Generative Adversarial Nets

For generating synthetic Imaging Mass Spectrometry data

Willem van der Linden

Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Generative Adversarial Nets

For generating synthetic Imaging Mass Spectrometry data

Thesis

Willem van der Linden

June 25, 2024

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

Abstract

This report investigates the use of Generative Adversarial Nets (GANs) specifically for oversampling Imaging Mass Spectrometry spectra. IMS is a technique used to measure the spatial distribution of molecules, which is valuable in fields like oncology and biomarker discovery. GANs, on the other hand, are a class of machine learning frameworks where two neural networks, the generator, and the discriminator, are trained simultaneously through adversarial processes. The generator creates synthetic data, while the discriminator tries to distinguish between real and synthetic data.

GANs-based oversampling aims to increase classifier performance by adding data to classes that are underrepresented in the original data. Synthetic oversampling is especially relevant in IMS data as the measuring technique is destructive, making acquiring more real samples impossible. GANs have been shown to outperform other oversampling techniques such as SMOTE on various datasets. Applying GANs directly to the dataset proved unsuccessful in this oversampling task.

Different possible causes of the limited performance of the GANs are studied leading to improved experiment results using spectra reduced in dimension and the Wasserstein GANs with gradient penalty. Even though with these changes to the experiment the GANs appear to generate more realistic data, using this data for oversampling does not increase overall classifier performance. Rather, it steers the classifier to overfitting towards the minority classes.

This report demonstrates that applying the designed GANs for oversampling minority classes on this dataset does increase classifier performance. However, it is shown that GANs can be trained on IMS data and that GANs might be of use for applications with IMS data besides oversampling.

Table of Contents

Preface

This document is my master thesis on "Generative Adversarial Nets for generating synthetic imaging Mass Spectrometry data". This thesis is the final document, concluding over a year of work on the subject. Throughout this period, I have reviewed the existing literature and implemented the framework using real data. This document compiles the most significant experiments and conclusions drawn from my research.

I would like to thank my supervisor Dr. ing. Raf van de Plas for the opportunity to work on the frontier of AI applied to IMS data.

I want to thank Dr. Lukasz Migas for providing background information on the data used in the various experiments.

A special thank you goes to my daily supervisor Ir. Roger Moens for the weekly meetings and regular feedback, criticism, and motivation.

Finally, I want to thank all other PhD-candidates and students working on similar topics. The regular group meetings were a great help and motivator to bring this thesis to a successful conclusion.

Willem van der Linden
Delft University of Technology, June 25, 2024

"Generative AI is like a mischievous genie in a bottle, granting your wishes with a twist of randomness and a sprinkle of absurdity. Just when you think you've mastered it, it unleashes a dancing elephant in your data set!"

— *ChatGPT*

Chapter 1

Introduction

This work gives a practical introduction to using Generative Adversarial Nets (GANs) with Imaging Mass Spectrometry Imaging Mass Spectrometry (IMS) data. IMS is a measurement of the molecular mass distributions with respect to the spatial location. The resulting data contains a mass spectrum for every pixel, making a dataset relatively large. In this work, GANs are used to study the mass distributions i.e. the effect of adding generated spectra to the original data in a classification task.

GANs are a relatively novel way to model the data distribution of data. To do this a parametric generator function maps noise from a set distribution to generated samples. A parametric discriminator function aims to distinguish between generated and original data. Based on the accuracy of the discrimination both parametric functions are updated until the generated and original samples are indistinguishable. If the generated data is indistinguishable from the original data the generator has successfully learned the distribution of the original data.

GANs have been applied for many different machine learning tasks such as clustering, classification, and anomaly detection. In this work, the focus is on generating data. To verify that the data is of useful quality we use the data for oversampling minority classes. Useful is here defined as sufficiently realistic data that can be interchanged with the original data without changing classifier performance and the generated data should increase classification performance if generated data is added to minority classes.

1-1 Problem statement

Minority classes are a common challenge in machine learning. A minority class is defined as a class that is underrepresented in a dataset. Minority classes can hinder a classification algorithm and might develop a bias toward the majority classes. A few different approaches are used to reduce the negative effect of data imbalance. The easiest method is undersampling majority classes, i.e. reducing the number of samples of overrepresented classes. This method has the downside of throwing out samples and possibly reducing the variance in the dataset thereby limiting classifier performance.

Ideally, minority classes are averted by measuring more samples, which is not always possible. In the case of IMS data, a limiting factor for getting more samples is the destructive nature of the measurement.

Another method to correct for so-called class imbalance is the synthetic oversampling of minority classes. This means that additional, synthetic data is constructed such that the minority classes are represented as often as the majority classes. A naïve way to oversample data is to copy samples. A common way to oversample data is by using SMOTE. Recently multiple studies have shown that using GANs for oversampling instead of SMOTE can sometimes result in better classifier performance[1, 2].

1-2 Goal

This project aims to show that GANs can be applied to IMS data and that using GANs can lead to new methods for analyzing this data. To verify that GANs indeed learn the data distribution of IMS data, this work specifically studies the use of GANs for oversampling minority classes in IMS data. To do this, conditional GANs are trained on a multi-class (more than 2 classes) dataset. After training, the GANs are used for oversampling and the original training dataset is extended by generated data. The resulting classification accuracy is measured on a test set unseen by the classifier and GANs. The workflow of GANs-based oversampling is given in Figure 1-1. Note, that instead of GANs, other oversampling techniques like SMOTE follow the same workflow.

The resulting classifier results are compared to three baseline methods: adding no data, adding simple augmented data, and adding SMOTE-generated data.

1-3 Document overview

The main results of this research including most relevant backgrounds and methods are summarized in a paper included in this report as chapter 4. Besides this paper, the rest of this report can be read as additional information for this paper:

In chapter 2 background information on both IMS data and GANs is given. This chapter discusses some background on IMS techniques as well as preprocessing of data that is outside of the scope of this project but still relevant for the data used. The section on GANs goes in-depth on the theory behind adversarial learning, gives some examples of different GANs in practice, and introduces performance indicators used to check the quality of generated data.

In chapter 3 different techniques for scaling the data are introduced as well as the baseline methods (SMOTE and data augmentation) and the classifier used in the experiments.

In chapter 4 the paper is included, this paper can be read as a stand-alone document.

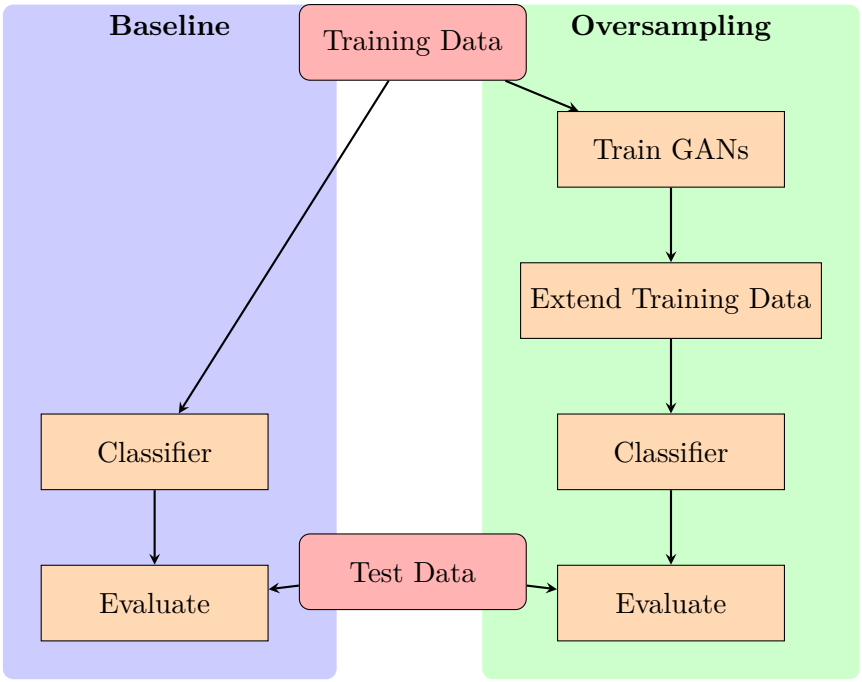


Figure 1-1: Workflow of GANs-based oversampling. On the left-hand side: the baseline where only the unbalanced training data is used to create a classifier. On the right-hand side: the same training data is used to train GANs these GANs are then used to generate minority samples thereby extending the dataset. This extended dataset is used to create a classifier that is tested against the same original test dataset as the baseline classifier.

In chapter 5 the steps for the design of the generator and discriminator functions are explained. This chapter contains some tests to evaluate if the chosen function is sufficiently complex to model the data. In this chapter, a benchmark test is included to verify that the GANs are implemented correctly.

In chapter 6 different experiments are summarized. Using the results from these experiments a final experiment on the full dataset is executed.

This document ends with a conclusion and recommendations for further research and different research directions for GANs for IMS. The main results from these experiments are verified by an additional dataset in Appendix A. Additionally, the appendix includes a short experiment on anomaly detection Appendix B and alternative designs of neural networks.

Chapter 2

Background

To use GANs for oversampling of IMS data, basic knowledge is required of the data and the GANs framework. This chapter introduces some background information on IMS; the basic working of IMS is explained, followed by information on IMS data in general and an introduction to the dataset used for experimentation.

GANs are explored in depth, first, the basic optimization and notation are introduced followed by some relevant modifications. As additional information, some applications of GANs beyond data generation are given.

2-1 Imaging Mass Spectrometry

Imaging Mass Spectrometry (IMS) is a technique to measure the distribution of molecules with respect to their spatial location, i.e. Mass Spectrometry (MS) on a spatial grid. MS allows for the label-free discovery of the chemical composition of a sample. By measuring the mass spectra for every pixel in a grid a spatial resemblance of the different spectra is captured.

An analogy can be made with other imaging techniques such as fluorescence microscopy or hyperspectral imaging. Where most known imaging techniques capture color (light ranges) for every pixel, IMS captures different molecular masses.

IMS has huge potential in different fields. For example, in oncology research, IMS can be used to expedite drug development as it allows for the study of the effect of drugs on the tumor microenvironment[3]. Another application is in biomarker discovery in toxicity studies. IMS can be used to study the effects of administered drugs on a tissue sample, giving insights into the possible adverse effects of the medicine[4].

Imaging mass spectrometry is a rapidly growing field, as evidenced by the increasing number of publications shown in Figure 2-1. Research in the IMS field goes into the applications e.g. drug development[5, 6] as well as statistical and dimensionality reduction methods to process the vast amounts resulting from an IMS process.[7, 8]

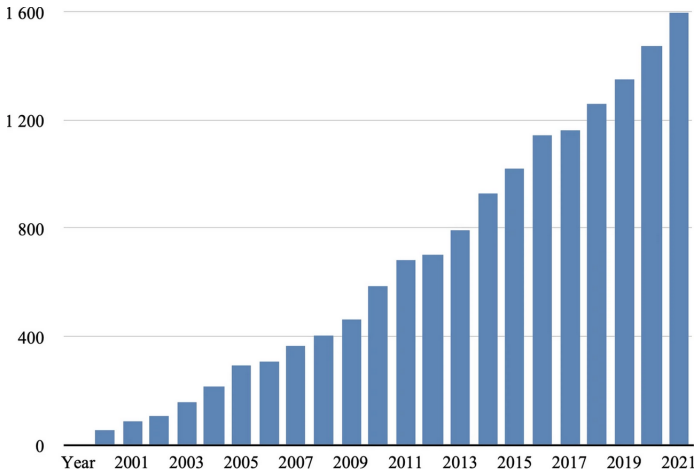


Figure 2-1: Number of publications on 'mass spectrometry imaging', based on search results on PubMed. (From [9])

In the following sections, an introduction to the working principle of IMS is given, followed by an introduction to IMS data in general as well as the IMS data that is used in the rest of this work.

2-1-1 Working principle of IMS

Imaging mass spectrometry is mass spectrometry on a spatial grid. The working principle of IMS therefore is very similar to mass spectrometry. An ionizing probe is used to ionize a part of a sample. Ions break from the sample and are captured and measured by a mass analyzer. For ionization, capturing and measuring different techniques are available. In Figure 2-2 IMS is visualized, at the top left a piece is tissue is ionized by a UV laser. Every red dot is a spot where the tissue is measured resulting in a mass spectrum visualized in the top right. If for every pixel the same mass bin is selected a spatial view of the intensity of the selected mass is generated as shown in the bottom right corner.

The most used form of ionization in IMS appears to be Matrix Assisted Laser Desorption and Ionization (MALDI). In MALDI a matrix solution is applied to the sample before measuring. This matrix reduces the wear of the sample thereby allowing for more distinguished masses can be measured before the tissue becomes too damaged and the measurement unreliable. MALDI allows for large mass ranges to be measured; from 100 up to 500.000 m/z[10].

Another technique for ionization is Secondary Ion Mobility Separation (SIMS). SIMS has a higher spatial resolution than MALDI, however, the maximum mass-to-charge ratio that can be measured is relatively limited to approximately 1000m/z[10].

A method for measuring the ion stream often combined with MALDI is Time Of Flight (TOF). TOF accelerates the ions with short electromagnetic pulses. By measuring the resulting velocity (time in which the ions fly a certain distance) the acceleration and therefore the mass relative to the given charge can be calculated. In (2-1) the two physical principles used for calculating the mass-to-charge ratio are given; Newton's second law of motion and Lorentz

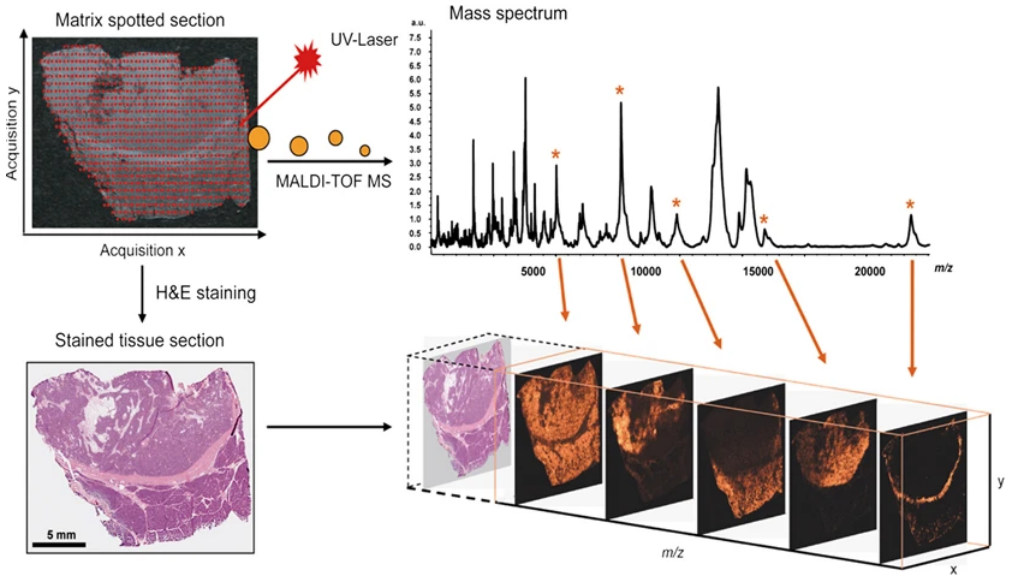


Figure 2-2: Principle of MALDI IMS. Top left: data collection, top right: mass spectrum of a single pixel, bottom left: tissue sample, bottom right, spatial representation of single mass bins. From[5]

force[11].

$$\begin{aligned} F &= ma \\ F &= q(E + v \times B) \\ ma &= q(E + v \times B) \\ \frac{m}{q} &= \frac{E + v \times B}{a} \end{aligned} \tag{2-1}$$

In (2-1) F is the force applied to the ion, q is the charge of the ion, E is the electrical field, and $v \times B$ is the vector cross-product of the ion velocity and applied magnetic field. The mass-to-charge ratio m/z is dimensionless as z is proportional to the charge of a proton e i.e. $z = q/e$. Depending on the ionization method used ions can have different charges which leads to the observation that e.g. an ion of 12 Dalton (carbon-12) is detected as 12m/z for single-charged ions while the same mass is detected as 6m/z with double-charged ions.

2-1-2 IMS data

An IMS dataset contains a mass spectrum for every pixel as shown in Figure 2-2. As every mass bin is represented in every pixel, a spatial view of the mass intensities of single mass bins can be created as shown in the bottom right of Figure 2-2. The number of mass bins, in this work, considered the size of the spectra, is dependent on the mass range as well as the mass resolution (or mass resolving power) used during the measurement. The mass resolving power or the separation between peaks is denoted Δm and the ratio $m/\Delta m$ can be used to measure the ability of the mass spectrometer to separate ions[12].

The raw data resulting from an experiment undergoes multiple processing steps such as smoothing, baseline correction, peak alignment, and peak picking to prepare the data for

downstream statistical analysis[13]. Peak picking reduces the signal-to-noise ratio and aims to select the peaks needed for analysis. Additionally, peak integration can be used together to combine neighboring peaks, thereby decreasing the number of peaks[14].

An imaging mass spectrometry data set quickly grows in size as it is a 3D tensor with two spatial dimensions commonly denoted x and y and one mass dimension. A single pixel of this tensor contains a single spectrum and looking at one m/z-bin results in a 2D image as shown on the right side of Figure 2-2. Using a higher number of peaks, either a larger ranger or with a lower mass ratio, or a higher spatial resolution drastically increases the size of the dataset. Datasets of multiple GB up to hundreds of GB are not uncommon, which makes it difficult to process. e.g. A dataset of multiple GB can not be loaded (into working memory) in its entirety on a simple laptop let alone being able to do analysis.

To make analyzing the large IMS datasets less difficult, i.e. possible on simpler hardware, a lot of attention from the IMS research is focused on dimensionality reduction. Dimensionality reduction aims to reduce the data size while preserving the most important features captured in the dataset. Examples of dimensionality reduction are principal component analysis (PCA) and non-negative matrix factorization (NMF). In multiple experiments in chapter 6, dimensionality reduction in the form of NMF is used. NMF is especially attractive for IMS as the data is inherently non-negative.

NMF splits the dataset (matrix) into two smaller matrices by minimizing the difference of the original data and a multiplication of the two smaller matrices. In (2-2) X denotes the data matrix of N spectra of size n, W denotes the weights and H can be seen as a matrix containing base spectra that build up the original spectra. c Is the number of weights or base spectra that should be used to approximate X. The difference is in this case calculating using the squared Frobenius norm, as this was used in the experiments as well.

$$\begin{aligned} &\min_{W,H\geq 0} ||X - WH||_F^2 \\ \text{Where } &X \in \mathbb{R}_+^{N\times n} \\ &W \in \mathbb{R}_+^{N\times c} \\ &H \in \mathbb{R}_+^{c\times n} \end{aligned} \tag{2-2}$$

Data used for experimentation

For experimentation (see chapter 6) a mouse kidney with a Staphylococcus bacterial infestation is used. In Figure 4-1a the total ion count of every pixel is given. The shape of the kidney is visible, the two lighter areas show where the bacterial colonies are located.

The infected mouse kidney was obtained from the Skaar Laboratory (Vanderbilt University), snap-frozen at -80 °C, and cryo-sectioned at 10 μm thickness, using a CM3050 S cryostat. The tissue was thaw-mounted onto a conductive indium tin oxide-coated glass slide. DAN in a solution of 90% acetonitrile was sprayed at a surface density of 3.6 $/mm^2$ at 85 °C; 1:1 ratio of carbonate buffer (pH 10.3) and 250 mM sodium acetate in a solution of 30% methanol sprayed at a surface density of 6.8 μg/mm2 at 85 $\circ$C.

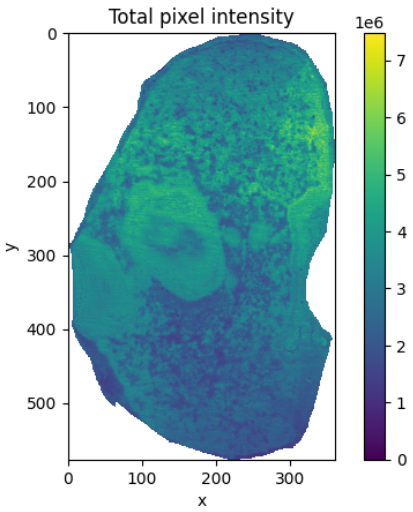


Figure 2-3: Total ion count, all intensities measured in one pixel summed

The mouse kidney data was acquired using the timsToF fleX (MALDI) mass spectrometer (Bruker Daltonik, Bremen, Germany) in Quadrupole Time Of Flight (QTOF) mode of operation. Tissue imaging data (161,547 pixels) were collected at 15 μm pixel size using 500 shots per pixel and 53% laser power in negative ionization mode from m/z 400 to 1,400.

The data was peak integrated, resulting in 573 mass bins for every pixel. The resulting peaks are not equidistant. The average distance between peaks is 1.2 Dalton, but Figure 2-4 shows that the peaks are not equally spaced. During experimentation the spacing between mass bins is not considered, i.e. no adjustments are made to compensate for unequal distance between bins.

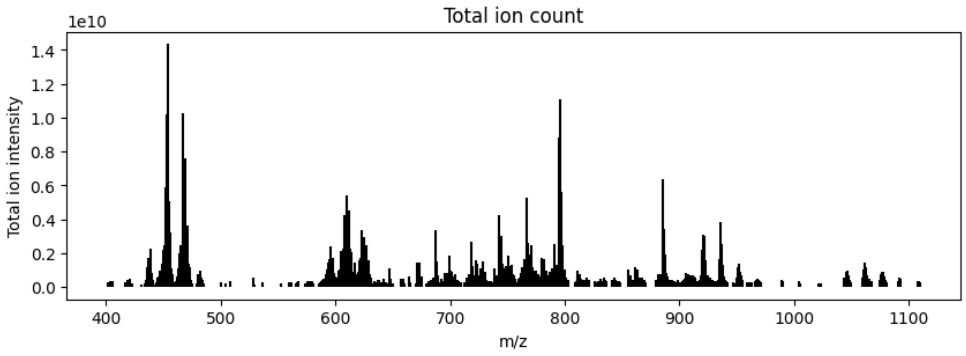


Figure 2-4: Total ion intensity for every mass bin, on the y-axis the intensity and on the x-axis the mass to charge ratio.

2-2 GANs

Generative Adversarial Nets (GANs) is a relatively novel way to learn the underlying distribution of data. As the name suggests, multiple neural nets are pitted against each other. The

basic GANs consist of two neural networks; a generator network maps a noise vector to a generated sample and a discriminator network judges if a sample is either real or generated. By updating both networks, eventually, the generator can generate samples indistinguishable from the original data.

In this section, GANs are introduced in two parts:

- In subsection 2-2-1 is explained how GANs work. The original adversarial loss function as well as different extensions and modifications are introduced.
- In subsection 2-2-2 a few examples of how GANs are used in practice are given. This includes the

Additionally, in subsection 2-2-3 different validation methods used for the quality control of GANs are introduced.

2-2-1 GANs basics

GANs were first introduced in 2014[15]. Like most variations of GANs, the original GANs use a generator function to map noise to a generator output and a discriminator function (sometimes called a critic function[16]) that tries to distinguish between real and generated samples. The goal of the generator function is to model the data distribution. What sets GANs apart from most other machine learning-based modeling techniques such as auto-encoders is that a learned function provides the error.

Figure 2-5 shows the layout of the basic GANs data flow. Noise with a distribution p_z is used as input to a parametric generator function. The generator outputs generated samples with distribution p_g. The discriminator predicts if its input is real i.e. from p_{data} or generated i.e. from p_g

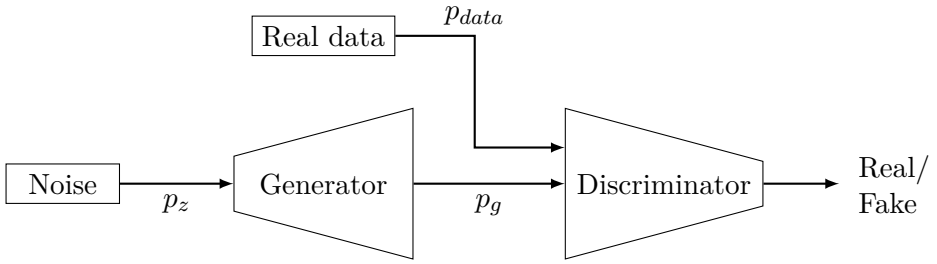


Figure 2-5: Scheme of basic GANs-framework

To update the two parametric functions the minimax-optimization scheme is given in (2-3). Note that in this work the parameters of the generator and discriminator functions are left out for readability. e.g. Optimizing over the parameters of G means optimizing the parameters θ_G but is denoted as optimizing G.

$$\begin{aligned} &\min_G \max_D V(G,D) \\ &V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_z}[\log(1-D(G(\mathbf{z})))] \\ &\qquad = \mathbb{E}_{\mathbf{x}\sim p_{data}}[\log(D(\mathbf{x}))] + \mathbb{E}_{\hat{\mathbf{x}}\sim p_g}[\log(1-D(\hat{\mathbf{x}}))] \end{aligned} \tag{2-3}$$

In (2-3) $\mathbf{x}$ denotes an original sample from the original data distribution p_{data}, $\mathbf{z}$ denotes a noise realization from the noise distribution $\mathbf{z}$ and $\hat{\mathbf{x}}$ a sample generated from $\mathbf{z}$ i.e. $\hat{\mathbf{x}} = G(\mathbf{z})$. This expression of the adversarial learning procedure is based on binary cross-entropy, a common optimization statement used for binary classification based on regression. The task of the discriminator can be seen as a binary classification between real and generated samples. A sigmoid function is used as the output of the discriminator function, meaning that $D(\cdot) \in (0, 1)$.

The min-max problem is considered optimal if changing G or D does not increase or decrease the score of $V(G, D)$. The optimal solution for this problem is a Nash equilibrium, denoted $V(G^*, D^*)$. Formally, a Nash equilibrium can be written as in (4-4)[17]. In other words, if the generator is optimal any discriminator will result in a lower value than the value from the of the optimal discriminator, and if the discriminator is optimal any not optimal generator will result in a higher function value.

$$V(G^*, D) \leq V(G^*, D^*) \leq V(G, D^*) \tag{2-4}$$

Finding the theoretical optimum of (2-3) is fairly straightforward. First, an expression for the optimal discriminator is found by writing out the expectations and using a change of variables. Note that here is assumed that the resulting generated samples $\hat{x}$ are optimal and therefore one expectation over both original and generated samples is the same as the two separate expectations. From this, the derivative over the samples is set to zero resulting in an expression for the optimal discriminator. (2-5)

$$\begin{aligned} V(G, D) &= \int_x p_{data}(x) \log(D(x)) dx + \int_{\hat{x}} p_g(\hat{x}) \log(1 - D(\hat{x})) d\hat{x} \\ &= \int_x p_{data}(x) \log(D(x)) + p_g(x) \log(1 - D(x)) dx \\ \frac{d}{dx} V(G, D^*) &= \frac{d}{dx} \int_x p_{data}(x) \log(D^*(x)) + p_g(x) \log(1 - D^*(x)) dx = 0 \\ \Rightarrow D^*(x) &= \frac{p_{data}(x)}{p_{data}(x) + p_g(x)} \end{aligned} \tag{2-5}$$

The optimal generator G^*, in theory, produces samples indistinguishable from the original samples, i.e. $p_g = p_{data}$. Therefore, from (2-5) it follows that $V(G^*, D^*) = -log(4)$. This can be interpreted as the discriminator not being able to distinguish original from generated data and predicting $\frac{1}{2}$ on average. The function value $V(G, D)$ reaching $-\log(4)$ does not mean the global optimum is reached; a discriminator that can't distinguish between real and generated samples will return these values no matter the quality of the samples.

Additionally, it is assumed that the generator can learn to accurately replicate the distribution p_{data}. However, this means that the generator should have the capacity to learn this data distribution. A simple example of how this capacity could be a problem is if the original data is standardized, meaning that approximately 98% will be within the range of (-3,3), but the output of the generator function is determined by a tangent hyperbolic function all values outside of (-1,1) range will never be generated.

In general, for GANs can be stated that a Nash equilibrium might not exist, which is proven (by counterexample) for multiple different GANs[18].

conditional GANs

The GANs framework can easily be extended to include class labels. This was first introduced by the conditional GANs (cGANs)[19]. In this case, both the generator and discriminator are conditioned by a label. The noise input to the generator is extended with an encoded class label, commonly implemented using one-hot encoding or an embedding layer (the latter is used in this work). Similarly, the labels are added to the original and generated samples as input to the discriminator. In Figure 2-6 the schematic of the cGANs is given. Note that the class label is used as an input to the discriminator and is not given as an output. In contrast to e.g. ACGAN (see subsection 2-2-2), cGANs can not be used for classification directly.

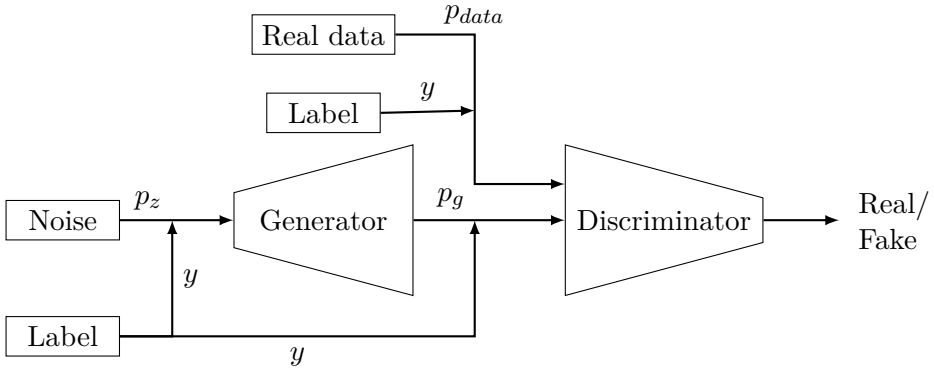


Figure 2-6: Flow diagram of conditional Generative Adversarial Networks (cGANs).

The original GANs optimization given in (2-3) is extended by using the class labels as inputs to the generator and discriminator function as seen in (2-6). The main benefit of using cGANs over the original GANs is the ability of the GANs to distinguish between different modes of labeled data. First and foremost, this helps in training the GANs by allowing for mode (class) specific learning. Secondly, cGANs allow for class-specific data generation; by giving a specific class label to the generator a sample supposedly belonging to that class will be generated. This is necessary for the minority class oversampling as intended in this work.

$$\begin{aligned} &\min_G \max_D V(G,D) \\ &\text{With } V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x}|\mathbf{y})}[log(D(\mathbf{x},\mathbf{y}))] + \mathbb{E}_{\mathbf{z}\sim p_z(\mathbf{z}|\mathbf{y})}[log(1-D(G(\mathbf{z},\mathbf{y})))] \end{aligned} \tag{2-6}$$

In (2-6) the conditional expectations are denoted $\mathbb{E}_{\mathbf{q}\sim p_Q(\mathbf{q}|\mathbf{y})}$ i.e. the expectation of sample q is being from the distribution of Q conditioned on y.

Alternative adversarial loss

Optimizing the generator and discriminator function can be a tedious process. The original GANs have a major downside; the quality of the update steps is dependent on the quality of the discriminator. If the discriminator is insufficiently trained the function value of $V(G,D)$ is not useful for updating the generator. If the discriminator is near perfect, however, the function $V(G,D)$ goes to zero meaning that using gradient descent-based methods does not update either function sufficiently and the training progression comes to a halt.

A solution to this problem is training the discriminator for multiple steps with the hope of the discriminator staying slightly ahead of the generator. However, setting a hyperparameter to sufficiently train the discriminator without it becoming too far ahead and the value of $V(G, D)$ going to zero, is difficult as the convergence rate changes during training depending on the performance of the adversary.

The problem described above can be solved by using the Wasserstein Generative Adversarial Nets (WGANs)[16]. Using the WGANs architecture and optimization scheme the discriminator can be trained for an unlimited number of steps without the training becoming unstable or stopping (e.g. $V(G, D)$ going to zero). As written in the publication and motivated by studying the equations below, training the discriminator to optimality provides a better gradient for the generator update, and therefore the discriminator should be trained to optimality before every generator step.

The WGANs optimization function is motivated by the Wasserstein distance measure. The original GANs optimization is related to the Jensen-Shannon (JS) divergence. The JS divergence measure becomes constant if the distribution of the real and generated data distribution does not intersect[16].

A more promising distance metric is the Wasserstein distance also referred to as Earth Mover (EM) distance. EM refers to how much it costs to move all parts of one distribution to the other. This distance metric is defined as follows:

$$W(\mathbb{P}_{data}, \mathbb{P}_g) = \inf_{\gamma \in \prod(\mathbb{P}_{data}, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \tag{2-7}$$

Where $\prod(\mathbb{P}_{data}, \mathbb{P}_g)$ is the set of all joint distributions $\gamma(x, y)$ whose marginals are $\mathbb{P}_{data}$ and $\mathbb{P}_g$. To evaluate the Wasserstein distance effectively, some derivations are needed. Using the Kantorovich-Rubinstein duality the Wasserstein distance can be written as a supremum under the condition that $f(\cdot)$ is a Lischitz-1 function (denoted $||f||_L \leq 1$)

$$W(\mathbb{P}_{data}, \mathbb{P}_g) = \sup_{||f||_L \leq 1} \mathbb{E}_{x\sim\mathbb{P}_{data}}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \tag{2-8}$$

To enforce the Lipschitz constant in the given optimization problem all weights should be in a constraint set.

$$\max_{w \in W} \mathbb{E}_{x\sim\mathbb{P}_{data}}[f_w(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f_w(x)] \tag{2-9}$$

To enforce weights being in this set, the weights are clipped to [-c,c]. c Here is a hyperparameter that should be chosen before training. To implement the EM first the output layer of the Discriminator is changed from a sigmoid function to a linear function. This changes the range of the output from $(0, 1)$ to $(-\infty, \infty)$.

In the paper is pointed out that clipping the weights is not a good way to enforce the Lipschitz constraint. Also, choosing bounds that are too large might again result in a vanishing gradient, and choosing bounds too small can lead to slow or even no convergence. **wGANs-gp!** (**wGANs-gp!**) [20] proposes to add another term to the loss function to act as a penalty function. Penalty functions are a common way in (nonlinear) optimization to enforce constraints[21].

$$L = \min_{\theta_G} \max_{\theta_D} \quad \underbrace{\mathbb{E}_{\mathbf{z}\sim\mathbb{P}_z}[D(G(z))] - \mathbb{E}_{\mathbf{x}\sim\mathbb{P}_{data}}[D(x)]}_{\text{Wasserstein Loss}} + \underbrace{\lambda\mathbb{E}_{\mathbf{y}\sim\mathbb{P}_y}[(||\nabla_{\mathbf{y}} D(\mathbf{y})||_2 - 1)^2]}_{\text{(Weighted) gradient penalty}} \tag{2-10}$$

Where $\mathbb{P}_y$ is sampled uniformly along straight lines between pairs of points of $\mathbb{P}_g$ and $\mathbb{P}_{data}$. λ is the penalty coefficient in the paper $\lambda = 10$ is used. Literature empirically shows that using this penalty function results in faster convergence than the original WGAN on multiple example datasets[20].

Deep convolutional GANs

GANs can take a long time to converge or even not converge at all[22].Some of the issues occurring during training can be reduced using e.g. the Wasserstein GANs presented in section 2-2-1. Instead of changing the optimization, additional attention to the generator and discriminator function can also increase stability and convergence rate. The paper on Deep Convolutional Generative Adversarial Nets (DC-GANs)[23] provides multiple guidelines for creating better-suited neural networks. The largest change is using strided convolutions for the discriminator and fractional strided convolution layers instead of the fully connected layers used in previous works. So-called Deep Convolutional GANs (DC-GANs) form the basis of many different commonly used GANs such as StyleGANs[24, 25, 26].

Next to the use of convolutional layers, the use of batch normalization and LeakyReLU activation layers is recommended. Even though DC-GANs show fast convergence in different publications no guarantee can be given that the DC-GANs perform better than other GANs. DC-GANs[23] uses the original optimization function given in (2-3), however, the DC-GANs network architectures can be used with different optimization schemes, such as the WGAN optimization presented in section 2-2-1.

2-2-2 Applied GANs

The GANs on itself can be used for generating data, by using the trained generator, or for anomaly detection, by using the trained discriminator. This work mostly focuses on the data generation of GANs, but with different modifications, GANs can be used for other tasks such as clustering or classification. In the following sections, some background information is given for applying GANs directly for downstream tasks.

Classification with GANs

In this work, the focus is on generating data and looking at the effect of adding this data to a dataset used with an external classifier. Instead of using an external classifier, GANs can be used for classification directly. Classification is a well-known example where machine learning excels[27]. The discriminator of GANs on itself already is a classifier. It checks if the sample belongs to the "real class" or the "fake class". If the discriminator is sufficiently trained on specific samples it can act as a classifier directly, this principle is for example used for anomaly detection (section 2-2-2).

Auxiliary Classifier Generative Adversarial Nets (AC-GAN)[27] extend the original GANs by using the discriminator as a classifier. Similar to conditional Generative Adversarial Nets (cGANs) (see section 2-2-1) the Generator takes a class label as input resulting in a fake sample $\hat{x} = G(z, y)$ where z is the input noise and c is the class label. In contrast to cGANs,

AC-GAN does not provide labels to the discriminator. Instead, the discriminator outputs a label (as well as the real/fake prediction) as shown in Figure 2-7. This is implemented using a Softmax activation function on the final layer of the neural network, parallel to the sigmoid activation (used for real/fake classification).

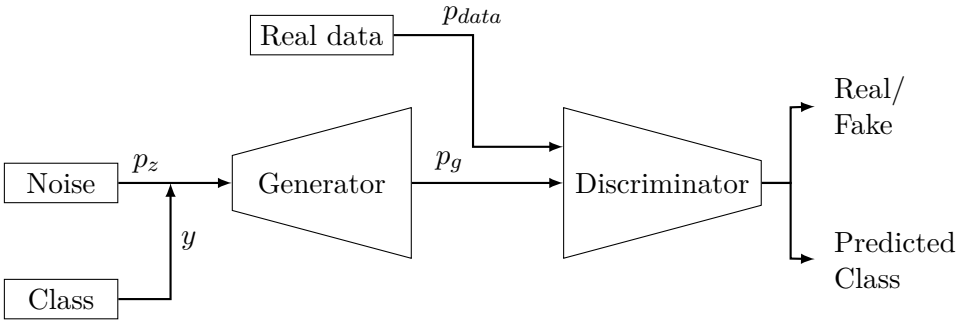


Figure 2-7: Flow diagram of AC-GAN

In addition to the changes to the layout as shown in Figure 2-7, additional changes are made to the optimization problem to include a term for the classification performance. Specifically, the generator aims to maximize $L_c - L_s$ and the discriminatory aims to minimize $L_c + L_s$. Where L_c is the classification-loss given in (2-11) and L_s given in (2-12) is the adversarial loss similar as seen before in (2-3) and (2-6). (2-11) contains the expectations of the discriminator predicting the correct class y.

$$L_c = \mathbb{E}_{y\sim p_y} log(D(x)) + \mathbb{E}_{c\sim p_y} log(D(G(c,y))) \tag{2-11}$$

$$L_s = \mathbb{E}_{x\sim p_{data}(x)} log(D(x)) + \mathbb{E}_{z\sim p_z(x|y)} log(1 - D(G(c,y))) \tag{2-12}$$

The intended task for AC-GAN is condition data synthesis, where the auxiliary classification task aims to better replicate the full data distribution. What makes the AC-GAN useful for classification is the data efficiency; the number of labeled samples can be relatively limited [28]. The original AC-GAN suffers from stability issues during training, especially with a higher number of classes. Some improvements have been made to the AC-GAN, for example, an extension to the optimization function is made resulting in more variance in the generated samples[29].

Clustering with GANs

Bidirectional Generative Adversarial Nets (BiGAN)[30] extends the normal GANs framework by adding an encoder that encodes the input sample x to a latent space z. In Figure 2-8 the BiGAN architecture is shown. The encoder E should learn to invert the generator G. The learned latent representations z can be used as labels.

The original minimax formulation given in (2-3) is changed to include the encoder:

$$\min_{\theta_G,\theta_E} \max_{\theta_D} V(D(\theta_D,\cdot), E(\theta_E,\cdot), G(\theta_G,\cdot))$$
$$\text{With } V(D,E,G) = \mathbb{E}_{x\sim p_{data}}[\underbrace{\mathbb{E}_{z\sim p_E(\cdot|x)} logD(x,z)}_{logD(x,E(x))}] + \mathbb{E}_{z\sim p_z}[\underbrace{\mathbb{E}_{x\sim p_G(\cdot|z)} log(1-D(x,z))}_{log(1-D(G(z),z))}] \tag{2-13}$$

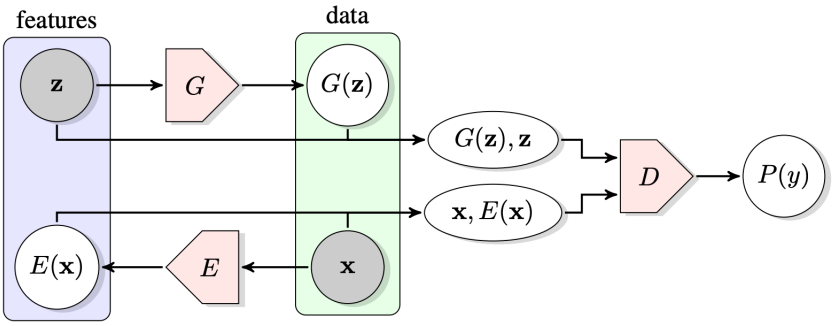


Figure 2-8: flow diagram of BiGAN (from [30])

Where $\mathbb{E}_{z \sim p_E}$ is the expectation of z being either the original z or the output of the encoder $E(x)$ and $\mathbb{E}_{x \sim p_G}$ is the expectation of x being either generated ($G(z)$) or a real sample (x). The discriminator uses both the latent input and an example (real or generated data) as input as shown in Figure 2-8.

BiGAN is compared to other methods for unsupervised classification on multiple different datasets. In the tests presented, it outperformed other methods such as k-means clustering.

Other methods similar to BiGAN are InfoGAN[31], Adversarial Learned Inference (ALI)[32] and ClusterGAN[33].

Anomaly detection with GANs

Detecting anomalies is a challenging and important problem, faced in many fields a.o. in manufacturing [34, 35], medical diagnosis[36, 37] and fraud detection[38]. Anomalies can be defined as; “examples that do not follow the general pattern present in the dataset”[36].

GANs can be used for unsupervised anomaly detection directly i.e. without changing the framework or optimization. By training GANs on only non-anomalous data, the discriminator can be used as a binary classifier to detect anomalies. A successfully trained discriminator will predict anomalous samples to be fake or generated.

Different studies use a modified GAN architecture to increase further the effectiveness of detecting anomalies. For example, AnoGAN[39] uses the original framework but extracts the features of the final hidden layer of the discriminator to show where anomalies are on an image. For extracting these features an additional step is needed for every new sample, making AnoGAN a time-consuming method for anomaly identification.

Other variations of GANs for anomaly detection include GANomaly[40] and EGBAD[41]. Both of these architectures use the approach of the bidirectional GANs described in section 2-2-2. EGBAD performs better in terms of accuracy than GANomaly on multiple different datasets[42].

In Appendix B an anomaly detection experiment is included that uses the original GANs. This experiment shows nicely how a trained discriminator can be used to detect anomalous spectra.

2-2-3 GANs performance indicators

In machine learning, the optimization function, usually called loss function, can be a good indicator of how well the algorithm is doing. For example, binary cross entropy (BCE) or mean squared error (MSE) will decay towards zero as the algorithm becomes better. With the adversarial loss, there is no such luxury, as the loss or optimization value is dependent on the performance of the discrimination. As seen in subsection 2-2-1 the theoretical optimum of the GANs either is not reached or the optimal value is reached even though the generator is not optimal. In literature is spoken about interpretable loss i.e. the value of the loss is no direct indicator of performance[17].

Instead of looking at the loss function for performance, GANs are usually validated after training by multiple different performance indicators. The performance indicators can be split into two groups, quantitative measures that put a value on the performance, and qualitative measures that depend on user experience. A nice overview of methods used for GANs validation is given in two papers[43, 44]. Some of these performance indicators are also used to check training progression and determine stopping criteria. For example by generating samples over time and seeing how these samples become more recognizable (by human experts) over time or by calculating a quantitative indicator such as the Inception Score at every (few) iteration(s) to show the trend of this score.

Qualitative validation

Qualitative methods for validating the generated output rely on user experience. For example, in images, especially natural images, the generated samples should be similar to the real thing to the human user. Many different ways of qualitative validation methods are used, most notably:

- Turing test variations; used by [45]
- Rapid scene categorization, where individuals are briefly shown real and generated samples and have to decide which is which[46].
- Preference Judgement, where individuals rank different generator models based on generated output. (Comparing different synthetic images without real images)[43].

The methods above are primarily useful for natural images. For other types of data, the downside of needing experienced users becomes more and more difficult, as the distinguishable features might not be as apparent. Additionally, qualitative methods can fail to detect overfitting or mode collapse, i.e. a user might not notice that generated samples have little to no variance.

For IMS data, looking at comparing real and generated spectra can give a first indication of the quality of the generated samples. For example, in some experiments, it is very clear that the generated samples have very little variance i.e. all generated spectra look the same while in comparison the original spectra show some variation between the samples.

To further help with qualitative validation, next to plots of the spectra a plot is generated with the mean of the spectra (original and generated) with around it a shaded area of three

times the standard deviation. Figure 2-9 shows such a result as an example. Here it is clear that the mean of the generated data follows the same trend as the real data, however, the standardization of the generated spectra is limited, meaning the generator is overfitting towards the mean of the data. Note that in this image, and other in images of this kind, these spectra are plotted as a continuous line for ease of inspection. The values the interpolated lines take between mass indices should not be assigned any meaning.

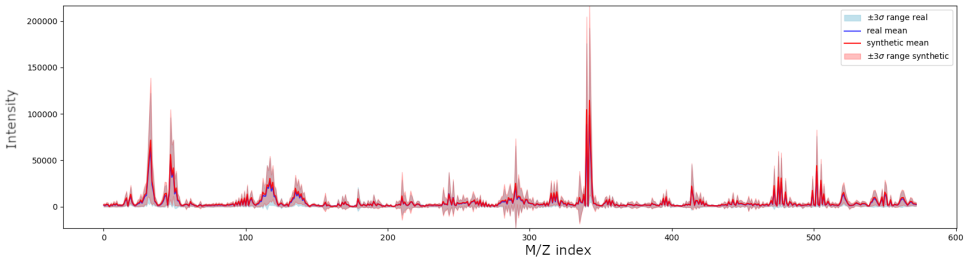


Figure 2-9: Example of a spectral plot used for quality control, here the mean and 3 times standard deviation of both the original (test) and generated data are given. The data is generated and scaled back to its original scale. On the x-axis the mass bin index, and on the y-axis the intensities in the original scale

Quantitative validation

Especially for comparing different GANs or different discriminator and generator architectures, a quantitative evaluation of the generated data is necessary. The most common methods of quantitative validation seen in the literature are the Inception Score and Fréchet Inception Distance (FID). Those two methods rely on a pre-trained neural network (Inception) for image recognition. In this work, the focus is on spectra meaning that the Inception-based scores can not be used. However, taking inspiration from the FID, the Fréchet Distance (FD) can be used.

Another method very useful for validating GANs generated data is using classification performance as an indicator of data quality. This method uses an external classifier and compares classifier results. Below both the FD and classification methods are further explored. In chapter 6 these two methods are used in the experiments.

Fréchet distance

The FD is a distance measure based on the Wasserstein-2 distance for normal distributions. The FID measures this distance on an intermediate layer of the Inception model. This distance measure can be used on the features as well under the assumption that the data follows a (multivariate) normal distribution. This distance measure is on the data directly in literature as well[47].

The FD distance is the minimal l2 distance between two multivariate normal distributions $P = \mathcal{N}(\mu_X, \Sigma_X)$ and $Q = \mathcal{N}(\mu_Y, \Sigma_Y)$. Here μ and Σ are the mean and covariance respectively. Writing out this l2-norm given in (2-14) gives the derivation and definition given in (2-15). The full derivation and motivation of the FD measure is given in [48].

$$FD^2(P,Q) = \min_{X,Y} |X - Y|^2 \tag{2-14}$$

$$\begin{aligned} FD^2(P,Q) &= |\mu_x - \mu_y|^2 + |\sigma_x - \sigma_y|^2 \\ &= |\mu_x - \mu_y|^2 + tr(\Sigma_x^2 + \Sigma_y^2 - 2(\Sigma_x\Sigma_y)^{1/2}) \end{aligned} \tag{2-15}$$

Where:

$$\begin{aligned} P = \mathcal{N}(\mu_x, \Sigma_x), \mu_x \in \mathbb{R}^n, \Sigma_x \in \mathbb{R}^{n\times n} \\ Q = \mathcal{N}(\mu_y, \Sigma_y), \mu_y \in \mathbb{R}^n, \Sigma_y \in \mathbb{R}^{n\times n} \\ \Sigma_x = \sigma_x\sigma_x^\top \\ \Sigma_y = \sigma_y\sigma_y^\top \end{aligned} \tag{2-16}$$

This FD score is used in different ways. First, it is used as an online performance indicator to determine if the GANs are still improving. In Figure 2-10 an example is shown of the FD distance decreasing over time. On the horizontal axis are the iterations of the GANs and the y-axis shows the FD distance. This distance is calculated between the generated data and an unseen test set. Note that the scale of the data (standardization or otherwise as introduced in subsection 3-1-3) influences the value the FD takes. To compare between different runs, with e.g. other classes it can be useful to scale the data back before calculating the FD-distance. A plot over time is only used as an indicator of the progress of the GANs.

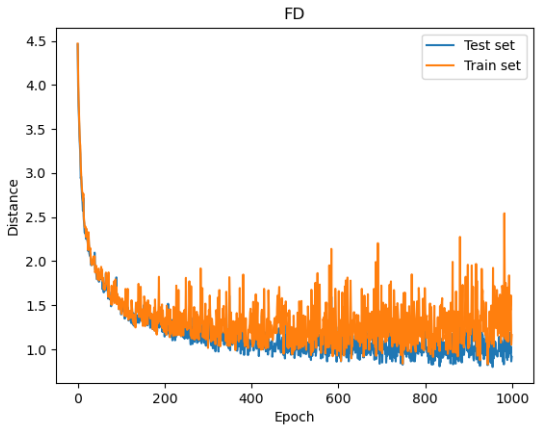


Figure 2-10: FD distance between generated and original samples

A second way how the FD is used is by looking at the scaled-back data and comparing the FD of the generated data with the FD of the original data. For example, looking at the distance between the original training data and original test data or looking at the training data of a specific class compared to the training data of another class. The intuition behind these two examples is that the generated data should ideally be as close to the test data as the training data. And that the distance between the generated data of a certain class should be closer to the original data of that class.

These results can be visualized in a matrix as shown in Figure 2-11. Here five classes are used and the FD is determined between test data and generated (fake) data. If the data is sufficiently well generated the expectation is that the the FD is low on the diagonal meaning that the generated distribution of a certain class is closest to the real distribution of that class.

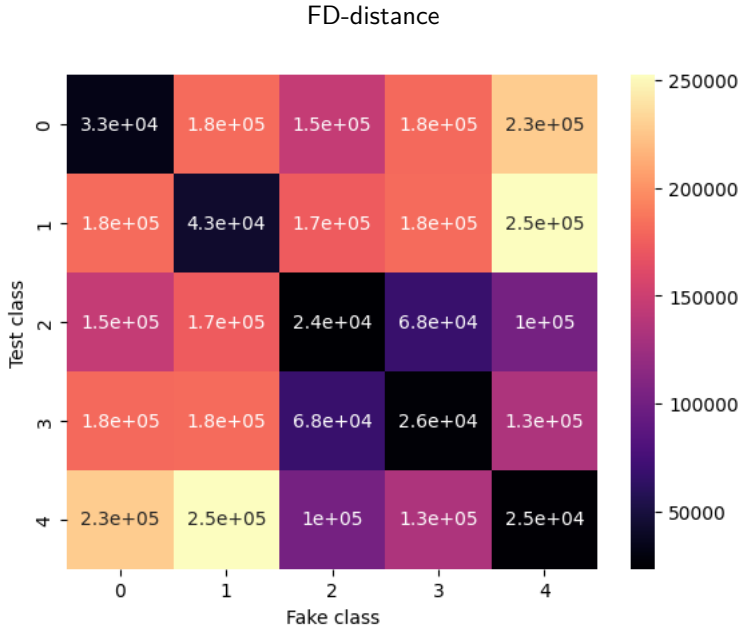


Figure 2-11: FD matrix of GANs trained on 50 NMF features of 5 majority classes. For reference, the FD between the total train and test set in this experiment is approximately 30.000. All values are based on data being scaled back to its original scale.

Classifier accuracy

Another method is using generated data to train a classifier and see how well the classifier does on original test data. This method is substantially different from the oversampling task; in this method, all original data in the training set is replaced by the training data. The resulting classification scores can be compared to classifier scores if it was trained on the original data.

An alternative way is to use a classifier fitted on the original data and a generated test set to see if the generated data is classified correctly. This approach has a major downside: if all data belonging to the same class is the same, i.e. no variance exists within the class. The classifier accuracy can be high as all data is classified correctly, however, the data is not useful for oversampling.

In the experiments, this method is used to study the effects of the dimension of the data and the effect of the number of classes. For a quick indication, one can look at the average recall score and the recall-confusion matrix. For a more detailed comparison between the resulting classifiers, one can look at the recall and precision scores for each class separately.

Chapter 3

Methods and data

The data used for the experiments in chapter 6 is already briefly introduced in subsection 2-1-2. In this chapter, the steps needed to use this data are explained. In section 3-1 the initial steps to prepare the given data for the experiments are given. After this, in section 3-2, the motivation behind oversampling as well as different oversampling techniques are summarized. As a final step, the used classifier is introduced in combination with methods to measure the accuracy of the classifier.

3-1 Preprocessing

The provided data consists out of a large data file and a file containing the semi-orthogonal NMF matrices. From these matrices, every original spectrum is assigned a cluster, henceforth referred to as a class. After assigning class labels and selecting a subset of the data to use for an experiment, the data is scaled such that all values are within a set range.

During experimentation, some results are based on a reduced number of features; a dimensionality reduction step is performed on the data between selection and scaling. The methods used for the labeling, dimensionality reduction and scaling are explained below.

3-1-1 Labelling

For the task of classification, labeled data is needed. The provided labels are created using semi-orthogonal-NMF. Semi-orthogonal NMF is a special kind of Non-negative matrix factorization (NMF) where the weight matrix of W has an orthogonality constraint: $WW^\top = I$. This constraint results in a clustering property, as every row of W has a single non-zero entry[49]. The index of this entry is the cluster where the sample corresponding to that row is assigned to.

NMF splits the data into a base-spectra matrix H with every row holds a spectrum representation. The weight matrix W has a weight of how much the corresponding base spectrum

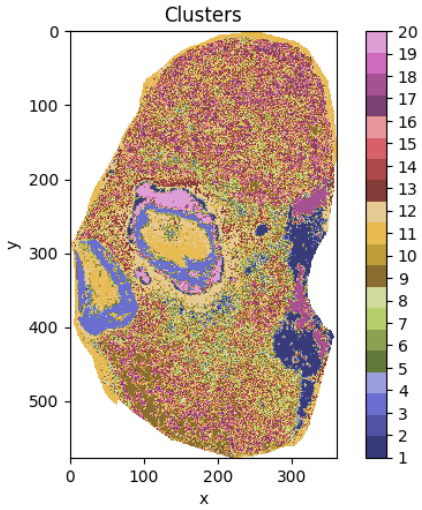


Figure 3-1: Clustered data

Class	#Samples	%	Class	#Samples	%
1	14056	8.70%	11	12937	8.01%
2	2343	1.45%	12	14463	8.95%
3	9485	5.87%	13	4523	2.80%
4	679	0.42%	14	3257	2.02%
5	18	0.01%	15	6416	3.97%
6	449	0.28%	16	722	0.45%
7	16147	10.00%	17	15149	9.38%
8	11969	7.41%	18	6118	3.79%
9	18443	11.42%	19	14591	9.03%
10	5955	3.69%	20	3827	2.37%

Table 3-1: Number of samples of each class

is represented in the original spectrum on every row. Adding the orthogonality constraint results in a H matrix containing several base spectra and a W matrix that assigns every original spectrum to a single base spectrum.

The spatial location of the resulting clustered spectra can be visualized by color coding each cluster as shown in Figure 4-1b. This figure shows some of the features present in the data such as the location of the bacterial colonies visible as two circular areas. In this image, it is clear that not every color (cluster) occurs in equal amounts. The number of samples (spectra) in each class is given in Table 3-1.

3-1-2 Dimensionality reduction

During experiments, the effect dimension of the samples is researched by changing the number of features. To reduce the feature space NMF is used to split the data into a W and H matrix where W contains a number of weights for every sample. These weights are used as a representation of the original spectra in a lower dimensional space. All analysis in these

lower dimension experiments is done on the weights and not the original mass bins. This means that the GANs learn to generate weights and the classification steps are also done on the weights.

The generated data can be scaled back to the original space by multiplying the generated samples by the H matrix. In this method there are two sources where the quality of the generated samples can be hindered; dimensionality reduction by NMF is not loss-less and NMF is not guaranteed to find the optimal solution.

The dimensionality reduction step is implemented using the Scikit-learn library[50]. The initialization method is set to 'nndsvd' (Nonnegative Double Singular Value Decomposition) as this was the standard in older library versions. The maximum number of iterations is set to 500 (default is 200).

3-1-3 Scaling

In literature, a lot of focus is on applying GANs to images with pixel values commonly in the range of 0 to 255. The original GANs[15] use a sigmoid generator output meaning that all values have to be scaled to a range of 0 to 1. DC-GAN suggests using a tangent hyperbolic output[23], giving a range of -1 to 1. The choice for the Tanh function over the sigmoid function is the steeper gradient around 0 this function provides, which allows for faster converging GANs.

To have all data in the range achievable by the generator, the data has to be scaled. This is commonly done by min-max-scaling[23]. However, as the IMS data at hand is not in the typical range of 0 to 255 but has many outliers with values well over 500.000 multiple scaling methods are considered. Below a short overview of the methods is given.

Similarly to the dimensionality reduction step, the data can be scaled back to have both the original data and generated data in the same scale. This scaling is with all classes used for the experiment at once. This means that for standard scaling a single class has not necessarily zero mean and unit variance for all mass bins.

Minmax scaling

Minmax scaling first reduces all values by subtracting the lowest value everywhere. Next, all values are divided by the largest difference in the data. This results in the data being in the range of 0 to 1. All data is then multiplied by 2 and 1 is subtracted to get the range -1 to 1. In (3-1) x_i is a single datapoint in the original dataset, $\tilde{x}_i$ denotes the scaled datapoint, and $min(X)$ and $max(X)$ are the smallest and largest values of the original dataset respectively.

$$\tilde{x}_i = 2\frac{x_i - min(X)}{max(X) - min(X)} - 1 \tag{3-1}$$

A downside of this scaling method is that all features are scaled by the same factor. In the case of the IMS dataset, the largest value is an outlier, which results in many already low values becoming very small even though these values might still hold important information.

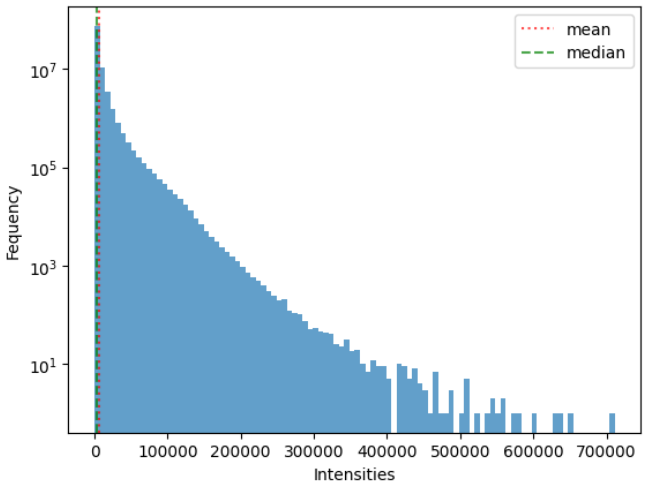


Figure 3-2: Distribution density plot of the intensity values. The mean and median are marked by vertical line. Note that the frequency axis is in logarithmic scale to ensure visibility.

Minmax scaling featurewise

One way to reduce the effect the min-max scaling has on the lower values is by applying the scaling featurewise. This means the scaling is applied to every mass bin separately, resulting in a scaled spectrum that looks more uniform and in which mass bins with low intensities hold a similar value to the original data. In (3-2) the subscript the argument m indicates the mass bin.

$$\tilde{x}_i(m) = 2\frac{x_i(m) - min(X(m))}{max(X(m)) - min(X(m))} - 1 \tag{3-2}$$

This method of scaling is less prone to the effect of outliers in the data because every outlier only effects a single massbin. However, a single massbin containing outliers might still be scaled with a large amount.

Standard scaling

A method that uses the minimum and maximum of the data and therefore is not hindered by outliers is standard scaling. Standard scaling scales the distribution to have zero mean and unit variance while keeping the distribution intact. Similarly to the featurewise min-max scaling, standard scaling is applied to each mass bin separately.

In (3-3) $M(X(m))$ is the mean of the data of mass bin m and $\sigma(X(m))$ is the standard deviation of the data corresponding to mass bin m.

$$\tilde{x}_i(m) = \frac{x_i(m) - M(X(m))}{\sigma(X(m))} \tag{3-3}$$

Figure 3-3 shows the effect of standard scaling. Note that the mass bins in the scaled plot are drawn from the middle, as the data is centered around zero.

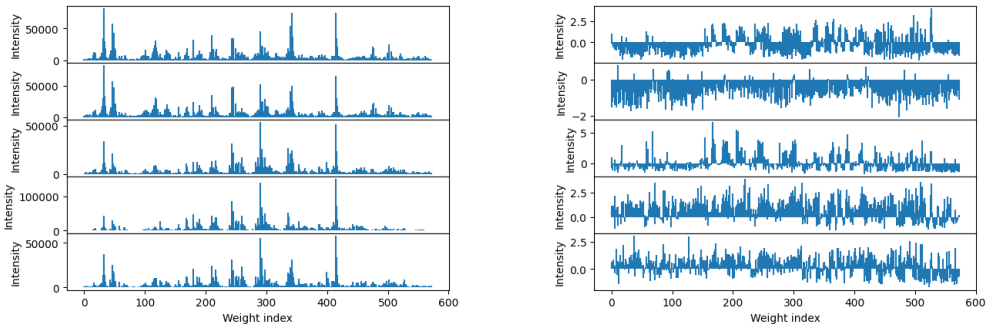


Figure 3-3: Example of standard scaling. (a) 5 original spectra, (b) the same spectra scaled using standard scaling.

3-2 Oversampling

GANs can be used for oversampling minority classes[1, 2]. In this work, using GANs for oversampling is compared with two other oversampling methods. The first method is oversampling by adding noise to the data. The second method is the Synthetic Minoirty OverSampling Technique (SMOTE), a method designed for oversampling based in interpolation between samples. Below, these two methods are briefly clarified.

3-2-1 Simple data augmentation

Data augmentation is a common technique to generate more samples without repetition[51]. The basic principle is to take a sample and add some noise to this sample to make it slightly perturbed in the hope of creating artificial variation within the dataset. A simple way to do this is by applying noise to the original samples to generate new samples. In the implementation, we use the mean of a class and add a random value based on the standard deviation. New samples denoted $\hat{x}_i$ are created by taking the vector mean of the data $X \in \mathbb{R}^{N \times n}$ denoted $M(X)$. To this mean vector p times the standard deviation of that mass bin. p is a random vector normally distributed with zero mean and standard deviation 1.

$$\hat{x}_i = M(X) + \mathbf{p}\sigma(X) \qquad \text{with } p \sim \mathcal{N}(0, I) \tag{3-4}$$

3-2-2 SMOTE

Synthetic Minority Oversampling Technique (SMOTE)[52] is a common framework for learning from imbalanced data[53] and is used in with IMS data as well[54, 55]. SMOTE interpolates neighboring samples of the same class. The algorithm can be summarized by the following steps:

1. For samples belonging to the same class, select one sample.
2. From this selected sample, find a number of the nearest neighbors.
3. The difference between the sample and each neighbor is multiplied with a random value between 0 and 1.

4. The resulting value is added to the original sample to generate a synthetic sample.

This process is repeated until the desired number of synthetic samples are created. In Figure 3-4 The steps given above are visualized: for sample Y_i four neighboring samples are selected denoted $Y_{i,1}$ through $Y_{i,4}$. On the imaginary lines between sample Y_i and its neighbors, a new sample is created at a random position defined by the random value w.

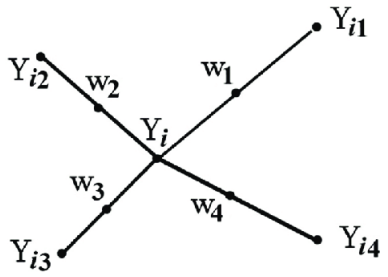


Figure 3-4: Visulatisation of SMOTE (from[56])

Numerous extensions and variations for SMOTE have been developed over the years[53]. In this work, only the original algorithm is considered. SMOTE is implemented using the imbalanced-learn Python library[57].

3-3 Classification

Classification is a task where an algorithm decides what subgroup of data a sample belongs to. In this work, we consider supervised machine learning based classification, i.e. a machine learning algorithm that learns to distinguish between subsets of data where the subsets are predetermined and used to create the decision variable. The goal of a classifier is to predict a class label of an unseen example, e.g. in this work a label based on a spectrum.

There is a distinction between regression models and classification problems, where regression assigns a continuous value and classification a discrete value. A regression example is the discriminator of the GANs that uses a sigmoid function to assign a value between zero and one. Putting a threshold value on top of this sigmoid function would make this a classifier. In this work, classification and regression are used synonymously.

3-3-1 Classifiers

There exist many different ways of building classification algorithms. A very common binary classifier is Logistic Regression (LR). This classifier is used in section 5-5 in accordance with the cited literature. LR differentiates between two subgroups of data but can not directly be used for multiclass classification.

A very popular method for classification using machine learning is the use of convolutional neural networks and other deep learning strategies. This is also applied to IMS data[58], however, using these deep learning models takes a lot of computational time. Therefore only simpler methods are used in this work. Specifically, for all experimentation except the benchmark Linear Discriminant Analysis (LDA) is used for classification.

3-3-2 Linear discriminant analysis

Linear discriminant analysis (LDA) is a method commonly used for classification in IMS[59, 58, 60]. LDA assumes that classes are linearly separable and that the covariance Σ is equal for every class.

The formulation of LDA is given in (3-5). Here Σ is the covariance matrix, y is the label. LDA computes the probability of sample x belonging to class k based on the distance of the sample to the distribution. More specifically the class asignment follows from finding the argument k that maximizes the posterior $P(y = k|x)$.

$$log(P(y = k|x)) = -\frac{1}{2}(x - \mu_k)^t \Sigma^{-1}(x - \mu_k) + log(P(y = k)) + Cst \tag{3-5}$$

LDA indeed works better for classification if a dataset is balanced[61].

3-3-3 Classifier accuracy

To verify how well a classifier does, different measures can be used. Below, a short overview is given of the different options including the downsides of some options. As this work is on multiclass classification the definitions are given for multiple classes with the notion of true and false positives and negatives as given in the confusion matrix Table 3-2. This table holds the information on the classifier performance for class 1. As in Table 3-2 the rest of this

		Prediction		
		Class 1	Class 2	Class 3
Truth	Class 1	TP	FN	FN
	Class 2	FP	TN	TN
	Class 3	FP	TN	TN

Table 3-2: Multi class confusion matrix for class 1

section describes correctly classified positive samples as True Positive (TP), and correctly classified negative samples as True Negative (TN). Negative samples incorrectly classified as positive False Positive (FP) and positive samples incorrectly classified as negative False negative (FP). All these values contain a number of samples such that the sum of the row elements is equal to the number of samples in that class.

Accuracy

Accuracy is defined as the number of correct predictions divided by the total number of predictions:

$$ACC = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} = \frac{TP + TN}{TP + TN + FP + FN} \tag{3-6}$$

From the accuracy score, it is not clear if the inaccuracy comes from false negatives or false positives.

Recall

Recall is defined as the number of correct true predictions (true positives) divided by the number of actual positives (sum of true positives and false negatives).

$$REC = \frac{\text{Correct predictions}}{\text{Sum of row}} = \frac{TP}{TP + FN} \tag{3-7}$$

The recall score does not consider the false positives i.e. recall is invariant under overfitting. For example, if a classifier predicts class 1 for all samples, the number of false negatives is zero and the recall score of class 1 is 100%. However, if the recall scores of all classes are considered, overfitting can be detected as all recall scores except for class 1 will be 0.

For a fast visualization of a multiclass classifier performance, the recall matrix can be used instead of the standard confusion matrix. An example of a recall matrix is given in Figure 3-5. The recall matrix is the confusion matrix where every entry is divided by the sum of the row (divided by the support i.e. the number of samples of that class). This way of looking at the confusion (recall) matrix circumvents a skewed view due to unequal samplings of each class; all values will be in the range of 0 to 1. A perfect matrix would have ones on the diagonal and zero elsewhere.

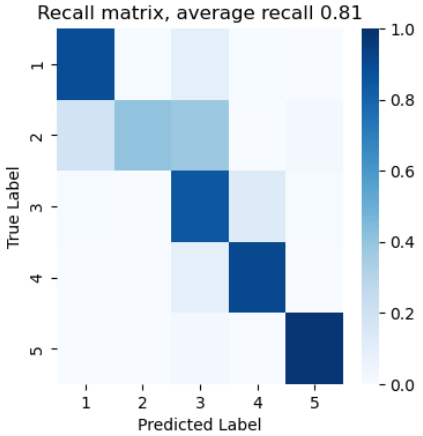


Figure 3-5: Example of a recall matrix: the optimal scores for all classes is 1 on the diagonal and zero everywhere else. This matrix shows good classification for all classes except for the class labeled two.

Precision

Precision is defined as the number of correct true predictions (true positives) divided by the number of wrong positive predictions (sum of false negatives).

$$PREC = \frac{\text{Correct predictions}}{\text{Sum of column}} = \frac{TP}{TP + FP} \tag{3-8}$$

Precision is used to detect overfitting to a specific class. For example, if all samples are predicted to belong to the same class the recall score will be high (100%). However, the precision score will be low as a result of the large number of false positive predictions.

F1-score

The recall and precision scores can be combined into the F1-score:

$$F1 = 2\frac{PREC \times REC}{PREC + REC} \tag{3-9}$$

The F1 score on itself is not sufficient as it gives no indication of the ratio of false negatives and false positives.

Use of classifier scores

In this work, the focus of classifier accuracy is on the recall score. Using the recall matrix a first glance estimate of how well the classification works is made. As a second part for verification, the precision score is used to detect if the classifier is not overfitting to a specific class.

Chapter 4

Paper

An investigation of generating IMS-spectra using GANs

Abstract

Imaging Mass Spectrometry is a technique that measures molecular mass distributions with respect to their spatial location. The resulting dataset contains a mass spectrum for every pixel. If divided into different classes, the number of spectra belonging to the same class can vary significantly, with for example thousands of spectra belonging to one class but only hundreds to a different class, thereby limiting the performance of classifiers. As IMS is destructive, generating additional original samples is not possible. The data imbalance problem therefore can be counteracted by generating synthetic samples belonging to the underrepresented class. A commonly used technique to generate additional samples is SMOTE.

Recently, generative adversarial nets (GANs) have been used instead of SMOTE for the oversampling of minority classes. Using GANs-based oversampling can result in better-performing classifiers than using SMOTE oversampling. GANs is a method of machine learning in which the combination of two functions tries to learn the distribution of data. The first function generates samples from noise, while the second function aims to distinguish these generated samples from the original samples. By updating the two functions the generated samples eventually should be indistinguishable from the original data.

In this work, conditional Wasserstein Generative Adversarial Nets with gradient panelty (cWGANs-gp) is implemented and tested in various ways on IMS data to oversample minority classes in a multiclass setting. This paper focuses on different experiments in an investigation of why working on full spectra is unsuccessful. By limiting the number of features (by dimensionality reduction) the implemented GANs can generate very similar data (based on classifier testing).

On the dataset used, using a lower number of features, our GANs can slightly increase spectral classifier (LDA) accuracy on minority classes with the downside that the classifier overfits

to the minority classes. SMOTE performs slightly better than the GANs, leading to the conclusion that using GANs to oversample minority classes in this IMS dataset is not useful. However, GANs might still hold great potential in other applications for IMS data such as anomaly detection or classification.

4-1 Introduction

Imaging Mass Spectrometry (IMS) is a technique that measures the mass distributions of a sample with respect to their spatial location. IMS allows for untargeted discovery of a large range of biomolecules. This has huge potential in e.g. drug discovery, and pathology research[6]. For example, IMS is used to study tumors[58] and enhances the development of new medicine by giving insight into the drug distribution within the tissue[5].

IMS data contains a mass distribution (spectrum) for every spatial location (pixel). Mass distributions contain the intensities of many different small mass ranges (mass bins), depending on the technology used the number of mass bins measured can go into the 100-thousands, for every pixel. This results in a data collection of multiple gigabytes.

Different methods are being developed to mine the considerable amount of information within an IMS dataset. Research goes into dimensionality reduction[62, 63] and machine learning methods[64] such as deep convolutional networks to classify tumors[58].

A relatively new method in machine learning is Generative Adversarial Nets (GANs)[15]. This setup of machine learning can be used for different machine learning tasks, for instance classification[65] and clustering[33]. What makes GANs apparently well suited for IMS data is the ability to learn from limited labeled data[66, 15]. To our knowledge, GANs have not been used with IMS data until now.

In this work, the generative ability of GANs is studied. To show that data generated by GANs is useful, generated data is added to a dataset to increase classification performance. More specifically, spectra generated with GANs are used for minority oversampling to lessen the effect of class imbalance. Resolving the problem of minority classes in IMS data by measuring more data is not possible due to the destructive nature of mass spectrometry, underlining the need for synthetic oversampling.

Class imbalance is further explored in the following subsection after which GANs are introduced. Only the basics of GANs theory are included in this paper, for more information please refer to the appendix or the cited literature.

In section 4-3 the dataset and different data processing steps are discussed. In this section, the evaluation metrics to be used in the experiments are introduced as well. During the design of the GANs multiple intermediate steps are taken, which are included in section section 4-4. After the design, multiple different experiments are conducted of which the results are given in section 4-5. That section contains increasingly more complicated experiments, the last of which is a test on the full dataset.

4-1-1 Synthetic minority oversampling

Class imbalance is a common problem in machine learning[67, 68, 69]. Especially in biological samples, where e.g. the rate between positive (healthy) and negative (diseased) samples can

be large.[66, 70]

A few different methods exist to decrease the negative effect of class imbalance. Undersampling majority classes throws out samples belonging to the majority class with the downside that the decrease in the number of samples results in a significant loss of data and possible loss of generalizability[71].

Instead of undersampling majority classes, oversampling minority classes is another way to reduce the limitations an imbalanced dataset poses. Oversampling means sampling from the minority class data distribution such that the number of samples is sufficient. To sample from this distribution, a simple data augmentation technique can be used. A bit more common is using Synthetic Minority Oversampling Technique (SMOTE)[72]. SMOTE creates new samples by interpolating between neighboring samples with the hope of preserving the original distribution without directly copying data.

SMOTE may add noise to the training data thereby only slightly increasing or even decreasing classifier performance[73]. Data generated by GANs can be used as an alternative to SMOTE-generated data. Multiple studies[1, 2] empirically show that adding data generated by GANs can outperform the SMOTE algorithm.

4-2 Background

This section introduces some prerequisite knowledge about GANs; subsection 4-2-1 goes into the basic mathematics behind the adversarial learning process and subsection 4-2-2 gives additional information on how GANs are used in practice.

4-2-1 Generative Adversarial Neural Nets

Generative Adversarial Nets (GANs) are a machine learning method that aims to learn a data distribution by trying to replicate this distribution from a noise distribution. GANs consist of (at least) two parametric functions, a generator function that maps a noise input to a generated sample, and a discriminator function that classifies a sample to either belong to the real or generated data. The quality of this binary classification is used to update both the generator and discriminator. Intuitively, if the classification of real and generated data is good, the generator can and should be improved by updating the parameters of the generator function. Conversely, the classification quality can be improved by updating the discriminator function.
The optimization of the two functions can be posed as the following minimax problem (4-1). (Note; abuse of notation, for readability the parameters are left out.)

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}}[\log(D(x))] + \mathbb{E}_{z\sim p_{noise}}[\log(1 - D(G(z)))] \tag{4-1}$$

Where $G(z)$ is a generated sample also denoted $\hat{x}$. $\mathbb{E}_{x\sim p_{data}}$ Is the expectation of the vector x belonging to the real data distribution and $\mathbb{E}_{z\sim p_{noise}}$ is the expectation of the vector z being from the noise distribution (equivalent to $\mathbb{E}_{\hat{x}\sim p_g}$; the expectation of $\hat{x}$ being from the generated distribution). The output of the discriminator should be between 0 and 1 where 1 will be a real sample and 0 a generated sample.

The minimax equation as posed in (4-1) can be explained as a distance metric in which the discriminator tries to maximize the measured distance between the real and generated distribution without influencing the distribution. The generator tries to minimize the distance by changing the generated distribution.

In conditional GANs (cGANs)[19]both the discriminator and generator function are conditioned using a class label. It is important to make the distinction between conditional GANs, that use the class label as input for both functions, and e.g. Auxiliary-Classifier GANs [65]that give the class label as a discriminator output and can be used as for classification directly. Note that in this work the focus is on generating data to use with an external classifier, designing an optimal classifier network, or using GANs for classification is not explored. An introduction to previous works that use GANs for classification or other tasks directly is given in subsection 4-2-2.

The optimization of the minimax function is implemented in two steps. With the suggestion found in literature[15] the second part of the equation is changed to a maximization step for the Generator to increase stability during training. Only the second part of the equation is influenced by the generator function, meaning that for optimizing the generator the first term of the equation can be left out.

$$\max_D \mathbb{E}_{x\sim p_{data}}[\log(D(x))] + \mathbb{E}_{z\sim p_{noise}}[\log(1 - D(G(z)))] \tag{4-2}$$

$$\max_G \mathbb{E}_{z\sim p_z}[\log(D(G(z)))] \tag{4-3}$$

The minimax game should end if the generated data exactly matches the original data distribution $p_{data} = p_g$. If the distribution of the real data equals the distribution of the generated data and the discriminator is perfect, the solution of the minimax objective function should be $2\log(\frac{1}{2})$ on average as the discriminator has no way of distinguishing two equal distributions. This result can intuitively be explained by even a perfect discriminator not being able to distinguish between original and generated data and therefore predicting $\frac{1}{2}$ on average. A derivation for this solution is given in subsection 4-2-1.

This optimal value is not likely to be reached, to due the difficulties during the training of the GANs and the finite capacity of the generator and discriminator function. In game theory the optimal value of a two-player game, a so-called Nash equilibrium, is defined as the value resulting that neither player changing their policy will result in a better score. Given the function $V(G, D)$ as the GANs minimax optimization function as (4-1) it can be stated that the Nash equilibrium (G^*, D^*) satisfies:

$$V(G^*, D) \leq V(G^*, D^*) \leq V(G, D^*) \tag{4-4}$$

Where D^* and G^* denote the optimal discriminator and generator respectively. The theoretical optimum is usually not reached and a Nash equilibrium might not even exist because of the finite capacity of both the generator and discriminator[18].

As pointed out in different papers[15, 16] the minimax problem as posed in (4-1) has an inherent problem if the generator and discriminator are not converging at the same rate, i.e. the generator starts to effectively trick or the discriminator or the discriminator distinguishes generated data from original data without fail. If the discriminator is perfect (or far ahead) the optimization function given in (4-3) goes to infinity.

To deal with this issue, the Wasserstein distance metric is used[16]. This distance metric is implemented with a gradient penalty[20]. The motivation for using the Wasserstein-gp metric for experimentation can be found in section 2-2-1 and subsection 6-2-1.

$$\min_G \max_D \mathbb{E}_{z\sim p_{noise}}[1 - D(G(z))] - \mathbb{E}_{x\sim p_{data}}[D(x))] + \lambda \mathbb{E}_{\mathbf{y}\sim \mathbb{P}_y}[(||\nabla_{\mathbf{y}} D(\mathbf{y})||_2 - 1)^2] \tag{4-5}$$

In (4-5) the discriminator function $D(\cdot) \in \mathbb{R}$ instead of $D(\cdot) \in (0, 1)$ as is the case in (4-1). $\mathbb{P}_y$ is sampled uniformly along straight lines between pairs of points of $\mathbb{P}_g$ and $\mathbb{P}_{data}$. λ is the penalty coefficient in the original paper $\lambda = 10$ is used. It is empirically shown that using this penalty function results in faster convergence than the original WGAN on multiple example datasets[20].

In contrast to the original GANs minimax equation (4-1), the discriminator in the WGAN can (and should) be ahead of the generator[16]. To achieve this, the generator is updated only every n_{dis} steps. The number of discriminator update steps per generator update can be tuned by looking at the accuracy of the discriminator predictions. In algorithm 1 the WGAN-gp algorithm is summarized. In this algorithm, the on gradient descent based optimizer Adam[74] is used.

Algorithm 1 WGAN-gp algorithm

Require: λ, n_{dis}
for i=1,...,epochs **do**
  for b=1,...,batches **do**
    if b MOD n_{dis}==0 **then**
      sample $z \sim \mathcal{N}(0, I)$, $y \sim \mathcal{Y}$
      $G \leftarrow \text{Adam}(\nabla_G \text{mean}(-D(G(z|y))))$
    end if
    sample $z \sim \mathcal{N}(0, I)$, $\epsilon \sim (0, 1)$, $(x, y) \sim \mathbb{X}_b$
    $\hat{x} \leftarrow G(z, y)$
    $\tilde{x} \leftarrow \epsilon x + (1 - \epsilon)\hat{x}$
    $D \leftarrow \text{Adam}(\nabla_D \text{mean}(D(\hat{x}|y) - D(x) + \lambda(||\nabla_{\tilde{x}} D(\tilde{x}|y)||_2 - 1)^2)))$
  end for
end for

4-2-2 Applied GANs

Research aimed at improving the GANs framework usually work on smaller images and standard datasets like MNIST[75] or fashion-mnist[76] as benchmarks[23, 16, 20]. As mentioned, alternative minmax formulations are suggested, such as using the Wasserstein distance (4-5). Changes to the algorithm such as unrolled GANs[77] or changes to the overall structure by adding a third function have been proposed[30, 78]. There seems to be no consensus on the best GANs in general, as this is dependent on the goal of the application.

GANs are used in many different applications in 2D images. Well-known examples are generating random high-resolution images[79] and mixing the style of images[80].

GANs research is not limited to "normal" 2D images, for example, GANs are used in radiology for a.o. image reconstruction and data augmentation[81]. Similarly, GANs are used to generate additional samples to account for imbalances in MRI data for brain disease diagnosis[82]. Specifically, it is shown that adding missing modalities generated by conditional GANs can improve multiclass classification better than classical data augmentation.

Additionally to images, MRI or otherwise, GANs also have been successfully applied for oversampling in tabular data. [2] show that binary classification can be improved by GANs for different datasets and for different classifiers. In comparison to SMOTE minor improvements are made on 3 benchmark datasets (with a varying number of features) as well as their own dataset.

It has been empirically shown that training using Wasserstein GANs for minority oversampling outperforms DC-GAN on a fraud detection dataset [83].

In this work, GANs are used on the spectra and not on the images that can be created by looking at a single mass bin. By studying the spectra conclusions can be made on, for example, diseased vs healthy tissue[60]. A different approach to generating IMS-data using GANs is looking at the spatial representations of separate mass bins. GANs might be used to generate a spatial representation of a non-existent mass bin, the exact use case for this method is not obvious to us but can be explored in a subsequent work. A starting point would be to look at GANs for high resolution[84].

4-3 Methods and data

The data used for experimentation is a mouse kidney. The 161.547 pixels (spectra) were collected at $15\mu m$ pixel size from m/z 400 to 1400. The data is peak integrated into 573 mass bins. A spatial representation of the data is given in Figure 4-1a, The total ion intensity per mass bin is given in Figure 4-2. More information on the provided data such as instrumentation is given in chapter 2.

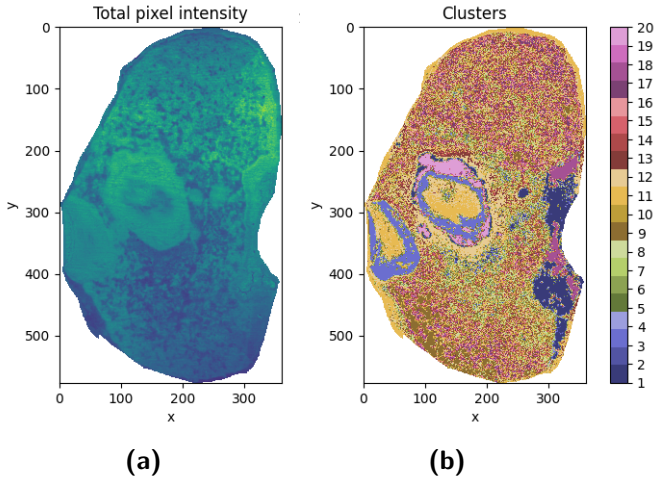


Figure 4-1: Spatial view of the dataset,4-1a the total ion count of every pixel, 4-1b: the location of the different spectra clusters.

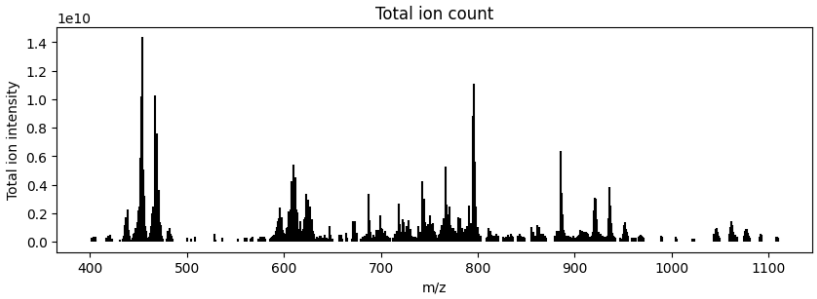


Figure 4-2: Total ion-count of the 573 mass bins with on the x-axis the mass/charge ratio. Mass bins are unequally spaced in the range of 401 to 1110 Dalton.

4-3-1 Preprocessing

Classification is simulated by clustering the dataset with semi-orthogonal Nonnegative matrix factorization (NMF). NMF splits a data matrix X into a feature matrix H and a weight matrix W, both with only positive elements. Additionally, W is orthogonal which means that every row of W holds exactly 1 nonzero value. The index of this value assigns the sample of the corresponding row to a cluster. In (4-6) the orthogonal NMF minimization problem is given.

The spectra are used as separate signals and will be referred to as the original data going forward. The generator is expected to eventually produce spectra that can be added to the training data to increase classifier performance.

$$\begin{aligned} &\min_{W,H} ||X - WH||_F \\ &\text{s.t. } W, H \geq 0 \\ &\quad WW^\top = I \end{aligned} \tag{4-6}$$

The clustering method and its effects on the clusters compared to other clustering methods are not further explored in this work. The cluster assignment deducted from the weight matrix is considered a class assignment to be used for a classification task. In Figure 4-1b the different labels are represented as colors in their respective spatial location. The choice of 20 different clusters is considered arbitrary.

The number of samples belonging to each class is given in Table 4-1. In this table, it is clear that this dataset is very unbalanced, with 20 classes every class should contain approximately 5% of the total number of samples. Classes represented with less than 5% of the samples are defined as minority classes. As can be seen in the results in section 4-5 the classes containing 3% or 4% of the total do not cause classification issues.

For training GANs and fitting the classifier, the data is split into a training and test set. At least 20% of the data of each class is used for testing. A maximum of 5000 samples is used for training. i.e. If more than 6000 samples are available, 5000 are used for training and all other samples are used for testing.

To study the effect of GANs a subset of the classes is selected in most experiments. After selecting the clusters to be used, the selected data is normalized. This normalization step is necessary to make the range reachable by the generator. In literature, the data is usually

Class	#Samples	%	Class	#Samples	%
1	14056	8.70%	11	12937	8.01%
2	2343	1.45%	12	14463	8.95%
3	9485	5.87%	13	4523	2.80%
4	679	0.42%	14	3257	2.02%
5	18	0.01%	15	6416	3.97%
6	449	0.28%	16	722	0.45%
7	16147	10.00%	17	15149	9.38%
8	11969	7.41%	18	6118	3.79%
9	18443	11.42%	19	14591	9.03%
10	5955	3.69%	20	3827	2.37%

Table 4-1: Number of samples of every class with the percentage of the total number of samples

scaled using minmax such that all values are between 0 and 1. Minmax-scaling has a huge downside if there are many outliers within the data. Where natural images usually have a pixel intensity between 0 and 255, the data at hand has values over 500.000. In (4-7), x_i is the i^{th} sample of the original data X and $\tilde{x}_i$ is the scaled sample.

$$\tilde{x}_i = 2\frac{x_i - min(X)}{max(X) - min(X)} - 1 \tag{4-7}$$

Instead of minmax scaling Equation 4-7, the data is standardized featurewise using (4-8). This means that for every mass bin (denoted argument m) the mean of that mass bin is subtracted from every intensity value and then divided by the corresponding standard deviation. Every mass bin therefore has zero mean and unit variance. Note that this is done on all the selected classes used in an experiment. Looking at a single class, the mean is not necessarily zero and the variance is not necessarily one.

$$\tilde{x}_i(m) = \frac{x_i(m) - M(X(m))}{\sigma(X(m))} \tag{4-8}$$

4-3-2 Quantitative evaluation

To verify the quality of the generated data and compare different GANs two different quantitative validation methods are used. Note that here only quantitative methods are given, qualitative tests such as HYPE[85] or Rapid Scene Categorization[46] used for natural images are much more difficult to obtain as looking at spectra is not as intuitive as looking at images. As a qualitative method, some plots of the generated and real distributions are given in the Appendix B.

Numerous quantitative validation methods used for GANs exist and are summarized in literature[43, 44]. In this work two different methods are used; the Fréchet distance and classification performance. These two methods are briefly introduced below.

Fréchet Distance

The first quantitative method is the Fréchet distance. This measure is often used in GANs[86, 87] on intermediate layer outputs of a pre-trained inception network (e.g. Inception v3[88]).

The Fréchet Distance (FD) can be used on the original data space as well[47]. The Fréchet distance is the L2 Wasserstein distance between two multivariate normal distributions. Note that the assumption is used that the data follows a multivariate normal distribution. The FD is defined as in (4-9)

$$\begin{aligned} FD(Q,R) =& \sqrt{||\mu_Q - \mu_R||_2 + Tr(\Sigma_Q + \Sigma_R - 2(\Sigma_R\Sigma_Q)^{\frac{1}{2}})} \\ & \text{With } Q \sim \mathcal{N}(\mu_Q, \Sigma_Q), \quad R \sim \mathcal{N}(\mu_R, \Sigma_R) \end{aligned} \tag{4-9}$$

Where Σ denotes the standard deviation and μ denotes the mean of the multivariate normal distributed datasets Q and R. The motivation and derivation of the FD measure are given in section 2-2-3. Note that the FD takes a positive value in $\mathbb{R}$, however, it only serves as a relative comparison as scaling the data drastically changes the value of the FD.

Classifier performance

As a second validation measure, classification performance is used. The intuition is that if GANs produce data very similar to the original data, replacing original training data with GAN-generated data should not hinder classifier performance.

To test this a Linear Discriminant Analysis (LDA) classifier is fitted on the original training data. A GAN network is trained on the same data. After GAN training the GAN produces an entire training set with the same number of samples per class as in the original training data. A new LDA is fitted to this GAN-generated training data. To compare the LDA fitted to GAN data and the LDA fitted to original data, a separate test set is used.

Note that this method can not be used effectively as the original LDA has limited performance because of e.g. class imbalance. To verify GANs in this way, a subset of majority classes is selected.

4-3-3 Baselines

To show the effect of minority oversampling using GANs, the classification method is also used on the original data (without oversampling denoted W/O) as well as SMOTE and a simple data augmentation technique. As simple data augmentation, a noise vector z scaled by the covariance matrix Σ_x is added to the mean μ_x of the original data to get a new sample $\hat{x}$.

$$\hat{x} = \mu_x + \Sigma_x z, \qquad z \sim \mathcal{N}(0, I) \tag{4-10}$$

For classification, LDA is used. Creating a better classifier using e.g. neural nets is not part of this research. LDA is commonly used for IMS data as a baseline classifier. Similarly to e.g. linear regression (LR), fitting the LDA is very fast compared to the training of Neural networks. Unlike LR, LDA can be used for multiclass classification directly.

As explored in literature[61], LDA indeed works better for classification if a dataset is balanced.

4-3-4 Hardware and code

All experimentation is implemented in Python and the main scripts are available on `github.com/WvdL1995/IMS_GANs`. The GANs algorithm is implemented using Pytorch[89], for SMOTE the imbalanced-learn[57] library is used. Other algorithms like LR and LDA are implemented using scikit-learn[90].

All experiments mentioned in this paper can be run on a single NVIDIA GTX 1050 TI with 4 GB of video memory. To expedite training some experiments were performed in parallel using an NVIDIA RTX A6000.

Training GANs on the full dataset as shown in section 6-1 took approximately six hours. The majority-only experiments ran in under two hours.

4-4 Design

Designing the generator and discriminator functions is an important step as the design dictates the capacity of what the model can learn as well. Additionally, the design of the functions has a huge influence of the stability of the adversarial learning process. In literature, the design decisions of both networks are not always explained and in most research, the standard number of layers and neurons as presented in e.g. the original GANs[15] are used.

Work that goes toward optimal generator and discriminator design such as alphaGAN[91] goes in the direction of automatic machine learning. Using the proximal duality gap the generator network is optimized for a specific discriminator. In this work, the auto-ML direction is not further explored. Instead, as a first step in designing GANs the capacity of the generator and discriminator are studied separately. Different architectures are compared to a baseline score and based on this score a decision on the architecture to use for experimentation is chosen.

4-4-1 Discriminator testing

To verify that the discriminator design has sufficient capacity, the neural network is used as a binary classifier. The intuition behind this test is that if the discriminator network can learn to distinguish between 2 different classes of spectra or between real data and noise from an untrained generator, it holds enough complexity that it can work as a discriminator during training. Different network architectures were compared in this way, leading to the conclusion that, of the tested architectures a network with 3 linear connections works best for this dataset.

4-4-2 Generator design

To test the generator capacity in a similar way to the discriminator the neural network is used as a decoder network that maps the noise vector z to an output signal. The reference output signal is generated by a function $f(\cdot)$ that takes the same input z as the decoder. The goal of this experiment is to minimize the difference between $G(z)$ and $f(z)$ as shown in (5-5). The function $f(z)$ can be any function, an experiment with a linear function is included in the

appendix. Assuming that a linear mapping of noise can not model the data, a second-order function is used for the analysis below.

$$\min_{G} ||f(z) - G(z)||_2^2 \tag{4-11}$$

The choice for $f(z) = Az^2 + Az + b$ comes from the assumption that if the data is from a multivariate normal distribution $\mathcal{N}(\mu, \sigma\sigma^\top)$ it can be described by a linear function $f(z) = \sigma z + \mu$ with $z \sim \mathcal{N}(0, I)$. Where $AA^\top$ is a lower rank approximation of the covariance matrix, calculated by the truncated singular value decomposition, of the data used in the experiments below. b Is the mean of the data.

Using this method, it is concluded that a fully connected network with 4 linear connections is best for this dataset. The results of this test are included in section 5-3. In further experiments, the generator design with 4 linear connections will be used.

4-5 Experimental results

Applying the GANs to the entire dataset does not give satisfying results. These results are given in section 6-1. To investigate why the training GANs on the full training set and generating minority classes does not work as intended, multiple smaller experiments are done below.

4-5-1 Benchmark

To test if the cGANs are correctly implemented, as a first step one of the benchmarks seen in literature[2] is tested with our design. This benchmark is a binary classification task on the Wisconsin Breast Cancer dataset[92]. This dataset contains 30 features for every sample. The results on this benchmark are given in section 5-5. In these results, we see that our design improves on the referenced results. Additionally, the results show the clear effect of different scaling methods on the baseline: If standard scaling is used instead of min-max-scaling the baseline classifier immediately performs significantly better even without oversampling.

From testing with this benchmark dataset, it is concluded that our implementation of the cGANs works as intended. Results seen in the literature can be replicated and improved upon.

4-5-2 5 majority classes

As a first step in testing the designed GANs on IMS data, the GANs are tested on 5 of the majority classes. This relatively low number of classes allows for faster testing of different configurations. Because all classes have a large and equal number of samples in the training set (set to 5000), adding generated data to classes does not give a clear insight into the quality of the generated spectra. Instead, the training data is replaced entirely following the quantitative verification technique described in subsection 4-3-2. As a baseline score, the original training data is used to fit the LDA classifier. SMOTE, in this case, replaces all

training data. Finally, conditional GANs and conditional Wasserstein GANs are trained on the training data after which the generators are used to regenerate the training set based on the training labels.

The results of this experiment is given in Table 4-2. From these results, it is clear that both the conditional GANs as well as the conditional Wasserstein GANs cannot adequately generate data. In search of the cause of the limited performance, the signal-to-noise ratio

	Baseline		SMOTE		cGAN		cWGAN-gp	
Class	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
1	0.92± 0.01	0.99±0.00	0.94±0.01	0.98±0.01	0.30± 0.06	0.28±0.06	0.22±0.05	0.19±0.02
3	1.00± 0.00	1.00±0.00	1.00±0.00	1.00±0.00	0.40± 0.05	0.15±0.03	0.38±0.04	0.10±0.01
7	0.88± 0.00	0.86±0.01	0.88±0.00	0.85±0.00	0.22± 0.09	0.24±0.01	0.07±0.04	0.26±0.01
8	0.93± 0.00	0.85±0.00	0.89±0.01	0.85±0.00	0.13± 0.04	0.16±0.06	0.19±0.07	0.16±0.01
9	0.97± 0.00	0.99±0.00	0.97±0.01	0.99±0.00	0.14± 0.11	0.25±0.12	0.13±0.02	0.31±0.02

Table 4-2: Data replacement results of training GANs on 5 majority classes. Scores collected over 3 runs.

was increased by applying dimensionality reduction and transforming the data back to the original data space. Using this method, no improvement was made. Therefore, it can be concluded that noise in the data is not the (self-standing) factor limiting the performance.

4-5-3 Influence of dimensionality

Besides noise, a different cause of the GANs failing can be the dimension of the signal. As pointed out in literature reducing the spectral dimension of hyperspectral images to only a few components leads to better performance than the same procedure with a larger number of components[93].

To study what the effect of the dimension of the features is on the GANs, different numbers of features are used in the same experiment using only 5 majority classes. The total (macro average) recall score resulting from selecting a different number of features is given in Figure 4-3. More information on this experiment is given in subsection 6-2-3.

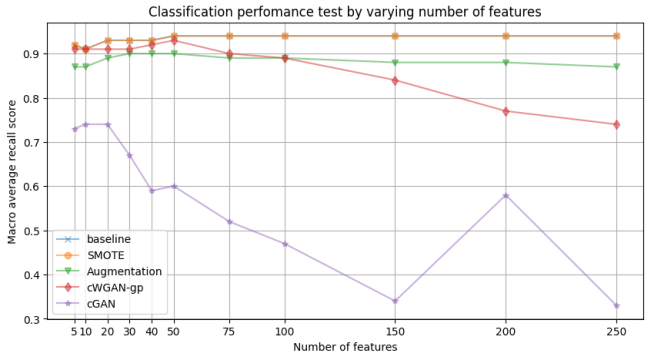


Figure 4-3: Classification performance by an increasing number of features

Note that the baseline (classifier based on original data) performs consistently well no matter the number of features used. The same holds for the classifier on SMOTE-generated data.

Conditional GANs perform significantly worse than other options. The conditional Wasserstein GANs performs well for a lower number of features, but the classifier performance starts to decay if the number of features is increased.

The result presented in Figure 4-3 suggests that the GANs model has more difficulty with more complicated data (i.e. larger number of features). Allowing for more complicated relations between features to be modeled by increasing the number of layers in both the discriminator and generator network does not improve this result. In subsection 6-2-4 the results from the same experiments are given using an additional layer in both networks. It is important to note that this difference is observed by applying the same hyperparameters i.e. the same number of iterations and the same learning rate.

4-5-4 Influence of number of classes

The results on the benchmark dataset given in section 5-5 are on binary classification. To study the behavior of the conditional GANs to be used for multiclass data generation, the number of classes is increased in a similar way as the number of features was increased in subsection 6-2-3. The full experiment setup can be found in the subsection 6-2-6.

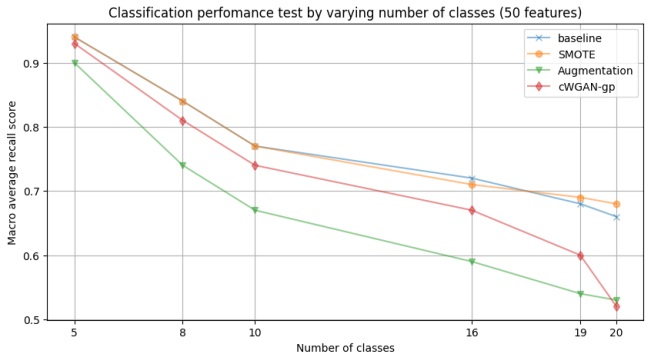


Figure 4-4: Increasing number of classes

From Figure 4-4 it can be seen that training GANs on an increasing number of samples does not drastically influence performance; the scores of the GAN-generated data follow the same trend of the baseline classifier up until above 16 classes. After 16 classes more classes with a relatively low number of samples are added.

4-5-5 Varying the latent space

The dimension of the latent space seems not that well studied. To test the effect of a different-sized latent space the size of the noise input n_z is varied and the resulting replacement scores are given in Table 4-3. A description of this experiment is given in subsection 6-2-5. This experiment shows only marginal change if the latent dimension is changed. Note that the number of inputs does change the number of trainable parameters. Based on the results below, n_z is set equal to 100 following most literature.

n_z	Recall
10	0.87
20	0.91
50	0.91
100	0.91
200	0.92

Table 4-3: Changing the size of the latent vector n_z

4-5-6 Training on majority and minority classes

The aim of the research is to train a GAN on all classes and then oversample the minority classes. It can be assumed that GANs have more difficulty learning the underrepresented classes as these classes are shown less often during the training iterations, this is also shown in Figure 4-4 where a slower Fréchet distance decay is shown if classes with a lower number of samples. In Figure 4-5 the Fréchet distance decay is shown. All classes are represented with 5000 samples in the training dataset except for the class labeled 1. After 200 epochs the majority classes no longer improve significantly however the quality of the minority class is still increasing.

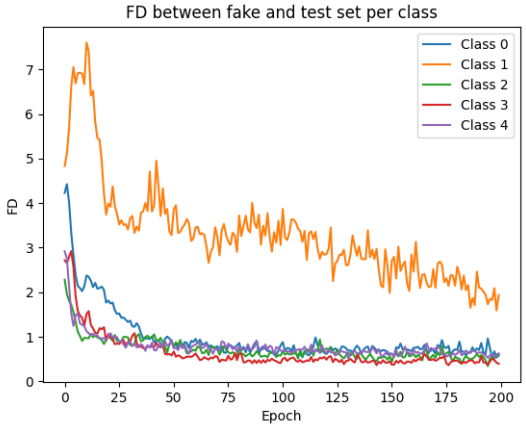


Figure 4-5: Experiment with single minority class

	Baseline		SMOTE		Simple		cWGAN-gp		Number of samples	
Class	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Train	Test
1	0.91	0.99	0.84	0.99	0.85	0.99	0.86	0.99	5000	9056
4	0.48	0.34	0.96	0.08	0.57	0.15	0.74	0.09	543	136
7	0.88	0.86	0.86	0.87	0.85	0.81	0.86	0.86	5000	11147
8	0.93	0.84	0.91	0.85	0.92	0.81	0.91	0.85	5000	6969
9	0.97	0.99	0.96	0.99	0.95	0.99	0.97	0.99	5000	13443
Average (m)	0.83	0.81	0.90	0.76	0.83	0.75	0.87	0.76		
Average (w)	0.92	0.93	0.90	0.93	0.89	0.91	0.90	0.93		

Table 4-4: Oversampling results on part of the dataset.

Table 4-4 shows the resulting classifier if only a single class is oversampled.

Training the GANs on only underrepresented classes and adding generated data to these underrepresented classes does not improve the overall classifier results as the generated data can be similar to the majority classes, which is not seen during GAN training. The summary of the results of this approach for the full dataset is given in Figure 4-6. The macro average recall score on the full dataset is 56% which is significantly worse than the baseline and all other results presented in Table 4-5.

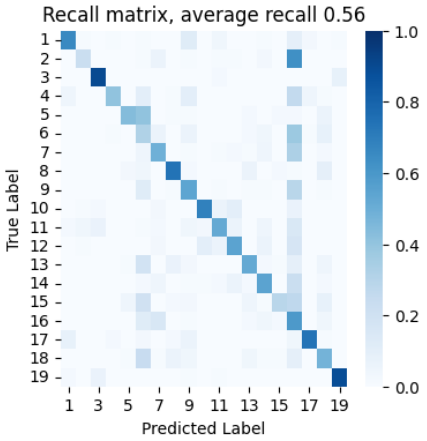


Figure 4-6: Recall matrix resulting from GANs oversampling where GANs are trained on only minority classes.

4-5-7 Full dataset

Considering the results from all smaller tests, the experiment on 'all' classes is conducted using 50 features. The results of this experiment are given in Table 4-5. In Table 4-5 multiple interesting minority classes are highlighted. These highlighted classes are better classified with the use of oversampling. Both SMOTE and GANs oversampling improve the recall score for these classes. However, by observing the precision scores and the average scores it can be deduced that overall performance does not increase, rather the classifier starts to become biased towards the minority classes.

While GANs-based oversampling increases the recall score on minority classes, the loss in recall of the majority classes results in no total improvement. SMOTE does increase the macro average score of the classifier and can therefore be considered the better oversampling option for this dataset. Note that even though the macro average recall increases slightly, the weighted average recall does decrease which means that the classifier becomes biased towards the minority classes.

4-6 Conclusion

From the conducted experiments, two main conclusions can be made:

Firstly, GANs can be used to generate artificial data that resembles the training data sufficiently well to serve as a replacement. We show that the performance of GANs decreases

	Baseline		SMOTE		Simple		cWGAN-gp		Number of samples	
Class	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Train	Test
1	.71 ± .03	.95 ± .00	.66 ±.01	.95±.01	.69±.01	.84±.00	.68 ± .02	.94± .00	5000	9056
2	.79 ± .06	.14 ± .03	.92 ±.01	.08±.01	.90±.02	.07±.00	.87 ± .02	.08 ±.01	1874	469
3	.90 ± .01	.83 ± .01	.90 ±.01	.81±.01	.89±.01	.88±.01	.91 ± .01	.81 ± .01	5000	4485
4	.38 ± .06	.12 ± .02	.73 ±.04	.09±.01	.49±.02	.09±.01	.39 ± .12	.12± .02	543	136
5	-	-	-	-	-	-	-	-	0	0
6	.58 ± .09	.06 ± .02	.82 ±.02	.03±.00	.79±.05	.02±.00	.65 ± .07	.04 ± .01	359	90
7	.53 ± .01	.75 ± .01	.52 ±.01	.75±.01	.45±.00	.68±.01	.52 ± .02	.74 ± .01	5000	11147
8	.65 ± .04	.59 ± .01	.60 ±.00	.61±.01	.53±.02	.56±.01	.59 ± .01	.60 ± .01	5000	6969
9	.74 ± .02	.95 ± .00	.69 ±.01	.95±.00	.63±.02	.94±.00	.71 ± .01	.95 ± .00	5000	13443
10	.89 ± .02	.16 ± .00	.84 ±.00	.18±.00	.83±.02	.13±.00	.85 ± .02	.17 ± .00	4764	1191
11	.72 ± .01	.91 ± .01	.70 ±.01	.92±.01	.71±.01	.83±.01	.70 ± .00	.92 ± .01	5000	7937
12	.55 ± .04	.82 ± .01	.49 ±.01	.83±.01	.46±.01	.82±.01	.49 ± .01	.82 ± .01	5000	9463
13	.57 ± .02	.24 ± .01	.58±.02	.22±.01	.42±.02	.24±.01	.57 ± .03	.22 ± .01	3618	905
14	.65 ± .02	.20 ± .03	.70 ±.02	.16±.01	.57±.03	.14±.01	.73 ± .03	.15 ± .01	2605	652
15	.66 ± .03	.31 ± .01	.61 ±.01	.33±.01	.45±.01	.27±.01	.60 ± .01	.32 ± .01	5000	1416
16	.60 ± .14	.09 ± .02	.80 ±.04	.05±.00	.62±.03	.04±.00	.73 ± .07	.05 ± .00	577	145
17	.61 ± .02	.73 ± .00	.58 ±.01	.74±.01	.51±.01	.67±.01	.59 ± .01	.73 ± .01	5000	10149
18	.76 ± .02	.60 ± .04	.73 ±.01	.60±.02	.71±.01	.66±.03	.74 ± .03	.58 ± .05	5000	1118
19	.60 ± .04	.69 ± .01	.50 ±.01	.72±.01	.34±0.01	.65±.01	.53 ± .02	.71 ± .01	5000	9591
20	.85 ± .02	.62 ± .02	.87 ±.01	.59±.01	0.87±.01	.59±.01	.84 ± .01	.63 ± .02	3061	766
Average (m)	.67 ± .01	.51±.01	.70±.01	.50±.01	.63±.01	.48±.00	.67± .01	.50±.01		
Average (w)	.66 ± .02	.77±.00	.62±.00	.78±.00	.57±.00	.74±.00	.63± .01	.77±.00		

Table 4-5: Results on full dataset except class 4. Average score and standard deviation over 5 runs. The average (m) is the macro average and the average (w) is the weighted average

with a larger number of features, calling for classification as well as GANs-based generation on a dimension-reduced approximation of the data. Increasing the number of trainable parameters does not show a clear increase in performance, leading to the hypothesis that not the network architecture of the neural networks but rather the adversarial framework is the limiting factor.

Secondly, on this specific dataset, oversampling only increases the classification of minority classes slightly, with the downside of decreasing precision and overfitting to minority classes. SMOTE increases performance a little more than GANs do, leading to the conclusion that GANs should not be used for oversampling on this dataset. SMOTE takes seconds to generate new samples, while GANs take hours to train (depending on the number of training samples and hardware used).

4-7 Further research

From this research can be concluded that GANs are not useful for oversampling in this IMS dataset. The results are validated with a second dataset (see Appendix A). Still, applying the same methods to a different dataset, for example, a binary IMS dataset with only 2 classes, should be considered in the future.

As shown by the experiments on majority classes only, an increasing number of classes is not a hindering factor for GANs performance. Applying GANs for oversampling in a multiclass setting is not yet a common occurrence in literature. Based on this research, GANs do have potential for multiclass oversampling tasks.

To further increase performance, other neural network architectures can be applied such as Deep Convolutional networks[23, 94] and residual networks[95]. A novel approach in GANs

design is the use of auto machine learning to optimize the neural network by changing the architecture as shown by alphaGAN[91] and autoGAN[96].

Instead of changes to the neural nets, the adversarial learning framework can also be further improved for example by using AC-GANs instead of the WGANs studied in this work.

Besides generating data, GANs can be used for many different applications. In IMS, anomaly detection by using the discriminator or even by using the specialized architectures such as AnoGAN[39] and GANomaly[40] might be useful to find the spatial location of spectra that do not fit in the general distribution of the data. Also, GANs that can be used for classification or clustering directly, such as ACGAN[65] and BIGAN[30].

As briefly pointed out in section 4-2, GANs can be used to generate spatial representations of a single m/z-bin instead of spectra. To do this, GANs capable of handling high resolutions should be considered such as StyleGANs[24].

Chapter 5

Design

This chapter explains the steps taken in the design of the implemented GANs. The design is an important step but it should be emphasized that during experiments changes were made to the initial design. For example, after the first experiments using the original GANs the change was made to the Wasserstein GANs, this change is motivated in subsection 6-2-1.

To ensure that the generator and discriminator both can learn their respective tasks with the IMS data some tests were conducted to choose a valid function design. These tests are given in section 5-2 and section 5-3. The components to build the generator and discriminator functions are introduced in section 5-1.

5-1 Network architectures

The successful training of GANs falls and stands with the quality of the parametric function. No literature has been found that uses functions different from neural networks. In this work, neural networks are used as functions as well. The design of these neural networks is done by testing a few different options based on designs seen in the literature.

To motivate the design choices, subsection 5-1-1 first introduces some basic concepts of the different parts that make up a neural network. The main concepts are based on literature[97].

5-1-1 Building blocks

Neural networks are intricate, usually nonlinear, parametric functions. At the basis of these functions are blocks of simpler functions, that can be combined in various ways to change the model complexity and behavior.

Below, a short introduction is given to the building blocks used in the experiment conducted for this work. Throughout the examples and equations, a vector $\mathbf{x}$ is considered the input to the neural network and a vector $\mathbf{y}$ is considered the output. The trainable parameters are denoted as θ, note that the trainable parameters are left out of the equations to increase

readability. A neural network consists of multiple layers, where every layer is a combination of inputs followed by an activation function. E.g. a neural network consisting of three layers can be written in the following way (5-1):

$$\mathbf{y} = f(x) = f_1(f_2(f_3(x))) \tag{5-1}$$

Here, output $\mathbf{y}$ is described by a function $f(x)$ and $f(x)$ consists of 3 functions (layers) in series. The first layer, the input layer, takes the sample $\mathbf{x}$. The final layer, the output layer outputs the vector y all intermediate layers are commonly called hidden layers, these layers combine the outputs from the previous layer. Every layer contains a method of combining inputs (e.g. linear) with an activation function. Additional regularization can be applied to the layers to make training the neural network more effective[98].

Linear layer

A linear layer, also known as a fully connected[23] or dense layer[99] applies a linear transformation to its input. A linear layer consists of a weight matrix W and a bias vector b. The output of the layer is the sum of the weight matrix multiplied by the input vector and the bias as given in (5-2)

$$y = Wx + b \tag{5-2}$$

A visual interpretation of a linear layer is given in Figure 5-1. This figure shows that all output features from every layer connect to all inputs of the next layer i.e. all layers are fully connected.

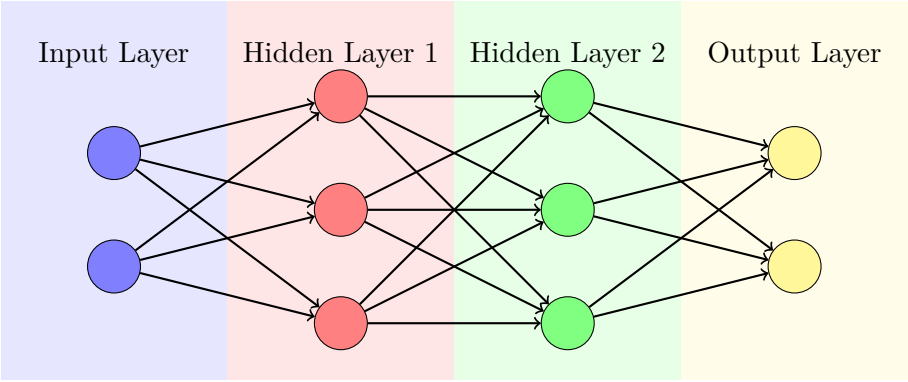


Figure 5-1: Fully connected network with 2 hidden layers.

A major downside of fully connected layers is that the number of parameters (weights) grows rapidly as the number of features increases. If the input to the neural network has for example a million features (e.g. image of 1024 by 1024 pixels) and the input layer has only 1000 outputs the weight matrix holds already a billion weights. Even larger inputs or a higher number of outputs makes updating the weights too time-consuming.

As shown in Figure 5-1, the number of nodes of one layer does not have to match the number of nodes of the subsequent layer. The number of nodes can grow or shrink i.e. linear layers can be used for both encoding and decoding data.

Convolutional layer

A convolutional layer extends the concept of a linear layer by focusing on local patterns. Instead of fully connecting all input and output nodes, it uses small learnable filters (kernels) that slide over the input sequence, performing element-wise multiplication and summation. In (5-3) the convolutional layer is given. Here $y(i)$ is the i^{th} output feature determined by the sum over m features of the input data x multiplied by the m^{th} kernel (K) element. m Is the number of features in the kernel. Note that here a 1D kernel is used resulting in a vector of outputs. This kernel can be extended into a 2D kernel giving rise to multiple output channels. Please note that this is different from a 2D-convolution operation; in 2D-convolution the kernel moves in two directions.

$$y(i) = \sum_{m} x(i-m)K(m) \tag{5-3}$$

Figure 5-2 gives an overview of the convolution operation. In this case, 3 input features are combined through a filter to create a single output feature. Note that the number of output features is lower than the input features. To compensate for this, additional features (usually zeros) can be added to both ends of the input (so-called padding).

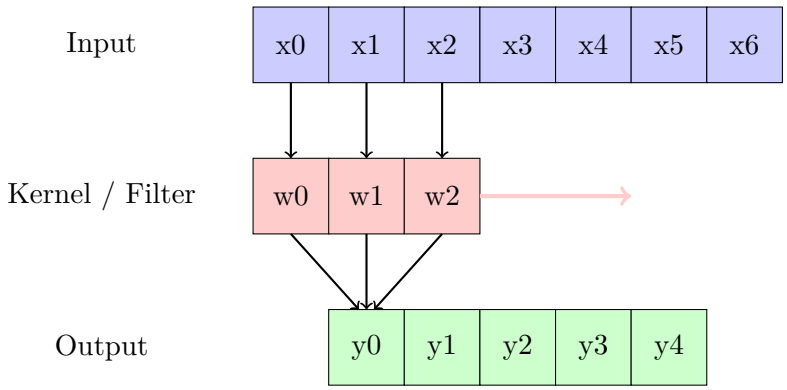


Figure 5-2: In purple; the input layer, In red the kernel; In green the output layer. The red arrow indicates the movement of the kernel.

Convolutional layers are computationally efficient, as the number of weights depends on the size of the kernel and not on the number of inputs. This means that no matter the size of the input the number of trainable parameters (number of weights in the kernel) stays the same.

The opposite of the convolution operation is the inverse convolution or deconvolution. Convolution lowers the dimension and is usually used for encoding data to a lower space such as in the discriminator. Deconvolution is used for decoding data from a lower to a higher dimension and can be used for the generator.

Activation functions

Activation functions introduce nonlinearities into the network enabling it to model more intricate data relationships. An activation function takes the output of a node as an argument.

Some of the most common functions are given in Appendix C. The derivates are included to show the effect of the layers on the backpropagation step.

Batch normalization

Batch normalization is a technique used in neural networks to improve training stability and performance. It normalizes the inputs of each layer, ensuring that they have a consistent mean and variance across mini-batches. By mitigating issues like internal covariate shift, batch normalization accelerates convergence and allows for higher learning rates[100].

Dropout

Dropout is a regularization technique used in neural networks to prevent overfitting. During training, dropout randomly deactivates some neurons in the network at each iteration, forcing the network to learn redundant representations of features. This approach helps the model generalize better by reducing reliance on specific neurons and mitigating the risk of overfitting to the training data. By improving generalization, dropout enhances the model's performance on unseen data[101].

Label embedding

To include labels in the input of the neural network an embedding layer is used. An embedding layer maps a discrete input (class label) to a continuous vector. This mapping can be seen as a linear layer where the number of inputs is the number of classes. The output, which is concatenated with the noise vector, can be of any chosen length. In this work, the output length of the embedding layer is chosen equal to the number of classes.

5-1-2 Optimizer

In the original GANs[15] Stochastic Gradient Descent (SGD) is used to optimize both neural networks based on the chosen optimization function. However, later works[23, 20] use the Adam optimizer. In short, Adam computes *"individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients"*[74]. By using the moments of the gradients, the algorithm converges faster as SGD.

The use of momentum is discouraged in some literature on GANs[16] as it would lead to instability and GANs not converging. This is however disputed in other literature that does choose Adam over SGD[20]. In this work, Adam is used for optimization.

5-2 Discriminator Design

The successful training of GANs is largely based on the effectiveness of the discriminator. If the discriminator fails to distinguish real from fake data, the updates to the generator become meaningless.

The input of the discriminator is a spectrum (vector) with the corresponding class label (integer). The output of the discriminator is a prediction of the input either being a real or generated sample. In the original adversarial loss function given in (2-3) the prediction should be in the range of 0 to 1. The Wasserstein loss (2-7) expects a prediction in $\mathbb{R}$.

Design considerations

Based on the original GANs[15]and conditional GANs[19]the first discriminator tested is a neural network with fully connected layers. Based on suggestions found in literature[23]Leaky ReLU is used on all layers except the output layer which uses a sigmoid activation function. To accommodate the condition on the input an embedding layer is used. During experimentation, the choice was made to incorporate dropout in all hidden layers as suggested in literature[23].

The discriminator takes a spectrum $\tilde{x}$ (either real or generated) with the corresponding label y. The output p of the generator is between 0 and 1 corresponding to predicting 'fake' and 'generated' respectively.

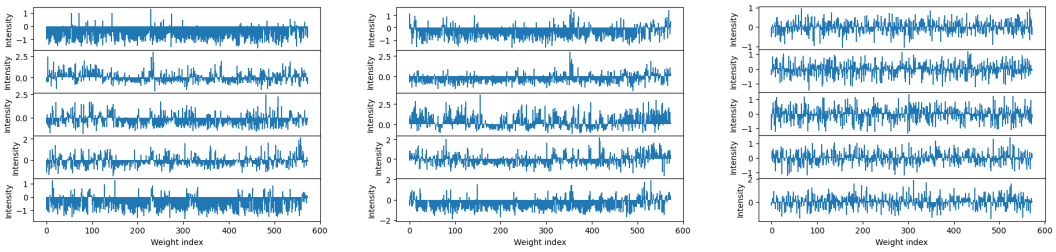
$$p = D(\tilde{x}, y) \in (0, 1) \tag{5-4}$$

When using the Wasserstein GANs the output activation function is removed leaving a fully connected linear output layer. This gives the discriminator a range of $(-\infty, \infty)$ The number of connections in each layer and the number of layers is determined by the experiments below.

5-2-1 Discriminator verification

To test if the discriminator architecture is sufficient to learn to distinguish between original and (imperfect) fake samples, the architecture is tested as a binary classifier. The intuition is that, if the discriminator can distinguish different classes of the original data, it can distinguish between real and imperfect generated data. Additionally, a test is conducted to see if the discriminator can distinguish between real data and pure noise (from an untrained generator). A few example spectra used for these tests are given in Figure 5-3.

Note that, even though the discriminator is designed to incorporate labels in its decision-making, during this test all labels are set to a static value. These tests are conducted on



(a) **(b)** **(c)**

Figure 5-3: The discriminator is used as a regression model classifier to distinguish between class 1 and class 2 and between class 1 and noise. (a) Example from class 1, (b) Examples from class 2, (c) Examples of noise samples

a few different architectures. In the tables (Table 5-1 and Table 5-2) fc(n) denotes a fully connected network with n linear connection between layers. A few tests were conducted using deep convolution networks (DC(n)), here n is a factor in the number of channels in the convolutional layers. Full descriptions of these architectures can be found in section D-1 and subsection 5-2-2.

The scores of the different fully connected options indicate that 3 or 4 linear connections give the best results. Based on these results, the 3-connection fully connected network is used in

	Accuracy at iteration			Final Average	
Model	5	10	50	Recall	Precision
DC (8)	0.50	0.60	1.00	0.95	0.96
DC (32)	0.65	0.70	1.00	1.00	1.00
fc (1)	0.99	1.00	1.00	0.98	0.99
fc (3)	1.00	1.00	1.00	0.99	0.99
fc (4)	1.00	1.00	1.00	1.00	1.00

Table 5-1: Noise test; all scores on test set. DC: deep convolutional network, fc: fully connected network.

	Accuracy at iteration			Final Average	
Model	5	10	50	Recall	Precision
DC (8)	0.50	0.60	1.00	0.95	0.96
DC (32)	0.39	0.55	0.61	0.87	0.90
fc (1)	0.63	0.78	0.78	0.87	0.82
fc (3)	0.99	1.00	1.00	0.94	0.93
fc (4)	1.00	1.00	1.00	0.92	0.93

Table 5-2: Binary test; all scores on test set. DC: deep convolutional network, fc: fully connected network.

further experimentation. In the next section, the implemented network is given. The other networks used in this test are given in section D-1.

5-2-2 Final discriminator design

Based on the test results given in the previous section, the discriminator architecture given in Table 5-5 is used for experiments. Note that the number of classes (19) as well as the length of the input vector (spectrum size of 573) changes in some experiments. The input sizes and therefore the number of trainable parameters are changed accordingly. The number of parameters of a linear layer is the number of input features multiplied by the number of output features (the W matrix) plus the number of output features (the bias b).

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	592 (573+19)	512	303616 (592x512+512)
LeakyReLU	512	512	- -
Linear	512	512	262656 (512x512+512)
Dropout (0.4)			- -
LeakyReLU	512	512	- -
Linear	512	1	513 (512x1+1)
Sigmoid			- -
	Trainable parameters: 567146		

Table 5-3: Discriminator design with 3 linear connections. The sigmoid output on the final layer is omitted in the Wasserstein GANs

To further clarify this design, we relate it to the graph given in Figure 5-4. In this graph, the gray circle indicates the embedded labels and the purple input circles indicate the signal (spectrum). Next, the two hidden layers have an equal number of neurons, from the second hidden layer all output features are combined into a single output layer.

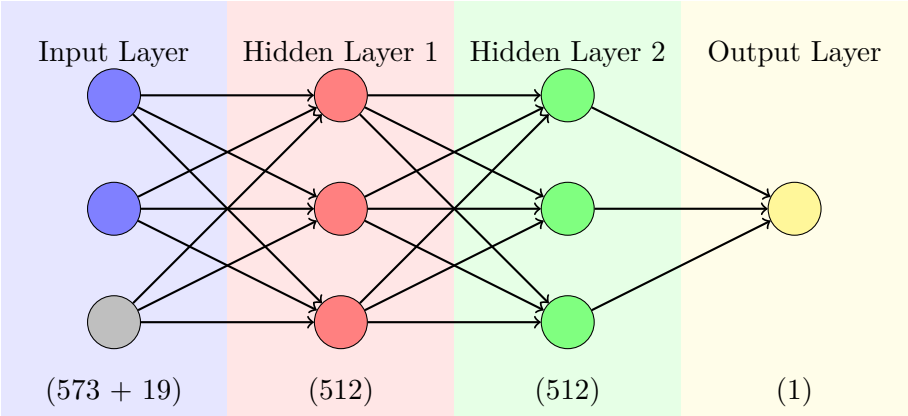


Figure 5-4: Fully connected network with 2 hidden layers.

5-3 Generator Design

For the design of the generator, the design mustn't limit the mapping from noise to a generated sample. The output range of the generator has to be considered in relation to the scaling.

5-3-1 Design considerations

The generator eventually has to be able to map a noise input to an output indistinguishable from real data. In literature, the generator usually uses a tangent hyperbolic output which puts the range of all features in the interval of $(-1, 1)$. As discussed in subsection 3-1-3, depending on the scaling of the data this range has to be changed. In this work, the choice is made to use standard scaling for the majority of experiments. To accommodate the range the final layer of the generator uses no activation, i.e. a fully connected linear output mapping is used.

5-3-2 Generator verification

To test the generator capacity in a similar way to the discriminator the neural network is used as a decoder network that maps the noise vector z to an output signal. The reference output signal is generated by a function $f(\cdot)$ that takes the same input z as the decoder. The goal of this experiment is to minimize the difference between $G(z)$ and $f(z)$ as shown in (5-5). The function $f(z)$ can be any function, an experiment with a linear function is included in the appendix. Assuming that a linear mapping of noise can not model the data, a second-order function is used for the analysis below.

$$\min_{G} ||f(z) - G(z)||_2^2 \tag{5-5}$$

The choice for $f(z) = Az^2 + Az + b$ comes from the assumption that if the data is from a multivariate normal distribution $\mathcal{N}(\mu, \sigma\sigma^\top)$ it can be described by a linear function $f(z) = \sigma z + \mu$ with $z \sim \mathcal{N}(0, I)$. Where $AA^\top$ is a lower rank approximation of the covariance matrix, calculated by the truncated singular value decomposition, of the data used in the experiments below. b Is the mean of the data.

The function f(z) is approximated by different candidate generator models $G(z)$ and for comparison linear regression and linear least squares are used as baseline. To compare the different models both the mean squared error and Fréchet distance (see section 2-2-3) are used. The tests were executed multiple times to verify that the training stability did not cause a large variance in the results. The results are given in Table 5-4

G(z)	MSE	FD
LLS	2.19 ±0.02	27.25 ±0.17
LR	1.18 ±0.02	13.37±0.26
fc (1)	1.57 ±0.27	29.50 ±0.74
fc (3)	0.36 ±0.00	10.56 ±0.05
fc (4)	0.25 ±0.01	8.78 ±0.09
fc (5)	0.67 ±0.01	17.89 ±0.08
DC (8)	1.72 ±0.08	29.86 ±1.46

Table 5-4: Results of quadratic function (Average scores ± standard deviation over 3 runs). LLS: Linear least squares, LR: Linear regression

5-3-3 Final generator design.

Based on the test results given in the previous section, the following generator architecture is used for experimentation:

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	119 (100+19)	128	15360 (119x128+128)
LeakyReLU	128	128	- -
Linear	128	256	33024 (128x256+256)
BatchNorm1d	256	256	- -
LeakyReLU	256	256	- -
Linear	256	512	131584 (256x512+512)
BatchNorm1d	512	512	- -
LeakyReLU	512	512	- -
Linear	512	573	293949 (512x573+573)
BatchNorm1d	573	573	- -
	Trainable parameters:		474278

Table 5-5: Generator design with 4 linear connections.

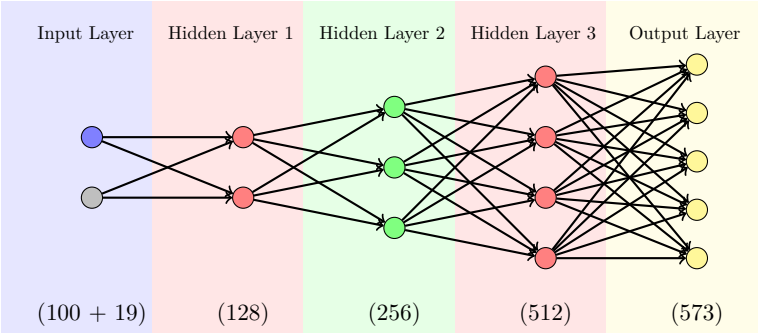


Figure 5-5: Generator design overview. The purple circle represents the input noise vector and the gray circle represents the embedded label. The number of circles represents the number of neurons in that layer.

5-4 Training and hyperparameters

During experimentation, the choice was made to switch from the original GANs to the more stable Wasserstein-based GANs. This choice is explained in subsection 6-2-1. For optimizing the discriminator and generator functions, the Adam[74] optimizer is used with a learning rate of 0.0001 for the discriminator. The learning rate for the generator is scaled with the number of discriminator steps for every generator step (where a step is updating over a single minibatch). The stopping criteria for optimization are a total number of iterations of 1000, which proved enough to observe convergence in the different quality measures. Additionally, training is stopped if the discriminator or generator loss explodes (would go over a threshold for multiple update steps).

The data is split into minibatches with a batch size of 64. The last batch is dropped in every iteration to ensure that no minibatch with a single sample exists, the batch normalization in the neural networks relies on having multiple samples in each update step.

5-5 Benchmark

To test if the GANs are correctly implemented, as a first step one of the benchmarks used in literature[1] is tested with the design described in previous sections. As noted in subsection 3-1-3 standard scaling was used for the IMS-data, however, to compare this benchmark min-max is used in accordance with the cited paper. In table Table 5-6 the results are given. In this table, the neural network architecture of both the discriminator and generator is based on the paper to see if the results are equal. The scores of the baseline (W/O), SMOTE, and conditional GANs based on logistic regression are very similar to the scores reported.

As noted in section 3-3 LR can not be used for multiclass classification. Therefore the results are also given for the linear discriminant analysis. What springs out immediately is that the baseline LDA already performs better than the LR classifier. Also, only smote can increase performance slightly, while both cGANs and cWGAN-GP both decrease performance.

If, instead of the very small neural networks (single layer) used in the cited literature, our own design is applied in the same experiment. The GANs in fact do increase the performance of

Method	Metric	W/O	SMOTE	cGAN	cWGAN-gp
LR	Precision	0.90	**0.91**	**0.91**	**0.91**
	Recall	0.83	0.87	**0.89**	0.85
	F1-score	0.85	0.88	**0.90**	0.87
LDA	Precision	**0.97**	**0.97**	0.94	0.93
	Recall	0.94	**0.95**	0.91	0.92
	F1-score	0.95	**0.96**	0.92	0.92

Table 5-6: macro average scores, data scaled using minmax, model architectures based on paper

both the LR and LDA classifier. The results are given in Table 5-7. From these scores, we see a clear improvement of the GANs scores leading to the conclusion that a more complicated model can model the data with more detail.

Table 5-8 shows that if standardization is used instead of minmax scaling, the baseline score of the LR classifier improves to better scores than previously seen by any method. The designed GANs can not further increase performance for the LR classifier, however, the LDA classification is improved by the cWGAN-gp.

Method	Metric	W/O	SMOTE	cGAN	cWGAN-gp
LR	Precision	0.90	**0.91**	0.90	0.88
	Recall	0.83	0.87	**0.90**	0.87
	F1-score	0.85	0.88	**0.90**	0.88
LDA	Precision	**0.97**	**0.97**	**0.97**	**0.97**
	Recall	0.94	**0.95**	0.94	**0.95**
	F1-score	0.95	**0.96**	0.95	**0.96**

Table 5-7: macro average scores, data scaled using minmax, model architectures based on ours (fc(3))

Method	Metric	W/O	SMOTE	cGAN	cWGAN-gp
LR	Precision	**0.97**	**0.97**	0.91	0.91
	Recall	**0.97**	**0.97**	0.93	0.93
	F1-score	**0.97**	**0.97**	0.92	0.92
LDA	Precision	**0.95**	**0.95**	0.90	0.94
	Recall	0.91	0.92	0.92	**0.95**
	F1-score	0.92	0.93	0.91	**0.94**

Table 5-8: Macro average scores, standardized data, based on our architecture

Chapter 6

Experiments

This chapter shows some of the important experiments that lead to the conclusions given in this report. The first experiment used the full dataset and in this experiment, all minority classes were oversampled using GANs. This proved unsuccessful, however, why this did not work as intended was not clear from the results. Many different experiments are conducted to find if and how GANs can be successfully applied to IMS and specifically for oversampling minority IMS classes.

6-1 Initial experiment

After implementing the designed architectures as described in the previous chapter, the first experiment was on the full dataset, meaning on all classes and all features (full spectra). This experiment is characterized by the following settings:

- Using all 19[1] classes
- Using all features (mass bins)
- Using standard scaling
- Using conditional GANs (adversarial loss as given in (2-6))

6-1-1 Baseline results

The baseline scores of classification are visualized in the recall matrices given in section 6-1. Based on the average recall scores it can be concluded that the data augmentation baseline lowers the performance of classification. SMOTE, however, does increase the recall score.

From these visualizations, the effect of the class imbalance is not directly visible. In Table 6-1 the full results are given in combination with other resulting classifiers fit to oversampled data.

[1] Class labeled 5 is left out of experimentation due to the severe lack of examples

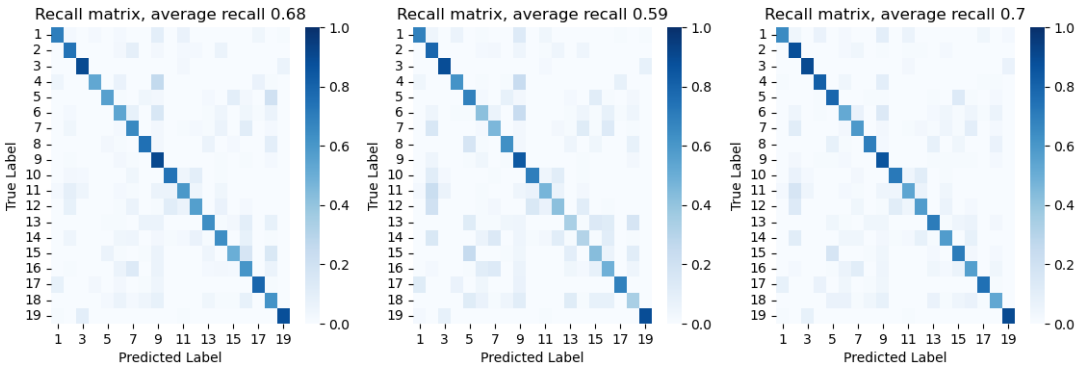


Figure 6-1: (a): the recall scores without oversampling, (b): the recall scores with augmented oversampling, (c): the recall scores with SMOTE oversampling

6-1-2 GAN quality

After running the experiment, the oversampling score of the GANs is significantly worse than the baseline scores. To investigate why the results are lacking the validation methods given in section 2-2-3 are used. First, the replacement test shows that the GANs did not learn to generate distinguishable classes as shown in Figure 6-2. In the recall figure, we expect a diagonal line high scores similar to Figure 6-1.

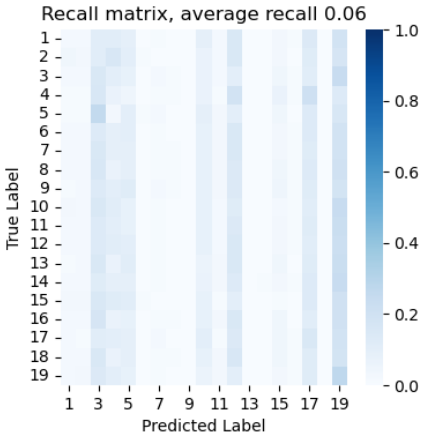


Figure 6-2: Recall scores on test set, LDA fit on data generated by GANs

A second observation, based on a quantitative method is the lack of variance of the generated samples. Figure 6-3 shows the mean and standard deviation of generated samples and original samples of a single class. The mean is not estimated correctly and the generated samples show no variance.

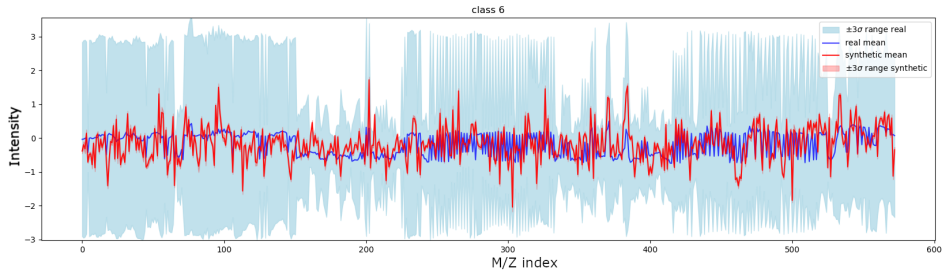


Figure 6-3: Visualization of the distribution of a single class, in blue mean and standard deviation of the original data and in red the mean and standard deviation of the generated data.

6-1-3 Comparison

In Table 6-1 the full oversampling results are given. The gray rows show the minority classes that score low in the baseline score. SMOTE gives a slight improvement in recall score. However, this improvement comes with the cost of reduced precision meaning that the added data leads to overfitting towards minority classes.

The implemented conditional GANs can not generate data that helps with the classification scores. It rather decreases performance, the sole reason the average scores are still relatively high is that the majority class data remains untouched. Figure 6-2 shows what happens if all training data is replaced: the average score drops to approximately 1/19 meaning that a classifier trained on the generated data randomly guesses every class.

	Baseline		SMOTE		Simple		cGAN		Number of samples	
Class	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Train	Test
1	0.70	0.96	0.65	0.96	0.68	0.95	0.73	0.95	5000	9056
2	0.74	0.15	0.88	0.09	0.97	0.06	0.00	0.00	1874	469
3	0.90	0.86	0.90	0.85	0.89	0.84	0.93	0.85	5000	4485
4	0.53	0.17	0.82	0.11	0.62	0.10	0.04	1.00	543	136
5	-	-	-	-	-	-	-	-	0	0
6	0.56	0.07	0.79	0.03	0.73	0.02	0.00	0.00	359	90
7	0.54	0.75	0.52	0.75	0.42	0.65	0.57	0.73	5000	11147
8	0.66	0.60	0.59	0.62	0.47	0.57	0.71	0.56	5000	6969
9	075.	0.95	0.70	0.96	0.63	0.94	0.81	0.94	5000	13443
10	0.91	0.16	0.86	0.17	0.84	0.13	0.88	0.18	4764	1191
11	0.74	0.89	0.72	0.90	0.70	0.87	0.78	0.89	5000	7937
12	0.60	0.80	0.54	0.80	0.47	0.81	0.69	0.78	5000	9463
13	0.57	0.25	0.58	0.22	0.42	0.17	0.30	0.36	3618	905
14	0.64	0.20	0.71	0.16	0.37	0.10	0.01	0.56	2605	652
15	0.64	0.31	0.58	0.33	0.32	0.17	0.73	0.28	5000	1416
16	0.50	0.09	0.71	0.05	0.42	0.02	0.00	0.00	577	145
17	0.61	0.73	0.57	0.74	0.48	0.64	0.65	0.71	5000	10149
18	0.79	0.58	0.75	0.58	0.69	0.74	0.81	0.57	5000	1118
19	0.62	0.68	0.53	0.69	0.34	0.59	0.70	0.62	5000	9591
20	0.88	0.63	0.90	0.61	0.89	0.60	0.83	0.72	3061	766
Average (m)	0.68	0.52	0.70	0.51	0.59	0.47	0.53	0.56		
Average (w)	0.67	0.77	0.63	0.77	0.55	0.73	0.70	0.76		

Table 6-1: Oversampling results, in gray the most minority classes that reduce classification performance

6-2 Progressive experimentation

From the initial experiment, it is clear that oversampling using the implemented cGANs does not work. However, what the limiting factor is, is not clear from the results. Many experiments have been conducted to determine why results seen in literature can not be reproduced with this dataset. From these experiments, the most significant results are shown in the following sections.

6-2-1 wGAN-gp

As explained in section 2-2-1 the quality of the updates to both the discriminator and generator network is dependent on the quality of the discriminator. However, updating the discriminator for more steps as the generator is discouraged as this hyperparameter is difficult to tune as explained in section 2-2-1.

To circumvent the difficulties that might arise from fine-tuning the number of discriminator update steps the original GAN optimization is changed to the Wasserstein metric with gradient penalty as presented in (2-10). This alternative formulation allows for unlimited training of the discriminator. After some small experiments is concluded that 100 discriminator update steps for every generator step yields the desired effect.

With the higher number of discriminator update steps, it can be assumed that the quality of the update step of the generator is high. Therefore the learning rate for the generator is set higher than the learning rate of the discriminator.

The main difference this change to the Wasserstein GANs has brought is the increased variance in the generated data. Figure 6-4 shows how the mean of a single class is relatively well estimated and the standard deviation of the generated data overlaps with the standard deviation of the original data. This is a large step from the cGANs results seen previously.

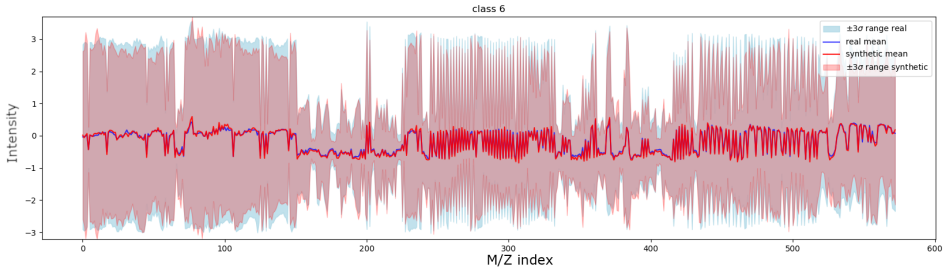


Figure 6-4: Visualization of the distribution of a single class, in blue mean and standard deviation of the original data and in red the mean and standard deviation of the generated data.

Even though the statistics of the generated spectra look very similar to the original data in the visualization. Using the training data replacement test shows that the data generated by the Wasserstein GANs is still useless. This result is given in Figure 6-5.

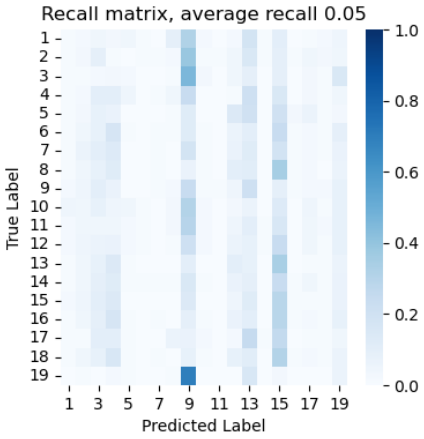


Figure 6-5: Recall scores on test set, LDA fit on data generated by wGAN-gp

6-2-2 Noise reduction

Implementing the conditional Wasserstein Generative Adversarial Nets with gradient panelty (cWGANs-gp) appears to improve the quality of samples, however, samples can not be used for fitting a classifier. To further investigate why this does not work the data is denoised by applying NMF to the data and scaling it back by multiplying the resulting W and H matrix. The results of this experiment are similar to the previous experiment using WGANs-gp on the full spectra. There is a slight improvement in the average recall score as shown in the recall matrix given in Figure 6-6. This minor improvement is not considered to be proof that noise is the source of the oversampling of the GANs not working.

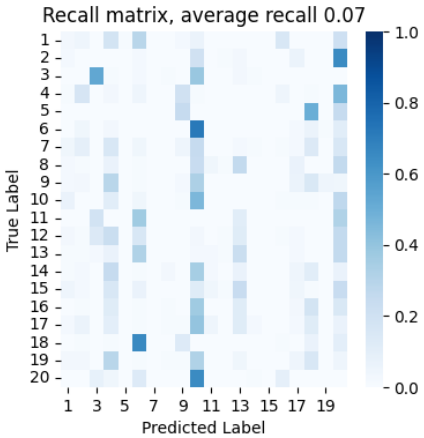


Figure 6-6: Recall scores on test set, LDA fit on data generated by wGANs-gp. The original data is first denoised by applying dimensionality reduction and increasing the dimension back

6-2-3 Dimensionality

The dimension of the signal might be a limiting factor in the performance. In the literature on GANs-based oversampling is stated that reducing the dimensions to oversampling on reduced dimensions gives better results and that increasing the number of dimensions does not increase the results[93]. To test this, NMF is applied and the GANs are trained on the weights (the W matrix). To expedite this experiment only five classes are used for this experiment. The effect of the number of classes is explored in subsection 6-2-6.

Figure 6-7 shows the macro average recall score of an LDA fitted on generated data. On the x-axis there is an increasing number of features used. SMOTE and the baseline don't differ in this test. The performance of simple data augmentation decays slightly by a higher number of features. cWGAN-gp performce quit well up to 100 features but starts to decay there. cGANs performs significantly worse than cWGANs-gp and decreases faster as well. Based on

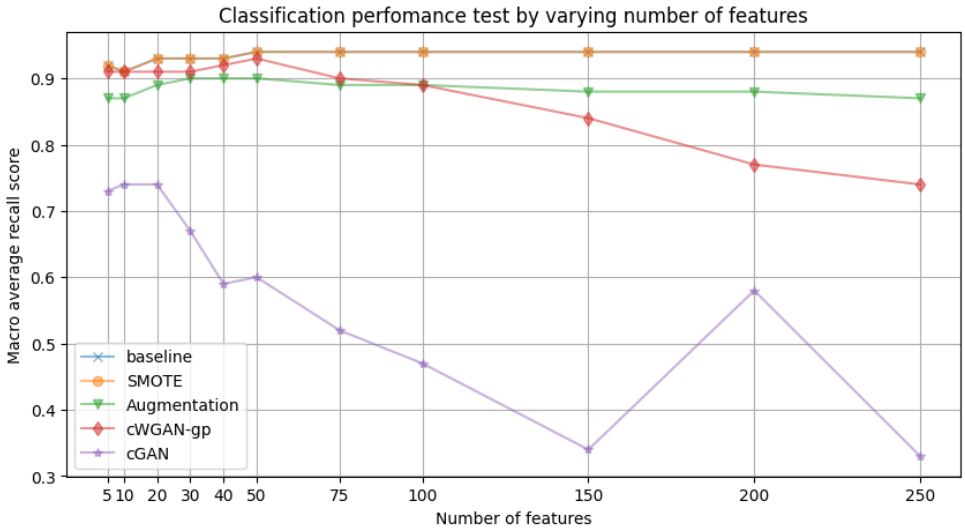


Figure 6-7: Recall score of replacement test, Discriminator uses 3 and Generator uses 4 linear connections.

this experiment it is concluded that working with a lower dimensional representation of the data works better with GANs. As the optimal recall score for the wGANs is reached with 50 features, this number is considered ideal for further experimentation.

6-2-4 Model complexity

The previous results show that the dimension of the signal is a problem for the GANs. From this can be assumed that the model is insufficiently complicated to model the data in higher dimensions. To test this hypothesis, the dimensionality experiment is repeated with both the generator and discriminator networks having one additional layer.

Figure 6-8 shows a similar trend to the trend seen in the previous section, however instead of the additional dimension helping the recall score, the score decay happens earlier and faster.

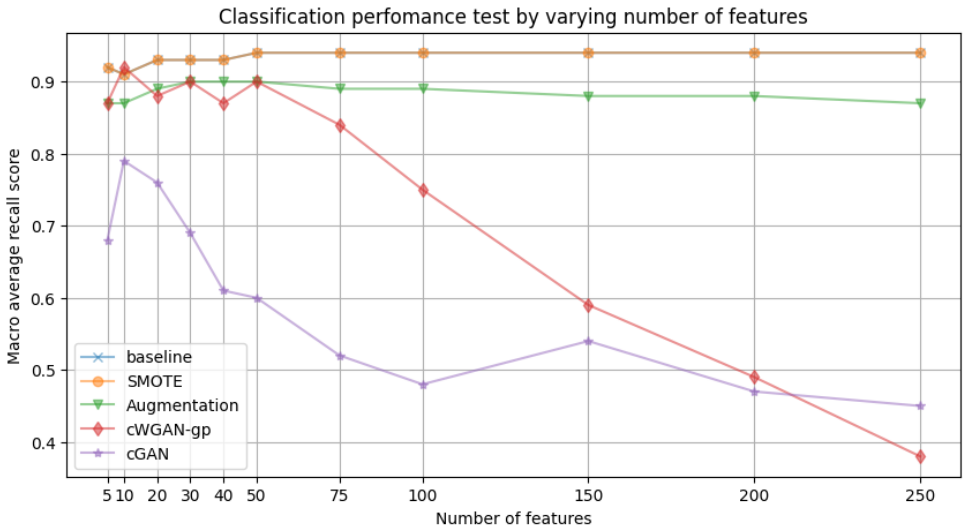


Figure 6-8: Recall score of replacement test, Discriminator uses 4 and Generator uses 5 linear connections.

6-2-5 Latent space dimension

A hyperparameter that seems te be closely related to the variation in the final dataset is the number of latent variables used to generate a sample. In literature, a latent vector of length 100 is most common[23, 15] but no studies have been found that explicitly study the effect of the size of this latent space.

A few different sizes are tested as noted in the Table 6-2. Only a subtle change is observed between the size of the latent variables. Therefore the conclusion is drawn that this vector length is not significant for overall performance. The size of 100 often-seen literature is used for further experimentation.

In Table 6-2 50 features are used in accordance with previous experiments. Again, 5 majority classes are used. The score in the table is the average recall score resulting from replacing the data.

n_z	Recall
10	0.87
20	0.91
50	0.91
100	0.91
200	0.92

Table 6-2: Changing the size of the latent vector n_z

6-2-6 Number of classes

The number of features has a big influence on the performance of the GANs as shown in subsection 6-2-3. A second variable that introduces more complexity to the data is the

number of classes. Some of the previous experiments have been on only 5 majority classes to show study the capabilities of GANs without the difference in the number of samples in mind. To study the effect of the number of samples the number of classes is increased as shown in Figure 6-9. In this test, the number of features used is 50. This figure shows a decay in performance, however, the decay in the performance of the GANs follows the same trend as the baseline techniques.

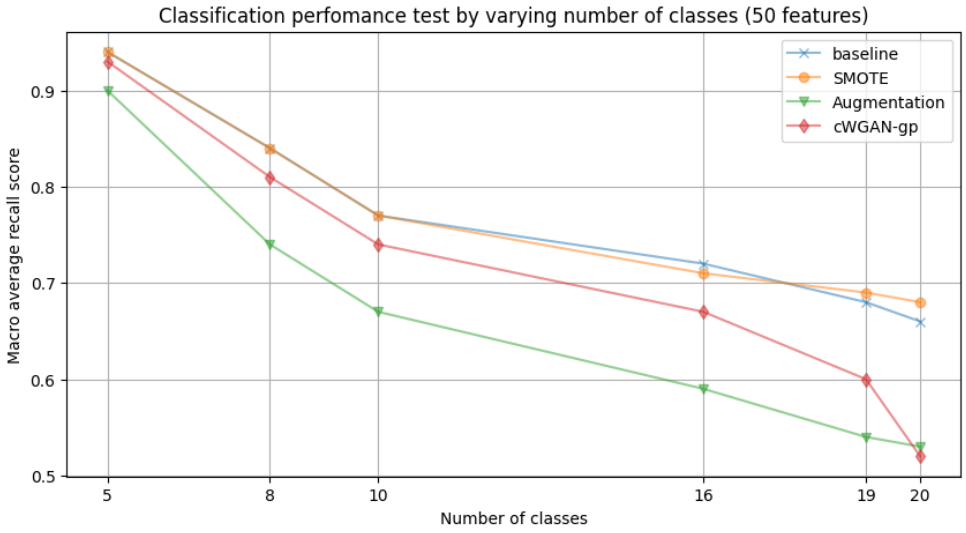


Figure 6-9: Varying the number of classes.

6-2-7 Minority only training

Instead of training on all classes, another approach would be to train the GANs only on the classes that need oversampling, i.e. train GANs only on minority classes. However, a downside of this method is that the GANs learn to generate data that is not necessarily distinguishable from majority classes but only from minority classes.

Applying this method using the Wasserstein GANs with the reduced dimensions as discussed previously does not result in better performance compared to the initial test conducted in section 6-1. The GANs are trained in class 2,4,6 and 16.

6-3 Final experiment

Based on the results, the initial experiment on the full dataset is repeated with the following settings:

- All 19[2] classes
- Using a reduced number of features (50 NMF-weights)
- Using standard scaling
- Using conditional WGANs with gradient penalty

[2]Class labeled 5 is left out of experimentation due to the severe lack of examples

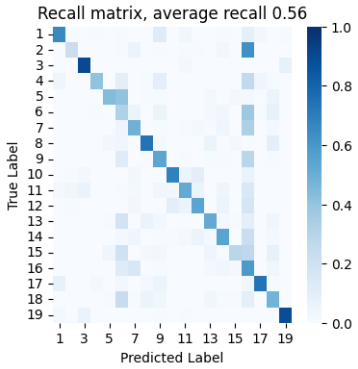


Figure 6-10: Recall matrix of GANs-based oversampling using 50 features. cWGANs-gp trained on only minority classes.

6-3-1 Baseline results

Similar to the initial test described in section 6-1 the baseline results are collected and visualized in recall matrices below. Note that the baseline scores based on the reduced dimensions slightly differ from the original baseline scores as presented in section 6-1.

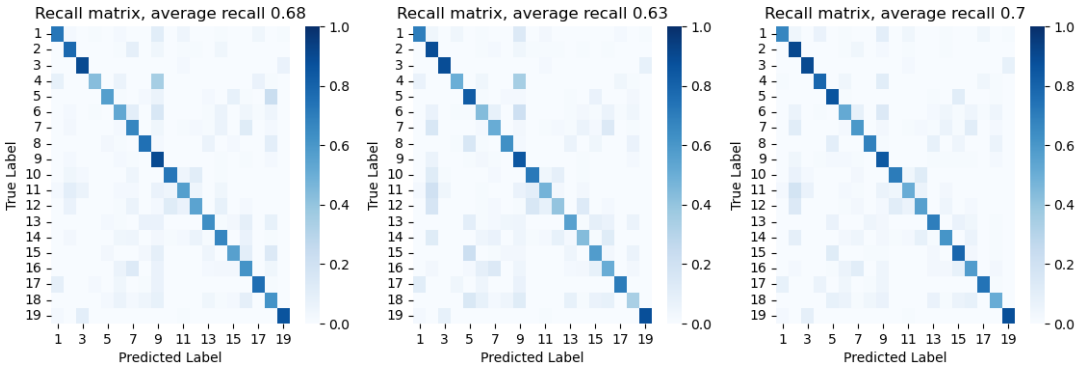


Figure 6-11: (a): the recall scores without oversampling, (b): the recall scores with augmented oversampling, (c): the recall scores with SMOTE oversampling

6-3-2 GAN quality

As a first indicator of the GANs performance the training data for classification is entirely replaced, as explained in section 2-2-3. The resulting recall scores are visualized in a recall matrix given in Figure 6-12. This figure shows that the 4th class is not correctly classified, however, all other classes show relatively good scores as indicated by the diagonal darker blue line. Comparing this to the initial experiment results given in Figure 6-2 a significant improvement is made using the insights gained in previous experiments.

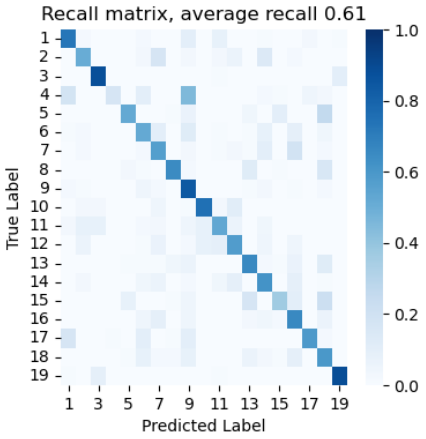


Figure 6-12: Recall matrix replacement test using data generated by wGAN-gp for training and original data for testing.

6-3-3 Comparison

The full results are given in Table 6-3. Comparing these results to the initial cGANs shows a significant performance increase for GANs-based oversampling. However, the increase in minority classes scores comes with a decrease in majority classes i.e. GANs-based oversampling does not result in an overall performance of classifier performance.

	Baseline		SMOTE		Simple		cWGAN-gp		Number of samples	
Class	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Train	Test
1	.71 ± .03	.95 ± .00	.66 ±.01	.95±.01	.69±.01	.84±.00	.68 ± .02	.94± .00	5000	9056
2	.79 ± .06	.14 ± .03	.92 ±.01	.08±.01	.90±.02	.07±.00	.87 ± .02	.08 ±.01	1874	469
3	.90 ± .01	.83 ± .01	.90 ±.01	.81±.01	.89±.01	.88±.01	.91 ± .01	.81 ± .01	5000	4485
4	.38 ± .06	.12 ± .02	.73 ±.04	.09±.01	.49±.02	.09±.01	.39 ± .12	.12± .02	543	136
5	-	-	-	-	-	-	-	-	0	0
6	.58 ± .09	.06 ± .02	.82 ±.02	.03±.00	.79±.05	.02±.00	.65 ± .07	.04 ± .01	359	90
7	.53 ± .01	.75 ± .01	.52 ±.01	.75±.01	.45±.00	.68±.01	.52 ± .02	.74 ± .01	5000	11147
8	.65 ± .04	.59 ± .01	.60 ±.00	.61±.01	.53±.02	.56±.01	.59 ± .01	.60 ± .01	5000	6969
9	.74 ± .02	.95 ± .00	.69 ±.01	.95±.00	.63±.02	.94±.00	.71 ± .01	.95 ± .00	5000	13443
10	.89 ± .02	.16 ± .00	.84 ±.00	.18±.00	.83±.02	.13±.00	.85 ± .02	.17 ± .00	4764	1191
11	.72 ± .01	.91 ± .01	.70 ±.01	.92±.01	.71±.01	.83±.01	.70 ± .00	.92 ± .01	5000	7937
12	.55 ± .04	.82 ± .01	.49 ±.01	.83±.01	.46±.01	.82±.01	.49 ± .01	.82 ± .01	5000	9463
13	.57 ± .02	.24 ± .01	.58±.02	.22±.01	.42±.02	.24±.01	.57 ± .03	.22 ± .01	3618	905
14	.65 ± .02	.20 ± .03	.70 ±.02	.16±.01	.57±.03	.14±.01	.73 ± .03	.15 ± .01	2605	652
15	.66 ± .03	.31 ± .01	.61 ±.01	.33±.01	.45±.01	.27±.01	.60 ± .01	.32 ± .01	5000	1416
16	.60 ± .14	.09 ± .02	.80 ±.04	.05±.00	.62±.03	.04±.00	.73 ± .07	.05 ± .00	577	145
17	.61 ± .02	.73 ± .00	.58 ±.01	.74±.01	.51±.01	.67±.01	.59 ± .01	.73 ± .01	5000	10149
18	.76 ± .02	.60 ± .04	.73 ±.01	.60±.02	.71±.01	.66±.03	.74 ± .03	.58 ± .05	5000	1118
19	.60 ± .04	.69 ± .01	.50 ±.01	.72±.01	.34±0.01	.65±.01	.53 ± .02	.71 ± .01	5000	9591
20	.85 ± .02	.62 ± .02	.87 ±.01	.59±.01	0.87±.01	.59±.01	.84 ± .01	.63 ± .02	3061	766
Average (m)	.67 ± .01	.51±.01	.70±.01	.50±.01	.63±.01	.48±.00	.67± .01	.50±.01		
Average (w)	.66 ± .02	.77±.00	.62±.00	.78±.00	.57±.00	.74±.00	.63± .01	.77±.00		

Table 6-3: Results on full dataset except class 4. Average score and standard deviation over 5 runs. The average (m) is the macro average and the average (w) is the weighted average

These results of using trained GANs to oversample minority class IMS spectra are verified on a different dataset. This experiment is included in Appendix A. Some example spectra are included in Appendix E.

Conclusion and Recommendations

This work introduces Generative Adversarial Nets (GANs) for Imaging Mass Spectrometry data (IMS). GANs can be used to generate data that can be used for oversampling minority class data to increase classifier performance. GANs have been shown to outperform other oversampling techniques such as SMOTE on different datasets.

In this work, Wasserstein GANs with gradient penalty is implemented to oversample minority classes of an IMS dataset. Specifically, GANs are used to generate spectra of multiple different classes.

Training the original GANs to generate spectra proved unsuccessful. By studying the performance of GANs for a different number of classes and with reduced spectra length. The final conclusion is reached by using cWGAN-gp on a reduced spectrum size. The GANs can replicate the data well based on a classifier performance test. However, using the GANs to oversample the minority classes does not increase overall classifier performance.

On this dataset GANs are outperformed by SMOTE, leading to the conclusion that GANs should not be used on for the task of oversampling on this dataset. This result is verified on a second dataset, leading to a similar conclusion. Testing on different datasets, e.g. in a binary classification setting, could still be a good addition and next step to this research. Additionally, other modifications or network architectures can be explored. For example, deep convolutional GANs have not been thoroughly explored in this work.

Other steps to further improve on the generative capability of the implemented GANs that have not been explored in this work are feature matching[102] or using the AC-GANs[27] instead of the conditional GANs.

GANs are commonly used for image generation. Using GANs on images generated by selecting a single mass bin can be used with GANs to for instance generate spatial views of non-existent mass bins. A direct application for this is not clear to us.

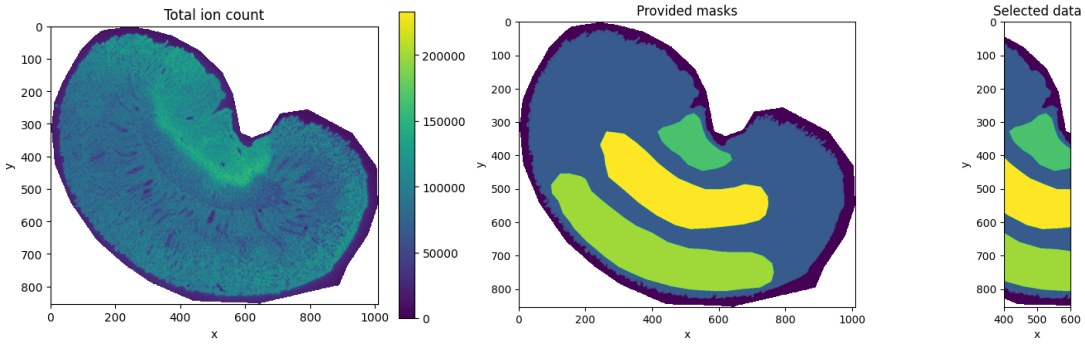
Other research directions using GANs should also not be discarded. GANs used for anomaly detection might be a huge opportunity in IMS data. In the appendix, a short experiment already shows promising results. These results were created by using a reduced size spectrum (using a lower number of features) in accordance with the previous conclusions.

Appendix A

Murine Kidney

The results presented in this report are verified on a second dataset. This dataset is an IMS dataset of a healthy Murine kidney. The main difference between this dataset and the dataset used for the majority of the dataset is how the spectra are divided into classes. In the presented work on the diseased kidney, NMF was used as a clustering method, while the healthy kidney data included masks provided by an expert to point out regions of interest in the data. Figure A-1a shows the total ion count image of this kidney, Figure A-1c shows the masks of the data.

This dataset includes more pixels/ spectra (591534). And the spectra contain 1428 m/z bins. To decrease the time of this experiment only a part of the pixels is selected. The data is sliced as shown in Figure A-1c, this part of the data still contains all classes.



(a) **(b)** **(c)**

Figure A-1: (a) total ion count of the Murine kidney, (b) provided masks; purple: measured pixels, blue: acquisition mask, green: cortex mask, yellow: medulla mask, turquoise: cortex mask, (c) data selected for experimentation

The selected data contains 141607 spectra divided over 5 classes as indicated by the different colors shown in Figure A-1. All these classes have a relatively larger number of samples as shown in Table A-1. There is no obvious effect of class imbalance if the data is split normally with 5000 samples of every class in the training data as shown in Table A-2.

The results below are created using 50 features based on an NMF approximation of the data as seen in the experiments in chapter 6.

Class	Samples
1	15665
2	54791
3	25424
4	27443
5	18284

Table A-1: Total number of samples of every class

Class	Precision	Recall	Training	Test
1	0.90	0.96	5000	10665
2	0.94	0.55	5000	49791
3	0.74	0.95	5000	20424
4	0.67	0.97	5000	22443
5	0.72	0.95	5000	13284
Average (m)	0.97	0.87		
Average (w)	0.82	0.78		

Table A-2: Results with equal training size

A-1 Baseline

To simulate the effect of class imbalance, class 4 is undersampled. From all classes, 5000 samples are used for training, except class 4 from which only 500 samples are used. Comparing Table A-3 to Table A-2 shows that the effect of undersampling on this dataset is very limited; the average scores stay the same. The effect of SMOTE on this dataset is similar to the the results presented before: the classification overfits toward the minority class but the average scores stay similar to the unbalanced results.

	No oversampling		SMOTE			
Class	Precision	Recall	Precision	Recall	Training	Test
1	0.88	0.96	0.90	0.95	5000	10665
2	0.86	0.61	0.94	0.55	5000	49791
3	0.74	0.95	0.75	0.94	5000	20424
4	0.75	0.84	0.71	0.97	500	26943
5	0.73	0.95	0.72	0.95	5000	13284
Average (m)	0.97	0.87	0.80	0.87		
Average (w)	0.82	0.78	0.83	0.79		

Table A-3: Baseline results, the gray row indicates the undersampled class.

A-2 GANs results

For this dataset the GANs designed in chapter 5 are not changed.

Similar to the results presented in the main part of this work the quality of GANs generated data decreases as the number of features increases. This is shown in Figure A-2 where the most left pictures shows the recall matrix of an LDA classifier fitted to GAN-generated data where the GANs are trained on full spectra. Using 200 components, shown in the middle image, shows a significant increase in performance but further decreasing the number of features in this case improves the accuracy even further. This is the same result as seen in subsection 6-2-3.

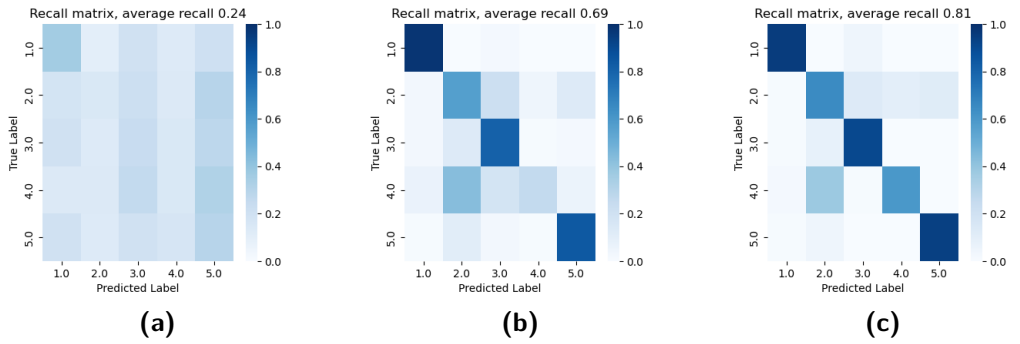


Figure A-2: Classifier scores with, (a) 1423 (all) features, (b) 200 features, and (c) 50 features

Based on the results seen in subsection 6-2-3 and the results given in Figure A-2 50 features are selected for the training of GANs on this dataset. The results are presented in Table A-4. Similar to what was observed in the main study, using SMOTE for oversampling leads to an increased recall and a decreased precision of the minority class. However, for this dataset, with artificially undersampled classes SMOTE does not increase overall performance.

GANs in this case even slightly decrease overall performance meaning that the generated data is not a perfect representation of the data distribution.

	No oversampling		Augmentation		SMOTE		GANs			
Class	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Training	Test
1	0.88	0.96	0.88	0.96	0.90	0.95	0.89	0.92	5000	10665
2	0.86	0.61	0.95	0.52	0.94	0.55	0.84	0.61	5000	49791
3	0.74	0.95	0.74	0.93	0.75	0.94	0.74	0.94	5000	20424
4	0.75	0.84	0.70	0.97	0.71	0.97	0.74	0.83	500	26943
5	0.73	0.95	0.68	0.97	0.72	0.95	0.71	0.95	5000	13284
Average (m)	0.97	0.87	0.79	0.87	0.80	0.87	0.79	0.86		
Average (w)	0.82	0.78	0.82	0.78	0.83	0.79	0.79	0.78		

Table A-4: Baseline results, the gray row indicates the undersampled class.

A-2-1 Conclusion

This experiment underlines the results found in the chapter 6. Applying the conditional Wasserstein GANs with gradient penalty to the full spectra is unsuccessful. Reducing the

number of features increases the performance of GANs as shown in Figure A-2. However, even with a low number of values, the GANs do not increase overall performance. In this experiment, SMOTE also did not affect the overall performance in a meaningful way.

Appendix B

Anomaly detection

GANs can be used for anomaly detection by training on only 'normal' data. After successful training, the discriminator function will predict unseen data that is different from the 'normal' (anomalous) to be fake. This property of the discriminator can be exploited for semi-supervised learning. Here is assumed that a small part of the data is known to be healthy or non-anomalous. Training the GANs on this small part of data should create a discriminator that can be used to identify unhealthy or anomalies in unseen data.

Using the kidney with bacterial colonies, the anomaly detection property of GANs can be visualized with a small experiment. Of this dataset it is known that 1. bacterial colonies are present in the middle of the sample, 2. folding effects influence some of the spectra located near the edges. Therefore, a simple mask is created to filter out a small portion of data that is almost certainly healthy. This mask is given in Figure B-1.

In the total ion image, some bright-colored regions already indicate the locations of bacterial colonies, however not every bright spot corresponds to anomalous data. Training the original

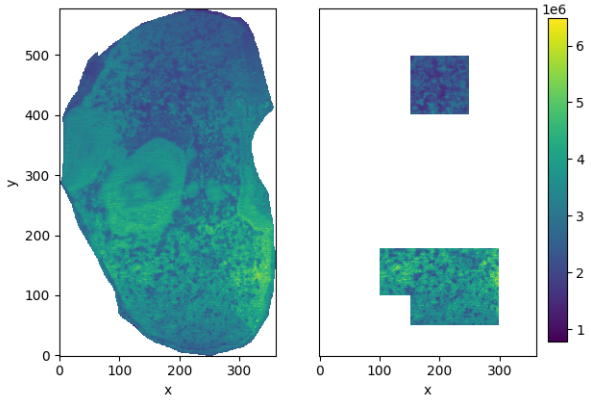


Figure B-1: On the left, the full total-ion-count image, on the right the total ion count of the spectra (in the corresponding location) selected as healthy.

GANs on the selected data and then using the discriminator on all spectra gives predictions

in the left of Figure B-2. In this image, a darker color or higher value means a larger chance of the data being anomalous. For reference, the figure on the right shows the result of K-means, an unsupervised method to distinguish different modes in the data. This test

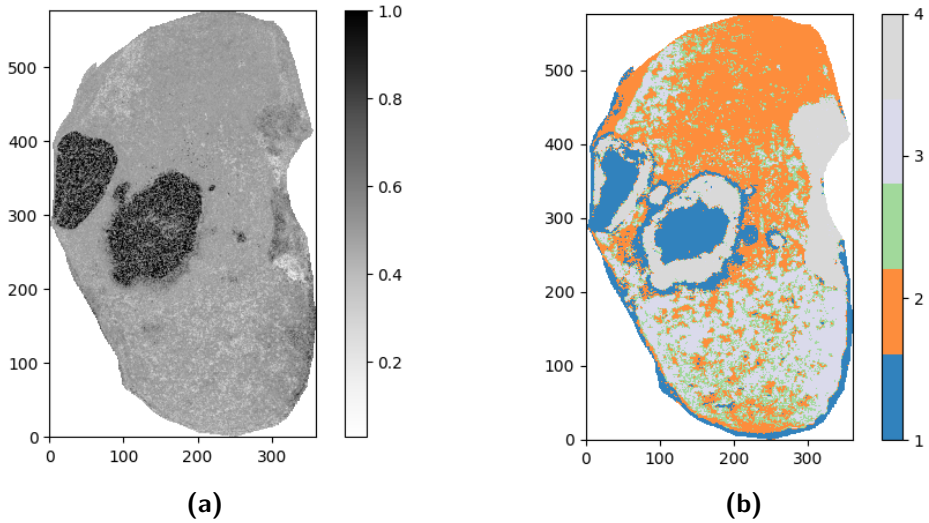


Figure B-2: (a) Prediction of real and fake classes. (b) K-means clustering for reference

shows the capabilities of GANs for anomaly detection. No comparison or validation is used to verify the accuracy of the detected anomalies. In future work, anomalies can be further explored and studied using GANs and maybe even specialized GANs such as GANomaly[40] or AnoGAN[40].

Appendix C

Activation functions

Below an overview is given of the most common activation fucntions.

Linear activation

A linear activation function does not change the input (y=x).

$$f(x) = x \qquad f(x) \in (-\infty, \infty)$$
$$f'(x) = 1$$

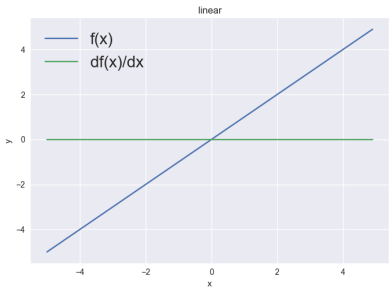


Figure C-1: Linear activation with its derrivative

Sigmoid activation

The sigmoid activation is often used for binary classification, the (rounded) output 0 corresponds to the first and the (rounded) output 1 corresponds to the second class. A generator in GANs can use a sigmoid activation on the final layer if the data is scaled accordingly (to (0,1)).

$$f(x) = \frac{1}{1+e^{-x}} \qquad f(x) \in (0,1)$$
$$f'(x) = f(x)(1-f(x)) \qquad f'(x) \in (0, \frac{1}{4}]$$

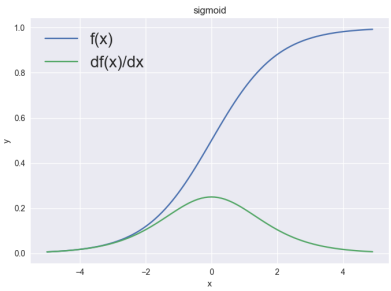


Figure C-2: Sigmoid activation with its derivative

Tanh activation

The Tangent hyperbolic activation function is very similar to the sigmoid function. The range is doubled (-1 to 1, instead of 0 to 1). The main important difference is the derivative, which takes a larger value for the Tanh close to 0.

$$f(x) = tanh(x) \qquad f(x) \in (-1, 1)$$
$$f'(x) = 1 - tanh^2(x) \qquad f'(x) \in [1, 0)$$

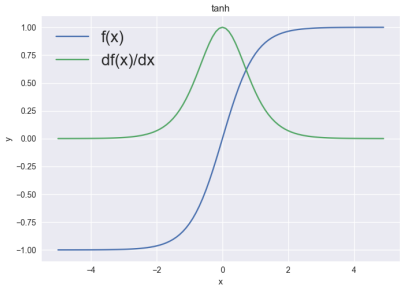


Figure C-3: Tanh activation with its derrivative

ReLU activation

The Rectifier Linear Unit function is equal to the linear activation for positive inputs. It is zero for negative inputs. This activation function is often used in intermediate layers of a neural network.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \qquad f(x) \in [0, \infty)$$
$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases} \qquad f'(x) \in \{0, 1\}$$

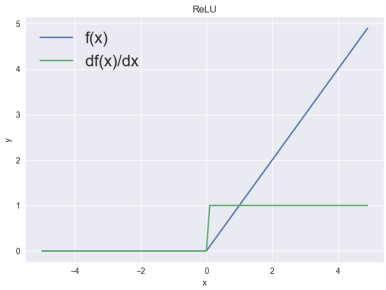


Figure C-4: ReLU with its derivative

Leaky ReLU activation

Similar to ReLU, except Leaky ReLU can have negative values. Using this layer can lead to faster training as Leaky ReLU has a nonzero gradient for negative inputs.

$$f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha x, & \text{otherwise} \end{cases} \qquad f(x) \in (-\infty, \infty)$$
$$f'(x) = \begin{cases} 1, & \text{if } x > 0 \\ \alpha, & \text{otherwise} \end{cases} \qquad f'(x) \in \{\alpha, 1\}$$

With $0 < \alpha < 1$

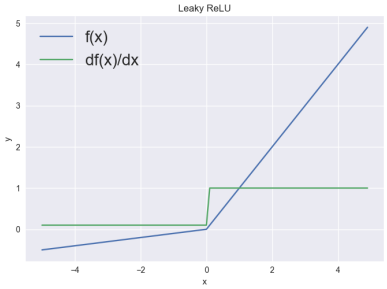


Figure C-5: Leaky ReLU with its derivative ($\alpha = 0.1$)

Softmax activation

The softmax activation function takes a vector as input and produces a vector as output. The output vector corresponds to probabilities. This can be used for multiclass classification. Where every entry in the output vector corresponds to the probability of the input corresponding to the class. In the equations below, K is the number of classes. x_i and x_j denote

the i'th and j'th value of the vector $\mathbf{x}$

$$f(\mathbf{x}) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \qquad \mathbf{x} \in (0,1)^K$$

In the example below the input to the NN corresponds is predicted to correspond to the second class (as the second entry of the output vector has the highest probability).

$$\begin{bmatrix} 1.3 \\ 5.1 \\ 2.1 \\ 0.7 \\ 1.1 \end{bmatrix} \longrightarrow \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \longrightarrow \begin{bmatrix} 0.02 \\ 0.90 \\ 0.05 \\ 0.01 \\ 0.02 \end{bmatrix}$$

Appendix D

Additional neural net designs

Multiple designs are tested as discussed in chapter 5. The designed are summarized below. Testing with e.g. different dropout rates and different numbers of neurons per layer is not discussed in this document.

D-1 Discriminator designs

D-1-1 Fully connected networks

The main focus of the different designs discussed in this work are on the number of layers in a fully connected neural network. For the discriminator the number of neurons in every hidden layer is kept the same throughout the different designs. The output layer of discrimination is a scaler by definition and the inputsize is dependent on the number of (embedded) classes and the number of features in the dataset.

Single linear connection fc(1)
The simplest possible discriminator network is a direct connection between the in and output.

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	592 (573+19)	1	593 (592x1+1)
Sigmoid			- -
	Trainable parameters:		954

Table D-1: Discriminator design with 1 linear connection. The sigmoid output on the final layer is omitted in the Wasserstein GANs

Four linear connections fc(4)
The main design used in this work has three linear connections (see chapter 5). Adding an additional layer results in the design given below in Table D-2.

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	592 (573+19)	512	303616 (592x512+512)
LeakyReLU	512	512	- -
Linear	512	512	262656 (512x512+512)
Dropout (0.4)			- -
LeakyReLU	512	512	- -
Linear	512	512	262656 (512x512+512)
Dropout (0.4)			- -
LeakyReLU	512	512	- -
Linear	512	1	513 (512x1+1)
Sigmoid			- -
	Trainable parameters:		829411

Table D-2: Discriminator design with 4 linear connections. The sigmoid output on the final layer is omitted in the Wasserstein GANs

D-1-2 Convolutional designs

DC with 8 channels DC(8)
In addition to the fully connected networks, some testing with varying convolutional networks was done as well. The chosen kernel size is included in the brackets. The number of output channels of the convolutions is added as a column. The stride for all convolutions is equal to 2. This design is given in Table D-3

Layer type	Output channels	Input shape	Output shape	Number of parameters
Embedding	1	19	19	361 (19x19)
Linear	1	592 (573+19)	512	303616 (592x512+512)
LeakyReLU	1	512	512	- -
Convolution (5)	8	512	8x258	48
LeakyReLU	8	32x258	8x258	- -
Dropout (0.4)	8			- -
Convolution (4)	16	32x258	16x131	528
LeakyReLU	16	16x131	16x131	- -
Dropout (0.4)	16			- -
Convolution (4)	32	16x131	32x67	2080
LeakyReLU	32	32x67	32x67	- -
Dropout (0.4)	32			- -
Convolution (4)	64	128x67	64x35	8256
LeakyReLU	64	64x35	64x35	- -
Dropout (0.4)	64			- -
Flatten	1	64x35	2240	- -
Linear	1	2240	1	2241 (2240x1+1)
Sigmoid				- -
			Trainable parameters:	317130

Table D-3: Discriminator design with characteristic 8. The sigmoid output on the final layer is omitted in the Wasserstein GANs

DC with 32 channels DC(32)
An additional convolutional network is created for comparison with a different channel scaling. Where the previous example doubles the number of channels starting at 8 this alternative design given in Table D-8 starts with 32 output channels of the first convolution layer.

Layer type	Output channels	Input shape	Output shape	Number of parameters
Embedding	1	19	19	361 (19x19)
Linear	1	592 (573+19)	512	303616 (592x512+512)
LeakyReLU	1	512	512	- -
Convolution (5)	32	512	32x258	192
LeakyReLU	32	32x258	32x258	- -
Dropout (0.4)	32			- -
Convolution (4)	64	32x258	64x131	8256
LeakyReLU	64	64x131	64x131	- -
Dropout (0.4)	64			- -
Convolution (4)	128	64x131	128x67	32896
LeakyReLU	128	128x67	128x67	- -
Dropout (0.4)	128			- -
Convolution (4)	2556	128x67	256x35	131328
LeakyReLU	256	256x35	256x35	- -
Dropout (0.4)	256			- -
Flatten	1	256x35	8960	- -
Linear	1	8960	1	8961 (8961x1+1)
Sigmoid				- -
			Trainable parameters:	485610

Table D-4: Discriminator design with characteristic 32. The sigmoid output on the final layer is omitted in the Wasserstein GANs

D-2 Generator designs

For the generator design the number of numbers is increased in each added layer. The number of inputs is dependent on the number of classes and the size of the latent vector. Below the generator designs tested in chapter 5 are summarized.

D-2-1 Fully connected networks

Single linear connection fc(1)

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	119 (100+19)	573	68760 (119x573+128)
Batchnormalization	573	573	- -
		Trainable parameters:	69121

Table D-5: Generator design with 1 linear connection.

Three linear connections fc(3)

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	119 (100+19)	128	15360 (119x128+128)
LeakyReLU	128	128	- -
Linear	128	256	33024 (128x256+256)
Batchnormalization	256	256	- -
LeakyReLU	256	256	- -
Linear	256	573	147261 (256x573+573)
Batchnormalization	573	573	- -
	Trainable parameters:		197664

Table D-6: Generator design with 3 linear connections.

Three linear connections fc(5)

Layer type	Input shape	Output shape	Number of parameters
Embedding	19	19	361 (19x19)
Linear	119 (100+19)	128	15360 (119x128+128)
LeakyReLU	128	128	- -
Linear	128	256	33024 (128x256+256)
Batchnormalization	256	256	- -
LeakyReLU	256	256	- -
Linear	256	512	131584 (256x512+512)
Batchnormalization	512	512	- -
LeakyReLU	512	512	- -
Linear	512	1024	525312 (512x1024+573)
Batchnormalization	1024	1024	- -
LeakyReLU	1024	1024	- -
Linear	1024	573	587325 (1024x573+573)
Batchnormalization	573	573	- -
	Trainable parameters:		1296550

Table D-7: Generator design with 5 linear connections.

D-2-2 Convolutional design

A single convolutional network as the generator is tested. Here, the deconvolution or transpose-convolution operation is used to decode a lower dimensional vector to a higher dimensional space. The number of channels is first increased from 1 to 8 and then the number of channels is lowered while the signal length increases.

Layer type	Output channels	Input shape	Output shape	Number of parameters
Embedding	1	19	19	361 (19x19)
Linear	1	119 (100+19)	128	15360 (119x128+128)
Batchnormalization	1			2
LeakyReLU	1	128	128	- -
Deconvolution (4)	8	128	8x256	40
Batchnormalization	8			16
LeakyReLU	8	8x256	8x256	- -
Deconvolution (4)	4	4x512	4x512	132
Batchnormalization	4			8
LeakyReLU	4	64x131	64x131	- -
Deconvolution (4)	2	4x512	2x1024	34
Batchnormalization	2			4
LeakyReLU	2	128x67	128x67	- -
Deconvolution (4)	1	2x1024	1x1025	9
Batchnormalization	1			2
LeakyReLU	1	1x1025	1x1025	- -
Linear	1	1025	573	587898 (1025x573+573)
Sigmoid				- -
			Trainable parameters:	605054

Table D-8: Generator design with characteristic 8. Kernel of 4 in all layers

Appendix E

Example spectra

Below, some randomly selected examples are given of original and generated data. All data is scaled back to its original scale. The generated spectra are created by a generator trained using cWGAN-gp with 50 NMF features. The real spectra are taken from the test set. Each figure shows 5 spectra of the mentioned class.

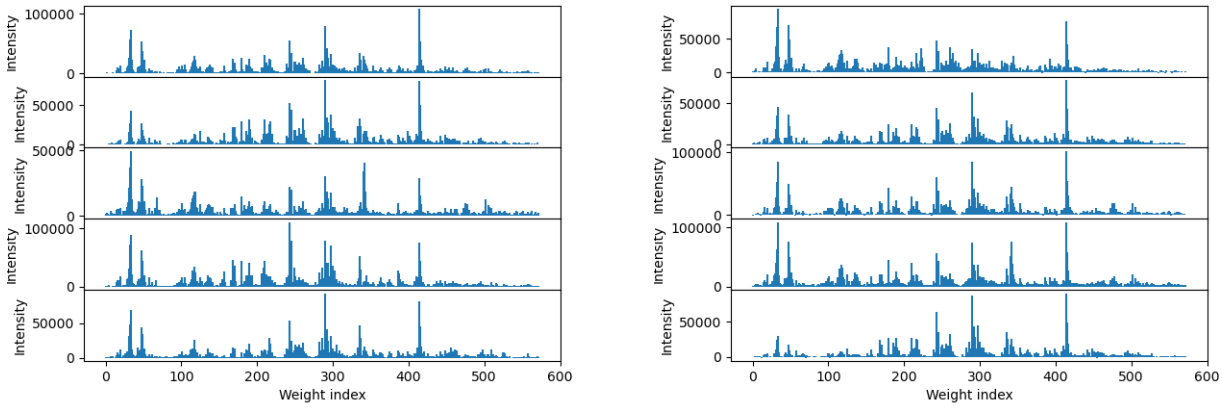
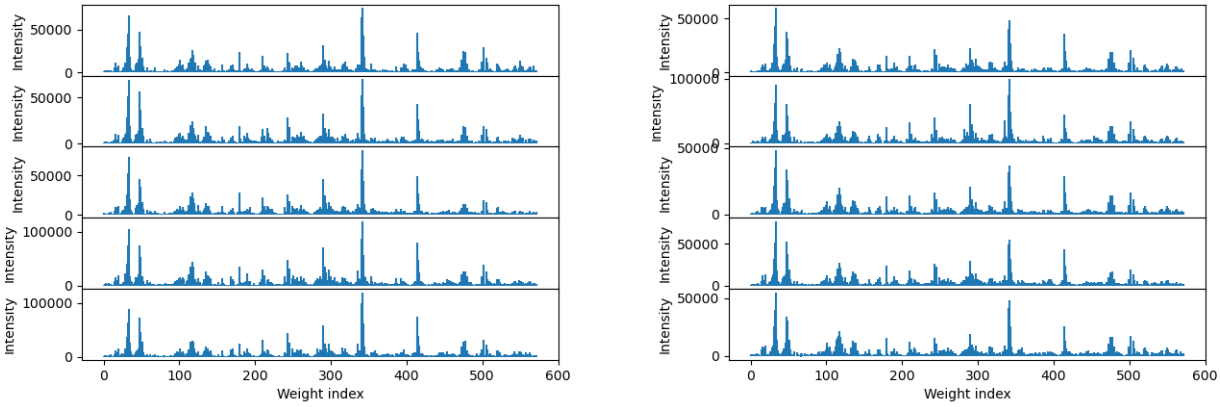
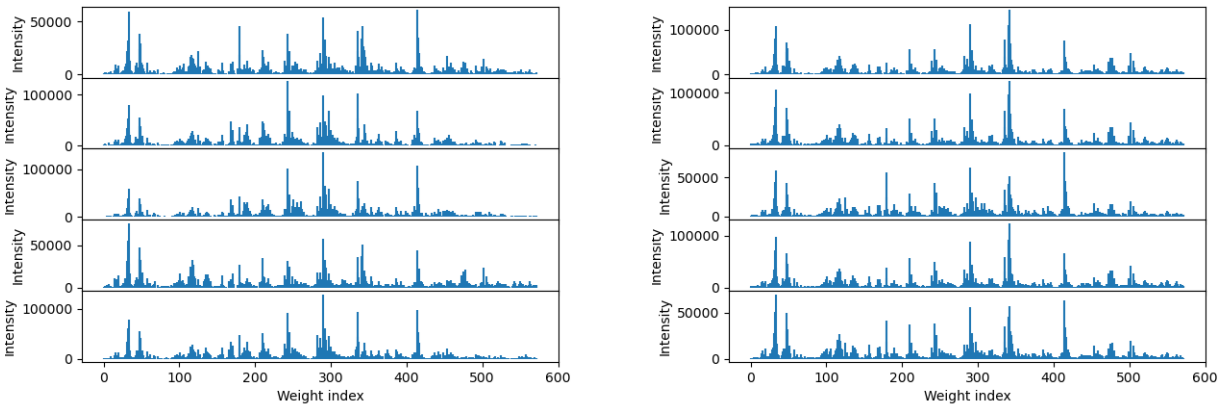


Figure E-1: (a) real and (b) generated spectra, both of class 1.



(a) **(b)**

Figure E-2: (a) real and (b) generated spectra, both of class 8.



(a) **(b)**

Figure E-3: (a) real and (b) generated spectra, both of class 16.

Bibliography

[1] J. Engelmann and S. Lessmann, "Conditional wasserstein gan-based oversampling of tabular data for imbalanced learning," *Expert Systems with Applications*, vol. 174, p. 114582, 2021.

[2] C. Charitou, S. Dragicevic, and A. d. Garcez, "Synthetic data generation for fraud detection using gans," *arXiv preprint arXiv:2109.12546*, 2021.

[3] R. Goodwin, J. Bunch, and D. McGinnity, "Mass spectrometry imaging in oncology drug discovery," *Advances in cancer research*, vol. 134, pp. 133–171, 2017.

[4] H. Meistermann, J. L. Norris, H.-R. Aerni, D. S. Cornett, A. Friedlein, A. R. Erskine, A. Augustin, M. C. D. V. Mudry, S. Ruepp, L. Suter, *et al.*, "Biomarker discovery by imaging mass spectrometry: transthyretin is a biomarker for gentamicin-induced nephrotoxicity in rat," *Molecular & Cellular Proteomics*, vol. 5, no. 10, pp. 1876–1886, 2006.

[5] M. Aichler and A. Walch, "Maldi imaging mass spectrometry: current frontiers and perspectives in pathology research and practice," *Laboratory investigation*, vol. 95, no. 4, pp. 422–431, 2015.

[6] A. Nilsson, R. J. Goodwin, M. Shariatgorji, T. Vallianatou, P. J. Webborn, and P. E. Andrén, "Mass spectrometry imaging in drug development," *Analytical chemistry*, vol. 87, no. 3, pp. 1437–1455, 2015.

[7] E. A. Jones, S.-O. Deininger, P. C. Hogendoorn, A. M. Deelder, and L. A. McDonnell, "Imaging mass spectrometry statistical analysis," *Journal of proteomics*, vol. 75, no. 16, pp. 4962–4989, 2012.

[8] T. Alexandrov, "Maldi imaging mass spectrometry: statistical data analysis and current computational challenges," *BMC bioinformatics*, vol. 13, no. Suppl 16, p. S11, 2012.

[9] D. Pietkiewicz, S. Plewa, M. Zaborowski, T. J. Garrett, E. Matuszewska, Z. J. Kokot, and J. Matysiak, "Mass spectrometry imaging in gynecological cancers: the best is yet to come," *Cancer Cell International*, vol. 22, no. 1, p. 414, 2022.

[10] E. R. A. van Hove, D. F. Smith, and R. M. Heeren, "A concise review of mass spectrometry imaging," *Journal of chromatography A*, vol. 1217, no. 25, pp. 3946–3954, 2010.

[11] S. S. Rubakhin and J. V. Sweedler, "Mass spectrometry imaging," *Rubakhin SS, Sweedler JV,(Eds.)*, pp. 21–49, 2010.

[12] K. K. Murray, "Resolution and resolving power in mass spectrometry," *Journal of the American Society for Mass Spectrometry*, vol. 33, no. 12, pp. 2342–2347, 2022.

[13] C. Bauer, R. Cramer, and J. Schuchhardt, "Evaluation of peak-picking algorithms for protein mass spectrometry," in *Data Mining in Proteomics: From Standards to Applications*, pp. 341–352, Springer, 2010.

[14] F. Falcetta, L. Morosi, P. Ubezio, S. Giordano, A. Decio, R. Giavazzi, R. Frapolli, M. Prasad, P. Franceschi, M. D'Incalci, *et al.*, "Past-in-the-future. peak detection improves targeted mass spectrometry imaging," *Analytica Chimica Acta*, vol. 1042, pp. 1–10, 2018.

[15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," vol. 3, p. 2672 – 2680, 2014. Cited by: 33546.

[16] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International conference on machine learning*, pp. 214–223, PMLR, 2017.

[17] S. Sidheekh, A. Aimen, and N. C. Krishnan, "On characterizing gan convergence through proximal duality gap," in *International Conference on Machine Learning*, pp. 9660–9670, PMLR, 2021.

[18] F. Farnia and A. Ozdaglar, "Do gans always have nash equilibria?," in *International Conference on Machine Learning*, pp. 3029–3039, PMLR, 2020.

[19] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[20] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville, "Improved training of wasserstein gans," *Advances in neural information processing systems*, vol. 30, 2017.

[21] C. A. Floudas, *Nonlinear and Mixed-Integer Optimization*. Oxford University Press, 1995.

[22] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for gans do actually converge?," in *International conference on machine learning*, pp. 3481–3490, PMLR, 2018.

[23] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2016. Cited by: 2486.

[24] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 4401–4410, 2019.

[25] A. Sauer, T. Karras, S. Laine, A. Geiger, and T. Aila, "Stylegan-t: Unlocking the power of gans for fast large-scale text-to-image synthesis," *arXiv preprint arXiv:2301.09515*, 2023.

[26] V. Sampath, I. Maurtua, J. J. Aguilar Martin, and A. Gutierrez, "A survey on generative adversarial networks for imbalance problems in computer vision tasks," *Journal of big Data*, vol. 8, pp. 1–59, 2021.

[27] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans," in *International conference on machine learning*, pp. 2642–2651, PMLR, 2017.

[28] S. Kamal, A. Mujeeb, M. Supriya, *et al.*, "Generative adversarial learning for improved data efficiency in underwater target classification," *Engineering Science and Technology, an International Journal*, vol. 30, p. 101043, 2022.

[29] M. Kang, W. Shim, M. Cho, and J. Park, "Rebooting acgan: Auxiliary classifier gans with stable training," *Advances in neural information processing systems*, vol. 34, pp. 23505–23518, 2021.

[30] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.

[31] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," *Advances in neural information processing systems*, vol. 29, 2016.

[32] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially learned inference," *arXiv preprint arXiv:1606.00704*, 2016.

[33] S. Mukherjee, H. Asnani, E. Lin, and S. Kannan, "Clustergan: Latent space clustering in generative adversarial networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 4610–4617, 2019.

[34] J. Yang, S. Li, Z. Wang, H. Dong, J. Wang, and S. Tang, "Using deep learning to detect defects in manufacturing: a comprehensive survey and current challenges," *Materials*, vol. 13, no. 24, p. 5755, 2020.

[35] J. Liu, J. Guo, P. Orlik, M. Shibata, D. Nakahara, S. Mii, and M. Takáč, "Anomaly detection in manufacturing systems using structured neural networks," in *2018 13th world congress on intelligent control and automation (wcica)*, pp. 175–180, IEEE, 2018.

[36] T. Fernando, H. Gammulle, S. Denman, S. Sridharan, and C. Fookes, "Deep learning for medical anomaly detection–a survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 7, pp. 1–37, 2021.

[37] A. Ukil, S. Bandyoapdhyay, C. Puri, and A. Pal, "Iot healthcare analytics: The importance of anomaly detection," in *2016 IEEE 30th international conference on advanced information networking and applications (AINA)*, pp. 994–997, IEEE, 2016.

[38] D. Lasaga and P. Santhana, "Deep learning to detect medical treatment fraud," in *Proceedings of the KDD 2017: Workshop on Anomaly Detection in Finance* (A. Anandakrishnan, S. Kumar, A. Statnikov, T. Faruquie, and D. Xu, eds.), vol. 71 of *Proceedings of Machine Learning Research*, pp. 114–120, PMLR, 14 Aug 2018.

[39] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs, "Unsupervised anomaly detection with generative adversarial networks to guide marker discovery," in *Information Processing in Medical Imaging: 25th International Conference, IPMI 2017, Boone, NC, USA, June 25-30, 2017, Proceedings*, pp. 146–157, Springer, 2017.

[40] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon, "Ganomaly: Semi-supervised anomaly detection via adversarial training," in *Computer Vision–ACCV 2018: 14th Asian Conference on Computer Vision, Perth, Australia, December 2–6, 2018, Revised Selected Papers, Part III 14*, pp. 622–637, Springer, 2019.

[41] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient gan-based anomaly detection," *arXiv preprint arXiv:1802.06222*, 2018.

[42] F. Di Mattia, P. Galeone, M. De Simoni, and E. Ghelfi, "A survey on gans for anomaly detection," *arXiv preprint arXiv:1906.11632*, 2019.

[43] A. Borji, "Pros and cons of gan evaluation measures," *Computer Vision and Image Understanding*, vol. 179, p. 41 – 65, 2019. Cited by: 319; All Open Access, Green Open Access.

[44] A. Borji, "Pros and cons of gan evaluation measures: New developments," *Computer Vision and Image Understanding*, vol. 215, p. 103329, 2022.

[45] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[46] E. L. Denton, S. Chintala, R. Fergus, *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," *Advances in neural information processing systems*, vol. 28, 2015.

[47] S. Tang, "Lessons learned from the training of gans on artificial datasets," *IEEE Access*, vol. 8, pp. 165044–165055, 2020.

[48] C. R. Givens and R. M. Shortt, "A class of wasserstein metrics for probability distributions.," *Michigan Mathematical Journal*, vol. 31, no. 2, pp. 231–240, 1984.

[49] J. Kim and H. Park, "Sparse nonnegative matrix factorization for clustering," tech. rep., Georgia Institute of Technology, 2008.

[50] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[51] J. Hemmerich, E. Asilar, and G. F. Ecker, "Cover: conformational oversampling as data augmentation for molecules," *Journal of cheminformatics*, vol. 12, no. 1, p. 18, 2020.

[52] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[53] A. Fernández, S. Garcia, F. Herrera, and N. V. Chawla, "Smote for learning from imbalanced data: progress and challenges, marking the 15-year anniversary," *Journal of artificial intelligence research*, vol. 61, pp. 863–905, 2018.

[54] L. Chen, T. Zhang, and T. Li, "Gradient boosting model for unbalanced quantitative mass spectra quality assessment," in *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, pp. 394–399, IEEE, 2017.

[55] S. Shahryari Fard, *Improving protein identification in mass spectrometry imaging using machine learning and spatial spectral information*. PhD thesis, Université d'Ottawa/University of Ottawa, 2022.

[56] A. S. Hussein, T. Li, C. W. Yohannese, and K. Bashir, "A-smote: A new preprocessing approach for highly imbalanced datasets by improving smote," *International Journal of Computational Intelligence Systems*, vol. 12, no. 2, pp. 1412–1422, 2019.

[57] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning," *Journal of Machine Learning Research*, vol. 18, no. 17, pp. 1–5, 2017.

[58] J. Behrmann, C. Etmann, T. Boskamp, R. Casadonte, J. Kriegsmann, and P. Maaβ, "Deep learning for tumor classification in imaging mass spectrometry," *Bioinformatics*, vol. 34, no. 7, pp. 1215–1223, 2018.

[59] J. Promchan, D. Günther, A. Siripinyanond, and J. Shiowatana, "Elemental imaging and classifying rice grains by using laser ablation inductively coupled plasma mass spectrometry and linear discriminant analysis," *Journal of Cereal Science*, vol. 71, pp. 198–203, 2016.

[60] S. A. Thomas, Y. Jin, J. Bunch, and I. S. Gilmore, "Enhancing classification of mass spectrometry imaging data with deep neural networks," in *2017 IEEE symposium series on computational intelligence (SSCI)*, pp. 1–8, IEEE, 2017.

[61] J. Xie and Z. Qiu, "The effect of imbalanced data sets on lda: A theoretical and empirical analysis," *Pattern recognition*, vol. 40, no. 2, pp. 557–562, 2007.

[62] S. A. Thomas, A. M. Race, R. T. Steven, I. S. Gilmore, and J. Bunch, "Dimensionality reduction of mass spectrometry imaging data using autoencoders," in *2016 IEEE symposium series on computational intelligence (SSCI)*, pp. 1–7, IEEE, 2016.

[63] A. M. Race, R. T. Steven, A. D. Palmer, I. B. Styles, and J. Bunch, "Memory efficient principal component analysis for the dimensionality reduction of large mass spectrometry imaging data sets," *Analytical chemistry*, vol. 85, no. 6, pp. 3071–3078, 2013.

[64] A. Jetybayeva, N. Borodinov, A. V. Ievlev, M. I. U. Haque, J. Hinkle, W. A. Lamberti, J. C. Meredith, D. Abmayr, and O. S. Ovchinnikova, "A review on recent machine learning applications for imaging mass spectrometry studies," *Journal of Applied Physics*, vol. 133, no. 2, 2023.

[65] Z. Zhou, X. Zhai, and C. Tin, "Fully automatic electrocardiogram classification system based on generative adversarial network with auxiliary classifier," *Expert Systems with Applications*, vol. 174, p. 114809, 2021.

[66] J. J. Jeong, A. Tariq, T. Adejumo, H. Trivedi, J. W. Gichoya, and I. Banerjee, "Systematic review of generative adversarial networks (gans) for medical image classification and segmentation," *Journal of Digital Imaging*, vol. 35, no. 2, pp. 137–152, 2022.

[67] A. Viloria, O. B. P. Lezama, and N. Mercado-Caruzo, "Unbalanced data processing using oversampling: machine learning," *Procedia Computer Science*, vol. 175, pp. 108–113, 2020.

[68] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[69] J. L. Leevy, T. M. Khoshgoftaar, R. A. Bauder, and N. Seliya, "A survey on addressing high-class imbalance in big data," *Journal of Big Data*, vol. 5, no. 1, pp. 1–30, 2018.

[70] D. J. Dittman, T. M. Khoshgoftaar, R. Wald, and A. Napolitano, "Comparison of data sampling approaches for imbalanced bioinformatics data," in *The twenty-seventh international FLAIRS conference*, 2014.

[71] X.-Y. Liu, J. Wu, and Z.-H. Zhou, "Exploratory undersampling for class-imbalance learning," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 2, pp. 539–550, 2008.

[72] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[73] F. Grina, Z. Elouedi, and E. Lefevre, "A preprocessing approach for class-imbalanced data using smote and belief function theory," in *Intelligent Data Engineering and Automated Learning–IDEAL 2020: 21st International Conference, Guimaraes, Portugal, November 4–6, 2020, Proceedings, Part II 21*, pp. 3–11, Springer, 2020.

[74] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[75] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[76] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[77] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," *arXiv preprint arXiv:1611.02163*, 2016.

[78] J. Chen, J. Konrad, and P. Ishwar, "Vgan-based image representation learning for privacy-preserving facial expression recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 1570–1579, 2018.

[79] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 8110–8119, 2020.

[80] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[81] V. Sorin, Y. Barash, E. Konen, and E. Klang, "Creating artificial images for radiology applications using generative adversarial networks (gans)–a systematic review," *Academic radiology*, vol. 27, no. 8, pp. 1175–1185, 2020.

[82] M. Rezaei, T. Uemura, J. Näppi, H. Yoshida, C. Lippert, and C. Meinel, "Generative synthetic adversarial network for internal bias correction and handling class imbalance problem in medical image diagnosis," in *Medical Imaging 2020: Computer-Aided Diagnosis*, vol. 11314, pp. 82–89, SPIE, 2020.

[83] Q. Wang, X. Zhou, C. Wang, Z. Liu, J. Huang, Y. Zhou, C. Li, H. Zhuang, and J.-Z. Cheng, "Wgan-based synthetic minority over-sampling technique: Improving semantic fine-grained classification for lung nodules in ct images," *IEEE Access*, vol. 7, pp. 18450–18463, 2019.

[84] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz, and B. Catanzaro, "High-resolution image synthesis and semantic manipulation with conditional gans," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[85] S. Zhou, M. Gordon, R. Krishna, A. Narcomey, L. F. Fei-Fei, and M. Bernstein, "Hype: A benchmark for human eye perceptual evaluation of generative models," *Advances in neural information processing systems*, vol. 32, 2019.

[86] M. Lucic, K. Kurach, M. Michalski, S. Gelly, and O. Bousquet, "Are gans created equal? a large-scale study," *Advances in neural information processing systems*, vol. 31, 2018.

[87] C.-I. Kim, M. Kim, S. Jung, and E. Hwang, "Simplified fréchet distance for generative adversarial nets," *Sensors*, vol. 20, no. 6, p. 1548, 2020.

[88] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[89] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," *CoRR*, vol. abs/1912.01703, 2019.

[90] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[91] Y. Tian, L. Shen, G. Su, Z. Li, and W. Liu, "Alphagan: Fully differentiable architecture search for generative adversarial networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 10, pp. 6752–6766, 2021.

[92] M. O. S. N. Wolberg, William and W. Street, "Breast Cancer Wisconsin (Diagnostic)." UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C5DW2B.

[93] L. Zhu, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Generative adversarial networks for hyperspectral image classification," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 9, pp. 5046–5063, 2018.

[94] N. Aloysius and M. Geetha, "A review on deep convolutional neural networks," in *2017 international conference on communication and signal processing (ICCSP)*, pp. 0588–0592, IEEE, 2017.

[95] D. Nemirovsky, N. Thiebaut, Y. Xu, and A. Gupta, "Countergan: Generating counterfactuals for real-time recourse and interpretability using residual gans," in *Uncertainty in Artificial Intelligence*, pp. 1488–1497, PMLR, 2022.

[96] X. Gong, S. Chang, Y. Jiang, and Z. Wang, "Autogan: Neural architecture search for generative adversarial networks," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3224–3234, 2019.

[97] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, Massachusetss: The MIT Press, 2016.

[98] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How does batch normalization help optimization?," *Advances in neural information processing systems*, vol. 31, 2018.

[99] A. M. Javid, S. Das, M. Skoglund, and S. Chatterjee, "A relu dense layer to improve the performance of neural networks," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2810–2814, IEEE, 2021.

[100] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[101] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[102] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," *Advances in neural information processing systems*, vol. 29, 2016.

Glossary

List of Acronyms

AC-GAN	Auxiliary Classifier Generative Adversarial Nets
BiGAN	Bidirectional Generative Adversarial Nets
cGANs	conditional Generative Adversarial Nets
cWGANs-gp	conditional Wasserstein Generative Adversarial Nets with gradient panelty
DC-GANs	Deep Convolutional Generative Adversarial Nets
EM	Earth Mover
FD	Fréchet Distance
FID	Fréchet Inception Distance
GANs	Generative Adversarial Nets
IMS	Imaging Mass Spectrometry
MALDI	Matrix Assisted Laser Desorption and Ionization
MS	Mass Spectrometry
NMF	Non-negative matrix factorization
QTOF	Quadrupole Time Of Flight
SMOTE	Synthetic Minority Oversampling Technique
WGANs	Wasserstein Generative Adversarial Nets

List of Symbols

$D(\cdot)$	Discriminator function
$G(\cdot)$	Generator function
Σ	Covariance matrix
$\mathbb{E}_{n \sim p_Q}$	Expectation of n being from distribution Q
μ	Mean

σ	Standard diviation
FN	False Negative
FP	False Positive
m/z	Mass to charge ratio
mM	millimolar
TN	True Negative
TP	True Positive