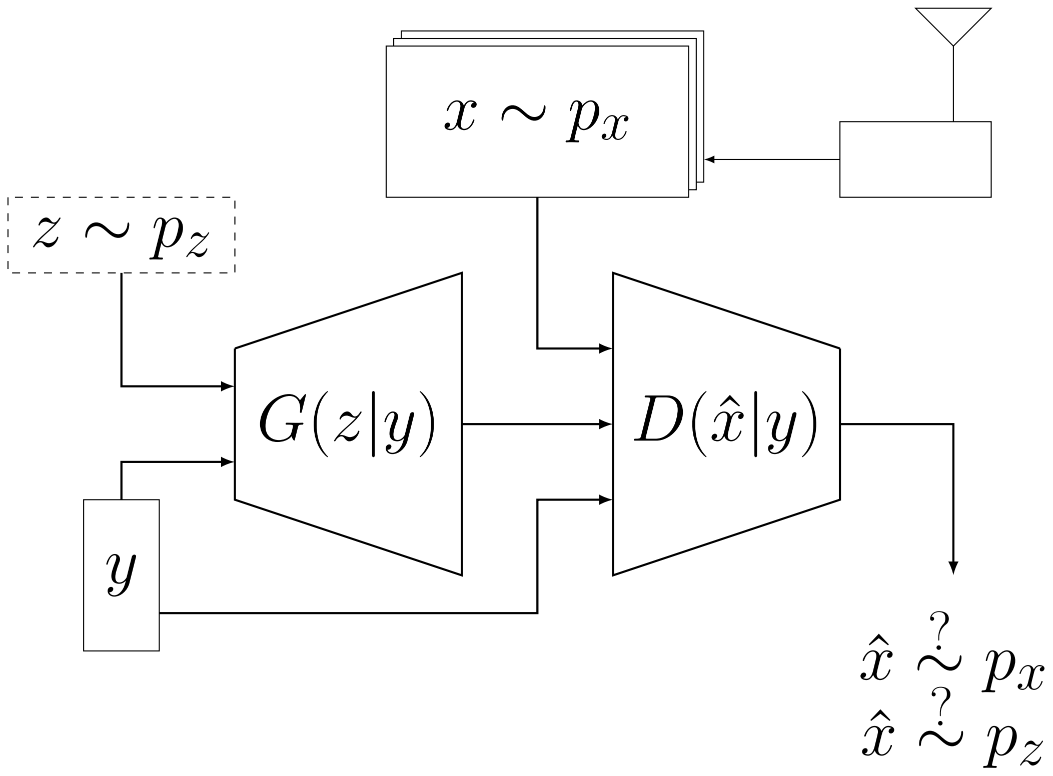
CHALMERS
UNIVERSITY OF TECHNOLOGY



$x \sim p_x$
$z \sim p_z$
$G(z|y)$
$D(\hat{x}|y)$
y
$\hat{x} \overset{?}{\sim} p_x$
$\hat{x} \overset{?}{\sim} p_z$

Generating Radar Video Data Using Generative Adversarial Nets

Master's thesis in Complex Adaptive Systems

MARTIN KAUPPINEN DAVID OLSSON

Department of Physics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:TIFX05

Generating Radar Video Data Using Generative Adversarial Nets

MARTIN KAUPPINEN
DAVID OLSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2020

Generating Radar Video Data Using Generative Adversarial Nets
MARTIN KAUPPINEN
DAVID OLSSON

Supervisor: Sven Nilsson, SAAB
Examiner: Mats Granath, Department of Physics, Chalmers University of Technology

Master's Thesis 2020:TIFX05
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Schematic of conditional GAN, taking radar data as real input.

Typeset in LaTeX
Gothenburg, Sweden 2020

Generating Radar Video Data Using Generative Adversarial Nets

MARTIN KAUPPINEN
DAVID OLSSON
Department of Physics
Chalmers University of Technology

Abstract

In development of radar tools, for example machine learning for target classification and simulation of processing chains, there is a need for large amounts of complex-valued data recorded under real conditions by a radar. However, large and properly labelled data sets of this kind are time consuming and expensive to collect. Currently such applications instead use simulated data, or smaller sets of real recorded data, limiting the development of new applications.
In this thesis, we investigate the possibility to utilise Generative Adversarial Nets (GANs) to extend existing radar data sets in order to get out of this low data regime. Existing techniques are combined and further developed for generating complex-valued radar data, with analysis of the quality of the generated data and its relevance. We conclude that in this context, GANs could be used to extend existing radar data sets, though more work is needed to make it perfectly realistic, which is non-trivial.

Keywords: neural networks, generative adversarial networks, radar, machine learning, complex valued

Acknowledgements

We would like to thank SAAB for the opportunity to work with them on this thesis. Especially our supervisor Sven Nilsson, who – along with the rest of the X Innovation Lab team – made us feel welcome and helped us utilise all necessary tools available at SAAB.

Thanks also go to our fellow thesis students at SAAB, who helped with providing valuable insights and support which helped the project move forward when we needed fresh ideas from people working in similar areas.

Finally, we would like to thank our examiner Mats Granath, for all the help he provided.

Martin Kauppinen and David Olsson, Gothenburg, May 2020

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

Abbrevation	**Definition**
ALI	Adversarially Learned Inference
CWALI	Conditional Wasserstein Adversarially Learned Inference
FFT	Fast Fourier Transform
GAN	Generative Adversarial Net
MMD	Maximum Mean Discrepancy
ReLU	Rectified Linear Unit
STFT	Short-Time Fourier Transform
WGAN	Wasserstein GAN

Notation

This section describes the meaning of certain mathematical notations used in the paper.

Notation	Description
$\underset{\boldsymbol{x}\sim p(\boldsymbol{x})}{\mathbb{E}}[f(\boldsymbol{x})]$	Expected value of a function of random vector $\boldsymbol{x}$ distributed as $p(\boldsymbol{x})$
$\langle L \rangle$	Arithmetic mean of a series, shorthand for $\frac{1}{N}\sum_{i=1}^{N} L^{(i)}$
$\lVert \cdot \rVert_2$	The ℓ^2-norm, or Euclidean norm, of a vector
$\lVert \cdot \rVert_F$	The Frobenius norm of a matrix
$\nabla_x f$	Gradient of f with respect to the elements of x, i.e. $\left[\frac{\partial f}{\partial x_1}\ \frac{\partial f}{\partial x_2}\ \cdots \frac{\partial f}{\partial x_n}\right]^T$
$\bar{\boldsymbol{x}}$	An overbar denotes a generated version of a value
$\hat{\boldsymbol{x}}$	A circumflex denotes either a combination of real and generated values, or that whether the value is generated or not is unknown

1
Introduction

Data is more important than ever, and collecting large amounts of it is crucial for many applications. It has become vital not only to record the data, but to have it labelled, and possibly even generate new data from existing data sets. This is especially true for radar data, where it might be both difficult and expensive to record enough data of the desired type, using the desired type of radar. Generative Adversarial Nets (GANs) have, primarily for images, proven to be able to solve exactly this problem; to generate new images from smaller data sets with desired attributes. This thesis report will investigate the possibility of using this technique to generate realistic raw radar video data.

1.1 Background

Radar technology is used extensively in aeronautics, marine, and military applications for detection of objects along with determining their distance from, and velocity relative to the radar. Radio signals are transmitted from the radar and are reflected off of some object, whose position and velocity can be determined through the characteristics of the reflected signal. The raw data that a radar module outputs is generally represented as complex values when converted to a digital signal and is signal-processed and data-processed in multiple ways to transform the information into something a human can more easily understand.

In times where the type of object is unknown, machine learning algorithms can be utilised in order to properly classify the radar detection. This is usually done on processed data using e.g. the movements of the object, though in theory it should be possible to do on raw data as well. A machine learning system like this would be able to faster classify detected objects, without having to wait for signal and data processing to be complete. Though this processing is fast, any time gain in defensive applications is highly valuable as it can allow for more detected objects, or more time to act. This requires large data sets for training the models.

However when working with radar data, one is often working in a low data regime, where the size of the data sets is generally too small for such models. Thus training of machine learning algorithms and testing of every processing step in the chain of processing algorithms can be difficult, especially when wanting to test a specific kind of object detection or scenario. As not every possible scenario happens often enough for it to be feasible to record, and since e.g. recording from an aeroplane is expensive, there is a need both to extend existing data sets and to develop specific scenarios. Therefore these extensions and scenarios have to be generated in some way, either through simulations or some other generative technique.

In 2014, Ian Goodfellow et al. introduced a novel technique for generating samples from a known distribution of data points using a method they dubbed GAN [1]. This method is based on two neural networks, the generator network and the discriminator network, who compete against each other during training. The generator learns to, from some noise vector in what is called latent space, generate data points that look like the known real data points, and the discriminator learns to discriminate between real and fake data points. The aim with this competition is to train such a good generator that the neither the discriminator, nor a human or any computer, can separate the generated data from the known. The technique has been used successfully for generating images, and should in principle work for radar data as well.

1.2 Aim

The aim of this master's thesis is to investigate the possibility to use Generative Adversarial Networks to generate raw radar video data, and to build a foundation on which it would be possible to use these networks in existing applications.

The research questions of the thesis are as follows:

- Can GANs be used to generate radar data from a specification?
- Is the data generated of such quality that it can be used to test real radar application software?
- Is the data generated of such quality that it can be used to train other machine learning applications?

The work delivers the following:

- An overview of current research with regard to GAN.
- A demo of a GAN that can produce raw radar data.
- Suggestions on how to proceed, either with further studies or industrialisation of the work done

1.3 Scope

In this thesis project, we will limit the scope to generating smaller cut-outs from real radar data. Specifically, the generated data will be of fixed length, and only a subset of relevant discrete range intervals will be generated. However, this should pose no problem to later integrating this into simulations by simply dropping these generated cutouts into place where the simulation engineer requires them to be.

The implemented GAN should be able to generate several different classes of objects, such as jets, propeller planes, and helicopters, but do not need to take any other conditions in account. Also, the aim is not to create the best GAN possible to generate radar data, but simply to prove that it is possible. This means that the most advanced implementations of GANs will not be used.

1.4 Previous work

The focus of much of the previous work regarding GAN has been to generate images of different kinds. One of the first improvements made to Goodfellow's original GAN was when Mirza and Osindero developed a conditional GAN [2], that for example can generate a specific number in the MNIST database of handwritten digits [3]. In 2015 Radford et al. made a Deep Convolutional GAN, with improved performance compared to the original version [4]. Multiple solutions were developed the following year, which efficiently translate a picture to the noise vector used as input to generate that specific picture. This inference function was trained while training the GAN, and the solutions were developed by Donahue et al. [5], and Dumoulin et al. [6]. Techniques to improve the networks were also suggested by Salismans et al. the same year [7].

To improve the robustness of the training, Arjosky et al. made use of minimising the Wasserstein distance [8]. Gulrajani et al. later improved this idea by using gradient penalty instead of weight clipping [9].

To apply neural networks to complex-valued data (which the raw radar data is usually represented as) one can either separate the real and imaginary parts of the data and run through a regular real-valued neural network or implement true complex-valued networks. The first option is more similar to real-valued solutions and therefore have a simpler implementation, but the options are not intuitively equivalent. A paper by Guberman [10] compare the two solutions and show equal or better performance of the split real-valued variant in experiments. The paper also gives a walk-through of the necessary complex calculus and building blocks necessary for implementing such networks.

There are previous papers for using GAN to generate raw complex-valued radar data. One notable example is [11] by Sun et al., where the authors generated Polarimetric Synthetic Aperture Radar (PolSAR) data using a complex-valued GAN. They also solve the problem of complex-valued neural networks by re-implementing the building blocks of neural networks based on the algebraic properties of complex numbers. However, as SAR is a different radar technology compared to the ground-based systems used in this thesis, the solutions for SAR are as such not quite applicable to our work. SAR data is more similar to photographs than the radar video data used in this work.

2

Theory

2.1 Neural nets

We begin this chapter with a brief refresher of neural nets. For a more in-depth understanding of this subject, we recommend [12]. Neural nets are a collection of computational nodes and connections between them, meant to model neurons and neural connections in the brain.

2.1.1 Classical

The standard, dense, feed-forward neural net consists of a number of neural layers. Each layer contains neurons that take inputs from the neurons of the previous layer and outputs the results of their computation to the neurons of the next layer. This is done by taking a weighted sum of the previous layer's outputs and then applying some sort of nonlinear function on it, called an *activation function*. Layers between the input layer and the output layer are called hidden layers. Figure 2.1 shows a diagram of this structure.

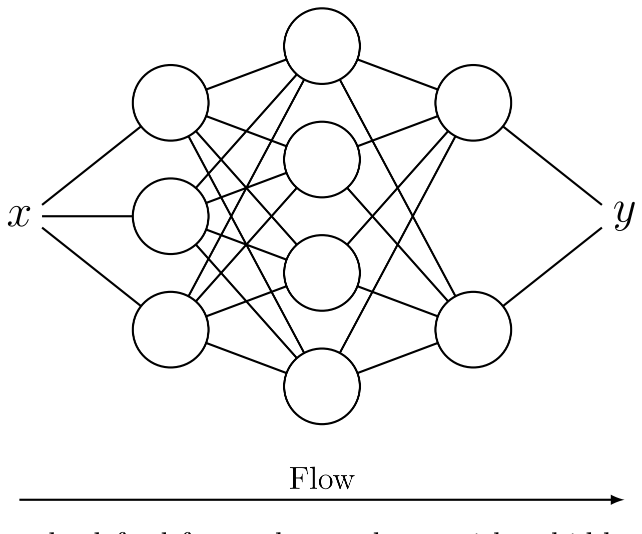


Figure 2.1: A standard feed-forward neural net with a hidden layer. Input is x and output is y, and each circle is a neuron.

This structure has been shown to perform well in e.g. classification tasks, where the input is some unlabelled data sample vector one wishes to automatically classify. This is achieved by algorithmically tweaking the weights of the weighted sums that the neurons compute such that the net performs classification well on a known, already labelled data set. Stochastic gradient descent is the standard algorithm for this, which updates the weights such that the value of an objective function is minimised, though other algorithms for this exist.

2.1.2 Convolutional

The convolutional neural net differs from the classical neural net. Instead of neurons that take weighted sums of every neuron from the previous layers, the convolutional neural net has small *kernels* that are convolved over the input – taking small, localised weighted sums over all positions on the input and constructs a feature map. Figure 2.2 shows a diagram of this. The feature maps learn to detect local features of the input. Kernel convolution can then be performed on the resulting feature maps, creating more feature maps in a deeper layer. Generally speaking, the more layers, the more complex features the net can learn. Usually a convolutional neural net has some final layer or layers that are standard dense neural layers that represent the output of the net. These neurons could for example give a specific output pattern if a dog is detected in an image.

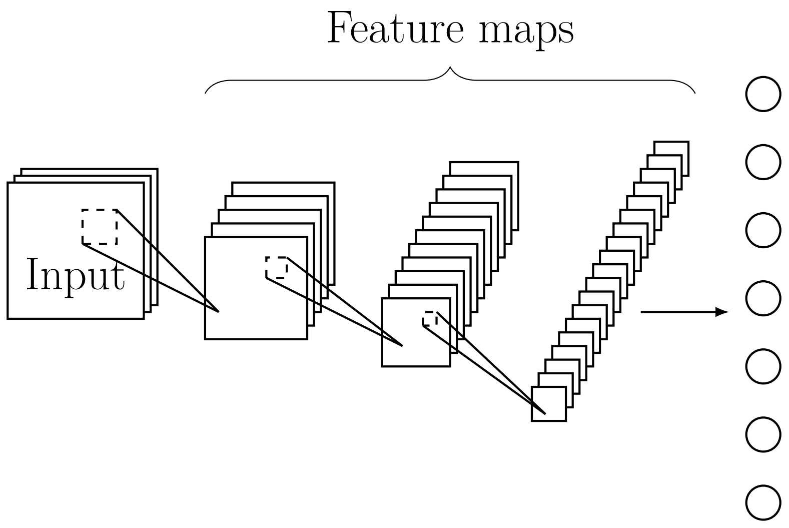


Figure 2.2: A convolutional neural net with three layers of feature maps.

Note that the kernels are not simple two-dimensional matrices, but also have a depth dimension equal to the depth of the layer being convolved over. Thus a 5×5 kernel convolved over a layer with 10 feature maps is actually a $5 \times 5 \times 10$ kernel.

Other than these structural and conceptual changes to the architecture, convolutional neural nets are trained in the same manner as the classic dense feed-forward neural nets, algorithmically minimising the objective function until a local minimum is reached.

2.2 Generative Adversarial Nets and variants

The concept of GAN was first introduced in 2014 by Ian Goodfellow et al. in [1]. In this class of neural network architectures, two networks are pitted against each other where one learns to generate fake data that looks like samples of a prior distribution, while the other learns how to distinguish between real and generated samples. The generative network (G) samples a latent noise vector $\boldsymbol{z}$ from some high-dimensional latent space $\mathcal{Z} \subseteq \mathbb{R}^n$ according to some distribution $p_{\boldsymbol{z}}$ (often an isotropic Gaussian or uniform hypercube), and outputs sample that should look similar to samples from the true sample space $\mathcal{X}$. As such G can be viewed as a (differentiable) function:

$$G \,:\, \mathcal{Z} \to \bar{\mathcal{X}} \tag{2.1}$$

The hope is that G will generate a generated sample space $\bar{\mathcal{X}}$ that is very similar to the real data space $\mathcal{X}$. The discriminator (D) then looks at a generated or real sample and outputs the probability that they are real. In other words, D is a function:

$$D \,:\, \hat{\mathcal{X}} \to [0,1] \tag{2.2}$$

$$D(\hat{\boldsymbol{x}}) \approx P(\hat{\boldsymbol{x}} \in \mathcal{X}) \tag{2.3}$$

where $\hat{\mathcal{X}} = \mathcal{X} \cup \bar{\mathcal{X}}$. Figure 2.3 shows a conceptual overview of the architecture. During training, one can imagine there being a “switchbox” between the two networks such that the discriminator network is randomly fed either a true data sample or a generated one.

The output of the discriminator network D is during training fed back to G as well as D, such that G will learn to minimise the probability of D classifying samples correctly and D will learn to maximise this probability. Thus the two networks are competing in a minimax game with value function $V(G,D)$ [1]:

$$\min_G \max_D V(G,D) = \min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}}[\log D(\boldsymbol{x})] + \mathop{\mathbb{E}}_{\bar{\boldsymbol{x}}\sim p_{\bar{\boldsymbol{x}}}}[1 - \log D(\bar{\boldsymbol{x}})] \tag{2.4}$$

where $\bar{\boldsymbol{x}} = G(\boldsymbol{z})$, and the distribution $p_{\bar{\boldsymbol{x}}}$ is implicitly defined as the latent distribution $p_{\boldsymbol{z}}$ after going through the generator. This sort of notation is used throughout this paper for brevity. The loss is a version of the log-loss, which is related to the standard (binary) cross-entropy loss often used in machine learning applications.

The authors show in [1] that this will lead to a generator network which learns to produce a distribution in such a way that the Jensen-Shannon divergence between the true and generated data distributions is minimised. The Jensen-Shannon divergence is a measure of how similar two probability distributions are, having a minimum of zero when two distributions are exactly equal, and is therefore a good measure to minimise when implementing a GAN.

As the only purpose of the discriminator is to validate whether a sample is real or generated, it is usually discarded after training. There is rarely any use of confirming the realism of samples outside of GAN training, and only the generator is needed to create new data.

Instead of a feed-forward network like in the original implementation, Radford et al. instead made use of convolutional nets in their Deep Convolutional GAN [4]. The discriminator is in that implementation a convolutional network, while the generator makes use of transposed convolution, which is a kind of convolution that creates output images with larger dimensions than the input. Another possible solution is to simply up-sample the data in the generator each layer (duplicating each data point, e.g. scaling one pixel to several) followed by convolution, which gives a similar result.

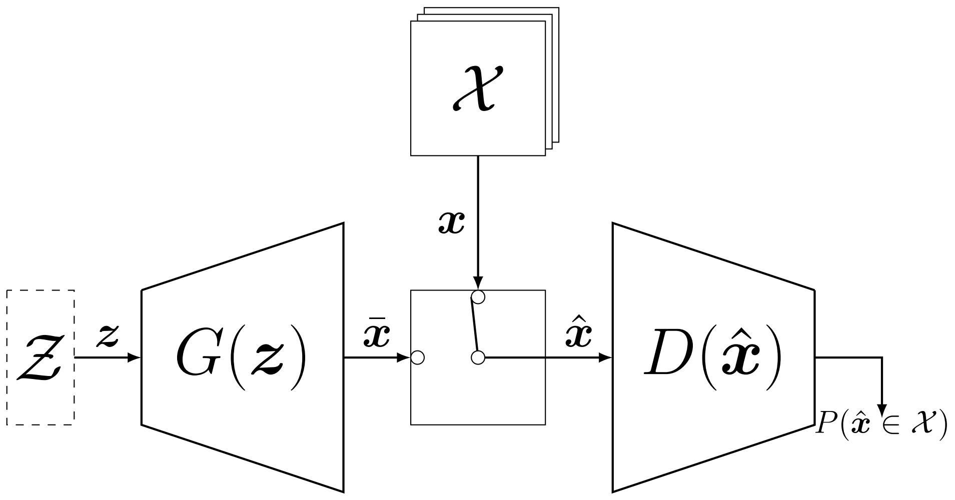


Figure 2.3: Diagram of GAN architecture. $\mathcal{Z}$ is the latent space, $\mathcal{X}$ the set of training data, $G(\boldsymbol{z})$ the generator and $D(\hat{\boldsymbol{x}})$ the discriminator. The output $P(\hat{\boldsymbol{x}} \in \mathcal{X})$ is the discriminator's certainty of $\hat{\boldsymbol{x}}$ being drawn from the training set.

2.2.1 Convergence

The minimax game between the generator and the discriminator, can also be viewed as a zero-sum game without cooperation. This means the optimum of the GAN would be a Nash equilibrium, an equilibrium where neither the generator, nor the discriminator has an action that improves its utility [13]. It has been proven that every game with a finite number of players and action profiles has at least one Nash equilibrium (proof can be found, for example, in Chapter 3 of [14]).

Unfortunately, due to its structure, this does not apply to GANs in general, as discussed in e.g. [13]. Thus there is no guarantee for the GAN to converge to an optimal equilibrium, further investigated by for example Goodfellow in [15]. There might still exist equilibrium points, and possibly even global optima, but there is no guarantee for it. One example of this, which is a common problem in GANs is called *mode collapse*, when the generator learns to generate one or a few modes of the data distribution which the discriminator thinks are real, and therefore sticks with generating only those. The discriminator on the other hand, might be restricted by step size etc. and is unable to break free from this local optimum. Consequently, GANs are heavily dependent on their many hyper-parameters to gain stability in this non-converging behaviour, meaning that developing GANs is highly non-trivial.

2.2.2 Conditional GAN

While GANs are a useful tool to generate data similar to a known distribution, in its original form the GAN generator network does not know *what* it is generating, only that it looks like it is sampled from the true data distribution. Since we in this work wish to generate different kinds of radar images based on some conditions, the generator needs to be able to be told exactly what to generate. A method for conditioning the generator and discriminator on extra information such as class labels was introduced in [2]. Along with a latent vector or data sample, an attribute vector from some attribute space $\mathcal{Y}$ is drawn. Attributes are embedded in some layer into both the generator and the discriminator. The discriminator then learns not only to determine whether a sample is generated or not, but whether it matches the attributes. Training on the discriminator output, the generator updates not only to generate more realistic samples, but samples more closely related to the attributes. The mappings thus instead look like Equations (2.5) and (2.6).

$$G : \mathcal{Z} \times \mathcal{Y} \to \bar{\mathcal{X}} \tag{2.5}$$

$$D : \hat{\mathcal{X}} \times \mathcal{Y} \to [0, 1] \tag{2.6}$$

In this version the minimax game is simply given by Equation (2.7) [2].

$$\min_G \max_D V(G, D) = \min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\text{data}}} [\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathop{\mathbb{E}}_{\bar{\boldsymbol{x}} \sim p_{\bar{\boldsymbol{x}}}} [1 - \log D(\bar{\boldsymbol{x}}|\boldsymbol{y})] \tag{2.7}$$

In practice, $\boldsymbol{y}$ can be implemented as a one-hot encoded vector.

2.2.3 Wasserstein GAN

In a standard GAN, the discriminator is often defined to learn to output values close to 0 for generated samples and values close to 1 for true samples. That is, it learns to approximately output $P(\hat{\boldsymbol{x}} \in \mathcal{X})$. The specific values are arbitrary of course, but they are bounded. However this poses a problem: if the generator finds some sample $\bar{\boldsymbol{x}}$ to generate that will always fool the discriminator, learning for the generator stops, as it will learn to always generate that specific sample. This is the mode collapse phenomenon described in Section 2.2.1, and is a very common problem when training GANs.

A variant called Wasserstein GAN (WGAN) was proposed in [8] which rather than minimizing the Jensen-Shannon divergence instead minimises the so-called Earth-Mover/Wasserstein-1 distance. In essence, if one imagines two probability distributions as piles of dirt, the Wasserstein distance is the amount of dirt and the distance one would have to move it to transform one pile into the other. In [8] it is explained and proven why it is more sensible to minimise this weaker metric than the stronger Jensen-Shannon divergence.

In practice, this means that the output of the discriminator is linear and unbounded instead of some bounded non-linearity function. The Wasserstein discriminator has the goal of instead separating the values of what it classifies as true or generated samples. For example it might try to assign negative values to real samples and positive values to fake samples, then learning to output larger magnitudes the more sure it is of its classification. This forces the generator to get out of mode collapse, as when training progresses, the outputs of real samples will grow in magnitude, and the generator has to learn to generalise and create diverse samples to keep up.

The difference in mapping compared to a standard GAN lies only in the discriminator:

$$D : \hat{\mathcal{X}} \rightarrow \mathbb{R} \tag{2.8}$$

The WGAN networks are competing in a slightly different minimax game with a value function that looks like follows:

$$\min_G \max_D V(G, D) = \min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\text{data}}}[D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\bar{\boldsymbol{x}} \sim p_{\bar{\boldsymbol{x}}}}[D(\bar{\boldsymbol{x}})] \tag{2.9}$$

In [8], the authors suggested clipping the weights of the discriminator to stay in a small interval such as $[-0.2, 0.2]$ in order to keep the parameters in a compact space. However,they also admit to this being a poor solution to enforce constraints. An improvement to the Wasserstein GAN was introduced in [9], which instead added a penalty to the loss of the discriminator based on the gradient norm of the discriminator, giving the minimax game in Equation (2.10).

$$\min_G \max_D V(G, D) = \min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\text{data}}}[D(\boldsymbol{x})] - \mathop{\mathbb{E}}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}}[D(\bar{\boldsymbol{x}})] + \lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}} \sim p_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1)^2] \tag{2.10}$$

$\hat{\boldsymbol{x}}$ here is an interpolated vector between $\boldsymbol{x}$ and $\bar{\boldsymbol{x}}$. Given a number $\alpha \in [0, 1]$, $\hat{\boldsymbol{x}} = \alpha\boldsymbol{x} + (1 - \alpha)\bar{\boldsymbol{x}}$. In essence, this will enforce a unit gradient norm along interpolation lines between real and generated data. This gradient penalty has the effect of stabilising training and avoiding the exploding/vanishing gradients problem. During training, one wants this gradient penalty term to decrease to zero and/or stabilise to indicate good and stable training.

2.2.4 Adversarially Learned Inference

In some cases, it might be useful to be able to both generate novel samples and to modify real samples in some way, adding or removing attributes etc. A framework for doing this was introduced by Dumoulin et al. in [6] known as Adversarially Learned Inference (ALI) (the same idea can be found in [5] by Donahue et al, which was developed independently). The paper achieves this by splitting the generator into two networks called the encoder (G_z) and decoder (G_x). A conceptual diagram can be seen in Figure 2.4. The decoder is the same as the standard GAN generator: a mapping from the latent space to the data space, while the encoder learns to perform the reverse mapping; that is, it maps data samples to latent space representations. In short:

$$G_x : \mathcal{Z} \rightarrow \bar{\mathcal{X}} \tag{2.11}$$

$$G_z : \mathcal{X} \rightarrow \bar{\mathcal{Z}} \tag{2.12}$$

The discriminator takes both the latent space and the samples from both the decoder and the encoder, instead of just real and generated samples, i.e. real (or at least not generated through machine learning) latent space with generated samples from the decoder, and generated latent space with real samples from the encoder:

$$D : \hat{\mathcal{X}} \times \hat{\mathcal{Z}} \to [0, 1] \tag{2.13}$$

Both the encoder and the decoder are then trying to convince the discriminator that the output comes from the other net, while the aim of the discriminator is to correctly determine which net generated what output. The minimax game is defined as:

$$\min_G \max_D V(G, D) = \min_G \max_D \mathop{\mathbb{E}}_{\boldsymbol{x} \sim p_{\text{data}}} [\log D(\boldsymbol{x}, \bar{\boldsymbol{z}})] + \mathop{\mathbb{E}}_{\bar{\boldsymbol{x}} \sim p_{\bar{\boldsymbol{x}}}} [1 - \log D(\bar{\boldsymbol{x}}, \boldsymbol{z})] \tag{2.14}$$

where $\bar{\boldsymbol{x}} = G_x(\boldsymbol{z}), \bar{\boldsymbol{z}} = G_z(\boldsymbol{x})$.

The reason one would prefer a solution with adversarially learned inference is because other generative solutions (like variational auto-encoders or auto-regressive models) can have drawbacks like smearing the output or needing excessive computational power. Training an encoder similar to the encoder in ALI post-hoc on the other hand does not give as good result. For details regarding this, see [6].

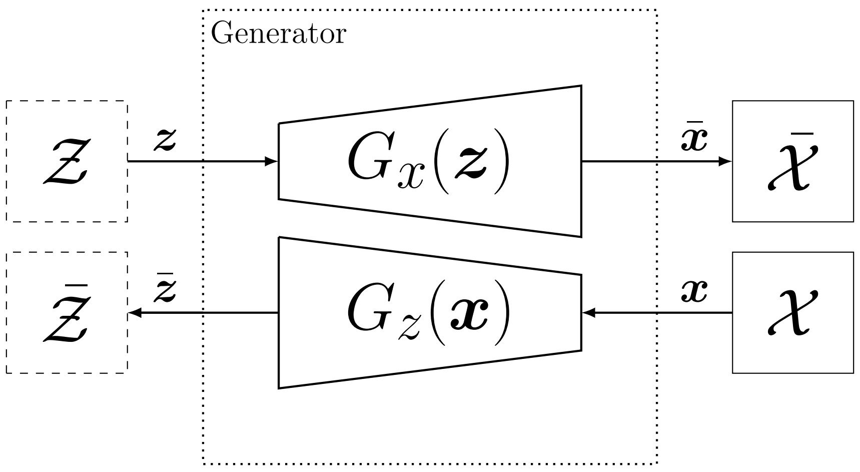


Figure 2.4: Diagram of ALI generator. $\mathcal{Z}$ is the latent space and $\mathcal{X}$ the set of training samples, and $\bar{\mathcal{Z}}$, $\bar{\mathcal{X}}$ are their generated counterparts. G_x is the decoder and G_z is the encoder.

2.2.5 Evaluation metrics

To evaluate the data quality of data samples generated by a GAN, a good metric is needed. There are many proposed evaluation metrics for GANs, but they all have some sort of drawback, intractability, or areas where they are not applicable. An overview of a lot of quantitative and some qualitative metrics can be found in [16].

One big problem with GANs is that there is no robust way to – beyond visual inspection – determine whether or not the generated data is actually any good. Two of the most common metrics in evaluating the performance of GANs are Inception Score [7] and the Fréchet Inception Distance [17], however they have some problems and have received some criticism, as they depend on the Inception model, trained on the ImageNet database, and could therefore give misleading results when applied outside this data set [18].

2.3 Radar theory

Most people are familiar with the general concept of radar, though not the details. In this section we will go through the basics of how radar works so that the reader may better appreciate where and how the complex numbers come into play in the problem. Most of the information in this section comes from [19], which is an excellent resource for the inner workings of radar.

2.3.1 Basics

At the most basic level, a radar is simply a radio transmitter and receiver. Pulses of radio waves are transmitted and the radar then listens for the echoes of these waves as they hit some object. This makes radar a powerful tool for object detection no matter the weather or time of day. If the radio waves are transmitted in narrow directed beams, the radar can determine direction and by measuring the time between transmission and reception, range can be determined. Given a few successive echoes of the same object, its velocity can be determined by exploiting the Doppler effect. The Doppler effect is the name of the phenomenon where waves emitted or reflected from a moving object will experience a shift in frequency depending on if the object is moving towards or away from the observer.

Without any sort of reference, the received radio waves of course have little meaning. At the transmitting/receiving end there is therefore a reference frequency generator with which the received signals are compared. Using this reference signal, any phase or frequency shifts can be detected and used in calculations to determine the velocity of any objects in the radar detection volume.

2.3.2 Phasors

A phasor is simply a rotating vector which can entirely represent a sinusoidal signal. The norm of the phasor is set equal to the peak amplitude of the signal, and the rotation speed of it is set to the frequency of the signal such that the phasor completes as many full revolutions per second as the represented signal's frequency. Figure 2.5 shows an example of a phasor. Looking at the projection of the phasor on the y-axis, it becomes clear how it represents a sinusoidal signal, as the projection at time t is equal to $A\sin(\omega t + \phi)$.

At the radar transmitter, the reflected signal is compared to the reference signal to determine its relative phase, denoted ϕ in Figure 2.5. The frequency of the phasor is then determined through comparing the phase of subsequent signals.

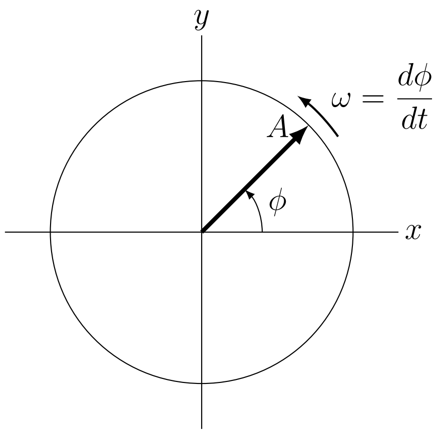


Figure 2.5: A phasor representing a signal with amplitude A, some phase ϕ, and frequency ω.

At this point it should be clear why complex numbers are used in the context of radar. Several pulses in complex polar form can represent a phasor completely.

2.3.2.1 IQ-representation

Complex numbers – and therefore phasors – can also be represented in rectangular form. The phasor is split up into its two axes: the in-phase (I) axis and the in-quadrature (Q) axis. Figure 2.6 shows these components of the phasor. When the phasor has a Q-component of 0 and a positive I-component, the received signal is in phase with the reference signal. An I-component of 0 means the received signal either lags or leads the reference signal by $\frac{\pi}{2}$ radians.

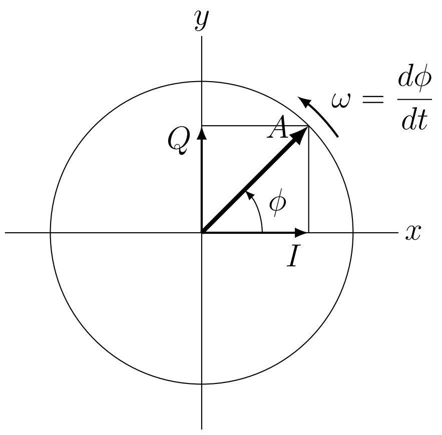


Figure 2.6: A phasor representing a signal with amplitude A, some phase ϕ, and frequency ω, with its I and Q components displayed.

The main advantage of IQ-representation is that when utilising digital Doppler filtering, having two signals corresponding to in-phase and in-quadrature enables distinguishing of the direction of the Doppler shift (away from or towards the transmitter). The data which we are using in this work is represented in IQ-format for this reason.

2.3.3 Pulse compression

For practical reasons, the pulses sent out by a radar are not of constant frequency. Instead they are coded in some way in order to increase both range and velocity resolution. One common way is chirping, where the frequency of the pulse increases linearly with time, starting from some base frequency. However chirps are harder to represent with phasors, and one would have to know the exact transmission settings of the radar in order to be able to interpret them. To combat this, giving a more universal representation of the signal regardless of transmission mode, *pulse compression* is utilised. Figure 2.7 shows how a chirp is pulse compressed, which converts it to a sinusoidal signal of constant frequency that is more readily represented by phasors and for which knowledge of the transmission mode is not needed. This works by delaying parts of the wavefront based on its frequency, resulting in a single frequency, short duration pulse with a higher amplitude than the received frequency-modulated chirp.

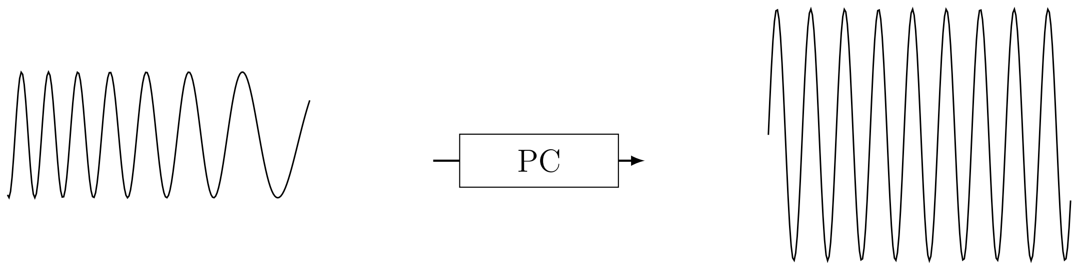
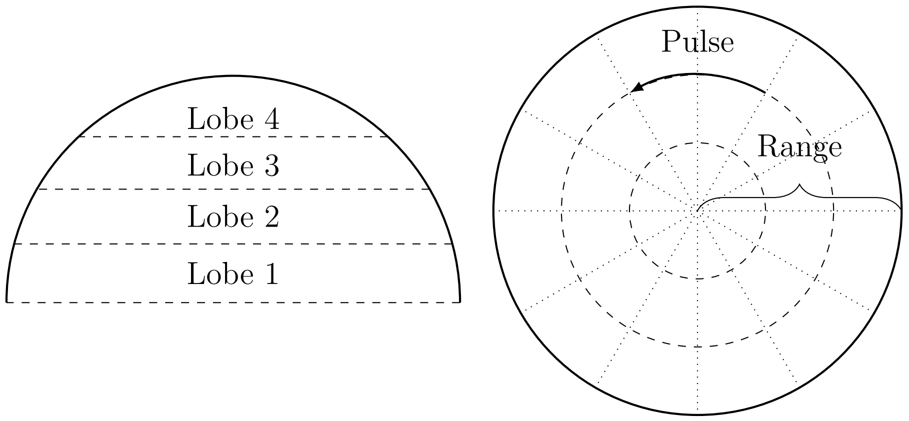


Figure 2.7: Pulse compression works by delaying the received chirp wavefront based on frequency.

The data that has been recorded for this thesis and which is to be generated is pulse compressed, precisely because this makes the data more generic and applicable regardless of radar type or settings.

2.3.4 Azimuth, altitude, range

As a rotating ground-based radar sweeps across the hemispherical detection volume, sending out pulses, it records the reflections in a three-dimensional array. The dimensions correspond to azimuth, altitude, and range. Though in this work they will instead be called pulses, lobes, and range. The intuition behind this is simply that the radar sends out signal pulses as it sweeps through the angles of azimuth, and the altitude dimension is divided up into sections for which the established nomenclature is lobes. For a rotating radar, the pulse where an object is first detected also gives the direction towards the object. However for both stationary and rotating radars, the pulse also gives the time *when* an object was detected, as the pulses are transmitted in discrete time steps. Figure 2.8 shows how the detection volume is divided up. Note that the figure has very few sectors of azimuth for clarity. An actual radar detection volume is of much higher resolution, with much smaller azimuth sectors.



(a) Side view showing lobes.

(b) Top view showing azimuth and range.

Figure 2.8: Side and top view of a radar detection volume of a theoretical rotating ground-based radar with 4 lobes, 12 pulses per rotation denoting the direction towards detected objects, and 3 range bins.

A simpler intuition for this is that the radar records a complex-valued matrix for each lobe in the detection volume, with a width representing the number of pulses or time steps, and the height of the matrix representing the range. Range intervals are discretised into bins. Each element of the matrix is then a phasor, in IQ-representation. See Figure 2.9, which shows an example of what the phasors would look like for a detected stationary object. For a moving object, the phase of the central phasors would over time rotate at a rate related to the object's velocity. In order to make this representation more meaningful, signal processing is utilised on the raw data.

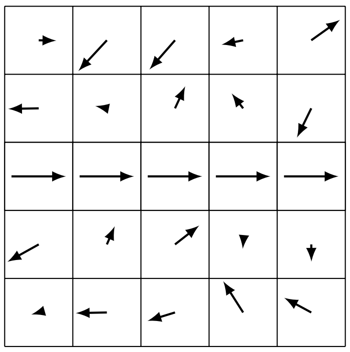


Figure 2.9: Example of a matrix of received phasors showing a stationary object in the central range bin. Each column is a sample, and each row is a range bin.

2.3.5 The Fourier transform

As radar technology deals with both time series and waveforms, the Fourier transform becomes highly relevant to determine the shift in frequency due to the radial velocity of an object. Recall the general form of the Fourier transform and its inverse:

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\, dt \tag{2.15}$$

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)e^{j\omega t}\, d\omega \tag{2.16}$$

This transform is applied to the received IQ-data on each range bin over the pulse dimension to obtain a range-Doppler plot. Figure 2.10 exemplifies the change in representation that follows applying the transform. Of course, the data being discrete samples of the continuous signals involved, the discrete Fourier transform is used instead:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{j2\pi}{N}kn} \tag{2.17}$$

$$x_k = \frac{1}{N}\sum_{n=0}^{N-1} X_n e^{\frac{j2\pi}{N}kn} \tag{2.18}$$

Note that the output of the Fourier transform ($X(\omega)$ or X_k) is complex-valued. The exact interpretation of the resulting Doppler frequency bins are dependent on the parameters governing the specific transmission mode of the radar, but in general correspond to the reflector's radial velocity with respect to the radar transmitter.

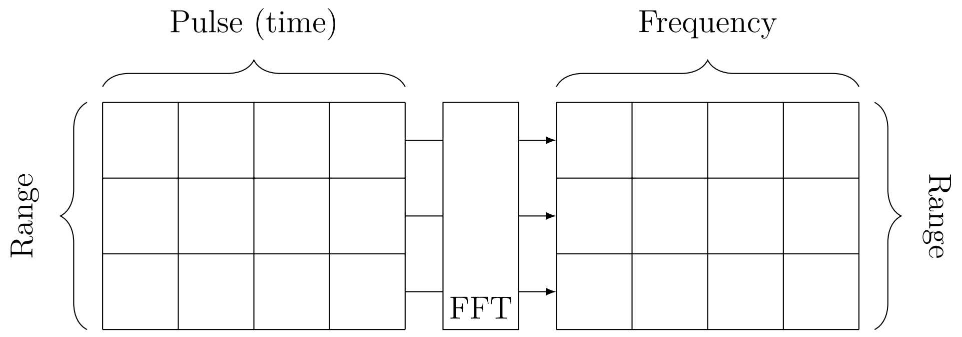
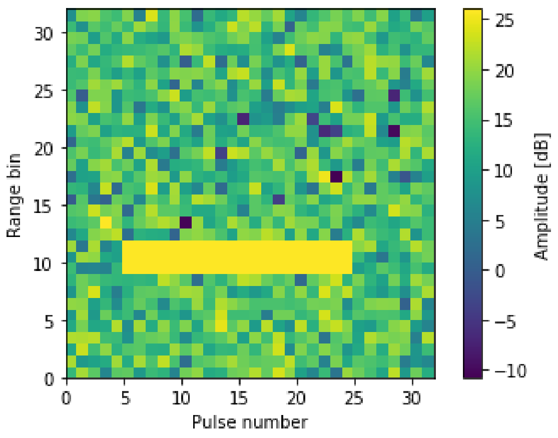
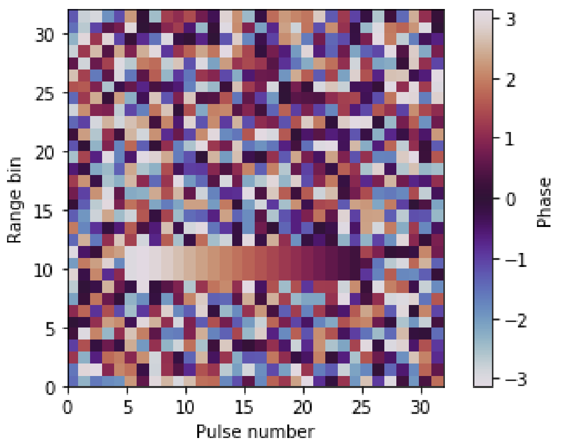


Figure 2.10: FFT is applied over time in each range bin and lobe separately.

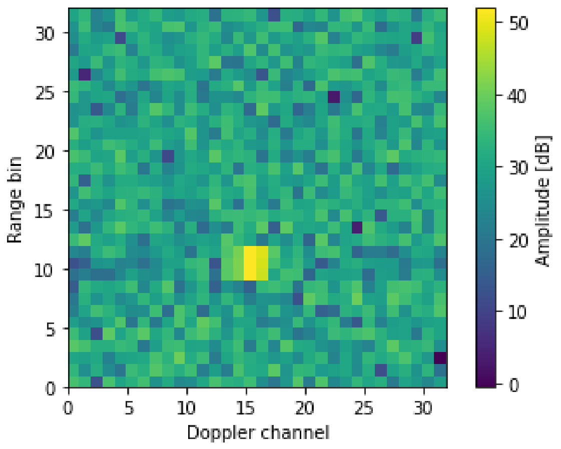
To give a concrete example, Figure 2.11 shows a simulated raw radar image and the amplitude of its Fast Fourier Transform (FFT) representation. The simulation is entirely fictional and its numerical parameters and results are not representative of any specific real radar system. In this simulated example, an "object" has been placed around range bin 10, which was detected between pulses 5 and 25 with an amplitude only slightly higher than the background noise. Taking the Fourier transform over each range bin gives a single blip in the range-Doppler plot, representing the object's velocity.



(a) Amplitude data.



(b) Phase data.



(c) Amplitude data in frequency domain.

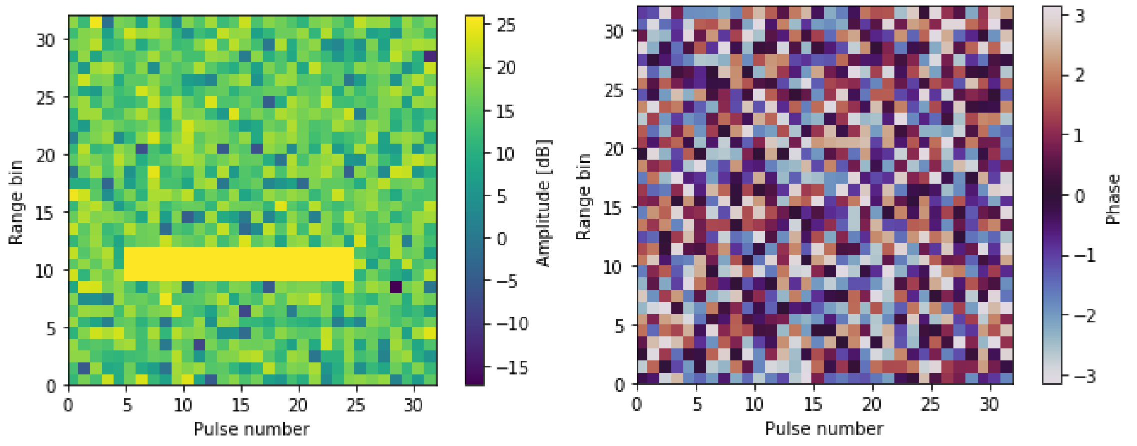
Figure 2.11: Simulated object with coherent phase. Fourier phase is discarded as it does not contain visible patterns relevant for this thesis.

No further processing has been done on this, which is usually done in real radar applications, as this example is only to demonstrate the relevance of the Fourier transform and what a typical radar detection looks like in both time and frequency space.

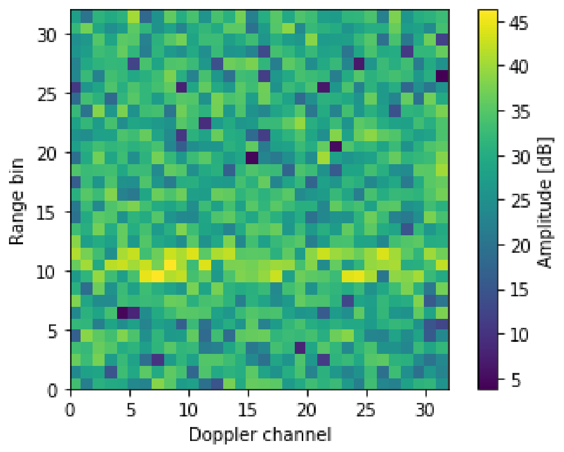
2.3.6 Phase coherence

A sinusoidal signal has not only an amplitude and frequency, but also a phase. When transmitting radar pulses, it is important to take care to make sure that the phase is *coherent*, that is, that every transmitted pulse is separated by an integer number of wavelengths. Failure to do so will cause problems with calculating the Doppler shift of the received signal, and thus by extension calculating the velocity of the detected object. In 2.11 it can be seen that where the object is located in the raw data, the phase of the complex numbers changes smoothly, and the Fourier-transformed plot shows a distinct blip in a specific Doppler channel.

However, if the simulated object is instead given a completely random phase at each point, calculating the velocity of it breaks down. In Figure 2.12 this is shown. The object becomes smeared in Fourier space and thus it is impossible to ascertain the exact velocity of it.



(a) Amplitude data. **(b)** Phase data.



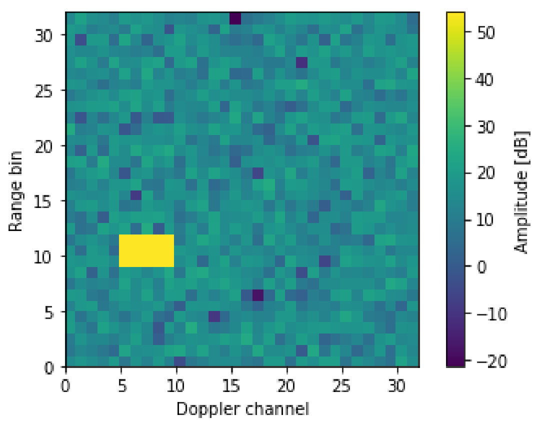
(c) Amplitude data in frequency domain.

Figure 2.12: Simulated object with incoherent phase.

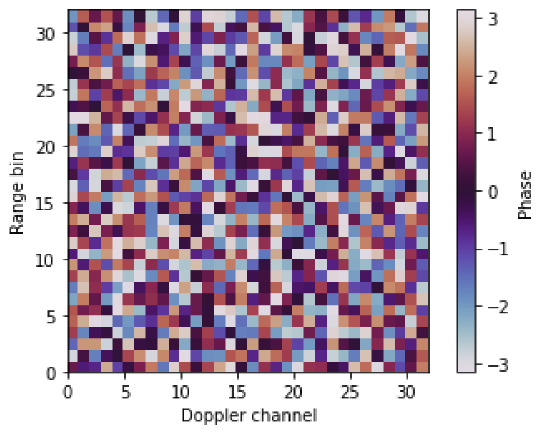
This gives a clear view of what the generated raw data should look like with regard to both amplitude and phase to be considered realistic. A realistic sample of video data should contain an object somewhere with amplitude larger than the background noise, and whose phase is coherent. Indeed, this is characteristic of the recorded data contained in our data set. Alternatively, the plot of the Fourier-transformed data should contain some clear spike in amplitude around a range bin with limited spread in the Doppler dimension. For examples of real data characteristics, see Appendix B.

The concept of phase could be discussed from an inverted point of view as well, i.e having a clear spike in the Doppler transformed amplitude, and then applying the inverse Fourier transform. It is then necessary for the time domain phase to show the coherence with the time domain amplitude, as described above. But if one simply applies the Fourier transform on an amplitude with clear spike, and a random phase, this necessary coherence does not emerge, which is shown in Figure 2.13. This means that if generating data in the frequency domain, it is necessary either for the network to learn the relation between the amplitude and the phase (or have it strictly built in to the network), or be able to generate a phase post hoc from an earlier generated amplitude.

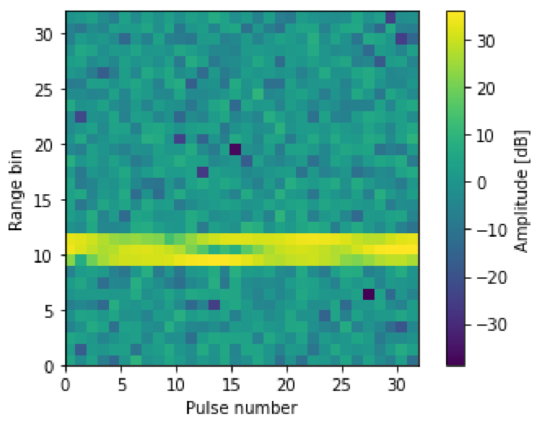
It can be noted that it is possible to generate a phase from amplitude when using e.g. Short-Time Fourier Transform (STFT)[20], a technique based on the Fourier transform. However, phase generation from amplitude is out of the scope of this thesis.



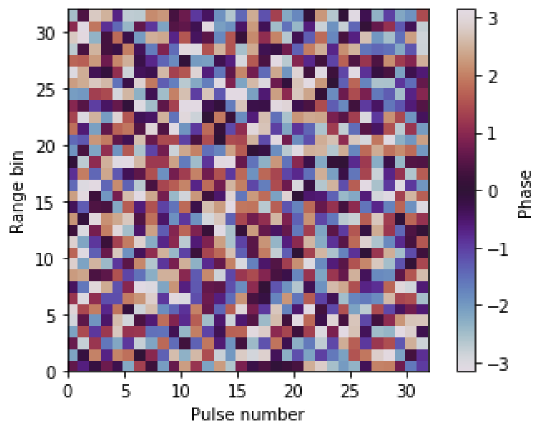
(a) Simulated amplitude data in frequency domain.



(b) Random phase data in frequency domain.



(c) Amplitude data in time domain, inverse Fourier transformed from simulated data.



(d) Phase data in time domain, inverse Fourier transformed from simulated data.

Figure 2.13: Simulated amplitude with random phase in frequency domain, showing incoherent phase in time domain.

3 Methods

3.1 Timeline

The first weeks of the project were dedicated to literature studies and smaller experiments with GAN to generate handwritten digits from the MNIST data set, along with radar studies. After that, work on generating processed radar data of drones and birds was done in order to see if GAN could be used for generating something that originated from raw radar data, the results of which can be seen in Chapter 4. This was the prestudy.

Once this was done, we turned our attention to the raw radar data and used the knowledge gained thus far to construct a GAN for generating these raw radar images. Herein lies the bulk of the project and what we base our conclusions on. Note that we use the word “raw” although this is not strictly true. The data that has been recorded and which should be generated is pulse compressed, though still raw in the sense that no further signal or data processing has been done on it. This makes the data more generic, as the properties of truly raw radar data are entirely dependent on the type of radar and its transmission settings.

3.2 Equipment

The work was done using both raw and processed IQ-data recorded from a ground-based radar. Neural networks were trained on a dedicated GPU-server at SAAB containing Nvidia RTX Quadro GPU:s.

For building the neural networks, the framework Tensorflow [21] was used for computations and specifically the Python library Keras.

3.3 Network architecture experiments

The study started by examining if there were any previous network architectures that were themselves suitable for generating the radar data. Since the data points are essentially pictures, the initial hypothesis was that one or more of the networks described in Section 2.2 would work well on the data.

3.3.1 CWALI

Based on the previously mentioned experiments we developed a hybrid model which we have dubbed Conditional Wasserstein Adversarially Learned Inference (CWALI). As the name implies, it is a hybrid of Wasserstein GAN [8, 9], ALI [6], and CGAN [2]. This combination was chosen as we needed the conditionality, wanted to avoid mode collapse, and wanted to have the option to generate latent noise from data samples. Thus the model is built up from three different convolutional networks: the encoder, the decoder, and the discriminator, where the loss function is based on Wasserstein loss with gradient penalty. Mappings are defined according to

$$
\begin{aligned}
G_x &: \mathcal{Y} \times \mathcal{Z} \to \bar{\mathcal{X}} \\
G_z &: \mathcal{X} \times \mathcal{Y} \to \bar{\mathcal{Z}} \\
D &: \hat{\mathcal{X}} \times \mathcal{Y} \times \hat{\mathcal{Z}} \to \mathbb{R}
\end{aligned}
$$

The resulting minimax game which our network plays is therefore a combination of Equations (2.7), (2.10) and (2.14) with the following value function:

$$\mathop{\mathbb{E}}_{\boldsymbol{x}\sim p_{\text{data}}}[D(\boldsymbol{x},\boldsymbol{y},\bar{\boldsymbol{z}})] - \mathop{\mathbb{E}}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}}[D(\bar{\boldsymbol{x}},\boldsymbol{y},\boldsymbol{z})] + \lambda \mathop{\mathbb{E}}_{\hat{\boldsymbol{x}}\sim p_{\hat{\boldsymbol{x}}}}[(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}},\boldsymbol{y},\hat{\boldsymbol{z}})\|_2 - 1)^2] \tag{3.1}$$

where $\bar{\boldsymbol{x}} = G_x(\boldsymbol{z}), \bar{\boldsymbol{z}} = G_z(\boldsymbol{x})$, and $\hat{\boldsymbol{x}}$ is an interpolation sample between and $\boldsymbol{x}$ and an $\bar{\boldsymbol{x}}$. The loss function is Wasserstein with gradient penalty, where the discriminator takes in an image, conditions, and a latent noise vector. Rather than using standard stochastic gradient descent in the weight updates, we elected to use the Adam optimiser [22], as this optimiser is commonly used in other GAN applications. This optimiser differs from gradient descent in that the learning rate is adapted during training. The training looks much like the one proposed in [9]. Note that $\bar{\boldsymbol{x}}$ uses the same $\boldsymbol{y}$ as the $\boldsymbol{x}$ it is compared to in the gradient penalty step (line 6-7).

Algorithm 1: CWALI training procedure.

```
Input: $m$ – Size of minibatch
Input: $n_d$ – Number of times discriminator is updated per training step
Input: $\alpha, \beta_1, \beta_2$ – Learning rate and Adam optimiser parameters
1  while $G_x, G_z$ not converged do
2      for $t = 1..n_d$ do
3          for $i = 1..m$ do
4              Sample data $\boldsymbol{x} \in \mathcal{X}$, class $\boldsymbol{y} \in \mathcal{Y}$, latent vector $\boldsymbol{z} \in \mathcal{Z}$, random
                 number $\epsilon \sim U[0,1]$
5              $\bar{\boldsymbol{x}} \leftarrow G_x(\boldsymbol{y},\boldsymbol{z})$
6              $\hat{\boldsymbol{x}} \leftarrow \epsilon\bar{\boldsymbol{x}} + (1-\epsilon)\boldsymbol{x}$
7              $L^{(i)} \leftarrow D(\bar{\boldsymbol{x}},\boldsymbol{y},\boldsymbol{z}) - D(\boldsymbol{x},\boldsymbol{y},\bar{\boldsymbol{z}}) + \lambda(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}},\boldsymbol{y},\hat{\boldsymbol{z}})\|_2 - 1)^2$
8          end
9          $D \leftarrow \text{Adam}(\nabla_D\langle L\rangle, D, \alpha, \beta_1, \beta_2)$
10     end
11     Sample batch of latents $\{\boldsymbol{z}^{(i)}\}_{i=1}^m \subset \mathcal{Z}$ and data $\{\boldsymbol{x}^{(i)}\}_{i=1}^m \subset \mathcal{X}$
12     $G_x \leftarrow \text{Adam}(\nabla_{G_x}\langle -D(G_x(\boldsymbol{z}))\rangle, G_x, \alpha, \beta_1, \beta_2)$
13     $G_z \leftarrow \text{Adam}(\nabla_{G_z}\langle -D(G_z(\boldsymbol{x}))\rangle, G_z, \alpha, \beta_1, \beta_2)$
14 end
```

The possibility to train a separate network for every specific class exists, instead of using a conditional GAN. This might even be the correct choice when having very few classes. But if the aim is to generate data from a more complex description – which in the radar context could be speed, distance, position etc. – the number of classes, and therefore networks, would grow to an unmanageable size. This makes the design choice of having separate networks for each class unfeasible, and a conditional GAN is used in this thesis.

3.3.2 Layers and activation

Although a variety of combination of hidden layers was tried, they were all based upon a general layout, which will now be described. The decoder (which in a more traditional GAN is called the generator) consists of three layers of upsampling and convolution, with the number of feature maps halved each successive layer, apart from the output layer which always had two output channels. The hidden layers have Rectified Linear Unit (ReLU) as activation function. Each layer also has batch normalisation. After the final convolution layer, a cropping layer exist to take the range bin dimension down to 21, as other solutions for upsampling to reach 21 would have meant only one upsampling step. The last part of the net splits up the two channels that have been generated so far and runs them through different activation functions, which has the effect of generating an amplitude and a phase for all our complex numbers. The amplitude part is simply run through a ReLU function, as amplitudes are strictly non-negative but potentially unbounded. The phase part is run through a scaled hyperbolic tangent function, which is bounded between $-\pi$ and π, generating a proper phase. With these two combined, the generator finally outputs a complex-valued matrix. Figure 3.1 shows a diagram of this structure.

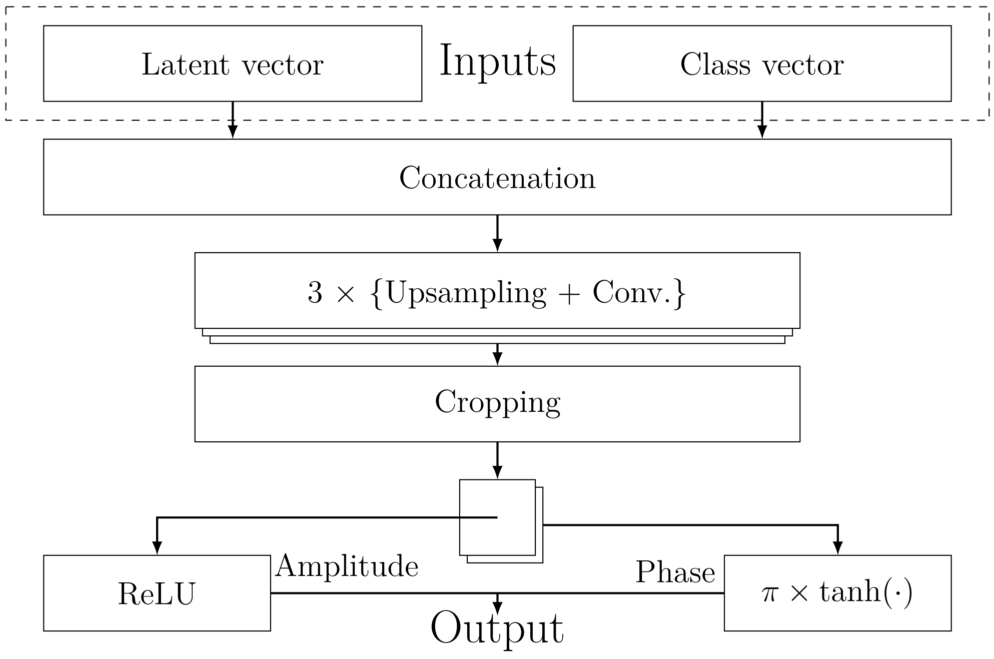


Figure 3.1: Diagram of our CWALI model's decoder.

The discriminator and the encoder both consist of four convolution layers, with leaky ReLU as activation function. At each layer the dimensions are halved, and the number of feature maps are doubled. Diagrams of these two networks can be seen in Figures 3.2 and 3.3. Initially, the number of feature maps of the first layer in the networks were set to 32 for the generator, and 16 for the discriminator and the encoder, as it was similar to the network used for images in the prestudy. However, this was soon increased considerably, to 256 feature maps for the generator, and 64 for the discriminator and encoder. More details about the network layers and parameters can be found in Appendix A.

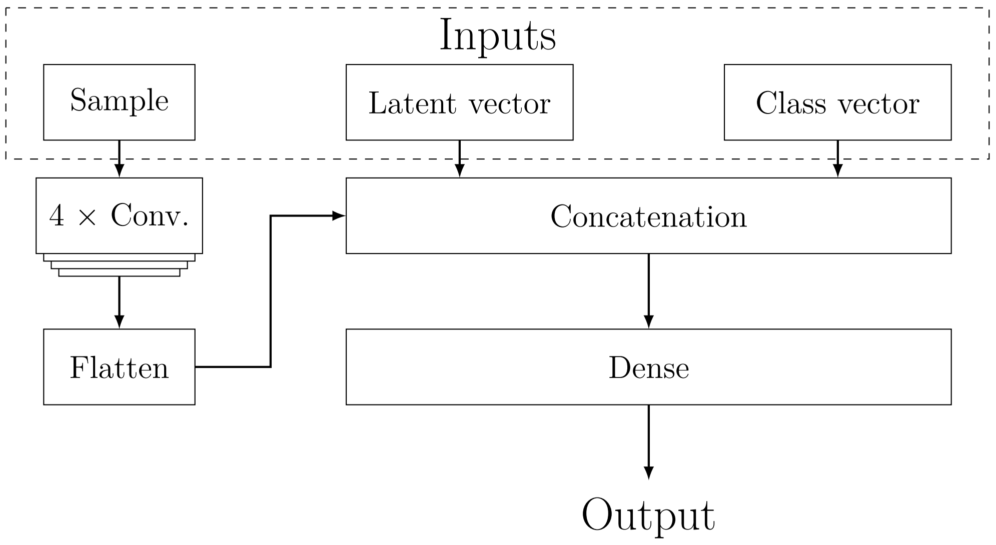


Figure 3.2: Diagram of our CWALI model's discriminator.

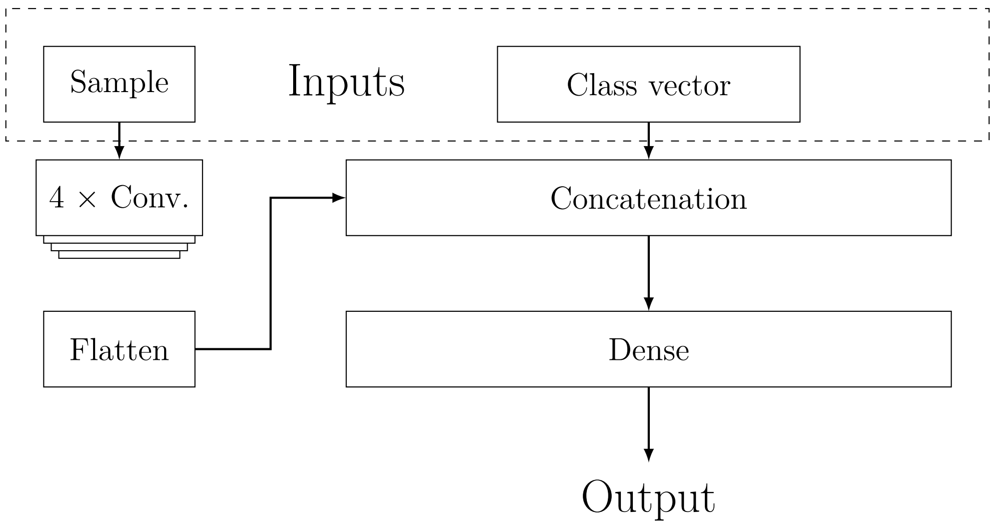


Figure 3.3: Diagram of our CWALI model's encoder.

3.4 Data set

The data used to train the networks were annotated recordings in IQ-format. Detected objects in the recordings were extracted from the recording by discarding irrelevant noise around them. These cut-outs made up the training data set, meaning each sample on IQ-form consisted of complex numbers in a matrix of 36 pulses by 21 range bins. To get even clearer training samples, only detections with a clear Doppler detection were selected, and to improve the data even further, k-means clustering (which clusters data such that the distances from the mean within clusters are minimised) was applied, and the cluster with the clearest Doppler detections was selected. The result was a training data set consisting of around 15000 samples of object detections, together with the class of the object. The classes of this data set were jet planes, propeller planes, helicopters, drones, and birds.

The data in the prestudy was based on a similar data set, but processed further. Two different classes of images were in the data set: drones and birds. They are called images in the literal sense: they were .jpg-files, which were constructed from radar data, but were not in and of themselves radar data. Constructing these images was done by picking out the column on the Doppler axis of a plot where the object is located, and appending it to an image consisting of just such columns from every plot in the sequence. This creates an image consisting of range bins and extracted Doppler channels over time.

3.5 Preprocessing

As mentioned previously the data was saved in rectangular IQ-format. One problem with this in machine learning contexts is that the numbers can be potentially very large in both the positive and negative directions of both axes. Thus, we instead convert this data to polar form and train the nets to generate the phase and logarithm of the amplitude plus one in order to improve numerics. The generated data can then be converted back to IQ form by exponentiating the amplitude, subtracting one, and converting from polar to rectangular complex form according to standard methods. In the data the amplitudes were in the range $[0, 10^6]$. Using logarithms gives a range of $[0, 14]$, a more natural range for the neural network output layer.

Algorithm 2: Conversion between log-amp and IQ before and after training.

```
Result: Generated IQ-data
1 $A \leftarrow |IQ|,\ \phi \leftarrow \arg(IQ)$          // IQ-data to polar form
2 $A \leftarrow \ln(A+1)$
3 $(A, \phi) \leftarrow \mathrm{train}(A, \phi)$          // Train nets
4 $A \leftarrow \exp(A) - 1$          // Exponentiate generated amplitude
5 $IQ \leftarrow A(\cos\phi + i\sin\phi)$          // Convert to rectangular form
```

The reason for the addition and subtraction of unity is again for the sake of numerics. Since the amplitude of a complex number is in the range of $[0, \infty)$ and $\lim_{x\to 0} \ln x = -\infty$ the net would have to generate potentially very large negative values. Adding one to the amplitude before the logarithm gives a function range that is more manageable. The equivalence can be easily seen through

$$\begin{aligned} x &\in [0, \infty) \\ \hat{x} &= x + 1 \in [1, \infty) \\ \bar{x} &= \ln \hat{x} \in [0, \infty) \\ \exp(\bar{x}) - 1 &= \hat{x} - 1 = x \end{aligned}$$

Using data on polar form in the network suggests the use of different activation functions for the amplitude and the phase in the output layer of the generator. As the phase is bounded between $-\pi$ and π, hyperbolic tangent (tanh) multiplied with π was used as activation function for the output layer of the phase. The amplitude is only limited on the lower side (to zero), so ReLU was used for the output layer of the amplitude. Alternatively, one could use a linear activation function here, letting the network learn to only output positive numbers for the amplitude. However, since amplitude is defined to be non-negative there is not much reason for doing so.

To make the data more natural to find patterns in, it is Fourier transformed in the radar signal processing chain. To make use of the same idea, a variant of the network training on Fourier transformed data was also used. The data outputted in this variant was therefore also in the frequency domain, but as the Fourier transform is invertible, raw data in time domain can easily be retrieved. See Algorithm 3. Note that the radar plots have a large mean value, meaning the lowest frequencies will give a high amplitude in the Fourier transformed plots. Since the discrete Fourier transform assumes periodicity, this effect also bleeds over into the highest frequencies, however these problems are disregarded in this thesis.

As patterns are more noticeable in frequency domain for amplitude plots , and time domain for phase plots, training on data from both domains might make the network learn more, and generate more realistic data. An additional algorithm, similar to Algorithm 3 was therefore tried, where the input data was Fourier transformed, but inverse transformed before inputted into the discriminator together with the frequency domain data.

Furthermore, a version trained only on the Fourier amplitudes was also examined.

Algorithm 3: Conversion between log-amp and IQ, with Fourier transform, before and after training.

```
Result: Generated IQ-data
1 $z \leftarrow \mathcal{F}(IQ)$                        // Fourier transform IQ-data
2 $A \leftarrow |z|,\ \phi \leftarrow \arg(z)$           // Convert to polar form
3 $A \leftarrow \ln(A + 1)$
4 $(A, \phi) \leftarrow \mathrm{train}(A, \phi)$         // Train nets
5 $A \leftarrow \exp(A) - 1$                // Exponentiate generated amplitude
6 $z \leftarrow A(\cos\phi + i \sin\phi)$           // Convert to rectangular form
7 $IQ \leftarrow \mathcal{F}^{-1}(z)$       // Inverse Fourier transform generated data
```

3.6 Evaluation

Evaluation of this GAN is necessarily more qualitative than quantitative, as there do not exist machine learning models that are pretrained on complex-valued radar data, so the most common evaluation metrics – Inception Score and Fréchet Inception Distance – are not applicable as such, given the criticisms and potential problems that have been pointed out with these measures. In order to have some quantitative measure of performance, we have chosen (squared) Maximum Mean Discrepancy (MMD) as a measure of how good the GAN-generated samples are. This measure is computationally simple and requires no separate pretrained model, as it compares generated samples directly with samples from the data distribution and computes their dissimilarity. The implementation used is the one in [23] for an unbiased estimator of the MMD.

Given a characteristic kernel function $k(\cdot,\cdot)$ which measures similarity of two samples, the MMD between two distributions $\mathbf{P}$ and $\mathbf{Q}$ is given by

$$M_k(\mathbf{P},\mathbf{Q}) = \underset{\boldsymbol{x},\boldsymbol{x}'\sim\mathbf{P}}{\mathbb{E}}[k(\boldsymbol{x},\boldsymbol{x}')] - 2\underset{\boldsymbol{x}\sim\mathbf{P},\boldsymbol{y}\sim\mathbf{Q}}{\mathbb{E}}[k(\boldsymbol{x},\boldsymbol{y})] + \underset{\boldsymbol{y},\boldsymbol{y}'\sim\mathbf{Q}}{\mathbb{E}}[k(\boldsymbol{y},\boldsymbol{y}')] \tag{3.2}$$

In practice we of course only have access to a finite number of samples for each distribution, so given sample sets $X = \{\boldsymbol{x}_1...\boldsymbol{x}_m\} \sim \mathbf{P}$ and $Y = \{\boldsymbol{y}_1...\boldsymbol{y}_n\} \sim \mathbf{Q}$, the unbiased population estimator for this is given by

$$\hat{M}_k(X,Y) = \frac{1}{\binom{m}{2}}\sum_{i\neq j}^{m} k(\boldsymbol{x}_i,\boldsymbol{x}_j) - \frac{2}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} k(\boldsymbol{x}_i,\boldsymbol{y}_j) + \frac{1}{\binom{n}{2}}\sum_{i\neq j}^{n} k(\boldsymbol{y}_i,\boldsymbol{y}_j) \tag{3.3}$$

For the characteristic kernel function, we used the rational quadratic kernel:

$$k_\alpha(\boldsymbol{x},\boldsymbol{x}') = \left(1 + \frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_2{}^2}{2\alpha}\right)^{-\alpha} \tag{3.4}$$

However since we want to check the similarity of images/matrices, we instead use the Frobenius norm, rather than the L_2 norm used for vector-valued samples. The value for α we use is 0.5, such that the kernel simplifies to

$$k(\boldsymbol{x},\boldsymbol{x}') = \left(1 + \|\boldsymbol{x}-\boldsymbol{x}'\|_F{}^2\right)^{-\frac{1}{2}} = \frac{1}{\sqrt{1 + \|\boldsymbol{x}-\boldsymbol{x}'\|_F{}^2}} \tag{3.5}$$

This has a maximum of 1 when the two matrices are exactly equal and decreases towards 0 the larger the Frobenius norm of their difference gets. This gives Equation (3.2) a value of zero when the two distributions are equal, and larger values for distributions that are different. Thus if we generate distributions that are similar to the data, Equation (3.3) should give a small value. However it should be noted that this is purely a quantitative measure, and gives no indication of qualitative performance.

Furthermore, we also check the generated samples using a neural net trained by a separate group of thesis students at SAAB using the same data for a different project, which achieved 83% validation accuracy and 90% test accuracy. This net was however only trained on two of the five classes of objects we generate and therefore gives an incomplete picture of the performance of our model.

4

Results

4.1 Processed data experiments

We created the predecessor of CWALI during the prestudy phase of the project, which was given the task of generating images similar to the ones in the data set. The validation of these generated images was done by an existing classification network trained at SAAB on these kinds of images. The network in question was trained on images of three classes: drones, birds, and noise. Since our GAN had not been trained to generate noise, this did cause some issues. However, the results were still promising. In Table 4.1, the confusion matrix of the classification network can be seen. The rows correspond to what our GAN was asked to generate, and the columns to what the classification network classified the images as. Our GAN generated 512 examples of each image.

	Drone	**Bird**	**Noise**
Drone	512	0	0
Bird	23	351	138
Noise	-	-	-

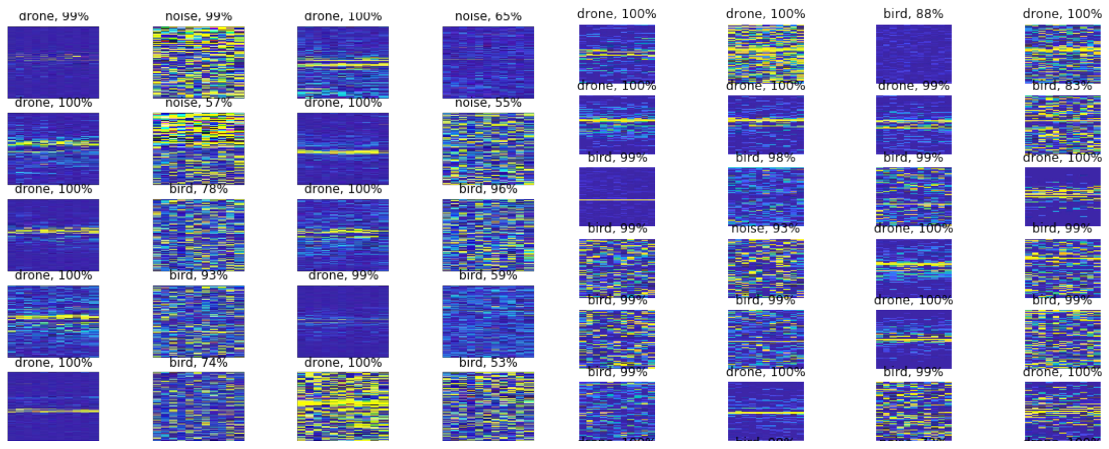
Table 4.1: Confusion matrix of classification network. Rows represent the generated class and columns represent what the classification network classified the samples as. E.g. 23 generated birds were classified as drones.

As the confusion matrix shows, our GAN managed to entirely fool the classification network when it came to generating drones, but had a bit more trouble generating birds, since it had not been trained on noise and thus could not distinguish between birds and noise. It is not surprising that it is harder to distinguish birds and noise than any other combination, as birds in general give a weaker radar reflection, and therefore inherently resemble noise. In Table 4.2, the average confidence of the classification network is shown, along with how many times it misclassified each class.

	Confidence	**# of misclassifications**
Drone	99.95%	0
Bird	75.65%	161

Table 4.2: Confidence of classification network, given 512 examples of each class.

Clearly this GAN was able to well generate novel samples from the given image data set. A comparison between the training data set and generated data can be seen in Figure 4.1. This result is perhaps not very surprising; as GANs have been used extensively in the field of machine learning to generate image data with great success. This result thus aligns with other research results regarding GANs, and showcases the strength of this generative technique when it comes to generating images; as these images are not typical photographs. However these processed images were not the main aim of the project to generate.



(a) Generated samples. **(b)** Training samples.

Figure 4.1: Comparison of training data and generated data from prestudy. Above each image is the classification and confidence given by the classification network. For generated samples, images generated as drones are in odd columns, and birds in even.

4.2 Raw data

The results from generating raw data are split into plots of training metrics together with visual inspection of the result, and other kinds of more data-driven analysis of the results of the best performing networks. The amplitudes of certain figures of this chapter have been redacted as they are deemed confidential.

4.2.1 Training and generation

When instead investigating the raw, complex valued video data, the first experiments were conducted with the naïve approach of training on IQ-data in time domain (i.e. no preprocessing) and with in principle the same design as the architecture that rendered good results in the prestudy. Unfortunately the results were not good, as seen in Figure 4.2. There are no objects, and there are clear signs of mode collapse.

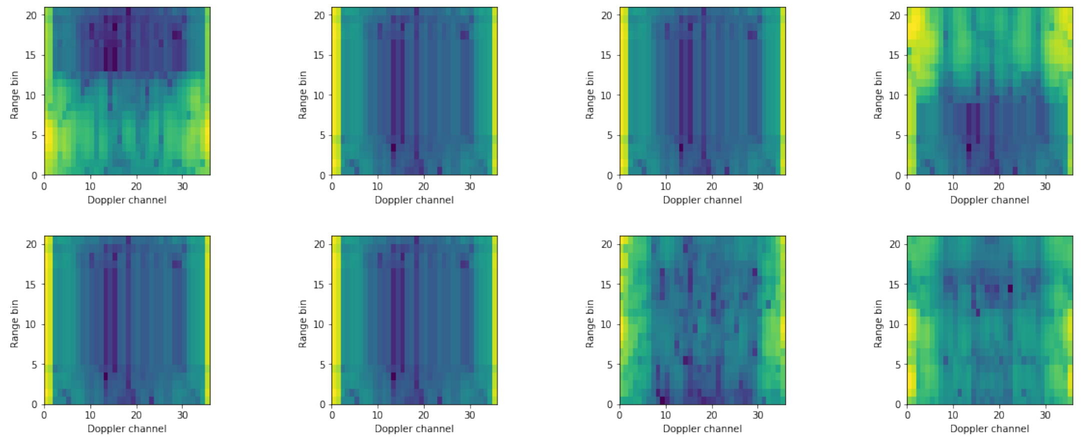


Figure 4.2: A selection of range-Doppler plots, displaying amplitude in frequency domain, from generated data. Narrow network trained on IQ-data. No clear objects visible. Amplitudes redacted.

As the networks in the prestudy were rather narrow (as in having few feature maps each layer), making the networks wider was also tested. This improved results, as seen in Figure 4.3. There are areas with higher amplitude, although the appearance does not resemble real data; the objects are for example more smeared out in the range dimension. As this wider type of networks anyway seemed to improve the result, that architecture was kept for the following experiments.

It can also be noted that there are signs of phase coherence in the time domain visible in Figure 4.4, meaning the network is able to handle the connection between the amplitude and the phase. It could be clearer, and the existence of some coherence with a decently generated Doppler amplitude is expected, as the data was generated in the time domain; the time domain phase contain necessary information for the frequency domain amplitude, as explained in Section 2.3.6.

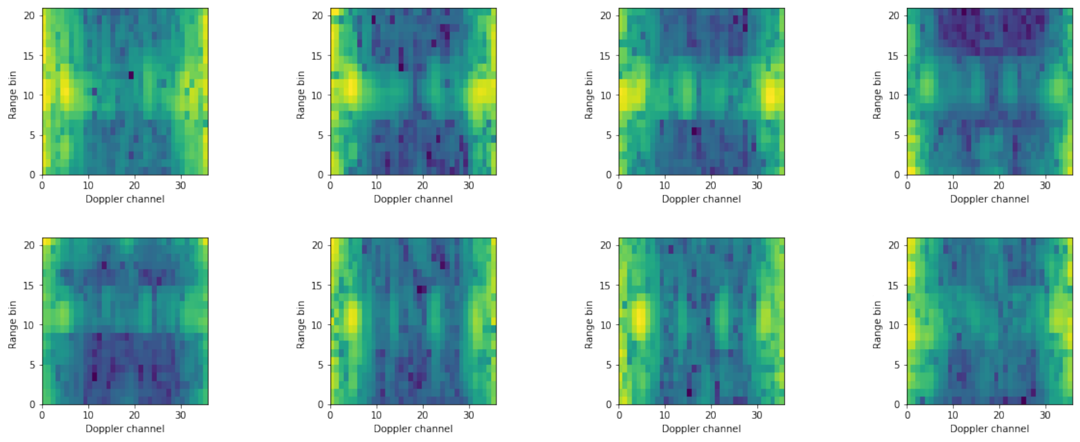


Figure 4.3: A selection of range-Doppler plots, displaying amplitude in frequency domain, from generated data. Wide network trained on IQ-data. Objects vaguely visible. Amplitudes redacted.

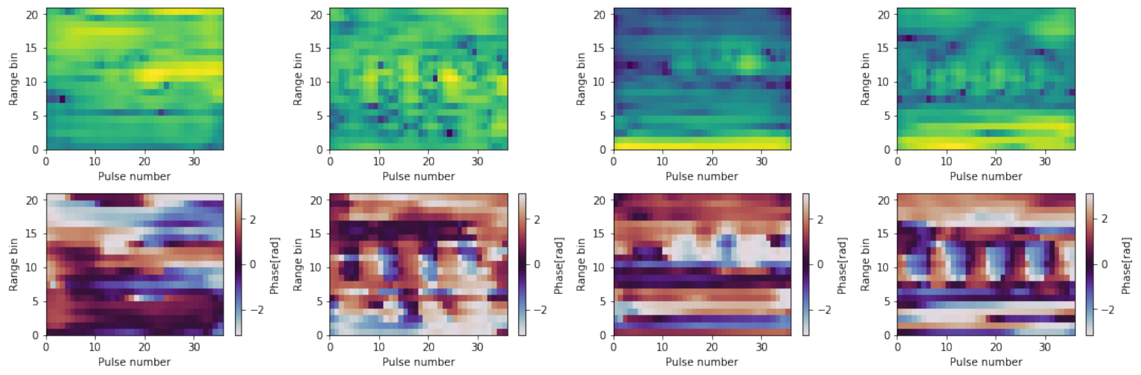
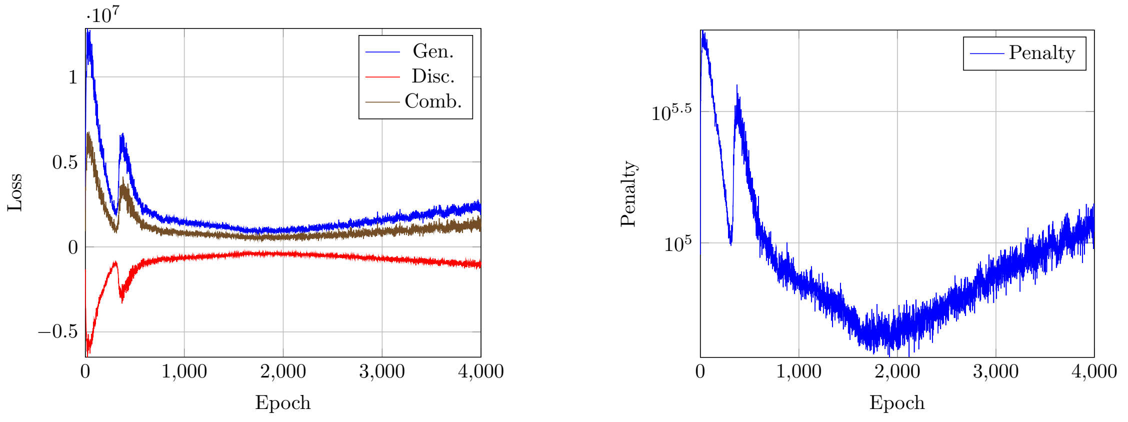


Figure 4.4: A selection of range-pulse plots, displaying amplitude and phase in time domain, from generated data. Wide network trained on IQ-data. Some phase coherence at objects can be noted. Amplitudes redacted.

One difference from the prestudy using images can now be noted; images have a known range of possible values, which the IQ-data does not have. The images in the prestudy were for example in the range [0,255] (and rescaled to be between [-1,1]), while the IQ-values are in the range $[-10^5, 10^5]$. This makes the plots for loss and penalty (see Figure 4.5) harder to analyse, as they also take very large numbers, and it also possibly makes it harder for the networks to learn. The decrease in loss and penalty was also less than hoped for, neither going to zero nor stabilising as discussed in Section 2.2.3.



(a) Loss of networks separately and combined.

(b) Value of gradient penalty in discriminator training. Note the large numbers on the loss axis.

Figure 4.5: Metrics of a wide network trained for, and generating, data on rectangular form.

The natural step was then to limit the data using input data on polar form and using the logarithm of the amplitude. This gives more reasonable dynamics of the networks, but the generated result is not ideal. The objects in the frequency domain amplitude, shown in Figure 4.6, are smeared in the Doppler channel dimension, coming from the time domain amplitude not being smooth enough. Also, the phase shows little coherence in the time domain, which can be seen in Figure 4.7. The aim of having metrics on a more manageable level was on the other hand fulfilled, as seen in Figure 4.8

When instead training (and generating) Fourier transformed data, the results were more promising, at least by visual inspection, see Figure 4.9. The metrics also show a reasonable behaviour when training, with diminishing loss and penalty until it does not change, as seen in Figures 4.10 and 4.11. The accuracy in Figure 4.12 shows a behaviour where the generator is successively improving in comparison to the other nets. Accuracy is defined for the discriminator as average percentage of correct classifications, while accuracy for the generator is the average percentage of generated samples that were classified as real.

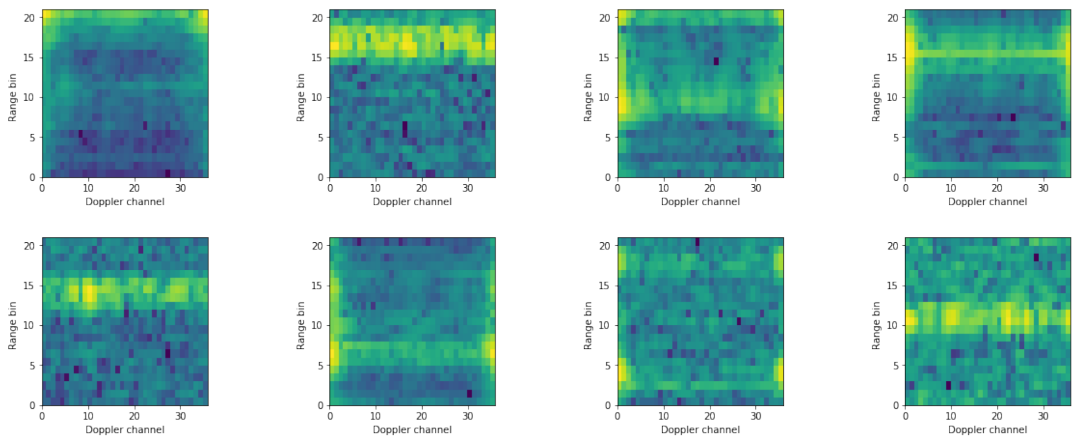


Figure 4.6: A selection of range-Doppler plots, displaying amplitude in frequency domain, from generated data. Network trained on polar data with logarithmic amplitude. Objects vaguely visible, but smeared in Doppler channel dimension. Amplitudes redacted.

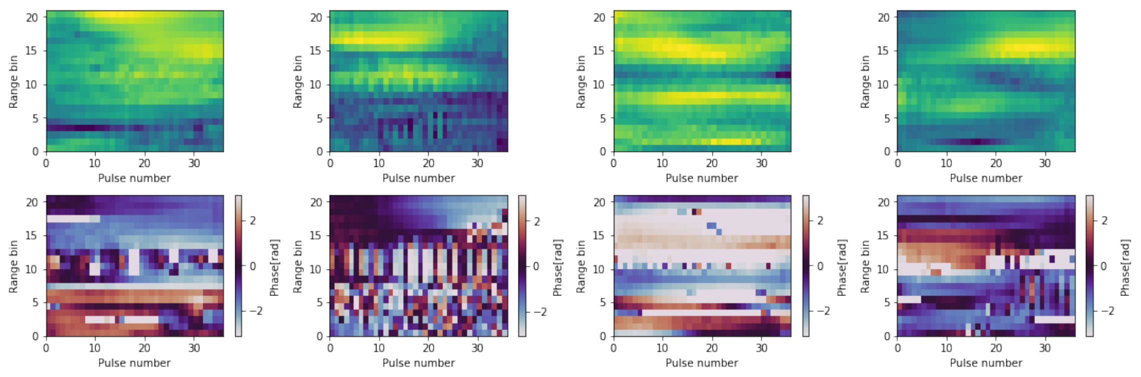
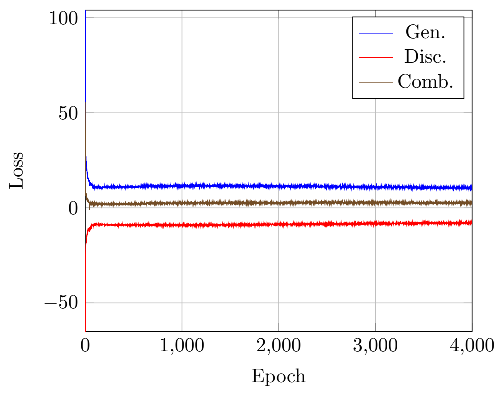
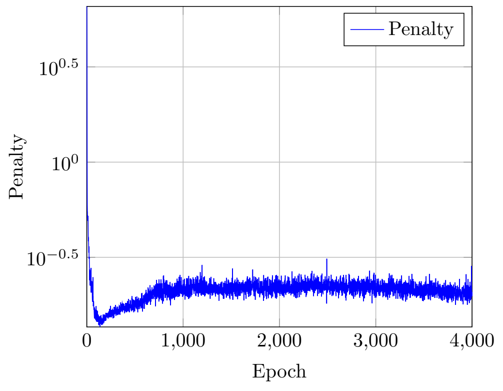


Figure 4.7: A selection of range-pulse plots, displaying amplitude and phase in time domain, from generated data. Network trained on polar data with logarithmic amplitude. Little to no phase coherence at objects. Amplitudes redacted.



(a) Loss of networks separately and combined.



(b) Penalty in discriminator training.

Figure 4.8: Metrics of a network trained for, and generating, data on polar form with logarithmic amplitude.

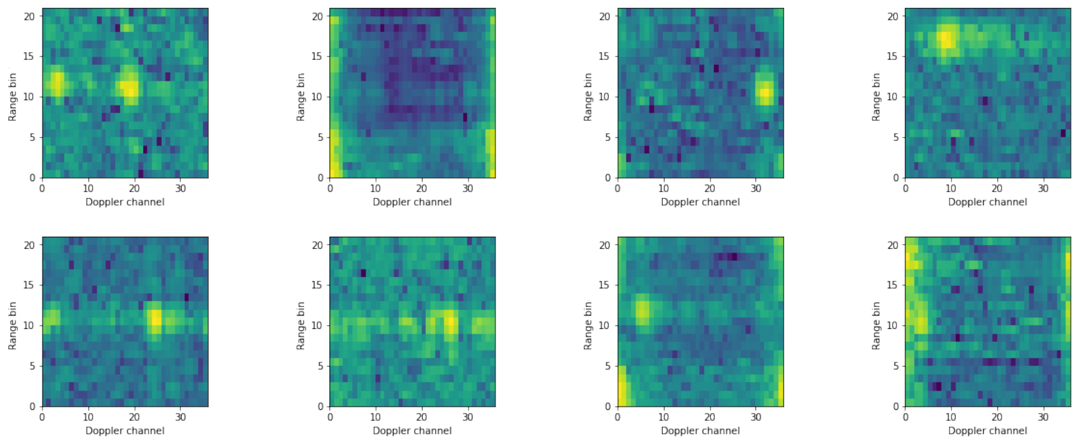
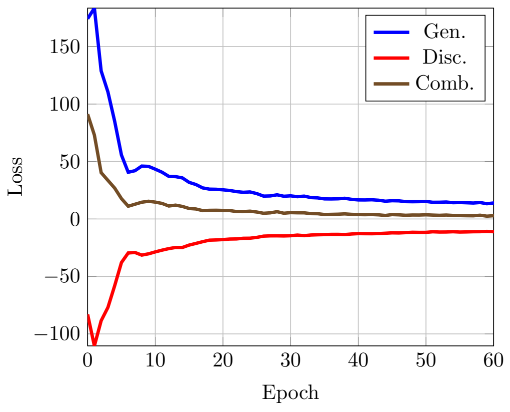
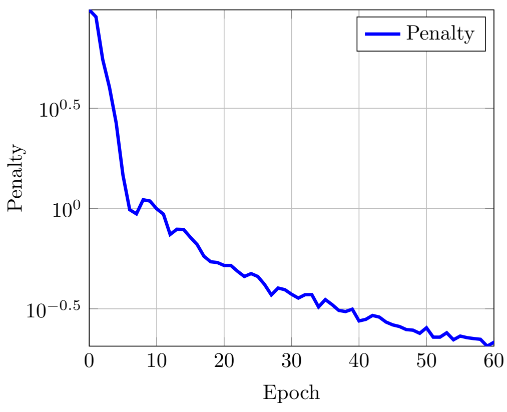


Figure 4.9: A selection of range-Doppler plots, displaying amplitude in frequency domain, from generated data. Network trained on Fourier transformed data with logarithmic amplitude as input. Clear objects visible in almost all plots. Amplitudes redacted.

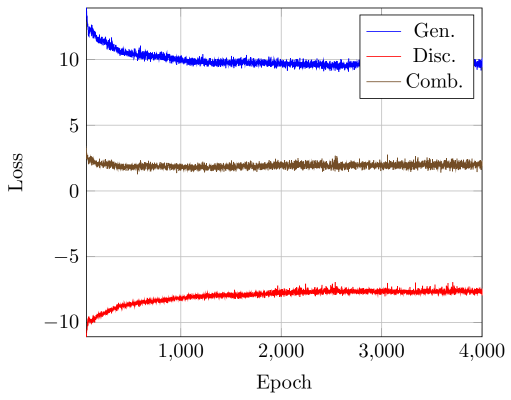


(a) Loss of networks separately and combined.

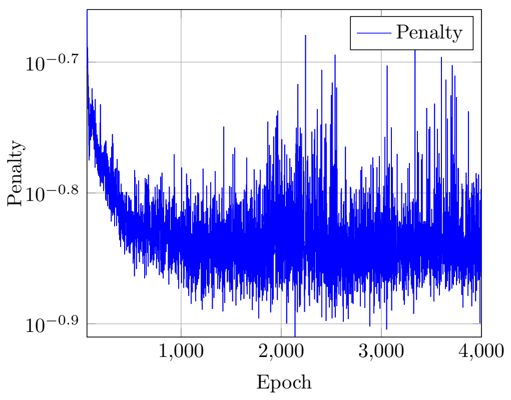


(b) Penalty in discriminator training.

Figure 4.10: Metrics of a network trained on, and generating, Fourier transformed data on polar form. Only first 60 epochs for readability.



(a) Loss of networks separately and combined.



(b) Penalty in discriminator training.

Figure 4.11: Metrics of a network trained on, and generating, Fourier transformed data on polar form. First 60 training epochs omitted for readability.

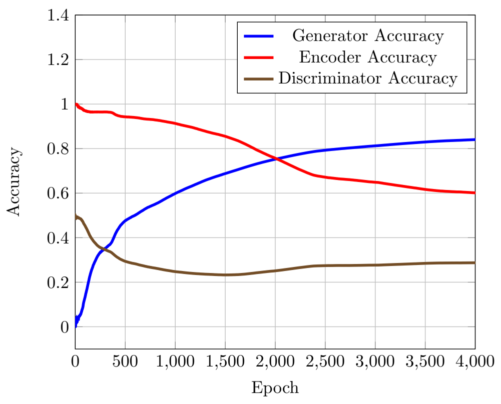


Figure 4.12: Accuracy of a network trained on, and generating, Fourier transformed data on polar form.

The results when using the version where the discriminator takes both time and frequency domain data as input show no major difference in the plots compared to the version that only uses frequency domain data. See Figure 4.13 for the frequency domain plot of the amplitude. Possibly the phase coherence is slightly improved, but only marginally, compare Figure 4.14 and Figure 4.15.

When discarding the phase in the training, seen in Figure 4.16, the objects appear to be clearer and more realistic. However, as the phase inherently is absent in this model, it was not possible to analyse this network further.

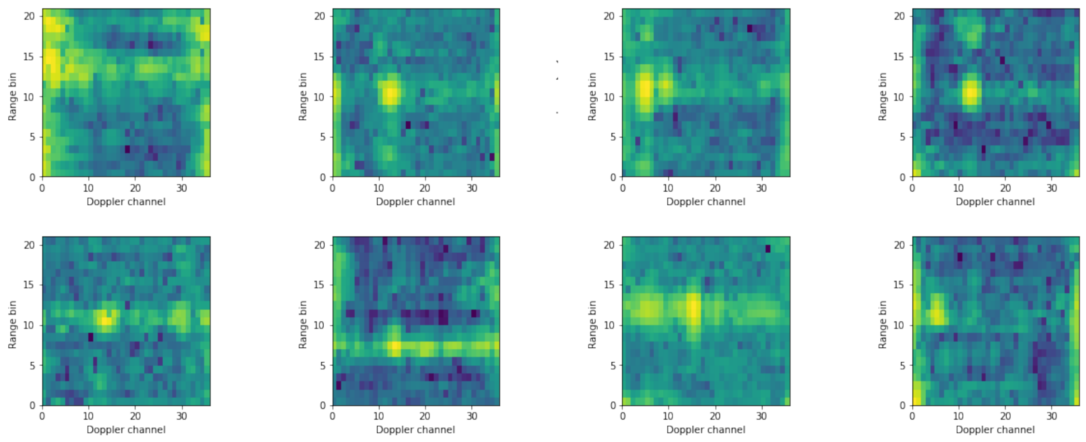


Figure 4.13: A selection of range-Doppler plots, displaying amplitude in frequency domain, from generated data. Network trained on time domain data, as well as Fourier transformed data with logarithmic amplitude as input. Objects clearly visible in almost all plots. Amplitudes redacted.

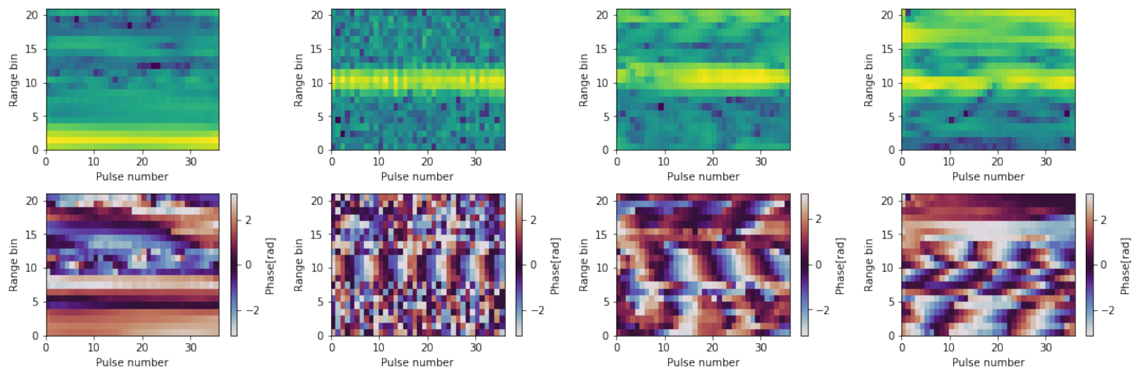


Figure 4.14: A selection of range-pulse plots, amplitude and phase in time domain, from generated data. Network trained on Fourier transformed data, with logarithmic amplitude as input. Mostly visible coherence in phase at objects. Amplitudes redacted.

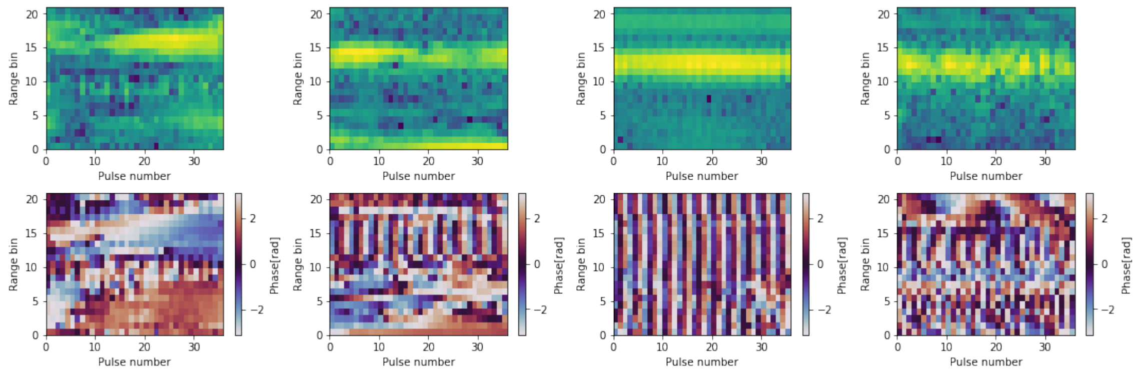


Figure 4.15: A selection of range-pulse plots, amplitude and phase in time domain, from generated data. Network trained on time domain data as well as Fourier transformed data, with logarithmic amplitude as input. Mostly visible coherence in phase at objects. Amplitudes redacted.

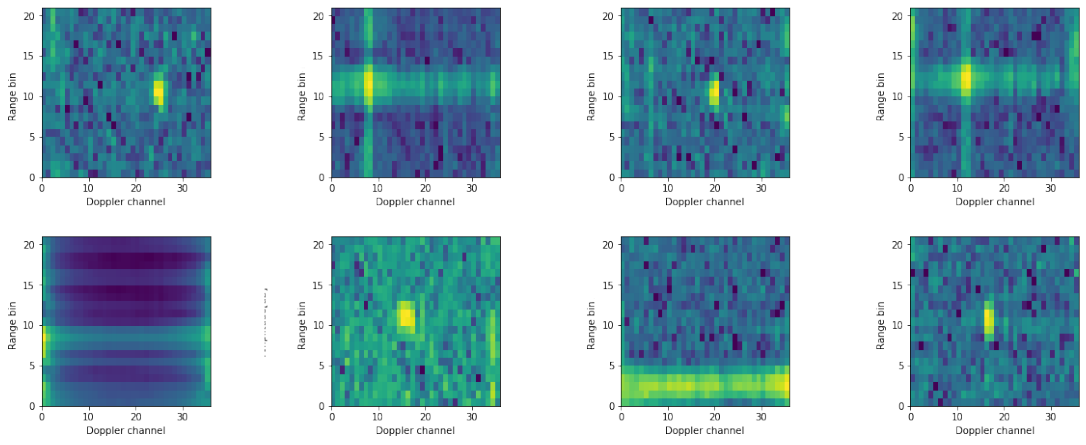


Figure 4.16: A selection of range-Doppler plots, displaying amplitude in frequency domain, from generated data. Network trained on, and generating, Fourier transformed data with logarithmic amplitude without phase. Objects very clearly visible in almost all plots. Amplitudes redacted.

4.2.2 Qualitative evaluation results

In the end, it seems the CWALI model was able to generate decent-looking radar images, at least by visual inspection. The frequency domain plots often show a clear object with a reasonable amplitude. Shuffling these examples, a human would have difficulty distinguishing real from fake. An uncurated collection of frequency domain amplitude plots of both real and generated samples can be found in Appendix B. However it seems that most classes look very similar, which is most likely due to the k-means filtering, which extracted samples that had clear objects in them. The difference in the data set without k-means filtering can also be seen in Appendix B.

4.2.3 Quantitative evaluation results

Although good quantitative metrics are hard to find, some examples are given in the following sections.

4.2.3.1 Maximum mean discrepancy

Sampling 500 real examples of every class and generating an equal amount, an estimate of the maximum mean discrepancy can be calculated and seen in Table 4.3. This maximum mean discrepancy estimate has used the rational quadratic kernel with $\alpha = 0.5$ as described in Section 3.6.

Data set	Jet	Propeller	Helicopter	Drone	Bird
k-means	$4.6 \cdot 10^{-4}$	$3.9 \cdot 10^{-5}$	$1.0 \cdot 10^{-5}$	$2.3 \cdot 10^{-4}$	$2.4 \cdot 10^{-4}$
Full data	$9.8 \cdot 10^{-4}$	$6.8 \cdot 10^{-5}$	$6.3 \cdot 10^{-5}$	$2.9 \cdot 10^{-4}$	$9.5 \cdot 10^{-5}$

Table 4.3: Maximum mean discrepancy between the data set and generated samples of each class.

As the table shows, our model manages to generate samples with very low MMD for every class compared with the training data, indeed if one takes the MMD between the data sets and completely random data or between classes in the training data, the results are a couple of orders of magnitude larger. However, comparing classes in the generated data gives similar values as in Table 4.3. Calculating the MMD between the given data set and a set of random complex matrices with elements of similar magnitude consistently gave an MMD of around 2. This gives some quantitative measure of how well the model manages to generate samples. One can also see the difference the use of k-means clustering makes; for almost all classes the generated and training data showed more similarities.

4.2.3.2 Classification network

Using the neural net trained by another group of thesis workers, as described in Section 3.6, by generating 1000 images of jets and 1000 images of helicopters, the results in Table 4.4 were obtained.

Class	**Correctly classified**
Jet	82.7%
Helicopter	15.7%

Table 4.4: Correct classifications by neural net.

Keep in mind that “correct” here means that the neural net classified the samples as what we told our model to generate. As the table shows, CWALI manages pretty well with generating jets according to the neural net. However the generation of helicopters seems to be a bit lacking.

5 Discussion

In this thesis we developed a novel hybrid generative adversarial network dubbed CWALI, which has shown good success in generating processed radar data. When generating raw radar video data, which was the goal of the project, the results were less impressive. Still, it managed to generate somewhat realistic data, and this shows the potential of utilising GAN in this application. Here follows a discussion of the results, with suggestions for possible improvements, together with a conclusion of the thesis.

5.1 Architecture

The architectures of the networks used in the thesis, both in terms of input and output data, and underlying variant of the GAN concept can be discussed at length, with the foundation of the results shown in the previous chapter. Primarily the aspects of quality and feasibility of using the architectures in contexts radar applications are in focus in the following sections.

5.1.1 Input and output data

The results show that what type of data the network takes as input has large impact on the quality of the generation, and that the progression of preprocessing solutions was crucial. Taking an identical approach as to images was not possible; the results were very bad. But just increasing the size of the network gave clear improvements. Switching to polar form changed the output and the dynamics, but not necessarily the quality. Still this was a step in the right direction, and when training on Fourier transformed data, the quality clearly improved. Surprisingly, using more information in the form of training the discriminator on data in both time and frequency domain did not improve the quality. Intuitively this should help the network learn more patterns, but this was not the case. Possibly, the network needed to be larger for it to be able to make use of all this information. The image-like architecture only generating amplitude data predictably performed well, but was in principle out of this projects scope, which was to generate time domain raw radar data. As the amplitude data was in frequency domain, and there was no phase, it was not possible to convert to time domain data. It can be noted as an interesting observation for future applications, though.

To conclude the discussion regarding the data, an architecture using and generating data on polar form, with logarithmic amplitude, in frequency domain seem to be the best option for generating raw radar data, although combining the domains, or only generating phase show some potential.

5.1.2 GAN technique

Regarding the general GAN architecture used in this thesis, called CWALI, it showed relatively good potential in generating realistic radar data, but as it is not the most advanced implementation of a GAN, it is likely that it could be improved. Also, there were no theoretical indications as to why using ALI – the concept of using a third network to generate latent vectors from images – is the best choice (as the inference of latent space was not used in this work), but merely empirical indications of it performing slightly better. This technique could in principle be discarded, though we kept it in our model as it did show better results. That Wasserstein GANs have good performance is well known (as well as tested in the prestudy), and is therefore most certainly a good choice in this context.

As the aim of the thesis was to generate data from a description, some kind of conditional GAN was needed, but here the results show clear room for improvement; the network did not learn well enough to separate the different types of objects, shown both with MMD and the classifying network. Of course it might be so that MMD is not the perfect metric, and that the classification network is a neural network, with all that that entails. Still, it is an indication of that the conditionality of the network should be improved. Considering that the network in the prestudy managed to handle the different classes well, this should not be an impossible problem to solve, but as of now the generated radar data does not follow the conditions well enough. To have a conditional network should still be a better solution than to have separate networks for each class, at least if the aim is to add more labels in the future.For this limited application though, separate networks could be of interest. On the subject of MMD, it also shows that the generated data is rather similar to the training data, which confirms what could be seen visually.

There exist even more advanced GAN architectures that generate more impressive results when in the realm of images, and which could possibly do so with radar data as well. This would also be interesting to investigate further, but to fit a more complicated architecture as foundation for this thesis was not feasible, as it was not known what would be necessary to change to achieve good performance for the radar data.

As the aim was to prove the concept of generating radar data using GANs and not necessarily to do it perfectly, the chosen architecture was for this application sufficient. A more complicated architecture would most certainly have made it harder to implement the changes required for the move from images to radar data. Applying a more sophisticated base architecture could improve the quality though, and at least some improvement in the area of conditionality is needed to make radar data generating GANs a useful tool.

5.2 Convergence and hyper-parameters

Although the generated data is not at the same level as recorded data, the better example show that it is not impossible to somewhat realistic raw radar video data using GAN. The reason the result is not more satisfying could be the integral problem of convergence of GANs, i.e. the minimax game does not necessarily have a Nash equilibrium or global optimum. Even if it did, the training is unstable and heavily dependent on hyper-parameters, and the number of hyper-parameters is not small.

As the aim of this thesis was not to find the optimal radar data generating method, but to prove the concept of generating radar data using GAN (which in itself is a challenge), the focus was never to investigate every possible combination of parameters and networks. Some different variants of hyper-parameters, such as learning rate, width of networks were tested, and what is described in this thesis is a decent baseline of those variants, but having a wider network might for example give better result, at the expense of training time, and when testing many different architectures, training time is a limiting factor. Also, tweaking hyper-parameters on a complex and stochastic system like a GAN requires tests of many different combination of parameters, as well as averaging of the results, which is out of this project's scope. Rather, working solutions from other contexts (mainly images) were used as heavy inspiration when developing the networks for complex-valued radar data, but what is a good and stable setup, with convergent training for images might not be the same setup in the application of this thesis; as GANs are as sensitive as they are, it is even quite likely. For example, the selection of Adam as optimiser might give good results for images, but possibly not for radar data. Still, a not too small portion of the generated data shows good potential. It is therefore reasonable to believe that the concept is good, although the perfect network or combination of parameters has not been found. This could very well be investigated further.

5.3 The phase problem

The phase contains a lot of information, although it is not necessarily easy to analyse, and is usually discarded halfway through the signal processing of a radar. An object is clearly marked (especially in frequency domain) in the amplitude plot, but not in the phase plot. It seems like it is easier to find and generate the relevant patterns in the amplitude than in the phase, which also the results in Section 4.2.1 suggest. When only training on and generating amplitudes, at least through visual inspection, the results are noticeably improved. This is not surprising, as a plot with only amplitudes is very similar in structure to an image with only one channel, which a GAN should be able to handle well, and even the network used in the prestudy had good results when generating images (with three channels). This suggests the network had difficulty learning the amplitude/phase relationship, but as it clearly manages to generate a phase with coherent patterns at the generated object when converting to time domain, it has some success.

Not being able to generate phase well can be a serious issue if the aim is to generate realistic data. How much information that might be lost in a badly generated phase has not been investigated in this thesis, but that is somewhat beside the point; if the phase does not match what is seen in reality, the generated data is not realistic.

This leads to how one might try to solve this issue, and there are a couple of possible solutions. First of all, the solution to the problem of complex numbers and neural networks has in this thesis been to use real numbered networks and hope that they will learn the relations between the channels. Developing a true complex-valued network could be a solution to better preserve the dynamics of the data, but as there exist no libraries etc. for complex numbers in common machine learning softwares, this has to be created separately, which is time consuming. Another possible solution might be to simply only generate the amplitude with a GAN, and with some other technique generate the phase from the amplitude. As such techniques of this kind exist, it could be possible to develop a solution for this context as well.

5.4 Lack of evaluation metrics

As most evaluation metrics depend on pre-trained models trained on different data sets, none of these are entirely applicable to our models. Using a model trained on pictures of objects to evaluate whether our GAN can generate radar data will not provide a relevant metric. There is the option of transfer learning of course, and modifying the input and output layers of the inception model before retraining it to classify radar data, but this is also problematic for the same reasons why the Inception Score and Fréchet Inception Distance aren't applicable, as radar video data is quite different from the images contained in the ImageNet database and other similar machine learning training data sets.

The classic method of evaluating neural nets, to keep held out validation and test data sets also does not map cleanly onto GANs, as the generator implicitly learns from the data set supplied at training, and keeping some data samples away from the training introduces the risk of the generator potentially not learning to generate certain modes of the data distribution.

For these reasons, the only meaningful evaluation metrics were to classify generated images using classification nets trained on the same data, created by another group of thesis students at SAAB. The results of these evaluations gives the best possible insight other than visual inspection into whether or not the generated data seems realistic or not. Though this method of evaluation is also questionable as it gives no indication of whether or not the generator has got stuck in mode collapse, meaning one still has to resort to visual inspection of the generated data to verify that it is diverse. In our case, this is indeed the case and the generator seems to not be stuck in mode collapse. In short, there is a great need within the field of GANs for generalised and robust evaluation metrics, which at the time of writing is lacking.

The fact that the best evaluation metric for GANs in general is visual inspection is a serious concern when dealing with radar data. It is easy to see if an image is realistic or not, but it is not as simple with raw radar video. If generated data is to be used a certain application, it has to be possible to see whether the data actually is what is expected, and that is a non-trivial problem when it is not possible for a human to look at it and make a decision. The same problem occurs when expanding data sets, as the number of samples is likely to reach a size so large that humans can not verify the quality. This could make it impossible to use GANs as a practical tool.

5.5 Conclusion

Generative adversarial networks is a powerful generative technique, but it suffers from great training instability and lacks rigorous evaluation metrics. As such, it is difficult to find proper parameters which will generate any given type of data. This thesis show that it is possible to generate raw, realistic radar video data, but that the quality is not high enough for it to be usable in a product as of now. For generating images on the other hand the technique is more mature, and could be used to e.g. expand data sets of already processed radar data. If the aim is to expand a data set, and specifically to generate more samples of classes with very few samples, it is crucial that the quality of the generated data is high, and that the generated samples of a certain class matches real data. This is rarely the case with GANs in general, and certainly not in this radar application. Also, as long as there is no efficient way to evaluate whether the generated data is what was expected, of the quality expected, it is difficult to find a use case for the technique, despite its potential.

5.6 Future work

If GANs are to be used in radar-based applications, generating the fully complex-valued raw data is not the best approach. However, if one would instead generate only the amplitudes of the raw data and from this reconstruct a coherent phase, the data might actually start looking realistic. We did not implement this sort of phase retrieval due to time constraints. Also, the conditionality of the data generation should be improved, or at least analysed further, for this technique to be a useful tool. Even further investigation in finding good hyper-parameters could improve the performance significantly in such unstable architectures as GANs.

Another option is to instead of generating raw video, to generate at least somewhat signal-processed video. This would reduce the parameter space and might be easier and more stable to implement than trying to generate the raw video.

Our GAN only generates short-time, short-distance cutouts from a single lobe of the radar data. Expanding to generating the full height of the detection volume would be a natural extension to this work, such that the GAN would generate a 3D detection volume.

Bibliography

[1] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[2] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[3] Yann LeCunn. *The MNIST database of handwritten digits*. http://yann.lecun.com/exdb/mnist/. Accessed: 2020-03-18. 1998.

[4] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2015. arXiv: 1511.06434 [cs.LG].

[5] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. *Adversarial Feature Learning*. 2016. arXiv: 1605.09782 [cs.LG].

[6] Vincent Dumoulin et al. *Adversarially Learned Inference*. 2016. arXiv: 1606.00704 [stat.ML].

[7] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].

[8] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[9] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[10] Nitzan Guberman. *On Complex Valued Convolutional Neural Networks*. 2016. arXiv: 1602.09046 [cs.NE].

[11] Qigong Sun et al. *Semi-supervised Complex-valued GAN for Polarimetric SAR Image Classification*. 2019. arXiv: 1906.03605 [eess.IV].

[12] Bernhard Mehlig. *Lecture Notes – Artificial Neural Networks*. Accessed: 2020-01-21. 2019. URL: http://physics.gu.se/~frtbm/joomla/media/mydocs/ann.pdf.

[13] Farzan Farnia and Asuman Ozdaglar. *GANs May Have No Nash Equilibria*. 2020. arXiv: 2002.09124 [cs.LG].

[14] Y. Shoham and K. Leyton-Brown. *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2008. ISBN: 9781139475242.

[15] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2016. arXiv: 2016.00160 [cs.LG].

[16] Ali Borji. *Pros and Cons of GAN Evaluation Measures*. 2018. arXiv: 1802.03446 [cs.CV].

[17] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium.* 2017. arXiv: `1706.08500 [cs.LG]`.

[18] Shane Barratt and Rishi Sharma. *A Note on the Inception Score.* 2018. arXiv: `1801.01973 [stat.ML]`.

[19] George W. Stimson. *Introduction to airborne radar.* 2nd ed. SciTech, 1998.

[20] Andrés Marafioti et al. *Adversarial Generation of Time-Frequency Features with application in audio synthesis.* 2019. arXiv: `1902.04072 [cs.SD]`.

[21] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/`.

[22] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. arXiv: `1412.6980 [cs.LG]`.

[23] Arthur Gretton et al. "A Kernel Two-Sample Test". In: *Journal of Machine Learning Research* 13.25 (2012), pp. 723–773. URL: `http://jmlr.org/papers/v13/gretton12a.html`.

A

CWALI architecture

This appendix contains the layer structure of the three adversarial nets of polar form CWALI. The tables generally denote the shape of the output that is fed into the next layer using a tuple of numbers, representing the size of each dimension of the output. Output dimension inbetween layers is not fixed and varies depending on layer type, and while the tables show every layer, the progression is not strictly linear from top to bottom as the output channels of some layers can be split and processed separately. See Section 3.3.2 for figures.

A.1 Discriminator

Table A.1 shows the structure of the CWALI discriminator. The slope value of the leaky ReLU activation function is in all layers set to 0.2, and the kernel and stride size where applicable is equal, as this gave the best results.

Layer	Type	Activation	Kernel size	Strides	Output shape
1	Input (Image)	–	–	–	(36,21,2)
2	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(18,11,64)
3	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(9,6,128)
4	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(5,3,256)
5	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(3,2,512)
6	Flatten	–	–	–	(3072,)
7	Input (Latent)	–	–	–	(100,)
8	Input (Class)	–	–	–	(5,)
9	Concatenation	–	–	–	(3177,)
10	Dense	Linear	–	–	(1,)

Table A.1: Layer-by-layer description of the CWALI discriminator.

Layers 6-8 are independent of each other, and get concatenated by layer 9 before being fed into layer 10. To supplement the table, a diagram of the architecture can also be found in Figure 3.2

A.2 Decoder

Table A.2 shows the structure of the CWALI decoder. See Figure 3.1 for a diagram.

Layer	Type	Activation	Kernel size	Strides	Output shape
1	Input (Latent)	–	–	–	(100,)
2	Input (Class)	–	–	–	(5,)
3	Concatenation	–	–	–	(105,)
4	Dense	ReLU	–	–	(2304,)
5	Reshape	–	–	–	(3,3,256)
6	2D Upsampling	–	–	(3,2)	(9,6,256)
7	2D Convolution	ReLU	(5,5)	(1,1)	(9,6,128)
8	Batch Norm.	–	–	–	(9,6,128)
9	2D Upsampling	–	–	(2,2)	(18,12,128)
10	2D Convolution	ReLU	(5,5)	(1,1)	(18,12,64)
11	Batch Norm.	–	–	–	(18,12,64)
12	2D Upsampling	–	–	(2,2)	(36,24,64)
13	2D Convolution	None	(5,5)	(1,1)	(36,24,2)
14	2D Cropping	–	–	–	(36,21,2)
15	Activation	ReLU	–	–	(36,21)
16	Activation	$\pi \times$Tanh	–	–	(36,21)
17	Stack	–	–	–	(36,21,2)

Table A.2: Layer-by-layer description of the CWALI decoder.

A.3 Encoder

Finally, the encoder, which encodes samples to latent space. It is very similar to the discriminator, save for the inputs and an activation function. Table A.3 and Figure 3.3 describe this final net.

Layer	Type	Activation	Kernel size	Strides	Output shape
1	Input (Image)	–	–	–	(36,21,2)
2	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(18,11,64)
3	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(9,6,128)
4	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(5,3,256)
5	2D Convolution	Leaky ReLU	(5,5)	(2,2)	(3,2,512)
6	Flatten	–	–	–	(3072,)
7	Input (Class)	–	–	–	(5,)
8	Concatenation	–	–	–	(3077,)
9	Dense	Tanh	–	–	(100,)

Table A.3: Layer-by-layer description of the CWALI encoder.

B

Data comparisons

This section contains amplitude plots of FFT-processed data.

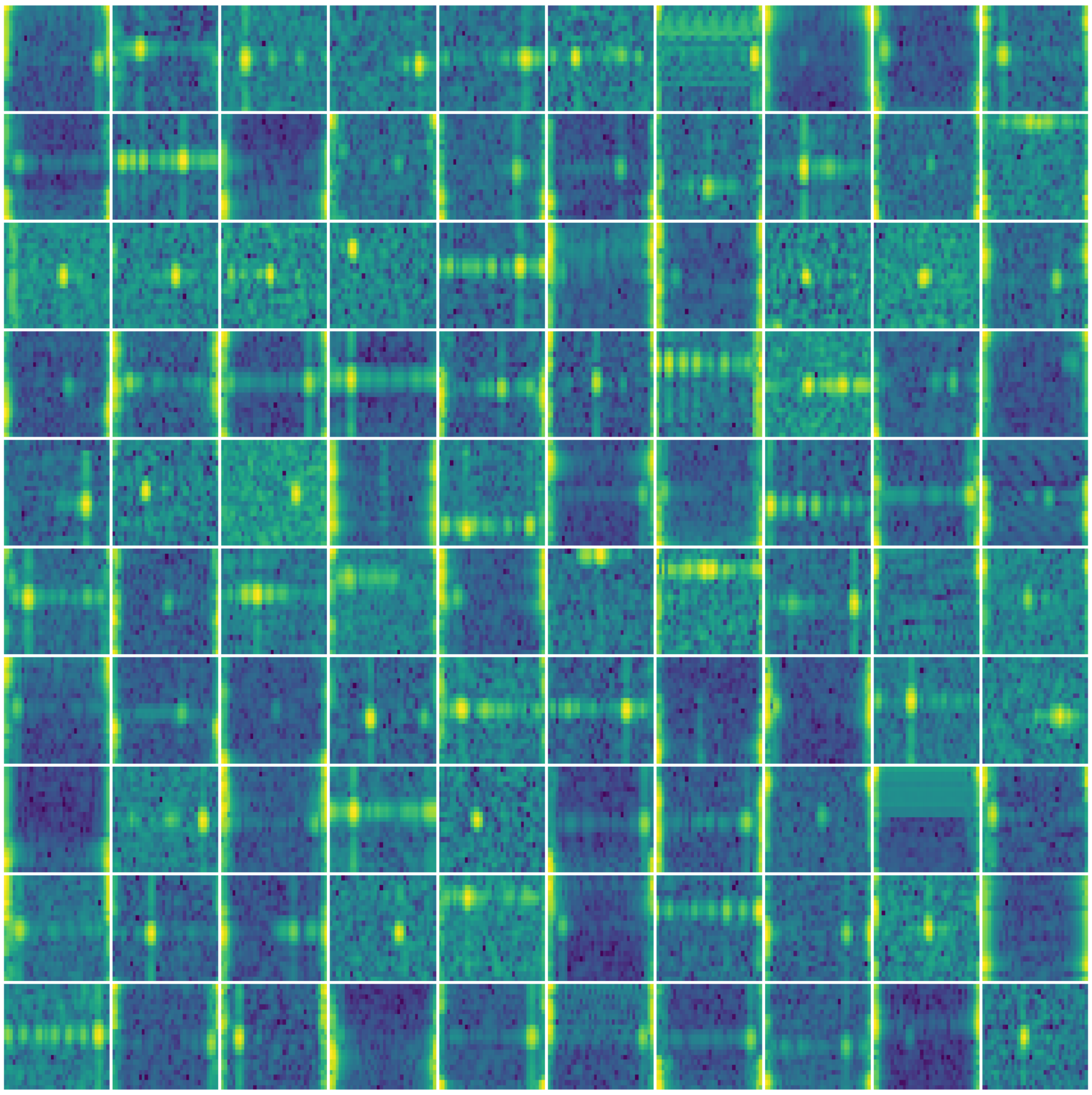


Figure B.1: Example of 100 plots of jets from the training dataset.

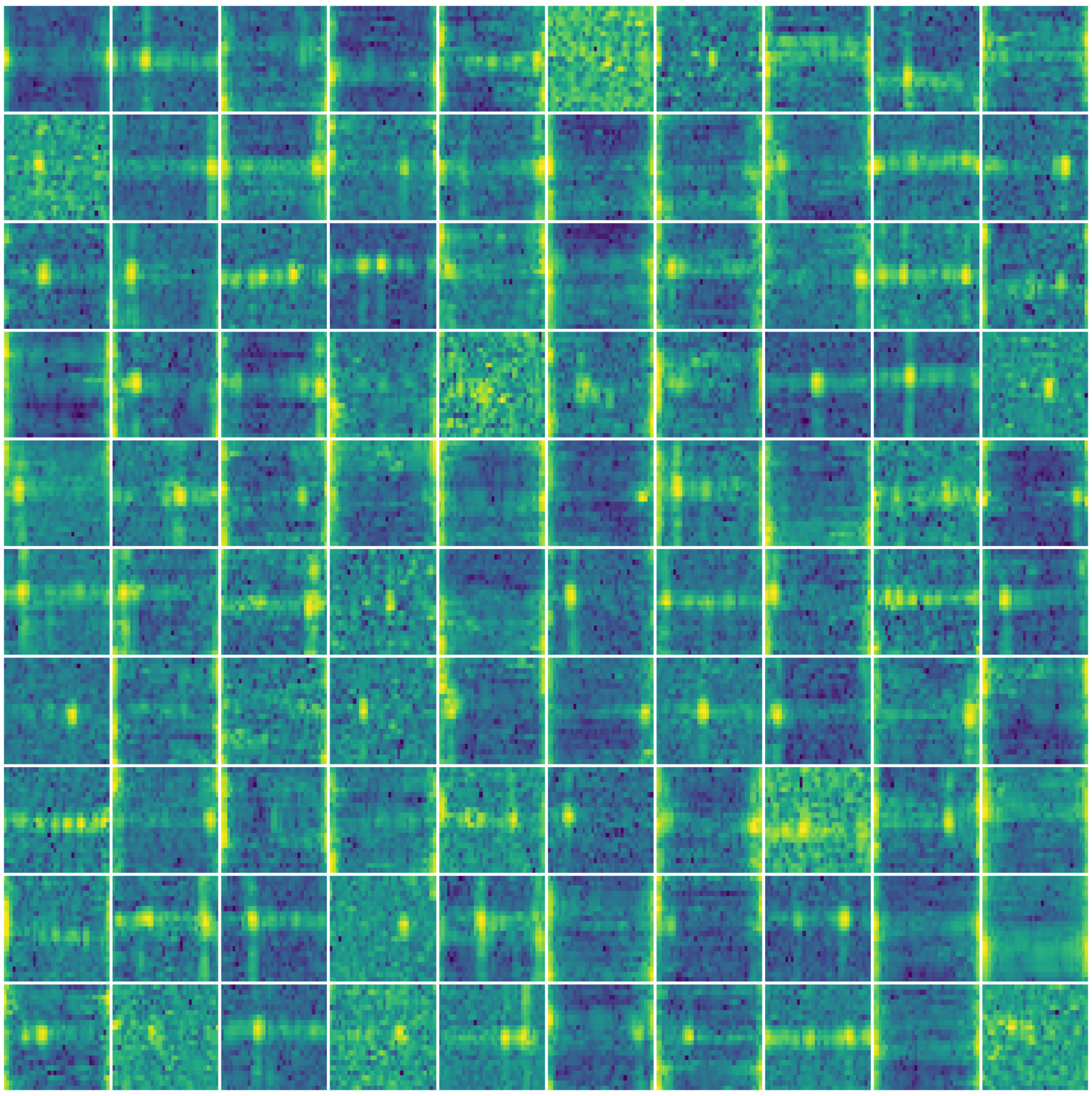


Figure B.2: Example of 100 plots of generated jets.

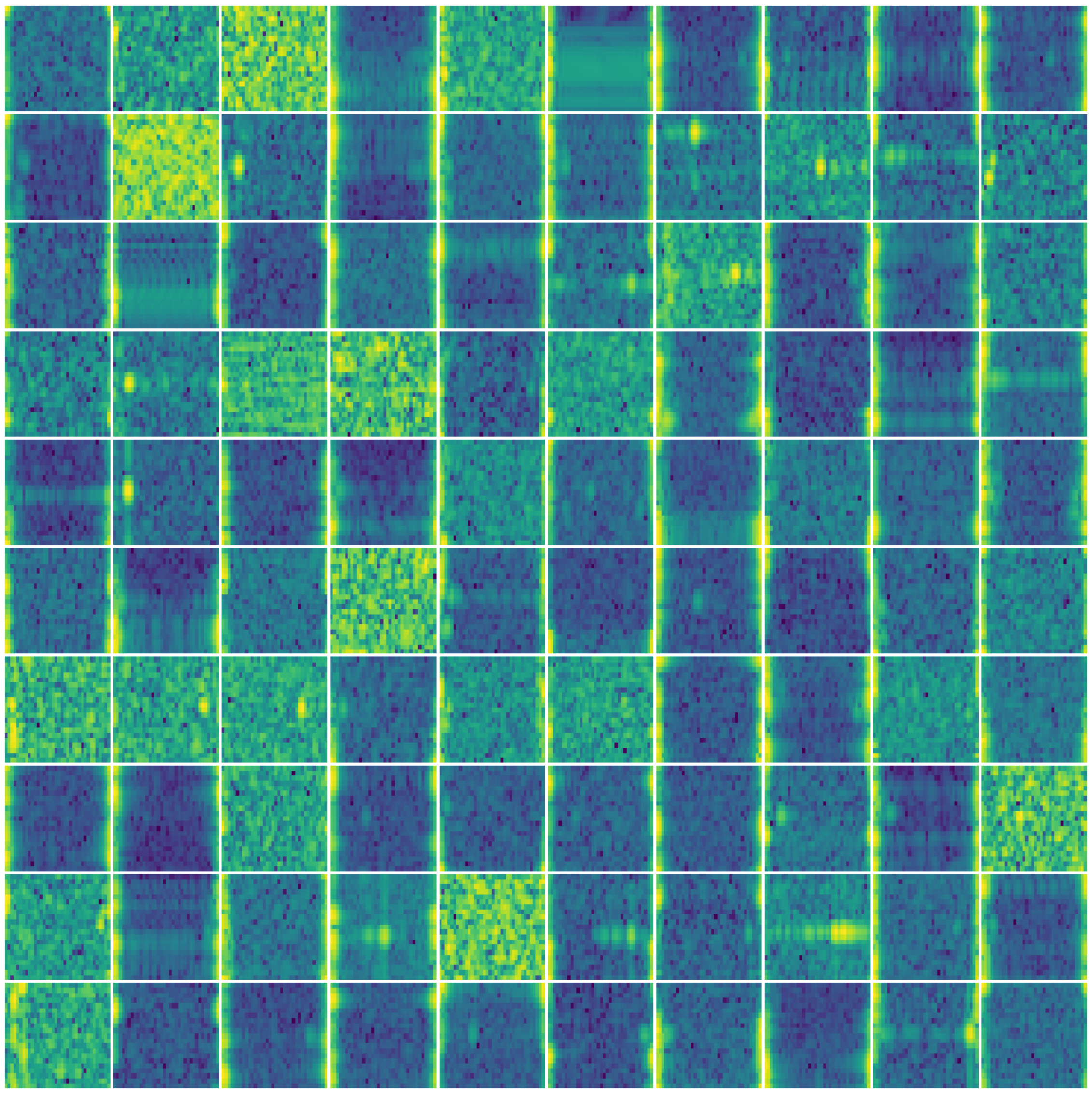


Figure B.3: Example of 100 plots of jets from the training dataset before *k*-means filtering.

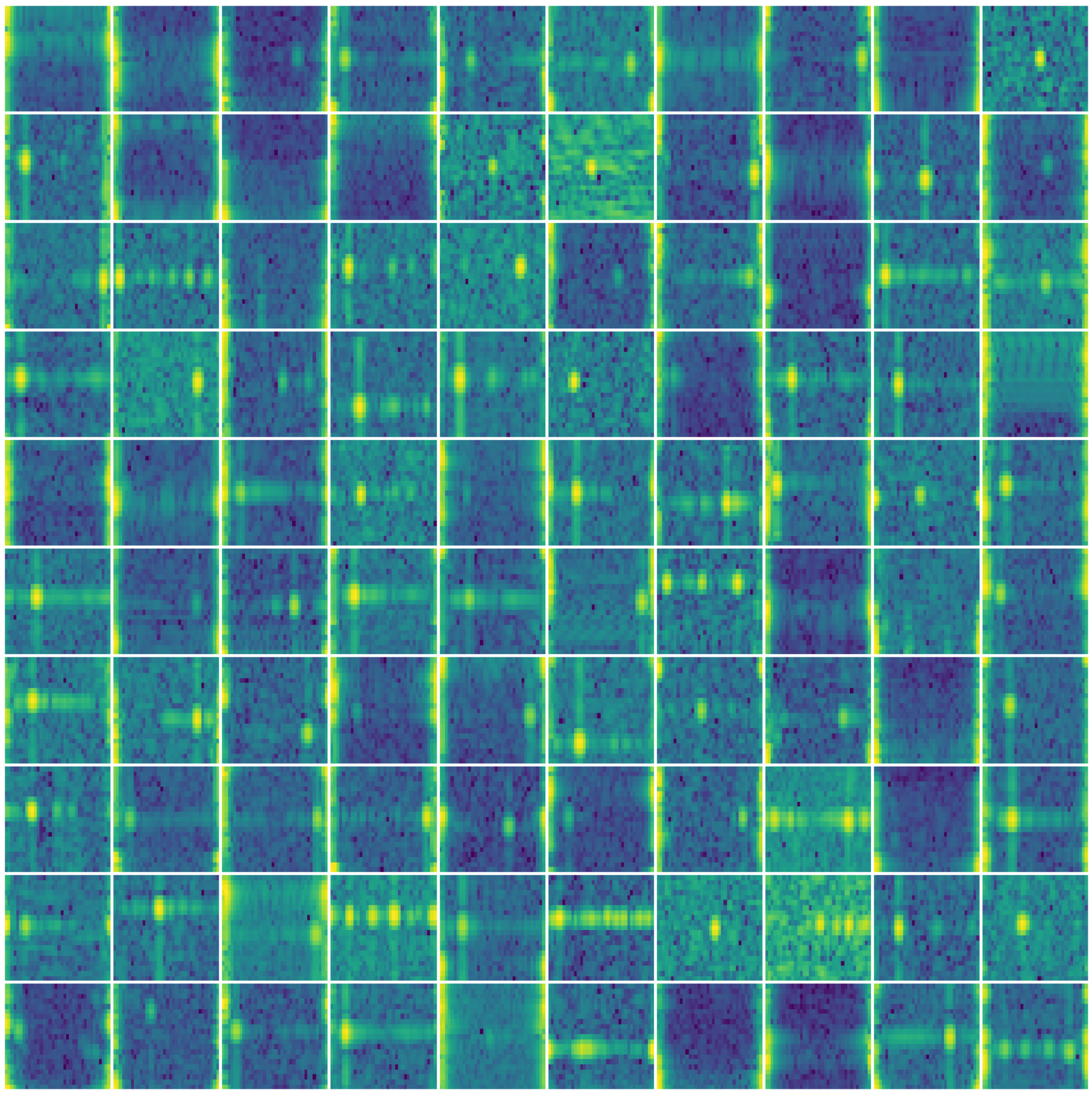


Figure B.4: Example of 100 plots of propeller planes from the training dataset.

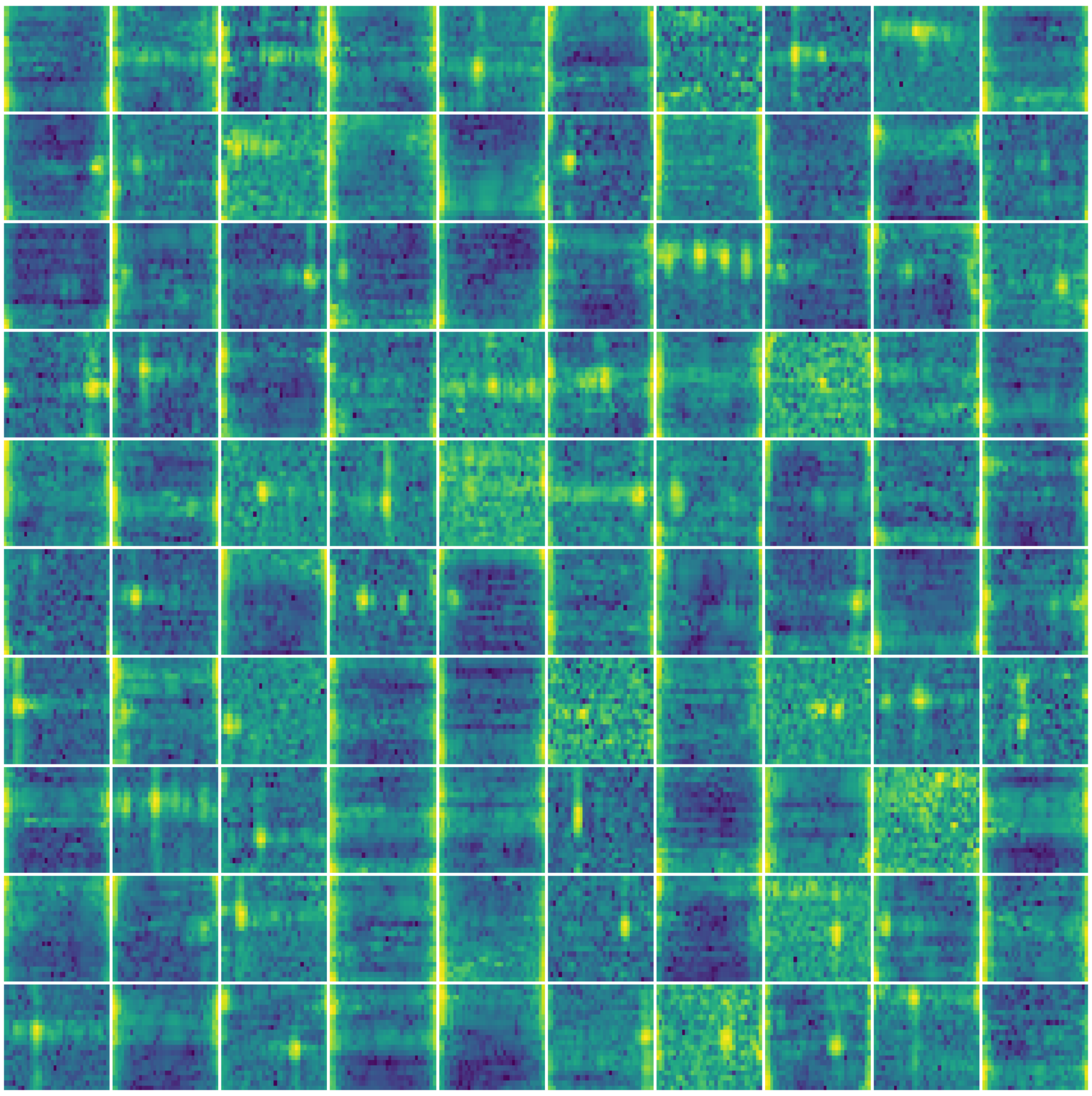


Figure B.5: Example of 100 plots of generated propeller planes.

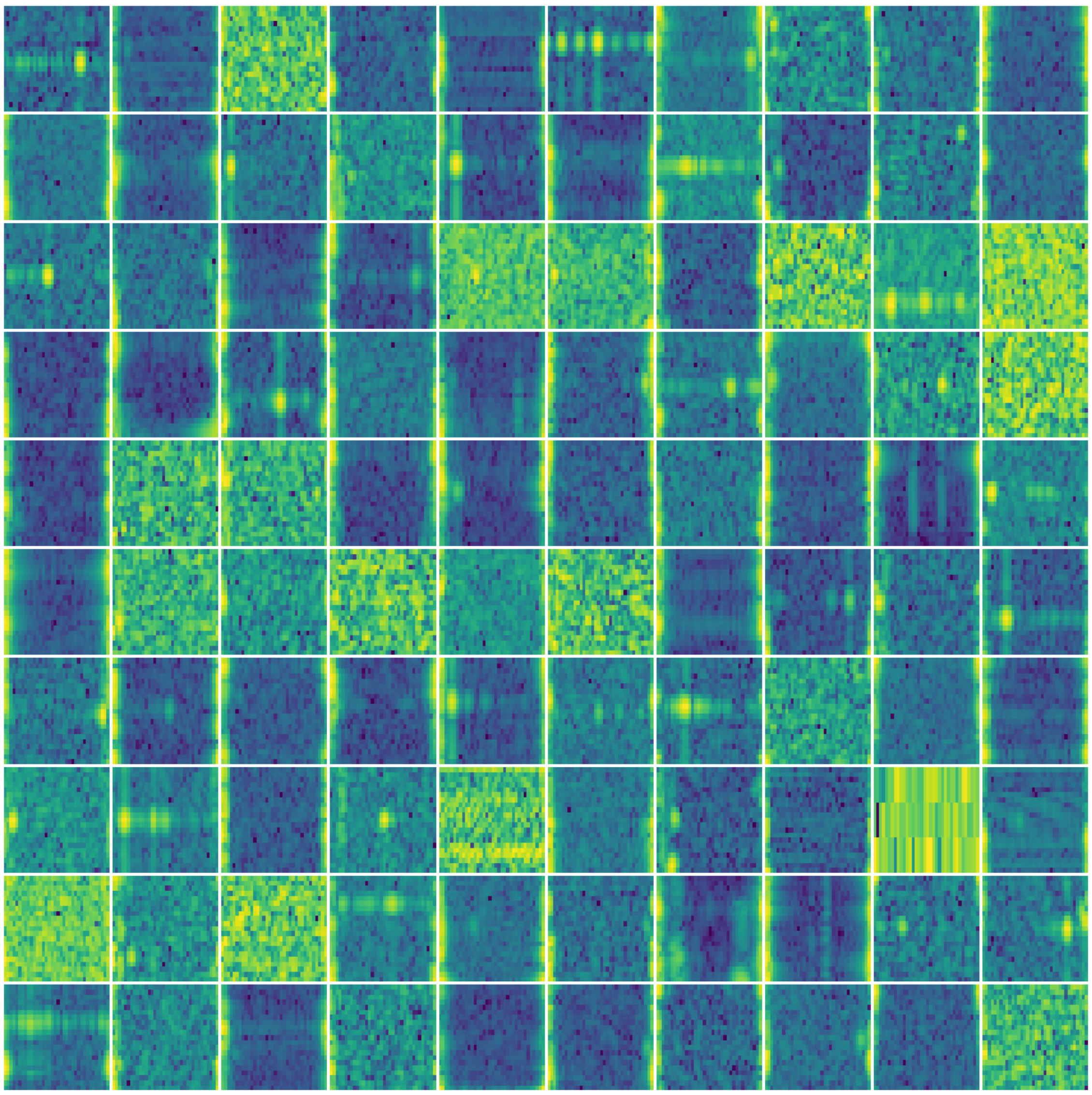


Figure B.6: Example of 100 plots of propeller planes from the training dataset before *k*-means filtering.

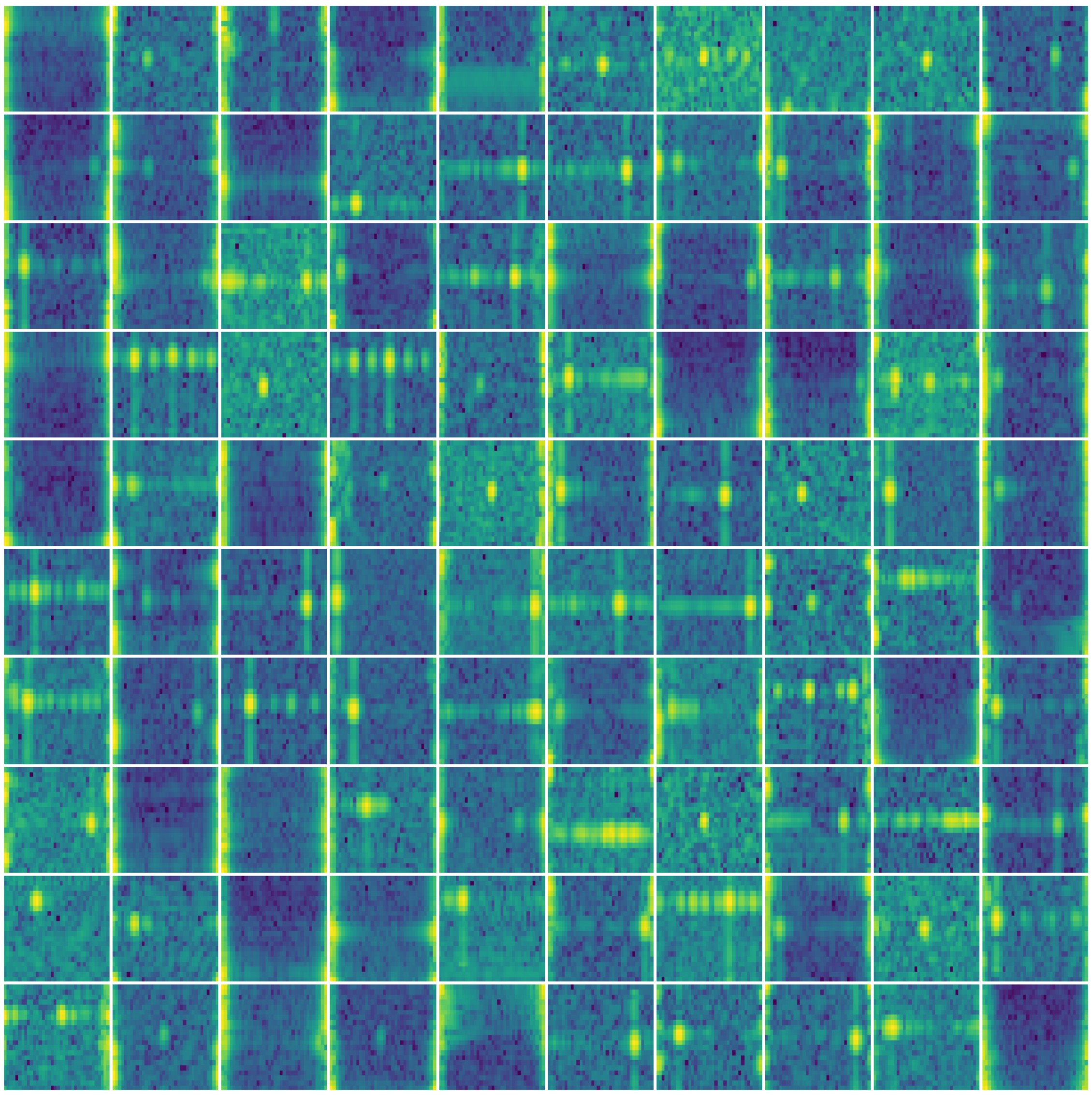


Figure B.7: Example of 100 plots of helicopters from the training dataset.

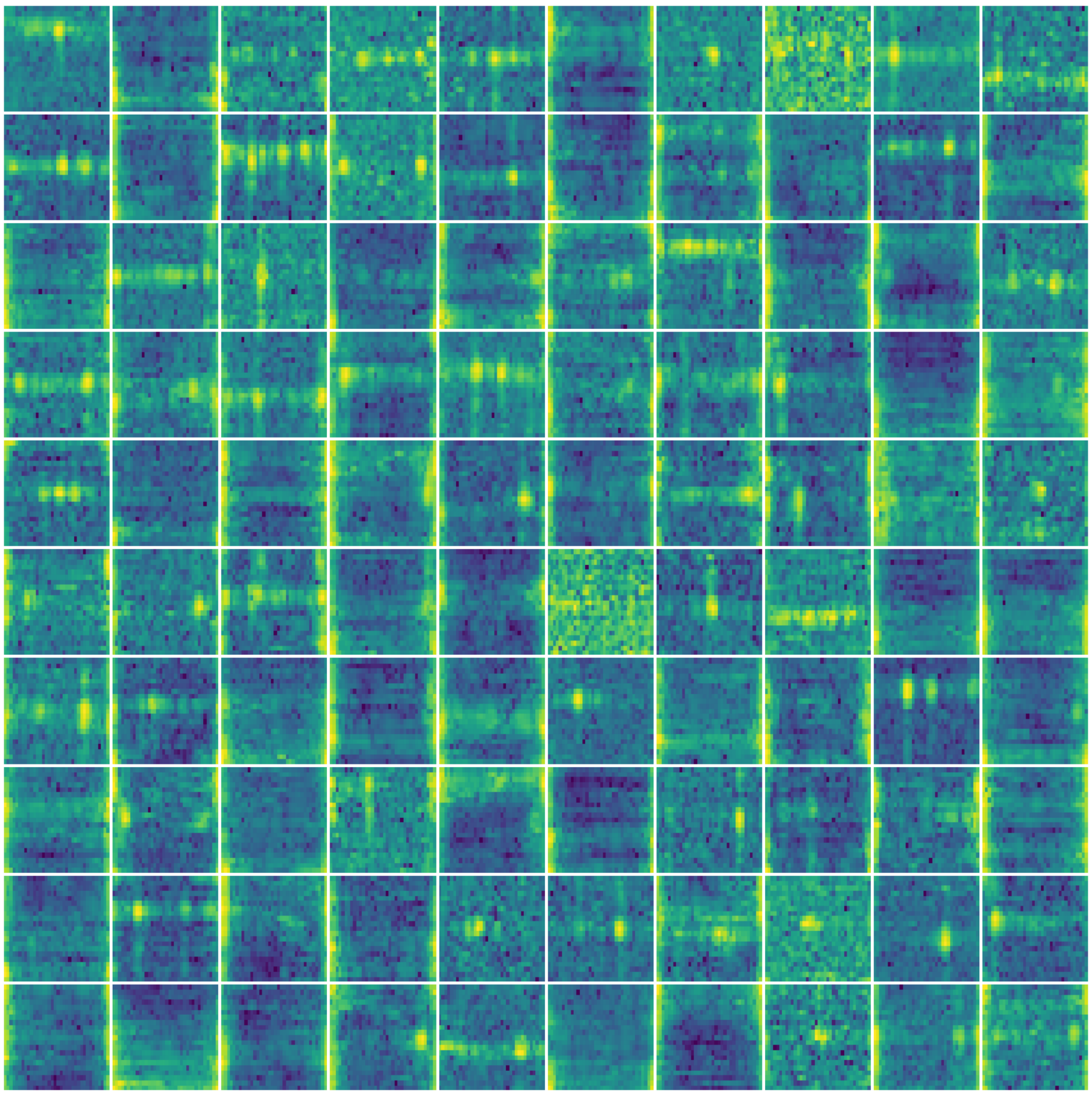


Figure B.8: Example of 100 plots of generated helicopters.

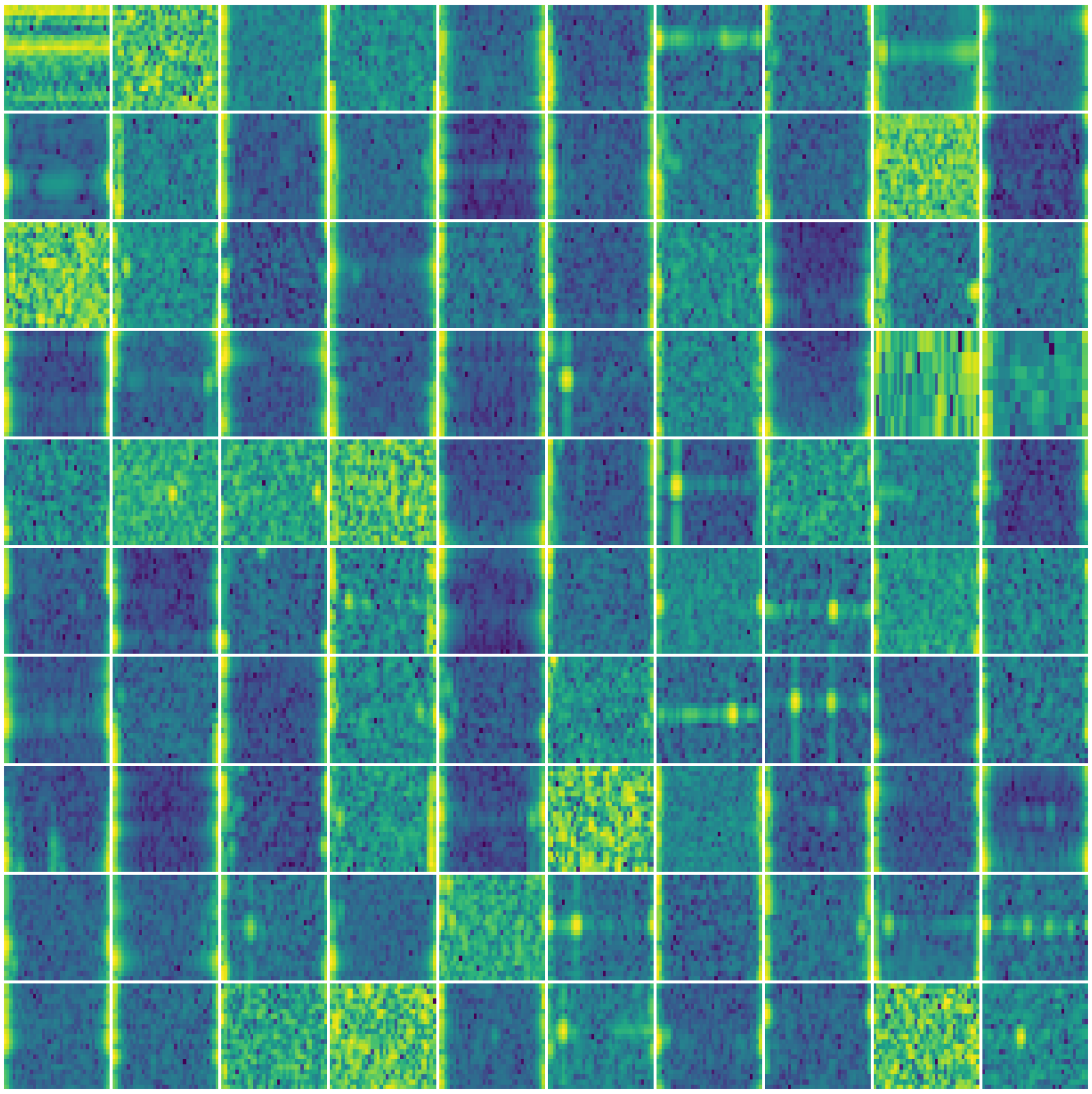


Figure B.9: Example of 100 plots of helicopters from the training dataset before k-means filtering.

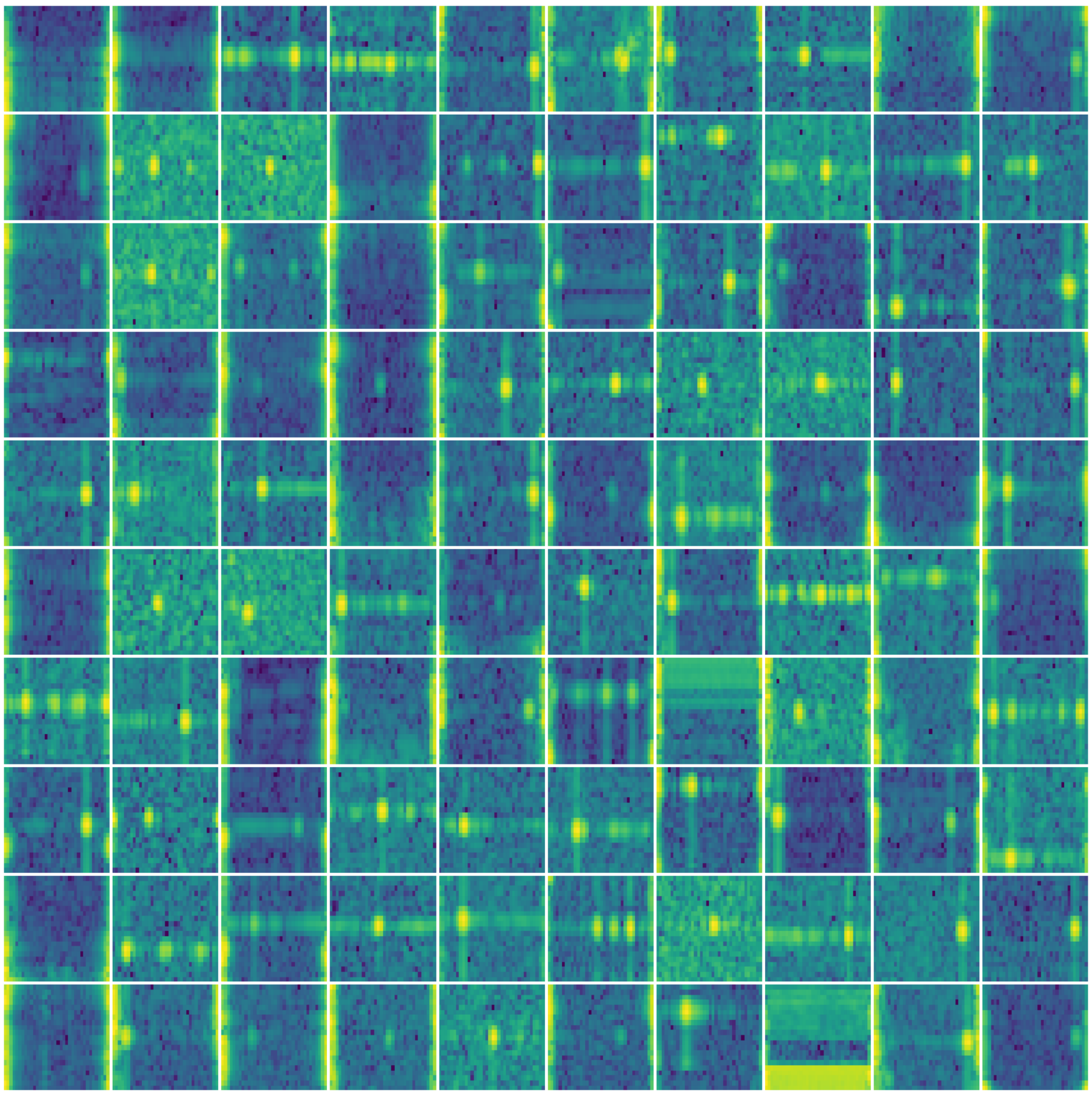


Figure B.10: Example of 100 plots of drones from the training dataset.

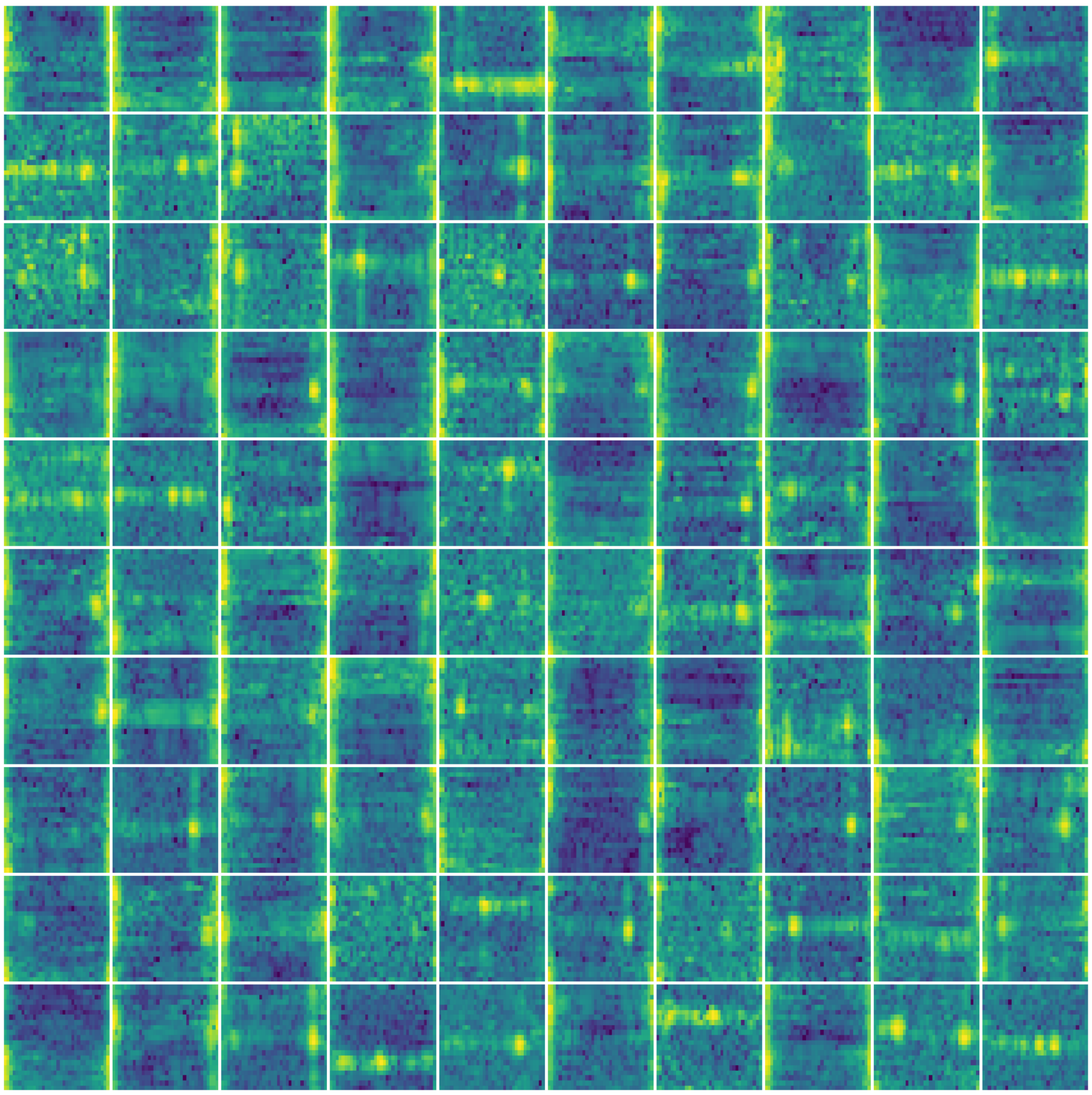


Figure B.11: Example of 100 plots of generated drones.

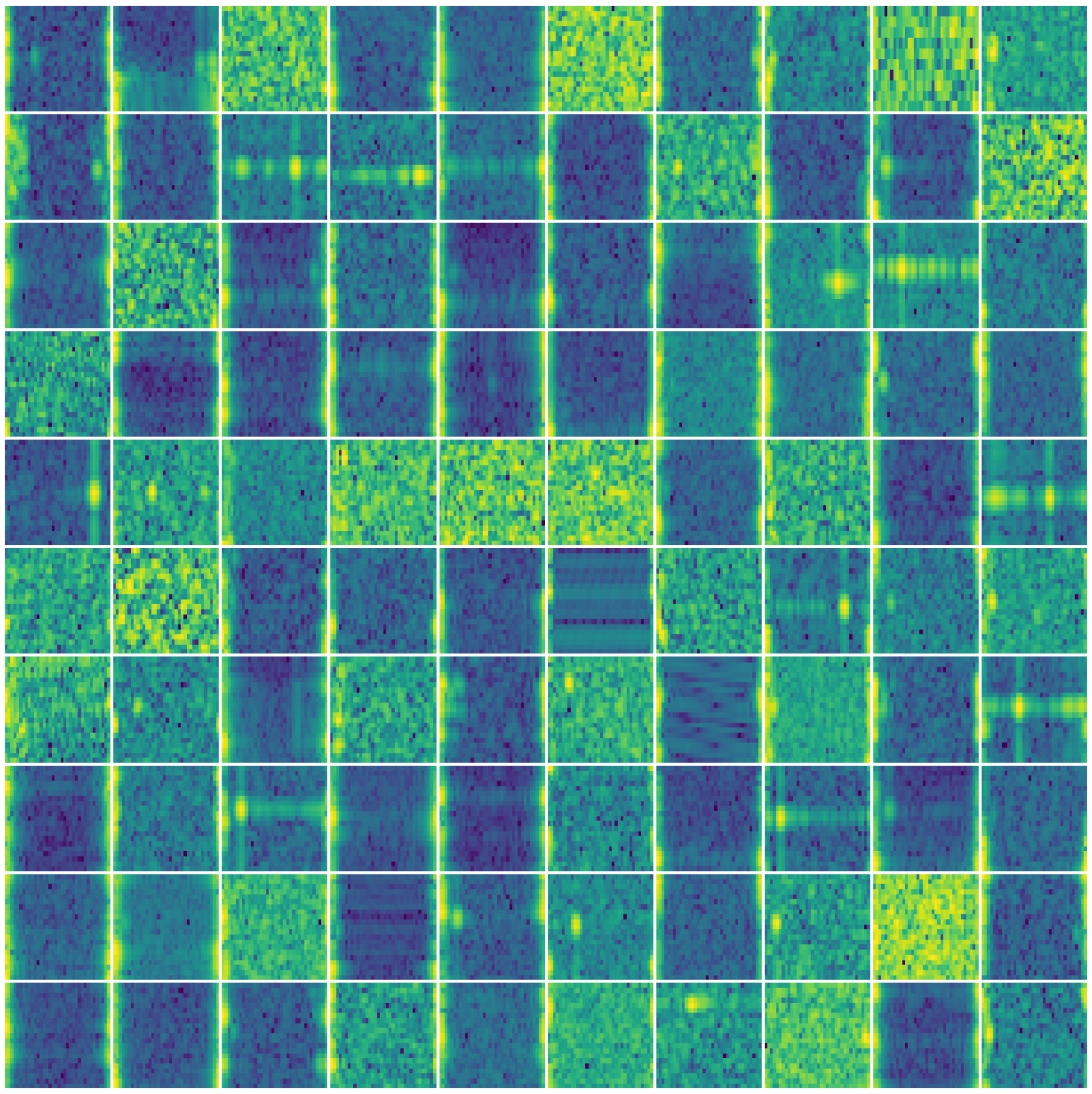


Figure B.12: Example of 100 plots of drones from the training dataset before k-means filtering.

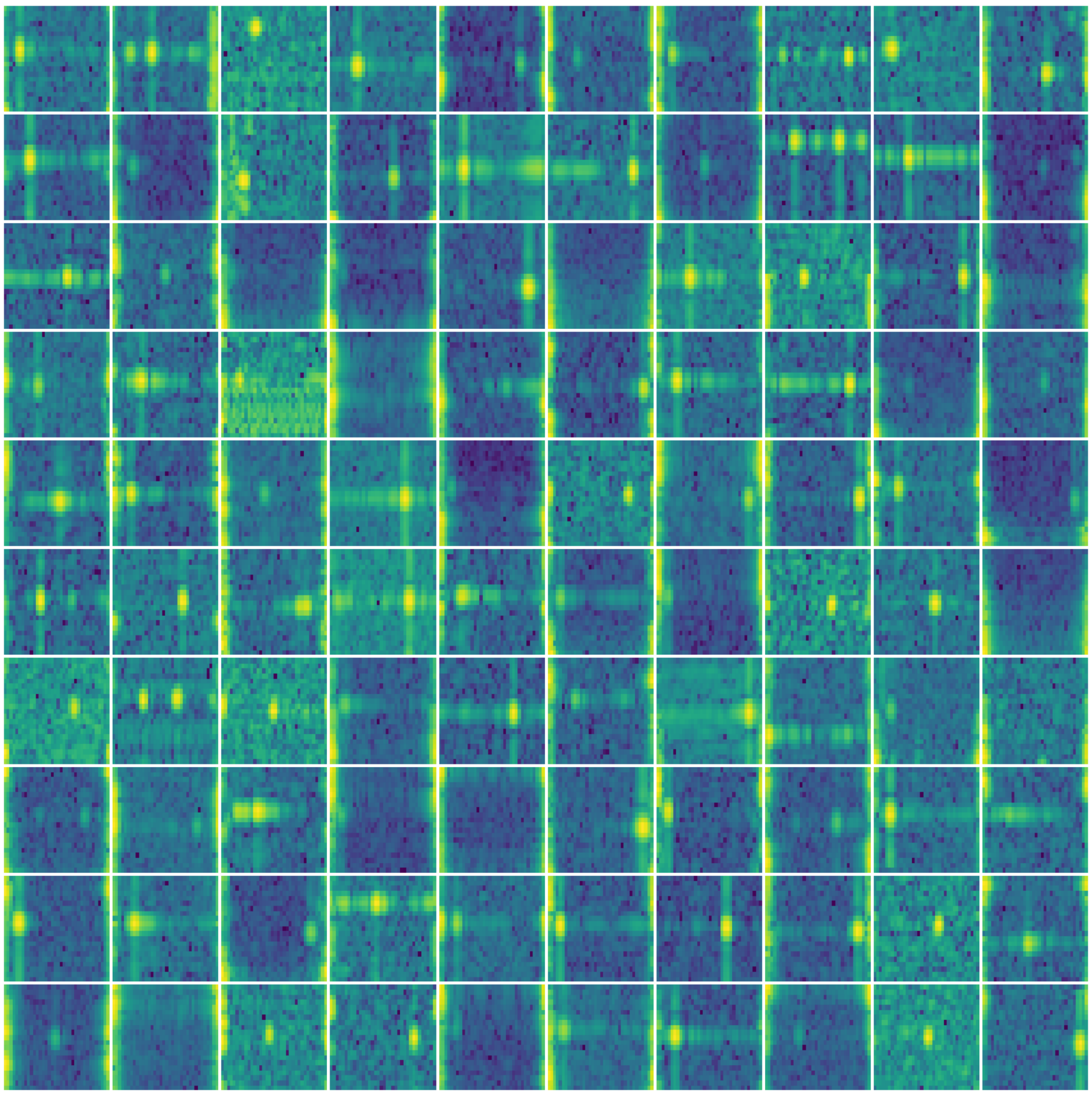


Figure B.13: Example of 100 plots of birds from the training dataset.

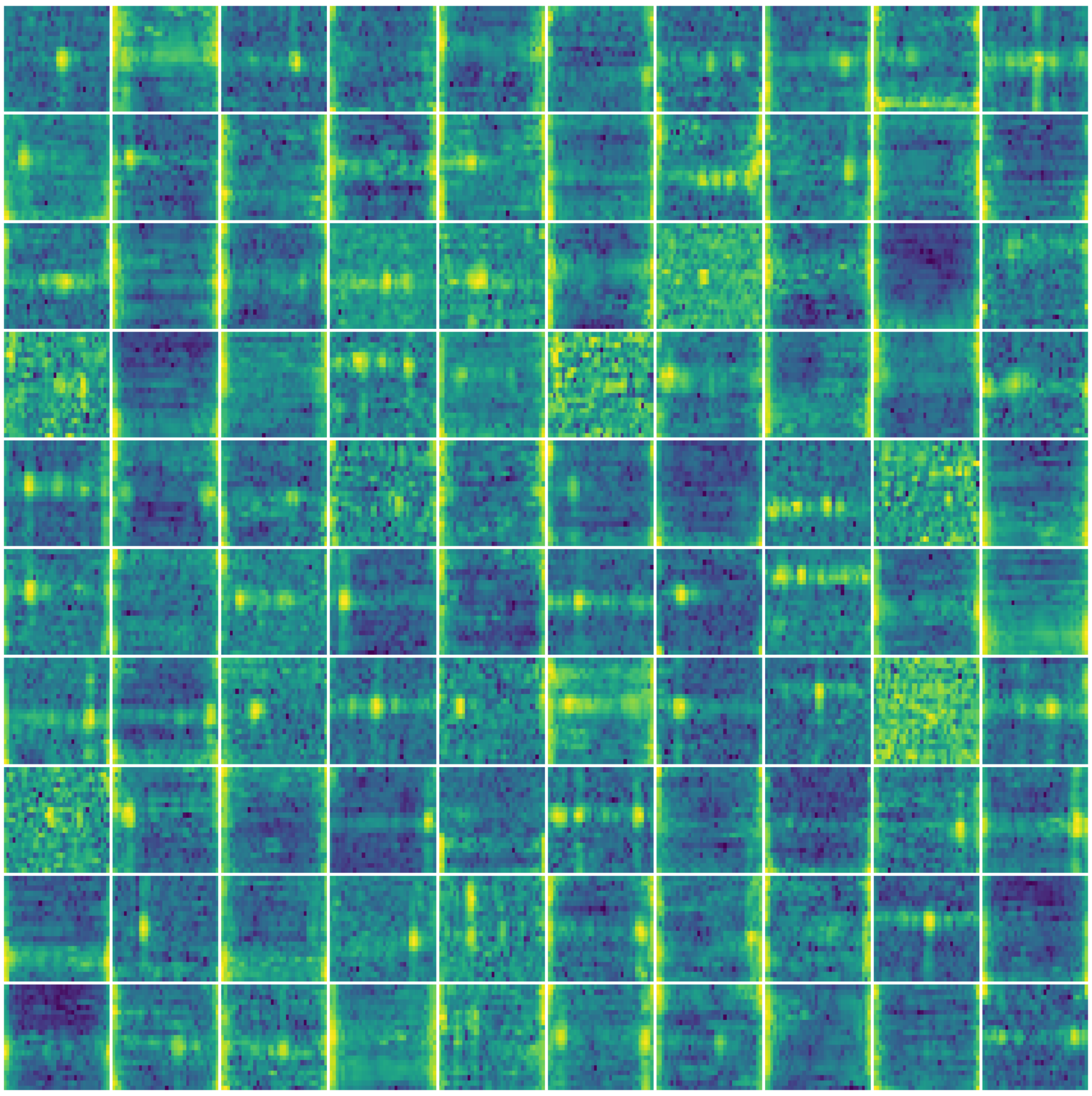


Figure B.14: Example of 100 plots of generated birds.

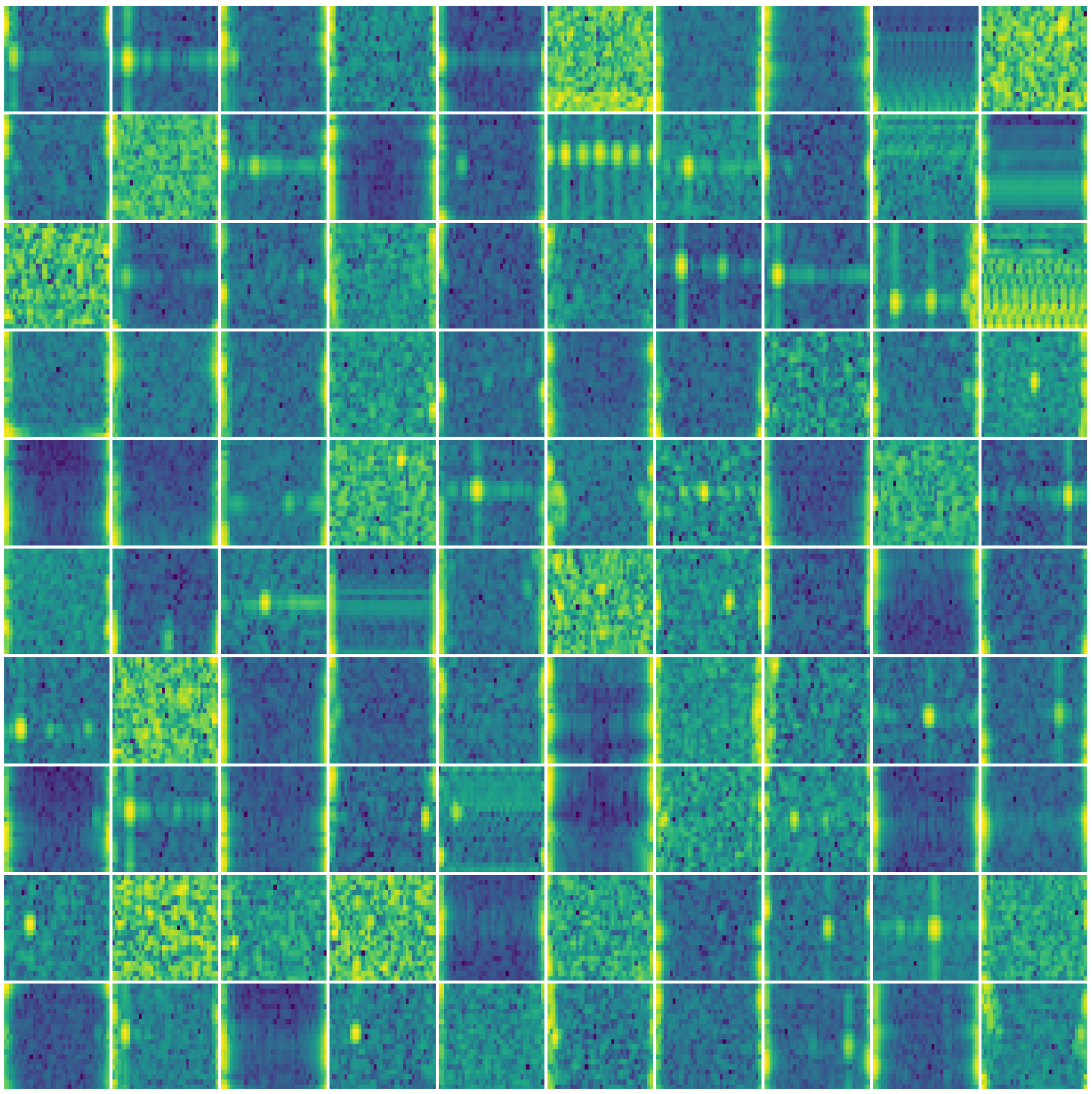


Figure B.15: Example of 100 plots of birds from the training dataset before k-means filtering.