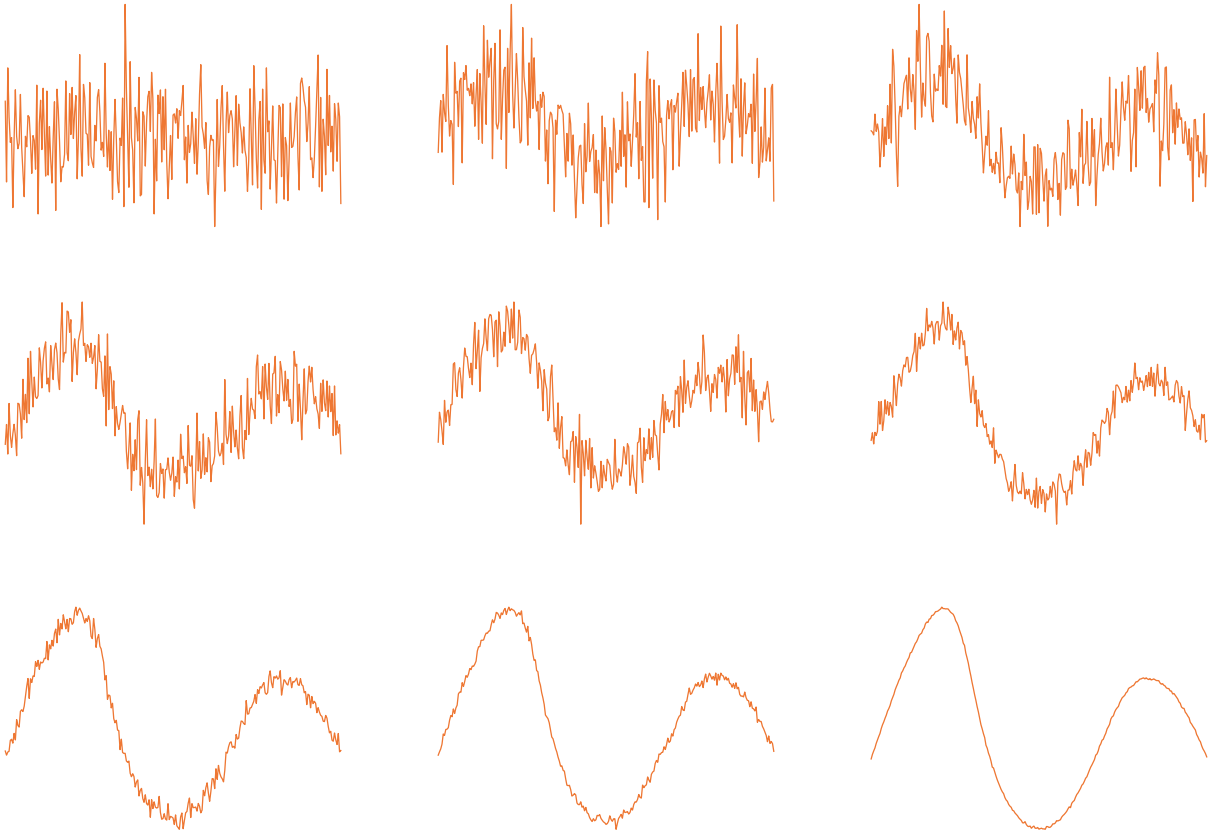


Generative Adversarial Network for Generation of Artificial Microwave Data for Stroke Detection

Master's thesis in Engineering Mathematics and Computational Science
Master's thesis in Complex Adaptive Systems

EBBA EKBLOM AND REBECCA SVENSSON

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Generative Adversarial Network for Generation of Artificial Microwave Data for Stroke Detection

EBBA EKBLOM AND REBECCA SVENSSON

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2021

Generative Adversarial Network for Generation of Artificial Microwave Data for Stroke Detection
EBBA EKBLOM AND REBECCA SVENSSON

Supervisors:
Albin Jonasson Svärdsby and Ann-Sophie Hilkert, Medfield Diagnostics AB

Examiner:
Kristian Gustafsson, Department of Physics

Master's Thesis 2021
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Demonstration of the evolution of the generated data during a training session. In the beginning only noise is generated, but towards the final epochs the output has the desired shape.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Generative Adversarial Network for Generation of Artificial Microwave Data for Stroke Detection
EBBA EKBLOM AND REBECCA SVENSSON
Department of Physics
Chalmers University of Technology

Abstract

This study aims to explore the possibilities of generating microwave data with a Generative Adversarial Network (GAN), in order to expand the existing data set and increase the performance of a stroke detection algorithm. Key challenges of the project relate to the small data set size and samples with many features. The generation of data was done with a Conditional Wasserstein Generative Adversarial Network. Due to the low data regime, the effects of adding DeLiGAN was also investigated. In addition to generating data with a GAN, this study also covers methods for the evaluation of generated data. To evaluate the quality of the generated data, a separate classifier network is utilised.

Evaluation of the generated data in classification problems, as well as visualisation of distribution coverage, indicate that the data is of good quality and represent the distribution of original data well. However, results also show that the generated data cannot completely substitute the real data, and is deemed to be lacking in some quality measure. Still, the results are promising and the project concludes that it certainly is possible to generate microwave data which is to be used for stroke detection, with great potential for further improvements.

Keywords: Generative Adversarial Networks, Wasserstein GAN, Conditional GAN, DeLiGAN, microwave, haemorrhagic stroke

Acknowledgements

To begin with, we would like to thank our supervisors Albin Jonasson Svärdsby and Ann-Sophie Hilkert for your invaluable support and all the interesting discussions. Your guidance and input to the project has been of great value and greatly appreciated.

Additionally, we would like to direct our gratitude towards Medfield Diagnostics AB for the warm welcome and for the opportunity to write our thesis with you. Working with such an intriguing topic has been an excellent way to finish our studies at Chalmers.

Finally, we want to express our gratitude to our family and friends. Your support and encouragement during this thesis and during all our years at Chalmers have been truly invaluable. Thank you for all the good and the bad times that we have been through, and for all memories we have made together.

Ebba Ekblom and Rebecca Svensson, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1

Introduction

The use of artificial intelligence is becoming more widespread in society and changing the way of living for many people. While artificial intelligence and machine learning have become an integral part of our society, it is an emerging topic within the medical field and yet to be widely used for diagnostic purposes [1]. However, artificial intelligence already shows signs of outperforming clinicians in a wide variety of medical applications, for instance, mammography screening [2] and surgical precision [3].

One technique that has had a big impact in many areas of application is deep learning. The key to a successful deep learning implementation, regardless of area of utilisation, is large quantities of qualitative data. With a large data set that is balanced between classes, correctly labelled and fully represents the population it is drawn from, there are great possibilities to use deep learning to solve a multitude of problems. Due to this, data has turned into a valuable possession and to have high-quality data in large amounts is regarded as a great asset. However, data collection can be both difficult and costly which limits the possibilities of deep learning. When having limited amounts of data, it is difficult to train a network that is generalised and robust to outliers. This makes it important to utilise the data one already has as efficiently as possible.

A common approach to make a network robust when training data is insufficient is to use data augmentation to introduce more variation to the data set which aids the learning [4]. Typical data augmentation methods for image data are to translate and rotate images and introduce noise or scaling samples. This way, the samples in the set are varied in some way which helps to create a more generalised network. Another way to combat the challenge of small data sets is to use the available data to create more samples. One recent technique to do so is using a generative adversarial network (GAN) to produce synthetic data.

A GAN was first proposed in 2014 by Goodfellow et al. [5] and has seen a fast rise in popularity since then. The network consists of two neural networks; one generator and one discriminator. The end goal is for the generator to generate synthetic data that is indistinguishable from the original data set. The discriminator will direct the learning of the generator by distinguishing between synthetic and real samples. By letting these two networks compete with each other during training, the generator

will converge to produce synthetic data that follows the distribution of the real data. Eventually, a point is reached where the discriminator cannot distinguish between synthetic and real samples any longer.

1.1 Background

Stroke is one of the most common causes of death today [6]. The term stroke typically refers to two different kinds of condition, ischaemic stroke and haemorrhagic stroke [7]. Ischaemic stroke occurs when the blood vessels in the brain are blocked by a clot, while a haemorrhagic stroke occurs when a blood vessel burst and build-ups of blood forms and damages the brain. When showing signs of stroke, time until treatment is critical [8]. Optimally, treatment should begin before arriving at a hospital which can greatly increase chances of survival and full recovery [9]. For optimal treatment, the diagnosis needs to be quick and accurate. Diagnosis is made using a combination of physical tests and scans of the patient's brain. Today, these tests can include measurements of cholesterol and blood sugar levels, blood pressure and the pulse of the patient [10]. Another important tool during the diagnostic process includes a score on the National Institutes of Health Stroke Scale (NIHSS), which is composed of 11 different items on which a patient receives a score of ability [11]. At the hospital, a computed tomography (CT) or magnetic resonance imaging (MRI) scan is performed to determine the type, location and severeness of the stroke [10].

In order to aid medical professionals in the diagnostic process, Medfield Diagnostics AB aims to launch a portable device, Strokefinder MD100 [12], for detecting stroke in pre-hospital environments. This device is not intended to be used instead of other diagnostic tools at a hospital, but rather be used as a complement to existing methods in ambulances and similar settings. The Strokefinder MD100 utilises antennas to measure the transmission and reflection of microwaves passing through the brain [13]. This measurement is then fed to an artificial intelligence algorithm that can indicate whether the patient has been affected by a stroke or not.

As for many other algorithms in the field of artificial intelligence, more data can often improve performance. However, the gathering of clinical data required for training the network is both time-consuming and expensive which limits the availability of high-quality data. Therefore, there is an interest from Medfield Diagnostics AB to investigate if new samples can be created with the use of GANs.

1.2 Scope

This project aims to investigate the possibilities of using GANs as an augmentation technique for microwave data from Medfield Diagnostics AB. The long term purpose of the investigation is to enlarge their clinical data set to enhance their stroke detection algorithm. The data sets that are used contain high dimensional microwave data and the data sets are relatively small. Since GANs are more commonly used

with image data and typically require large data sets for training, it is not obvious how and if training a GAN with these data sets is possible. Therefore, this study intends to be a proof of concept rather than optimising performance. As a starting point, a data set containing simulated measurements are used. This data set aims to simulate how microwaves ought to behave during a measurement with the stroke detection device but is both less complex than the real measurements and consists of a larger number of samples than there exist on real patients. Building on this, a data set from Medfield Diagnostics Strokefinder MD100 [12] is used as well. The samples in this set are not from measurements on patients but instead measured on a model of a human head. These are necessary steps to take before moving on to the much more complex real clinical data consisting of measurements on actual patients, which instead can be considered in future projects. Important to note is that this project will only use data representing haemorrhagic strokes.

The key challenges in the project stem from the properties of the used data. When working with deep learning, small data sets can problematic and GANs are especially sensitive to the lack of a large set of training samples. Another considerable challenge is that there are roughly 10 times as many features as the number samples for the two data sets. Therefore one focus area is how to adapt the network and the data so that GANs can be used as an augmentation method for small data sets with many features. Furthermore, as stated previously, most existing GAN implementations are based on image data and therefore use convolutional layers in the GAN. Because the microwave data in this project has no obvious image representation, the study is limited to exploring the possibilities of using multilayer perceptrons when building the GAN. An unfortunate result of this is however that the available information from previous studies becomes limited.

Considering that image data is by far the most common data used with GANs, the metrics used to evaluate a GANs performance are designed for that. A significant part of this project is therefore to determine what evaluation methods and metrics are suitable to use when working with multidimensional microwave data.

The project's main focus is to determine if it is possible to generate data of this kind with a GAN and how to evaluate the generated data. Therefore tuning of hyperparameters to optimum is not prioritised. For the most important hyperparameters, for example, learning rate, a shallow search is performed to ensure that the used values are not eradicating the performance.

1.3 Related studies

Since 2014 when Goodfellow et al. [5] first introduced the concept of combining a generator and a discriminator into a generative adversarial network (GAN), there have been extensive studies and expansions in the field. Even though the original GAN showed huge potential there are some issues. This includes mode collapse, meaning that the generator collapses to producing one type of identical samples, vanishing gradients, and difficulty in general to reach convergence. Many suggestions

for how to improve performance and counteract the known issues with the original GAN has been presented since. This includes a study by Salisman et al. [14], that aims to improve the original GAN. The article covers many suggestions on how to adapt training and data to stabilise training and reach convergence, such as using mini-batches and one-sided label smoothing.

Other studies have suggested more extensive and fundamental changes, and these often result in a new type of GAN. An example of this is the Wasserstein GAN (WGAN) [15]. Here, the Wasserstein distance is suggested to replace the original sigmoid binary cross-entropy function as a new cost function to both stabilise training and ensure that the generated samples are not identical to one another. This loss function calculates the Wasserstein distance between the training data and the generated data and uses this information to train the network. Another type of GAN that aims to solve the previously mentioned issues with the original GAN is the Least Squares GAN (LSGAN) [16]. Here, the loss function in the original GAN is exchanged for a least-squares loss function. Both of these GANs have a smooth loss function that saturates slower than the loss function of the original GAN.

In 2017 Lucic and Kurach et al. [17] performed a large scale study covering the performance gain from using a range of different cost functions. They showed that with a large computational budget, where all hyperparameters can be optimised, the cost function becomes less relevant. However, this project is limited in time and resources and therefore a large scale hyperparameter search is not feasible. Lucic and Kurach et al. [17] compared the performance for GANs with different loss functions on four different image data sets. As the complexity of the data increased, a slightly larger difference in performance between the different GAN types became visible. For simple data sets like MNIST [18], all GANs reached a very similar level of performance. When trained on more complex data sets like CelebA [19], using the Wasserstein distance showed to have an advantage. The data used in this project is not images, and cannot directly be compared to any of the data used in the study by Lucic and Kurach et al. However, due to the high dimensionality of the microwave data, it is probable that the cost function that gave the highest performance for the more complex image data will be favourable in this project as well.

In addition to changes in cost functions, there is a large variety of new types of GANs and applications that have been introduced. As already mentioned, GANs are mainly known to produce good results for image data, for example, generating realistic-looking images of people based on photographs [20]. Due to this, it is natural that most studies focus on how to improve performance for image data, for example, the Deep Convolutional GAN (DCGAN) [21]. However, studies on other types of data have been performed as well, including speech recognition [22], time series [23] and trajectory prediction [24]. Although using microwave data is a small field in GAN research, there exist previous studies that are using GANs on microwave data as well [25, 26]. A very common choice when working with non-image data and GANs is to use 2D convolutions regardless of the shape of the data. Due to this, there are not many studies available except for the original GAN paper by Goodfellow et al. [5] that uses multilayer perceptrons as this project intends to

do. Both of the studies mentioned above that use microwave data use convolutional neural networks (CNN) in their GANs which suggest that it is not required to have image data to use CNNs in GANs. However, the usage of CNNs in a GAN for the data used in this project will be left for future studies.

As stated, augmentation of scarce clinical data sets may be crucial for machine learning application in medicine. Using GANs to augment medical data has already been tested with success [23, 27, 28]. Once again, images of some kind are by far the most common data type to use in this application as well. For example, Frid-Adar et al. [28] use a DCGAN to generate more samples of liver lesions that are then combined with the original data which successfully improves the performance of a separate classifier. However, there are applications that use non-image data as well. Hyland et al. [23] use real-valued medical time series data as input to the GAN, in this case, a Recurrent Conditional GAN (RCGAN). An RCGAN is a combination of a Recurrent GAN (RGAN) [29] and a Conditional GAN (CGAN) [30]. The RGAN utilises a recurrent neural network (RNN) to build a GAN, which is suitable for time series data. The CGAN enables the GAN to generate samples belonging to separate classes by including a class label in the data. This allows the GAN to capture class-specific features that are then included in the generated data. Another application that does not use image data is a study by Liu et al. [27]. Their data is a mass spectrometry peak distribution that is used for determining cancer stages. The GAN used in this study was a Wasserstein GAN with small-scaled multilayer perceptrons building the model. The data used to train the GAN has considerably fewer features than the high-dimensional microwave data used in this project, and the number of samples was higher than the number of features in contrast to the data sets available to this project. Still, this reinforces that it is possible to use multilayer perceptrons in a GAN.

2

Theory

This thesis builds on the theory behind generative adversarial networks (GAN), which will be presented in this chapter. The implementation is however extended beyond the original GAN formulation, including a conditional GAN and the Wasserstein distance as a cost function. As stated in Section 1.2, the project also deals with small amounts of data and in Section 2.4, the theory behind a method to overcome this problem is presented.

2.1 Generative adversarial networks

The idea of a Generative Adversarial Network (GAN) was first published in an article by Goodfellow et al. [5] in 2014. The purpose of a perfectly trained GAN is to generate data with an identical distribution as the training data. To do this, a model combining two different networks are used in a way that combines supervised learning with unsupervised learning. The model is made up of a generative network G and a discriminative network D, which have opposite training objectives that combine into the GANs training objective. Figure 2.1 illustrates the coupling between the two different networks. The generator aims to produce data that is as similar to the training data as possible. The discriminator is trained to distinguish between the fake data and the real data so that when fed a sample it outputs a probability of it being real. These contradicting objectives force both of the two networks to improve as training goes on and can be described as a minimax game. As the discriminator gets better at identifying the synthetic data, the generator gets better at producing data that imitates the original data. The end goal is to reach a point where the generated data is indistinguishable from the original.

2.1.1 The discriminator

The discriminator is an ordinary classification model that uses supervised learning to classify inputs as real or fake. The input consists of either a sample of training data with label 1 or a fake sample from the generator with label 0. The network is trained to discriminate between the two and output the probability of the sample belonging to the real distribution. By using the binary cross-entropy cost,

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim p_{\mathrm{r}}}\left[\log D(x)\right] - \frac{1}{2}\mathbb{E}_{z\sim p_z}\left[\log(1 - D(G(z)))\right], \tag{2.1}$$

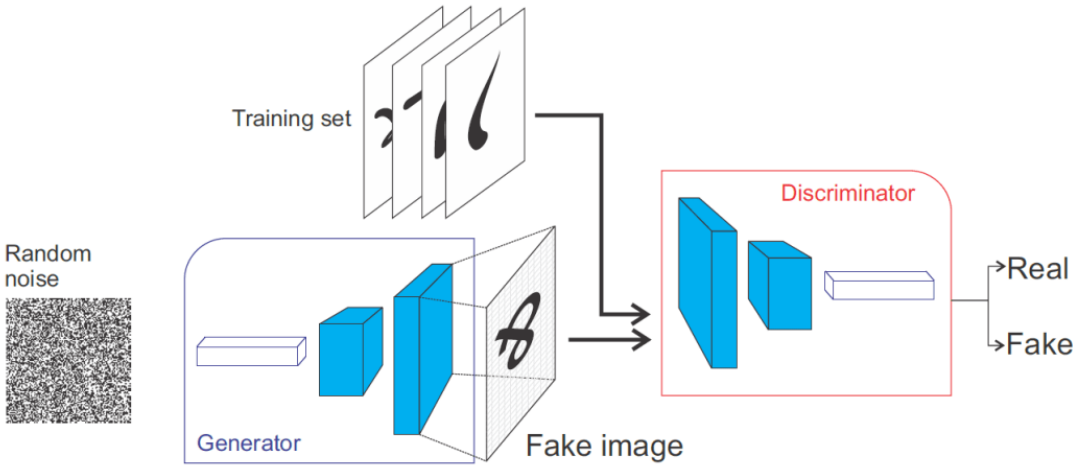


Figure 2.1: A schematic overview of a generative adversarial network. In this case the samples are images, but the data can be of any representation. Image from [31]

as the cost function, the discriminator is penalised for inaccurate classifications [32]. In this, $\boldsymbol{x}$ is drawn from the distribution of real data p_r and the generator is fed noise $\boldsymbol{z}$ from some prior distribution p_z. The cost of the discriminator is dependent of its own parameters $\boldsymbol{\theta}^{(D)}$ as well as the generator's parameters $\boldsymbol{\theta}^{(G)}$. However, it can only control its own parameters $\boldsymbol{\theta}^{(D)}$. The first term corresponds to the prediction of the discriminator for real data, and it is maximal when the discriminator is certain that the sample is real. In contrast, the second term is maximal when the discriminator is certain that the sample is fake and therefore $D(G(z))$ is 0. Optimally, the discriminator makes correct predictions for both kinds of samples which results in both terms equal to 0.

2.1.2 The generator

The objective of the generator is to generate output that is of a distribution p_g as similar as possible to the distribution of the real data p_r. This is done by feeding noise $\mathbf{z}$ from p_z as input to the generator, which then transforms this noise into a sample that is supposed to be similar to the training data. Using noise as input ensures that the produced data is varying and sampled from the entire distribution. Different data sets and network architectures perform differently with different noise distributions and dimension of input noise [33]. However, using a Gaussian distribution with dimension 100 is a common choice regardless of architecture or data set since this is often good enough. As shown in [33], there is rarely any point in increasing the dimension of the noise beyond 100 since the performance gain is minimal for most data sets and architectures.

The generator feeds into the discriminator network, which creates a dependency on the discriminator's performance. This means that there is a penalty if the generated image is not good enough and the discriminator manages to classify it as a fake sample. To achieve this, the cost function for the generator is simply

$$J^{(G)} = -J^{(D)}. \tag{2.2}$$

Due to the costs immediate relation, the training of the entire GAN can be expressed with a value function, V, as given by Equation 2.3 [5]. This value function represents the minimax game that occurs during training, which stems from the fact that the costs have opposite signs and therefore the aim is to minimise the generator cost and maximise the discriminator cost.

$$\min_G \max_D V(D, G) = \min_G \max_D (-J^{(D)}(\theta^{(D)}, \theta^{(G)})) \tag{2.3}$$

In practical applications, it is not optimal to use this cost function, Equation 2.2, for the generator since it often results in very poor performance with the network not learning anything beyond a few iterations. This occurs since the performance of G is poor in the beginning of training, which leads D to easily distinguishing between real and generated samples. As a result of this, $J^{(D)}$ becomes very small and the feedback to the generator is therefore minimal due to vanishing gradients with the term $\log(1 - D(G(z)))$ being saturated. To overcome this problem Goodfellow et al. [5] suggests to instead use

$$J^{(G)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{z\sim p_z}[\log(D(G(z)))] \tag{2.4}$$

as the cost function for the generator. There are two major differences between this cost function and the original one. The first term of the original cost function is removed since it does not contribute to the generators gradients. Furthermore, the aim is now to maximise the probability of the discriminator classifying the sample incorrectly instead of, as previously, minimising the probability of the discriminator classifying the sample correctly. Note however that the cost functions in Equation 2.2 and Equation 2.4 still have the same fixed point, but the latter one will not result in vanishing gradients in early learning [5]. The primary motivation behind this cost function is that both the discriminator and the generator will have strong cost feedback even when they are performing poorly. It is important to note that when using this cost function for the generator, the model is no longer zero-sum and it can't be described by a single value function anymore [32].

2.1.3 Model training

For the GAN to perform optimally the generator and the discriminator cannot be too imbalanced where one drastically outperforms the other. To achieve this the two networks must be trained together. However, they cannot be trained simultaneously since this would be a too complex problem due to the need for the generator and the discriminator to adapt their training to one another. Instead, training is alternated between the two networks as described by Algorithm 1.

As training goes on the capacity of D and G will increase. When the capacity of both of them is high enough convergence will eventually be reached. This means

Algorithm 1 Training schedule of a GAN where minibatch stochastic gradient ascent (D) and descent (G) is used.

1: Initialise models D and G with costs as stated above
2: **for** number of iterations **do**
3: Draw m samples of noise from p_z
4: Draw m samples from p_r
5: Update D using its stochastic gradient, using the cost in Equation 2.1.
6:
7: Draw m samples of noise from p_z
8: Update G using its stochastic gradient, using the cost in Equation 2.4.

that the output from the generator will be so similar to the samples from the real distribution that the discriminator cannot distinguish between them even though the discriminator has a high capacity.

In practice, convergence is very rare in GANs [14]. Training is hard, and a lot of data is required. Most often, the two networks performance oscillates and as one improves it undoes recent improvements of the other model. One major problem that can occur when training GANs is mode collapse. This happens when the generator learns to transform several different noise inputs to the same output sample. When this happens, the output of the generator is plausible so that it fools the discriminator initially. However, as training proceeds, the discriminator learns to recognise this sample as the fake one and therefore will always be correct. This way the network gets stuck in a local minimum and the generator continues to output one or a small set of samples over and over again.

There are many suggestions to solve convergence problems and to stabilise the training of the GAN. The most successful and popular method is to use the Wasserstein distance, also known as the Earth-Mover distance, as the cost functions for the system.

2.2 Wasserstein GAN

The original formulation of the GAN cost function quantifies the distance between the generator distribution p_g and the discriminator distribution p_r using the binary cross-entropy in Equation 2.1 [5]. However, some problems arise when training GANs, such as vanishing gradient and mode collapse, as explained above. An attempt to avoid such problems was presented by Arjovsky et al. in their Wasserstein-GAN (WGAN), where the original cost has been replaced by the Wasserstein distance [15]

$$W\left(p_r, p_g\right) = \inf_{\gamma \in \Pi(p_r, p_g)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]. \tag{2.5}$$

This is a distance measure between two distributions $p_r(x)$ and $p_g(y)$, where $\Pi\left(p_r, p_g\right)$ is the set of all joint distributions $\gamma(x, y)$ whose marginal distributions are p_r and p_g. A more intuitive interpretation of the Wasserstein distance - also called Earth-Mover

(EM) distance - is that $\gamma(x, y)$ is the amount of mass that must be transported from x to y in order to transform the distribution $p_r(x)$ into $p_g(y)$. The Wasserstein distance can then be thought of as the optimal transport plan [15]. In this section, it is demonstrated how the Wasserstein distance also can be used as a cost for a GAN.

Optimization problems may be viewed from two perspectives, the primal or the dual formulation. In general, the solution of the dual problem is a lower bound to the solution of the primal. However, if strong duality holds - as for the case to be demonstrated here - the two solutions are equal. As stated by Arjovsky et al. in their article about Wasserstein GAN, it is intractable to compute Equation 2.5 for all possible joint distributions $\Pi(p_r, p_g)$. Instead, they propose a transformation of the Wasserstein distance [15], using the Kantorovich-Rubenstein duality [34] to get

$$W(p_r, p_g) = \frac{1}{K} \sup_{\|f\|_L \leq K} \mathbb{E}_{x \sim p_r}[f(x)] - \mathbb{E}_{y \sim p_g}[f(y)]. \tag{2.6}$$

In this dual form of the Wasserstein distance, $f(x)$ must be a Lipschitz continuous function, where K is the Lipschitz constant. Furthermore, if the function f is parametrised by $\omega \in \Omega$ such that $\{f_\omega\}_{w \in \Omega}$ where Ω is a compact space, then the Wasserstein distance can be used as a cost function and be computed up to a multiplicative constant as

$$J(p_r, p_g) = W(p_r, p_g) = \max_{\omega \in \Omega} \mathbb{E}_{x \sim p_r}[f_\omega(x)] - \mathbb{E}_{z \sim p(z)}[f_\omega(g_\phi(z))]. \tag{2.7}$$

This means that the discriminator will learn the function f that is parametrised by ω. The objective of the generator is to learn the function g_ϕ, which is the generator function parametrised by ϕ. This is achieved by instead minimising Equation 2.7 over ϕ, similarly to the min-max situation shown for the original GAN cost in Equation 2.3. Learning is done by differentiation and backpropagation, and the discriminator weights and generator weights are updated separately using gradient descent as before. The discriminator weights are updated by

$$\nabla_\omega \left[\frac{1}{m} \sum_{i=1}^{m} f_\omega(x^{(i)}) - \frac{1}{m} \sum_{i=1}^{m} f_\omega(g_\phi(z^{(i)})) \right] \tag{2.8}$$

and the generator weights ϕ are updated by

$$-\nabla_\phi \frac{1}{m} \sum_{i=1}^{m} f_w(g_\phi(z^{(i)})), \tag{2.9}$$

where z is a random variable, and g_ϕ is the function of the generator parameterised by ϕ [15].

Using the Wasserstein distance as a cost function means that the discriminator no longer yields a probability of a sample being real or fake. Instead, it becomes a critic that gives a measure of the realness of each sample, which in practice is achieved by using a linear output of the network. The cost of the critic is now also interpretable as the distance between the distribution of generated samples and the

real distribution of samples. As it approaches zero, the two distributions approach each other. Furthermore, as opposed to the discriminator in the original GAN formulation, the critic can be trained to optimality. Overtraining the discriminator in the original GAN leads to vanishing gradients, but Arjovsky et al. demonstrate how the critic cannot saturate and instead converges to a linear function that gives clean gradients [15]. This also means that mode collapse is avoided for Wasserstein GANs. In practice, the critic is trained n times for every time the generator is trained, and this is set to $n = 5$ by Arjovsky et al. [15].

2.2.1 Ensuring Lipschitz-continuity

The theory of the Wasserstein metric implies that f is a K-Lipschitz continuous function. If the weights ω lie in a compact space Ω, then all the functions f_ω are K-Lipschitz for some K. In the original paper, weight clipping is suggested as a method to ensure that the weights are contained in a compact space. This means that the weights are restricted after each update to a compact space $[-c, c]$.

The weight-clipping method has however been associated with problems with convergence, and an improved version of the Wasserstein GAN has been proposed that uses gradient penalty instead [35]. Since a differentiable function is 1-Lipschitz if it has at most a norm of 1 everywhere, we can instead constrain the gradient norm of the critic with respect to the output. This is achieved by adding a penalty term to the cost function. The real samples are denoted x and follow the distribution p_r, and generated samples are denoted $\tilde{x}$ and follow the distribution p_g. In addition to these, samples $\hat{x}$ are drawn from a sampling distribution p_s. This distribution is implicitly defined by sampling from straight lines between pairs of points in p_r and p_g. As in Section 2.1, the critic will now be denoted $D(x)$ and the generator function will be denoted $G(z)$, where z comes from the normal distribution $p(z)$. With the penalty term added to Equation 2.7, the new critic cost function becomes

$$J^{(D)} = \mathbb{E}_{x\sim p_r}[D(x)] - \mathbb{E}_{\tilde{x}\sim p_g}[D(\tilde{x})] + \lambda \mathbb{E}_{\hat{x}\sim p_s}\left[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2\right], \tag{2.10}$$

where λ is the penalty weight parameter.

A problem arises when using the gradient penalty in combination with batch normalisation, which has been used by most other GAN implementation. Since the objective penalises with respect to each input independently, it becomes invalid under batch normalisation. An alternative to using batch normalisation, or to not use normalisation at all, is to use layer normalisation, which does not introduce correlation between samples. Instead, a layer is normalised over all the summed inputs to the neurons in that layer. Thus all the units in a layer share the same mean and variance, as opposed to batch normalisation where the mean and variance for each neuron is computed across samples [36].

2.3 Conditional GAN

The theory up to this point can be used to generate samples that approach the real distribution p_r. However, there is no control of the class of each separate sample. For the purpose of augmentation, it is crucial to know the class label of the samples if they are to be used in training. In order to generate data with a specific mode, Mirza and Osindero proposed the conditional GAN (CGAN) where samples are conditioned on class labels [30].

The idea behind the conditional GAN (CGAN) is to condition the generator and the discriminator on the same extra input y, for instance, a class label. The cost function for the original GAN formulation Equation 2.1 then becomes

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{x\sim p_r}\left[\log D(x|y)\right] - \frac{1}{2}\mathbb{E}_{z\sim p_z}\left[\log(1 - D(G(z|y)))\right] \tag{2.11}$$

When applied to a Wasserstein GAN the cost becomes

$$J^{(D)} = \mathbb{E}_{x\sim p_r}[D(x|y)] - \mathbb{E}_{z\sim p(z)}[D(G(z|y))] + \lambda\mathbb{E}_{\hat{x}\sim p_s}\left[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - 1)^2\right], \tag{2.12}$$

with the distributions p_r, p_g and p_s as defined in Section 2.2.1

The two inputs are combined in some hidden representation. According to Mirza and Osindero [30], the adversarial network allows for considerable flexibility in how the hidden representation is composed. In their implementation, the two inputs are mapped to separate hidden layers, and the two hidden representations are then concatenated. Others have instead embedded the label y into a vector that can either be concatenated or multiplied with a hidden input representation[37]. Regardless of how the two inputs are combined, it may however be an advantage to combine them early in the network [38].

Figure 2.2 demonstrates the results from the first article on conditional GANs when conditioning the data set MNIST[18] on the class label 0-9[30]. The conditioning makes it possible to determine from which class a generated sample should be, and each row shows generated samples from one specific class. This is a clear example of how the extra conditioning has directed the learning towards class-specific features.

2.4 DeLiGAN

Typically, GAN implementations work with very large data sets. However, when the aim is to use a GAN for augmentation, the existing data set may be relatively small. This means that some adaptations to the ordinary GAN implementations are required in order to still learn a complex distribution of samples when the set size is small. When Gurumurthy et al. proposed DeLiGAN (Generative Adversarial Networks for Diverse and Limited Data) they argue that for a diverse but small data set, it is infeasible to increase the depth of the network [39]. Instead, they suggest a

Figure 2.2: An example of conditioning the data set MNIST on the class label 0-9 from the first implementation of a CGAN by Mirza and Osindero[30]. The conditioning makes it possible to select which class the generator should create a sample of, and each row shows generated samples from one class.

reparametrisation of the latent space into a Gaussian mixture model [39], such that

$$p_z(z) = \sum_{i=1}^{N} \psi_i g(z_i|\mu_i, \Sigma_i). \tag{2.13}$$

Here $g(z_i|\mu_i, \Sigma_i)$ represents the probability of z in the normal distribution $\mathcal{N}(\mu_i, \Sigma_i)$, and the weights ψ_i are assumed to be uniform. By letting μ_i and Σ_i be trainable parameters, the modelling power of the prior distribution is increased. This allows for a more complex distribution of samples and also directs learning towards high probability regions in the latent space.

To sample from this distribution, one of the N Gaussians is selected and then the reparametrisation trick introduced by Kingma et al. [40] is used. A sample from the chosen Gaussian is represented by a deterministic function of μ_i and σ_i, such that

$$z = \mu_i + \sigma_i \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, 1). \tag{2.14}$$

Thus, one only needs to sample $\epsilon \sim \mathcal{N}(0, 1)$ and then compute Equation 2.14. The goal is to add $\boldsymbol{\mu} = [\mu_1, ..., \mu_N]^T$ and $\boldsymbol{\sigma} = [\sigma_1, ..., \sigma_N]^T$ as learnable parameters to the generator. Samples from the N Gaussians are generated using Equation 2.14, which is then passed on to the generator and discriminator. During training, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are updated using gradients that arise from the generator's cost function. However, an issue arises since $p_r(G(z))$ has maxima at the μ_i. This means that the generator will try to decrease the σ_i, and they may collapse to zero. To avoid this, an L_2 regulariser is added to the generator cost, which takes the form

$$J^{(G)} = \mathbb{E}_{z \sim p(z)}[D(G(z|y))] + \lambda \sum_{i=1}^{N} \frac{(1 - \sigma_i)^2}{N}. \tag{2.15}$$

The purpose of the L_2 regulariser is to penalise the σ_i as it approaches 0. Instead, the regulariser will push the value of the σ_i towards 1, as this is where the term has its minimum value. Gurumurthy et al. show with experiments that DeLiGAN outperforms classical GAN models in low data regimes and leads to more stable training.

3

Methods

This project is defined by the unique data used to train the GAN. In this section, a more in-depth description of the used data sets and the preprocessing of the data is given. Following this, the training procedure and GAN architectures are described. Finally, the section covers how the GAN is evaluated.

3.1 Data sets

Two different data sets have been used throughout this project. This section aims to describe these data sets in more detail and the similarities and differences between them. The first data set consists of simulated data, which is a simulation of how the microwaves ought to behave when passing through the head of a patient. The other data set consists of real measurements on an artificial head, referred to as a phantom patient. Table 3.1 gives an overview of the number of samples and features in the two data sets, including how the two sets split into training, test and validation sets. The two data sets are described in more detail in Section 3.1.1 and 3.1.2.

Table 3.1: An overview of the two different data sets that are used to train the GAN including how each data set is split into a training, a test and a validation set. See Section 3.4 for further information regarding how the different sets are used during evaluation, and Section 3.1.1 and 3.1.2 for further information regarding the two data sets.

Number of:	Simulated data	Phantom data
Samples	4000	2605
Samples in training set	3306	2152
Samples in validation set	174	114
Samples in test set	520	339
Total features	136000	22680
Used Features	32040	7056

3.1.1 Simulated data

The samples in this set originate from simulations of how antennas would register microwave data on patients with cerebral haemorrhage and healthy patients. This

gives measurements that have no noise and the interference between the antennas is minimal. The data set contains 4000 samples, evenly distributed between the two classes healthy and bleeding. With each sample, there is metadata available such as location and size of the bleeding, size of the simulated head and thickness of the patient's hair.

There are $N = 16$ antennas in each measurement which are placed according to Figure 3.1. During a measurement, the antennas take turn in transmitting and receiving, resulting in $N^2 = 256$ combinations. Due to symmetry between the antenna pairs, each measurement consists of $N(N+1)/2 = 136$ transmitter-receiver pairs, referred to as channels. 16 of these are channels where the receiving antenna is the same as the transmitting antenna, therefore these channels are referred to as reflection channels. The remaining $N(N-1)/2 = 120$ channels are referred to as transmission channels. For each channel, a frequency sweep with 1000 measured points is performed in the interval 0 to 3 GHz. This results in 136.000 features.

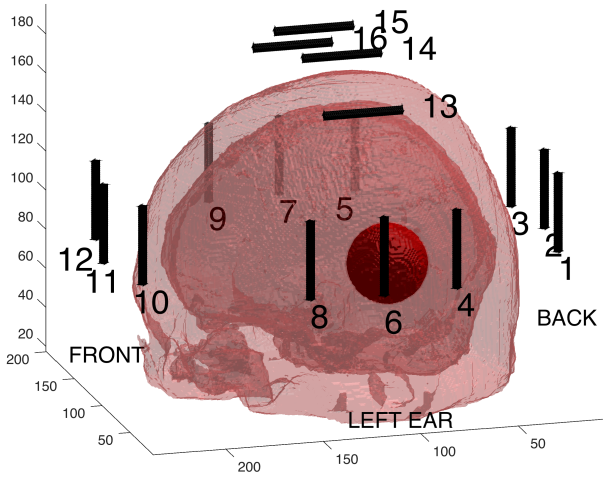
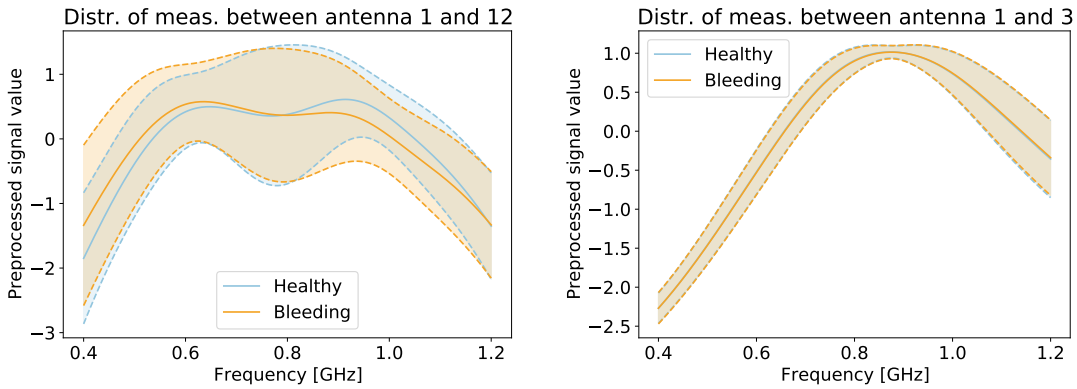


Figure 3.1: A schematic view of the placement of the antennas when generating the simulated samples. The dark red area represents a bleeding which would impact the behaviour of microwaves transmitted and received from the antennas. Image generated by Medfield Diagnostics AB.

If a bleeding is present, this will show up on multiple channels. Therefore it is important that the GAN can catch correlations between channels as well as catching features in a single channel. The shapes of the signals vary a lot between channels, and the variations between samples are different for different channels. Antennas that are located closely together, for example when antenna 1 is transmitting and antenna 3 is receiving, result in low variations between samples. For antenna pairs where the signal passes through a more significant part of the brain, for example when looking at the signal passed from antenna 1 to antenna 12, the variations between samples is a lot larger. The signal shape in these kinds of channels seems to be more affected by the presence of a bleeding. This is very reasonable since the

signal is more likely to pass through the area of a possible bleeding. The variations are shown in Figure 3.2 using the mean and standard deviation of the signals in the respective channels.



(a) **(b)**

Figure 3.2: The mean value of all samples in the simulation data set is shown along with the standard deviation. The data set is split into healthy samples and bleeding samples, and then preprocessed as described in Section 3.2 with the exception that the data is not standardised over samples. In (a) the signal passes from the back of the head to the front but in (b) the antennas are located beside each other.

3.1.2 Phantom data

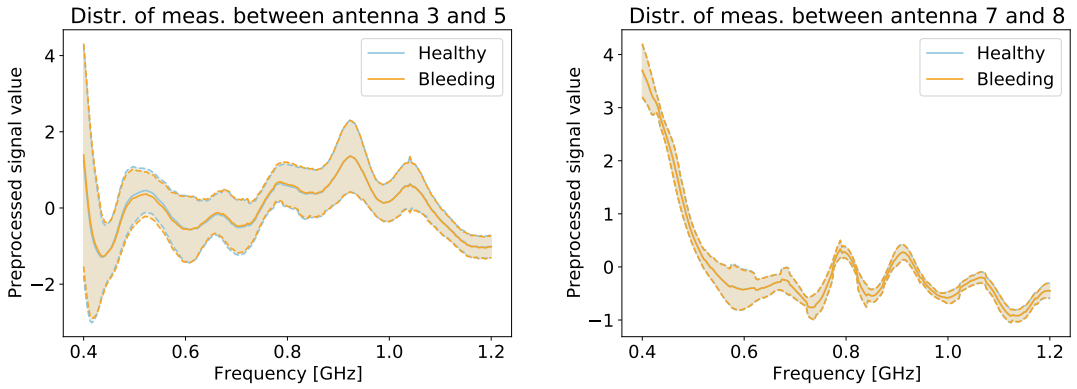
The samples in this data set are one step closer to real clinical data since they are created using the stroke detecting device Strokefinder MD100 [12] but measuring on a replica of a human head instead of a person. Since the measurements are collected using a physical device rather than being simulated, the measurements are not ideal and observational errors are present due to both systematic errors in the device and random errors that will always occur in a measurement. This results in samples that have higher levels of noise than the simulation data. In total there are 2605 samples, out of which 1301 are samples with a bleeding present and 1304 are of the healthy class. The difference in the number of samples in the two classes is small enough to consider them to be evenly distributed between the two classes.

One major difference between the phantom data and the simulated data is the number of features. Measurements are performed using only $N = 8$ antennas, and thus resulting in $N(N+1)/2 = 36$ channels using the same measuring method as described for the simulation data. Out of these 36 channels, 8 are reflection channels and $N(N-1)/2 = 28$ are transmission channels. The antennas are placed similarly to the illustration of the antenna placement for the simulation data in Figure 3.1, but reduced in number. Similarly, as for the simulation data, a frequency sweep is done for each channel. For this data set, each frequency sweep contains 630 measured points in the interval 0 to 2 GHz. In total there are approximately 23,000 features.

Another key difference is that all bleedings are located at only 10 specific locations in the phantom data, in contrast to the simulation data. The size of the bleedings

is still varied, however. The restriction of bleedings to fixed positions reduce the complexity of the data in this particular aspect, but this does not necessarily mean that the overall complexity is lower than for the simulated data.

Compared to the simulation data, the differences between samples where a bleeding is present or not are very small. In Figure 3.3, the mean and standard deviation of all samples are shown for two pairs of antennas. Antennas 7 and 8 are placed on the back of the patients head, thus this channel corresponds to a measurement between antenna 1 and 3 in the simulated data. Therefore, the similarities between the healthy and bleeding samples in Figure 3.3a are expected, just as shown in Figure 3.2a for the simulated data. A more distinct difference between the healthy class and the bleeding class for phantom data was found for the channel corresponding to measurements between antennas 3 and 5. Antenna 3 is located by the patients left ear, and antenna 5 is placed just above the patient's right ear. Note that for simulated data, the channel corresponding to measurements between antenna 1 and 12 was used instead to visualise a difference between healthy and bleeding samples. However, it is reasonable to believe that the different positions of bleedings in the two data set affect which channels that show the clearest distinction between the classes. Still, the difference between the two classes in the phantom data is very small even for this channel.



(a) **(b)**

Figure 3.3: The mean value of all samples in the phantom data set is shown along with the standard deviation. The data set is split into healthy samples and bleeding samples, and then preprocessed as described in Section 3.2 with the exception that the data is not standardised over samples. In (a) the signal passes from one side of the brain to another, (b) the antennas are located close to one another.

3.2 Data preprocessing

Since the data has a lot of features compared to the number of samples, the first step in preprocessing is to remove unwanted and unnecessary features. By visualisation of the measurements, it is noticeable that not all frequencies in the frequency sweep contain information and only a specific frequency interval is required to fully

represent the data. Therefore, this frequency interval between 0.4 and 1.2 GHz is selected, so that only about one-fourth of the frequency measurements remains. In addition to this, only the transmission channels are used since the reflection channels only contain limited information for bleeding classification. The smaller frequency interval and fewer channels result in a reduction in the number of features from $\sim 140,000$ to $\sim 30,000$ and from $\sim 23,000$ to $\sim 7,000$ for the two data sets respectively. This is unfortunately still a considerable amount of features compared to the small number of samples available.

The microwave data is originally given in complex values but to simplify the data the absolute values are used instead. The information loss that follows from this choice has not been investigated, and a classifier may perform better if the complex values were used. However, it is possible to train a classifier using absolute values and since training a classifier to optimum is not the goal of this project the information loss due to the removal of the phase is not seen as an issue.

Lastly, the data is standardised by

$$x_{standardised} = \frac{x - \bar{x}}{\sigma_x}, \tag{3.1}$$

which centers the data around 0 with a standard deviation of 1. As described previously in Section 3.1, the samples are 2-dimensional. Therefore, standardisation is first computed across the frequencies to receive the same signal amplitude for all samples. If the samples are not standardised across frequencies, the signal amplitude will vary with the size of the patient's head. After this, the data set is also standardised across samples, as is common practice when training neural networks.

3.3 Training of the GAN

A conditional Wasserstein GAN with gradient penalty (CWGAN-GP) is used in this project. The combination of the conditional GAN and the Wasserstein GAN results in the cost function in Equation 2.12 as described in Section 2.2 and 2.3. The training procedure for a CWGAN-GP is described in Algorithm 2 along with values of major hyperparameters. In addition to this, a CWGAN-GP combined with DeLIGAN is used. When adding DeLIGAN to the CWGAN-GP, the input noise to the generator is scaled and translated according to trainable parameters. The CWGAN-GP is also trained with noise added to the training data as a means of regularising and generalising the model. The added noise has a Gaussian distribution with a mean of 0 and a standard deviation of 0.05. Lastly, the CWGAN-GP combined with DeLiGAN with noise added to the input data is used as well.

3.3.1 Network architecture

A limitation of this project is to only use multilayer perceptrons in the GANs. However, as stated in Section 1.2, the vast majority of GAN implementations use convolutions, which is reasonable when working with image data. The samples

Algorithm 2 Training schedule of a CWGAN-GP where the optimiser Adam is used with learning rate 10^{-5}. When training the GAN the parameters $epochs = 10000$, $n = 5$ and minibatch size $m = 64$ are used. The prior distribution p_z is chosen to be a Gaussian distribution with mean 0 and standard deviation 1, and z is of dimension 100. If DeLIGAN is used, then the noise drawn in row 4 and 13 will be scaled and translated with trainable parameters. If noise is to be added to the training data, then this is done in the critic on row 7 and 14.

1: Initialise models D and G using the loss from Equation 2.12
2: **for** number of *epochs* **do**
3: **for** n iterations **do**
4: Draw m noise samples z from p_z
5: Generate m fake samples, $G(z)$
6: Draw m real samples from training set
7: Get predictions, $D(x)$, for real and fake samples
8: Compute loss for D based on predictions
9: Update D using its stochastic gradient
10: Draw m real samples from validation set
11: Get predictions, $D(x)$, for real samples
12:
13: Draw m samples of noise from p_z
14: Get predictions, $D(x)$, generated samples
15: Update G using its stochastic gradient

from the data set used in this project can be represented in 2D, with channels lined next to each other. However, one can argue that computing a 2D convolution on this does not make sense, because there is no obvious spatial correlation across channels. Furthermore, computing 1D convolutions on a whole flattened sample may be unreasonable due to the discontinuity points between channels. Lastly, there is the possibility of running 1D convolutions on the channels separately. However, since the position of a bleeding may affect which channels change the most due to the presence of the bleeding, it may be unwise to treat the channels independently. For instance, the computations on a single channel may have the prediction that it is a healthy channel when in reality there is a bleeding that only shows on certain other channels. More specifically for a GAN, this may lead to that each channel must show a bleeding in order for a sample to be classified as a bleeding, which may not be the case for real samples. Due to the scope and limitations of this project, it has not been investigated if using 1D convolutions on the channels separately results in sensible data. Instead, the GAN implementation in this project is focused on dense fully connected layers for both the generator and critic, where a whole sample is inputted as a flattened array.

The GAN consists of a separate generator network and a critic network, that are combined into one GAN model. The overall structure of both networks can be seen in Figure 3.4. The generator consists o four hidden layers with 64, 128, 128 and 256 neurons respectively where leaky ReLu is used as an activation function and batch normalisation is applied to all hidden layers. On the output layer, no

activation function is applied so that the output is linear. The critic network has three hidden layers with 256, 64 and 16 neurons each. The activation function used in this network is leaky ReLu as well, but layer normalisation is used instead of batch normalisation on all hidden layers. A dropout layer is placed immediately after the input layer in the critic to increase the generalisability of the network. This network also has a linear output with no activation function applied to it.

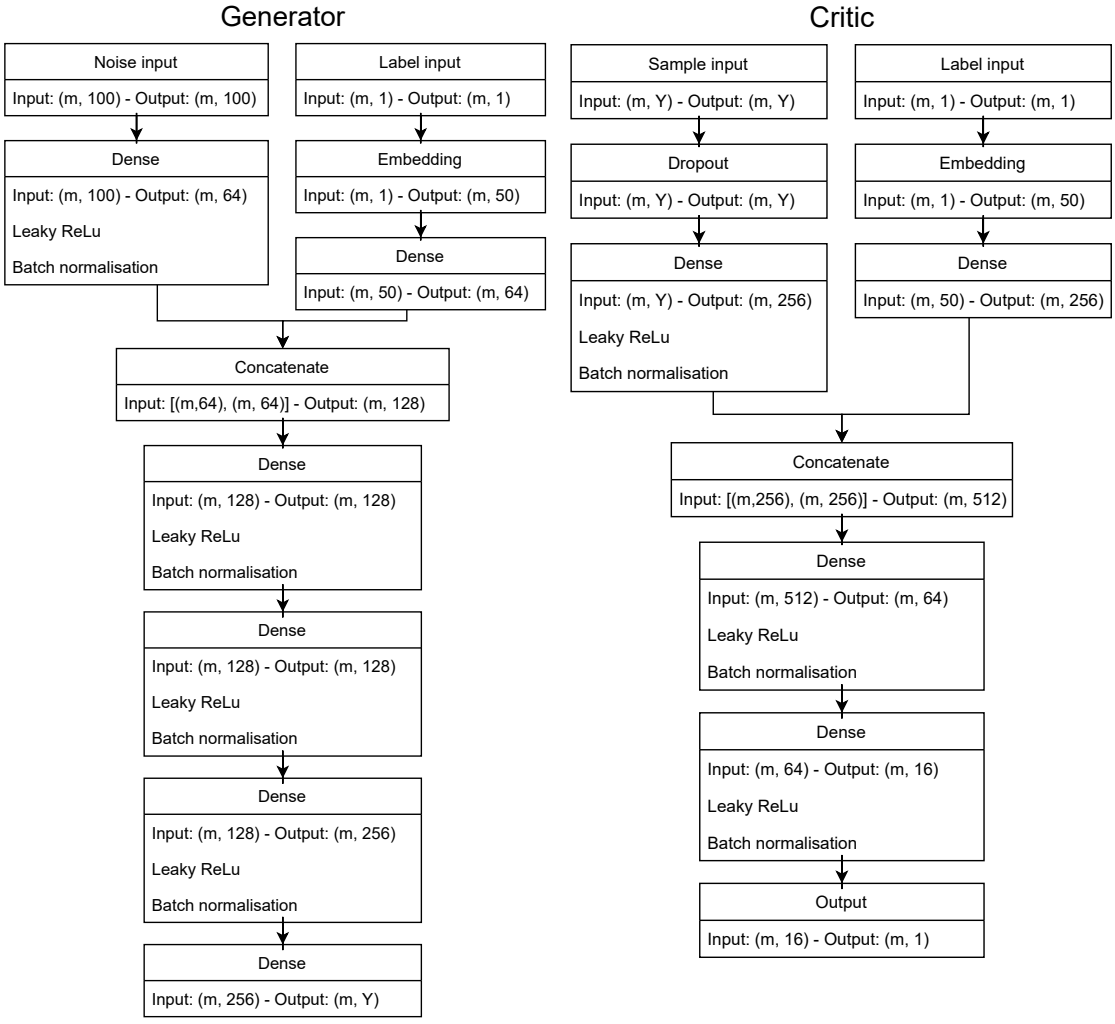


Figure 3.4: Network architecture for the generator 3.4a and the critic 3.4b of a CWGAN-GP. The minibatch size m is set to 64 and the number of used features Y in the sample depends on the data set, see Table 3.1. When DeLIGAN is added, the noise input to the generator is translated and scaled before going into the first dense layer. If the GAN is trained with noise added to the training data, then this is done just before the dropout layer in the critic.

3.3.2 Overfitting during training

In all kinds of machine learning, using small data sets often result in problems like overfitting and GANs are no exception to this [41, 42]. When feeding a limited number of samples to the critic it starts to overfit and learn to identify these particular samples in the training set rather than learning to recognise the distribution. This results in the generator receiving feedback from the critic that pushes the generator to create copies of existing samples in the training set.

A first step to counteract overfitting is to identify when the critic begins to overfit. To do this, a validation set is taken out of the training set and it is used to gain knowledge about the critic's ability to generalise by comparing the outputs from the critic for the training set, validation set and generated samples. The critic output is a score on how real or fake the critic considers a sample to be, where a higher score indicates the sample is considered more real. If the output for real and generated samples are very similar, the critic has a hard time distinguishing between them. If the differences in outputs are large, however, then the generator does not produce good enough samples. When using a validation set, comparing the output of the critic for validation samples and training samples provides information about overfitting. If the critic is overfitted it will have learnt to recognise only the samples in the training set as real, rather than general features in the distribution of real samples. Therefore the validation samples will be scored as fake samples as well [41], and the generator will be forced to only create copies of the training set in order to fool the discriminator.

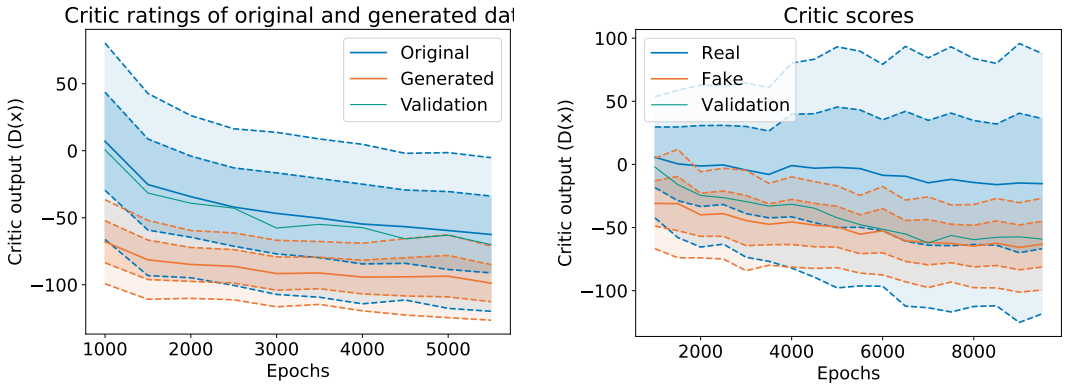
Figure 3.5 visualises the mean and standard deviations of critic ratings of the real training set, the generated samples, and the unseen real validation set. Figure 3.5ademonstrates the desired behaviour of the critic. The mean score of original samples from the training set is higher than the mean score of the generated samples, and the mean score of samples from the validation set is scored similarly to the real samples in the training set. Figure 3.5b instead demonstrates the outputs from an overfitted critic. In this figure, the mean score of samples from the validation set is scored similarly to the generated samples, which indicates that the critic only recognises the samples from the training set as real and cannot generalise this to the validation set.

3.4 Evaluating performance of a GAN model

There are many ways to measure the performance of a GAN model, but not yet an established preference for one or another metric. Many of these measurements are designed to be used with image data, and they are therefore not applicable to this study.

3.4.1 Description of chosen evaluation methods

Since the main aim of the project is to explore the possibilities of using a GAN model to generate data in order to improve the accuracy of a classifier, it is sensible to



(a) **(b)**

Figure 3.5: Critic outputs for real and fake samples and a validation set of real samples. The mean rating of each data set is represented by the full lines, and the first and second standard deviations are represented by the dashed liens. Figure 3.5a shows the desired behaviour where no overfitting is present, and the validation set is rated similarly to the real data in the training set. In contrast, Figure 3.5b shows the critic ratings when the critic is fully overfitted, and the validation set is rated similarly to the fake generated data.

evaluate the model in a way that assesses this property in particular. Two evaluation methods well suited to this was presented by Hyland et al. [23]. These two methods, Train on Synthetic, Test on Real (TSTR) and Train on Real, Test on Synthetic (TRTS) aim to evaluate the generators ability to produce representative data.

When using the TSTR evaluation technique, the original data set is split into a training set and a test set. The training set is used as input to the GAN. After the training phase is completed, the outputs of the generator are fed as the input training data to a classifier, with the purpose to classify healthy and bleeding samples. Once the classifier is trained, the test set of the original data is used as a test set for the classifier. Algorithm 3 gives an overview of the TSTR evaluation method. The TSTR method especially evaluates to what extent the generated data can be used in real applications. This makes this evaluation method very good for this area of application.

The TRTS method is in a sense the reverse of the TSTR method and the evaluation process is described in Algorithm 4. Similarly, the training set from the original data is used as input to the GAN and synthetic data is generated after training is completed. Now, the test set from the original data set is used to train a classifier and the synthetic data is used to test the classifier. The TRTS method instead focuses on evaluating if the generated data is convincing. In contrast to the TSTR method, TRTS would not be able to catch if the generator suffered mode collapse. In general, TRTS can give a lot of information about the performance of the GAN but TSTR is more ideal for this project. However, both will be used and compared to when a classifier is both trained and tested on real data.

Algorithm 3 Process of evaluating generated data using TSTR method.

1: Split data set in *train*, *test*
2: Train GAN using *train* to obtain D, G
3: Generate *synthetic* data with G
4: Train *classifier* with *synthetic* data
5: Test *classifier* using *test* data

Algorithm 4 Process of evaluating generated data using TRTS method.

1: Split data set in *train*, *test* data
2: Train GAN using *train* to obtain D, G
3: Generate *synthetic* data with G
4: Train *classifier* with *test* data
5: Test *classifier* using *synthetic* data

An article by Liu et al. [27] has a closely related topic to this project and uses an evaluation method similar to the TSTR method successfully. They aim to use a Wasserstein GAN to increase a data set containing cancer-staging data in order to improve the performance of a classifier. The key difference between their method and TSTR as proposed by Hyland et al. is that Liu et al. only uses synthetic samples that managed to fool the critic instead of using all generated samples as Hyland et al. do. The workflow of the method used by Liu et al. is shown in Figure 3.6.

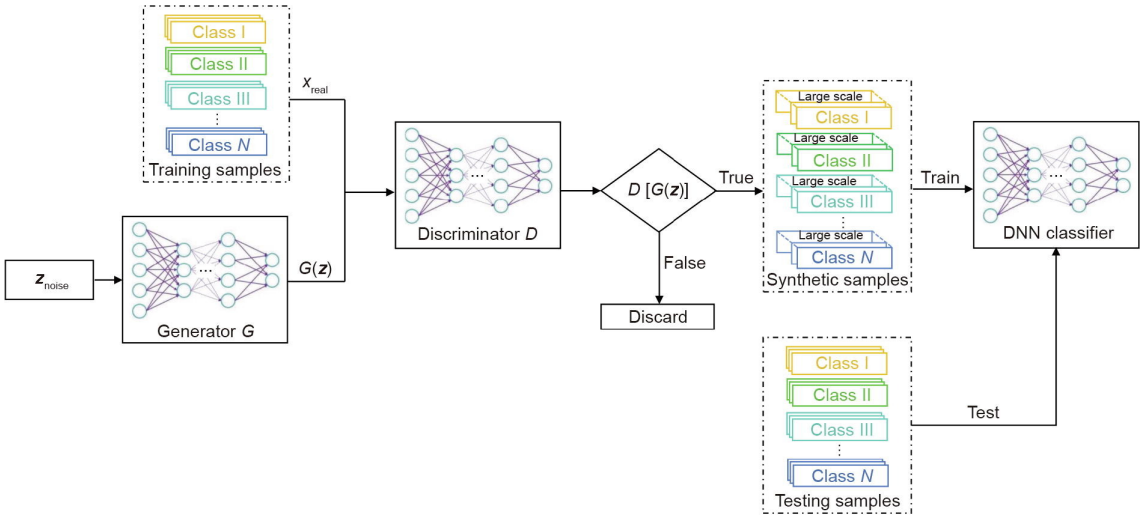


Figure 3.6: Demonstration of the evaluation method used by Liu et al. [27]. Samples that are labelled as false are discarded, while samples labelled as true, i.e. samples that fooled the critic, are fed to the DNN classifier, where false and true correspond to fake and real in this project.

There are both advantages and disadvantages in using only synthetic samples that fools the critic. By using the critic to select synthetic samples that pass as real samples, it is ensured that the samples added to the data are of high quality. However, when the aim is to use the generator as a way to augment a data set, it is important that the synthetic data represents the entire distribution. It is possible that the generator is able to produce a subset of samples with a high accuracy, and by adding only this subset a bias can be introduced in the data set. Furthermore, the purpose of the evaluation method is to give a measurement of the performance of the generator. By using all data generated from the generator the evaluation gives a complete representation of the performance. For this reason, the original TSTR and TRTS as presented by Hyland et al. will be used in this project.

In addition to the TSTR and TRTS methods, a third method is used to evaluate the data. The purpose of the GAN is to generate data that can be added to the real data as an augmentation method, and this way increase the performance of a classifier. In order to simulate this, two classifiers are trained separately, one on only real data and one on a mixture of real and generated data, and the accuracies are compared. This evaluation method will be referred to as Train on Mixed, Test on Real (TMTR). Figure 3.7 visualises the differences between these two classifiers. The first of the two classifiers are trained on a smaller part of the GANs training set. Testing is done using a separate test set containing only real samples that haven't been seen by the GAN. This classifier will be used as a baseline for measuring the quality of the generated data. The other classifier is trained on a data set that contains a number of real samples from the GANs training set and equally many generated samples. This classifier is evaluated using the same test set as for the first classifier. The accuracies from these two classifiers are compared, and if the generated data does represent the original data the second classifier will outperform the first one due to the increase in training samples. If the entire training set with real was to be used when training the classifiers they would have a very high accuracy and it would not be possible to improve performance by adding more samples and thus not providing any information about the quality of the generated data.

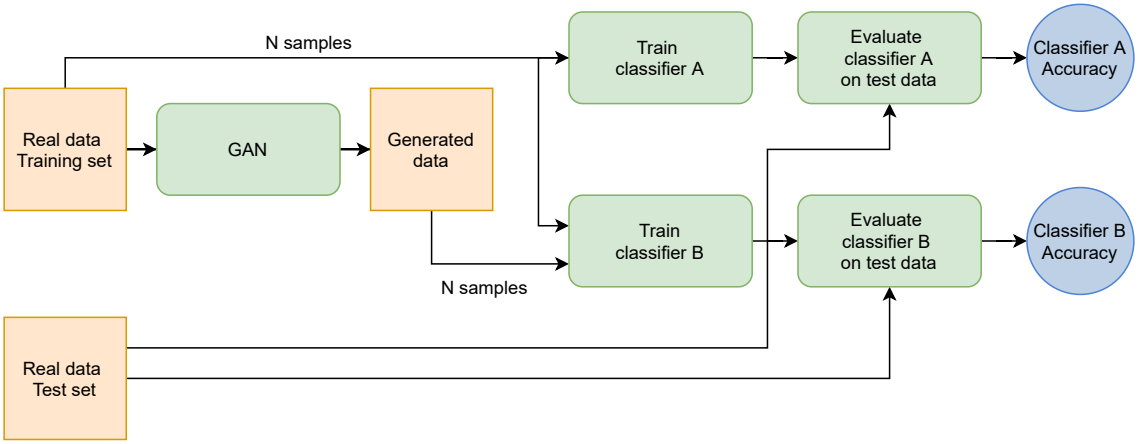


Figure 3.7: Classifier A is trained on a small set of real data. Classifier B is trained on a small set of real data combined with as many samples of generated data. Both classifiers are evaluated on the same test set. If the generated data is of high quality, classifier B has a higher accuracy than classifier A.

The classifier used in all evaluations is a simple multilayer perception with two hidden layers with 100 neurons each. Both hidden layers have a rectified linear unit as the activation function. Stochastic gradient descent is used as an optimiser and the loss function is a binary cross-entropy function.

3.4.2 Interpreting results from evaluation methods

The choice of evaluation methods is of great importance to give an accurate and complete picture of the performance of the GAN. Since there are no established methods available to evaluate GANs trained on non-image data, multiple evaluation

techniques have been used in this project to ensure that all aspects of the data are captured in some way. None of the evaluation techniques gives a complete picture on their own but needs to be combined with other methods. Below, the three methods TSTR, TRTS and TMTR will be discussed with regards to their advantages and shortages and how to interpret different results.

TSTR

The TSTR method aims to evaluate if the generated data spans the entire set of real data. If the GAN is trained optimally and the generated data fully represents the real data, then the TSTR accuracy should be equal to the accuracy obtained when training and testing on the same number of real samples. When the TSTR accuracy is lower than when training on real data, the GAN has not fully captured all features correctly or the entire distribution is not represented in the generated data. As stated in Section 3.4, TSTR will give a poor accuracy if mode collapse occurs during training. For the generated data to be used as intended to train a classifier, it is necessary that it does not introduce an unexpected bias by only representing the original distribution partially. Due to this, TSTR is a highly relevant evaluation method to use when the aim is to use the generated data as an augmentation method.

It is not possible to tell if the GAN has overfitted from the TSTR accuracy. If the GAN is largely overfitted and the generator creates copies of the training set, then the TSTR accuracy will be as high as the TRTR accuracy. Due to this, it is important to ensure that the generated data is not copies of the training set with other methods, as described in Section 3.3.

TRTS

This method can be used to measure the quality of the generated samples and the generators ability to create samples that are well separated into two classes. Ideally, the TRTS accuracy would be equal to the TRTR accuracy since it would imply that classification is as easy or difficult with both generated and original data. A TRTS accuracy that is lower than TRTR would indicate that some key feature that the classifier uses to separate the two classes is missing. If the TRTS accuracy is higher than the TRTR accuracy, however, then the generated data is simplified in some way. That could happen if the GAN gets too focused on a few features that are important for classifying and creates samples that have these features well represented and somewhat exaggerated rather than generating a sample that includes a multitude of different features.

TRTS cannot give any information about the overall distribution of the generated samples. If the GAN has suffered mode collapse and the generated data set consists of only one, or a few, samples repeated over and over again, then a high accuracy will still be obtained as long as this sample is of good quality. Fortunately, it is easy to spot if mode collapse has happened by plotting a few samples and comparing them to one another. Due to this, a combination of TSTR and TRTS will manage to evaluate both the distribution and quality of the generated data. Note that it is

still important to avoid overfitting.

TMTR

This evaluation method is a direct simulation of how the generated data is meant to be used outside of this project. In a pragmatic sense, the training of a GAN is successful as long as there is some increase in accuracy when adding the generated data. If the data is of poor quality or does not represent both classes accurately, then the TMTR accuracy will not be higher than the accuracy obtained when training on only the original samples. The TRTR accuracy where the training set is made up of as many samples as the combined training set for TMTR gives an indication of the expected maximum increase. An advantage of using TMTR is that the accuracy will only increase when adding generated samples if these samples are different from the original data. Therefore, TMTR will not give any performance if the GAN is overfitted and the generated samples are copies of the training data.

If the training set of real data is big, the TRTR accuracy will be very high even before the addition of generated data. For some size of the training set, the classifier will saturate and its performance will not increase even if more data is added. This makes the possibilities for improvement by adding the generated samples limited when starting with a large set of original data. Due to this, the number of original samples used in training has to be reduced until the classifier accuracy is low enough for there to be room for improvement. Whilst reducing the size of the training set, the aim was to find a balance between a relatively low amount of data to clearly see improvements but still keep the standard deviation across classifier trainings low. Using a smaller set of real samples makes the TMTR evaluation method lose the advantage of being able to catch overfitting. If the entire training data set was used, eventual copies in the generated data set would definitely already be represented in the combined set and therefore not contribute when training. When only a subset of the original training data is used, it is possible that added generated samples brings new information to the training of the classifier even though it is actually a copy of an original sample since that specific original sample might not be included in the subset.

4

Results

The results from training the different kinds of GANs as described in Section 3.3, when evaluated using TSTR, TRTS and TMTR as described in Section 3.4 are presented here. The results are split into a first part concerning the simulation data set, and a second part for the phantom data set. For each type of data, results have been obtained for four different kinds of GANs. The first is a CWGAN-GP, the second is a CWGAN-GP combined with DeLiGAN which will be referred to as CWGAN-GP + DeLiGAN. The third is a CWGAN-GP with noise added to the original samples during training, referred to as CWGAN-GP + noise. Lastly, CWGAN-GP + DeLiGAN + noise is a combination of the previous two.

Each evaluation method is benchmarked against a TRTR classifier. Ideally, the TSTR, TRTS and TMTR classifier has an accuracy on par with the TRTR classifier. Using two different benchmark classifiers for TSTR and TRTS is necessary because they are trained on a different number of samples, and the performance of the classifier is largely affected by this. Furthermore, TSTR is also compared to a classifier trained on only generated data and tested on generated data (TSTS). The TSTS accuracy is also equal to the TRTR accuracy if the generated data is ideal. The purpose of TMTR is however to demonstrate an eventual increase in accuracy by adding the generated data to the original. Therefore, this is compared to both a classifier trained on only the original samples from the mixed data set, and a classifier trained on a proportional increase of only original data. However, if the accuracy on the original data is already very high or the classifier saturates at some accuracy level, it can be hard to note the effect of the addition of generated data. Therefore, the entire training set is not used for TMTR, and the number of original and generated samples for the simulated and phantom data set will be presented in the separate sections below.

When training the GAN multiple times, variations occur due to the random nature of the training and the random splitting of the data sets. On top of that, variations occur during evaluation when the classifiers are trained. Because of this, the results that follow are presented with a mean and a standard deviation. For each type of GAN training for which results are presented, the GAN has been trained three times. Each training is then evaluated 10 times, which in total yields thirty values for each kind of GAN. The mean and standard deviation for a specific kind of GAN is then computed from these thirty values. Note that the standard deviation then

accounts for the variations in both the training of the GAN and for the evaluation methods. The mean and standard deviation on each of the three runs have also been computed separately for a CWGAN-GP, and this is presented in Tables A.1, A.2, A.3 and A.4 in the appendix.

Results are visualised by plotting a chosen channel of randomly selected generated data that has been matched to original data using the summed square error as a distance measure, as done in Figure 4.1. For each kind of GAN, an upscaled plot is shown first to demonstrate the noise levels in the generated data. This is followed by smaller sized plots to demonstrate the variability in the data and how well it manages to match the original. The summed square error can be computed either for only the chosen channel or the entire sample. The match on a single channel is done by randomly selecting a generated sample and then computing the summed square error for that channel for all original samples of the same class. This demonstrates how well the GAN has learnt the behaviour of the specific channel. When matching on an entire sample, the randomly selected generated sample is compared to all original samples of the same class. Now, the summed square error is computed individually for all channels and the summed square errors are then added together. The least total summed error determines the match. This way of matching demonstrates whether the GAN has pieced the channels together in a similar way as the original data. Both of the above methods are demonstrated for all four kinds of GANs, note however that even though the match has been computed for the entire sample - only one channel is visualised for simplicity.

Further visualisations are however required to investigate to what extent the generated data manages to represent the distribution of the original data. Therefore, the mean and standard deviation for a chosen channel is computed across samples. It is computed separately for healthy and bleeding samples and is overlayed with the same computations for generated data. In order to also consider the distribution of entire samples, principal component analysis (PCA) has been performed. Plots of the first and second principal components are shown for each kind of GAN, coloured according to class and data type. An example of these visualisations is found in Figure 4.11.

Lastly, the plots of the critic output are included. As described in Section 3.3, these are used to determine the extent of overfitting over the course of training a GAN. Since each kind of GAN is trained three separate times, the plots shown in this section are a chosen example for each kind of GAN. This is also the case for all the visualisations described and has been deemed reasonable due to the great similarity between separate trainings of the same kind of GAN.

4.1 Results on simulated data

Table 4.1 shows the accuracy on TSTR and TRTS, and Table 4.2 shows the performance on TMTR for simulated data. TSTR and TRTS are benchmarked against TRTR and TSTS as described above. TMTR is computed from 300 original samples

and 300 generated samples, and this is compared to both the accuracy of a classifier trained on only 300 original samples and one trained on 600 original samples.

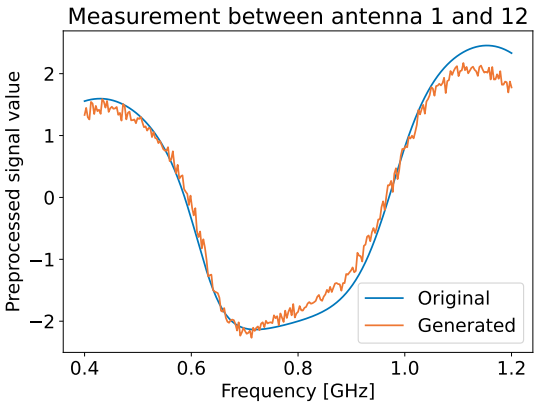
Figure 4.1, 4.3, 4.5 and 4.7 show generated samples of the channel representing measurements between antenna 1 and 12 for CWGAN-GP, CWGAN-GP + DeLiGAN, CWGAN-GP + noise and CWGAN-GP + DeLiGAN + noise respectively. The mean and standard deviation across samples for the same channel together with PCA for the entire samples are shown in Figure 4.2, 4.4, 4.6 and 4.8 for the four different kinds of GANs. Examples of critic outputs for all the different kinds of GAN are shown in Figure 4.9.

Table 4.1: Accuracy of four kinds of GANs trained on simulated data, using TSTR and TRTS. Results are presented as a mean and standard deviation computed from three separate trainings of each kind of GAN, each evaluated 10 times. In the ideal case, the accuracies of TSTR, TSTS and TRTS are equal to the respective accuracy of the TRTR classifiers.

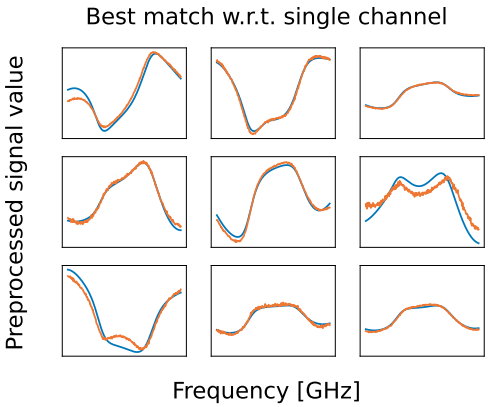
	TRTR	TSTS	TSTR	TRTR	TRTS
Number of training samples for classifier	3027	3027	3027	520	520
CWGAN-GP	0.89 ±0.016	0.98 ± 0.011	0.86 ± 0.013	0.78 ± 0.036	0.74 ± 0.034
CWGAN-GP + DeLiGAN	0.91 ± 0.013	0.995 ± 0.0031	0.80 ± 0.020	0.79 ± 0.020	0.72 ± 0.021
CWGAN-GP + noise	0.91 ± 0.019	0.979 ± 0.0080	0.88 ± 0.019	0.79 ± 0.021	0.75 ± 0.021
CWGAN-GP + DeLiGAN + noise	0.91 ± 0.012	0.994 ± 0.0042	0.84 ± 0.019	0.79 ± 0.012	0.74 ± 0.014

Table 4.2: Accuracy of four kinds of GANs trained on simulated data, using TMTR. Results are presented as a mean and standard deviation computed from three separate trainings of each kind of GAN, each evaluated 10 times. With ideally generated data, the accuracy of the classifier trained on the mixed data set is equal to the accuracy of the classifier trained on as many original samples.

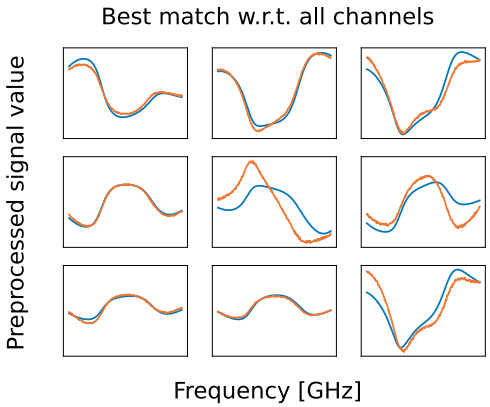
	TMTR		
Training data for classifier	300 original	300 original + 300 generated	600 original
CWGAN-GP	0.73 ± 0.040	0.79 ± 0.033	0.80 ±0.027
CWGAN-GP + DeLiGAN	0.75 ± 0.030	0.81 ± 0.026	0.82 ± 0.022
CWGAN-GP + noise	0.74 ± 0.040	0.81 ± 0.019	0.81 ±0.021
CWGAN-GP + DeLiGAN + noise	0.72 ± 0.060	0.82 ± 0.023	0.80 ±0.034



(a) CWGAN-GP

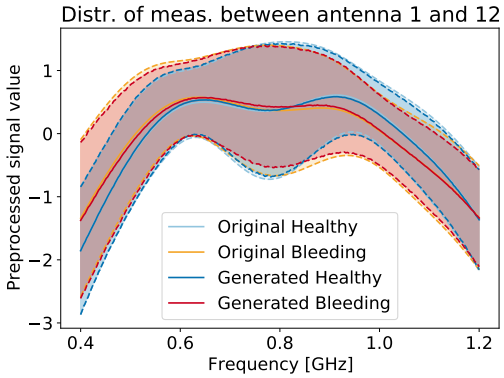


(b) CWGAN-GP

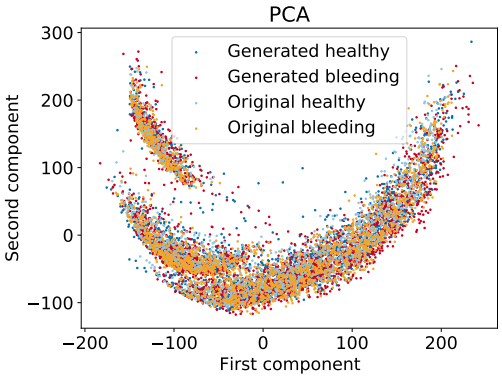


(c) CWGAN-GP

Figure 4.1: Randomly selected generated samples from CWGAN-GP on simulated data, matched to original samples. Figure 4.1a and 4.1b show a match to the closest real sample with respect to the channel for antenna 1 and 12. Figure 4.1c also shows this channel, but the match is computed on an entire sample.

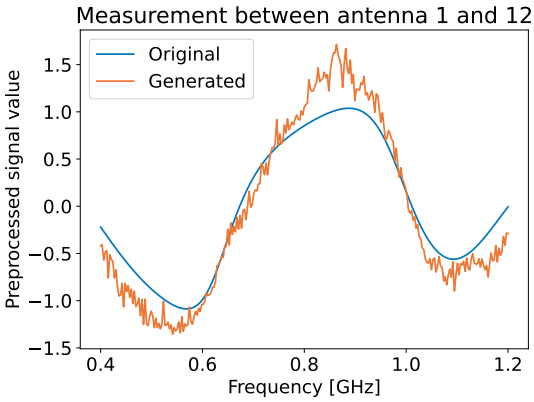


(a) CWGAN-GP

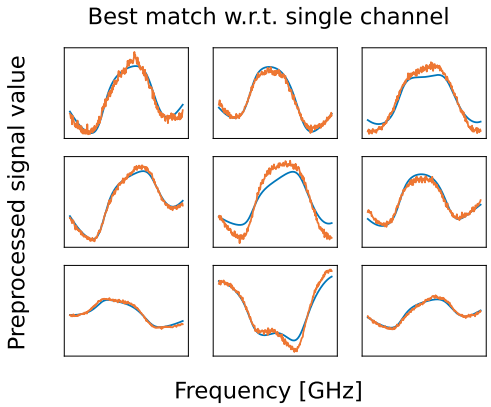


(b) CWGAN-GP

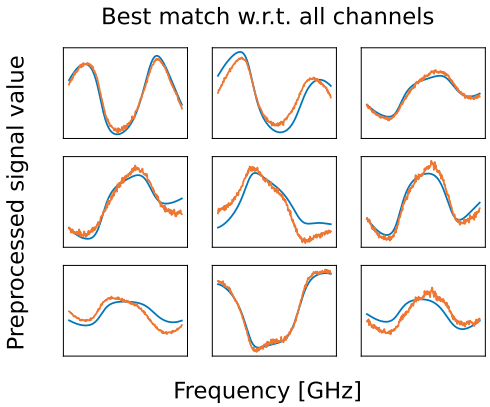
Figure 4.2: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.2a and PCA in Figure 4.2b is used to compare the distribution of generated data to the distribution of the original data.



(a) CWGAN-GP + DeLiGAN

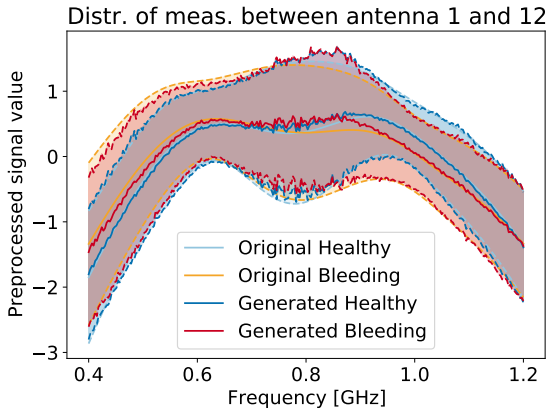


(b) CWGAN-GP + DeLiGAN

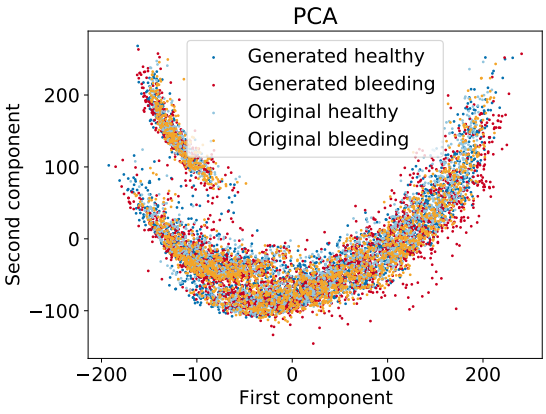


(c) CWGAN-GP + DeLiGAN

Figure 4.3: Randomly selected generated samples from CWGAN-GP + DeLiGAN on simulated data, matched to original samples. Figure 4.3a and 4.3b show a match to the closest real sample with respect to the channel for antenna 1 and 12. Figure 4.3c also shows this channel, but the match is computed on an entire sample.

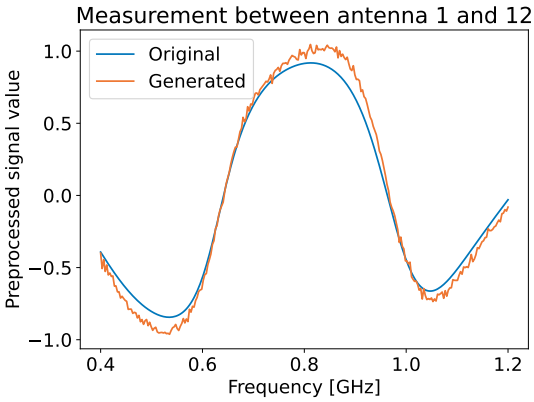


(a) CWGAN-GP + DeLiGAN

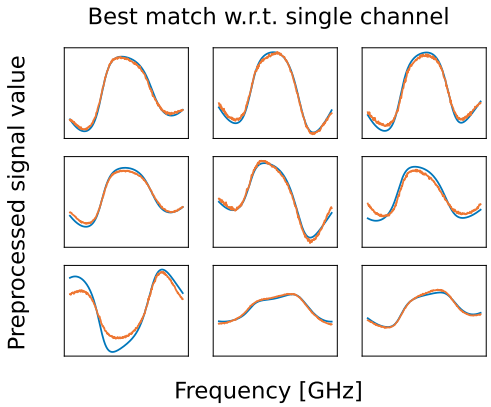


(b) CWGAN-GP + DeLiGAN

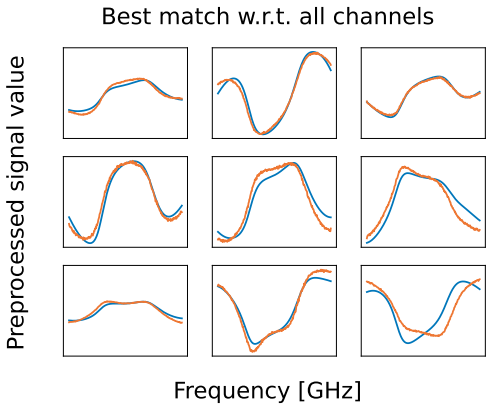
Figure 4.4: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.4a and PCA in Figure 4.4b is used to compare the distribution of generated data to the distribution of the original data.



(a) CWGAN-GP + noise

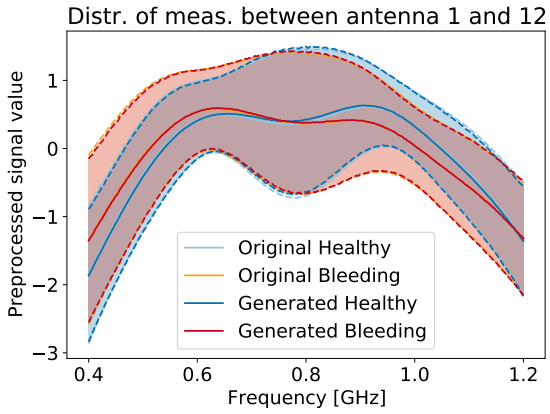


(b) CWGAN-GP + noise

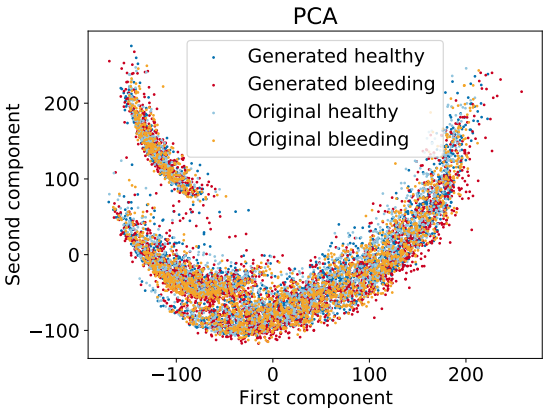


(c) CWGAN-GP + noise

Figure 4.5: Randomly selected generated samples from CWGAN-GP + noise on simulated data, matched to original samples. Figure 4.5a and 4.5b show a match to the closest real sample with respect to the channel for antenna 1 and 12. Figure 4.5c also shows this channel, but the match is computed on an entire sample.

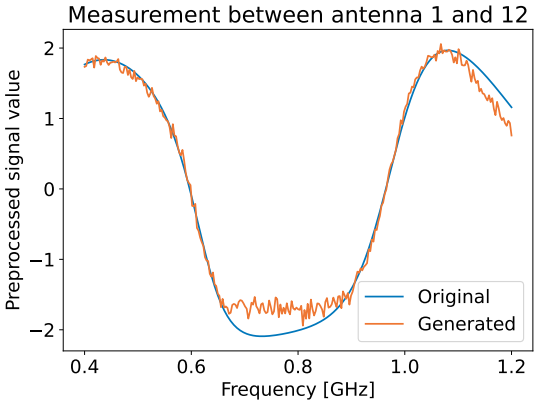


(a) CWGAN-GP + noise

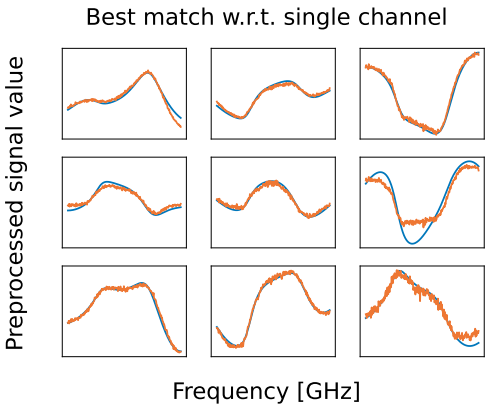


(b) CWGAN-GP + noise

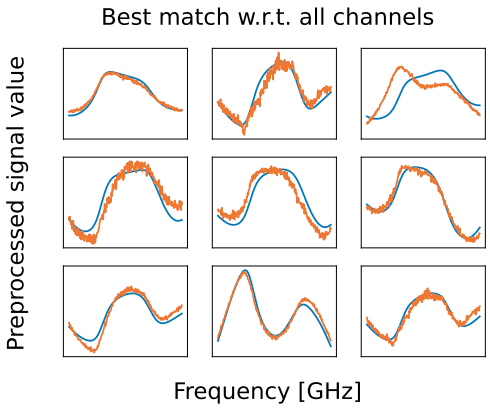
Figure 4.6: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.6a and PCA in Figure 4.6b is used to compare the distribution of generated data to the distribution of the original data.



(a) CWGAN-GP + DeLiGAN + noise

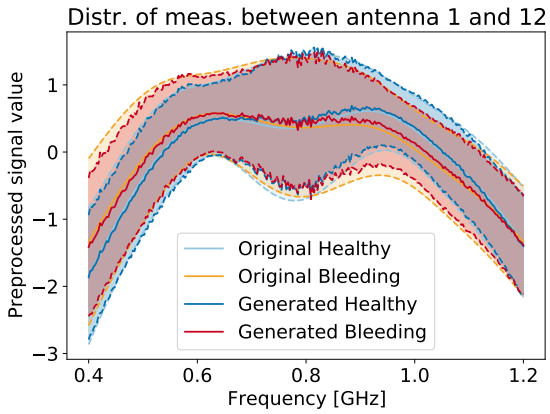


(b) CWGAN-GP + DeLiGAN + noise

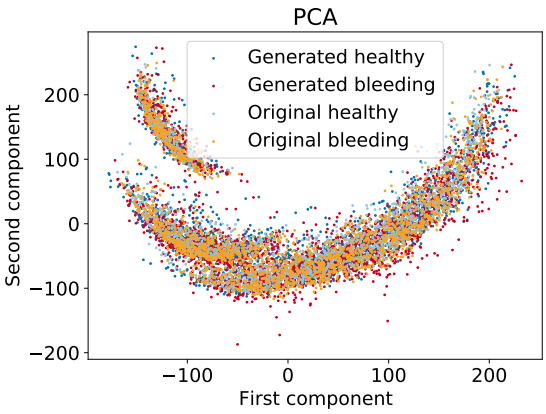


(c) CWGAN-GP + DeLiGAN + noise

Figure 4.7: Randomly selected generated samples from CWGAN-GP + DeLiGAN + noise on simulated data, matched to original samples. Figure 4.7a and 4.7b show a match to the closest real sample with respect to the channel for antenna 1 and 12. Figure 4.7c also shows this channel, but the match is computed on an entire sample.

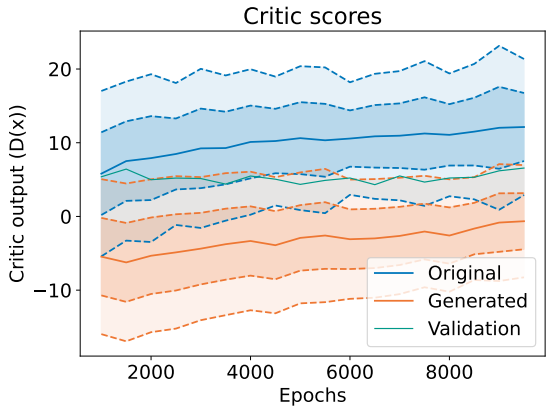


(a) CWGAN-GP + DeLiGAN + noise

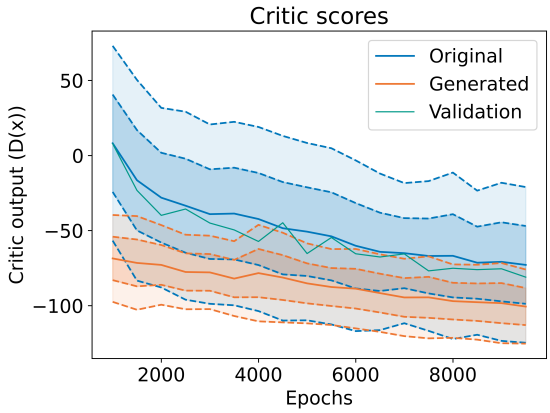


(b) CWGAN-GP + DeLiGAN + noise

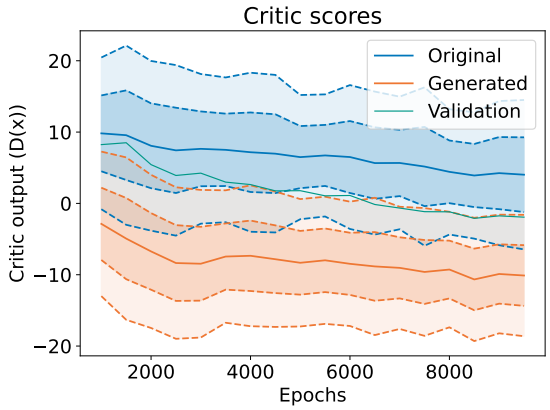
Figure 4.8: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.8a and PCA in Figure 4.8b is used to compare the distribution of generated data to the distribution of the original data.



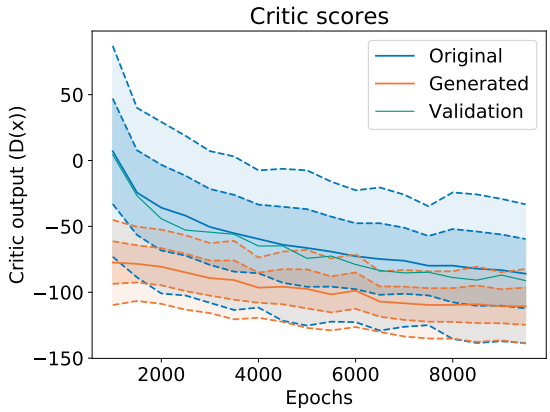
(a) CWGAN-GP



(b) CWGAN-GP + DeLiGAN



(c) CWGAN-GP + noise



(d) CWGAN-GP + DeLiGAN + noise

Figure 4.9: Examples of critic output for CWGAN-GP 4.9a, CWGAN-GP + DeLiGAN 4.9b, CWGAN-GP + noise 4.9c and CWGAN-GP + DeLiGAN + noise 4.9d over the course of 10000 epochs using simulated data. The outputs of original and generated samples are compared to a validation set of unseen original data.

4.2 Results on phantom data

Table 4.3shows the accuracy on TSTR and TRTS, and Table 4.4shows the accuracy on TMTR for phantom data. TSTR and TRTS are benchmarked against the accuracy on TRTR and TSTS for the same data set size, as described previously. TMTR is computed for 600 original and 600 generated samples, which is compared to TRTR for both 600 and 1200 samples. A larger subset of the training data for the phantom data set compared to the simulated data is used because of the increased complexity of the phantom data.

Figure 4.10, 4.12, 4.14 and 4.16 show generated samples of the channel representing measurements between antenna 3 and 5 for CWGAN-GP, CWGAN-GP + DeLiGAN, CWGAN-GP + noise and CWGAN-GP + DeLiGAN + noise respectively. The mean and standard deviation across samples for the same channel together with PCA for the entire samples are shown in Figure 4.11, 4.13, 4.15 and 4.17 for the

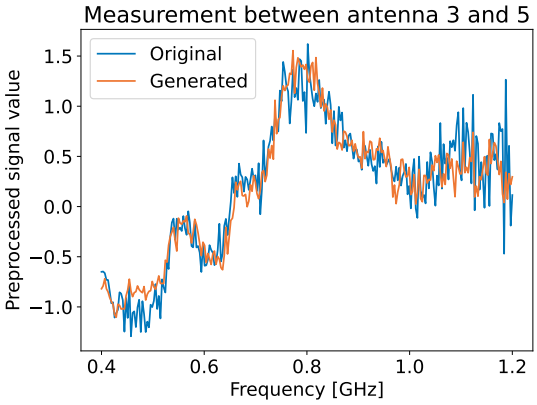
four different kinds of GANs. Examples of critic outputs for the four kinds of GAN are shown in Figure 4.18.

Table 4.3: Accuracy of four kinds of GANs trained on simulated data, using TSTR and TRTS. Results are presented as a mean and standard deviation computed from three separate trainings of each kind of GAN, each evaluated 10 times. In the ideal case, the accuracies of TSTR, TSTS and TRTS is equal to the respective accuracy of the TRTR classifiers.

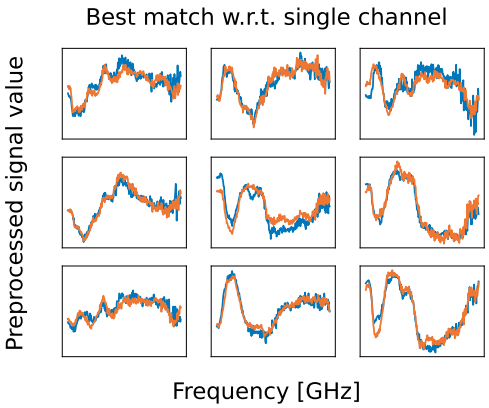
	TRTR	TSTS	TSTR	TRTR	TRTS
Number of training samples for classifier	2266	2266	2266	339	339
CWGAN-GP	0.67 ±0.027	0.65 ± 0.048	0.59 ± 0.036	0.57 ± 0.024	0.53 ± 0.011
CWGAN-GP + DeLiGAN	0.69 ± 0.030	0.8 ± 0.14	0.57 ± 0.031	0.55 ± 0.025	0.54 ± 0.024
CWGAN-GP + noise	0.67 ± 0.030	0.67 ± 0.038	0.61 ± 0.026	0.53 ± 0.022	0.51 ± 0.013
CWGAN-GP + DeLiGAN + noise	0.66 ± 0.026	0.88 ± 0.050	0.60 ± 0.035	0.53 ± 0.031	0.54 ± 0.027

Table 4.4: Accuracy of four kinds of GANs trained on simulated data, using TMTR. Results are presented as a mean and standard deviation computed from three separate trainings of each kind of GAN, each evaluated 10 times. With ideally generated data, the accuracy of the classifier trained on the mixed data set is equal to the accuracy of the classifier trained on as many original samples.

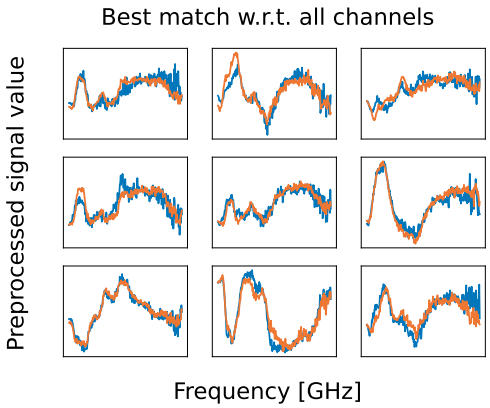
	TMTR		
Training data for classifier	600 original	600 original +600 generated	1200 original
CWGAN-GP	0.59 ± 0.045	0.60 ± 0.045	0.65 ± 0.030
CWGAN-GP + DeLiGAN	0.59 ± 0.044	0.59 ± 0.030	0.65 ± 0.043
CWGAN-GP + noise	0.58 ± 0.048	0.60 ± 0.037	0.65 ± 0.036
CWGAN-GP + DeLiGAN + noise	0.57 ± 0.038	0.60 ± 0.031	0.64 ± 0.036



(a) CWGAN-GP

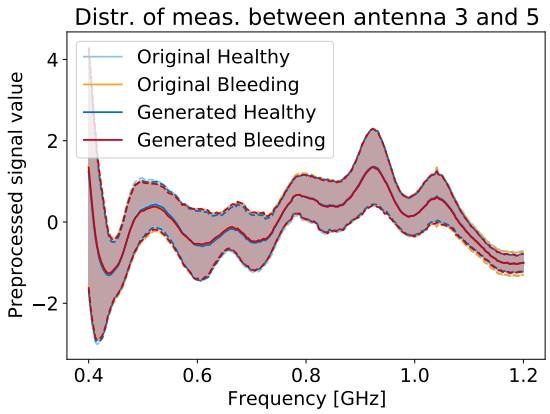


(b) CWGAN-GP

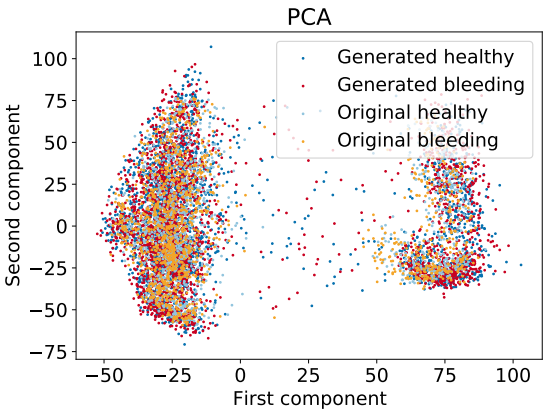


(c) CWGAN-GP

Figure 4.10: Randomly selected generated samples from CWGAN-GP on phantom data, matched to original samples. Figure 4.10a and 4.10b show a match to the closest real sample with respect to the channel for antenna 3 and 5. Figure 4.10c also shows this channel, but the match is computed on an entire sample.

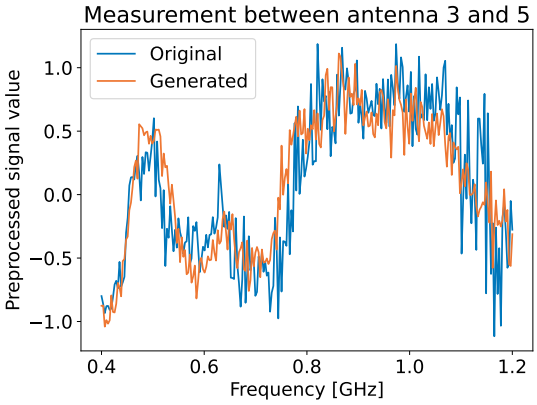


(a) CWGAN-GP

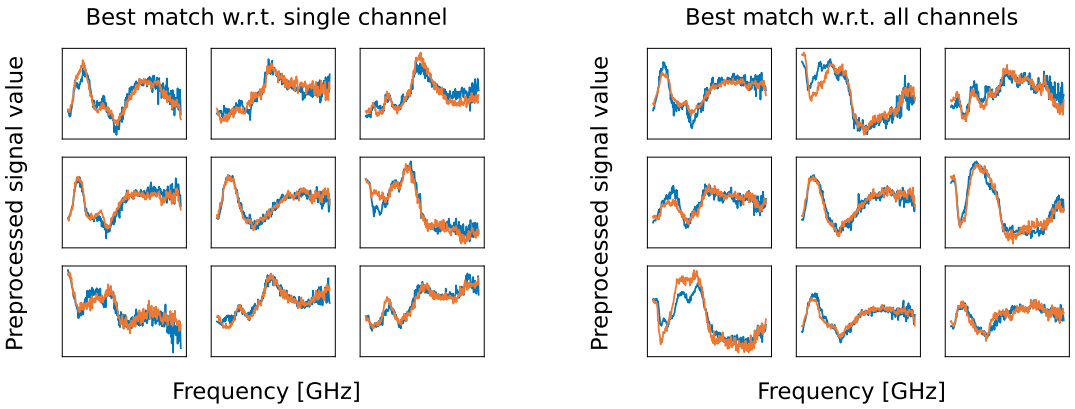


(b) CWGAN-GP

Figure 4.11: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.11a and PCA in Figure 4.11b is used to compare the distribution of generated data to the distribution of the original data.



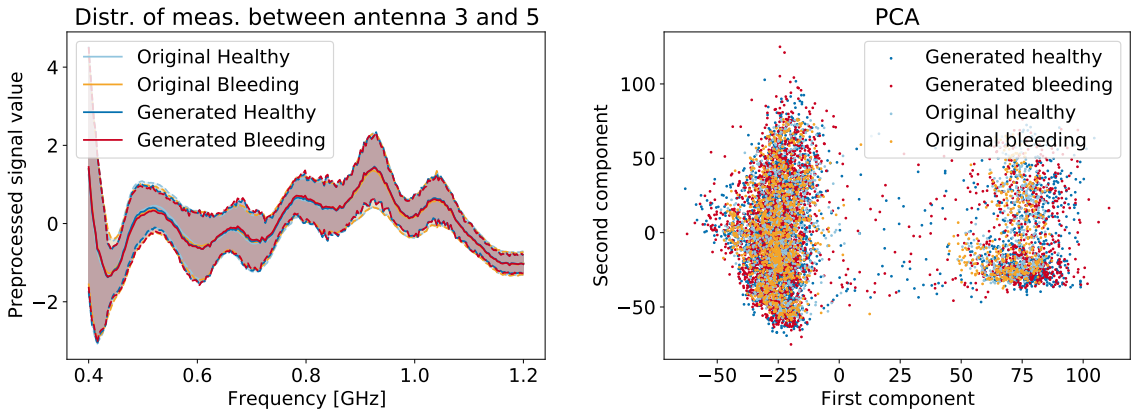
(a) CWGAN-GP + DeLiGAN



(b) CWGAN-GP + DeLiGAN

(c) CWGAN-GP + DeLiGAN

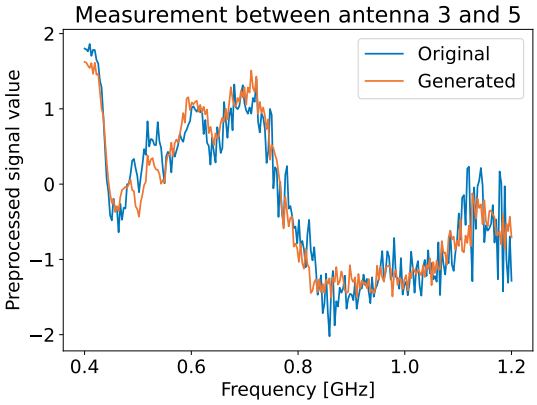
Figure 4.12: Randomly selected generated samples from CWGAN-GP +DeLiGAN on phantom data, matched to original samples. Figure 4.12a and 4.12b show a match to the closest real sample with respect to the channel for antenna 3 and 5. Figure 4.12c also shows this channel, but the match is computed on an entire sample.



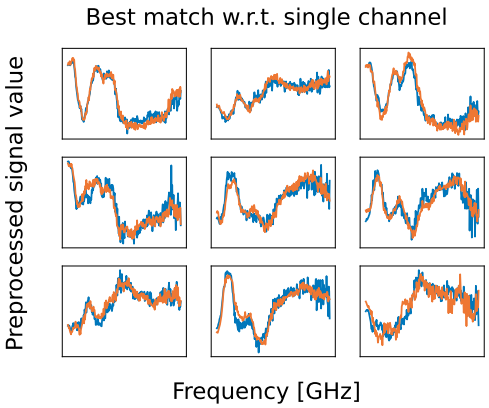
(a) CWGAN-GP + DeLiGAN

(b) CWGAN-GP + DeLiGAN

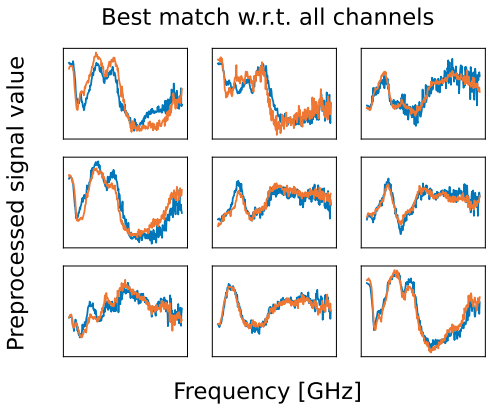
Figure 4.13: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.13a and PCA in Figure 4.13b is used to compare the distribution of generated data to the distribution of the original data.



(a) CWGAN-GP + noise

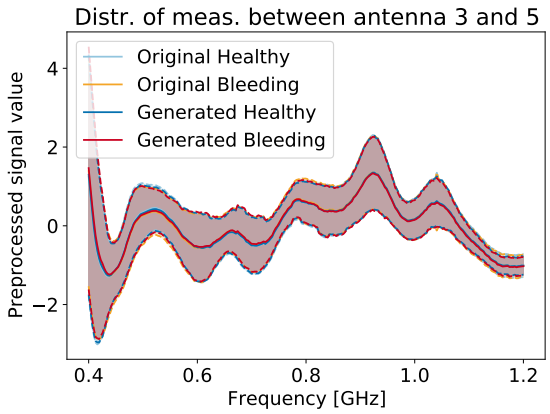


(b) CWGAN-GP + noise

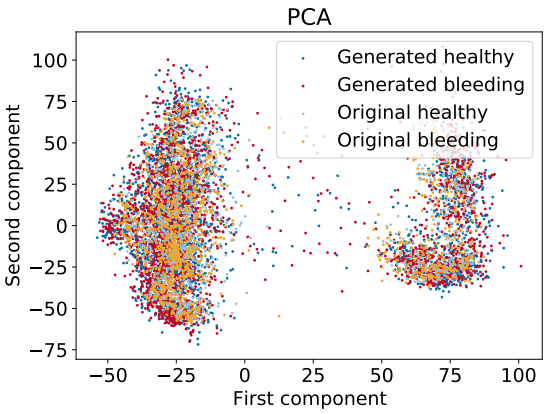


(c) CWGAN-GP + noise

Figure 4.14: Randomly selected generated samples from CWGAN-GP + noise on phantom data, matched to original samples. Figure 4.14a and 4.14b show a match to the closest real sample with respect to the channel for antenna 3 and 5. Figure 4.14c also shows this channel, but the match is computed on an entire sample.

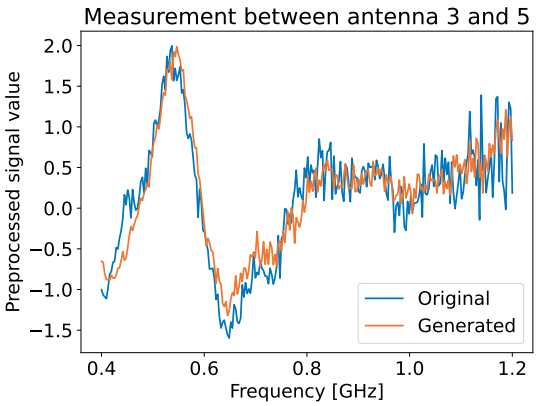


(a) CWGAN-GP + noise

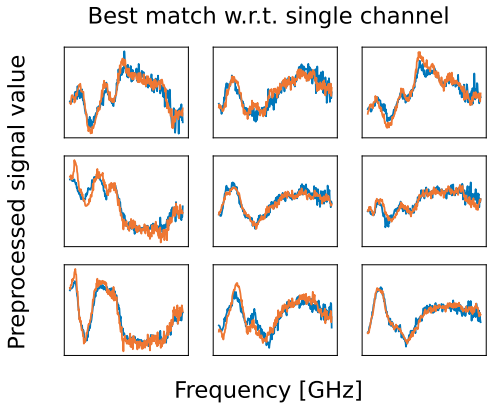


(b) CWGAN-GP + noise

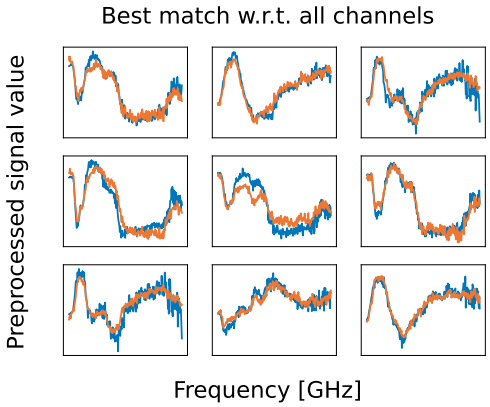
Figure 4.15: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.15a and PCA in Figure 4.15b is used to compare the distribution of generated data to the distribution of the original data.



(a) CWGAN-GP + DeLiGAN + noise

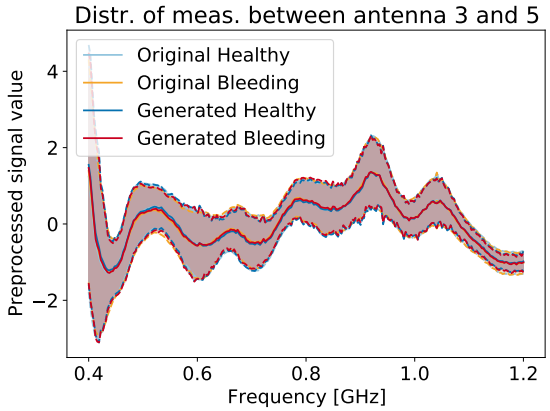


(b) CWGAN-GP + DeLiGAN + noise

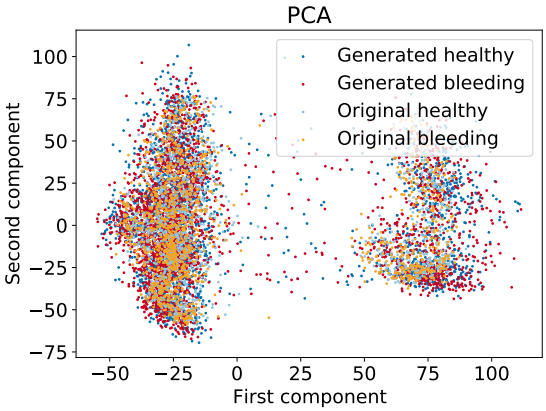


(c) CWGAN-GP + DeLiGAN + noise

Figure 4.16: Randomly selected generated samples from CWGAN-GP + DeLiGAN + noise on phantom data, matched to original samples. Figure 4.16a and 4.16b show a match to the closest real sample with respect to the channel for antenna 3 and 5. Figure 4.16c also shows this channel, but the match is done on an entire sample.

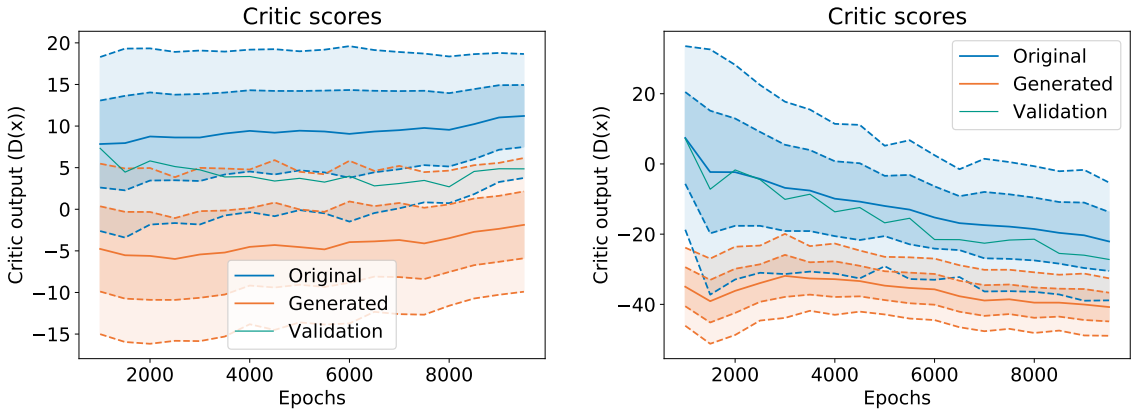


(a) CWGAN-GP + DeLiGAN + noise



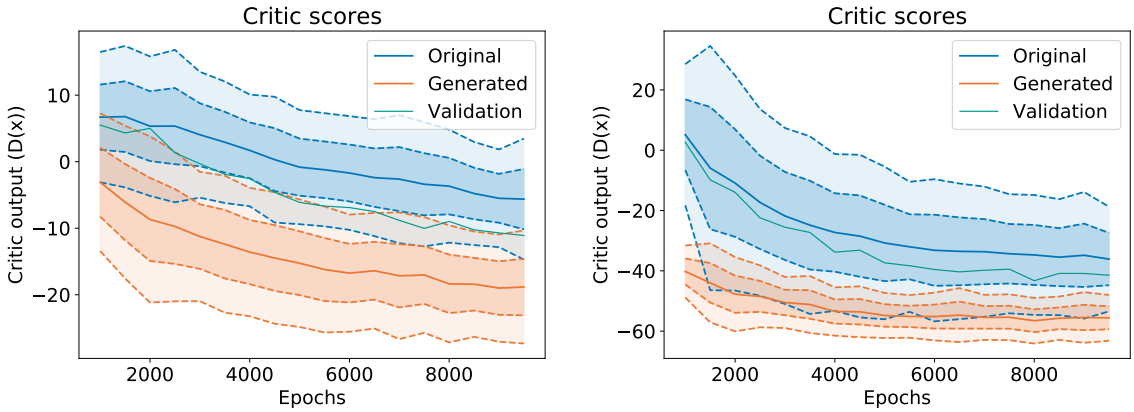
(b) CWGAN-GP + DeLiGAN + noise

Figure 4.17: The mean value (full line) and one standard deviation from the mean (dashed line) across samples in Figure 4.17a and PCA in Figure 4.17b is used to compare the distribution of generated data to the distribution of the original data.



(a) CWGAN-GP

(b) CWGAN-GP + DeLiGAN



(c) CWGAN-GP + noise

(d) CWGAN-GP + DeLiGAN + noise

Figure 4.18: Examples of critic output for CWGAN-GP (4.9a), CWGAN-GP + DeLiGAN (4.9b), CWGAN-GP + noise (4.9c), and CWGAN-GP + DeLiGAN + noise (4.9d) over the course of 10000 epochs using phantom data. The outputs of original and generated samples are compared to a validation set of unseen original data.

5

Discussion

The results that are presented in Chapter 4 are a combined measure of how successful the GAN models and the training of them were, and to what extent the used evaluation methods gives an accurate picture of the generated data. Therefore, it is not obvious to what amount the different factors have affected the results. This section aims to interpret the obtained results as well as give probable explanations for unexpected behaviours. Finally, suggestions for improving and continuing this work is discussed as well.

5.1 Interpretations of results

The obtained results as presented in Chapter 4 will be discussed by comparing the performance during evaluation with visualisations of the generated data. The evaluation methods will be interpreted according to Section 3.4.2.

5.1.1 Simulated data

The simulated data set is the easier of the two, which is clearly seen in the performance level on original data. A GAN trained on this data also reaches relatively good levels of performance on all three evaluation methods. The differences between the four kinds of GANs are small, but CWGAN-GP + noise scores the highest on both TSTR and TRTS as seen in Table 4.1 while CWGAN-GP + DeLiGAN + noise scores the highest on TMTR. Worth noting is however that the differences between performance on mixed data and only original data for all four kinds of GANs are small enough to arise due to the randomness in the training of a neural network, both during training of the GAN and training of classifiers in evaluation. Furthermore, the random splitting of the data into train and test sets can also affect the performance to a certain extent. A key difference between generated and original data is however the level of noise. The original data is completely free of noise, while all GANs generate data with more or less prominent levels. To what extent this affects the performance of generated data is at the moment uncertain.

An important aspect to consider in the context of performance is the extent of overfitting that has occurred. From Figure 4.9 it is clear that the addition of DeLiGAN to CWGAN-GP is an efficient method to counteract this. This relates well to the

purpose of DeLiGAN, which is to generate diverse samples even for small data sets. However, this GAN also performs the worst in evaluation, and it seems that despite the diversity among generated samples, the quality is somehow lacking. An increase in performance is instead gained by the addition of noise in training, but for this GAN the extent of overfitting is still an issue that needs to be considered. Possible effects of the partial overfitting that is being observed, and why this is undesirable are discussed more in-depth in Section 5.2. To solve the issue of overfitting, and still keep high levels of performance, a combination of CWGAN-GP + DeLiGAN and the addition of noise was considered interesting. The result is a GAN that does not overfit due to the diversity generated by DeLiGAN as demonstrated in Figure 4.9d, and a good representation of samples by the aid of noise which results in high-performance levels in Tables 4.2 and 4.2. Even though the performance on TSTR and TRTS is slightly lower for CWGAN-GP + DeLiGAN + noise than CWGAN-GP + noise, it can be argued that the former is better still due to less overfitting.

Distribution coverage

For simulated data, the channel corresponding to measurements between antenna 1 and 12 was chosen for visualisation due to its large variance across samples. Despite this, all four kinds of GAN seem to match the distribution represented by mean and standard deviation on this channel to a relatively large extent, as can be seen in Figures 4.2a, 4.4a, 4.6a and 4.8a. Out ff the four, CWGAN-GP + DeLiGAN shows the highest levels of noise, while the addition of noise to CWGAN-GP displays an almost perfect match. The combination of the two still exhibits large levels of noise, but as already stated it reaches higher levels of performance on all evaluation methods than CWGAN-GP + DeLiGAN, without increasing overfitting.

The PCA plots in Figure 4.2b, 4.4b, 4.6b and 4.8 also demonstrate an overlap of generated samples and original samples. Furthermore, generated samples do not seem to be placed in the exact same position as the original samples. This indicates that the generator has learnt the general distribution of the original data, but also introduces new information in the form of variability. The advantage of these is that they take the entire samples into consideration, not just a specific channel. However, the largest variance does not seem to be between the two classes since there is no visible separation between them for the first and second principal component. For the data to be useful, it is crucial that the generator has learnt the class-specific features. In order to investigate this, one would need to investigate for which principal components the separation of classes occur, and how well this separation is represented by the generated data. Furthermore, since the class separation is not visible for the first and second component, it is likely that other behaviours in the data are hidden among higher-order components as well. For instance, there could be a shift between the original and the generated distribution that is visible for other principal components. Such a shift could indicate that the channels are not combined in a sensible way. With an irrational combination of channels, key features for separate classes could be left out, or the classifier could rely too much on specific channels.

Even though there is no separation of classes, there is clustering in the data with respect to unknown features. Of more interest in the perspective of data generation, is that the majority of outliers are generated samples. Depending on what feature causes the clustering, this could be both a positive and negative result. One possibility is that the outliers simply represent something that is physically non-existent in reality, and the generator has failed to fit the appropriate distribution of samples. Another possibility is however that these samples are filling gaps in the original distribution. Imagine if there was clustering based on the bleeding position of the simulated patient, and only a low number of specific positions exist in the simulated data set. Then samples that are placed between these clusters are filling a gap in the distribution that has a real equivalent. In order to make any conclusions on the matter, further investigation is required regarding the effect of these outliers on performance, and what might be causing the clustering.

A closer look on performance metrics

Looking at Table 4.1, the overall performance on TSTS is higher than the corresponding performance on TRTR. Meanwhile, TSTR never reaches the same level of performance as TRTR. This indicates that the generator manages to create two different classes with class-specific features, but the distinction between the two classes is not the same as for original data, which means that the generated data is not a perfect match to the original distribution. Such a mismatch could arise if the generator for instance learns some feature that has a high correlation to bleeding samples, but that feature is not actually used by the classifier for classification.

Another explanation is that the generated data is a simplification of the original data and therefore misses out on certain behaviours in the original data, which would explain why most original samples are still classified correctly in TSTR. This is also supported by the increase in TSTS for GANs with low TSTR, and a decrease in TSTS when TSTR performance increases, although the differences in Table 4.1 are still small enough to be due to randomness. However, one such simplification would be a mode collapse of the generator to a limited number of types of samples. From looking at the variety of samples in Figure 4.1, 4.3, 4.5 and 4.7, one can see that there are different kinds of samples in the generated data, and therefore it seems that the generator hasn't suffered mode collapse. Even if there is no mode collapse, the generator could still miss out on certain parts of the original distribution, without collapsing to produce copies of some samples over and over again. This is however contradicted by the distributions plotted using the first two PCA components, since all parts of the original distribution are covered by the generated data. Instead, the outliers are from the generated data. That would rather indicate that the generated data will train a classifier for a wider diversity of samples. As stated earlier regarding class separation for PCA plots, it is required to investigate the distributions further to capture the class-specific behaviours of both original and generated data to fully determine the coverage of the original distribution by generated data.

The loss in performance on TRTS for all GANs indicates that the classifier struggles to classify the generated data. A possible explanation is that the quality of the

generated data may be lower than the original, as discussed in Section 3.4.2. This may relate to the increased noise levels in generated samples compared to the original ones, as seen in Figures 4.1a, 4.3a, 4.5a, and 4.7a. Another reason for the lower quality of generated samples could be that the GAN fails to learn some important feature related to the two classes, which is mentioned above as an explanation for the low TSTR performance as well. As discussed above, if the generator for instance fails to combine the channels appropriately it is possible that the generated samples are not representative of an original sample. If this is the case, one could expect an increased difference between generated and original samples when the entire sample is matched to an original, compared to when only one channel is matched as in Figure 4.10, 4.12, 4.14 and 4.16. However, even if such a difference might exist, it is too small to be spotted in these Figures. Therefore, before making any conclusion, one should investigate the composition of channels further. The general effect on performance from specific channels, and how frequent different shapes are in the data sets are two examples that at the moment are unknown, but could give important information regarding how the samples are made up.

As mentioned regarding TMTR, all four kinds of GAN seem to perform well as presented in Table 4.2. Since this evaluation method is closest to the scenario that the generated data is intended to be used for, it is a great indicator that the GAN has managed to perform the task of creating good enough samples. One can also note that CWGAN-GP + DeLiGAN + noise reaches a higher level of performance compared to the 600 original samples. However, in relation to the standard deviation, the difference between a mixed data set and a set of original data is small enough for this to be considered an effect of randomness, and is not judged to be a significant difference. Regardless of this, it still demonstrates that there is no loss in performance when replacing half of the data with generated samples. It is however worth noting that only a subset of the training set is used for evaluation. As already mentioned in Section 3.4.2, for an overfitted GAN this may indicate a performance increase even though only copies of the total distribution of sample have been added. Considering that neither CWGAN-GP + DeLiGAN with nor without the addition of noise show any signs of overfitting, it can be argued that this effect would not be observed for these GANs. Taking into account that TSTR does not reach as high as TRTR on the same amount of data, it seems likely that there should be a cutoff where there no longer is a performance gain to add more generated data. Whether this is related to a lower quality of generated samples, or limited variation in the generated distribution would require a better understanding of the data distribution for both data sets.

5.1.2 Phantom data

The phantom data displays much more complex behaviours, which is clearly reflected in the decreased performance on original data compared to the simulated data set. A major difference to the simulated data set is also the number of samples, which is decreased by approximately 800 samples when training the GAN. Additionally, the level of noise in the individual samples has increased in the original data compared to the simulated data. Due to this, the decrease in overall performance is in line with

what was expected. Unfortunately, this means that the even smaller training set size for the classifier in TRTS means that no significant level of performance is reached on neither original nor generated data. Some performance is however reached on TSTR, but similarly to the simulated data, it is lower than the corresponding TRTR performance for all kinds of GAN. On TMTR there is a small increase in performance when adding generated to train the classifier, which indicates that the generated data is of high enough quality to contribute in training. However, the increase is not as substantial as when the same amount of real data is added to the training set which and shows that the generated data is not generated perfectly. In contrast to what the poor results received from the evaluation methods indicates, the plotted samples in Figure 4.10a, 4.12a, 4.14a and 4.16a shows generated data that seems to be very similar to the original data. Additionally, when matching the generated data to the original data with regards to the channel as done in Figure 4.10b, 4.12b, 4.14b and 4.16b, the generated data is similar but not identical to the original data. The same behaviour is seen when matching the generated data to the original data with regards to the entire sample as well, as seen in Figure 4.10c, 4.12c, 4.14c and 4.16c.

All four kinds of GAN show some indication of partial overfitting, although it is more prominent for CWGAN-GP and CWGAN-GP + noise. As already stated, the data set size is relatively small for the phantom data and this may also affect the extent of overfitting. For simulated data, the addition of DeLiGAN effectively counteracted overfitting. This effect is visible for the phantom data as well, although not as clear as for the simulated data. The lowered effect of adding DeLiGAN to the network might be due to the smaller data set size. Even though there are varying levels of overfitting in the different kinds of GANs, the performance levels are similar for all evaluation methods. For all evaluations, the standard deviation across evaluations is relatively high. Therefore, it is unclear what effect overfitting has on this data type.

During the evaluation of the phantom data, it became evident that the used classification model is not good enough to classify the data with satisfactory accuracy. By looking at the small class separation in Figure 3.3b, it is not surprising that the classifier struggles. A closer look at the classifier performance shows that it rarely is able to classify both classes correctly, but rather reaches around 80% accuracy on one class and stays at 50% on the other. This behaviour is seen for both original and generated data. Due to this, all evaluation metrics have to be treated a bit cautiously since the classifier accuracy doesn't really represent the quality of the data but rather the classifier capacity. To be able to evaluate the generated data properly, the classifier would have to be improved a lot. Since classifier optimisation was deemed to be outside of the scope of this project, an alternative to this would be to increase the class separation in the data set instead. One way to do this could be to remove smaller bleeding sizes from the data set since one can imagine there to be a larger separation between healthy samples and samples with large bleedings. Since this was not done, it is important to keep the classifiers limited capacity in mind when discussing the received results since it is uncertain to what extent the metrics represent the quality of the generated data. One can argue that if the generated data accurately represents the original data, then the classifier should behave simil-

arly for both data sets. However, the classifier behaves in an unpredictable way and the outcome of the training varies a lot across evaluations even when training and testing on original data. Therefore, it is uncertain how a small difference between the original and generated data would affect the accuracy. It is possible that the classifier is unable to pick up on minor distinctions between the data set, and these differences become hidden in evaluation. On the other hand, minor differences might be exaggerated by the limited capacity of the classifier since it is unlikely that the classifier becomes robust enough to handle small variations which are supported by the high standard deviations across evaluations in Table 4.3 and A.4.

Performance on evaluation metrics

Similarly to the simulated data, some performance is reached for TSTR although it is lower than the performance on original data, as seen in Table 4.3. In contrast to simulated data though, CWGAN-GP with and without the addition of noise have a TSTS that is on par with TRTR. The two remaining GANS, i.e. the two with the addition of DeLiGAN still show a significantly higher TSTS compared to TRTR. As previously discussed, it is reasonable to believe that the generated data is simplified in some way when the TSTS is significantly higher than TRTR. However, this increase shows no significant effect on performance on TSTR, making it hard to draw any conclusions regarding what effect this has on the generated data.

Regarding TRTS, the overall performance shown in Table 4.3 is very low, even for the TRTR it is being compared to. Of the four kinds of GANs, both GANs with the addition of DeLiGAN score the highest on this evaluation methods, but the performance level is too low to make any claims that this is a significant difference. Without being able to evaluate the generated data properly no actual conclusions can be made. A larger training set for the classifier would be beneficial, but this would require removing data from the training set for the GAN. Unfortunately, a decrease in training set size for the GAN will lower the quality of the generated data. Since the data set size used for phantom already is smaller than for simulation, there is a risk that this is responsible at least partially for the decrease in performance.

Evaluation using TMTR shows does not give a substantial increase in performance for any of four kinds of GAN, which is shown in Table 4.4. The biggest increase is seen for CWGAN-GP + DeLiGAN + noise, but even for this type of GAN, the benefit of adding generated data is not close to the addition of real data. However, it is clear that the addition of generated data does not decrease the capacity of the classifier of classifying the test set. As previously discussed, due to the limited capacity of the classifier is probable that the classifier is not robust to handle variations in the data set. Therefore, the fact that there is not a visible decrease in performance when adding generated data implies that there are no major differences between the data sets.

Distribution coverage

As suspected from the lower performance on original data, the separation of classes for phantom data is less distinct compared to the simulated data set. For original

phantom data, this is clearly shown in Figure 3.3, where Figure 3.3b shows the same channel that is visualised throughout the results. The most distinct visible difference between the classes occurs around 0.5 GHz. Regarding the generated data, the separation of classes in this region is present for all kinds of GAN, but with varying amounts of distinction. It is most prominent for CWGAN-GP + DeLiGAN in Figure 4.13a, but on the other hand, this kind of GAN seem to have a less accurate fit in the region 0.9 - 1.1 GHz. Due to similarities between the different kinds of GANs, and low performance levels, it is hard to determine whether this is a positive or negative behaviour. Furthermore, even though this channel was chosen for visualisation due to its relatively large variability, other channels may still play an important role in classification. Thus, looking at a single channel gives a very limited view of the distribution coverage and makes it hard to draw conclusions.

The PCA plots in Figures 4.11b, 4.13b, 4.15b and 4.17b show a large overlap of distributions for generated and original samples, although there are some differences to be noted. As for simulated data, there is some clustering that is not based on class, and the majority of samples between the clusters are generated samples. It was already reasoned for simulated data that this can be either beneficial or a sign that the generated distribution is not good enough and this applies to the phantom data as well. Furthermore, for all kinds of GAN, there are varying levels of an increased portion of generated samples on the right edge of the right cluster. This is most prominent for the two GANs with the addition of DeLiGAN as seen in Figures 4.13b and 4.17b. Perhaps even more interesting is a region in the lower left part of the right cluster for CWGAN-GP in Figure 4.11b which contains almost only original samples. This indicates that there are parts of the distribution that have been missed out on by the generator, which can be part of the explanation for low performance. Once again though, since this specific kind of GAN does not perform significantly worse than the others kinds, it is hard to relate the distribution to a performance loss. In addition to this, since the class separation is not visible in the first two principal components it is not obvious how big these differences in distributions would be in a space where the class separation is visible.

The seemingly best fit of the distribution expressed in the first and second principal component is achieved by CWGAN-GP + noise, shown in Figure 4.15b. This GAN also scores the highest on TSTR and TMTR and gives an increase in accuracy by adding generated data in TMTR. However, the fact that this GAN seems to be the best one is not too unexpected due to the partial overfitting displayed in Figure 4.18c.

5.2 Overfitting

Overfitting has been observed as a problem when training the models during this study. As expected, this was more prominent when training networks with more layers and neurons, and by changing the network architecture the overfitting could be reduced somewhat. Still, when training the final model architecture, overfitting is present to some extent which can be seen in the critic output in Figures 4.9 and

4.18. However, none of these figures shows a training session with full overfitting where the validation set is rated similarly to the generated data. When training a CWGAN-GP with DeLiGAN or a CWGAN-GP with DeLiGAN and noise added to the training data, the overfitting is very low as seen in the critic output in Figure 4.9b, 4.9d, 4.18b and 4.18d. However the critic outputs in Figure 4.9a, 4.9c, 4.18a and 4.18c shows that the validation set is rated in between the original data and the generated data. This would mean that the critic is overfitted in some sense but not completely overfitted, and it is not obvious how partial overfitting affects the generated data.

One possible way that partial overfitting can present itself in the generated samples is that all generated samples do resemble original samples in some way but are still not identical to them as they would have been if the critic was fully overfitted. Since the desired output from the generator is samples that are similar to the original data, it is difficult to distinguish between a sample that is too similar and a sample that is different enough from the real samples but still realistic and have a sensible composition of channels. Due to the nature of the microwave data used here, it is impossible to tell if a generated sample is sensible or not just by looking at it. If the data is more similar to the original data than desired due to overfitting, then the generated data can still contribute to an increased performance similar to how adding noise to data is a data augmentation method. In an attempt to investigate this possibility, one can look at the channels in a generated sample individually and see if the channel is simply a copy of an original sample or if there is some differentiation between them. In Figure 4.1b, 4.3b, 4.5b, 4.7b, 4.10b, 4.12b, 4.14b and 4.16b a single channel is seen for multiple samples along with the original sample for which this channel is the most similar. One can see that for almost all generated samples, there is an original sample with this channel being very similar. Even though the generated samples do look very much like the original samples, they are not identical copies. When the entire generated sample is compared to the original sample that is the most alike, it is clear that the generated samples do have visible deviations from the original samples as seen in Figure 4.1c, 4.3c, 4.5c, 4.7c 4.10c, 4.12c, 4.14c and 4.16c. It is difficult to tell if the difference between generated and original samples is the sought diversity or if more differentiation is ideal and the similarities are due to the partial overfitting. A deeper study on this topic would be needed to certainly say that the partial overfitting results in generated data that is similar but not copies of the training data.

Another possibility is that some kinds of samples and channels are very well represented but other types are simply copies. For example, the GAN could have learnt to generate varieties of bleedings in one part of the brain but not another, or only bleedings of a certain size has been created correctly while another size is copied from the training set due to limited variation of this specific feature in the training set which increases the risk of overfitting. This could still help increase the performance of a classifier, but it is crucial that one is aware of this flaw in the generated data since it could introduce a bias in a classifier. One way of testing if this has happened is to split the test set in TMTR into smaller subsets with respect to some aspect, for example into subsets of small, medium and large bleedings. This way it

will become clear if the classifier used in TMTR has received different quality data for different bleeding sizes. This behaviour has not been observed for the generated data presented in Chapter 4. It is however possible that this behaviour is present some other feature than bleeding size.

As stated in Section 3.4.2, most of the evaluation methods do not take overfitting into account. Since training CWGAN-GP and CWGAN-GP + noise results in partial overfitting, the evaluation methods cannot be blindly trusted to determine which of the four types of GAN is the most effective. Before the final architecture of the GAN was set, full overfitting was observed for some architectures using the method described in Section 3.3 when training on simulated data. However, the generated data yielded very high accuracies for all three evaluation methods. This enforces the importance of keeping overfitting in mind when looking at the accuracies in Chapter 4.

Bearing in mind that the end goal of this project is to use the generated data to improve the performance of a classifier, one can argue that the accuracies on TMTR can be compared to one another as well regardless of the GAN is overfitted or not. TMTR can then be used to decide which GAN creates generated data that improves the performance of the classifier the most, but it is not certain that this will give a complete picture of the generated data and it is possible that the generated data introduces biases in some way.

5.3 Future work

Although promising results have been acquired during this project, there is room for improvement in order to further approach the end goal. A continuation of this work can however take many different directions, and in the following sections, these are divided into four main categories with a brief discussion of suggestions for future work.

5.3.1 Improve the used GAN model

There is potential to improve the performance of the used Conditional Wasserstein GAN beyond what has been done in this project. The treatment of the data sets and better tuning of the network could possibly improve performance without any fundamental changes to what has already been done.

Multilabel classes

This study makes use of a conditional GAN, with the two classes healthy and bleeding. However, it is not unreasonable to suspect that there is large variability within the bleeding class related to the different sizes of the bleedings. Therefore, one could expand the conditional GAN to a multiclass problem where the bleeding class is separated into multiple classes, for instance small, medium and large bleedings. It is possible that this could direct the learning towards specific features of a certain

bleeding size, while at the moment it is possible that a generated sample does not correspond well to a certain bleeding size.

One question that arises is how to split the bleeding sizes into categories, and how many classes that are sensible to implement. When the bleeding sizes are not discrete values, but rather continuous variables, it may be problematic to split them into discrete groups. It is also unclear what happens when samples have bleeding sizes very close to where the split for the size categories occurs. One way to solve that problem could be to use continuous conditional GAN [43, 44]. Although this so far is a relatively unexplored adaptation of GANs, it has shown to generate promising results.

Hyperparameter tuning

Optimisation of hyperparameters is of importance when training GANs, and different kinds of GANs are sensitive to hyperparameters and architectures to a varying degree [45]. The sensitivity of Wasserstein GANs is debated, with studies showing that it is robust towards changes in hyperparameters [45] and other studies showing its instability [17]. In general, GANs are sensitive to hyperparameters and architectures, and even if Wasserstein GANs are shown to be robust, optimising the hyperparameters is beneficial for performance.

Augment training data for the GAN

Data augmentation is usually an effective way to generalise a network, however, it is not obvious how to augment data in a GAN. If the training data fed to the discriminator is augmented, the discriminator would push the generator to generate samples similar to this which would lead to the generated data being augmented. Using Adaptive Discriminator Augmentation (ADA) [41] is a way to combat this by making the augmentation process not random, which allows the GAN to generate non augmented data. Using ADA has been shown to greatly reduce the need for large data sets. It has been shown that by using ADA, only a tenth of the number of samples is required without any loss in the quality of the generated data.

5.3.2 Other approaches to data preprocessing

The nature of the data set used to train the GAN to give rise to some of the key challenges of the project. Therefore, a deeper investigation of how to preprocess the data could be beneficial to achieve efficient training.

Feature selection

A main issue with the data is the many features in comparison to the data set size, and it is likely that this contributes towards the overfitting that can be seen for some of the GANs. It is therefore desirable to decrease the number of features. One common approach to achieve feature selection is to look at the principal components. However, for this data set, no principal components with a clear class separation

were found. Either a further investigation of whether PCA is an appropriate representation of the data is required, or another method for feature selection should be used. Future works to understand the importance of specific channels, and specific frequencies may aid decision making to reduce the number of features.

Alternative representation of data

The microwave data used in this thesis can be transformed into a time series via the inverse Fourier transform. Some previous studies have been performed on time series [29, 23], and it is a much more explored application area of GANs than data in the frequency domain. The studies mainly use recurrent GANs and have shown successful results in the generation of time series data. How this would apply to the data used in this project is uncertain, but it still provides an interesting direction for continued work.

One could also investigate the possibility of representing the data in some image format. Since the vast majority of GAN implementations are based on images and use convolutional networks, this would increase the number of relevant studies drastically. Having more guidelines to follow would simplify the decision making during implementation, and could point towards reasonable architectures that have shown to generate successful results previously.

5.3.3 Better evaluation and understanding of generated data

One of the main focus areas of the project was how to appropriately evaluate generated data. Although the chosen evaluation methods aim to imitate the intended usage, they also pose some limitations in their capacity to capture all aspects of generated data quality. Therefore, improvements of the evaluation methods and alternative measures of data quality are discussed in the following text.

Quantifying overfitting

Overfitting is a common problem in GANs, and also present to a varying extent for the implementation of this project. However, as has been discussed in this report, it is unclear what effect it actually has on the generated data. When visualising the data, it is clear that the generator has not generated exact copies, but at the moment there is no measure for how similar generated and real samples are. Such a measure would be beneficial for the evaluation of the data, and it would also require a measure of the desired similarity based solemnly on the variety in the real data set. One suggestion of such a measure is the Maximum Mean Discrepancy, which quantifies the distance between the real and generated data. Hyland et al. [23] have discussed this topic extensively, and using Maximum Mean Discrepancy are one of the suggested methods.

Investigate the connections of different channels

Even though the channels look good when visualised, it would be interesting to see how a classifier trained on the channels individually compares for real and generated

data. Primarily, it would provide information regarding the relative importance of the different channels. For instance, if some individual channel in the real data set shows a very high TRTR accuracy, it indicates that it plays an important role during classification. Therefore it is important that the generator has captured the behaviour of that specific channel.

Secondly, performance on individual channels would indicate whether the GAN actually manages to generate meaningful data for all channels. If that is the case, this would support the already discussed scenario where the GAN struggles to combine the channels. On the other hand, lower accuracy for generated data than real data for some channel would also point toward where the generator struggles to represent the real distribution.

More sophisticated classifier

It is suggested in Section 5.1.2 that the classifier is not good enough to evaluate the data satisfactorily. A first step forward to solve this issue is to optimise the architecture and hyperparameters of the classifier. However, it could also be of interest to complement the evaluation methods that depend on the classifier with methods that do not require a classifier. As discussed above, using metrics to quantify the differences in the distributions is one way to do this.

5.3.4 Alternative GAN models

There is a multitude of GAN varieties to use to generate more data. Some of these have fundamental differences in implementation to the ones used in this project. A few other GAN models are deemed to be interesting alternative ways of approaching the problem at hand, and these will be presented below.

Another approach to data augmentation with GANs

The focus of this project has been on generating realistic samples that can be used to extend the original data set and that way improve the performance of Medfield Diagnostics Strokefinder MD100. Another way of approaching the fundamental goal of improving the performance is to augment the training data instead of generated completely realistic samples. By using a Data Augmentation Generative Adversarial Network (DAGAN) [46] a wider variety of augmented samples is provided. In this GAN, the generator is fed both noise and a real sample as input instead of only noise as is the case for many other GAN implementations. The use of a DAGAN has been shown to successfully augment a range of different classifiers trained on different data set and increasing their performance. This approach seems to be interesting to explore since it is aimed at improving classifiers in the low-data regime.

Generation of class-specific data with semi-supervised learning

It is important that the class-specific features are captured by the GAN to make the data usable in classification tasks. By making use of semi-supervised learning in

the discriminator it can instead act as a combined discriminator and classifier that predicts which of the $N+1$ classes the sample belong to [47]. The $N+1$ samples are the N classes in the original data set, and 1 for generated data. Similarly to how a conditional GAN is used to enhance class-specific features, semi-supervised learning enables the discriminator to guide the generator towards generating samples that have a clearer class separation.

A combination of a conditional GAN and semi-supervised learning is suggested by Odena [47]. When conditioning the input to the generator, it is possible to use $2N$ classes in the discriminator which ought to help the generator even more. This would be an interesting approach to generate data that is well suited to use in classification tasks.

Convolutional networks in the GAN

It was argued that using convolutions on the data set in its raw format is problematic due to the discontinuity between channels. As stated before though, deep convolutional GANs (DCGAN) are widely used in other GAN implementations, and since this was never tried during this project it is still possible that it works for this data set as well. To minimise the spatial discontinuity, one can still argue that 1D convolutions are more sensible than 2D convolutions. However, this is correlated with how the data is represented. To represent the data in the best possible way, it is important to acquire a deeper understanding of the composition of channels.

6

Conclusion

To conclude, it is certainly possible to generate microwave data with enough class-specific features for the data to be used in classification problems. The generated data also shows promising results in the context of adding it to a set of real data in order to increase the performance of a stroke classifier, which is the intended usage of the data. However, even though the generated data largely represents the original data, it is still lacking in some unknown quality measure. Therefore, there is a need for a greater understanding of this type of data to be able identify what causes the remaining discrepancies between the generated and original data.

Some properties in the data were identified as key challenges for this project. The first was the small number of samples, and the second was the large number of features for each sample. These factors in combination are considered to be contributing factors to the overfitting occurring during training. However, the addition of DeLiGAN effectively counteracts this, but with a simultaneous loss in distribution coverage. Instead, to get a more representative distribution, it was beneficial to add noise to the training data. Furthermore, the third identified challenge was working with non-image data in contrast to most previous GAN implementations. Therefore, only multilayer perceptrons were used to implement the GANs, and as already stated the project concludes that the generation of data was successful.

A secondary focus of the project was to find appropriate evaluation methods for the generated data. The chosen evaluation methods were Train on Synthetic, Test on Real (TSTR), and Train on Real, Test on Synthetic (TRTS). In addition to these, the evaluation method Train on Mixed, Test on Real (TMTR) was developed to imitate the intended usage of the generated data. A combination of these manages to capture the quality of the data well, and form the foundations on which the conclusion of the project are based. However, none of the methods captures the effect of overfitting well enough to determine whether the generated data is too similar to the real data. Therefore, it is concluded that another metric for the similarity between the distributions would be favourable.

The classifier used during evaluation struggled to evaluate the phantom data appropriately, and this yielded unreliable results. However, visualisation of the generated data indicates that the GAN actually manages to generate representative data. Therefore, it is concluded that a better evaluation of the generated data from the phantom data set is required, although the results are still considered promising in the context of data generation to expand an existing data set.

A

Tables

Table A.1: Performance on TSTR and TRTS for simulated data on three identical runs. Both evaluation methods are compared to a classifier that is both trained on real data and tested on real data, where the the number of training is the same as for the corresponding evaluation method.

	TRTR	TSTS	TSTR	TRTR	TRTS
Number of training samples for classifier	3027	3027	3027	520	520
Run 1	0.893 ± 0.0075	0.98 ± 0.011	0.860 ± 0.0047	0.74 ± 0.042	0.71 ± 0.037
Run 2	0.88 ± 0.014	0.97 ± 0.011	0.85 ± 0.010	0.784 ± 0.0084	0.744 ± 0.0058
Run 3	0.909 ± 0.0094	0.980 ± 0.0070	0.878 ± 0.0056	0.803 ± 0.0057	0.772 ± 0.0089

Table A.2: Performance on TMTR for simulated data on three identical runs, measured as a mean and standard deviation. The addition of 300 synthetic samples is compared to the addition of 300 extra real samples.

	TMTR		
Test data for classifier	300 original	300 original + 300 generated	600 original
Run 1	0.71 ± 0.037	0.77 ± 0.030	0.79 ± 0.020
Run 2	0.72 ± 0.044	0.79 ± 0.025	0.79 ± 0.023
Run 3	0.75 ± 0.029	0.82 ± 0.027	0.83 ± 0.016

Table A.3: Performance on TSTR and TRTS for phantom data on three identical runs. Both evaluation methods are compared to a classifier that is both trained on real data and tested on real data, where the the number of training is the same as for the corresponding evaluation method.

	TRTR	TSTS	TSTR	TRTR	TRTS
Number of training samples for classifier	2266	2266	2266	339	339
Run 1	0.68 ±0.025	0.64 ± 0.030	0.58 ± 0.020	0.57 ± 0.030	0.53 ± 0.012
Run 2	0.67 ± 0.028	0.72 ± 0.037	0.62 ± 0.022	0.53 ± 0.031	0.51 ± 0.019
Run 3	0.67 ± 0.020	0.63 ± 0.027	0.56 ± 0.028	0.54 ± 0.016	0.519 ± 0.0062

Table A.4: Performance on TMTR for phantom data on three identical runs. The addition of synthetic data is compared to training on the whole training set of 2152 samples.

	TMTR	
Test data for classifier	2266 original	2266 original + 2266 generated
Run 1	0.68 ± 0.024	0.65 ± 0.022
Run 2	0.68 ± 0.025	0.65 ± 0.019
Run 3	0.67 ± 0.025	0.65 ± 0.014

DEPARTMENT OF PHYSICS
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY

Bibliography

[1] Thomas Davenport and Ravi Kalakota. “The potential for artificial intelligence in healthcare”. In: *Future Healthcare Journal* 6.2 (2019), pp. 94–98. ISSN: 2514-6645. DOI: 10.7861/futurehosp.6-2-94. eprint: https://www.rcpjournals.org/content/6/2/94.full.pdf. URL: https://www.rcpjournals.org/content/6/2/94.

[2] S.M. McKinney, M. Sieniek and V. et al. Godbole. “International evaluation of an AI system for breast cancer screening”. In: *Nature* 577 (2020), pp. 89–94. DOI: 10.1038/s41586-019-1799.

[3] XY. Zhou, Y. Guo and M. et al. Shen. “Application of artificial intelligence in surgery”. In: *Front. Med.* 14 (2020), pp. 417–430. DOI: 10.1007/s11684-020-0770-0.

[4] Agnieszka Mikołajczyk and Michał Grochowski. “Data augmentation for improving deep learning in image classification problem”. In: May 2018, pp. 117–122. DOI: 10.1109/IIPHDW.2018.8388338.

[5] Ian Goodfellow et al. “Generative Adversarial Nets”. In: *Advances in Neural Information Processing Systems*. Ed. by Z. Ghahramani et al. Vol. 27. Curran Associates, Inc., 2014, pp. 2672–2680. URL: https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

[6] Folkhälsomyndigheten. *Insjuknande i stroke*. Mar. 2021. URL: https://www.folkhalsomyndigheten.se/folkhalsorapportering-statistik/tolkad-rapportering/folkhalsans-utveckling/resultat/halsa/insjuknande-i-stroke/.

[7] National Center for Chronic Disease Prevention, Division for Heart Disease Health Promotion and Stroke Prevention. *About Stroke*. Jan. 2020. URL: https://www.cdc.gov/stroke/about.htm.

[8] 1177 Vårdguiden Peter Tuominen. *Stroke*. Feb. 2020. URL: https://www.1177.se/Vastra-Gotaland/sjukdomar--besvar/hjarna-och-nerver/stroke-och-blodkarl-i-hjarnan/stroke/.

[9] National Center for Chronic Disease Prevention, Division for Heart Disease Health Promotion and Stroke Prevention. *Stroke Treatment*. Nov. 2010. URL: https://www.cdc.gov/stroke/treatments.htm.

[10] National Health Service (NHS). *Diagnosis Stroke*. Aug. 2019. URL: https://www.nhs.uk/conditions/stroke/diagnosis/.

[11] *NIH strokeskala (NIHSS)*. URL: https://www.gu.se/neurovetenskap-fysiologi/nih-strokeskala-nihss.
[12] *Strokefinder MD100*. URL: https://www.medfielddiagnostics.com/products/.
[13] Medfield Diagnostics AB. *Medical microwave based systems*. URL: https://www.medfielddiagnostics.com/technology/.
[14] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: 1606.03498 [cs.LG].
[15] Martin Arjovsky, Soumith Chintala and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].
[16] Xudong Mao et al. *Least Squares Generative Adversarial Networks*. 2017. arXiv: 1611.04076 [cs.CV].
[17] Mario Lucic et al. *Are GANs Created Equal? A Large-Scale Study*. 2018. arXiv: 1711.10337 [stat.ML].
[18] Yann LeCun and Corinna Cortes. "MNIST handwritten digit database". In: (2010). URL: http://yann.lecun.com/exdb/mnist/.
[19] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.
[20] Tero Karras et al. *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. 2018. arXiv: 1710.10196 [cs.NE].
[21] Alec Radford, Luke Metz and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].
[22] Santiago Pascual, Antonio Bonafonte and Joan Serrà. *SEGAN: Speech Enhancement Generative Adversarial Network*. 2017. arXiv: 1703.09452 [cs.LG].
[23] Stephanie Hyland, Cristóbal Esteban and Gunnar Rätsch. "Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs". In: (June 2017). URL: https://arxiv.org/abs/1706.02633.
[24] Agrim Gupta et al. *Social GAN: Socially Acceptable Trajectories with Generative Adversarial Networks*. 2018. arXiv: 1803.10892 [cs.CV].
[25] Jiajun Zhang. "Deformable Deep Convolutional Generative Adversarial Network in Microwave Based Hand Gesture Recognition System". In: (Nov. 2017).
[26] M. Wang et al. "High-resolution Three-dimensional Microwave Imaging Using a Generative Adversarial Network". In: *2019 International Applied Computational Electromagnetics Society Symposium - China (ACES)*. Vol. 1. 2019, pp. 1–3. DOI: 10.23919/ACES48530.2019.9060477.
[27] Yufei Liu et al. "Wasserstein GAN-Based Small-Sample Augmentation for New-Generation Artificial Intelligence: A Case Study of Cancer-Staging Data in Biology". In: *Engineering* 5.1 (2019), pp. 156–163. ISSN: 2095-8099. DOI: https://doi.org/10.1016/j.eng.2018.11.018. URL: https://www.sciencedirect.com/science/article/pii/S2095809918301127.
[28] Maayan Frid-Adar et al. "GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification". In: *Neurocomputing* 321 (Dec. 2018), pp. 321–331. ISSN: 0925-2312. DOI: 10.1016/

j.neucom.2018.09.013. URL: http://dx.doi.org/10.1016/j.neucom.2018.09.013.

[29] Mohammad Fekri, Ananda Mohon Ghosh and Katarina Grolinger. "Generating Energy Data for Machine Learning with Recurrent Generative Adversarial Networks". In: *Energies* 13 (Dec. 2019). DOI: 10.3390/en13010130.

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

[31] Mahmoud Hamdy. *INTRODUCTION TO GANS*. Dec. 2019. URL: https://dataisutopia.com/blog/introduction-to-gans/.

[32] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. arXiv: 1701.00160 [cs.LG].

[33] Manisha Padala, Debojit Das and Sujit Gujar. *Effect of Input Noise Dimension in GANs*. Apr. 2020.

[34] Cédric Villani. "Grundlehren der mathematischen Wissenschaften". In: *Optimal Transport: Old and New.* Springer, 2009.

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

[36] Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton. *Layer Normalization*. 2016. arXiv: 1607.06450 [stat.ML].

[37] Emily Denton et al. *Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks*. 2015. arXiv: 1506.05751 [cs.CV].

[38] Guim Perarnau et al. *Invertible Conditional GANs for image editing*. 2016. arXiv: 1611.06355 [cs.CV].

[39] Swaminathan Gurumurthy, Ravi Kiran Sarvadevabhatla and Venkatesh Babu Radhakrishnan. *DeLiGAN : Generative Adversarial Networks for Diverse and Limited Data*. 2017. arXiv: 1706.02071 [cs.CV].

[40] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2014. arXiv: 1312.6114 [stat.ML].

[41] Tero Karras et al. *Training Generative Adversarial Networks with Limited Data*. 2020. arXiv: 2006.06676 [cs.CV].

[42] Yasin Yazici et al. *Empirical Analysis of Overfitting and Mode Drop in GAN Training*. 2020. arXiv: 2006.14265 [cs.LG].

[43] Xin Ding et al. "CcGAN: Continuous Conditional Generative Adversarial Networks for Image Generation". In: *CoRR* abs/2011.07466 (2020). arXiv: 2011.07466. URL: https://arxiv.org/abs/2011.07466.

[44] Yufeng Zheng, Yunkai Zhang and Zeyu Zheng. *Continuous Conditional Generative Adversarial Networks (cGAN) with Generator Regularization*. 2021. arXiv: 2103.14884 [cs.LG].

[45] William Fedus et al. *Many Paths to Equilibrium: GANs Do Not Need to Decrease a Divergence At Every Step*. 2018. arXiv: 1710.08446 [stat.ML].

[46] Antreas Antoniou, Amos Storkey and Harrison Edwards. *Data Augmentation Generative Adversarial Networks*. 2018. arXiv: 1711.04340 [stat.ML].

[47] Augustus Odena. *Semi-Supervised Learning with Generative Adversarial Networks*. 2016. arXiv: 1606.01583 [stat.ML].