Karlsruhe Institute of Technology

Subspace Generative Adversarial Learning for Unsupervised Outlier Detection

Bachelor's Thesis of

Denis Wambold

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD)

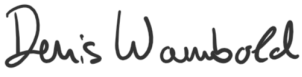
Reviewer:	Prof. Dr.-Ing. Klemens Böhm
Second reviewer:	TT-Prof. Dr. Pascal Friederich
Advisor:	M.Sc. Jose Cribeiro-Ramallo

29. May 2023 – 29. September 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text, and that I have observed the Guidelines for Research Integrity of Good Scientific Practice at the KIT Karlsruhe.

Karlsruhe, 29. September 2023



. . . . . .

(Denis Wambold)

Abstract

In recent years, Deep Learning has witnessed remarkable advancements, with unsupervised generative methods gaining widespread attention as they are applicable across diverse domains. Amongst these methods, Generative Adversarial Networks (GAN) [17] gained traction due to their ability to generate realistic data. With this ability, GANs have proven to yield impressive results for Outlier Detection [24]. While there are many different approaches to connect GANs with Outlier Detection, such as Outlier Detection by a latent subspace [44] or Active Adversarial Learning (AAL) [30], none of these models take into consideration that some outliers are not visible on the full Feature Space [25]. These hidden outliers remain unfound by current state-of-the-art GAN models. In response to this challenge, we introduce a novel extension of the AAL framework, connecting GANs with an Ensemble of Feature Subspaces. We propose the Feature Ensemble GAN (FeGAN), a GAN specifically tailored for Subspace Search with multiple Discriminators, each over their own, unique Feature Subspace. This innovative approach uncovers outliers that may exist exclusively within specific Feature Subspaces, an aspect otherwise overlooked by conventional methods.

Our experiments demonstrate the promising results and great potential of FeGAN for Outlier Detection. FeGAN consistently outperforms baseline models, highlighting the significance of identifying outliers that current state-of-the-art methods often miss.

Zusammenfassung

In den letzten Jahren hat Deep Learning bemerkenswerte Fortschritte gemacht, wobei vor allem unüberwachte, generative Methoden große Aufmerksamkeit erregt haben, da sie in vielen verschiedenen Bereichen anwendbar sind. Unter diesen Methoden haben Generative Adversarial Networks (GAN) [17] aufgrund ihrer Fähigkeit, realistische Daten zu generieren, an Bedeutung gewonnen. Mit dieser Fähigkeit liefern GANs nachweislich beeindruckende Ergebnisse bei der Ausreißererkennung [24]. Bekannte Ansätze hierfür sind die Ausreißererkennung durch einen latenten Unterraum [44] oder Active Adversarial Learning (AAL) [30]. Trotz der verschiedenen Möglichkeiten, GANs mit der Außreißererkennung zu verbinden, zeigen alle bisher bekannten Modelle diesselbe Schwäche: Sie berücksichtigen nicht, dass einige Ausreißer nur in Unterräumen der Attribute und nicht im gesamtem Raum sichtbar sind [25]. Diese versteckten Ausreißer werden von aktuellen GAN-Modellen nach dem neuesten Stand der Technik nicht entdeckt. Als Reaktion auf diese Herausforderung präsentieren wir eine neue Erweiterung des AAL-Frameworks, die GANs mit einem Ensemble aus Attribut-Unterräumen verbindet. Wir stellen das Feature Ensemble GAN (FeGAN) vor, ein GAN, das speziell auf die Unterraumsuche mit mehreren Diskriminatoren spezialisiert ist. Jeder dieser Diskiminatoren lernt auf seinem eigenen Attribut-Unterraum. Dieser innovative Ansatz entdeckt Ausreißer, die möglicherweise ausschließlich innerhalb bestimmter Attribut-Unterräume existieren, ein Aspekt, der sonst von herkömmlichen Methoden oft übersehen wird.

Unsere Experimente zeigen die vielversprechenden Ergebnisse und das große Potenzial von FeGAN für die Ausreißererkennung. FeGAN übertrifft durchweg Basismodelle und unterstreicht die Bedeutung der Identifizierung von Unterraum-Ausreißern, die mit aktuellen Methoden nach dem neuesten Stand der Technik häufig übersehen werden.

Contents

List of Figures

List of Tables

1. Introduction

In recent years, deep generative methods have enabled several subfields of Machine Learning to make great progress. Previously overshadowed by supervised methods, these unsupervised methods now thrive as they yield impressive performance without the need of labels. Particularly, the introduction of Generative Adversarial Networks (GAN) [17] and their game-theoretical approach to Deep Learning proved how powerful generative methods are. GANs manage to generate realistic data, thus they are suited for many different Machine Learning tasks such as Novelty Detection or Image Generation. Especially for high-dimensional data, GANs impress with their performance [24]. Over the years, several extensions to the GAN framework have been published [30, 44], leveraging the generative nature of the model to detect outliers. While there are models like MO-GAAL [30], which embed an Ensemble structure to GANs to generate outliers, or AnoGAN [44], which works with a latent representation of data to detect outliers, all of these GAN extensions still lack one property: Subspace Search. In high-dimensional spaces, outliers are sometimes only, or better, visible in some Feature Subspaces, making it hard to find them when only looking at the full set of Features [25].

That is why we want to combine the generative strength of GANs with a Feature Subspace Ensemble to tackle this missed opportunity. In this thesis, we develop and implement a Feature Ensemble GAN (FeGAN) for Outlier Detection on Feature Subspaces. Not only does this novel approach require a new target function, but it also faces challenges such as handling dependencies between features and the redesign of the vanilla zero-sum-game GANs play. Utilizing multiple Discriminators, FeGAN concentrates on independent Feature Subspaces to classify hidden outliers.

After extending the GAN framework with an Ensemble structure of Discriminators, each fitted on individual Feature Subspaces, we adjust the vanilla target function to incorporate one Generator and multiple Discriminators. Following Novelty Detection experiments against several shallow and deep baselines then prove our initial instinct and show the great potential of FeGAN for unsupervised Novelty Detection.

We start by introducing necessary fundamentals for the research of this thesis. After that, we present related work and go over their strengths and weaknesses. Next, we present our novel method FeGAN and give details about the extension of the GAN framework, followed by an explanation of carried out experiments. Finally, we evaluate all experiments thoroughly and conclude our work.

2. Fundamentals

In this chapter, we will introduce fundamentals of Neural Networks, Autoencoders, GANs, Ensemble methods and Novelty Detection. First, we introduce Neural Networks and their basic functionality. After that, we present two different Machine Learning models, Autoencoders and GANs. Next, we introduce the field of Novelty Detection. Finally, we present Ensemble methods and Feature Bagging.

2.1. Neural Networks

A Neural Network (NN) is a computational model consisting of many interconnected nodes, so called neurons. These neurons are organized in one of three categories of layers: Input layer, hidden layer or output layer. The input layer receives user-given input (data), hidden layers process the data, and the output layer then produces a result. While a NN only has one input and one output layer, it can have multiple consecutive hidden layers. There are many types of hidden layers, each fulfilling a distinct function. Each layer has a fixed but arbitrary size, which is defined by the amount of neurons in that layer. Essentially, each layer represents a specific function. Let I be the input layer, h_i one of $n \in \mathbb{N}$ hidden layers and O the output layer. The resulting Neural Network consisting of these layers, here showed as functions, has the following structure:

$$F = (O \circ h_n \circ \cdots \circ h_1 \circ I)$$

At this point, it is very important to mention that F is differentiable almost everywhere. Each neuron in the NN has a weight and each layer has an additional bias vector. These weights and bias vectors are crucial to the functionality of the Neural Network and define how the NN works. A layer itself represents a linear function. Now, linear functions by themselves are oftentimes not complex enough for many problems. Thus, a layer has the possibility to use an activation function, which introduces non-linearity to the NN. That way the Neural Network can capture more complex relationships. Without the activation functions, the whole NN could be represented by a single linear function.

As Neural Networks are a branch of Machine Learning, different learning paradigms of Machine Leaning also apply:

- Supervised Learning: In this scenario, the given data set contains labels for all data samples. All labels are available to the model.

- Unsupervised Learning: In this scenario, there are no labels at all for the given data set. Hence, the model cannot leverage additional information provided by labels.

- Semi-supervised Learning: In this scenario, there are a few data samples with available labels. The model can then use these few given labels to propagate them onto other, yet unlabeled, data samples.

Obtaining samples can be very costly, require human expertise, or can take too much time until they are available. In these cases, one would choose either unsupervised or semi-supervised learning. However, if the labels are available, they can bring valuable information to the learning process of a model [2].

2.1.1. Updating the weights and bias vectors

The learning process of a Neural Network boils down to updating the weights and bias vectors until seemingly optimal ones are found. To achieve this, a target function F, that will be optimized, is required. The learning process consists of two steps: First, a forward pass of an input sample through the Neural Network and all of its layers is executed. After that, the target function is applied. Since our goal is to minimize the target function with regards to the parameters (weights and bias vectors), we now use gradient descent [40] with learning rate γ to do so.

$$\omega_{t+1} = \omega_t - \gamma \nabla F(\omega_t)$$

During this process, back-propagation [42] allows us to update our parameters. Given the target function, the target value of a forward pass is propagated backwards through the layers. Then, while traversing the layers, the derivative of each layer is calculated gradually and connected via the chain rule (of calculus). This has to be done for every single parameter in the Neural Network until they converge or the change between iterations falls below a threshold. After that, ideally, the best parameters are found. However, this is not always the case. Since gradient descent aims to minimize the target function, it is possible to get stuck in local minima during the process. In that case, either the learning rate has to be adjusted or other mechanisms like momentum [40] have to be implemented.

2.2. Autoencoders

Autoencoders (AE) [43] consist of two Neural Networks: An Encoder E and an Decoder D. The goal of an AE is to reconstruct data. Given a data sample x, which E takes as input, E maps x to a latent space representation of smaller dimension. Then D receives this latent representation as input and maps is back to the original Feature Space of x. The AE aims to minimize the reconstruction error of this process.

2.3. Generative Adversarial Networks

After their introduction, Generative Adversarial Networks [17] quickly attracted science's interest due to their unsupervised advancements. GANs manage to implicitly capture the underlying data distribution of a given data set by optimizing an adversarial zero-sum game. One can imagine the GAN training procedure as a feud between an art forger and an art expert. While the art expert's job is to be really good at spotting fake art, the art

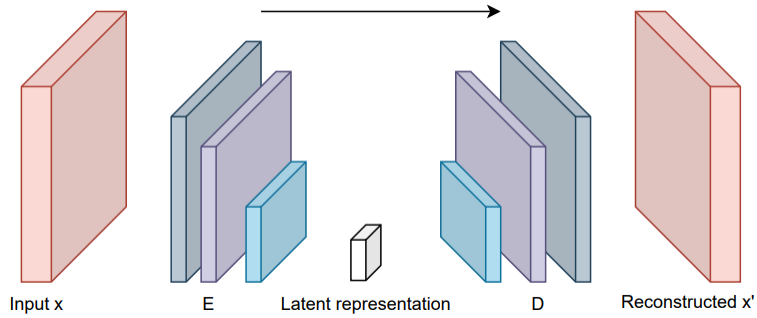


Figure 2.1.: Example architecture of an Autoencoder, traversed from left to right.

forger tries to trick the expert. During the game, the art forger becomes better and better at faking artwork, while, at some point, the art expert might not be able to distinguish between real and fake artwork anymore.

A GAN consists of two adversarial parts: a Generator and a Discriminator. The Generator G is a differentiable function in the form of a Neural Network with parameters θ_g. Given a data set $\mathcal{X}$ and a noise variable z, representing randomly drawn latent input, the Generator represents a function $G : z \mapsto p_g$, where p_g is the Generator's output distribution. We define the Generator as $G(z; \Theta_g)$. The Generator tries to mimic the underlying distribution of $\mathcal{X}$, here called p_{data}, and thus trains to get p_g to converge to p_{data} [17]. On the other side of the game, the Discriminator D is a differentiable function in the form of a Neural Network with parameters Θ_d aswell. We define this as $D(x; \Theta_d)$. In this case, $D(x)$ calculates the probability that x comes from p_{data} rather than p_g. D aims to maximize the probability of labeling all given samples, real and generated, correctly. Meanwhile, G aims to minimize this probability. In other words, G tries to maximize the probability of D labeling generated samples as real ones.

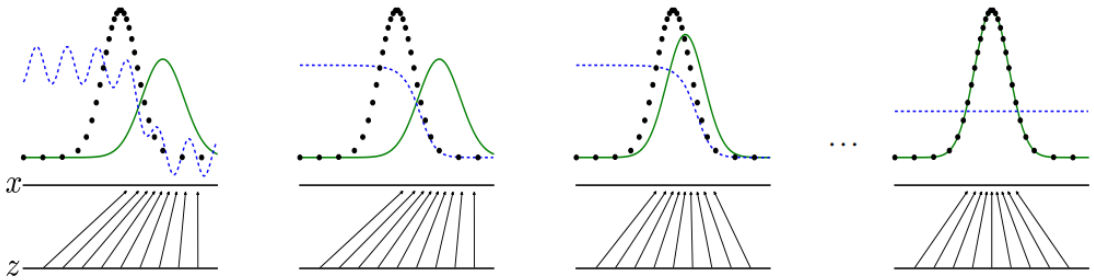


Figure 2.2.: A Figure from [17] showing the convergence of p_g to p_{data} from left (early in training) to right (advanced training).

In Figure 2.2, the black dotted line represents p_{data}, the green line represents p_g and the blue dashed line is the decision of the Discriminator. You can see, once the Generator learned to generate realistic samples well, the Discriminator cannot distinguish between real and generated samples anymore and thus is not better than random guessing. Realistically, this scenario is only achieved under optimal theoretical circumstances and not in practice.

2.3.1. Training

We train GANs by playing the following zero-sum game between the Generator G and the Discriminator D

$$\min_G \max_D \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))], \tag{2.1}$$

where p_z is the latent input distribution. To reach its goal, it is sufficient for G to minimize $\log(1 - D(G(z)))$. In early stages of training, G does not know how to generate realistic samples yet. This leads to the Discriminator rejecting the generated samples with a high confidence. In some cases, this results in an insufficient gradient for G, hindering G from learning well [17]. To solve this, it is possible for G to maximize $\log(D(G(z)))$ instead as this provides stronger gradients in early training [17]. The target function 2.1 can be equivalently reformulated as the binary cross entropy $y \log(p) + (1 - y) \log(1 - p)$, where, for a given sample x, $p = D(x)$ and

$$y = \begin{cases} 1 & \text{if x is real} \\ 0 & \text{if x is generated} \end{cases}$$

G and D play this game until D reaches Nash equilibrium or another stopping criterion. In the case of the Nash equilibrium, the Generator has become so good at generating real-looking samples, that $D(x)$ is no better than random guessing the origin of the sample [17].

2.4. Novelty Detection

Given a data set $\mathcal{X}$, Novelty Detection, also called One-Class classification by some authors, describes the process of identifying anomalous data samples (called outliers in this thesis) by learning first the non-anomalous distribution. Intuitively, we call a data sample x an outlier, if it does not fit with the other data samples of $\mathcal{X}$ and is too different from the rest of the data set. The challenge of this classification task is to decide when a sample is flagged as such.

Let us give an example. We assume that a data set $\mathcal{X} \in \mathbb{R}$ follows an underlying data distribution p_{fst}. Now, we observe x' from another data set $\mathcal{X}' \in \mathbb{R}$ following another distribution p_{snd}. Assuming these two distributions are different enough from each other, there is a high chance that

$$\mathbb{P}_{p_{fst}}(x') < \varepsilon, \quad \varepsilon << 1.$$

This implies that x' would be a statistical anomaly in $\mathcal{X}$. Thus, x' should be classified as an outlier if we classify on $\mathcal{X}$.

Outlier Detection. Contrary to the general understanding of Outlier Detection, Novelty Detection focuses on classifying new, unseen, data samples. Additionally, there is no general consensus in the field of Outlier Detection, whether Novelty Detection is unsupervised or semi-supervised [37]. Under the assumption that outliers are rare in data sets, it does

not make a difference and thus, Novelty Detection can be called unsupervised and one can sample the training set without filtering outliers. However, as this only holds under the assumption of rarity, one can also argue that Novelty Detection has to be semi-supervised. Though this might seem obvious as labels for the positive class are needed, it is not. Even in literature, there are contradicting statements for Novelty Detection methods like Deep SVDD [41]. In their publication, the original authors explicitly state that the method is unsupervised, while other authors call it semi-supervised [10] as labels for the normal class are required.

In this thesis, we abide to the assumption of rarity of outliers and call Novelty Detection unsupervised.

2.5. Ensemble methods

Ensemble methods describe a unique learning method combing the strengths of different underlying models [14]. An Ensemble typically consists of multiple classifiers working on the same data. The Ensemble takes into consideration each classifier's decision and acts according to the actual implementation of it. Simple Ensembles oftentimes use the weighted average of all classifiers' decisions [14]. Let E be the Ensemble consisting of $N \in \mathbb{N}$ classifiers C_i and $o \in D$ the object to be classified. For a weighted average, the Ensemble's decision is

$$E(o) = \sum_{i=1}^{N} w_i C_i(o),$$

where w_i is the weight of C_i and $\sum_{i=1}^{N} w_i = 1$. This approach allows to obtain high-accuracy classifications by combining multiple classifiers. [14] shows that such an Ensemble's classification accuracy is at least as high as a single classifier's.

An important Ensemble method to mention here explicitly is Bootstrap Aggregation (Bagging) [6]. In a Bagging Ensemble, each Classifier C_i trains on a different subset S_i of the original data. Each subset is drawn randomly with replacement.

2.5.1. Feature Bagging

Feature Bagging [21] is an Ensemble learning method trying to reduce restrictions arising in high-dimensional data sets oftentimes described as the curse of dimensionality [26]. In a Feature Bagging Ensemble (FBE), each classifier trains and predicts only in a subset of features. Again, each subset is drawn randomly and with replacement. We use the following algorithm to draw $K \in \mathbb{N}$ Feature Subspaces.

Outliers tend to hide in high-dimensional feature spaces [25]. These so called subspace-outliers are only visible in Feature Subspaces [25, 57]. As distances between objects in high-dimensional spaces grow more alike, there is a high probability of not finding outliers when applying Outlier Detection algorithms to these high-dimensional spaces [58]. Therefore a Feature Bagging Ensemble allowing for an efficient Subspace Search can be very powerful for high-dimensional classification tasks such as Novelty Detection [22, 8, 1]. By looking at subspaces of a smaller dimension, in which outliers are better visible

Algorithm 1 FeatureSubspaceSelection

Input: Dimension of the data set D; Number of classifiers K
Output: A Feature Subspace for each classifier

```
# Randomly draw K integers in the range of 1..D
dims ← RandomDraw(D,K)
subspaces ← Empty list
for i ∈ {1, ...K} do
    # Randomly draw subspace with dims[i] features
    space ← DrawSubspace(dims[i])
    # Prevent using the same subspace for different classifiers
    while space ∈ subspaces do
        space ← DrawSubspace(dims[i])
    Add space to subspaces
return subspaces
```

and the probability of finding them is higher, the accuracy of the prediction can be greatly increased [1].

3. Related Work

In this chapter we provide an overview of related work and methods, their strengths and weaknesses, starting with non-generative Outlier Detection methods. We then introduce Generative Adversarial Network (GAN) [17] methods for Outlier Detection as well as One-Class classification [32]. Finally, we propose FeGAN, the model developed in this Bachelor's thesis.

Let $\mathcal{X} \subset \mathbb{R}^d$ be a data set consisting of $N \in \mathbb{N}$ data samples.

3.1. Non-generative Novelty Detection

In this section we introduce the non-generative methods used to build a part of the baseline FeGAN is compared against. Namely, we present shallow methods followed by Kernel and deep Kernel methods.

3.1.1. Shallow methods

We define shallow methods as methods, that do not use Neural Networks. While there are Machine Learning models such as Support Vectors Machines, we call shallow, there are also other, very different, algorithms we call shallow. Especially for Classification and Outlier Detection tasks, there are powerful shallow algorithms. Most shallow Outlier Detection algorithms fall under one of three categories: Distance-Based, density-based, or angle-based. [9] evaluates shallow methods. In their study, they compare these methods by their best possible performance (with tuned hyperparameters). Specifically, they compare 12 different methods, versions of LOF [7], KNN [18] and ABOD [27] amongst others. For further information, please visit [9]. For our work, we focus on the overall best-performing methods of this study: LOF and KNN.

Shallow methods like KNN and LOF can be very powerful, especially on lower dimensional data sets. On high-dimensional data sets though, these two methods do not always manage to fully capture the underlying data distribution well. Especially properties related to the curse of dimensionality [26], such as the sparsity of spaces [52] or NN-Instability [3], are a tough challenge for distance and/or density-based shallow methods.

3.1.2. Kernel methods and One-Class Support Vector Machines

Kernel methods [23] are pattern analysis algorithms using linear functions to solve non-linear problems. A Kernel method relies on a Kernel function, which projects data samples onto a higher-dimensional space. In that higher-dimensional space, the Kernel method

then looks for a hyperplane to classify samples. One of the most popular Kernel methods is the Support Vector Machine (SVM) [20].

SVM. SVMs are supervised models used for classification and regression. To fulfil its task, the SVM splits the training data into two categories, exclusively and projects the samples into space such that the gap between the samples of different categories is as big as possible. This procedure is not always a Kernel method as the SVM can solve linear problems without the use of a Kernel. However, if needed, the SVM uses the Kernel trick to solve non-linear problems [45].

Additionally, with models such as OC-SVM [46] and SVDD [49], Kernel methods have also been introduced to the field of One-Class classification. Recently, there has been an expansion into deep learning, combining principles of Kernel methods with the variety of deep learning methods. A popular example of this is Deep SVDD [41].

SVDD. Similar to OC-SVM, Support Vector Data Descriptions (SVDD) [49] also project samples to a higher-dimensional space. However, SVDD now fits a hypersphere with center a with minimal volume containing all samples. To do so, they solve

$$\min R^2 + C \sum_i \xi_i \quad \text{with constraint} \quad ||x_i - a||^2 \leq R^2 + \xi_i,$$

where R is the radius, $\xi_i \geq 0$ is a penalizing parameter and C controls the trade-off between the volume and the errors.

Then, samples projected too far outside of the hypersphere are considered outliers. If the Kernel function K used for OC-SVM and SVDD has the property that $K(x, x) = 1$ $\forall x \in \mathcal{X}$, then the methods are equivalent.

3.2. Generative Novelty detection

Generative methods vary and come in deep and shallow forms. Depending on the actual underlying method, they have many different use cases and thus it is very important to choose the right one. Examples for shallow generative methods are the naive bayes classifier or the plain Kernel density estimation. All of these methods are well known, and work well if used correctly. However, shallow methods face difficulties in high-dimensional data sets, such as choosing relevant attributes or growing distances, which reduce their overall quality and ability to predict precisely [58]. That is why, especially for high-dimensional data sets, it is useful to opt for deep methods.

For the deep generative methods, we look at current state-of-the-art GAN Outlier Detection and One-Class classification methods. Specifically, we introduce BiGAN [15], AnoGAN [44] and MO-GAAL [30].

Due to their self-supervised generative nature, GANs are amongst the most popular deep learning methods for Outlier Detection [51, 13]. While there are different approaches to this problem, two stand out: Vanilla GAN methods and GAN methods using autoencoder-like extensions. Most vanilla GAN methods for Outlier Detection focus on the full Feature Space to identify outliers. Contrary to that, Autoencoder-like GAN models rely on a

smaller embedded subspace (latent space) rather than the full space to detect outliers. In this work, we extend this approach even further. We detect outliers by learning multiple Feature Subspaces instead of only one embedded subspace.

3.2.1. BiGAN

In some cases, it can be very useful to work with a semantic latent representation of data instead of the data itself. A GAN normally uses some kind of (semantic) latent space to generate data. However, prior to the introduction of BiGAN [15], there was no way for a GAN to learn the inverse mapping of the Generator. BiGAN has been proposed to tackle this issue.

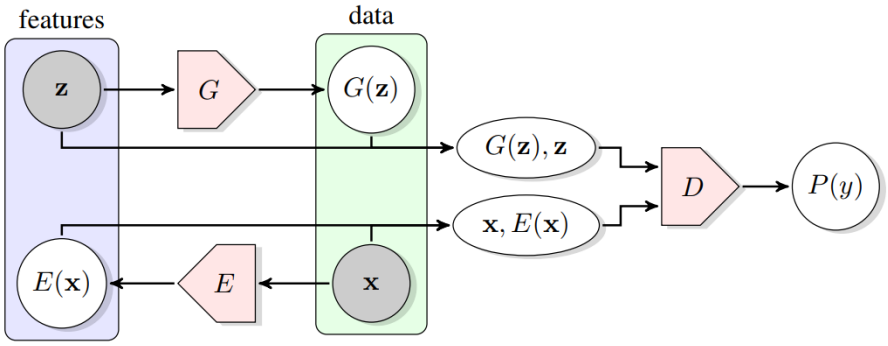


Figure 3.1.: BiGAN's architecture; taken from [15]

BiGAN is a GAN framework extending the vanilla GAN architecture by an Encoder E. While the Generator G learns a mapping from a latent input space $\mathcal{Z}$ to the sample space $\mathcal{S}$, E learns a mapping of normal samples $x \in \mathcal{S}$ back to the latent space $\mathcal{Z}$. At this point, it is important to mention, that there is no direct (or indirect) communication between G and E. The Encoder never sees generated samples $G(z)$. To achieve this goal, two adjustments to the GAN training process are made: While training the GAN, the Encoder is trained simultaneously. On top of that, the Discriminator D of the GAN is modified aswell. D now also takes input from the latent space and uses this to decide, whether a sample has been generated or not. Overall, the training objective is now

$$\min_{G,E} \max_{D} \mathbb{E}_{x \sim p_{data}(x)} [\underbrace{\mathbb{E}_{z \sim p_E(\cdot|x)} [\log(D(x,z))]}_{\log(D(x,E(x)))}] + \mathbb{E}_{z \sim p_Z(z)} [\underbrace{\mathbb{E}_{x \sim p_G(\cdot|z)} [\log(1 - D(x,z))]}_{\log(1-D(G(z),z))}],$$

where p_X is the distribution of $\mathcal{S}$ and p_Z is the distribution of $\mathcal{Z}$. Under optimal training circumstances, meaning that G and E are trained optimally, G and E will be approximate inverses [15].

BiGAN manages to learn the inverse mapping of the Generator with a seemingly simple extension of the original GAN framework. This allows to work with the sample space $\mathcal{S}$ as well as the latent representation from $\mathcal{Z}$. Even though, with a few adjustments, it is possible to use BiGAN as an anomaly detection method [54], BiGAN's actual goal is to learn the (approximate) inverse mapping of the Generator and not Outlier Detection explicitly.

3.2.2. AnoGAN

Closely related to BiGAN, with their AnoGAN framework, [44] proposed one of the first Outlier Detection methods for GANs in 2017. AnoGAN uses a vanilla GAN, trained on positive (normal) samples, to learn a mapping from the latent input space $\mathcal{Z}$ of the Generator G to the realistic sample space $\mathcal{S}$. During this step, the Generator G and the Discriminator D are trained using the standard GAN minimax game:

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

where p_{data} is the original distribution of $\mathcal{S}$ and p_Z is the distribution of $\mathcal{Z}$. Once the adversarial training of the GAN is finished (for example when D reaches a Nash equilibrium), the Generator learned a mapping $G : z \mapsto x$ from $\mathcal{Z}$ to $\mathcal{S}$. Now AnoGAN initiates the second step to learn the inverse of the Generator mapping, leveraging that the latent space has smooth transitions [38], meaning that two similar inputs of the latent space will produce similar outputs of the sample space. In other words, if we treat G as a function, this means that G is continuous and that there are no unexpectedly big leaps in output if the distance between two inputs is small. If there would not be smooth transitions, learning the inverse mapping would become more computationally expensive as the learning process would rely more on non-deterministic behaviour.

The second step is an iterative process aiming to find the corresponding $z \in \mathcal{Z}$ to a given sample $x \in \mathcal{S}$ such that $G(z)$ is visually most similar to x. Visual similarity is measured by the residual loss $\mathcal{L}_R$. Given x, AnoGAN randomly samples a $z_1 \sim \mathcal{Z}$ and generates the output $G(z_1)$. Now a new loss function is applied to receive an updated sample z_2. This process is repeated iteratively via $\gamma = 1, 2, ..., \Gamma$ backpropogation steps until the best matching z_Γ for x is found. The new loss function mainly consists of two parts, the residual loss and the discrimination loss. While the residual loss $\mathcal{L}_R(z_\gamma)$ takes into account the visual similarity between x and $G(z_\gamma)$, the discrimination loss $\mathcal{L}_D(z_\gamma)$ enforces $G(z_\gamma)$ to lie on the sample space $\mathcal{S}$.

$$\mathcal{L}_R(z_\gamma) = \sum |x - G(z_\gamma)| \quad \text{and} \quad \mathcal{L}_D(z_\gamma) = \sum |f(x) - f(G(z_\gamma))|$$

Because of the unsupervised training, an intermediate layer $f(\cdot)$ of the Discriminator is used to specify the statistics of a sample [44]. The weighted sum of both losses is used as the overall loss function for the learning process of the inverse.

$$\mathcal{L}(z_\gamma) = (1 - \lambda) \cdot \mathcal{L}_R(z_\gamma) + \lambda \cdot \mathcal{L}_D(z_\gamma)$$

The second step has to be executed for every single input sample x. After learning the inverse, one can derive an anomaly scoring system directly from $\mathcal{L}$. This scoring system does not have an upper bound, but a higher value represents a higher likelihood of the sample being anomalous.

$$\mathcal{A}(x) = (1 - \lambda) \cdot \mathcal{R}(x) + \lambda \cdot \mathcal{D}(x)$$

$\mathcal{R}(x)$ and $\mathcal{D}(x)$ stand for the values of $\mathcal{L}_R(z_\gamma)$ and $\mathcal{L}_D(z_\gamma)$ during the last iteration step, respectively.

AnoGAN's approach of taking the reconstruction error into consideration when classifying a data sample translates to an Outlier Detection by a latent space rather than all features. Thus, the size of the latent space is crucial to AnoGAN's ability of classifying. Additionally, using a only single latent space to detect anomalies directly implies that there is a high possibility of not all dependencies between features being fully represented in high-dimensional spaces.

3.2.3. Multiple-Objective Generative Adversarial Active Learning (MO-GAAL)

[30] were the first to combine the strengths of the GAN architecture with Ensemble methods for Outlier Detection. They first introduce SO-GAAL [30]. SO-GAAL builds on the GAAL framework [56] to counter the lack of information caused by the curse of dimensionality. Intuitively, GAAL proposes to generate samples, which the Discriminator is most uncertain about. SO-GAAL uses this approach to generate potential outliers with the need of uncertainty sampling [47]. The model's Discriminator D aims to distinguish between normal and anomalous data by having the Generator G generate potential informative outliers. However, due to its "One Generator - One Discriminator" architecture, SO-GAAL cannot overcome the mode-collapse [17] problem, which vastly reduces the performance of a GAN. Hence, they extend SO-GAAL to create MO-GAAL.

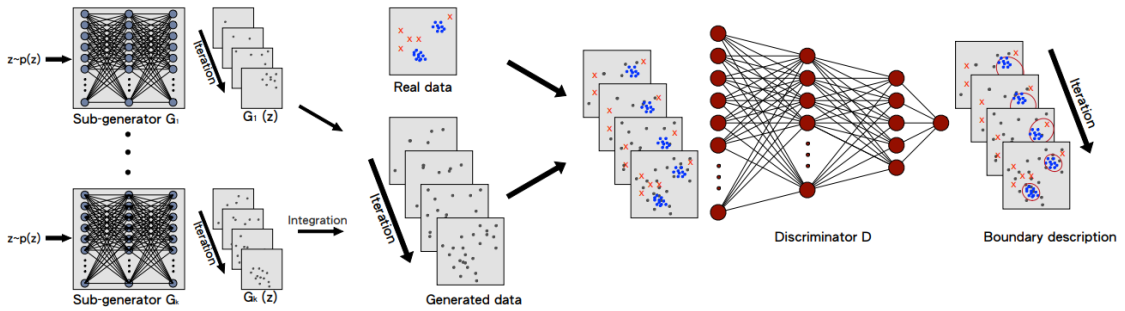


Figure 3.2.: MO-GAAL's architecture; taken from [56]

MO-GAAL consists of k Generators $G_{1:k}$ and one Discriminator D. The Generators each follow their own objective to each learn the generation mechanism of a subset X_i of real data samples, individually. It is important to mention that all $x \in X_i$ are similar to each other and $\bigcup_{i=1}^{k} X_i = \mathcal{X}$ in order to provide a reasonable reference distribution for the Generators. $\mathcal{X}$ is split by evaluating $D(x)$ and grouping samples with similar values due to the inverse of space smooth transition [38]. To properly train the Generators, the target value of $D(G_i(z))$ is changed from 1 to T_i, which is a representative statistic of the samples in X_i. Requiring the samples in X_i to be similar, the representative statistic is needed and cannot be left out of training. Without it, the samples in X_i are chosen randomly, preventing the Generators from learning properly. An example for such a representative statistic is the minimum output value of $D(X_i)$. This adaptation leads to the following optimization problem:

$$\max_{D} \frac{1}{2n}[\sum_{j=1}^{n} \log(D(x^{(j)})) + \sum_{i=1}^{k} \sum_{j=1}^{n_i} \log(1 - D(G_i(z_i^{(j)})))]$$

$$\max_{G_i} -\frac{1}{n}[\sum_{j=1}^{n} T_i \log(D(G_i(z_i^{(j)}))) + (1 - T_i) \log(1 - D(G_i(z_i^{(j)})))],$$

where n is the number of samples and n_i is the number of potential outliers generated by G_i. This model setup is able to handle the mode-collapse problem well while still generating outliers to reach the goal of distinguishing outliers and normal data samples. However, outliers are only detected on the full Feature Space. The mechanism of generating realistic outliers becomes harder with a growing number of features, restricting the capability of detecting those.

These state-of-the-art models and the advantages of Feature Ensembles for Outlier Detection motivate us to develop a new Outlier Detection method combining both. With FeGAN we want to utilize not only the generative properties of GANs but also leverage a Feature Ensemble to create, to our knowledge the first, GAN model to detect anomalies via Subspace Search. The use of a FBE allows us to look for subspace-outliers [25, 57], which improves the quality of Outlier Detection in high-dimensional spaces, as this is where subspace-outliers are common [25].

4. FeGAN

In this chapter, we formally introduce the model of this thesis, the Feature Ensemble GAN (FeGAN). First, we present the motivation and idea behind the model. After that, we explain the novel target function of FeGAN. Then, we explain how we incorporate an Ensemble method into the GAN framework. Finally, we present the Neural Network architectures we use for the Generator and Discriminators and go over details of the model and training process.

4.1. Introduction

Current state-of-the-art GAN frameworks [30, 44] show that generative methods have great potential as Outlier Detection methods. However, they lack one property. They do not take into account that there are subspace outliers as they do not detect outliers on different Feature Subspaces. Especially in high-dimensional spaces, this impacts prediction quality negatively. The multi-view property of subspaces [33, 53] is required to detect subspace outliers. Several authors show that implementing this property via efficient subspace search improves prediction quality [25, 57].

FeGAN is a Generative Adversarial Active Learning method [30] for unsupervised Subspace Outlier Detection. With FeGAN, we combine the strengths of generative methods and Feature Subspace search to tackle this problem. Extending the GAN framework [17], we add multiple Discriminators to the model to allow efficient Subspace Search. Since we extend the framework, we have to adjust the original target function 2.1 to take into account multiple Discriminators. Multiple Discriminators on their own are not capable of Subspace Search without the assignment of unique Feature Subspaces for each Discriminator. FeGAN thus implements Feature Bagging without repetition to solve this. In the following sections, we explain in detail how FeGAN extends the GAN framework.

4.2. Ensemble method

Figure 4.1 shows the architecture for FeGAN's Ensemble structure. It consists of one Generator G with k Discriminators D_i ($i \in \{1, ..., k\}$). Each Discriminator D_i is assigned their own Feature Subspace S_i. We select these Subspaces via Feature Bagging without repetition [21] using the FeatureSubspaceSelection algorithm (see Chapter 2). The Subspaces are of different (random) sizes and and not disjoint. While the Discriminators train independently, they share a joint impact on the Generator. That way, the Generator implicitly learns the chosen Subspaces.

During training, the Generator generates potential informative outliers on the full set of features. All Discriminators have access to the same data to train on, but project them,

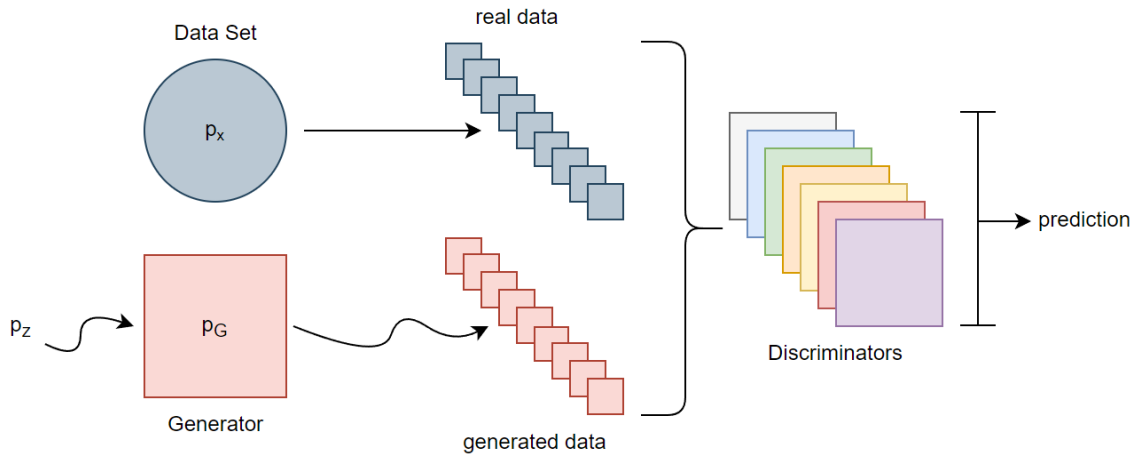


Figure 4.1.: FeGAN's architecture, where p_x is the distribution of the original data, p_G is the distribution of the Generator and p_z is the distribution of the latent space.

no matter if real or generated, onto their own Subspace S_i and then predict. With this setup, we look at subspaces rather than the full Feature Space to find outliers hiding in said subspaces [25]. Different subspace drawing algorithms, such as HiCS [25], can be implemented easily.

4.3. Target Function

Since the vanilla GAN zero-sum-game is only suited for one Generator and one Discriminator, we have to adjust it. Adding more Discriminators changes the target of the Generator. For each individual Discriminator D_i, the target function does not change from the original target:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{4.1}$$

Each Discriminator thus plays a zero-sum-game against the Generator. This approach of each Discriminator following their own objective, independent from other Discriminators, stays close to the Multi-Objective approach of [30] where multiple Generators each play their own game against one Discriminator. However, it is not possible for our Generator to follow multiple targets and play multiple zero-sum-games simultaneously. To solve this issue, G does not play against each Discriminator individually but rather against their mean. Therefore, we formulate the Ensemble adversary of the Generator as:

$$\hat{D} = \frac{1}{k}\sum_{i=1}^{k} D_i$$

Thus, the Generator's target function is

$$\min_G \max_{\hat{D}} \mathbb{E}_{x\sim p_{data}(x)}[\log \hat{D}(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - \hat{D}(G(z)))] \tag{4.2}$$

This target function 4.2 is only valid if it is differentiable. We assume, by definition, D_i ($\forall i \in 1..k$) is differentiable almost everywhere. Since the sum of differentiable functions is differential and the multiplication with a factor does not affect differentiability, $\hat{D}$ is differentiable aswell. Therefore, we conclude that target function 4.2 is differentiable.

Similar to [17], we also derive a training target for the Generator from 4.2:

$$\min \mathbb{E}_{z \sim p_z(z)}[\log(1 - \hat{D}(G(z)))]$$

However, as explained in section 2.3.1, this target can result in insufficient gradient for G to learn properly in early stages of training. Analogous to [17], this is solved by using

$$\begin{aligned} \max \mathbb{E}_{z \sim p_z(z)}[\log(\hat{D}(G(z)))] &= \max \mathbb{E}_{z \sim p_z(z)}[\log(\frac{1}{k}\sum_{i=1}^{k} D_i(z))] \\ &= \max \mathbb{E}_{z \sim p_z(z)}[\log(\frac{1}{k}) + \log(\sum_{i=1}^{k} D_i(z))] \end{aligned} \tag{4.3}$$

Because $\log(\frac{1}{k})$ is constant, it is sufficient for G to solve $\max \mathbb{E}_{z \sim p_z(z)}[\log(\sum_{i=1}^{k} D_i(z))]$.

Training algorithm. Our Ensemble structure in combination with the new target function leads to the following training algorithm for FeGAN:

Algorithm 2 FeGAN training

Input: Data set X, Number of Discriminators K
Initialize Generator G
 # D is the dimension of X
subspaces ← FeatureSubspaceSelection(D,K)
Initialize Discriminators $D_{1:K}$ *with unique subspaces*
for *epoch* ∈ {1, ..., *epochs*} **do**
  for *batch* ∈ {1, ..., *batches*} **do**
    noise ← Random noise $z^{(1)}, ..., z^{(m)}$ from latent space
    data ← Draw current batch $x^{(1)}, ..., x^{(m)}$
    for $j \in \{1...k\}$ **do**
      # Update Discriminator D_j *by ascending the stochastic gradient*
      $\nabla_{\Theta_{D_j}} \frac{1}{m} \sum_{i=1}^{m} [\log(D_j(x^{(i)})) + \log(1 - D_j(G(z^{(i)})))]$
    # Update generator by descending the stochastic gradient
    $\nabla_{\Theta_G} \frac{1}{m} \sum_{i=1}^{m} \log(1 - \frac{1}{k} \sum_{j=1}^{k} D_j(G(z^{(i)})))$

4.3.1. Architecture

The architecture of the Neural Network is crucial to the model's performance. Choosing a good architecture is difficult and choosing the best fitting architecture, assuming there is one, might even be impossible. This is due to the excessive run time required to train a network. In the case of FeGAN, we not only face one, but $k + 1$ consecutively trained

networks, increasing the computation time by a factor of $\approx k$. With this constraint, one cannot randomly try different architectures without losing a significant amount of time. Thus, a lot a domain knowledge is required to select or find an optimal network architecture [50, 12]. Without domain knowledge, the tuning of an architecture requires at least some kind of knowledge about the data sets the model will work on [29]. Additionally, the lack of labels in unsupervised and semi-supervised learning prohibits the use of a validation set (otherwise, there would be enough labels for much preferred supervised learning [2]). Without a validation set, tuning a model and the architecture is impossible, as this would require decisions based on the test set, which is very bad practice and leads to overfitting quickly [16]. Since this scenario is the case for FeGAN, we are forced to choose an architecture capable of handling different types of datas sets prior to training.

Our Ensemble structure allows the use of multiple weak learners. While they run the risk of overfitting on Feature Subspaces individually, they still help reduce overfitting of the overall model [14]. Following a similar approach to MO-GAAL [30], we utilize Dense Layers in our architecture. Dense Layers are fully connected hidden layers. This flexibility makes them suitable for a wide range of tasks, which is especially useful without prior knowledge about a data set. However, due to the simplicity of the structure of Dense Layers, we propose two slightly different versions of FeGAN to be able to handle data sets with different sizes and amounts of features.

V1. In the first version, the Generator comprises two Dense Layers of L neurons, each. L describes the dimension of the latent input space for the Generator. In our case, we set L to the number of features. The Discriminator also consists of two Dense Layers, one with S and one with one (1) neuron, where $S = \sqrt{\#samples}$.

V2. For data sets with higher dimensions, we introduce a second version of FeGAN aswell. In V2, we add two identical Dense Layers with L neurons to the Generator and two Dense Layers with S neurons to the Discriminator.

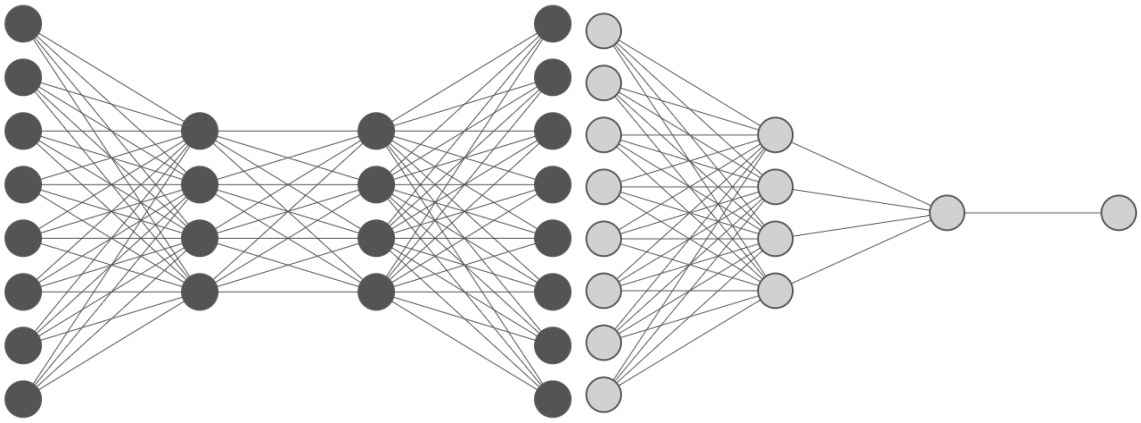
Network details. For every Dense layer except for one, we use the ReLU activation function to introduce non-linearity, as ReLU avoids the "vanishing gradient" problem many deep networks face and is computationally efficient [48]. The last Dense layer of each Discriminator is responsible for the final classification of the Discriminator and returns a probability. For that layer, we use the Sigmoid activation function, as this is common practice [35].

Keep in mind that the architecture for each Discriminator stays the same while the sizes of the layers adjust to each specific Feature Subspace. This short description excludes the input and the output layer.

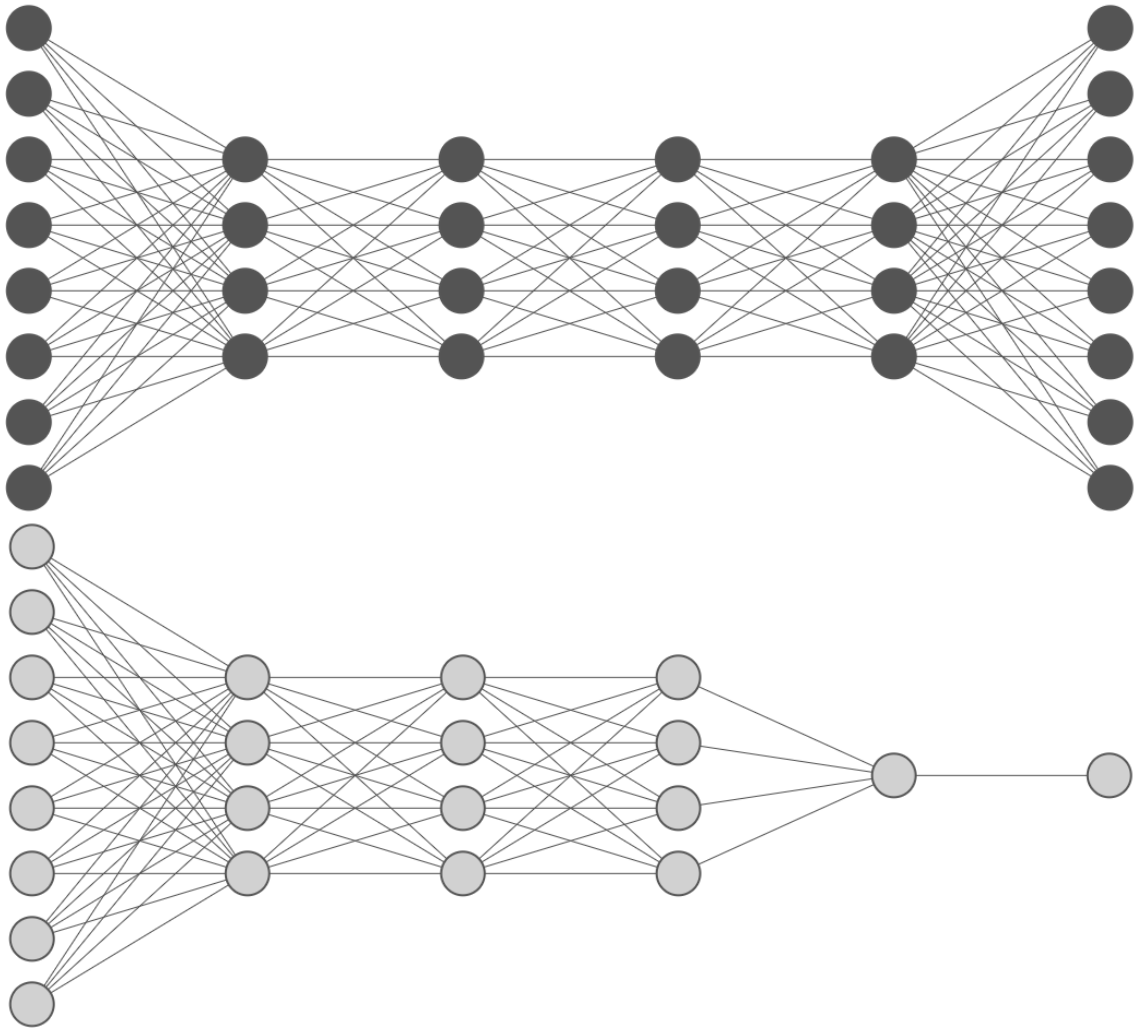
In Figure 4.2, we show the different versions of FeGAN. The left most layer is the input layer, while the last is the output layers. Every layer inbetween is a hidden (Dense) layer. We fixed the dimension of the data set to be 8, the number of samples to 16 for this example.

4.3.2. Training Details

To train FeGAN, we use a batch size of $\min(500, \#samples)$ and update the parameters via stochastic gradient descent [4]. Furthermore, the model trains with a *stop_epochs* parameter. *Stop_epochs* determines how many epochs the Generator trains for. After *stop_epochs* epochs, the Generator stops training and only generates samples for the Discriminators to train on. More specifically, the Discriminators train for a total of $3 * stop\_epochs$ epochs. Though this mechanism can result in the overfitting of individual Discriminators, it prevents an overfitting of the Generator. Since there is only one Generator, an overfitted Generator would negatively impact the performance while overfitted Discriminators do not (as much) due to the Ensemble structure. Also, to avoid getting trapped in local minima, we employ momentum for V2. Lastly, as suggested in Chapter 2.3.1, we implement the target function of FeGAN for the Generator and the Discriminators via binary cross entropy, as this allows an easier implementation with TensorFlow [31].



(a) V1 architectures: Dark: Generator, Light: Discriminator



(b) V2 architectures: Dark: Generator, Light: Discriminator

Figure 4.2.: This figure summarizes the architectures of FeGAN.

5. Experimental Design

To allow reproducibility, we specify and explain the conditions under which we conduct experiments. This includes data sets, state-of-the-art baselines, evaluation metrics, computational resources, hyperparameters, code and used libraries. First, we go over computational resources, and technical details. Then, we present the data sets we use to conduct the experiments. Next, we briefly introduce the SOTA baselines and our choice of hyperparameters for those. After that, we explain the evaluation metrics used for the experiments. Finally, we go over the experiments.

5.1. Experimental settings

5.1.1. Computational Resources and Technical Details

We run the experiments in a computational server running Ubuntu 22.04.2. The server is equipped with a NVIDIA GeForce RTX 3090 GPU and AMD EPYC 7443P 24-Core Processors paired with 132GB of RAM. All calculations were done using CUDA cores of the GPU. The code is written in Python [39] and for the machine learning parts of our model, we use the TensorFlow-Framework [31] version 2.12. For the baseline models, we use the implementations of PyOD [55] version 1.0.9.

5.1.2. Data sets

To conduct the experiments, we use 8 real-life data sets from [9]. In this study, we do not focus on low-dimensional data sets. Therefore, we exclude data sets from [9] with less

	#Instances	#Outliers	#Features
Ionosphere	351	126	32
KDDCup99	60632	246	38
Waveform	3443	100	21
Annthyroid	7200	534	21
Arrhythmia	450	206	259
Cardiotocography	2126	471	21
InternetAds	3264	454	1555
SpamBase	4601	1812	57

Table 5.1.: Comparison of the used data sets

than 20 features and then randomly select 8 of the extant 12 data sets. Furthermore, we use the normalized version without duplicates and categorical attributes of each data set.

5.1.3. Baselines

We use different models to compare FeGAN with. To have a broad representation, we include multiple examples from both, shallow and deep, methods. Our deep baselines consist of one non-generative method, Deep SVDD [41], and two state-of-the-art GAN methods for Outlier Detection, AnoGAN [44] and MO-GAAL [30]. For the shallow methods, we choose LOF, KNN and OC-SVM as they are amongst the most popular shallow methods and [9] show that they are statistically better on a large scale of data sets. Despite of the existence of more powerful deep methods, these three shallow methods remain competitive as an easier parametrization allows for more consistent high performance under average scenarios. When optimized, deep models tend to gain the upper hand [19].

In the following, we briefly introduce each baseline method except for AnoGAN and MO-GAAL, as these two models are part of the related work and are thoroughly explained in Chapter 3.

KNN. k-Nearest Neighbors (KNN) [18] is a distance-based algorithm. For a given distance function, KNN returns each sample $o \in \mathcal{X}$ where there are no more than k objects closer to the current query sample $q \in \mathcal{X}$ than o. The k-NN distance of o is the distance to the object that has the kth. biggest distance to o.

With appropriate adjustments, KNN can be used for various task such as classification or Outlier Detection. To detect outliers, the k-NN distance in combination with a reasonable, data set-specific, threshold can be used.

LOF. The Local Outlier Factor (LOF) [7] is a density-based Outlier Detection method. The algorithm compares the k-NN distance of an object $o \in \mathcal{X}$ to the k-NN distances of its nearest neighbors. The LOF is directly used as the outlier score. Only a high LOF implies o being an outlier. However, it is not trivial to interpret the outlier score as it is unclear how to determine whether the LOF is high.

OC-SVM. In an unsupervised setting, One-Class SVMs [46] project samples onto a higher-dimensional space. In that higher-dimensional space, OC-SVM then tries to separate all samples from their origin using a hyperplane. While doing that, it also tries to maximize the distance between said hyperplane and the origin. New samples projected between the origin and the hyperplane then are called outliers.

Deep SVDD. Deep Support Vector Data Description (DSVDD) [41] combines the objective of SVDD with deep learning. While fitting a hypersphere of minimal volume in the SVDD manner, it jointly trains a Neural Network to map samples into that hypersphere to learn a useful feature representation. That way, DSVDD defines an anomaly score to classify samples. Thus, the model is suited for One-Class classification because positive samples

are mapped close to the center of the hypersphere, while anomalous samples are mapped further away.

5.1.4. Hyperparameters

As done by [19], we use the default parameters of PyOD [55] version 1.0.9. If possible, [55] use hyperparameters recommended by the original authors of each method to provide fair comparison settings. If there are no recommended hyperparameters, PyOD opts for reasonable, field acknowledged, values. For further information regarding the hyperparameters, please refer to [55].

For FeGAN, we choose the following hyperparameters:

- The learning rate of the Generator: $lr_g = 0.001$
- The learning rate of the Subdiscriminators: $lr_d = 0.01$
- The number of Subdiscriminators: $k = 2 * \sqrt{\#dimensions}$
- The number training epochs: $stop\_epochs = 30$ (The total number of epochs is $3 * stop\_epochs$)

5.1.5. Evaluation Metric

AUC. To evaluate the quality of a model, we follow related research [44, 30, 41] and use the Receiver Operating Characteristic Area Under Curve (ROC AUC) [5]. The ROC AUC plots the True Positive Rate (TRP) on the Y-axis against the False Positive Rate (FPR) on the X-axis.

$$TPR = \frac{TP}{TP + FN} \quad FPR = \frac{FP}{FP + TN},$$

where TP are true positive and FN are false negative predictions. Since the ROC is based on rates rather than total numbers, it is insensitive to the number of outliers/imbalanced class members. We then calculate the area under the curve (the integral) of the plot. A ROC AUC near 1 implies that the model predicts very well while a value near 0 implies that the model predicts backwards. A value near $\frac{1}{2}$ implies that the model is no better than random guessing. Finally, compare the ROC AUC values of different models to assess which model works best for each data set.

Post hoc. For further comparison of FeGAN against the baselines, we use ranking based post hoc testing [36]. The experimental results we gather are ordinal and our models are independent, but run the experiments under the same conditions. Additionally, we assume that each model performs differently for each data set. Thus, the non-parametric Kruskal-Wallis test [28] is well suited for our use-case. Furthermore, due to its superior ability to balance Type I and Type II errors compared to other test like Nemenyi's [34], we choose the Conover-Iman test [11]. For that, we rank the median AUC of each model per dataset. We assign 1 to the highest AUC and 7 to the lowest. Now, the first step is to apply the Kruskal-Wallis test to check whether there are significant differences among the

models (based on a significance level α). If this test is negative, there is no need for further evaluation as there are no significant differences. Finally, we apply the Conover-Iman test. Assuming the Kruskal-Wallis test is positive with regards to α, the test now compares, pairwise, if there is a significant difference between the average rank of two models.

5.2. Experiments

5.2.1. One-Class classification

To compare the models, we analyze their performance inside the One-Class classification problem for Outlier Detection [32]. We use the same methodology for every model, including all baselines, and then gather the ROC-AUC of the predictions. To enable reproducibility and prevent the outcome of the experiment to be reliant on non-deterministic behaviour due to random splitting and feature selection, we repeat the experiment for five seeds. While more repetitions with different seeds help improve the stability of measurements, this was not feasible due to the large computational cost of deep models in combination with the time constraints of this thesis. In this experiment, we train the models on inlier samples only and follow Algorithm 3. The first step of the experiment is to split our data. We randomly choose 80% of the inlier class as the training set. The other 20% of the inlier class and all outlier samples then are assigned to the test set. For each seed individually, this split stays the same, meaning that every model trains on the same training set and test set. During the training process, we set the *stop_epochs* to 100. We do this to gain insight in the learning behaviour of FeGAN and also then choose the number of epochs where FeGAN performs the best measured by ROC-AUC. For every data set, except KDDCup99, we choose a batch size of min(500, #*samples*) as this proved to work well [30]. KDDCup99 is, by far, the biggest data set of all. With the resources available, training with a batch size of 500 would not be feasible. Therefore, we choose a batch size of 1000 in this case.

Algorithm 3 One-Class Classification

Input: Set of all models used = Models, List of seeds = Seeds
```
for seed ∈ Seeds do
    Set the current seed
    # Split the data in two sets
    train_set = 80% of the inlier samples
    test_set = outliers and the other 20% of the inliers
    for model ∈ Models do
        Train the model on the training set
        Calculate AUC of the model on the test set
Get median AUC and standard deviation of each model and compare them
```

5.2.2. Sensitivity to parameter K

The parameter K decides how many Discriminators FeGAN uses. Therefore, this also decides how many Feature Subspaces are used to find hidden outliers. FeGAN's runtime heavily depends on K, as it scales linearly with it. While more Feature Subspaces can result in better prediction accuracy, it might not be worth the additional required resources. Thus, it is important to keep the cost-benefit ratio of adding more Discriminators in mind.

To analyze this, we run the One-Class classification experiment with a range of values for K. We then choose, like with the One-Class classification, the highest reached median ROC-AUC for all values of K.

6. Evaluation

In this chapter, we evaluate the prediction quality of FeGAN by comparison with the baseline models, while also analyzing framework-specific properties of GAN influencing the performance of FeGAN.

First, we evaluate the the One-Class classification experiment with regards to the performance measured by the ROC-AUC [5]. After that, we evaluate the Conover-Iman test results [11], followed by an analysis of FeGAN's sensitivity to the number of Discriminators. Then, we talk about the training of our model and difficulties we face during the process. Finally, we talk about the limitations of FeGAN and this thesis.

6.1. One-Class Classification

To compare and evaluate models, we use the ROC-AUC. In table 6.1, we display the results of the experiments. We compare the median ROC-AUC value of each model across all the seeds. In the table, the highest value for each data set is marked bold.

Results FeGAN V1 and V2 are of different complexities. Thus, they perform differently for each data set as seen in table 6.1. To take that into account, we follow our recommendations and only look at one version for each data set, the one we would recommend based on the amount of features. Furthermore, it is important to mention that we select the highest reached median ROC-AUC for FeGAN over all epochs. We do this because developing a proper stopping criteria for an unsupervised method like FeGAN is not feasible time-wise.

Out of eight data sets we experiment on, FeGAN has the highest median AUC for five of those. This shows that, even without much domain knowledge to choose Neural Network architectures, FeGAN manages to outperform the shallow and deep baselines on a majority of our data sets, indicating that Feature Subspace search works well for GANs. Even with a random Feature Bagging Ensemble, where Feature Subspaces are not chosen optimally, this approach shows to outperform several baselines in these benchmark data sets. Looking at Table 6.1, one can observe FeGAN does not always outperform every single baseline model. We suspect this is due to different reasons. For one, we lack computational resources and time to test higher dimensional data sets. Thus, it is hard to gain further insight into the behaviour of FeGAN for these cases like InternetAds. Therefore, especially for high dimensional data sets, more experiments should be executed to properly evaluate FeGAN's performance for those. Another possible reason is that our chosen Network architecture might not be complex enough to properly learn the underlying data distribution for some data sets. However, even when not able to outperform baseline models in every case, one can easily deduce the overall great performance and future potential of FeGAN.

	Arrhythmia	Waveform	SpamBase	InternetAds
FeGAN V1	0.7488 ± 0.0146	0.8499 ± 0.0352	0.6641 ± 0.0159	0.7592 ± 0.0054
FeGAN V2	**0.7500 ± 0.0117**	0.8293 ± 0.0128	**0.7368 ± 0.0144**	0.8068 ± 0.0155
LOF	0,7277 ± 0,049	0,7561 ± 0,0083	0,7218 ± 0,0124	**0,8545 ± 0,0077**
KNN	0,7334 ± 0,0421	0,7658 ± 0,0053	0,7028 ± 0,0059	0,809 ± 0,005
OC_SVM	0.7442 ± 0.0379	0.5514 ± 0.0073	0.6288 ± 0.0092	0.7026 ± 0.0046
AnoGAN	0,4752 ± 0,0803	0,6439 ± 0,1559	0,4522 ± 0,092	0,5916 ± 0,1124
MO_GAAL	0,7447 ± 0,024	**0,8563 ± 0,0071**	0,6774 ± 0,0574	0,7024 ± 0,0299
Deep SVDD	0,7308 ± 0,0384	0,65 ± 0,0564	0,7249 ± 0,0367	0,8205 ± 0,0089
	Annthyroid	Cardiotocography	Ionosphere	KDDCup99
FeGAN V1	0.5030 ± 0.0269	0.6917 ± 0.0807	0.7910 ± 0.0344	**0.9884 ± 0.0124**
FeGAN V2	**0.7464 ± 0.0304**	**0.8864 ± 0.0443**	0.9035 ± 0.0143	0.9806 ± 0.0067
LOF	0,6769 ± 0,005	0,8038 ± 0,0094	0,9432 ± 0,0077	0,9222 ± 0,0029
KNN	0,6409 ± 0,0072	0,7763 ± 0,0054	**0,9778 ± 0,0077**	0,976 ± 0,0011
OC_SVM	0.4675 ± 0.0052	0.8496 ± 0.0079	0.8129 ± 0.0247	0.9854 ± 0.0006
AnoGAN	0,4875 ± 0,0453	0,703 ± 0,0844	0,8538 ± 0,0327	0,9367 ± 0,0699
MO_GAAL	0,5073 ± 0,0445	0,535 ± 0,0465	0,7198 ± 0,0277	0,2493 ± 0,0346
Deep SVDD	0,3328 ± 0,2809	0,6801 ± 0,2062	0,9511 ± 0,0151	0,9111 ± 0,0372

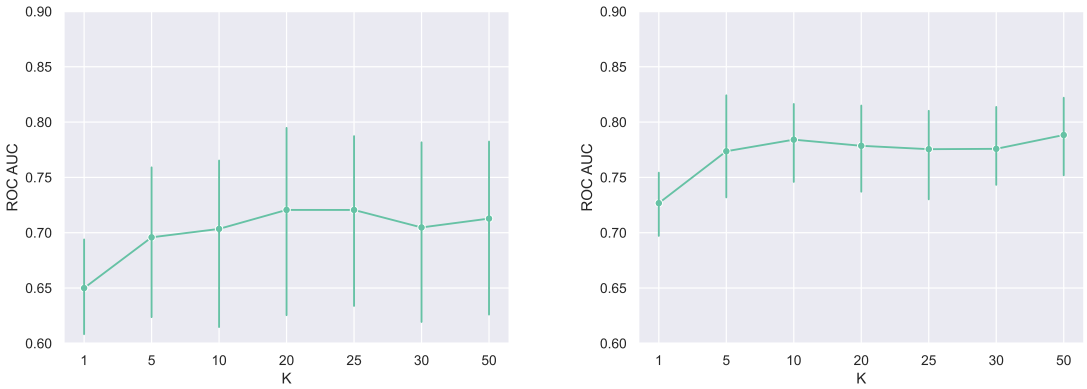
Table 6.1.: Results of One-Class classification. Median ROC-AUC ± standard deviation.

Additionally, we statistically test the performance of out model utilizing a Conover-Iman test [11] in Table 6.2 to assess whether FeGAN's performance ranks significantly higher than the other baseline models.

Conover-Iman. In table 6.2, we show the results of a Conover-Iman test after a positive Kruskal-Wallis test. The Conover-Iman test signals significant differences for pairwise comparison between methods. We test for $\alpha_1 = 0.1$, where '+' indicates a positive and '−' indicates a negative difference, and $\alpha_2 = 0.05$, signaled by '++' and '− −'. Clearly, the table shows that FeGAN outperforms four out of six baselines significantly even with regards

	FeGAN	LOF	kNN	OC-SVM	AnoGAN	MO-GAAL	DeepSVDD
FeGAN	=			++	++	++	++
LOF		=			++	+	
kNN			=		++	+	
OC-SVM	− −			=			
AnoGAN	− −	− −	− −		=		-
MO-GAAL	− −	−	−			=	
DeepSVDD	− −				+		=

Table 6.2.: Results for Conover-Iman's test on the rankings of the models



(a) V1 (b) V2

Figure 6.1.: Sensitivity of FeGAN to the number of Discriminators K

to α_2. Only for LOF and KNN there is no significant performance difference, which also is in line with table 6.1, as beating these two methods is, again, not trivial.

Overall, the results gained by the One-Class classification are very promising and motivate us to further work on FeGAN, improving the performance and prediction quality even more.

6.1.1. Sensitivity to parameter K

Looking at Figure 6.1, the first interesting observation is that FeGAN V2 outperforms V1 noticeably while facing less variance. Though this was not the goal of this experiment, it is an important finding and should be analyzed through further experiments.

As for the sensitivity to parameter K, V1 and V2 react differently. While V1 peaks around 20 to 25 Discriminators, V2 first peaks at 10 Discriminators and then goes up even further at 50. Since V1 should be used for smaller data sets, it is expected that, after a certain number, more Discriminators may not yield more performance. For V2 it is clear that the cost-benefit ratio of 10 versus 50 Discriminators is not worth it. The additional accuracy does not justify the much longer runtime.

However, the results of this experiment do not completely match with our One-Class classification experiment. There, we used $K = 2 * \sqrt{dimensions}$, while in this experiment, we used a fixed number of K for all data sets, regardless of their size or dimensionality. A more fine-grained analysis might bring more insight in how to dynamically choose K.

6.1.2. Training

Looking at the training histories of the Generator and Discriminators of FeGAN, as well as the ROC-AUC curve over the epochs, brings valuable information about the performance of FeGAN. In this section, we analyze different plots to gain in depth insight about the training process and possible flaws and strengths of FeGAN. Keep in mind that FeGAN consists of one Generator and k Discriminators. Hence, the displayed loss is the mean loss

over all Discriminators. In the plots, we show the mean ROC-AUC and the 95% confidence interval.

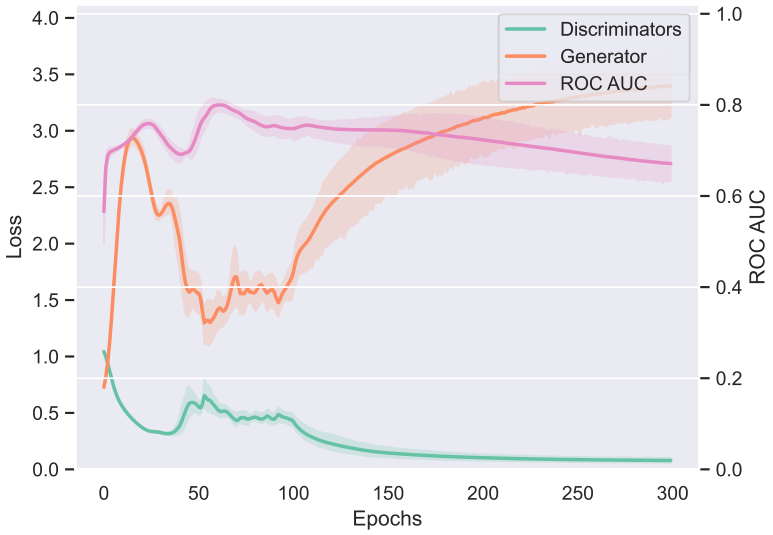


Figure 6.2.: Training history of FeGAN V2 on InternetAds

Looking at Table 6.1, it is clear that FeGAN lacks performance when training on InternetAds. With 1555 Features, this data set has the highest number of dimensions and it is not surprising that FeGAN struggles with this. The Generator as well as the Discriminators loss curves are not unusual for GANs [30] and are expected to behave as in Figure 6.2. Now, in this plot, it looks like FeGAN is still too weak. After about 60 epochs, the ROC-AUC starts dropping. This can be an indicator of overfitting, but since we still train with simple network architectures, overfitting the overall model on such a high dimensional data set is very unlikely. More likely, FeGAN simply fails to capture the underlying data distribution well. Possible resons for this could be the simple network architecture or the randomly chosen subspaces. With 2^{1555} possible Feature Subspaces, choosing only k informative subspaces to train on is very hard, especially with random Feature Subspace selection. Also, an Ensemble of k Discriminators is limited by the available computational resources. Training 1000 Discriminators for one model is, at least for this thesis, neither realistic nor feasible.

Figure 6.3 shows a negative example for FeGAN. The combination of a rapid drop in ROC-AUC and the sudden, fast, divergence of the Generator indicates that the Generator generates samples that are easy for the Discriminators to distinguish from real samples. Thus, the training process fails. However, a plausible explanation for this is the implemented stopping mechanism for the Generator. Right after the Generator stops learning, the ROC-AUC reaches a peak. But since the Discriminators keep learning after that, it explains why the Generator fails to perform, resulting in a strong case of overfitting for the Discriminators.

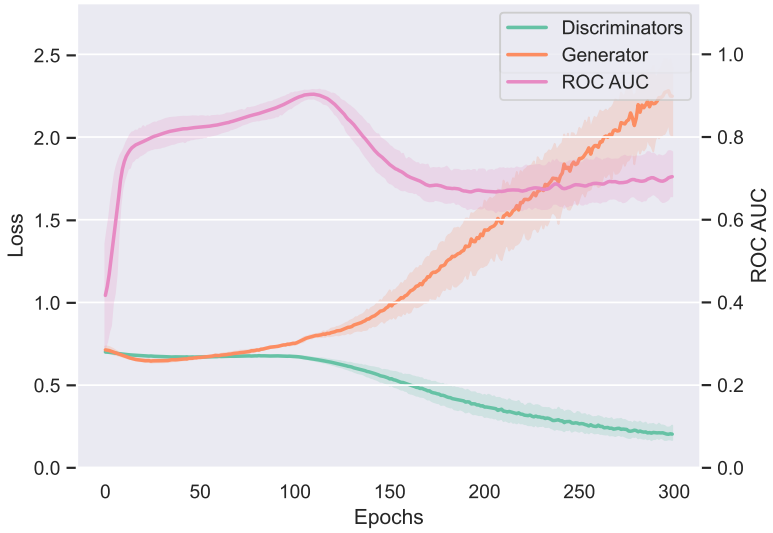


Figure 6.3.: Training history of FeGAN V2 on Ionosphere

Figure 6.4 is a positive example of FeGAN's training. Over time, the Generator's loss rises slowly and saturates (due to optimization problem 4.2), the Discriminators' loss sinks, and the ROC-AUC becomes higher over time without collapsing.

6.1.3. Limitations

The biggest limitations of this thesis are time and resources. As time is limited and the computational resources are scarce in our current experimental set-up, we have to focus on the most important parts of FeGAN, the Feature Subspace Selection and the extension of the GAN framework by an Ensemble. There is not much time to experiment with the Network Architectures or to study hyperparameter sensibility. We are convinced that adjusting FeGAN's Neural Network architectures to match different use cases/data sets will even further improve the accuracy of the model. But since we run experiments on several data sets, some of them very computationally expensive, these kinds of adjustments are not possible. Additionally, exploring a correct stopping criteria is a must for unsupervised methods. This is one of the most important parts missing in FeGAN and should be the first step towords a correct implementation, but, again, it was not possible during the short time-frame of this thesis.

Furthermore, we lack domain knowledge to implement these kinds of adjustments due to the unsupervised setting. While it is possible to make the Network architecture more precise by adding domain knowledge [12], this falls out of the scope of this thesis.

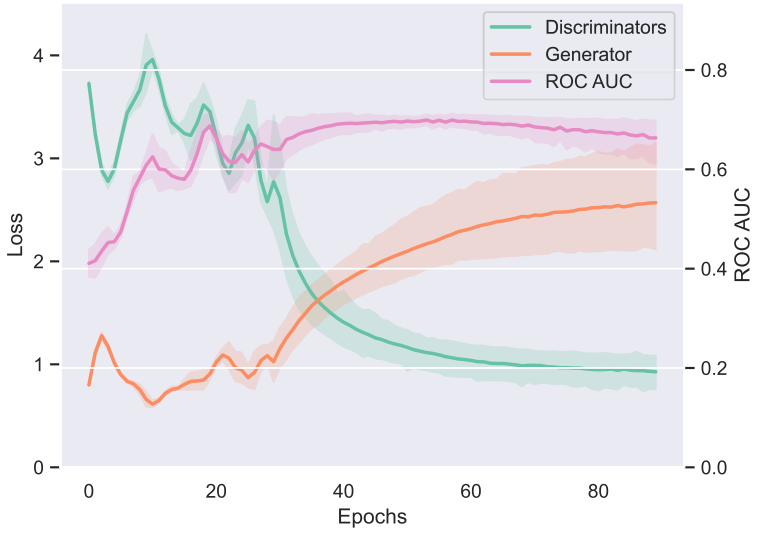


Figure 6.4.: Training history of FeGAN V2 on Annthyroid

7. Conclusion

In this thesis, we proposed FeGAN, a novel extension of the GAN framework [17], to enable the detection of hidden outliers. We extended the vanilla GAN model by multiple Discriminators to form an Ensemble, improving prediction quality. While each Discriminator learns on an individual, randomly drawn, Feature Subspace, the single Generator generates samples for all of them. This extension did not come without challenges, as the target function of a vanilla GAN does not allow the usage of multiple Discriminators at once. Thus, we adjusted the target function to take the novel Ensemble structure of our model into consideration.

After experimenting with FeGAN in the field of Novelty Detection and analyzing the results, we quickly noticed the great performance and potential in FeGAN. Beating most shallow and deep baselines, FeGAN shows that Subspace Search improves the ability to find (hidden) outliers without the need of a complex network architecture. However, during experimentation, we not only observed great results, but also that the choice of Feature Subspaces matters. While, in this thesis, random Subspace selection was sufficient to yield good performance, there were a few cases in which it became obvious that the random choice hindered FeGAN's full potential. Therefore, a different Subspace Selection method should be explored. Finally, we evaluated the impact of the number of Discriminators (k) in the Ensemble, which directly results in more explored Feature Subspaces. This last experiment showed that, while FeGAN's performance is dependent on this parameter, the performance of the model converges after a certain number of Discriminators. As FeGAN's runtime scales linearly with this number, this observation is helpful to prevent a waste of resources and time.

Finally, we conclude that, in this thesis, we showed the power of FeGAN and Subspace Search with GANs for Novelty Detection. We are aware this potential has not been fully exploited yet and recommend further research.

7.1. Future Work

Scarcity of computational resources and time constraints are the biggest obstacles as to why we could not research FeGAN further. Nevertheless, there are many open topics that should be tackled to improve FeGAN. First, analysis of FeGAN's behaviour when dealing with big data sets or data sets with high dimensionalities will bring more insight into how FeGAN handles theses cases, leaving room for adjustments of the model. Second, a stopping criteria should be explored. We assume this will improve the prediction quality of the model a lot due to the prevention of overfitting, but could not implement a stopping criteria, as this process is very time consuming. Additionally, the network architecture of FeGAN can also be adjusted and experimented with, as this broadens the use cases of FeGAN. Even

though we were interested in exploring both of these issues, they consume a lot of time for deep, unsupervised methods, making it hard for us to experiment. Furthermore, it is possible to extend the Ensemble structure to not only incorporate multiple Discriminators, but also multiple Generators. Following a similar approach to MO-GAAL [30], this could prevent the problem of mode collapse. However, this adjustment requires a new target function aswell. Lastly, a sensibility study of FeGAN to the selection of Feature Subspaces is required to evaluate how the Ensemble structure of the model handles different Subspace Selection methods.

Bibliography

[1] Charu C Aggarwal and Philip S Yu. "Outlier detection for high dimensional data". In: *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*. 2001, pp. 37–46.

[2] Charu C. Aggarwal. *Outlier Analysis*. Springer, 2013. Chap. 6. ISBN: 978-1-4614-6396-2. URL: http://dx.doi.org/10.1007/978-1-4614-6396-2.

[3] Kevin Beyer et al. "When is "nearest neighbor" meaningful?" In: *Database Theory—ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings 7*. Springer. 1999, pp. 217–235.

[4] Léon Bottou. "Stochastic gradient descent tricks". In: *Neural Networks: Tricks of the Trade: Second Edition*. Springer, 2012, pp. 421–436.

[5] Andrew P. Bradley. "The use of the area under the ROC curve in the evaluation of machine learning algorithms". In: *Pattern Recognition* 30.7 (1997), pp. 1145–1159. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(96)00142-2. URL: https://www.sciencedirect.com/science/article/pii/S0031320396001422.

[6] Leo Breiman. "Bagging predictors". In: *Machine learning* 24 (1996), pp. 123–140.

[7] Markus M Breunig et al. "LOF: identifying density-based local outliers". In: *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*. 2000, pp. 93–104.

[8] Robert Bryll, Ricardo Gutierrez-Osuna, and Francis Quek. "Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets". In: *Pattern Recognition* 36.6 (2003), pp. 1291–1302. ISSN: 0031-3203. DOI: https://doi.org/10.1016/S0031-3203(02)00121-8. URL: https://www.sciencedirect.com/science/article/pii/S0031320302001218.

[9] Guilherme O Campos et al. "On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study". In: *Data mining and knowledge discovery* 30 (2016), pp. 891–927.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey". In: *ACM computing surveys (CSUR)* 41.3 (2009), pp. 1–58.

[11] WJ Conover and RL Iman. *Multiple-comparisons procedures. Informal report*. Tech. rep. Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 1979.

[12] Tirtharaj Dash et al. "Incorporating domain knowledge into deep neural networks". In: *arXiv preprint arXiv:2103.00180* (2021).

[13] Federico Di Mattia et al. "A survey on gans for anomaly detection". In: *arXiv preprint arXiv:1906.11632* (2019).

[14] Thomas G Dietterich. "Ensemble methods in machine learning". In: *International workshop on multiple classifier systems*. Springer. 2000, pp. 1–15.

[15] Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. "Adversarial feature learning". In: *arXiv preprint arXiv:1605.09782* (2016).

[16] Matthias Feurer and Frank Hutter. "Hyperparameter optimization". In: *Automated machine learning: Methods, systems, challenges* (2019), pp. 3–33.

[17] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[18] Gongde Guo et al. "KNN model-based approach in classification". In: *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE: OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2003, Catania, Sicily, Italy, November 3-7, 2003. Proceedings*. Springer. 2003, pp. 986–996.

[19] Songqiao Han et al. "Adbench: Anomaly detection benchmark". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 32142–32159.

[20] Marti A. Hearst et al. "Support vector machines". In: *IEEE Intelligent Systems and their applications* 13.4 (1998), pp. 18–28.

[21] Tin Kam Ho. "The random subspace method for constructing decision forests". In: *IEEE transactions on pattern analysis and machine intelligence* 20.8 (1998), pp. 832–844.

[22] Tin Kam Ho. "The random subspace method for constructing decision forests". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.8 (1998), pp. 832–844. DOI: 10.1109/34.709601.

[23] Thomas Hofmann, Bernhard Schölkopf, and Alexander J Smola. "Kernel methods in machine learning". In: (2008).

[24] He Huang, Philip S Yu, and Changhu Wang. "An introduction to image synthesis with generative adversarial nets". In: *arXiv preprint arXiv:1803.04469* (2018).

[25] Fabian Keller, Emmanuel Muller, and Klemens Bohm. "HiCS: High contrast subspaces for density-based outlier ranking". In: *2012 IEEE 28th international conference on data engineering*. IEEE. 2012, pp. 1037–1048.

[26] Mario Köppen. "The curse of dimensionality". In: *5th online world conference on soft computing in industrial applications (WSC5)*. Vol. 1. 2000, pp. 4–8.

[27] Hans-Peter Kriegel, Matthias Schubert, and Arthur Zimek. "Angle-based outlier detection in high-dimensional data". In: *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2008, pp. 444–452.

[28] William H Kruskal and W Allen Wallis. "Use of ranks in one-criterion variance analysis". In: *Journal of the American statistical Association* 47.260 (1952), pp. 583–621.

[29] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: ().

[30] Yezheng Liu et al. “Generative adversarial active learning for unsupervised outlier detection”. In: *IEEE Transactions on Knowledge and Data Engineering* 32.8 (2019), pp. 1517–1528.

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

[32] Mary M Moya and Don R Hush. “Network constraints and multi-objective optimization for one-class classification”. In: *Neural networks* 9.3 (1996), pp. 463–474.

[33] Emmanuel Müller et al. “Outlier ranking via subspace analysis in multiple views of the data”. In: *2012 IEEE 12th international conference on data mining*. IEEE. 2012, pp. 529–538.

[34] Peter Björn Nemenyi. “Distribution-free Multiple Comparisons”. PhD thesis. Princeton University, 1963.

[35] Chigozie Nwankpa et al. “Activation functions: Comparison of trends in practice and research for deep learning”. In: *arXiv preprint arXiv:1811.03378* (2018).

[36] Dulce G Pereira, Anabela Afonso, and Fátima Melo Medeiros. “Overview of Friedman’s test and post-hoc analysis”. In: *Communications in Statistics-Simulation and Computation* 44.10 (2015), pp. 2636–2653.

[37] Marco AF Pimentel et al. “A review of novelty detection”. In: *Signal processing* 99 (2014), pp. 215–249.

[38] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434* (2015).

[39] Van Rossum. “Python tutorial, technical report CS-R9526”. In: *(No Title)* (1995).

[40] Sebastian Ruder. “An overview of gradient descent optimization algorithms”. In: *arXiv preprint arXiv:1609.04747* (2016).

[41] Lukas Ruff et al. “Deep one-class classification”. In: *International conference on machine learning*. PMLR. 2018, pp. 4393–4402.

[42] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning representations by back-propagating errors”. In: *nature* 323.6088 (1986), pp. 533–536.

[43] David E Rumelhart, James L McClelland, and CORPORATE PDP Research Group. *Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1: Foundations*. MIT press, 1986.

[44] Thomas Schlegl et al. “Unsupervised anomaly detection with generative adversarial networks to guide marker discovery”. In: *International conference on information processing in medical imaging*. Springer. 2017, pp. 146–157.

[45] Bernhard Schölkopf. “The kernel trick for distances”. In: *Advances in neural information processing systems* 13 (2000).

[46] Bernhard Schölkopf et al. “Support vector method for novelty detection”. In: *Advances in neural information processing systems* 12 (1999).

[47] Burr Settles. "Active learning literature survey". In: (2009).

[48] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. "Activation functions in neural networks". In: *Towards Data Sci* 6.12 (2017), pp. 310–316.

[49] David MJ Tax and Robert PW Duin. "Support vector data description". In: *Machine learning* 54 (2004), pp. 45–66.

[50] Steven Walczak and Narciso Cerpa. "Heuristic principles for the design of artificial neural networks". In: *Information and software technology* 41.2 (1999), pp. 107–117.

[51] Hongzhi Wang, Mohamed Jaward Bah, and Mohamed Hammad. "Progress in outlier detection techniques: A survey". In: *Ieee Access* 7 (2019), pp. 107964–108000.

[52] Roger Weber, Hans-Jörg Schek, and Stephen Blott. "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces". In: *VLDB*. Vol. 98. 1998, pp. 194–205.

[53] Martha White et al. "Convex multi-view subspace learning". In: *Advances in neural information processing systems* 25 (2012).

[54] Houssam Zenati et al. "Efficient gan-based anomaly detection". In: *arXiv preprint arXiv:1802.06222* (2018).

[55] Yue Zhao, Zain Nasrullah, and Zheng Li. "PyOD: A Python Toolbox for Scalable Outlier Detection". In: *Journal of Machine Learning Research* 20.96 (2019), pp. 1–7. URL: http://jmlr.org/papers/v20/19-011.html.

[56] Jia-Jie Zhu and José Bento. "Generative adversarial active learning". In: *arXiv preprint arXiv:1702.07956* (2017).

[57] Arthur Zimek, Ricardo JGB Campello, and Jörg Sander. "Ensembles for unsupervised outlier detection: challenges and research questions a position paper". In: *Acm Sigkdd Explorations Newsletter* 15.1 (2014), pp. 11–22.

[58] Arthur Zimek, Erich Schubert, and Hans-Peter Kriegel. "A survey on unsupervised outlier detection in high-dimensional numerical data". In: *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5.5 (2012), pp. 363–387.

A. Appendix

A.1. Code

The results of this thesis will be further researched and published in a scientific paper. Thus, our code will be released with that paper and not with this thesis.