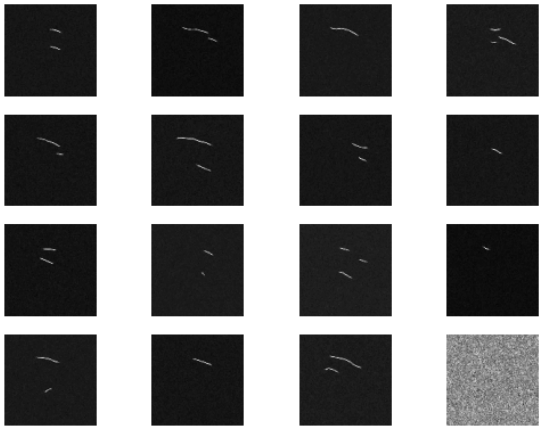
(j) Epoch 13.



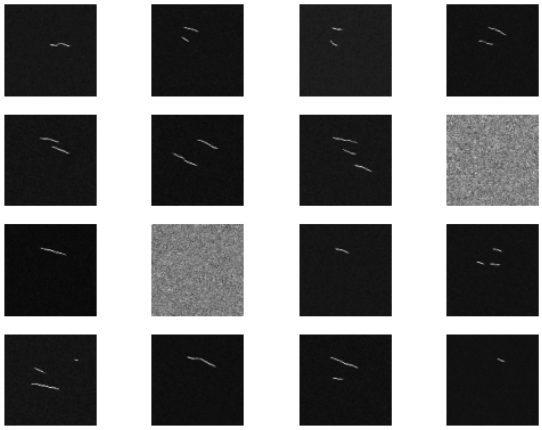
(k) Epoch 14.



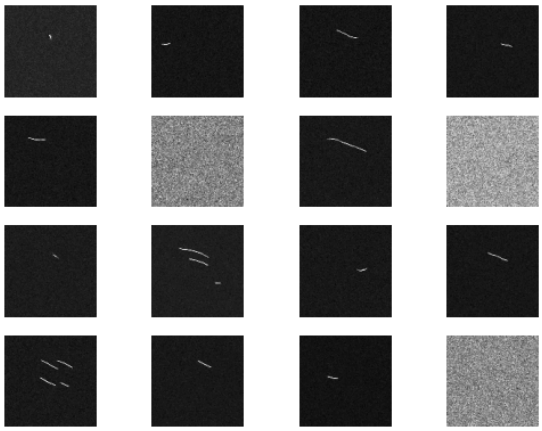
(l) Epoch 15.



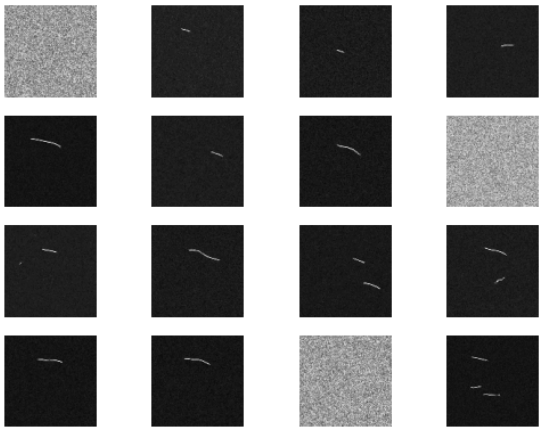
(m) Epoch 16.



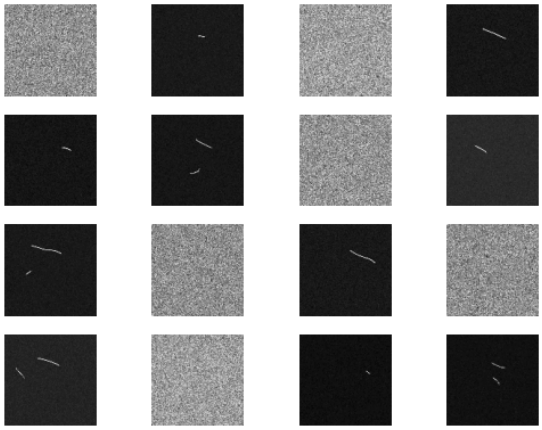
(n) Epoch 17.



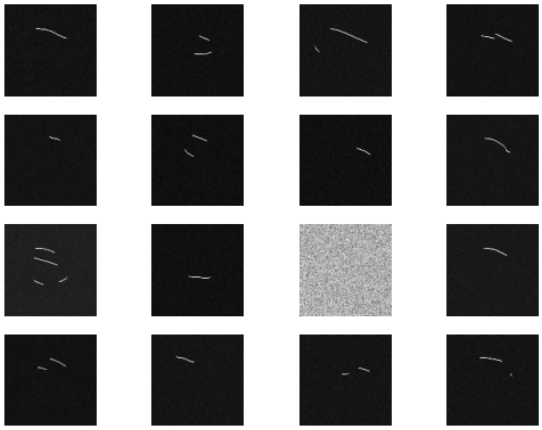
(o) Epoch 26.



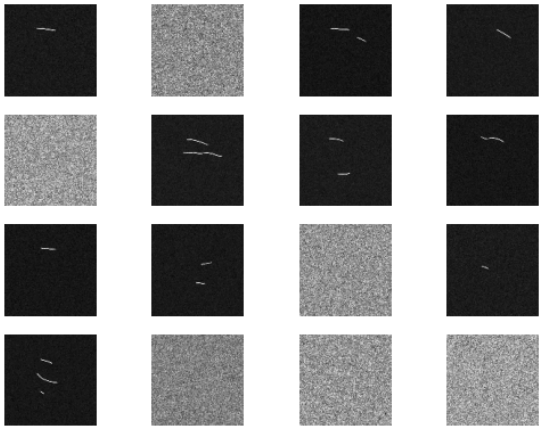
(p) Epoch 31.



(q) Epoch 40.



(r) Epoch 43.



(s) Epoch 58.

Figure F.1: Multiple samples of generated outputs of DDPM at epochs 1, 2, 3, 4, 5, 6, 7, 9, 10, 13, 14, 15, 16, 17, 26, 31, 40, 43, and 58.