technische universität
dortmund

Bachelor Thesis

Surveying the effects of Lipschitz continuity on the Robustness of Anomaly Detection Algorithms

Justin Wickes

June 12, 2024

Reviewer:
Prof. Dr. Emmanuel Müller
M.Sc. Simon Klüttermann

Technical University of Dortmund
Department of Computer Science
Chair IX - Data Science and Data Engineering
https://ls9-www.cs.tu-dortmund.de/

Contents

Contents

1 Introduction

1.1 Motivation and Background

Machine learning models are of ever-growing importance in the modern world. Whether it's recommendation algorithms used by services like Youtube and Netflix, autonomous vehicles, automated translation apps like Google Translate and DeepL, predictive text on smartphones, or even chatbots like ChatGPT, machine learning is becoming ever more ubiquitous. In an age of Big Data, machine learning is a tool that can help extract valuable insights from volumes of data so large, that the human mind struggles to comprehend them.

Anomaly Detecting models are a specific subset of models used in domains such as financial fraud, cybersecurity and medicine. Their explicit goal is to find patterns in data that do not conform to an expected behavior [12]. These anomalies can range from tumors or unorthodox growths found in medical imaging [66] to suspicious purchases on a credit card [9]. The presence of anomalies can be indicative of a larger issue or fault, that in many cases should be identified and remedied as soon as possible. These models can be appraised based on multiple criteria. One of which is performance, i.e. the sheer accuracy of their predictions. Another is Interpretability, which describes how predictable and well-understood the decisions of a model are [47]. Many models are black boxes, which makes understanding why a specific prediction or decision was made by a model difficult to comprehend. Yet another criteria is robustness, which in simple terms describes a given model's ability to preserve it's accuracy when faced with uncertainty. This final criteria is of particular interest to us.

Often, models perform well in "sterile" experiment environments, where hyperparameters of the model can be tuned to a well known and understood data set. When these models come into contact with reality and are actually implemented in security and safety critical domains, as anomaly detection often is, non-robust models can suffer. Anomalies, by their very nature, exhibit behavior that is different from the expected behavior of normal data. For this reason, robustness is critical in a plethora of machine learning models, but especially in anomaly detection, where detecting and understanding uncertainties is the goal. For example, self-driving cars, need to drive as safe as or even safer than a human being. These cars require robust anomaly detection systems in their sensors, in order to ensure road safety. For instance, cyberattacks, weather, and technical issues all have the potential to cause fatal crashes if the sensors

cannot correctly identify them as such [6]. It is possible, through adversarial attacks, which we will discuss in detail later, to cause self-driving cars to misclassify stop signs as speed limit signs [28]. This can cause these cars to speed through stop signs potentially causing fatal injuries. This is but one specific example, but for this reason, robustness is critical in anomaly detection models.

Throughout this paper we will describe how Lipschitz continuity, a mathematical property that guarantees an amount of certifiable robustness [49][29], affects various anomaly detection algorithms. These algorithms are namely the Deep Convolutional Autoencoder, DeepSVDD, and AnoGAN. We will explore the changes required to create Lipschitz continuous versions of these anomaly detection algorithms as well as how these algorithms perform under this Lipschitz constraint.

1.2 Thesis Structure

In Section 2 this paper will initially introduce the concept of unsupervised anomaly detection followed by a thorough explanation of each individually chosen anomaly detection algorithm. A portion of the math underpinning the function of these algorithms will be discussed along with any pertinent, more granular elements of their architectures such as convolutions and batch normalization [32]. Following this, Section 3 will discuss the concept of Robustness in detail along with the Lipschitz property. This section will also discuss why robust models are important and what specifically they are robust against. Section 4 will detail the implementation of these algorithms beginning with an explanation of Deel-Lip, a library that implements K-Lipschitz layers. Then each Algorithm's concrete implementation will be discussed, including architecture and hyper-parameter choices. Next in Section 5 our two main experiments will be discussed, along with their results. These results will be analyzed and discussed at length. Then in Section 6 possible improvements to our implementation, experiments, and even anomaly detection algorithm choices will be discussed. Finally in Section 7 our conclusion will summarize our results and analysis, condensing all that we've learned.

1.3 Related Work

The Lipschitz property is a well-understood mathematical property with many papers utilizing it to make certifiable gains in robustness and studying it's effects on the expressiveness of models. The foremost important of these are "Achieving robustness in classification using optimal transport with hinge regularization" by Serrurier et. al. [62], as well as "Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks" by Li et. al. [40]. The

latter being quite influential on the former as well. These papers offer a general implementation for K-Lipschitz networks and are thus of immense importance. The Deel-Lip library, which we use in our implementation of our Lipschitz continuous anomaly detection algorithms, is created and maintained by Serrurier and his associates, who based the implementation of the library on the findings in their paper. Naturally we must also name "Deep One-Class Classification" by Ruff et. al. [56] whose paper served as a basis for us to implement both our DCAE and DeepSVDD models. The DCAE was not a model that they had invented themselves however. The idea of the DCAE (at that point simply called a CAE) was first introduced in "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction" by Masci et. al. [43]. Our final anomaly detection algorithm was the AnoGAN, first published by Schlegl et. al. in "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery" [59]. These papers and their work all serve as the backbone of our further experiments.

There exists quite a few papers related to our area of research. Weng et. al. create a novel robustness metric based on the local Lipschitz constant of a network. They run into issues however, trying to efficiently estimate said Lipschitz constant and thus resort to using Extreme Value Theory to justify sampling points and their corresponding output to estimate the constant. Anil et. al. before their work on gradient norm preserving methods for convolutions [40], introduced gradient norm preserving Lipschitz continuous models for Multilayer Perceptrons [3]. This focus on a gradient norm preserving architecture, seeks to solve the expressiveness issues observed in Lipschitz constrained neural networks. Cisse et. al. [16] and Yoshida et. al. [80] were not the first papers to research the benefits of constraining the Lipschitz constant, but they were the first to achieve Lipschitz continuity through architectural constraints. Many previous methods had tried to constrain the network through kinds of regularization and while achieving convincing results had not actually enforced a global Lipschitz constant. These architectural-based methods provably enforce Lipschitz continuity in contrast. Both of these methods constrain the Lipschitz constant by constraining the spectral norm of the weights of their layers. Miyato et. al. similarly shows how spectral normalization can be applied to GANs to create Lipschitz constrained GANs that benefit from more stable training [46]. Finally Tsuzuku et. al. also enhance the robustness of their network using a Lipschitz constraint, proving in the process the adversarial robustness that a Lipschitz constraint can offer [73]. This is the first work, as far as we know, that so clearly formulates the adversarial robustness gains mathematically. Ono et. al. continues to try to improve on this work by providing a more computationally efficient version of the provably robust defense offered by Tsuzuku et. al. [49].

2 Anomaly Detection Algorithms

First we will describe the task of anomaly detection and discuss the different categories of anomaly detection. Then we will move to detailing the three anomaly detection algorithms that we have chosen for this experiment. The first of which will be a Deep Convolutional Autoencoder. We will explain in detail what an Autoencoder is as well as what convolutional models are. Next we will detail our second algorithm DeepSVDD, it's two variants, and the specific properties and architectural restrictions that DeepSVDD imposes on it's model. Finally we will describe our third anomaly detection algorithm AnoGAN. To do this we will first need to explain what a GAN is and then what a DCGAN is.

2.1 Anomaly Detection

Anomaly detection, as previously defined, is the study and identification of anomalies, sometimes referred to as outliers. The defining trait of these anomalies is that they stray from what would be defined as "normal" data, which is the overwhelming majority of data in any given domain. Unless otherwise specified, calling data "normal" refers to their orthodox behavior and has nothing to do with the Gaussian distribution. Anomaly detection is a wide discipline that encompasses a plethora of domains, so there exist many anomaly detection algorithms tailored specific to the domain of interest. The reasons anomaly detection algorithms may be employed can vary. One may simply wish to identify anomalies or noisy data in order to remove them from their data set, so they can ensure they are performing statistical analysis on clean data. The now more common reason one may perform anomaly detection is because the existence of the anomalies themselves are of interest [12]. Anomalies can often be indicative of a lack of understanding over a given domain and can also serve to check and correct human biases. Anomalies can also be indicative of critical failures or weakpoints in a system, one can simply consider the domains of cybersecurity or financial fraud. The study of anomalies themselves can lead to important insights about the domain that's being studied.

In figure 2.1, the cluster of datapoints N_1 and N_2 are normal data, whereas points o_1 and o_2 are outliers. O_3 is a cluster of points that could also be considered an outlier. In figure 2.1, the location of O_3 in the two dimensional space leads to it being classified as an anomaly just like the point anomalies o_1 and o_2. The volume of samples that lie within O_3 play a part in it's identification as an anomaly too. The number of points that lie within the normal clusters dwarfs

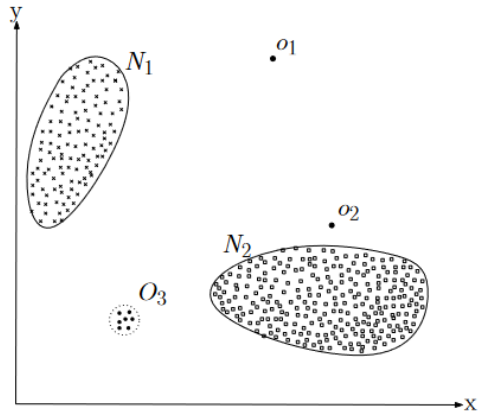

Figure 2.1: Example anomalies in a 2 dimensional space [12]

O_3 by such a large margin and thus O_3 is determined to be a collective anomaly as opposed to another smaller grouping of normal points. It is of course possible that a collective anomaly such as O_3 may in reality be normal, and that the distribution of the data set is not well representative of the real distribution. This leads into another challenge of anomaly detection, namely how to define normal data.

The first major consideration is that, a data set should capture the distribution of data that is present in reality. Acquiring a large enough labeled data set that represents all types of behavior can range from cumbersome to outright impossible. Once a sufficient data set is obtained, it can be difficult to form a general rule for finding anomalies as what is anomalous to one domain may not be anomalous in another. For example small fluctuations in values may be a trivial occurrence in one domain while being much more uncommon in another. Furthermore it can be difficult to create a precise boundary for separating normal data from anomalies in the first place, and in some cases the boundary may be inherently imprecise and unable to be specified in an exact manner. In some domains the concept of ”normal behavior” may not even be static and could change over the course of time, causing this modelling to be even more difficult. Likewise in many domains, anomalies are intrinsically dynamic, making it exceedingly difficult to account for the behavior of all anomalies. One must also consider that some anomalies may be misclassified as we have alluded to under Figure 2.1. Anomalies that lay outside some defined boundary between normal and anomalous data may in actuality be normal rather than anomalous and their misclassification may stem from an inadequate data set that is either inaccurate or fails to capture the the complete nature of the actual distribution of the data. Often anomalies are simply intrinsic to the domain they belong to, but some are caused by bad actors. These anomalies could be made to be camouflaged among normal data, making the distinction between normal and

anomaly that much harder. Such attacks will be discussed in further detail in Section 3. In summary, there are a multitude of pitfalls that can impede the goal of anomaly detection.

2.2 Categories of Anomaly Detection

Chandola et. al. describe 3 categories of anomaly detection algorithms [12]. The first of which is supervised anomaly detection. This form of anomaly detection presupposes that a data set with labeled normal and labeled anomalous data is given. This presupposition is challenging to fulfill for a variety of reasons. This labeling of data often needs to be done by a human being and is thus a laborious effort, as these data sets can contain thousands to millions of individual data points. Furthermore it can be difficult to obtain a data set that even covers all possible anomalies that can occur, and due to the aforementioned dynamic nature of anomalies, it is possible that new kinds of anomalies emerge. Moreover the amount of individual anomalies will very likely be much smaller than the amount of normal samples, which causes difficulties in modelling the behavior of the anomalies. Assuming all of these challenges can be overcome, then a supervised anomaly detection algorithm will usually be trained on both normal data and anomalies so that when it receives unseen, unlabeled data, it can sort them into one of the two classes.

The second category of anomaly detection is semi-supervised. This category, as the name suggests, lies in between supervised and unsupervised algorithms. For this category of anomaly detection, it is assumed that a portion of the available training data is labeled, not necessarily all of it. In the overwhelming majority of semi-supervised algorithms, only normal samples or a portion of normal samples are labeled and anomalies remain unlabeled. These methods focus on training a model to identify normal samples and all other samples are thus represented poorly by the model. The likelihood that a given sample was generated by the model is then used as a metric to determine which samples are anomalies. Some semi-supervised algorithms use labeled anomalous samples during training, but these algorithms are rare in comparison and many are domain specific.

The third category of anomaly detection algorithm is unsupervised. These techniques operate under the Assumption that the number of normal data points in a given data set is much larger than the number of anomalies. They therefore do not require labeled data of any kind and are thus very broadly applicable. They are however weak to data sets that challenge this assumption. Data sets that have a larger proportion of anomalies will increase the likelihood that normal samples are identified as anomalous by the model, as the model has a weaker frame of reference for normal behavior. The previous two categories (semi-supervised and supervised) make assumptions of their own however. Namely the class cluster assumption [13]. This is the assumption that sam-

ples that behave similarly will belong to the same class. The issue with this assumption lies in the nature of anomalies, that is, anomalies are defined as any behavior that diverges from what is considered normal. There must not necessarily be a cohesive behavior among all anomalies, that allows them to be clustered together. The data points within O_3 in Figure 2.1 could have well been identified as anomalous due to their similar behavior, but this is not the case for the points o_1 and o_2, who do not share behavior with any other points. For this reason, unsupervised anomaly detection is often the preferred method to identify anomalies, as it can be arduous and sometimes impossible to ensure that the anomalies labeled truly represent the behavior of all anomalies.

Some anomaly detection algorithms label samples directly as normal or anomalous in their output. These algorithms are then usually a form of binary classification. Many algorithms will output an anomaly score instead. This is usually a value that can be used to identify how likely a given sample is an anomaly. The analyst themselves has control over what is considered an anomaly by setting a threshold for the anomaly score. Every sample that receives a score above the threshold would be considered normal and everything below would be anomalous, in the case that a high score indicates normal behavior. For our choice of anomaly detection algorithms, they are all unsupervised algorithms. Unsupervised anomaly detection algorithms are, as previously stated, the most widely applicable algorithms. Furthermore, due to the inherently unpredictable nature of anomalies, unsupervised methods hold the most potential for practical application. Assuming they can perform at the same level or better than equivalent supervised or semi-supervised algorithms, unsupervised anomaly detection will likely become the standard in many domains, as labeling, which is expensive and can introduce human bias, can be avoided entirely. In our experiments all our algorithms output an anomaly score and we do not set a threshold, as this can be arbitrarily set.

2.3 Autoencoders

An Autoencoder is a type of artificial neural network used for unsupervised learning. What differentiates Autoencoders from the various other forms of artificial neural networks is their reliance on efficient reduced dimensional representations of data, which makes them fit for tasks relying on dimensionality reduction. Dimensionality reduction is the transformation of high-dimensional data into a representation with reduced dimensionality [74]. High-dimensional data can be images, video, or even speech signals used in speech processing. This reduced-dimensional representation should represent the intrinsic dimensionality of the data. For example, given a dataset that describes pairs of world heritage sites with their nation. One might have (Cologne Cathedral, Germany), (Great Wall of China, China), and (Yosemite National Park, USA), as a few examples. This particular data set will not truly be evenly randomly

spread through both dimensions however, pairings such as (Cologne Cathedral, China) or even (Statue of Liberty, Germany) are not representative of the properties of the data and thus never appear. The data set is two dimensional but the intrinsic dimension is lower, any combination of a world heritage site with any nation that isn't their own is not represented in the data. Dimensionality reduction can often be done as a preprocessing step before further analysis on a data set is made. Reduced dimensional representations can be much easier and efficient to work with than higher dimensional representations. Autoencoders actually seek to reduce the dimensionality of a given sample past it's intrinsic dimensionality to an even lower dimensional representation, as we will discuss later [35]. This makes them uniquely fit to replace tasks such as Principal Component Analysis, which is a linear dimensionality reduction technique, that can, due to it's linearity, only identify linear correlations between variables [36]. One of the main uses of Autoencoders is as a form of nonlinear Principal Component Analysis, that can identify more correlations due to it's nonlinearity, which allows it to be applied to more complex data where linear dimensionality reduction such as PCA fails.

Autoencoders once called "Autoassociative neural networks" are composed of three main components [35]. These components are now referred to as the encoder, decoder, and bottleneck/code, but were originally referred to as the mapping layer, demapping layer, and bottleneck. The figure below shows the architecture of an Autoencoder, which has remained relatively the same 3 decades later, with some aforementioned changes in the nomenclature.

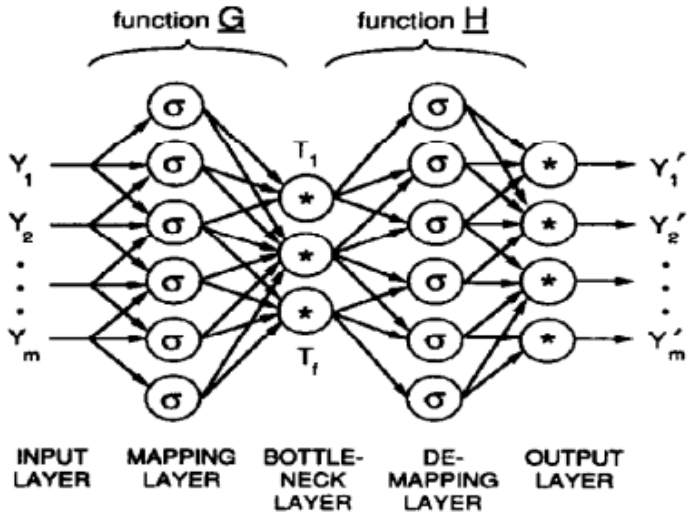

Figure 2.2: An Autoassociative neural network [35]. The σ symbol represents a dense node appended with a sigmoid activation function, whereas $*$ is either sigmoid or linear.

As previously stated an Autoencoder performs a form of dimensionality reduction on it's input samples, represented in figure 2.2 by $Y = \{Y_1, Y_2, ...Y_m\}$. The encoder tries to map these inputs belonging to Y from their feature space to a latent space representation T and this relationship can be represented by a function, namely [35] [36]:

$$T_i = G_i(Y), \quad i = 1, ..., f \tag{2.1}$$

This function G is a nonlinear vector function comprising of f many nonlinear functions. Each of the individual functions G_i is nonlinear due to their nonlinear activation functions. Figure 2.2 uses sigmoid activation functions, but rectified linear units [22], and other nonlinear activation functions can replace them. Then the output of the bottleneck layer are used as inputs for the second function, representing the decoder [35] [36]:

$$Y'_j = H_j(T), \quad j = 1, ..., m \tag{2.2}$$

Similarly this function is a nonlinear vector function of m individual nonlinear functions, which attempts to produce an approximation of the input Y based on the latent variables from the bottleneck T. This describes the goal of an Autoencoder, namely to approximate the identity function. However this is why the bottleneck should be smaller and have a lower dimensionality than the intrinsic dimensions of a given data set. If the bottleneck simply allowed the Autoencoder to map the input down to their intrinsic dimensions and recreated the input from these latent variables, then the Autoencoder would not learn anything. In this case it would simply be performing the identity function, which is not particularly useful, especially in the case of anomaly detection, where it is desirable for the Autoencoder to recreate normal samples well and anomalous samples poorly. For this reason, Autoencoders require a bottleneck with lower dimensionality, so that information is lost. Due to this required loss of information, Autoencoders must find efficient representations for samples, that maintain as much crucial information as possible. In this way an Autoencoder learns to mimic the identity function by learning efficient latent representations for input data [24]. These latent representations, or latent variables as some call them, are hidden variables that greatly influence the behavior of the data, although they are unobservable. To aid in this endeavor, the following loss is defined [35] [36]:

$$E = \sum_{p=1}^{n} \sum_{i=1}^{m} (Y_i - Y'_i)_p^2 \tag{2.3}$$

This loss function sums over the entire data set, taking the sum of squared errors between each input and output. In this way, the Autoencoder is encouraged to approximate the identity function by minimizing the difference between the input samples and the generated output. As with any other artificial neural network, the gradients of this loss function with respect to the

weights of the layers within the encoder, bottleneck, and decoder can be calculated through backpropagation and gradient descent can update the weights throughout the network.

Figure 2.2 depicts an early version of an Autoencoder with only 1 hidden layer per encoder and decoder, this is in a more modern context known as a shallow Autoencoder. A shallow architecture contains few hidden layers in comparison to deep architectures, which contain a multitude of hidden layers as their defining feature. Deep Autoencoders as well as deep networks in general benefit from a few advantages over shallow networks. Feedforward neural networks that have at least one hidden layer are so called Universal Approximators [30], meaning that they can approximate any function given that they have enough nodes within the hidden layer. However, greater depth, past a single hidden layer, can reduce the computational cost of representing certain functions, while also reducing the amount of training data that is required to accurately approximate certain functions [24]. Deep methods grant models the ability to learn representations of data that have multiple layers of abstraction, allowing them to represent complex data in a compact form [56]. Additionally they are fit to tasks dealing with hierarchical data, such as text or even images. Many shallow methods require extensive feature engineering in order to compete with deep methods in processing high-dimensional data, and as such are less applicable to these data sets. For this reason our algorithm will be a deep-based algorithm.

Autoencoders are applied for the task of anomaly detection due to their ability to learn efficient representations of data they are trained on. Hwang et al. describe anomaly detection as "a 2-class classification problem that discriminates limited normal patterns from infinite unspecified abnormal ones." [31]. When Autoencoders are applied for anomaly detection they are trained on training data that is assumed to be dominated by normal samples, and the Autoencoder maps these samples to a latent subspace, where it is assumed that the characteristics of anomalous samples deviate from normal ones [2][33]. This deviation in the latent subspace leads to the decoder reconstructing normal samples accurately and anomalous samples poorly. This is measured by a metric called the reconstruction error, which for a D-dimensional sample is [57] [14]:

$$Err(i) = \sqrt{\sum_{j=1}^{D} (x_j(i) - x_j(i)')^2} \tag{2.4}$$

Where similar to equation (2.3), $x_j(i)$ denotes the j-th variable for input sample $x(i)$ and $x_j(i)'$ denotes the j-th variable for reconstruction $x(i)'$. This reconstruction error can be used directly as an anomaly score, as it describes how different the generated sample is from the input.

2.3.1 Convolutional models

Convolutional models are an alternative to fully-connected models, which replace a fully-connected model's dense layers with convolutional layers. A convolutional layer is comprised of a number of filters or feature maps, and each filter has a convolutional kernel. This kernel contains the weights of the convolutional layer, whereas in a fully-connected network these weights describe the strength of the connections of each input node to the neural nodes of the following dense layer. In a fully-connected network each node of the prior layer is connected to every node in the following layer, creating a network of dense connections, with often thousands to millions of weights. Convolutional layers contain all their weights in a kernel, which is run over the entire input, performing the mathematical convolution operation in each step. When an input is fed into a convolutional layer, each filter will perform a different convolutional operation, as each filter has a different kernel. For each individual filter, however, their kernel is used across the entire input of the layer, which is called weight sharing. This is one of the distinct advantages of convolutional models, the number of free parameters that need to be optimized is lower than a fully connected model. For example, for a fully-connected model with 28x28 image as an input and 100 nodes in the first hidden layer, there are 78400 weights to be trained $(28 \cdot 28 \cdot 100)$. Whereas a convolutional model with a 28x28 image as input and a convolutional layer with a 5x5 kernel and 14 filters as the first hidden layer only has 350 weights $(5 \cdot 5 \cdot 14)$.

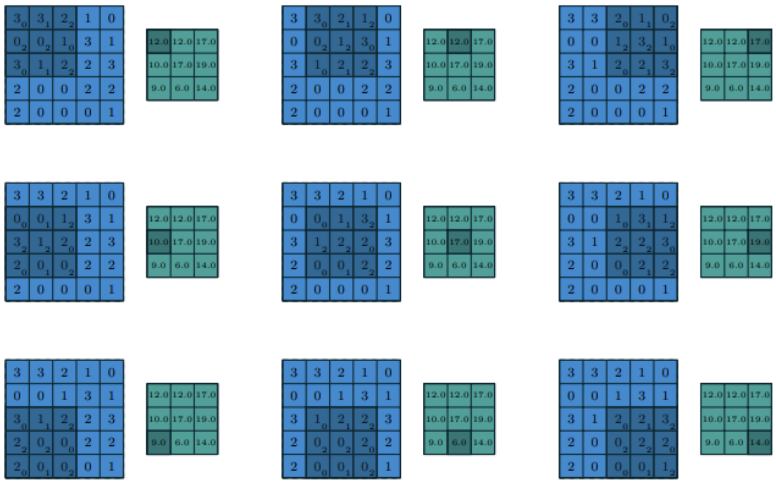

Figure 2.3: An example of a kernel (green) performing a convolution on an input feature map (blue) [19].

Weight sharing reduces the total number of weights being trained, reducing computational costs, allowing model to be trained on complex high-dimensional

data [37]. Additionally, a model having less parameters to train often translates to a faster training time [43] [39] [14]. Convolutional models are, for this reason, often implemented when the data is image-based. A challenge of constructing image-based models however is that the critical information needed to identify an image can be located anywhere in the image. A model must be shift invariant, so that if certain features appear shifted in location, the model still correctly identifies these features. A model must also be translation invariant and if possible distortion invariant for similar reasons. In order to make a fully-connected network translation invariant, shift invariant, or distortion invariant, there would need to be enough data to cover all possibilities, forcing the network to set similar weights in multiple places so that it can pick up on similar structures that could be located in different parts of an image [39]. This requires a much larger, possibly artificially augmented data set and much redundancy and inefficiency in the setting of the model's weights as nearly identical weight structures would be found throughout the mode. In comparison, convolutional layers are invariant to translations, shifts, and distortions that are not themselves crucial features [39].

Moreover, "Unlike in fully connected networks, where the value of each unit depends on the entire input to the network, a unit in convolutional networks only depends on a region of the input. This region in the input is the receptive field for that unit." [41]. What is referenced as a unit is the result of a single convolution operation and Luo et. al. refers to the fact that this result is dependent upon the region of the image covered by the kernel. When the stride of a convolutional layer is set lower than the size of that layer's kernel, then overlapping occurs, where the kernel will calculate convolutions over the same part of an image multiple times. This allows convolutions to extract local features (which encode information on where other features are in relation to each other) by restricting the receptive fields to be local but overlapping [39]. Fully-connected networks ignore the topology of the input, meaning that the input can be set in any order without altering the ultimate outcome of the training. One can imagine the receptive field to thus be unrestricted as all inputs map to every node of the first hidden layer. An additional tool that convolutional networks often employ the use of is subsampling layers, often pooling. Feature maps can still encode positional information of some particularly distinctive features found in images, which is undesirable. Thus pooling layers are employed to reduce the spatial resolution of the feature map [39]. This reduces the networks ability to rely on spatial information and the particular location of features in an image. These local receptive fields along with subsampling allow convolutional layers to be invariant to translations, shifts, and distortions, as it becomes less important where a feature is found in an image, but rather where that feature is in relation to other features. As such, rotating an image 90 degrees or shifting the input to the right by some pixel count, does not effect the outcome of the model's prediction. One can imagine a convolutional

network as identifying basic structures such as endpoints, corners, and edges in the initial convolutional layer, each filter identifying different simple structures. In these early layers the kernels may appear similar to Sobel operators for example [64]. These are $3x3$ kernels that are used often in the field of Computer Vision for edge detection in images, as their composition detects changes in pixel value either horizontally or vertically. Though there is no guarantee as to how exactly the kernels of the initial layers are composed, it can help to imagine them in a similar manner, detecting some combination of simple edges or endpoints. Then as more convolutional layers are appended, more complex and abstract structures are identified [39]. These structures theoretically should more and more begin to resemble crucial aspects of the image such as a part of a dog's face or the two loops in the number eight. From these more complex representations the neural network is then able to assign a likelihood that an image belongs to a certain class based on the aggregate of these features.

Due to the fact that we have chosen to use image-based data sets such as MNIST and CIFAR10, a convolutional Autoencoder is most fit to our task. Our first anomaly detection algorithm to augment will be a Deep Convolutional Autoencoder (DCAE).

2.4 DeepSVDD

Deep Support Vector Data Description (DeepSVDD) is an anomaly detection algorithm that uses a model with an explicit anomaly detection objective [56]. Most anomaly detection algorithms use models that are trained for another task, but are then augmented for anomaly detection, such as generative models. Generative models are models whose objective is to learn the distribution of the data they are given, with the goal of understanding the distribution well enough that they can generate samples from it. Examples of generative models are GANs and some Autoencoders such as Variational Autoencoders [53]. Even non-generative Autoencoders, such as those discussed in the previous section are trained with the objective of dimensionality reduction and are then adapted for anomaly detection. An issue with many of these models, such as Autoencoders is that the size of the bottleneck is a hyperparameter itself. As previously discussed a large bottleneck hinders learning by allowing the model to simply condense and reconstruct the input without any loss of information, but a bottleneck that's too small could lead to too much loss of information, hindering the model's ability to learn efficient representations of the data. Thus the compactness is a hyperparameter that must be optimized, but many of these methods, including our chosen DCAE are unsupervised, meaning it can be problematic to make assumptions about the distribution of the data. Many classical anomaly detection methods also fail when applied to higher dimensional data, as they were not built with scalability as a main concern. DeepSVDD seeks to offer an alternative that is scalable and requires no such assumptions about

the data to be made, other than the implicit assumption that all unsupervised anomaly detection algorithms have, which is that the number of normal samples outnumbers the number of anomalies.

DeepSVDD is based on two major methods, One-Class Classification [48] and Support Vector Data Description [70]. One-Class Classification, sometimes referred to as data description, is the method which has become the basis for many anomaly detection algorithms, namely training a model on normal data only, so that the model learns to identify if test data belongs to the distribution of the data it's trained on instead of trying to directly discriminate between normal and anomaly [27]. SVDD describes another method of data description, where the model tries to learn a hypersphere of minimal volume that encloses as much data as possible. This is similar to the objective of Support Vector Machines [18], which seek to form a hyperplane that separates data into different classes. DeepSVDD is also similar to other methods such as the algorithms created by Eskin et. al., which map data to a feature space and determine anomalies by identifying samples that have been mapped to sparse regions, i.e. regions distant from other clusters of data [20]. In the following subsections we will discuss how DeepSVDD functions and detail both the Soft-Boundary and One-Class variants. Then we will discuss interesting properties of DeepSVDD that must be considered when applying it as an anomaly detection algorithm.

2.4.1 Soft-Boundary and One-Class

There are two variants of DeepSVDD, the first of which being Soft-Boundary. DeepSVDD constructs a minimal enclosing hypersphere over the training data and uses the distance from the center of this sphere to determine which samples are anomalous in nature. Soft-Boundary allows some points to be mapped outside of the volume of the hypersphere, with the intention of labeling those within as normal and those without as anomalous. The Soft-Boundary DeepSVDD objective is defined as follows [56]:

$$\min_{R,W} R^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\max\{0, ||\phi(x_i;W) - c||^2 - R^2\} + \frac{\lambda}{2}\sum_{\ell=1}^{L}||W^\ell||_F^2 \tag{2.5}$$

So for some input space $X \subseteq \mathbb{R}^d$ and output space $F \subseteq \mathbb{R}^p$, we can define $\phi(\cdot;W) : X \to F$ to be a functional representation of a neural network with a set of weights $W = \{W^1, ..., W^L\}$ for the $L \in \mathbb{N}$ hidden layers of the network. As such for all $x \in X$, $\phi(x;W) \in F$ is the latent network representation of the input x. The radius R is defined to be greater than 0 and the center of the hypersphere c is given and is thus not a free parameter to be optimized. Given training data $D_n = \{x_1, ..., x_n\}$ with $D_n \subseteq X$, this objective seeks to minimize both the radius R and the weights of the network W [56]. The first term ensures that the enclosing hypersphere is of minimal volume, as the loss will want to minimize R^2.

The second term describes a penalty for all points in the latent space that lie outside of the hypersphere. $||\phi(x_i; W) - c||^2$ describes the distance of the latent representation to the center of the hypersphere and $||\phi(x_i; W) - c||^2 - R^2$ tells us whether that point is within the radius of the hypersphere or not. A positive value indicates it is outside while a negative value indicates it is enclosed by the hypersphere. For this reasons we have $\sum_{i=1}^{n} \max\{0, ||\phi(x_i; W) - c||^2 - R^2\}$ which takes the max between 0 and this value, so if a point lies outside the sphere, the function is penalized and otherwise nothing is added. The factor $\frac{1}{\nu n}$ describes how much the function should be penalized for points outside the hypersphere. The variable $\nu \in (0, 1]$ when large allows more points outside the hypersphere and less when it is set to a smaller value. The final term is a weight regularization term with the hyperparameter $\lambda > 0$. The norm $|| \cdot ||_F^2$ is the squared Frobenius norm [56]. The Euclidian norm for vectors is the L2 norm and the Euclidian norm for matrices is the Frobenius norm and is defined as the square root of the sum of the absolute squares of the elements of a given matrix.

Weight regularization seeks to encourage a model to have smaller weights. Neural networks minimize a loss function, so adding a term that sums up the weights, like our third term, forces the model to choose smaller weights in order to keep the loss minimal. The reason smaller weights are desirable is because larger weights can cause overfitting where the model adapts too strictly to the distribution of the training data, making it perform worse when new data is introduced. Larger weights can also cause sharp transitions in node functions and this causes sharp changes in output for small changes in input [55]. This can lead to an unstable and possibly overfitted network. For this reason DeepSVDD implements weight regularization and uses *lambda* to determine the strength of the regularization. The larger value lambda is set to, the higher the penalty for large weights is.

Due to the fact that the weights of the network and the radius have different scales that they operate on, using a singular learning rate while training the model may not be optimal. When training is underway, the weights are trained for some specified number of epochs as usual, while the radius is held constant and unchanging. After these epochs have elapsed the radius is updated using the weights of the most recent epoch and afterwards the weights continue to be trained while the radius is held constant again, etc. An anomaly score can be derived by calculating $||\phi(x_i; W) - c||^2 - R^2$, by subtracting the squared radius all negative scores are within the hypersphere and thus normal, while all positive values lie outside the hypersphere and can be considered anomalous. The radius is calculated via line search, which is an iterative approach that identifies a direction in which an objective function can be minimized, then using gradient descent the magnitude of the step can be determined, slowly approximating a local minimum.

Soft-Boundary is the more widely applicable of the two variants of DeepSVDD, as no assumptions of the data need to be made. In contrast, One-Class is a ver-

sion of DeepSVDD that is faster and more efficient than Soft-Boundary but only in the case that most of the data is normal. This version's objective function can be described as follows [56]:

$$\min_{W} \frac{1}{n} \sum_{i=1}^{n} ||\phi(x_i; W) - c||^2 + \frac{\lambda}{2} \sum_{\ell=1}^{L} ||W^{\ell}||_F^2 \tag{2.6}$$

One-Class has no radius that it is trying to minimize and thus all points lie within the hypersphere. This is described by the first term of the objective function, which simply uses the distance of the latent representation from the center as a penalty, so the model is encouraged to find a mapping that puts as many samples near the center as possible. By trying to minimize the volume of the hypersphere by minimizing the mean distance of all points to the center, anomalies can be identified as being those samples which are exceptionally far from the center. The distance of a point from the center of the hypersphere can thus be output and interpreted directly as an anomaly score for the purposes of anomaly detection. The second term is the same weight regularizer as in the Soft-Boundary objective function and is applied for the same reasons.

2.4.2 Properties of DeepSVDD

There are four important properties of the DeepSVDD algorithm that impact how an architecture intended for use with DeepSVDD should be adapted [56]. Three of these properties deal with hypersphere collapse, which is when the radius and thus the volume of the enclosing hypersphere becomes zero. This is to be avoided, and is considered a side effect of particularly ill-constructed models that violate the following properties.

The first property is that the center c of the DeepSVDD objective function cannot be a free parameter for optimization by the chosen optimizer of the model. This is because a trivial solution to the optimization objective can be found, where all weights are set to zero. Once the weights of a given neural network are all set to zero, then the network maps all inputs to the same constant output. If the center is a free parameter it can be set to the same value as this constant output, thus minimizing the objective function, but effectively causing hypersphere collapse as the radius will also be zero if all points are mapped onto the center. As such it is recommended to initialize the center intelligently, so that the network cannot reach this global minimum. Ruff et. al. recommend selecting the center by computing a forward pass and calculating the mean of the output of this pass and using this value as the center of the hypersphere. This is introduced as a relatively stable initialization for the center that does not require extensive computation to derive.

The second major property is that networks with a bias term have the potential to learn the trivial mapping of any input to the center. Similar to the

last trivial scenario, when the weights are all set to zero, then regardless of the input the output will be purely determined by the bias. The bias can thus allow the network to learn any constant mapping. This means that the bias can be optimized in such a way that every input maps to the center which causes hypersphere collapse.

The third property is that layers with bounded activation functions can induce a bias-like effect in subsequent layers when the inputs have at least one feature with the same sign across all inputs. Bounded activation functions, like sigmoids, can be saturated, meaning that they squeeze the range of all inputs. This is particularly important when inputs happen to have one feature k that is either positive or negative across all inputs, because every other feature can have their weights set to zero and the weights of k can be set arbitrarily large. The bounded activation thus allows the network to approximate it's bounds through setting the size of the weights for k, which creates a similar effect to a bias in the following layers. This as discussed in the second property, can lead to hypersphere collapse. This is why it is of the utmost importance that activation functions are either unbounded or only bounded by zero (which would have no effect on subsequent layers).

The fourth property, named the ν-property allows one to incorporate prior knowledge of the distribution in the training of the model[61]. This property more specifically allows one to incorporate prior beliefs about the proportion of outliers present in a data set. The term ν in the Soft-Boundary and One-Class objective functions is an upper bound on the fraction of outliers and a lower bound on the fraction of points on or outside the boundary[61]. In this way ν can be set intelligently, when prior knowledge exists and otherwise can be chosen arbitrarily and optimized.

The first three properties create restrictions for the creation of a DeepSVDD model. The center must be initialized and cannot be a simple free parameter for the optimizer to use in minimizing the loss function. As such an intelligent initialization of the center is imperative as otherwise the latent representations of the data may lay too far from the center whether they are normal samples or not, causing the hypersphere to cover only a small fraction of the data. The second and third parameter do not allow us to use biases in the layers or bounded activations such as sigmoid or the hyperbolic tangent. This leaves us with softplus and the multiple variations of ReLU as our options for activations. Finally we can use ν to include prior knowledge about our data, however in our experiments we will defer to Ruff et. al. for the choice of ν as a hyperparameter.

2.5 GANs

A Generative Adversarial Network (GAN) is a model containing two separate models, the generator and discriminator, which it pits against each other in a zero-sum minmax game. The goal, as with all two-player zero-sum games is to reach the Nash Equilibrium. This is defined as the point at which no player can increase their expected payoff by changing their own strategy [50]. The players of this game are the Generator G which tries to understand and replicate the data distribution of the input it's given and the Discriminator D which tries to decided whether it's given input has come from the training data or the Generator. The Discriminator thus wants to accurately classify it's input as either real or fake and the Generator wants to maximize the probability that the Discriminator makes a mistake. The Nash Equilibrium is then equivalent to G capturing the underlying distribution of the training data and D reaching an accuracy of 50% [25]. The goal of this process is to train a generative model, which in this case is the Generator, to be able to generate new samples from the same distribution as the training data. The Generator takes random latent noise as its input and constructs images from the latent representation. These new generated samples should ideally be indistinguishable from the training samples. When GANs were first introduced, both the Generator and the Discriminator were multilayer perceptrons, which allowed them to be trained using backpropagation[25]. The general structure of a GAN can be visualized below in Figure 2.4.

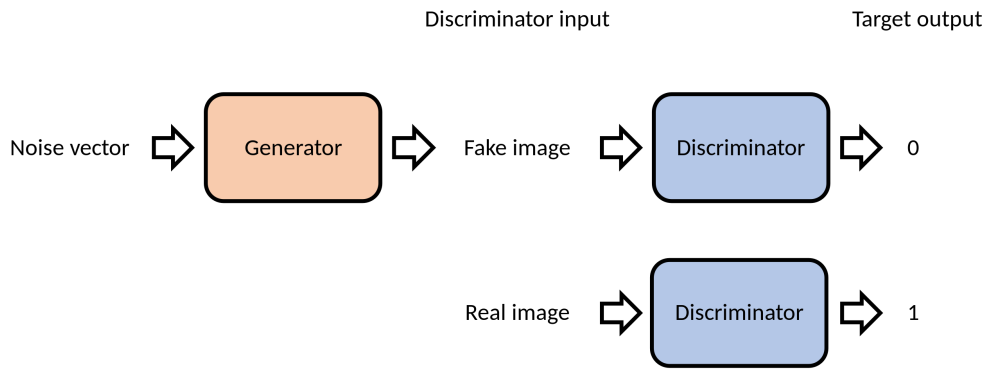

Figure 2.4: A Generative Adversarial Network [69]

The Generator's distribution over the data x is defined as p_g. A prior probability distribution is defined over the set of input noise for the Generator as $p_z(z)$. The Generator thus builds a mapping to the space of the training data with $G(z, \theta_g)$, where G is a function that represents the neural network for the Generator and θ_g are the parameters, i.e. the weights of this net. Similarly $D(x, \theta_d)$ can be defined as the functional representation of the Discriminator's

network, but instead of mapping to the data space it outputs a scalar value indicating the probability that the input x came from the data distribution p_{data} instead of the Generator's distribution p_g. These two models play the following minmax game [25]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{2.7}$$

$V(D,G)$ is defined as the value function of this game. The Discriminator D wants to maximize the probability of accurately labeling training examples, as seen in the first term, as well as generated samples, as seen in the second. Therefore, given a Generator G, the Discriminator D wants to maximize the value function $V(G,D)$. This translates in implementation to ascending the stochastic gradient of $\frac{1}{m}\sum_{i=1}^{m}[\log D(x_i) + \log(1 - D(G(z_i)))]$ for a batch of m training samples and noise samples. However, the Generator wants to minimize the second term $\log(1 - D(G(z)))$. Which is implemented by descending the stochastic gradient of $\frac{1}{m}\sum_{i=1}^{m}[\log(1 - D(G(z_i)))]$ for m noise samples [25]. It is proven by Goodfellow et. al. that the global optimum of this two-player minmax game is found when $p_g = p_{data}$ [25]. This describes the previously discussed goal of the GAN, namely for the Generator to approximate the underlying distribution of the training data.

The Discriminator must be "synchronized" well with the Generator during training. This means that the Discriminator and Generator should be updated together often [25]. If the Generator is trained too long with updating the Discriminator, then it is possible that the Generator will learn to collapse many latent values to the same data space sample. This is as well as the hypersphere collapse that DeepSVDD suffers from are called mode collapse and will be discussed further in Section 5 [71].

Radford et. al. introduced the Deep Convolutional Generative Adversarial Network (DCGAN) [54], which was not the first convolutional GAN, but set guidelines for successfully creating them. The first guideline is based off of the findings of Springenberg et. al., who describe what is called the "All Convolutional Net" [67]. This is a CNN that does not use any pooling layers. In the previous Section 2.3.1, we discussed how pooling layers are often integral to the successful application of CNNs because they do not allow the network to rely on spatial information to as much of an extent as without them. This aids CNNs in identifying common structures in images regardless of their location. Springenberg et. al. instead use strided convolutional layers that effectively perform a similar spatial decimation as pooling functions like max pooling. The main difference being that the network can learn the weights in the kernels of these strided convolutions, allowing the network to learn a possibly more optimal downsampling. The first guideline is thus to replace spatial pooling layers with strided convolutions. Following this is the second guideline, which is to remove fully-connected hidden layers, leaving the model almost exclusively consisting of convolutional layers. As such the the Generator G

comes to resemble a convolutional decoder, much like those found in DCAEs, while the Discriminator D resembles a CNN. The third and final guideline is to use Batch Normalization, which was found to be critical in preventing mode collapse [54]. In the following section we will detail what Batch Normalization is and how it functions.

2.5.1 Batch Normalization

Batch Normalization introduced by Szegedy et. al. is a way of making normalization a part of a model's architecture [32]. Before Batch Normalization, most normalization was done as a preprocessing step before data was fed into a machine learning model. One of the main reasons for using normalization, is to prevent particularly large features from dominating the learning process. A simple example that illustrates the importance of normalization would be to assume we are trying to build a model that predicts the price of new real estate. We have four features namely: the age of the building, the number of stories, the amount of bathrooms, and the square meter area of the plot of land. The fourth feature is likely to dominate the learning procedure as the magnitude of the values could range in the hundreds to the thousands, whereas the age likely ranges only from a few years to decades and the number of bathrooms and stories are likely to be some single digit. The model is thus likely to ignore or give much less weight to these other features purely because their scale is so much smaller. It may indeed be the case that all of these features are less important than the area for predicting the price of real estate, but we would ideally want the model to learn this and for it to consider at the beginning that all these features are more or less equal.

The change in a network's input distribution is defined as covariate shift [63], but Szegedy et. al. define the term internal covariate shift to indicate a change in the distribution of a network's activations [32], i.e. the change in distribution of a layer or sub-network. Covariate shift describes the phenomenon, where the distribution of the training data does not match up with the distribution of test data or real world data. Whereas internal covariate shift is the same idea extrapolated to a more granular level of the sub-network within a network. The training and hyperparameter tuning of deep models can be complex as the inputs of any given layer are dependent on and determined by the parameters of all previous layers. As such, small changes in a layer's distribution can be intensified throughout a network, leading to an increase in error rate [32]. A simple two-layer network can illustrate this problem. The first layer takes the input fed into the network and it's output is then fed into the second layer, which itself generates the output of the network as a whole. From the perspective of the second layer, the output of the first layer is it's input. The gradient descent step that updates the weights of this layer also treat the first layer's output as if it was simply the input of a standalone network. Thus for the same

reason one would want to sample the training and test data from the same distribution, one would also want the distributions of the layers of a model to remain similar, this is the idea behind the term internal covariate shift.

In order to remedy internal covariate shift, Batch Normalization can be applied after each layer, normalizing features before they are passed to the next layer. Batch Normalization was originally intended to be used with optimizers like mini-batch stochastic gradient descent, that update parameters based on mini-batches. Batches are used for the sake of computational efficiency as calculating the average of the gradients over all training samples is expensive, instead the average of the gradients over the other samples in the batch are calculated. This does lead to only an estimate of the gradient, but still functions in principal. Batch Normalization calculates the mean for each dimension of the layer input over the entire batch and then using this mean is able to calculate the variance for each dimension over the batch. For the dimension k of a d-dimensional layer input, the normalization is calculated as follows [32]:

$$\hat{x}^k = \frac{x^k - \mathbb{E}[x^k]}{\sqrt{Var[x^k]}} \tag{2.8}$$

Each normalized feature $\hat{x}^k$ thus has a mean of 0 and a variance of 1. However, Szegedy et. al. cleverly identify that if the input of a layer is simply normalized, that it may change what said layer can represent. They give the example of the sigmoid, where normalizing the inputs may lead to them becoming small enough that they are mapped to the portion of the sigmoid function where it is almost linear (within $[-1, 1]$) [32]. Due to this, after normalizing each feature, Batch Normalization scales and shifts the normalized value [32]:

$$y^k = \gamma^k \hat{x}^k + \beta^k \tag{2.9}$$

The parameter γ scales the normalized input and β shifts it. These two parameters are free parameters learnable by the model. If it were optimal for the normalization to be reversed and the identity transform used instead, then the original input is recoverable by the network learning $\gamma^k = \sqrt{Var[x^k]}$ and $\beta^k = \mathbb{E}[x^k]$. Batch Normalization can be applied to both fully-connected and convolutional networks. For convolutional neural networks, the parameters γ and β exist per feature map instead of per activation and the mean and variance are also calculated per feature map. This is due to the weight-sharing property of convolutional neural networks, where a single kernel is used for an entire feature map, it follows that the normalization should be calculated in a similar manner.

Batch Normalization, on top of combating internal covariate shift, also regularizes the model somewhat as individual training samples are viewed within the context of their mini-batch which generalizes the model and aids in reducing overfitting. For these reasons, Batch Normalization is a strong tool that can

lead to more consistent training. In addition to being an integral part of DCGAN architecture, Batch Normalization is used in our DCAE and DeepSVDD models as well.

2.5.2 AnoGAN

The AnoGAN is type of GAN used for the purposes of unsupervised anomaly detection. It was originally developed for the medical domain, as an alternative to the supervised anomaly detection algorithms in more common use, that required strenuous effort to label and annotate. AnoGAN was made as an unsupervised alternative that would identify disease markers from medical imaging [59]. It functions by first training a DCGAN on only normal samples, this GAN then learns a mapping from the latent space to the data space as previously discussed. What AnoGAN wishes to accomplish is to find the reverse mapping, that is a mapping from the data space to a point in latent space. Given a Generator that has presumably learned to approximate the distribution of the training data, then one can for a given image x try to find a z in the latent space that generates an image most similar to x. For all normal samples, the error between x and the generated image should be low, while for anomalous images the error will be larger, as the generator will create an image from the distribution of normal samples not anomalous ones [44].

The process by which this reverse mapping is found is the core of the AnoGAN model. At first random latent noise z_1 is generated and fed into the Generator of the trained DCGAN. From this we receive the generated image $G(z_1)$, which we can use for a loss function that will specify gradient updates for z_1, leading to a new latent point z_2 resulting from the gradient update. This is repeated for some predefined number of backpropagation steps and after the final gradient update, the latent point z_Γ is found. This point should be the closest point that could be found that leads to the most similar generated image [59]. The loss function that is used to determine the gradient updates for the latent points is comprised of two elements.

The first is the residual loss, which measures how similar the tested image is from the generated image [59]:

$$\mathcal{L}_R(z_\gamma) = \sum |x - G(z_\gamma)| \tag{2.10}$$

For this loss, if the Generator has perfectly captured the underlying distribution of the training data and the reverse mapping is also perfect, then x and $G(z_\gamma)$ should be identical which should lead to a loss of 0. The second element is the discrimination loss, which serves the purpose of ensuring that the generated image is from the same distribution as the training data, i.e. that it is representative of normal behavior. Instead of simply defining the discrimination loss as the output of the trained discriminator given the generated image $G(z_\gamma)$,

Schlegl et. al. define a loss based on feature matching [59]. Feature matching is the concept of identifying similar features in images that may be shifted, transformed, or be from a different perspective. This constrains the Generator to produce images that have a similar intermediate representation to those of the training samples. The discrimination loss is defined as follows [59]:

$$\mathcal{L}_D(z_\gamma) = \sum |f(x) - f(G(z_\gamma))| \tag{2.11}$$

Where the function f describes the output of some intermediate layer within the architecture of the Discriminator. This forces the optimization of the latent variable z to depend not on the decision of the Discriminator, but instead the internal feature representation of normal behavior. Salimans et. al. argue that overtraining on the Discriminator can cause instability in GANs and for this reason suggest feature matching as an apt alternative [58]. This works because, through training, Discriminators will need to learn what differentiates training data from generated data, as such their intermediate representations should capture some of the most important features that describe the training data. Being that the AnoGAN is trained only on normal samples, this feature representation should specify which aspects of normal data are of foremost importance. The total loss is then [59]:

$$\mathcal{L}(z_\gamma) = (1 - \lambda) \cdot \mathcal{L}_R(z_\gamma) + \lambda \cdot \mathcal{L}_D(z_\gamma) \tag{2.12}$$

This loss is used in every iteration of the reverse mapping process to identify the best latent value z that creates an image $G(z)$ that is closest to test image x. The number of total backpropagation iterations is to be defined before the latent mapping begins, but Schlegl et. al. choose 500 iterations [59]. This is 500 iterations per individual test sample, meaning that the reverse mapping process can often be lengthy in terms of computation time. As with all other anomaly detection algorithms we have discussed, an anomaly score is defined, which is simply the total loss (2.12) for the last iteration of the latent mapping.

With this we have a Generative Adversarial Network for unsupervised anomaly detection.

3 Robustness and Lipschitz

Here we would like to some of the particular details pertaining to Robustness and Lipschitz continuity. First we will introduce adversarial examples and discuss the danger that they pose to non-robust machine learning models, as well as the difficulties of defending against them. Then we will discuss robustness, why it is necessary as well as its costs. Finally we will discuss the most successful defensive strategies for adversarial attacks, bringing us to Lipschitz continuity as a general defensive measure as well as a method for achieving provable robustness.

3.1 Adversarial Examples

Adversarial examples describe a curious phenomenon, where a small intentional change to an input sample can cause said sample to become misclassified by a neural network [68]. These adversarial examples are often nearly indistinguishable from non perturbed images. It would be desirable for neural networks to be invariant to the same small changes as humans, but this discovery suggests a fundamental difference in how we and neural networks perceive data. As such adversarial examples leverage this gap in perception between humans and machine learning models, creating samples that humans would find as orthodox, but neural networks view completely differently. In the last decade following this finding, so called adversarial attacks have been the subject of much study [26] [52]. What has been found is that the overwhelming majority of standard machine learning models are weak to these adversarial attacks and can be relatively easily exploited. Consider a network represented by the function f and an input sample x with the label t, the goal of an adversarial attack is to craft a perturbation δ so that $f(x + \delta) \neq t$. This perturbation should notably be as small as possible, while still causing a misclassification. Otherwise one could simply "drown out" the original sample with noise, easily causing a misclassification. The goal of adversarial attacks is not simply to cause a sample to be misclassified but for said sample to be near indistinguishable from other normal unaltered samples. Some consider more powerful attacks, such as source-target misclassifications [52], which specify the source class to perturb as well as the target class, that one would like the input sample to be misclassified as. The most basic attack is the untargeted attack, which simply aims to misclassify inputs, with no explicit target.

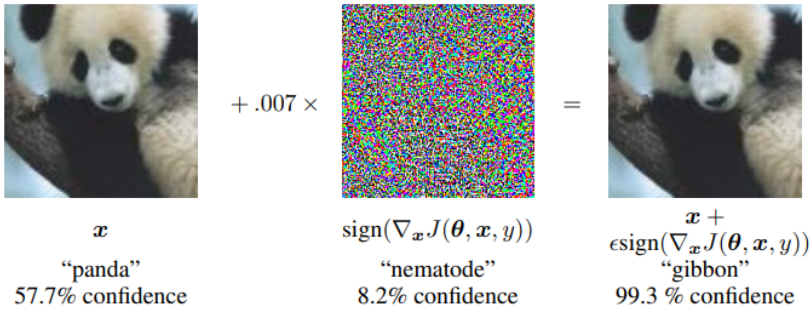

Figure 3.1: Example of an adversarial example that through miniscule perturbations causes an image of a panda to be misclassified as a gibbon [26].

The size of the perturbation is often measured with a mathematical norm, but an adversarial attack is not necessarily constrained to only L_p norm ball perturbations. Adversarial attacks can be manifested in many ways, such as through translations of pixels or even rotations, these attacks are however less studied. Even among the attacks that do perform L_p norm ball perturbations, there are different ways that these perturbations are applied. The Fast Gradient Sign Method [26] and Projected Gradient Descent are two different attack methods that both use L_p norm ball perturbations by adding some magnitude of perturbations opposite to the direction of gradient descent. These two attacks are white-box attacks that utilize internal information of a given model, namely its gradients, to successfully cause misclassifications. However there are also black-box attacks that require no information of a model's internal workings and parameters in order to perform adversarial attacks [51].

Additionally adversarial examples are, as alluded to in our introduction, not simply a digital phenomenon. They can also be a threat in the physical world, by physically altering objects to fool sensors and cameras [38]. This makes their threat more all encompassing and relevant for study. In recent years, various defenses against adversarial attacks have been designed. Many proclaimed defenses against adversarial attacks have then been proven to fail when another attack method is used or when aspects of the experiment are changed [11][5][10]. This is because, often, the attack success rate and the size of adversarial perturbation themselves are used as robustness metrics, and these are specific to a given adversarial attack. So when later another attack is made against these "seemingly" robust models, they are often successful and these models no longer appear robust [77]. In summary, adversarial attacks are varied and difficult to predict. What may defend against one attack may fail to defend against another and as new defenses against these attacks are being developed, the attacks themselves are further being improved upon in order to circumvent these defenses.

3.2 Robustness

Robustness is a metric and property of a model that describes said model's ability to maintain performance when dealing with uncertainty. This uncertainty can manifest itself through natural occurring anomalies, as well as maliciously crafted adversarial examples. Adversarial robustness is a more specific term meant to refer to the resistance of a model to adversarial attacks. Building robust models is thus critical in many domains and application scenarios. There does exist, however, a few trade-offs to building robust models that we would like to detail.

The first major trade-off is that compared to normal networks that simply aim to perform their default objective to the highest accuracy possible by minimizing their loss function, robust networks require more expensive training. This is because, in order to augment a normal network into a robust one, complexity in the form of extra constraints, parameters, training data, etc. must be added. The robustness of a model can be ensured through a myriad of ways, but one of the simplest and most obvious is a larger breadth of training data. Data that is naturally more varied will give a model a certain amount of robustness to uncertainties and some anomalies. Some models, like ours, obtain robustness by constraining the network, which requires normalization, matrix-based algorithms, and generally more computations at training time.

It is also very likely that the amount of data required to make normal generalizations is much lower than the amount necessary to make adversarially robust generalizations [60]. Intuitively, if one wanted a model to make adversarially robust generalizations, it must be shown a greater variety and distribution of data, possibly even including adversarial examples themselves. The adversarially robust model in simple terms needs to have a much stronger understanding of its data than a model that would be more vulnerable to adversarial attacks. Additionally the features learned by standard models are quite different from those learned by adversarially robust models [72]. This implies that even with infinite data, normal models may not become adversarially robust and instead specific tools and models must be developed to provide robustness.

There also exists a conflict between standard accuracy and robust accuracy, in that there has been proven to be a trade-off between accuracy and robustness [72]. This manifests due to the aforementioned difference in learned features. The data of a data set may have only have a few features that are strongly or moderately correlated to their labels, but many that are weakly correlated. It can occur that the aggregate of these weak features, seen holistically, are actually a much stronger indicator of the label than any individual other feature. Therefore a standard model will rely on these weakly correlated features to achieve a high classification accuracy. The problem with this, is that these weakly correlated features are especially vulnerable to adversarial attacks. If these features all together have such a strong correlation with the label of the

data, then an adversarial attack need only modify each of these weakly correlated features by a minute amount in a direction so as to lead to misclassification. Adversarially robust models, therefore, must abstain from relying to heavily on these weakly correlated features and need to instead use the more moderately or strongly correlated individual features, which can lead to a loss in accuracy. This describes the difference between what are referred to as robust features and nonrobust features [72]. It should be noted, that this situation cannot be remedied by acquiring and training on more data, as this phenomenon is inherent to the data distribution of the data set. As such, some data sets will exist, for which accuracy and robust accuracy are at odds.

These trade-offs help us understand the reasons why robustness is the topic of so much research. Models that are to be deployed and used in the real world with real data necessitate robustness, otherwise they are vulnerable to cyberattacks and erroneous data. At the same time, robust models have a few inherent costs to train, which creates tension between standard models which may achieve higher accuracy and performance but are otherwise ”brittle” and more ”durable” robust models that are generally less accurate and more expensive to train. The fact that robustness is not obtained ”for free” is the driving force behind much of its research. Thus much of the research focused on robustness seeks to find new ways of providing greater robustness, while maintaining accuracy, and providing scalability and computational efficiency.

Robust models do have some additional advantages however. Adversarially robust models are more interpretable compared to standard models [76] [72] [21]. This is because humans are generally invariant to adversarial examples, we do not perceive the minute perturbations that cause misclassification in other machine learning models. A model that is also invariant to these changes aligns better with human perception and thus it’s decisions become more interpretable to us. This is exemplified by the fact that the saliency maps of robust models are often more visually similar to their input images than non-robust models [21]. A saliency map highlights areas of specific interest, otherwise known as the discriminative features, of input images, showing us what a model deems as important for classification.

As alluded to before, robust classifiers must learn to use robust individually strongly correlated features to classify data. This reliance on robust features often translates into more interpretable saliency maps, as these features are often the more distinct features that we as humans perceive as particularly discriminative. Thus the structures present in these maps more closely resembles the overarching structures present in the input.

Up until now there have been three successful defense strategies for combating adversarial examples [62]. Agnostic defenses are defined as those defenses which are independent of the model. These defenses change the input or output of the model directly, but otherwise leave the model untouched. One example of this is Gaussian Random Smoothing [17], which adds random Gaussian

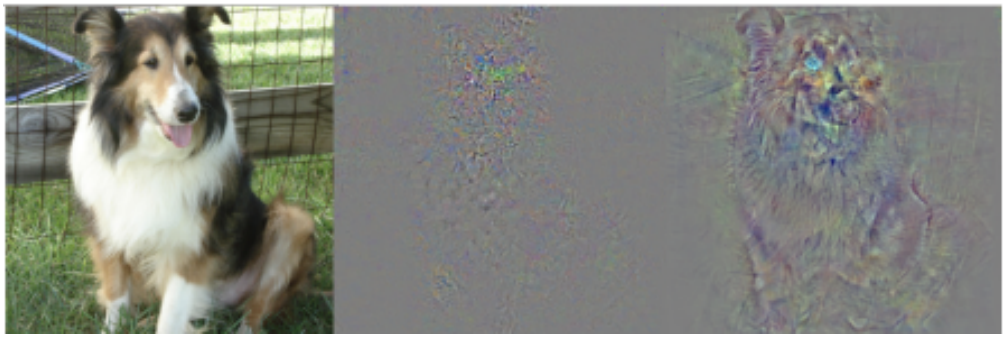

Figure 3.2: The input image (left) is compared with the saliency map of a non-robust model (center) and a robust model (right)[21]

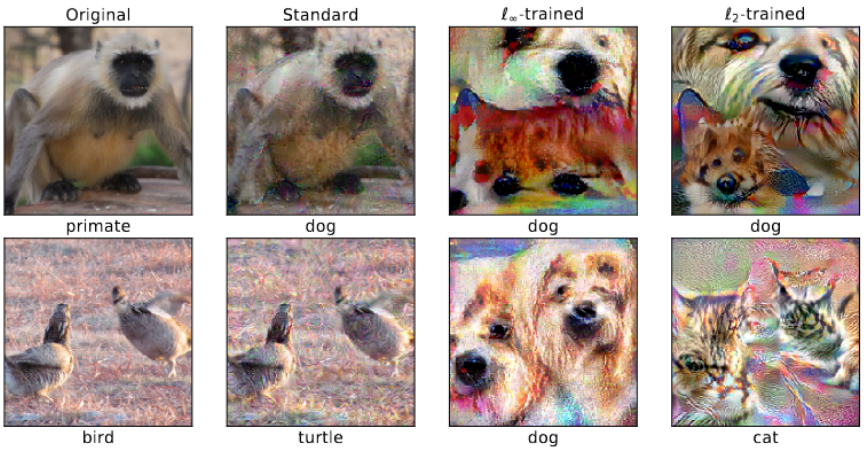

Figure 3.3: Examples of adversarial examples for a normal model and two adversarially trained models [72]

noise to the input samples, effectively ”drowning out” smaller perturbations like those caused by adversarial attacks. These strategies are some of the most varied in design and so are difficult to make overarching generalizations about. The second category of strategy is those that rely on saddle point optimizations [42], otherwise referred to as adversarial training. A saddle point is the result of a minimization along one axis and a maximization along an orthogonal axis, i.e. it is the same as the minmax game described in Section 2.5 when discussing the structure of GANs. The saddle point optimization problem is that a model will try to minimize it's loss by tweaking it's free parameters, i.e. it's weights, while an adversarial attack will try to maximize the loss by choosing a fitting perturbation. This formulation of the optimization problem as an outer minimization and an inner maximization allows Madry et. al. to configure the parameters of the model to minimize the effect that the worst-case

perturbation (the optimal perturbation for the inner maximization) can cause. This strategy, however has a major weakness, which is that it is vulnerable to unseen/unknown adversarial attacks, as it can only train to defend against those attacks that it knows [77]. So, strictly viewed, adversarial training only definitively offers robustness to adversarial perturbations in the training data. It must be assumed that the adversarial examples present in the training data are representative of those that could be found in the test data for an adversarially trained model to also be insensitive to perturbations in test data [80]. The final major strategy focuses on Lipschitz continuity, which we will discuss in the next section.

3.3 Lipschitz Property

The Lipschitz property limits the relation between input perturbation, whether from an adversarial attack or otherwise, and output variation. How much it limits this is based on the Lipschitz constant, which defines how much output variation is allowed for a given input perturbation. This means that for small changes in the input of a function, the output of said function should similarly be limited.

$$\forall x, y \in \mathbb{R}^n, ||f(x) - f(y)||_2 \leq L||x - y||_2 \tag{3.1}$$

For a differentiable function $f : \mathbb{R}^n \to \mathbb{R}^m$. The smallest L for which this equation holds true is the Lipschitz constant of this function. Here we can also differentiate between globally Lipschitz which is represented by the equation above and locally Lipschitz, which is defined as any function whose restriction to any neighborhood around a point is Lipschitz [77]. Essentially a function is locally Lipschitz if x and y can only be drawn from a compact set S, whereas a globally Lipschitz function is defined for all compact sets. For locally Lipschitz functions, the Lipschitz constant can be derived rather simply, assuming it is real-valued. Its Lipschitz constant is then the maximum norm of its gradient $\sup_x ||\nabla f(x)||_2$[77].

For these definitions we have defined the Lipschitz property under the L_2 norm, but it can be defined under differing norms. Here we would like to quickly summarize a few of the most common norms. Let x be a vector and A be a matrix. The vector 2-norm is $\sqrt{\sum_i x_i^2}$ while the matrix 2-norm (also referred to as the spectral norm) is the max singular value of A. The vector 1-norm is $\sum_i |x_i|$ which is the max absolute column sum for matrices. Finally the ∞-norm is defined as $\max_i |x_i|$ for vectors and the max absolute row sum for matrices. Of these, the L_2 (2-norm) and L_∞ (∞-norm) are the most common, when discussing Lipschitz continuity or adversarial perturbations.

A model's weakness to adversarial examples is often due to the Lipschitz constant of the neural network, which when unconstrained can become very

high [62]. This is exemplified by the explicit objective of an adversarial attack, which is to find the smallest perturbation possible that causes misclassification by the classifier. A Lipschitz constraint can thus be used to combat adversarial attacks, because the output variation will be bounded by the change in input. Theoretically this should mean that a much larger perturbation must be applied to cause misclassification, which increases the visibility of the attack.

This intuition about Lipschitz continuity holds true as bounding the Lipschitz constant gives certifiable adversarial robustness [29] [49] [73]. Given a K-Lipschitz model f that outputs logits and an input x with label t, we define the margin of x as [62] [49]:

$$M_f(x) = \max(0, f_t(x) - \max_{i \neq t}(f_i(x))) \tag{3.2}$$

This represents the distance of the output of x from the next most likely output class. Thus if $2K\epsilon < M_f(x)$ the input x will remain correctly classified under any perturbation Δx with $||\Delta x||_p < \epsilon$. One can similarly imagine each input sample as having an epsilon-ball in a metric space, where within this ball all inputs are classified the same [79]. In order for these inputs to be misclassified, their perturbation must be greater than the radius of the ball. In this way, Lipschitz continuity as a defense is theoretically attack-agnostic, as it does not matter how the perturbation is applied only its size is of concern. Furthermore, Sokolić et. al. show that a Lipschitz constrained network performs better on new unseen data than equivalent models with no constraint [65].

Now we understand what could potentially be gained from implementing a Lipschitz continuous architecture, but there is a major constraint. Estimating the Lipschitz constant of even a 2-layer Multilayer Perceptron is NP-Hard [75]. This means that it is not computationally efficient to acquire an exact Lipschitz constant for a neural network and therefore upper-bounds on the Lipschitz constant are a more realistic and useful metric. If we define $L(f)$ to be the Lipschitz constant of a Lipschitz continuous function f and similarly $L(g)$ to be the Lipschitz constant of a Lipschitz continuous function g, then:

$$||g(f(y)) - g(f(x))||_2 \leq L(g)||f(y) - f(x)||_2 \leq L(g)L(f)||y - x||_2 \tag{3.3}$$

This means that the Lipschitz constant of a composition of Lipschitz continuous functions is upperbounded by the product of their individual Lipschitz constants. Due to the fact that neural networks are simply a composition of linear and nonlinear functions, making each individual layer in a network adhere to a Lipschitz constant is the best way to ensure the whole network is Lipschitz continuous. This can be illustrated by the following equation which assumes f is a function representing a Multilayer Perceptron for simplicity's sake [62]:

$$L(n) \leq L(\phi_k) * L(W_k) * L(\phi_{k-1}) * L(W_{k-1}) * ... * L(\phi_1) * L(W_1 \cdot x) \tag{3.4}$$

Here $L(n)$ represents the global Lipschitz constant for the function n which represents the neural network. This Lipschitz constant is upperbounded by the product of the Lipschitz constants of linear and nonlinear functions composing this network. The activation functions ϕ of each layer are the nonlinear component, while the weights W are the linear components. This means that if every layer is 1-Lipschitz that the model is at most 1-Lipschitz. With this understanding it is rather simple to conceive of how a Lipschitz network could be composed, but how does one ensure that each layer maintains a certain Lipschitz constant?

First of all, certain activation functions such as ReLU and Sigmoid are 1-Lipschitz naturally and require no alterations to fit into a Lipschitz continuous model. The main issue is constraining the Lipschitz constant of the individual layers, i.e. their weights. The Wasserstein GAN is a GAN that requires Lipschitz continuity. The authors of the paper decided to limit the Lipschitz constant of each layer via weight clipping, that they themselves admit is a rather crude method [4]. A more elegant method of constraining the weights has been introduced by means of the spectral norm (matrix L_2 norm). The spectral norm σ of a matrix A is defined as [80]:

$$\sigma(A) = \max_{\xi \in \mathbb{R}^n,\ \xi \neq 0} \frac{||A\xi||_2}{||\xi||_2} \tag{3.5}$$

It can be computed using the power iteration algorithm [23], which finds the absolute value of the largest singular value in a matrix. The Spectral norm of the weights is equal to the upper bound of the Lipschitz constant of a dense layer [16]. Remember that the Lipschitz constant of a locally Lipschitz function is $\sup_x ||\nabla f(x)||_2$. From this follows $\nabla f(x) = \nabla W x = W \rightarrow \sup_x ||\nabla f(x)||_2 = ||W||_2 = \sigma(W)$. By dividing the weights by their respective spectral norm we can normalize the weights to ensure that they are 1-Lipschitz. For convolutional layers, normalizing by the spectral norm is not enough to accurately limit the upper bound of a layer's Lipschitz constant. For this specific scenario an additional constant Λ is introduced and the weights of the convolutional layer, i.e. the kernel is divided by the spectral norm of the kernel multiplied with this constant Λ. For convolutional layers with no stride, this constant can be derived simply as the kernel size, while for layers with stride Λ is the ceiling of the kernel divided by the stride. Serrurier et. al. actually invent a more complex formula for deriving the constant Λ that leads to a tighter estimation of the Lipschitz constant [62], but the basic concept remains the same. Spectral normalization is applied in the forward step before backpropagation.

Now we have an understanding of how one could ensure a global Lipschitz constant for a network, and we could theoretically create a network that is Lipschitz continuous. The issue that arises, however, is the low expressiveness of networks with low Lipschitz constants, such as a constant of 1. Expressiveness is a term that describes the breadth of functions that a neural network can approximate, having a low expressiveness translates to an inability to understand

the complexities of the data it is trained on. This lack of expressiveness stems from a phenomenon called Gradient Norm Attenuation [3] [40]. Assume, for example, $y = f(x)$ was 1-Lipschitz and $\mathcal{L}$ a loss function. The the norm of the gradient after backpropagating through f is [40]:

$$||\nabla_x \mathcal{L}||_2 = ||(\nabla_y \mathcal{L})(\nabla_x f)||_2 \leq ||\nabla_y \mathcal{L}||_2 ||\nabla_x f||_2 \leq L(f)||\nabla_y \mathcal{L}||_2 \leq ||\nabla_y \mathcal{L}||_2 \tag{3.6}$$

Thus the norm of the gradient after backpropagation is upperbounded by the norm of the gradient before the backpropagation step. This means that as the network trains, the gradients will likely become smaller, i.e. attenuate. As the gradients become smaller, it becomes more and more difficult for the model to adjust it's weights to acclimate to data that it is presented, thus leading to the aforementioned reduction in expressiveness. The solution to this is for each layer of the network to be gradient norm preserving. A network n with weight matrix W is gradient norm preserving if and only if [40]:

$$||W^T g||_2 = ||g||_2, \forall g \in \mathbb{G} \tag{3.7}$$

Where g is the gradient update vector for the weights and $\mathbb{G}$ is the set of possible values that the gradient vector can contain. Due to the fact that the L_2 matrix norm describes the highest singular value of the matrix, for this equation to hold true, every singular value in W must be 1. This can be achieved by enforcing orthonormality on the weight matrix. The Björck Orthonormalization algorithm calculates the closest orthonormal matrix given an input $W_0 = W$ through an iterative operation [8]:

$$W_{k+1} = W_k \left(I + \sum_{i=1}^{p} (-1)^p \binom{-\frac{1}{2}}{p} Q_k^p \right) \tag{3.8}$$

With $Q_k = 1 - W_k^T W_k$ and p as a hyperparameter that controls the number of computations performed. More computations allow for a more accurate approximation of the orthonormal matrix, but increases complexity and computation time for the network applying this algorithm. Björck Orthonormalization is applied in the forward step before backpropagation, like spectral normalization. Using the power iteration algorithm we can find the largest single value of the weight matrix and use this to upperbound the weights by 1, thus ensuring that the layer's are 1-Lipschitz. Then we can apply the Björck Orthonormalization algorithm to ensure that all the singular values of the weight matrix are not only bounded by 1 but are equal to 1, which makes them gradient norm preserving. Then depending on the Lipschitz constant of the layer, we can scale the output of the layer by said constant. In this way we can achieve a functional certifiably robust architecture for our anomaly detection algorithms.

4 Implementation

4.1 Deel-Lip

Deel-Lip is a library implemented by Serrurier et. al. as a companion to their research paper [62]. It is an implementation of K-Lipschitz layers for Tensorflow Keras. This specific implementation uses the same concepts introduced in the last section, namely spectral normalization and Björck Orthonormalization. This library implements a multitude of K-Lipschitz layers as well as Lipschitz continuous activation functions. It even implements a version of the Keras Sequential model, that allows one to set a global Lipschitz constant for the model. When we say K-Lipschitz when referring to Deel-Lip, we mean K-Lipschitz with respect to the 2-norm, as this is how they have defined and chosen to ensure Lipschitz continuity. The n-th root of this constant is then distributed as a constraint to each of the n Lipschitz continuous layers of the model, ensuring that the entire model is upperbounded by this parameter. The layers of particular note for us are the SpectralDense layer that can be used in place of normal Tensorflow dense layers and SpectralConv2D that similarly replace Conv2D layers from Tensorflow. SpectralDense functions as expected, but SpectralConv2D does not. Originally we applied SpectralConv2D in place of all normal Conv2D layers in order to make our DCAE, DeepSVDD model, and AnoGAN Lipschitz continuous. We found however that the SpectralConv2D layer often caused erroneous ”blurred” reconstructions in our Autoencoders and GANs.

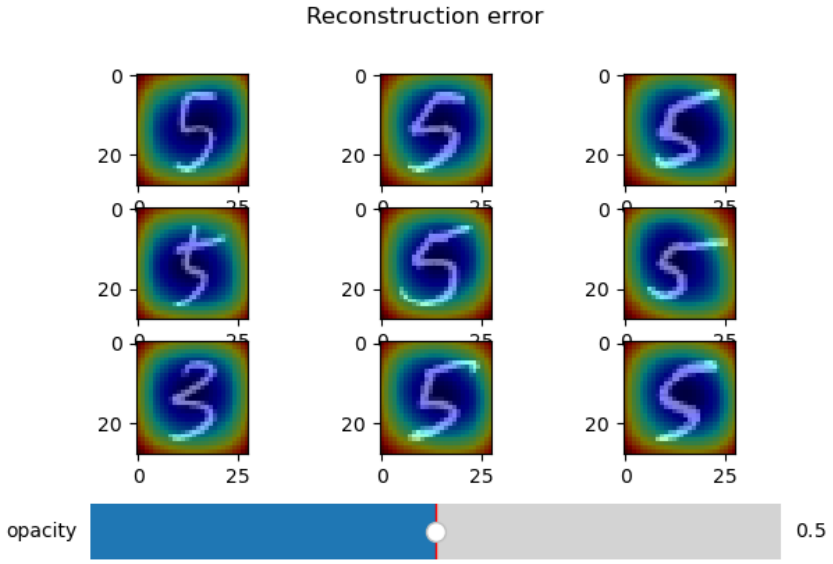

Figure 4.1: Example of the reconstructions generated from a Lipschitz DCAE's decoder using SpectralConv2D layers

In Figure 4.1 the white numbers are the original MNIST digits used as input and the multicolored blurred shapes are the reconstructions from a Lipschitz DCAE overlaid at half opacity. This phenomenon was observed for all inputs and for various Lipschitz constants. Increasing the Lipschitz constant, which should allow the model to be more expressive, did nothing to change the poor reconstructions. Replacing our convolutional decoders in our DCAEs and our convolutional generators in our DCGANs with dense versions using the aforementioned SpectralDense layers seemed to remedy this issue.

This can be visualized in Figure 4.2, where the reconstructions are very visibly improved. Though we are not quite aware of why the SpectralConv2D layers seem to lead to poor reconstructions, we have determined that it is incredibly likely that the implementation of the layers themselves are the problem. Consequently we have notified Serrurier et. al. of this issue, in the hopes that the problem can be further looked into. We still use SpectralConv2D in a more limited capacity, however. They are still present in our DCAE encoders, DeepSVDD models, and DCGAN discriminators, as it seems they can still condense information rather efficiently, as with normal convolutional layers.

Deel-Lip does also caution against using Batch Normalization or Dropout as they are not 1-Lipschitz. None of our standard anomaly detection models use Dropout, so it is easy for us to abstain against using it in our implementation. Due to the rescaling operation after the normalization in Batch Normalization,

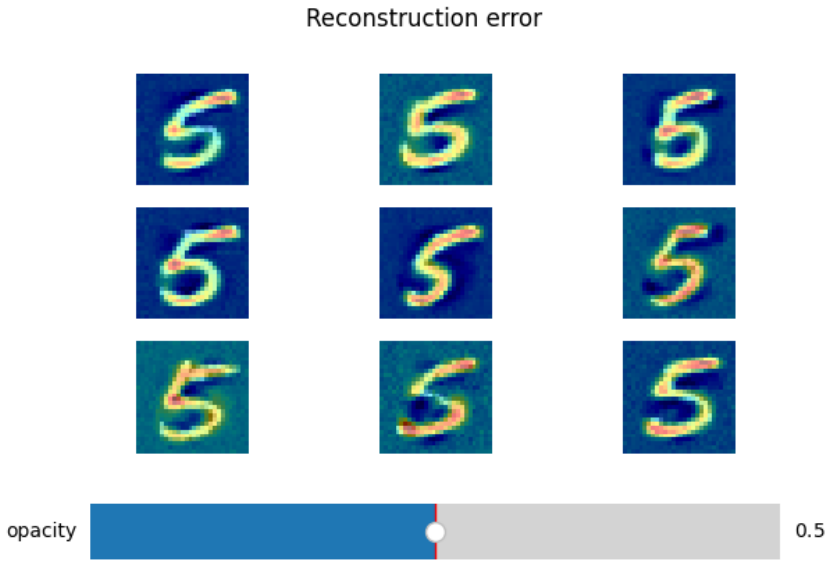

Figure 4.2: Example of the reconstructions generated from a Lipschitz DCAE's decoder using SpectralDense layers

it is not 1-Lipschitz. This means we must abstain from using Batch Normalization in our Lipschitz continuous anomaly detection models, which is quite prevalently used in our standard versions. We do not offer an alternative to Batch Normalization and instead completely omit it, which we will further discuss in Section 5. Additionally Deel-Lip offers a weight initializer called the SpectralInitializer, which we use to intialize the weights of our Lipschitz continuous models to be 1-Lipschitz and orthogonal using spectral normalization and Björck Orthonormalization. Thus the initialization of the weights is one other facet of the Lipschitz models which will often differ with the standard versions.

4.2 DCAE

For all of our anomaly detection algorithms we build a model for the MNIST dataset and a model for CIFAR10. Then we also build an equivalent Lipschitz version for each dataset as well. We base the architecture of our Autoencoders on the architectures described by Ruff et. al. [56]. In principle, the encoder of this Autoencoder shares the same architecture as the DeepSVDD model. The decoder is then built symmetrically to mirror the encoder. The loss used is Mean Squared Error.

Almost every activation function, except for the activation at the output of the Autoencoder, is a LeakyReLU. Where a ReLU is $\max(0, x)$ for a layer output x, a LeakyReLU is $\max(0, x) + \alpha \cdot \min(0, x)$. Instead of squeezing all negative values to 0, a LeakyReLU multiplies these values by α a negative slope that serves to reduce their magnitude. As long as the α is no larger than 1, a LeakyReLU is also 1-Lipschitz. Ruff et. al. suggest a leakiness $\alpha = 0.1$, so we follow suit. Additionally they use weight decay with $\lambda = 10^{-6}$, so we do the same. We also use the optimizer Adam with a batch size of 200 [34]. The weights of the layers are initialized per Glorot Uniform (also called Xavier Uniform), which initializes the weights so that they have the same variance across layers.

Finally Ruff et. al. suggest instituting a two-phase learning rate for finding a local minimum and then fine-tuning to approach the minimum. For the DCAE we train for 250 epochs at a learning rate of 10^{-4} before subsequently dropping to 10^{-5} for 100 epochs, training in total for 350 epochs. For the Lipschitz version of the DCAE, we found that it often did not reach a minimum within 350 epochs and as such removed this two-phase learning rate, opting instead to simply train for 350 epochs at a learning rate of 10^{-4}.

Originally we had built these models, as previously stated, using SpectralConv2D in the decoders. The simple upsampling offered by Keras was not itself 1-Lipschitz, because with a kernel size of $(2, 2)$ it would double the matrix norm of anything passed through. Deel-Lip did not offer 1-Lipschitz upsampling layers that did not affect the channel size, so originally we solved this with a simple Lambda layer that divided the values of the upsampling output by 2. Eventually we found a more elegant solution due to Radford et. al and Springenberg et. al., who suggest replacing pooling and upsampling layers with convolutional layers [54] [67]. Deel-Lip, as stated in the previous section, did offer K-Lipschitz convolutional layers, so instead we decided to implement transposed convolutional layers with a stride of 2 in place of upsampling layers. This change was made in both our standard DCAEs and our Lipschitz DCAEs. Later we were able to identify the poor reconstructions of our Lipschitz DCAEs as being the fault of the SpectralConv2D layers, and consequently replaced them with SpectralDense layers, removing the necessity for upsampling of any kind. Due to this the transposed convolutions can now only be found within our standard DCAEs. This is the only ”major” change made to our DCAEs in

comparison to the work of Ruff et. al. [54], but it ultimately should have no bearing on the outcome as these transposed convolutions function identical to upsampling layers, the only difference being that our network can learn the optimal kernels for upsampling, making our training slightly more expensive. Below are tables detailing our architectures.

MNIST DCAE			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	28x28x1
Conv2D	8	5x5	28x28x8
Batch Norm	N/A	N/A	28x28x8
Leaky ReLU	N/A	N/A	28x28x8
Max Pooling	N/A	2x2	14x14x8
Conv2D	4	5x5	14x14x4
Batch Norm	N/A	N/A	14x14x4
Leaky ReLU	N/A	N/A	14x14x4
Max Pooling	N/A	2x2	7x7x4
Flatten	N/A	N/A	196
Dense	32	N/A	32
Reshape	N/A	N/A	4x4x2
Leaky ReLU	N/A	N/A	4x4x2
Conv2DTranspose	2	5x5	8x8x2
Conv2D	4	5x5	8x8x4
Batch Norm	N/A	N/A	8x8x4
Leaky ReLU	N/A	N/A	8x8x4
Conv2DTranspose	4	5x5	16x16x4
Conv2D	8	5x5	16x16x8
Batch Norm	N/A	N/A	16x16x8
Leaky ReLU	N/A	N/A	16x16x8
Conv2DTranspose	8	5x5	32x32x8
Conv2D	1	5x5	28x28x1
Sigmoid	N/A	N/A	28x28x1

CIFAR10 DCAE			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	32x32x3
Conv2D	32	5x5	32x32x32
Batch Norm	N/A	N/A	32x32x32
Leaky ReLU	N/A	N/A	32x32x32
Max Pooling	N/A	2x2	16x16x32
Conv2D	64	5x5	16x16x64
Batch Norm	N/A	N/A	16x16x64
Leaky ReLU	N/A	N/A	16x16x64
Max Pooling	N/A	2x2	8x8x64
Conv2D	128	5x5	8x8x128
Batch Norm	N/A	N/A	8x8x128
Leaky ReLU	N/A	N/A	8x8x128
Max Pooling	N/A	2x2	4x4x128
Flatten	N/A	N/A	2048
Dense	128	N/A	128
Reshape	N/A	N/A	4x4x8
Leaky ReLU	N/A	N/A	4x4x8
Conv2D	128	5x5	4x4x128
Batch Norm	N/A	N/A	4x4x128
Leaky ReLU	N/A	N/A	4x4x128
Conv2DTranspose	128	5x5	8x8x128
Conv2D	64	5x5	8x8x64
Batch Norm	N/A	N/A	8x8x64
Leaky ReLU	N/A	N/A	8x8x64
Conv2DTranspose	64	5x5	16x16x64
Conv2D	32	5x5	16x16x32
Batch Norm	N/A	N/A	16x16x32
Leaky ReLU	N/A	N/A	16x16x32
Conv2DTranspose	32	5x5	32x32x32
Conv2D	3	5x5	32x32x3
Sigmoid	N/A	N/A	32x32x3

Lipschitz MNIST DCAE			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	28x28x1
SpectralConv2D	8	5x5	28x28x8
Leaky ReLU	N/A	N/A	28x28x8
Max Pooling	N/A	2x2	14x14x8
SpectralConv2D	4	5x5	14x14x4
Leaky ReLU	N/A	N/A	14x14x4
Max Pooling	N/A	2x2	7x7x4
Flatten	N/A	N/A	196
Dense	32	N/A	32
Leaky ReLU	N/A	N/A	32
Dense	64	N/A	64
Leaky ReLU	N/A	N/A	64
Dense	128	N/A	128
Leaky ReLU	N/A	N/A	128
Dense	784	N/A	784
Sigmoid	N/A	N/A	784
Reshape	N/A	N/A	28x28x1

Lipschitz CIFAR10 DCAE			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	32x32x3
SpectralConv2D	32	5x5	32x32x32
Leaky ReLU	N/A	N/A	32x32x32
Max Pooling	N/A	2x2	16x16x32
SpectralConv2D	64	5x5	16x16x64
Leaky ReLU	N/A	N/A	16x16x64
Max Pooling	N/A	2x2	8x8x64
SpectralConv2D	128	5x5	8x8x128
Leaky ReLU	N/A	N/A	8x8x128
Max Pooling	N/A	2x2	4x4x128
Flatten	N/A	N/A	2048
Dense	96	N/A	96
Leaky ReLU	N/A	N/A	96
Dense	192	N/A	192
Leaky ReLU	N/A	N/A	192
Dense	384	N/A	384
Leaky ReLU	N/A	N/A	384
Dense	768	N/A	768
Leaky ReLU	N/A	N/A	768
Dense	1536	N/A	1536
Leaky ReLU	N/A	N/A	1536
Dense	3072	N/A	3072
Sigmoid	N/A	N/A	3072
Reshape	N/A	N/A	32x32x3

4.3 DeepSVDD

For DeepSVDD we use the same architecture for both Soft-Boundary and One-Class. Just like the DCAEs we use weight decay with the same hyperparameter. We use Adam as an optimizer with a batch size of 200 again too. Additionally we use the same two-phase learning rate, but instead of 250+100 epochs we do 150+100 epochs also per Ruff et. al. [56]. We set the hyperparameter $\nu = 0.1$. This hyperparameter, as we have previously explained, dictates the proportion of the data that is anomalous. Ruff et. al. use both 0.01 and 0.1, choosing the one that leads to best performance. Due to time constraints we simply choose 0.1 across the board. We had also previously mentioned that it may be clever to update the radius of the enclosing hypersphere every few epochs instead of every single epoch. As such we update the radius every 5 epochs, also per Ruff et. al. We also initialize the DeepSVDD weights with the weights from

the encoder of the DCAE [43]. This is a pre-training procedure introduced by Ruff et. al. as an initialization option. Consequently we train a DCAE first, then take the final weights after training from the encoder and use them for initialization of the weights of the corresponding DeepSVDD model.

Finally we alter the value for the center for our Lipschitz models. For the standard models, we compute one forward pass and take the mean of the output of this pass. We then ensure to offset any value that is exactly zero, so as to avoid hypersphere collapse. This becomes the value for the center of our hypersphere. For the Lipschitz models, due to the Lipschitz constraint on the model, it is possible that the initial forward pass is not well representative of where the values eventually will be projected to towards the end of training. As such it is possible that we can initialize the center to be too far from the optimum, forcing the volume of the hypersphere to be much larger than necessary in order to enclose the data. For this reason we initialize the center to be near 1 by calculating the same exact center as for the standard models and then dividing this by the absolute value of itself. This makes our center less adaptable to the data, but also avoids particularly bad initializations that could be caused by the Lipschitz constraint's restrictions on output variation. Below are tables detailing our DeepSVDD model architectures.

MNIST DeepSVDD			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	28x28x1
Conv2D	8	5x5	28x28x8
Batch Norm	N/A	N/A	28x28x8
Leaky ReLU	N/A	N/A	28x28x8
Max Pooling	N/A	2x2	14x14x8
Conv2D	4	5x5	14x14x4
Batch Norm	N/A	N/A	14x14x4
Leaky ReLU	N/A	N/A	14x14x4
Max Pooling	N/A	2x2	7x7x4
Flatten	N/A	N/A	196
Dense	32	N/A	32

CIFAR10 DeepSVDD			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	32x32x3
Conv2D	32	5x5	32x32x32
Batch Norm	N/A	N/A	32x32x32
Leaky ReLU	N/A	N/A	32x32x32
Max Pooling	N/A	2x2	16x16x32
Conv2D	64	5x5	16x16x64
Batch Norm	N/A	N/A	16x16x64
Leaky ReLU	N/A	N/A	16x16x64
Max Pooling	N/A	2x2	8x8x64
Conv2D	128	5x5	8x8x128
Batch Norm	N/A	N/A	8x8x128
Leaky ReLU	N/A	N/A	8x8x128
Max Pooling	N/A	2x2	4x4x128
Flatten	N/A	N/A	2048
Dense	128	N/A	128

Lipschitz MNIST DeepSVDD			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	28x28x1
SpectralConv2D	8	5x5	28x28x8
Leaky ReLU	N/A	N/A	28x28x8
Max Pooling	N/A	2x2	14x14x8
SpectralConv2D	4	5x5	14x14x4
Leaky ReLU	N/A	N/A	14x14x4
Max Pooling	N/A	2x2	7x7x4
Flatten	N/A	N/A	196
Dense	32	N/A	32

Lipschitz CIFAR10 DeepSVDD			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	32x32x3
SpectralConv2D	32	5x5	32x32x32
Leaky ReLU	N/A	N/A	32x32x32
Max Pooling	N/A	2x2	16x16x32
SpectralConv2D	64	5x5	16x16x64
Leaky ReLU	N/A	N/A	16x16x64
Max Pooling	N/A	2x2	8x8x64
SpectralConv2D	128	5x5	8x8x128
Leaky ReLU	N/A	N/A	8x8x128
Max Pooling	N/A	2x2	4x4x128
Flatten	N/A	N/A	2048
Dense	128	N/A	128

4.4 AnoGAN

The AnoGAN was first proposed by Schlegl et. al. [59], but they refrain from making concrete suggestions for the architecture of this GAN, furthermore they test their AnoGAN on a completely different data set from ours. For this reason, we follow in the footsteps of Ruff et. al. yet again [56], as they train and test an AnoGAN for the sake of comparison to their DeepSVDD model. Therefore we base the general architecture on the DCGAN of Radford et. al. [54]. As such we use the Adam optimizer with a batch size of 128. This time we alter the hyperparameter β_1 to be 0.5 instead of its default value of 0.9. We train the MNIST DCGANs for 20 epochs and CIFAR10 DCGANs for 50 epochs before beginning the iterative AnoGAN process of finding latent values for each sample. For this step we do 500 iterative backpropagation steps per sample just like Schlegl et. al citeDBLP:journals/corr/SchleglSWSL17. For the LeakyReLUs, as opposed to the previous models we set an α of 0.2 and use a learning rate of 0.0002. Finally all the weights are initialized from a normal distribution with a mean of 0 and a standard deviation of 0.02. All of these changes are according to The particular composition of layers and filters are according to Metz et. al. [45], who train a GAN for MNIST and CIFAR10 datasets. If it is not clear from analyzing the output sizes in the tables below, every convolutional layer has a stride of 2.

MNIST Generator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	256
Dense	8192	N/A	8192
Batch Norm	N/A	N/A	8192
ReLU	N/A	N/A	8192
Reshape	N/A	N/A	4x4x512
Conv2DTranspose	256	3x3	8x8x256
Batch Norm	N/A	N/A	8x8x256
ReLU	N/A	N/A	8x8x256
Conv2DTranspose	128	3x3	16x16x128
Batch Norm	N/A	N/A	16x16x128
ReLU	N/A	N/A	16x16x128
Conv2DTranspose	64	3x3	32x32x64
Batch Norm	N/A	N/A	32x32x64
ReLU	N/A	N/A	32x32x64
Conv2D	1	5x5	28x28x1
Tanh	N/A	N/A	28x28x1
Conv2D	3	3x3	32x32x3
Tanh	N/A	N/A	32x32x3

MNIST Discriminator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	28x28x1
Conv2D	64	3x3	14x14x64
Input	N/A	N/A	32x32x3
Conv2D	64	3x3	16x16x64
LeakyReLU	N/A	N/A	14x14x64
Conv2D	128	3x3	7x7x128
Batch Norm	N/A	N/A	7x7x128
LeakyReLU	N/A	N/A	7x7x128
Conv2D	256	3x3	4x4x256
Batch Norm	N/A	N/A	4x4x256
LeakyReLU	N/A	N/A	4x4x256
Flatten	N/A	N/A	4096
Dense	1	N/A	1
Sigmoid	N/A	N/A	1

CIFAR10 Generator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	256
Dense	8192	N/A	8192
Batch Norm	N/A	N/A	8192
ReLU	N/A	N/A	8192
Reshape	N/A	N/A	4x4x512
Conv2DTranspose	256	3x3	8x8x256
Batch Norm	N/A	N/A	8x8x256
ReLU	N/A	N/A	8x8x256
Conv2DTranspose	128	3x3	16x16x128
Batch Norm	N/A	N/A	16x16x128
ReLU	N/A	N/A	16x16x128
Conv2DTranspose	64	3x3	32x32x64
Batch Norm	N/A	N/A	32x32x64
ReLU	N/A	N/A	32x32x64
Conv2D	3	3x3	32x32x3
Tanh	N/A	N/A	32x32x3

CIFAR10 Discriminator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	32x32x3
Conv2D	64	3x3	16x16x64
LeakyReLU	N/A	N/A	16x16x64
Conv2D	128	3x3	8x8x128
Batch Norm	N/A	N/A	8x8x128
LeakyReLU	N/A	N/A	8x8x128
Conv2D	256	3x3	4x4x256
Batch Norm	N/A	N/A	4x4x256
LeakyReLU	N/A	N/A	4x4x256
Flatten	N/A	N/A	4096
Dense	1	N/A	1
Sigmoid	N/A	N/A	1

Lipschitz MNIST Generator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	256
SpectralDense	8192	N/A	8192
ReLU	N/A	N/A	8192
SpectralDense	4096	N/A	4096
ReLU	N/A	N/A	4096
SpectralDense	2048	N/A	2048
ReLU	N/A	N/A	2048
SpectralDense	1024	N/A	1024
ReLU	N/A	N/A	1024
SpectralDense	784	N/A	784
Tanh	N/A	N/A	784
Reshape	N/A	N/A	28x28x1

Lipschitz MNIST Discriminator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	28x28x1
SpectralConv2D	64	3x3	14x14x64
LeakyReLU	N/A	N/A	14x14x64
SpectralConv2D	128	3x3	7x7x128
LeakyReLU	N/A	N/A	7x7x128
SpectralConv2D	256	3x3	4x4x256
LeakyReLU	N/A	N/A	4x4x256
Flatten	N/A	N/A	4096
SpectralDense	1	N/A	1
Sigmoid	N/A	N/A	1

Lipschitz CIFAR10 Generator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	256
SpectralDense	8192	N/A	8192
ReLU	N/A	N/A	8192
SpectralDense	6312	N/A	6312
ReLU	N/A	N/A	6312
SpectralDense	4608	N/A	4608
ReLU	N/A	N/A	4608
SpectralDense	3072	N/A	3072
Tanh	N/A	N/A	3072
Reshape	N/A	N/A	32x32x3

Lipschitz CIFAR10 Discriminator			
Layer	Number of Neurons/-Filters	Kernel	Output Size
Input	N/A	N/A	32x32x3
SpectralConv2D	64	3x3	16x16x64
LeakyReLU	N/A	N/A	16x16x64
SpectralConv2D	128	3x3	8x8x128
LeakyReLU	N/A	N/A	8x8x128
SpectralConv2D	256	3x3	4x4x256
LeakyReLU	N/A	N/A	4x4x256
Flatten	N/A	N/A	4096
SpectralDense	1	N/A	1
Sigmoid	N/A	N/A	1

5 Experiments

Two experiments are performed. The second experiment is the main experiment, where we, similar to Ruff et. al. [56], have an experiment for each class of MNIST and CIFAR10, where one class is treated as normal and the rest anomalous. In this specific experiment we are trying to assess the effects of a Lipschitz constraint on our three anomaly detection algorithms, comparing their performance and measured metrics to the standard versions. As we have detailed in the previous section, many of our hyperparameter choices are based on prior work and research and this was done intentionally to limit the amount of hyperparameters to optimize. The main hyperparameter that we have, however, is the global Lipschitz constant for our Lipschitz continuous models. Thus the first experiment is to find a suitable Lipschitz constant for each model type. This initial experiment is less comprehensive than the second and simply serves to give us both an idea of the effect of different Lipschitz constants on these models and which Lipschitz constants perform the best.

We measure the performance of these anomaly detection algorithms with two metrics. The first being the ROC-AUC. This stands for the Reveiver Operating Characteristic Area Under Curve. The ROC-AUC plots the true positive rate, the rate at which the model correctly predicts the positive class (in our case the positive class is the normal class) and plots this against the false positive rate. Which is the rate at which the model incorrectly predicts the positive class, i.e. anomalies are classified as normal. The area under this curve is then the ROC-AUC and can measure from 0.0 to 1.0. A 0.0 would signify that the model's predictions are completely wrong in all cases, whereas a 1.0 is the opposite, a classifier that is completely correct. The only other value for the ROC-AUC that is of particular note is 0.5, which would indicate that the model is more-or-less random in its behavior. The ROC-AUC is the main performance metric that we use to assess the accuracy of our models.

The second metric is a robustness metric. Robustness can be difficult to measure, luckily Lipschitz continuity provides certifiable robustness for models, so there is no doubt that our Lipschitz continuous models are more robust than their standard counterparts, when they function. The point of the robustness metric is to gain some insight into how much more robust, in a quantitative sense, they are. Trying to use equation (3.2) to prove the size of the margin for every sample would be computationally expensive and as we will detail later in this section, this would therefore not have been a viable metric, as we already run into issues with computation time. Thus we settle on a much simpler met-

ric, namely for input samples x and y:

$$\frac{||f(x) - f(y)||_2}{||x - y||_2} \leq L \tag{5.1}$$

This is simply another formation for the definition of Lipschitz continuity with respect to the $L2$-norm. Here the norm of the difference between two outputs is divided by the norm of their inputs to achieve an approximate lower bound on the Lipschitz constant. In order for this calculation to be computationally efficient, we simply calculate as many of these individual robustness calculations as can be done in 15 seconds. We use data from the test set for these computations. Then we take the highest value calculated, being that this metric is a lower bound, and repeat this process 10 times in total for a total of 150 seconds of computation time. At the end we have a list of 10 lower bounds for the Lipschitz constant and the largest among them becomes our approximate lower bound for the model. The reason we do not simply calculate the lower bound in one 150 second chunk, is so that we can use the standard deviation of this list of 10 lower bounds to assess how often the highest value or a value nearing the highest value was calculated. If the maximum lower bound was able to be calculated in multiple iterations then the distance (in standard deviations) of this maximum from the mean of the list of 10 lower bounds will naturally be lower. This allows us to assess the trustworthiness of our metric, as a lower distance from the mean will indicate that the value calculated is the highest lower bound that would be able to be calculated from the data.

So as stated, due to the inherent inexact nature of this metric, we provide alongside the calculated lower bound for the Lipschitz constant also the amount of standard deviations this value is from the mean of the values calculated through this process. For the vast majority of the robustness values, they lie within three standard deviations of the mean, which indicates quite a bit of variation for a list of 10 values. Nonetheless three standard deviations is not too egregious, it does however indicate that some models likely only found their maximum lower bound in one or two of the iterations. In these instances, more time could have been allotted to derive a stricter lower bound. These standard deviations simply serve to grant greater insight into the validity of this metric.

While the ROC-AUC can be used to compare different models with each other in terms of raw accuracy, our robustness metric's raw value means relatively little to us. The metric can only really be used to evaluate models in a comparative sense. For this reason we will only use our robustness metric to compare Lipschitz versions of models with their standard counterparts. We will not, for instance, be comparing the robustness metric of one Lipschitz model with another Lipschitz model. This also aligns with our goal, to identify and measure the changes caused by enhancing the robustness of these models via Lipschitz continuity. It is not within our purview to identify the "best" Lip-

schitz model. While we may comment on which models achieve the highest ROC-AUC, we will not make claims as to which is the most robust.

For all DCAEs and DeepSVDD models we preprocess the images via global contrast normalization with the 1-norm. This subtracts the mean from each image and then rescales the image so that their standard deviation is some constant value. After this we rescale the normalized images to the range of $[0, 1]$ with min-max-scaling [56]. For the AnoGAN the images are simply scaled to the range $[-1, 1]$, which is the same as the hyperbolic tangent activation function [54].

Thus we will spend the first section detailing our first experiment, showing our results and analyzing them. Then we will move onto the second, more comprehensive experiment. Here we will similarly discuss our results and also perform an analysis on the affect Lipschitz continuity has on these models, as well as explaining and investigating strange behavior observed in our experiments.

5.1 Deriving Suitable Lipschitz Constants

As previously stated, the goal of this experiment is to gain insight into how changing the Lipschitz constant effects these Lipschitz continuous models. Additionally we want to identify an "optimal" constant for use in the second experiment. First of all we have a selection of seven Lipschitz constants that we test on both the MNIST and CIFAR10 versions of our models. We wanted to have a rather wide breadth of constants to test, so we chose 0.5, 1, 2, 5, 10, 50, and 100. These represent the global Lipschitz constants for these models and as clarified in Section 4.1, for a model with n Deel-Lip layers, the nth root of this global constant is distibuted to the n layers. For any layer that is not directly from Deel-Lip, like ReLU, which is offered by Tensorflow, they are all 1-Lipschitz so as not to change the global constant.

For each model we train them according to the parameters from Section 4 on one class. This class is the 0 class for both datasets, so for MNIST it is the number 0 and for CIFAR10 it is an airplane. These models are trained with these classes only and then are tested using the entire test data set offered by each, which in both cases is 10,000 data points. All other classes that are not the 0 class are treated as anomalous. We choose the normal class 0 arbitrarily and any other class could have realistically been chosen. If time were not a consideration one could calculate these very same metrics for each class, which would have taken ten times the computational time. For the DCAE and both DeepSVDD models, they are trained and tested 3 separate times and the mean of their ROC-AUC from these 3 iterations is calculated along with the standard deviation. For the robustness metric, we take the max value for the lower bound from these 3 iterations instead. As stated before, the distance of this value from the mean, measured in standard deviations, is also supplied.

The AnoGAN is treated differently from these other models however. Rather early on in our experiments, we were able to identify that the AnoGAN algorithm is very expensive in terms of computational time. For an allotted time period of two weeks, the CIFAR10 AnoGAN was not able to complete one full iteration through all of the test data, while the MNIST version completed near the end of this time period. This is corroborated by Mattia et. al. [44], who also note that the computation time for the AnoGAN algorithm as a major weakness. For this reason, we limited the size of the test data to only 1000 samples instead of using the entire data set. We filtered out the 1000 samples out of the test data set according to a seed which we defined to be 1234 for all our GAN experiments. Even in the next main experiment we use this same seed. Additionally for this experiment only we perform just one iteration of training and testing instead of three. This was done in order to ensure that a suitable Lipschitz constant could be derived in time, as the second experiment could not begin until this one ended.

5.1.1 Results

The results of the first experiment are listed in two tables below. The first includes information on the ROC-AUC, while the second table contains the robustness metric.

Data Set	Lip. Const	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0.5	80.7 ±0.1	29.2 ±1.9	56.1 ±3.0	80.1
	1.0	83.9 ±0.1	30.5 ±2.9	54.2 ±2.5	95.1
	2.0	90.9 ±0.1	29.1 ±2.7	52.7 ±2.7	91.7
	5.0	92.5 ±0.1	77.0 ±13.2	82.5 ±2.3	81.0
	10.0	84.9 ±0.7	96.3 ±1.2	93.8 ±1.0	89.1
	50.0	98.7 ±0.1	97.2 ±0.3	97.1 ±0.9	81.3
	100.0	98.9 ±0.1	97.3 ±0.8	96.8 ±0.8	85.0
CIFAR10	0.5	45.5 ±0.3	40.0 ±1.5	44.5 ±0.2	61.7
	1.0	48.1 ±0.1	39.4 ±1.1	44.4 ±0.8	56.0
	2.0	53.2 ±0.2	40.3 ±1.7	58.3 ±0.9	65.0
	5.0	61.2 ±0.4	54.7 ±1.4	59.1 ±1.6	61.2
	10.0	65.1 ±0.4	53.9 ±1.9	58.0 ±0.8	60.1
	50.0	61.9 ±2.2	56.9 ±1.4	62.1 ±1.2	62.7
	100.0	59.5 ±0.2	61.6 ±1.4	63.4 ±1.1	49.3

Data Set	Lip. Const	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0.5	0.032 (1.555 σ)	0.041 (1.851 σ)	0.064 (1.605 σ)	0.068 (0.500 σ)
	1.0	0.066 (1.916 σ)	0.072 (1.801 σ)	0.106 (1.256 σ)	0.064 (0.968 σ)
	2.0	0.128 (2.034 σ)	0.145 (1.753 σ)	0.227 (2.055 σ)	0.072 (0.816 σ)
	5.0	0.309 (1.672 σ)	0.331 (1.506 σ)	0.311 (2.012 σ)	0.054 (0.333 σ)
	10.0	0.536 (1.588 σ)	0.237 (2.054 σ)	0.221 (1.165 σ)	0.069 (0.500 σ)
	50.0	1.127 (1.488 σ)	0.251 (1.776 σ)	0.238 (1.476 σ)	0.066 (0.423 σ)
	100.0	1.228 (1.722 σ)	0.301 (1.571 σ)	0.323 (1.629 σ)	0.070 (0.500 σ)
CIFAR10	0.5	0.008 (2.217 σ)	0.043 (2.073 σ)	0.034 (1.896 σ)	0.110 (0.655 σ)
	1.0	0.016 (2.500 σ)	0.087 (2.526 σ)	0.067 (1.528 σ)	0.059 (0.722 σ)
	2.0	0.032 (2.089 σ)	0.159 (1.403 σ)	0.141 (2.406 σ)	0.111 (0.655 σ)
	5.0	0.075 (1.697 σ)	0.184 (2.229 σ)	0.151 (1.723 σ)	0.115 (0.654 σ)
	10.0	0.367 (1.963 σ)	0.180 (1.973 σ)	0.154 (2.214 σ)	0.083 (0.803 σ)
	50.0	0.985 (2.322 σ)	0.149 (2.022 σ)	0.124 (2.536 σ)	0.120 (0.515 σ)
	100.0	1.002 (1.319 σ)	0.136 (2.209 σ)	0.121 (1.584 σ)	0.143 (0.655 σ)

Through this initial experiment we found that the AnoGAN, specifically the DCGAN that it uses to perform anomaly detection with, suffered mode collapse with every Lipschitz constant and with both data sets. Mode collapse occurs when the Generator of a GAN begins to create only a small set of images that it has identified as being able to fool the Discriminator [71]. The Generator then fails to capture the underlying distribution of the data as it becomes ”stuck” creating a limited set of outputs.

Figure 5.1 shows an example of the DCGAN's output after 20 epochs of training on the normal class of 0. While it does clearly generate zeros, each individual image in the figure is fed a different randomized latent noise and should thus generate a different image of a 0 for each image. This is clearly not the case as it generates the same image regardless of the noise it's given. Mode collapse is a relatively common failure state for GANs that are somewhat notorious for

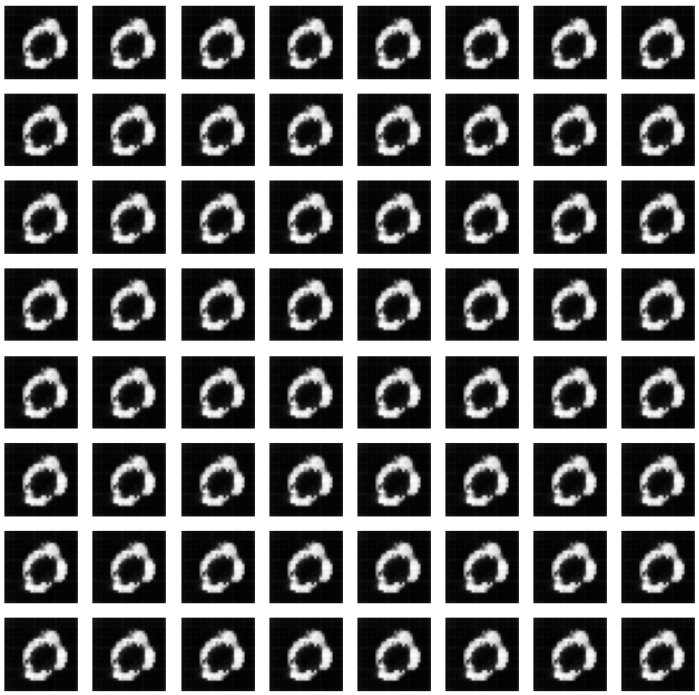

Figure 5.1: An example of mode collapse from a 1-Lipschitz MNIST AnoGAN.

having difficult training.

We also observe that the standard deviation for the 5-Lipschitz MNIST Soft-Boundary DeepSVDD model is strangely high. The first iteration achieved an ROC-AUC of 60.7, then the second 93.1 and finally 77.1 in the third. The reason for this is unknown, but is indicative of the model having a relatively large error rate and thus makes it unfit for a choice as a Lipschitz constant.

It can also be perceived that the ROC-AUCs for the CIFAR10 models are usually much lower. In fact a sizable amount of the CIFAR10 models, especially the ones with low Lipschitz constants, perform worse than a model with random behavior. The reason for this is the complexity of the data. The MNIST digits are grayscale and are only $28x28$, whereas the CIFAR10 images are RGB images with three color channels. Additionally the images are larger at $32x32$. These two differences mean that the CIFAR10 images are more complex in nature and thus it is generally more difficult for a neural network to find a good function that approximates them. Especially with lower Lipschitz constants, the models

lack the expressiveness to capture the complexity of the data and thus achieve particularly bad performance.

Additionally we observe that generally as the Lipschitz constant increases, the ROC-AUC also increases. This is to be expected as the larger the Lipschitz constant, the less constrained a model is and the easier it will be to find an optimum. The size of a Lipschitz constant should theoretically correlate directly with the robustness of a model. A smaller Lipschitz constant leads to a more robust, constrained model, that may suffer in terms of expressiveness which translates to poorer performance metrics. A larger Lipschitz constant should lead to a less robust model that performs better.

Finally we need a way to determine which Lipschitz constant should be chosen for the second experiment. Thus we developed a rather simple score for determining a suitable constant. For a set of seven models $M = \{M_{0.5}, M_{1.0}, M_{2.0}, M_{5.0}, M_{10.0}, M_{50.0}, M_{100.0}\}$ and a set of ROC-AUC scores $R = \{R_i | i \in M\}$, as well as lower Lipschitz bounds $L = \{L_i | i \in M\}$:

$$\max_i (R_i - \frac{1}{20} \cdot \frac{L_i}{\max L}) \tag{5.2}$$

We do not want to only evaluate the models based on how high the ROC-AUC is. So instead we create a score that finds the constant that maximizes equation (5.2). We divide the robustness metric L_i by the maximum metric calculated for the model across all Lipschitz constants, so that it is on the same range as the ROC-AUC ($[0, 1]$). Then we weigh the robustness metric less as we still want the dominant deciding factor to be the ROC-AUC. The constant $\frac{1}{20}$ can be made smaller or larger depending on how one wants to weigh either component of the equation. Ideally we find a lipschitz constant that creates a model with both a high ROC-AUC and a relatively low robustness metric, which translates to higher robustness.

In the end we determine to use the following Lipschitz constants:

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	2.0	50.0	50.0	1.0
CIFAR10	10.0	100.0	100.0	2.0

5.1.2 Analysis

Here we would like to do further analysis on some of the observations we have made in the previous subsection. First we would like to further discuss the mode collapse that the AnoGAN models suffer from. Upon further investigation into the possible cause of the mode collapse, we have determined that this failure of the model is likely based on the architecture of the DCGAN. As we have previously mentioned, Radford et. al. come up with a few guidelines for

building successful DCGANs [54]. One of those is the use of Batch Normalization, which they state helps to stabilize training and avert mode collapse. Our Lipschitz AnoGAN models, however, do not use Batch Normalization. They cannot use Batch Normalization due to the architectural constraints imposed by Deel-Lip. As we have previously mentioned, the rescaling that Batch Normalization performs is not necessarily 1-Lipschitz. The most likely cause of the mode collapse is the lack of Batch Normalization. This is difficult to test, as we cannot simply add Batch Normalization to these models because they are incompatible with Deel-Lip layers. We do know that our standard AnoGANs, which have standard DCGANs with Batch Normalization do not suffer mode collapse. Pictured below are two figures, which show the loss curves of a successful DCGAN and a DCGAN suffering mode collapse. It can be observed that the standard DCGAN "cleanly" moves towards an optimum, whereas the Lipschitz DCGAN seemingly finds a single solution that can reliably fool the Discriminator and sticks with it.

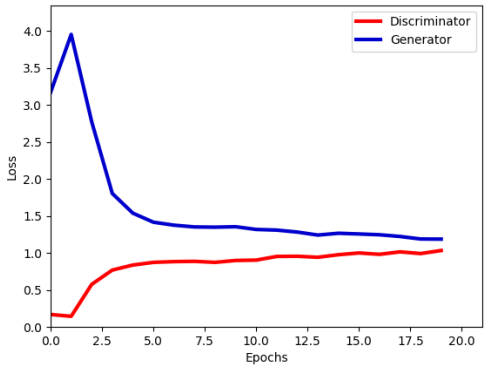

Figure 5.2: Loss of a successful standard DCGAN

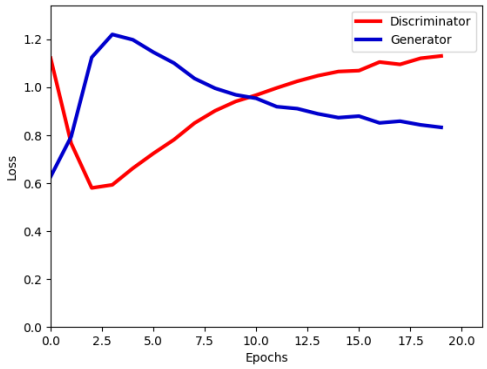

Figure 5.3: Loss of a Lipschitz DCGAN suffering mode collapse

Another strange behavior is that, although the Lipschitz AnoGANs universally suffer from mode collapse, their ROC-AUCs are relatively high considering this fact. This is due to two reasons. One of which is, although the model suffers from mode collapse, as visible in Figure 5.1, it still learns how to generate an image that resembles the trained upon class. When the AnoGAN peforms its iterative reverse mapping for a latent value that generates an image as close to the test sample as possible, this entire process is essentially invalidated, because no matter the latent value, the same image is generated. Nonetheless when this image is compared to other images of the same class it will likely look more similar to them than the anomalous images, thus leading to a relatively high ROC-AUC. The second reason is specific to the MNIST AnoGAN. We have already covered that we use the 0 class as the normal class for all these models. A consequence of this is that the 0 class for MNIST is the digit 0 rather obviously. What is important here to consider is that the numbers 0 and 1 are rather unique among the MNIST digits in that their structure is particularly easily identifiable. The other digits have naturally less resemblance to these numbers as well. So a compounding reason for why the ROC-AUC is so high is that the digit 0 is particularly easily identifiable.

We also notice that, quite oddly, for both DeepSVDD models, as the Lipschitz constant reaches 10.0 and higher, the robustness metric has a tendency to decrease. As our robustness metric is simply an approximate lower bound of the global Lipschitz constant of the model, one would expect this value to raise as the global Lipschitz constant increases. This is indeed what we observe for the DCAEs and AnoGANs, although the AnoGANs should likely not be considered due to their failure. It is possible that with larger Lipschitz constants, the model finds that the most optimal function to approximate generates outputs that are somewhat squeezed within a constrained range. The Lipschitz constant only guarantees the maximum variation in the ouput not the minimum

after all. We find this difficult to believe, but the DeepSVDD models, unlike the DCAE and GANs do not produce image outputs and thus it is a bit more difficult to identify why the outputs sampled would be more similar in models with larger Lipschitz constants. It is also possible that due to the nature of how we calculated our robustness metric that the "furthest" outputs were simply not randomly sampled within the 150 seconds of calculation time and it is likely that more time should have been given.

In a similar vein it can also be observed that, although the growth of the Lipschitz constant is correlated with the growth in performance via the ROC-AUC, it is not a linear correlation. This is not indicative of a problem or failure of the models, it is simply an observation we made. For example, doubling the Lipschitz constant from 1.0 to 2.0 for the MNIST DCAE, leads to about an 8.3% growth in the ROC-AUC, however between 2.0 and 5.0, which is more than double the Lipschitz constant, only a growth of 1.5% can be observed. For this very reason we did not want to choose a Lipschitz constant purely based on the ROC-AUC, because, for instance the Soft-Boundary DeepSVDD model achieves the highest ROC-AUC with a Lipschitz constant of 100.0. This constant is double the next lowest constant of 50.0, which achieves an ROC-AUC, that is only 0.1% worse.

We also find a general correlation between the size of the Lipschitz constant and the minimum loss attained at the end of training. This phenomenon is represented in all the Lipschitz models, but is most clearly recognizable in the DCAEs. Logically this observation makes sense. As we have already covered, a smaller Lipschitz constant constrains a model's learning and can lead to a lack of expressiveness. This smaller Lipschitz constant can prevent a model from achieving an "optimal" setting for its free parameters, thus causing the loss to remain high in magnitude at the end of training. As we evaluated which Lipschitz constant we should choose purely based on the ROC-AUC and our robustness metric, the Lipschitz constant choice of 2.0 for the DCAE is one that still has a rather high loss.

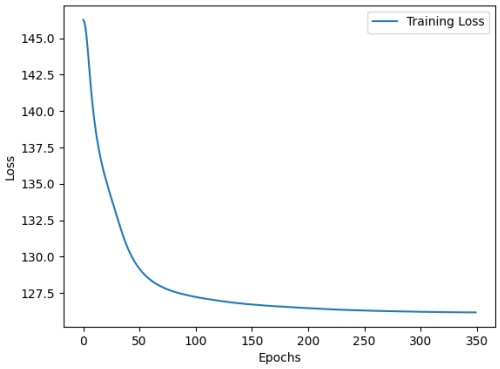

Figure 5.4: DCAE Loss with Lipschitz constant of 2.0

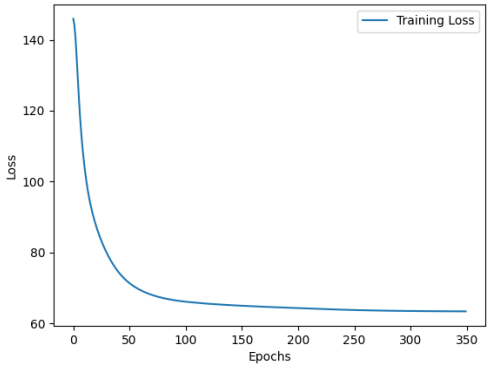

Figure 5.5: DCAE Loss with Lipschitz constant of 10.0

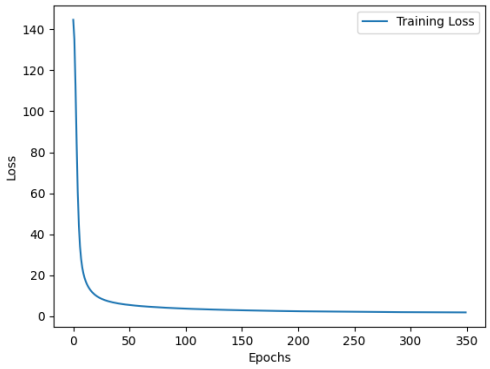

Figure 5.6: DCAE Loss with Lipschitz constant of 50.0

We finally note that both variants of the DeepSVDD models appear to require a higher Lipschitz constant of at least 10.0 to achieve good performance. In fact, for any Lipschitz constant smaller than 5.0 there seems to be quite a notable drop in performance. It is likely that our center for all Lipschitz DeepSVDD models being initialized around the latent point 1 leads to the models with lower Lipschitz constants performing much worse. A bad center initialization can lead to the enclosing hypersphere containing many anomalies or not being able to cover enough normal samples. This drives down the true positive rate and drives up the false positive rate, lowering the ROC-AUC.

5.2 Measuring Performance Changes

In this experiment we seek to understand the differences between standard and Lipschitz continuous models. For each class present in MNIST and CIFAR10, we have an experiment where that class is treated as normal and all others anomalous. For each of these, three iterations are performed in order to achieve a standard deviation and mean for our performance metrics. This time the AnoGAN also has three iterations for each class as opposed to the last experiment. However all the AnoGAN models, whether standard or Lipschitz, just as in the last experiment are limited to 1000 test samples with the seed 1234.

5.2.1 Results

The first two tables listed here depict the ROC-AUC and lower Lipschitz bound for each standard model when treating the listed class as normal. These statistics serve as a baseline for us to compare our Lipschitz models to.

Data Set	Normal Class	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0	98.5 ± 0.1	98.3 ± 0.2	98.0 ± 0.6	93.3 ± 1.8
	1	99.4 ± 0.0	99.6 ± 0.1	99.6 ± 0.0	99.5 ± 0.3
	2	89.1 ± 0.5	89.8 ± 0.9	91.3 ± 0.4	74.2 ± 2.0
	3	91.1 ± 0.5	90.6 ± 1.2	91.6 ± 1.4	78.7 ± 2.4
	4	93.4 ± 0.8	93.8 ± 0.8	93.7 ± 0.2	82.9 ± 1.1
	5	87.4 ± 0.7	87.8 ± 1.8	87.9 ± 2.4	78.4 ± 3.0
	6	96.5 ± 0.0	98.3 ± 0.3	98.4 ± 0.2	93.4 ± 1.1
	7	93.9 ± 0.0	93.1 ± 0.2	94.2 ± 0.2	90.7 ± 1.3
	8	89.5 ± 0.6	92.6 ± 1.4	93.0 ± 0.5	68.0 ± 0.7
	9	95.1 ± 0.1	95.2 ± 0.7	95.1 ± 0.7	86.8 ± 0.8
CIFAR10	airplane	59.0 ± 0.3	62.5 ± 1.2	61.8 ± 1.0	63.2 ± 2.4
	automobile	49.6 ± 0.2	60.5 ± 0.5	59.8 ± 1.0	44.9 ± 1.2
	bird	48.2 ± 0.1	47.8 ± 1.0	48.7 ± 0.5	64.9 ± 0.6
	cat	58.8 ± 0.1	54.9 ± 0.3	56.6 ± 0.3	51.6 ± 2.9
	deer	47.6 ± 0.1	55.5 ± 0.5	54.3 ± 0.2	70.6 ± 1.7
	dog	61.4 ± 0.3	61.1 ± 1.1	62.7 ± 0.5	50.9 ± 1.4
	frog	40.1 ± 0.3	60.9 ± 0.7	58.6 ± 0.5	66.8 ± 2.4
	horse	54.6 ± 0.2	60.4 ± 0.1	60.3 ± 0.7	52.3 ± 0.7
	ship	74.7 ± 0.1	73.6 ± 0.7	71.6 ± 2.3	68.1 ± 2.3
	truck	59.7 ± 0.3	65.9 ± 0.2	65.5 ± 0.4	51.6 ± 2.2

Data Set	Normal Class	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0	1.400 (1.630 σ)	0.792 (2.391 σ)	0.796 (2.365 σ)	4.783 (0.500 σ)
	1	1.631 (2.695 σ)	1.336 (1.517 σ)	1.494 (2.147 σ)	3.382 (0.355 σ)
	2	1.338 (1.984 σ)	0.776 (2.264 σ)	0.662 (1.683 σ)	5.594 (1.224 σ)
	3	1.459 (1.906 σ)	0.806 (1.659 σ)	0.794 (1.542 σ)	4.697 (0.655 σ)
	4	1.334 (1.923 σ)	1.253 (2.348 σ)	1.125 (1.319 σ)	4.511 (0.500 σ)
	5	1.313 (2.157 σ)	0.950 (1.836 σ)	0.810 (1.460 σ)	4.900 (0.739 σ)
	6	1.459 (1.679 σ)	0.741 (2.402 σ)	0.655 (2.258 σ)	4.538 (0.655 σ)
	7	1.450 (2.059 σ)	0.711 (1.885 σ)	0.661 (1.943 σ)	4.220 (0.500 σ)
	8	1.409 (2.193 σ)	0.759 (2.222 σ)	0.668 (1.624 σ)	4.616 (0.655 σ)
	9	1.588 (1.709 σ)	1.104 (1.694 σ)	1.136 (2.388 σ)	4.115 (0.816 σ)
CIFAR10	airplane	1.032 (2.078 σ)	0.576 (1.449 σ)	0.522 (1.233 σ)	3.684 (0.333 σ)
	automobile	1.011 (2.120 σ)	0.680 (2.386 σ)	0.724 (1.953 σ)	3.076 (0.333 σ)
	bird	1.017 (2.093 σ)	0.632 (1.287 σ)	0.608 (2.716 σ)	2.898 (0.500 σ)
	cat	1.010 (1.545 σ)	0.849 (1.624 σ)	0.864 (1.498 σ)	2.548 (0.500 σ)
	deer	1.019 (1.429 σ)	0.989 (1.348 σ)	1.025 (1.485 σ)	2.106 (0.485 σ)
	dog	1.008 (2.679 σ)	0.746 (1.977 σ)	0.809 (1.447 σ)	2.080 (0.372 σ)
	frog	1.025 (1.645 σ)	1.122 (1.760 σ)	1.252 (1.944 σ)	2.895 (0.998 σ)
	horse	1.008 (1.762 σ)	0.872 (2.660 σ)	0.903 (2.267 σ)	3.307 (0.655 σ)
	ship	1.011 (2.032 σ)	1.001 (1.805 σ)	1.020 (2.556 σ)	2.015 (0.809 σ)
	truck	1.014 (1.661 σ)	0.653 (1.785 σ)	0.741 (1.574 σ)	2.479 (0.483 σ)

The next two tables depict the performance and robustness metrics for our Lipschitz models that are using the Lipschitz constants derived from the last experiment.

Data Set	Normal Class	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0	90.9 ± 0.3	97.3 ± 0.8	96.7 ± 0.3	90.8 ± 5.0
	1	98.7 ± 0.0	97.6 ± 0.4	98.5 ± 0.6	98.3 ± 0.3
	2	81.6 ± 0.6	88.6 ± 3.0	82.8 ± 2.3	66.2 ± 7.2
	3	89.5 ± 0.1	87.3 ± 1.8	83.9 ± 1.2	74.8 ± 7.2
	4	86.6 ± 0.5	93.3 ± 1.0	90.7 ± 1.7	82.0 ± 3.6
	5	81.1 ± 0.8	84.5 ± 1.2	81.8 ± 2.2	63.9 ± 7.2
	6	82.8 ± 0.2	97.5 ± 0.6	94.7 ± 1.0	79.5 ± 3.4
	7	74.5 ± 0.5	90.0 ± 0.3	90.0 ± 2.3	89.2 ± 3.1
	8	87.1 ± 0.3	89.3 ± 2.5	85.5 ± 1.4	66.5 ± 3.4
	9	72.5 ± 0.3	92.4 ± 0.1	89.3 ± 0.3	80.0 ± 6.4
CIFAR10	airplane	65.7 ± 0.7	61.3 ± 1.3	64.8 ± 0.4	64.6 ± 2.7
	automobile	63.2 ± 0.4	63.2 ± 1.1	62.6 ± 1.5	41.6 ± 5.2
	bird	48.2 ± 0.7	50.4 ± 1.6	48.3 ± 1.7	57.6 ± 1.9
	cat	59.9 ± 0.4	56.9 ± 0.8	57.2 ± 1.1	55.0 ± 2.3
	deer	57.4 ± 1.1	57.0 ± 0.7	57.3 ± 3.4	72.9 ± 1.2
	dog	63.6 ± 0.8	62.2 ± 1.1	62.5 ± 0.4	53.1 ± 3.2
	frog	65.6 ± 0.6	69.2 ± 1.5	65.3 ± 1.8	69.3 ± 1.5
	horse	64.6 ± 0.1	62.7 ± 2.3	65.3 ± 2.3	55.0 ± 2.6
	ship	80.1 ± 0.3	73.0 ± 0.4	75.2 ± 0.4	62.0 ± 2.5
	truck	75.5 ± 0.1	71.4 ± 0.6	71.2 ± 0.2	55.7 ± 4.2

Data Set	Normal Class	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0	0.127 (2.129 σ)	0.256 (1.560 σ)	0.287 (2.315 σ)	0.075 (0.333 σ)
	1	0.165 (1.839 σ)	0.536 (1.880 σ)	0.658 (1.288 σ)	0.067 (0.500 σ)
	2	0.116 (1.759 σ)	0.332 (1.613 σ)	0.470 (1.933 σ)	0.080 (0.333 σ)
	3	0.119 (1.942 σ)	0.326 (1.530 σ)	0.461 (2.100 σ)	0.073 (0.816 σ)
	4	0.138 (1.398 σ)	0.349 (1.997 σ)	0.309 (1.706 σ)	0.087 (0.424 σ)
	5	0.115 (1.502 σ)	0.324 (1.825 σ)	0.383 (1.702 σ)	0.097 (0.500 σ)
	6	0.140 (1.934 σ)	0.304 (2.479 σ)	0.420 (1.929 σ)	0.098 (0.647 σ)
	7	0.144 (1.669 σ)	0.281 (2.792 σ)	0.259 (2.223 σ)	0.067 (0.500 σ)
	8	0.126 (2.018 σ)	0.303 (2.042 σ)	0.307 (2.175 σ)	0.072 (1.342 σ)
	9	0.144 (2.636 σ)	0.301 (1.616 σ)	0.241 (2.631 σ)	0.077 (1.000 σ)
CIFAR10	airplane	0.354 (1.920 σ)	0.138 (1.593 σ)	0.094 (1.693 σ)	0.104 (0.500 σ)
	automobile	0.431 (1.998 σ)	0.103 (1.710 σ)	0.093 (2.059 σ)	0.169 (0.655 σ)
	bird	0.300 (1.415 σ)	0.150 (2.522 σ)	0.111 (1.780 σ)	0.089 (0.571 σ)
	cat	0.448 (1.762 σ)	0.104 (1.863 σ)	0.080 (1.729 σ)	0.131 (0.333 σ)
	deer	0.434 (1.823 σ)	0.087 (1.445 σ)	0.066 (2.674 σ)	0.157 (0.333 σ)
	dog	0.445 (1.963 σ)	0.100 (2.117 σ)	0.074 (1.984 σ)	0.145 (0.627 σ)
	frog	0.436 (1.849 σ)	0.127 (2.197 σ)	0.100 (1.365 σ)	0.145 (0.655 σ)
	horse	0.385 (1.331 σ)	0.135 (1.765 σ)	0.122 (2.056 σ)	0.121 (0.500 σ)
	ship	0.457 (1.853 σ)	0.172 (1.560 σ)	0.152 (1.767 σ)	0.099 (0.997 σ)
	truck	0.398 (1.898 σ)	0.106 (1.690 σ)	0.087 (2.059 σ)	0.089 (0.655 σ)

5.2.2 Analysis

The mean ROC-AUC score over all normal classes attained by all standard models is listed in the table below.

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	93.4	93.9	93.9	84.6
CIFAR10	55.4	60.3	60.0	58.5

Below this is a table for the mean ROC-AUC for each Lipschitz continuous model.

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	84.5	91.8	89.4	79.1
CIFAR10	64.4	62.7	63.0	58.7

It can be clearly observed that the MNIST standard models perform better than the Lipschitz versions. This is to be expected based on Section 3.2, which states that robustness and accuracy are often somewhat at odds. In order to obtain robustness one often needs to make a trade-off for lower general performance. The MNIST DCAE performs 10.5% better than the corresponding Lipschitz DCAE, while the MNIST DeepSVDD Soft-Boundary and One-Class are only 2.3% and 5.0% better respectively. It is possible that the standard DCAE outperforms the Lipschitz version by such a large value due to the difference in architecture of their decoder. The difference in reconstruction quality can be visualized in Figure 5.7 and Figure 5.8, which depict reconstructions from a standard and Lipschitz DCAE respectively. The standard MNIST AnoGAN additionally performs 7.0% better than the MNIST Lipschitz AnoGAN, but being that the Lipschitz AnoGANs universally suffer from mode collapse, we do not find this statistic very persuasive.

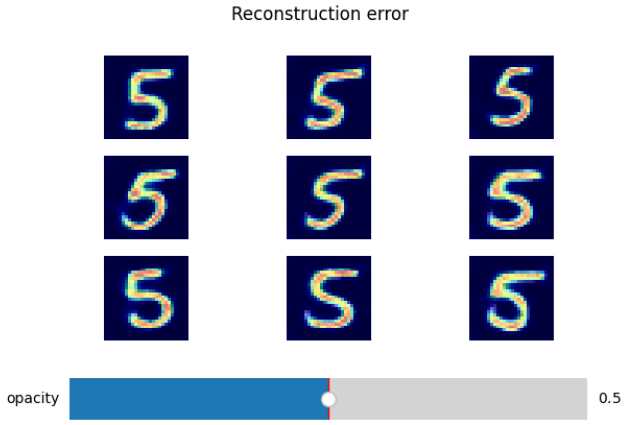

Figure 5.7: Nine most normal test samples for a standard DCAE trained on 5 with reconstructions overlaid at half opacity. The MNIST class 5 is the worst performing class across the board, simply due to the structure of the digit 5. One can notice nonetheless the clarity of the reconstructions.

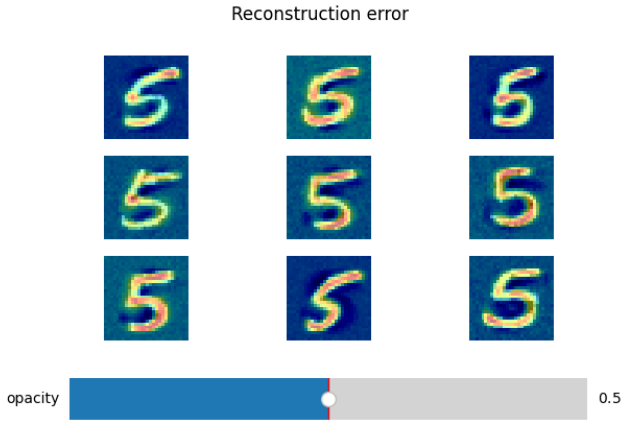

Figure 5.8: Nine most normal test samples for a Lipschitz DCAE trained on 5 with reconstructions overlaid at half opacity. Here the reconstructions are less clear and seemingly partially distributed into the background.

What is actually more interesting is the performance of the CIFAR10 models, as it seems that the Lipschitz versions essentially outperform the standard versions for almost every class. This is a rather surprising result as one would

expect a similar relationship as with the MNIST models, where a trade-off between standard and robust accuracy must be made. We find that the CIFAR10 Lipschitz DCAE performs 16.2% better than the standard version. While the CIFAR10 Lipschitz DeepSVDD Soft-Boundary is 4% better and One-Class is 5% better. The AnoGAN achieves practically the same mean ROC-AUC and as stated before, we find comparisons between the AnoGAN models to not be particularly meaningful.

Listed below is a table of percentage increases/descreases in performance from the standpoint of the Lipschitz models. For example a +10.5% indicates that the standard DCAE achieves a 10.5% higher ROC-AUC than the Lipschitz version. Similarly the CIFAR10 standard DCAE achieves a 14.0% lower ROC-AUC than the Lipschitz DCAE.

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	+10.5%	+2.3%	+5.0%	+7.0%
CIFAR10	-14.0%	-3.8%	-4.8%	-0.3%

The CIFAR10 Lipschitz DeepSVDD models achieve a relatively marginal increase in ROC-AUC, but the DCAE's performance increase is of particular note. The commonality between all of these models of course is the Lipschitz constraint, so we theorize that it's likely that by constraining the network it was able to become more robust to the larger variance in data that's present in the CIFAR10 data set. We cannot however ignore the possibility that the reason for the performance increase in the DCAE is the use of SpectralDense layers in the decoder. We think that it is possible that, for this particular data set, that the number of neurons in the dense layers was such a volume, that the model outperformed the standard convolutional variant, because it was able to better capture the complexity of the data. If this is the case, then one must note that dense layers, especially ones with as many neurons as those in the Lipschitz CIFAR10 decoder, are much more expensive to implement than convolutional layers. Unfortunately, due to the flaws in the Deel-Lip implementation of Lipschitz continuous convolutional layers, this is difficult to verify empirically.

Below is a table listing the maximum robustness metric calculated per standard model.

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	1.631	1.336	1.494	5.594
CIFAR10	1.032	1.122	1.252	3.684

Below is the same table for the Lipschitz continuous models.

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	0.165	0.536	0.658	0.098
CIFAR10	0.457	0.172	0.152	0.169

Here is a table detailing how many times higher the robustness metrics of the standard models were when compared to the Lipschitz models. For clarity's sake, we give the example of 8.24 which indicates that the standard DeepSVDD One-Class measured a lower bound on the Lipschitz constant that was 8.24 times higher than the Lipschitz variant.

Data Set	DCAE	DeepSVDD SB	DeepSVDD OC	AnoGAN
MNIST	9.88	2.49	2.27	57.08
CIFAR10	2.26	6.52	8.24	21.80

When we consider the possible robustness gains from our Lipschitz models we consider two aspects. The first of which is the provable robustness of Lipschitz models, which we have discussed in Section 3.3. Our Lipschitz models, assuming that the Deel-Lip layers contain no errors, are provably more robust than the standard models. This assertion does not require experiments to verify as it is a simple mathematical truth. The second aspect we consider is our robustness metric. We use this metric to gain insight into the possible magnitude of the robustness gain, so that we may in some way quantify the change. Of course we know that a loose lower bound on the Lipschitz constant of our Lipschitz continuous models is not strictly necessary, as we already know the exact global Lipschitz constant of these models. The purpose of the robustness metric is to provide a way to compare the robustness of the standard models with the Lipschitz models. As the standard model's Lipschitz constant is unconstrained, we can only use lower bounds. If we compare the highest lower bound calculated for the standard MNIST DCAE (1.631) with the highest calculated for the Lipschitz version (0.165), we find that the standard DCAEs lower bound is nearly ten times higher. Excluding the results of the AnoGAN, we find that generally the robustness metric measured by the Lipschitz models is between ten to two times lower than their standard counterparts. It seems quite clear that the Lipschitz models are much more robust than the non-Lipschitz models.

It is also to be considered that the potential robustness gains we discuss here are from models with global Lipschitz constants ranging from 2.0 to 100.0. For the Lipschitz DeepSVDD models for instance, we see a minimum of a two times lower robustness in comparison to the standard models. This "double robustness gain" is achieved by models with 50.0 and 100.0 Lipschitz constants. The potential robustness gains could be even higher, if the Lipschitz constant was optimized further. Most research and literature focuses on producing strictly 1-Lipschitz models. This is partially because, if you can build a 1-Lipschitz model,

you can build a K-Lipschitz model by simply scaling the outputs, but it's also due to the potential robustness gains of small constants like 1. As we have seen in the previous section, there is not necessarily a linear correlation between the Lipschitz constant and performance, as such there is a very high likelihood that much smaller Lipschitz constant could be chosen that more or less maintains the same performance.

It is difficult however to make sweeping declarative statements as to the exact magnitude of the robustness gain. This is due to the crudeness of our chosen metric. One may notice, that among the AnoGAN models, both standard and Lipschitz, that the robustness metric is often very few standard deviations from the mean. This is a positive aspect, as it indicates that the maximum lower bound was calculated in multiple of the ten total iterations, meaning enough time was allotted for the calculation of the bound to be reasonably accurate. An obvious issue that we have touched on already is the distance of the maximum lower bound from the mean for all the other models. For many of them the maximum value was over 2 standard deviations away, indicating a level of inaccuracy. This does not necessarily imply that the robustness metrics that have high distance from the mean are incorrect, but it implies that this particular maximum value was only computed in one or a few of the 10 total iterations. The higher the distance from the mean, the more likely there is another maximum lower bound for the Lipschitz constant. For the amount of data and the number of possible random combinations to be calculated, ten iterations of 15 seconds was likely insufficient to achieve the most accurate robustness metric. We also would like to remind the reader, that we had noticed in the Section 5.1.2, that the robustness metric for larger Lipschitz constants was strangely low compared to the metrics calculated for smaller Lipschitz constants. For the DCAEs this is less of a concern, but for the DeepSVDD models, it is a possibility that their recorded robustness metrics are much lower than they should be. The consequence of this would be that the mentioned robustness gains in the table above would in actuality be smaller. As such we must state that although the robustness gains made by the Lipschitz models are rather clear, there is a certain amount of untrustworthiness for our robustness metric. For this reason we supply the amount of standard deviations from the mean for each value, for the sake of clarity and honesty.

Moreover our robustness metric is calculated, as stated at the beginning of the chapter, with test data from the very same test data set that the model is tested on. This is also a particular weakness of our robustness metric. As we calculate it only with test data, this measures the robustness of the model and calculates a lower Lipschitz bound essentially for only normal and anomalous data. If we had instead generated images randomly, fed these images through our models to generate an output and used this data for the calculation, we could have possibly developed a stronger metric. Generating random images would have pushed our model to classify extremely anomalous inputs as well

as inputs possibly resembling adversarial examples. This sort of uncertainty is exactly what we want a robust model to be resistant to. As such the robustness metrics calculated are less expressive than we would have preferred.

In a similar vein, for both standard and Lipschitz AnoGANs, the number of standard deviations that the maximum calculated lower bound is from the mean is low across the board. Other than a few outliers, many of the calculated bounds are less than a standard deviation from the mean. Upon further investigation we find that this is caused by the list of 10 robustness values (one from each of the 15 second iterations). This list is often populated with the same value multiple times, meaning that in each 15 second iteration, the same max value is found. This is certainly due to the small size of the test data set. We limited the size of the test data set to 1,000 due to the high computational time of the AnoGAN algorithm. This has the effect of making it highly probable that in the 15 second timespan given, that the highest value can be randomly calculated. With only 1,000 data points, there are only 1 million possible unique combinations of two. Whereas for the other models with 10,000 test samples, the number of combinations is 100 million. We want to make clear that the more consistent robustness metrics are not a property of the AnoGAN algorithm or GANs in general and are simply due to the concessions that we had to make for the sake of computation time.

It is also quite noticeable that the robustness metric for standard AnoGANs is high across the board while being very low for Lipschitz AnoGANs. The explanation is rather simple. For the Lipschitz AnoGANs, they suffer from mode collapse and produce the same generated image no matter the input noise. The consequence of this is that when we calculate the robustness metric, the difference between two ”outputs”, i.e. two generated images is minuscule. When we divide the difference between two generated images by the difference between their latent noise we receive an extremely low robustness metric because the two generated images are nearly identical, so an already extremely small value is divided by another value to become even smaller. For the standard AnoGAN, the difference between generated images will actually be quite large, which leads to the higher robustness metric. This large value is made larger by the division by the difference between the two latent noise values, which often becomes a much smaller value than the difference between input samples in the DCAE for instance. Due to these factors, the lower bound on the Lipschitz constant is often much larger for standard AnoGAN models.

As with the previous experiment, the Lipschitz constrained AnoGAN models perform rather well despite mode collapse. One can notice that for the MNIST models, that normal classes such as 0, 1, or 7 seem to perform best, which again aligns with our expectations. Some classes are simply easier to approximate, while others such as 5 or bird from CIFAR10 are harder. The performance of the Lipschitz CIFAR10 AnoGAN is about on par with the standard version. One of the major differences between the standard and Lipschitz AnoGANs how-

ever is the standard deviations of their ROC-AUCs. The Lipschitz continuous AnoGANs have much higher standard deviations, which agrees with our expectations. A GAN suffering from mode collapse should naturally be more unstable and thus should exhibit more variance in its performance statistics.

The failure of the DCGANs is certainly regrettable, but we would like to posit that it was unknown whether or not the AnoGAN models would have actually been Lipschitz continuous even if they did not suffer from mode collapse. In order to ensure Lipschitz continuity, we used Deel-Lip layers that constrain the global Lipschitz constant of the model by constraining the constant of each individual layer. Thus we applied Deel-Lip layers to the DCGAN to create a Lipschitz continuous architecture. If the DCGAN had not suffered from mode collapse it is possible that it would've been Lipschitz continuous. However the AnoGAN may not have been. The AnoGAN uses the trained DCGAN for the residual and discriminator loss, but the AnoGAN process of iteratively finding a reverse mapping for a sample, is not itself constrained by any Lipschitz constant. If a sample was given as input to the Discriminator of the DCGAN, one could expect the output variation to be limited by the Lipschitz constant. A similar expectation can be made of the input noise fed into the DCGAN Generator. For each input sample for the AnoGAN, however, the output is a latent value that generates an image that is supposed to be as similar an image to the input sample as the generator can make. This iterative process of finding a latent value that generates an image within a certain neighborhood of the given input sample is represented by the residual loss (2.10), which measures the difference between the input sample and the image generated from the latent value. As such one can imagine the training process for the AnoGAN reverse mapping as a latent value slowly generating images that lay closer and closer to the input sample within image space. Ideally the latent value eventually generates an image that lays within some "neighborhood" of the sample image and any further changes to the latent value only change the generated image by a minute amount.

This relationship is unconstrained as far as Lipschitz continuity is concerned. What we mean by this is that the size of the step that can be made each iteration towards the neighborhood of the input sample is unconstrained. Both the change in the latent value and the "distance" that the generated image moves are not affected by the Lipschitz constant of the DCGAN at all. For this reason it would have been difficult to ensure the Lipschitz continuity of the model with the tools from Deel-Lip alone. As such, even if the AnoGAN had not suffered mode collapse, it was likely that it would be difficult to prove its Lipschitz continuity .

What becomes clear after these experiments is that, generally, Lipschitz models for anomaly detection on MNIST digits seem to perform worse than standard models but are much more robust. Often these Lipschitz models have relatively high Lipschitz constants too. Obviously a Lipschitz constant can be

arbitrarily high, but as seen in our first experiment, very reasonable results are achieved already at Lipschitz constants of 5.0 or 10.0. For the DeepSVDD models we ended up choosing constant of 50.0 for the MNIST data set and can likely be lowered. As we found in the first experiment, there seems to not be a linear correlation between the Lipschitz constant of a model and its performance. As such there is likely a constant smaller than 50.0 that achieves near the same performance, but is potentially much more robust. An even more interesting finding is that our Lipschitz continuous CIFAR10 models are not only more robust than the standard models, but on average perform better as well. Even if they had performed the same, this would have been quite interesting, as there are often trade-offs and costs associated with robustness. Our findings show that for the CIFAR10 models, the robustness gained is potentially not only free, but possibly actually improves general performance. Of course our experiments are rather limited in scope and our robustness metric could be more persuasive, but our data seems to suggest that the Lipschitz CIFAR10 models are better across the board in comparison to the standard models. This would make Lipschitz continuity a genuine consideration and a promising prospect for improving the robustness of certain anomaly detection models.

6 Improvements and Future Work

Through these experiments we have found many areas where potential improvements could be made if one wanted to make further research on the topic. The first and likely most major improvement would be to make our Lipschitz continuous architectures fully gradient norm preserving. We detailed at length in Section 3.3 the importance of gradient norm preservation in 1-Lipschitz models. We have also already detailed how Deel-Lip preserves the gradient norm through Björck Orthonormalization. This however only preserves the norm for the weights, i.e. the layers. The activation functions must also be gradient norm preserving for an entire model to be GNP. We use throughout most of our models ReLU and LeakyReLU, which are gradient norm preserving, but only for positive inputs [40] [3]. This means, that in order for ReLU activations to be gradient norm preserving they essentially have to give up their nonlinearity. This is suboptimal and the first improvement that could be made to our Lipschitz models is replacing, where possible, ReLU and LeakyReLU activations with GNP activations such as GroupSort or MaxMin, which are offered by Deel-Lip. GroupSort separates the output of a layer into groups and then sorts these groups in ascending order. MaxMin is simply a special case of GroupSort where the group size is two. These activations are both GNP and 1-Lipschitz [3] [62]. An additional benefit of GNP architectures is dynamical isometry [78]. This describes the phenomenon where the distribution of values in the input-output Jacobian of the model are all close to zero [3]. The input-output Jacobian is simply a matrix that contains the derivative of every dimension of the output with respect to every feature in the input. Essentially it is a matrix containing the first-order derivatives of a vector-valued function, in this case the vector valued function being the neural network. The benefits of dynamical isometry are improved training stability as well as speed [78].

A second improvement would be using a different GAN-based anomaly detection algorithm. The most egregious weakness of the AnoGAN algorithm is the test-time performance, as it in our case required 500 backpropagation steps per test sample. It has also been remarked that the DCGAN used for AnoGAN has the same objective as any standard GAN, and that improvements could also be made by altering the objective to take into account the eventual reverse mapping that must inevitably occur [44]. Possible improvements on the AnoGAN have been made in the form of the EGBAD [81] and the GANomaly [1], both of which are inspired by AnoGAN, but seek to solve it's inefficiencies. EGBAD

seeks to learn an encoder that can map samples directly to their latent representation during training. Whereas GANomaly uses an Autoencoder to learn the reverse mapping. We do note, however that the GANomaly algorithm is semi-supervised, which is a departure from our unsupervised algorithms.

As we remarked in the last section, our robustness metric could have been calculated differently. We could have generated random images or even specifically generated adversarial samples with which we could have used to derive a lower Lipschitz bound. This would likely be the easiest improvement to make on the robustness metric. Similarly, more metrics could have been calculated. As our results stand, our CIFAR10 Lipschitz models seem to outperform the standard models with respect to both of our metrics. We have, however no data on computational time. It could be possible that the Lipschitz neural networks take much longer to achieve their improved results, in fact this is something that was unfortunately not recorded in the experiments but was observed as the models were being refined. Generally Deel-Lip enhanced models seemed to take longer to train and test.

Another rather obvious improvement would be to test on more complex data sets. MNIST and CIFAR10 are rather simple data sets and are used for benchmarking models and for research. Their data is not well representative of reality. Larger data sets such as ImageNet have much larger variability in their data and are thus much more useful for understanding how Lipschitz continuity could be incorporated into real models that must interact with real data [37]. This would also aid in confirming or refuting our results on the CIFAR10 dataset that seems to suggest that Lipschitz models perform slightly better on complex data than standard models. It has been found that very high accuracy can be achieved by 1-Lipschitz models on data sets such as CIFAR100 [7], which individually has images of the same complexity as CIFAR10, but in total has 100 classes (20 superclasses), in comparison to CIFAR10's 10 classes. This high accuracy apparently strongly depends upon the chosen loss of the 1-Lipschitz model and that loss' hyperparameter choice. For our experiments we sought to create Lipschitz versions of anomaly detection algorithms, often restricting our ability to change the loss. Changing the loss for DeepSVDD or AnoGAN for example would have essentially led to the creation of a different algorithm. We wanted to see what could be done by implementing a Lipschitz continuous architecture. It could be prudent, however, to change or amend these losses for the sake of making a more Lipschitz compatible model.

There have also been improvements on the DeepSVDD algorithm [15], which seek to alleviate the strict architectural restrictions imposed by the original paper. These restrictions are necessary to prevent hypersphere collapse, but may cause the model to learn less than optimal features and limits the adaptability of the model. Chong et. al. seek to remove their necessity through regularization [15]. In their research they propose two possible regularized versions of DeepSVDD, one that injects random noise and another that penalizes the loss

when the variance of the mini-batch falls too low.

All of these are possible improvements that could be made to our experiments that have the potential to lead to even more promising findings.

7 Conclusion

In this work we have implemented and explored the functionality of three popular methods for unsupervised anomaly detection. We have explained in detail the core idea behind each of these algorithms and measured the effects of Lipschitz continuity on their performance. We explain in detail the theoretical gains offered by Lipschitz continuity and find with our experiments that Lipschitz continuous models have advantages and drawbacks. We find the choice of Lipschitz constant to be a vital hyperparameter in determining both the performance and robustness of the model. We additionally find that Lipschitz continuous models often must sacrifice a portion of their standard performance for the sake of improved robustness. This performance loss ranges however from moderate to minor, whereas the robustness gains are in all cases large. We also find that, in our experiments, Lipschitz continuous models seem to perform better on more complex and varied CIFAR10 data than standard non-Lipschitz constrained models. We additionally identify that a flaw exists within the Deel-Lip implementation of the SpectralConv2D layer, as its use led to unrecognizable reconstructions in DCAE and GAN models.

We find that Deep Convolutional Autoencoders seem to suffer both the greatest performance loss (MNIST DCAE) and gain (CIFAR10 DCAE) when Lipschitz-enhanced, though we do posit that this is possibly not due to the Lipschitz constraint itself, but rather the use of dense layers in the decoder. We also observe that both DeepSVDD variants seem to perform well under a Lipschitz constraint, but suffer majorly if the Lipschitz constant is too small. This is however, likely due to the initialization of the hypersphere center. This indicates that it may be possible for these models to perform well under a tighter Lipschitz constraint if special care is taken to tweak the initialization of the hypersphere center. Although our experiments to enhance an AnoGAN with Lipschitz continuity end in mode collapse for every model, we do identify one of the larger drawbacks to Lipschitz continuous models, namely the architectural constraints. We are confident that the cause of the mode collapse is linked to a lack of Batch Normalization layers in the model, which unaltered cannot be implemented in a Lipschitz continuous model. We also explain how a Lipschitz continuous architecture may not always be sufficient to ensure Lipschitz continuity, so while libraries like Deel-Lip are immensely helpful, one must consider the Lipschitz implementation of each algorithm carefully.

From our results we can conclude that Lipschitz continuity is a promising method for obtaining provable robustness while maintaining a reasonable performance. We do note that obtaining Lipschitz continuity through purely ar-

chitectural constraints, while a proven way to enforce a global Lipschitz constant for certain models, can also cause friction with certain anomaly detection algorithms and may not even be enough to ensure Lipschitz continuity for others. Thus we find that Lipschitz continuous models suffer from a difficulty to be implemented correctly. However we believe that if these hurdles can be overcome, that Lipschitz continuity may be a favorable choice as far as attack-agnostic defenses and provable robustness are concerned. We identify and list a myriad of improvements that could be made to both our models and our approach to the experiments. We note that many of these improvements have a strong potential to lead to even more convincing and persuasive results than we have found. We ask anyone seeking to further our work to take them into consideration. As anomaly detection models become more enmeshed with our everyday life, it is imperative that these models are robust to the inherent uncertainties and inherent variability of real data, as they are often deployed in safety-critical domains. We see Lipschitz continuous architectures, when they can be applied, as a way to provide said robustness with a reasonable trade-off.

8 Appendix

Figure 8.1: Standard AnoGAN trained on 9 as the normal class. This depicts 64 images that were generated after giving the Generator 64 random noise values.

Figure 8.2: Visualization of the residual error calculated by an AnoGAN trained on 9 as the normal class. The left image is the test sample, the middle is the residual error, and the right is the generated image.

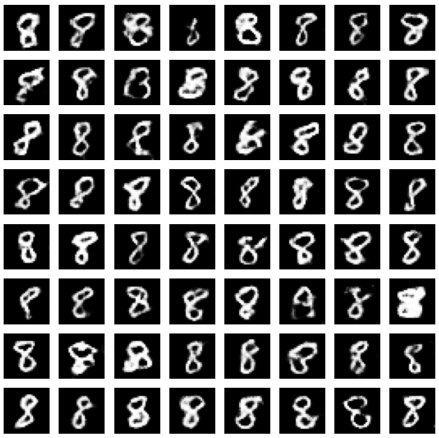

Figure 8.3: Standard AnoGAN trained on 8 as the normal class. This is one of the classes that achieved a relatively low performance, nonetheless the digit 8 can be clearly recognized in the majority of the images. There are a few, however, that do stand out as particularly poor reconstructions

Figure 8.4: Standard AnoGAN trained on the normal class deer. This is one of the better performing classes for the CIFAR10 AnoGANs. The deer in the images can be relatively difficult to identify, but a basic structure can be found in many.

Figure 8.5: Visualization of the residual error calculated by an AnoGAN trained on deer as the normal class. The left image is the test sample, the middle is the residual error, and the right is the generated image.

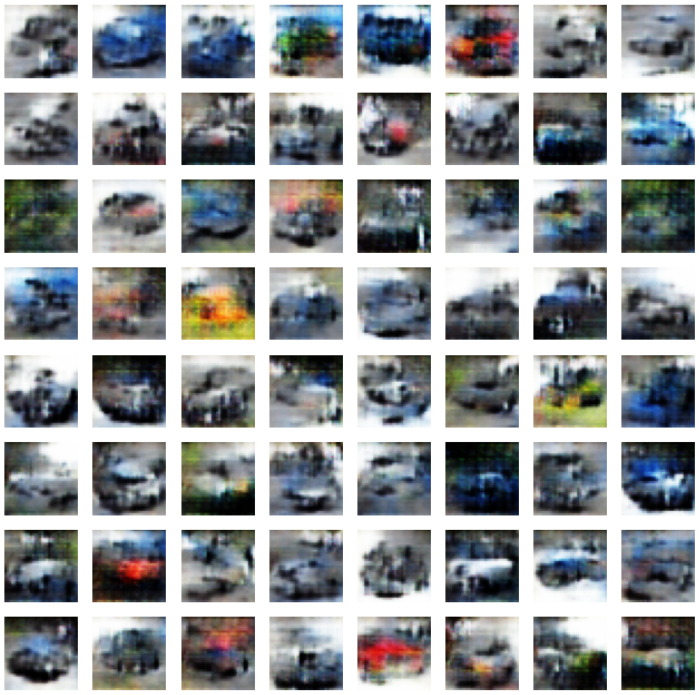

Figure 8.6: Standard AnoGAN trained on the normal class car. This is one of the worse performing classes for the CIFAR10 AnoGAN, however the basic structure of a car can still be identified.

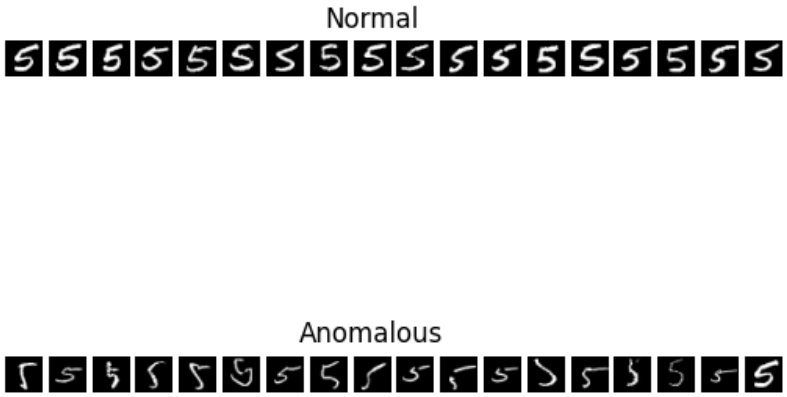

Figure 8.7: In class ranking of the 18 most normal and 18 most anomalous samples for a Lipschitz DCAE trained on 5 as the normal class. An in class ranking is defined as a ranking of samples consisting only of the normal/trained class. This serves to give us an idea of how accurate the Autoencoder is at identifying anomalous images. The anomalous images are seemingly fittingly ranked as anomalous.

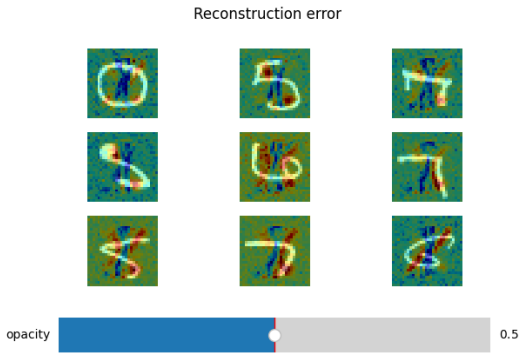

Figure 8.8: Nine most anomalous test samples for a Lipschitz DCAE trained on 1 with reconstructions overlaid at half opacity. The reconstruction error present in these images is very large in comparison to other DCAE models. The number 1 obviously is one of the most static digits, it can simply be approximated by a vertical line. For this reason models with 1 as a normal class often perform well.

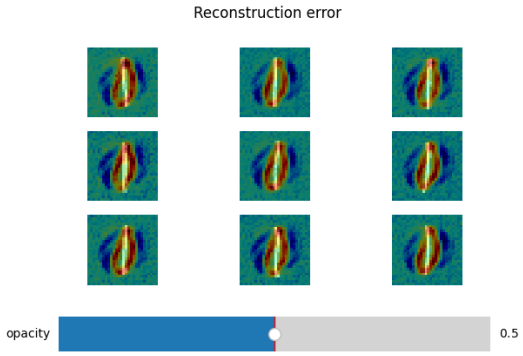

Figure 8.9: Nine most anomalous test samples for a Lipschitz DCAE trained on 0 with reconstructions overlaid at half opacity. It is interesting to see here how the mode tries to approximate the one by creating what is essentially a very horizontally squished zero.

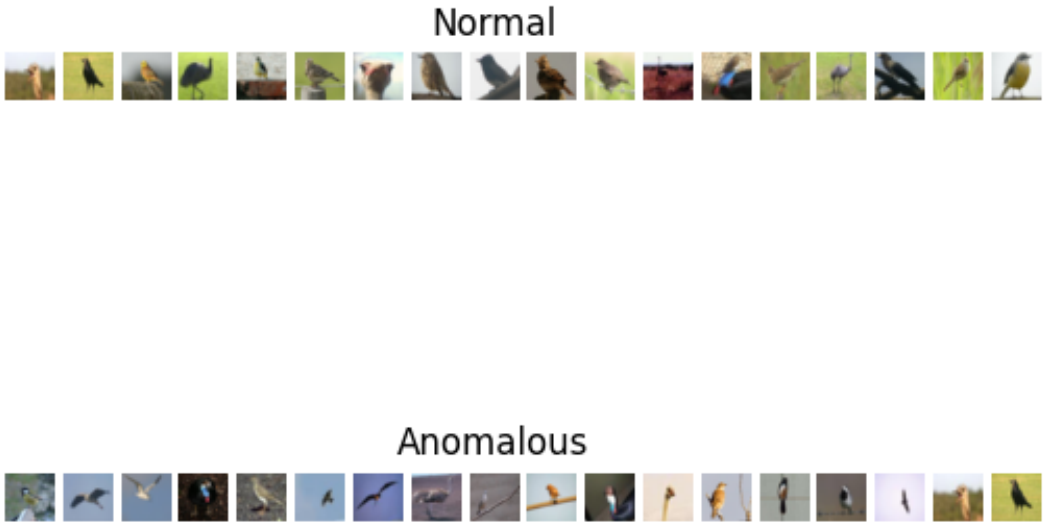

Figure 8.10: In class ranking of the 18 most normal and 18 most anomalous samples for a Lipschitz DCAE trained on birds as the normal class. The most normal images share a common perspective. The images with birds further in the distance are deemed anomalous.

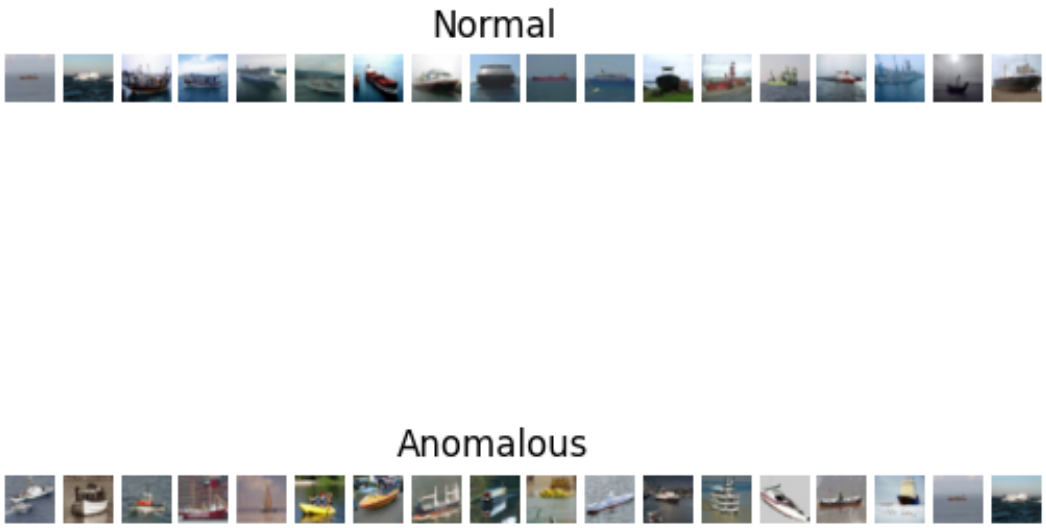

Figure 8.11: In class ranking of the 18 most normal and 18 most anomalous samples for a Lipschitz DCAE trained on ships as the normal class. The most normal images appear to be those where approximately the bottom half of the image is water and the top half is sky. Perspective and color seem to be the deciding factors here.

Bibliography

[1] Samet Akcay, Amir Atapour Abarghouei, and Toby P. Breckon. 2018. GANomaly: Semi-Supervised Anomaly Detection via Adversarial Training. *CoRR* abs/1805.06725 (2018). arXiv:1805.06725 http://arxiv.org/abs/1805.06725

[2] Jerone Andrews, Edward Morton, and Lewis Griffin. 2016. Detecting anomalous data using auto-encoders. *International Journal of Machine Learning and Computing* 6 (01 2016), 21.

[3] Cem Anil, James Lucas, and Roger B. Grosse. 2018. Sorting out Lipschitz function approximation. *CoRR* abs/1811.05381 (2018). arXiv:1811.05381 http://arxiv.org/abs/1811.05381

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

[5] Anish Athalye, Nicholas Carlini, and David A. Wagner. 2018. Obfuscated Gradients Give a False Sense of Security: Circumventing Defenses to Adversarial Examples. *CoRR* abs/1802.00420 (2018). arXiv:1802.00420 http://arxiv.org/abs/1802.00420

[6] Sihem Baccari, Mohamed Hadded, Hakim Ghazzai, Haifa Touati, and Mourad Elhadef. 2024. Anomaly Detection in Connected and Autonomous Vehicles: A Survey, Analysis, and Research Challenges. *IEEE Access* 12 (2024), 19250–19276. https://doi.org/10.1109/ACCESS.2024.3361829

[7] Louis Béthune, Alberto González-Sanz, Franck Mamalet, and Mathieu Serrurier. 2021. The Many Faces of 1-Lipschitz Neural Networks. *CoRR* abs/2104.05097 (2021). arXiv:2104.05097 https://arxiv.org/abs/2104.05097

[8] Å. Björck and C. Bowie. 1971. An Iterative Algorithm for Computing the Best Estimate of an Orthogonal Matrix. *SIAM J. Numer. Anal.* 8, 2 (1971), 358–364. https://doi.org/10.1137/0708036 arXiv:https://doi.org/10.1137/0708036

[9] Richard J. Bolton and David J. Hand. 2002. Unsupervised Profiling Methods for Fraud Detection. https://api.semanticscholar.org/CorpusID:14365948

[10] Nicholas Carlini and David A. Wagner. 2016. Defensive Distillation is Not Robust to Adversarial Examples. *CoRR* abs/1607.04311 (2016). arXiv:1607.04311 http://arxiv.org/abs/1607.04311

[11] Nicholas Carlini and David A. Wagner. 2017. Adversarial Examples Are Not Easily Detected: Bypassing Ten Detection Methods. *CoRR* abs/1705.07263 (2017). arXiv:1705.07263 http://arxiv.org/abs/1705.07263

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Comput. Surv.* 41, 3, Article 15 (jul 2009), 58 pages. https://doi.org/10.1145/1541880.1541882

[13] Olivier Chapelle, Bernhard Schölkopf, and Alexander Zien. 2006. *Semi-Supervised Learning*. The MIT Press. https://doi.org/10.7551/mitpress/9780262033589.001.0001

[14] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. 2018. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*. 1–5. https://doi.org/10.1109/WTS.2018.8363930

[15] Penny Chong, Lukas Ruff, Marius Kloft, and Alexander Binder. 2020. Simple and Effective Prevention of Mode Collapse in Deep One-Class Classification. *CoRR* abs/2001.08873 (2020). arXiv:2001.08873 https://arxiv.org/abs/2001.08873

[16] Moustapha Cisse, Piotr Bojanowski, Edouard Grave, Yann Dauphin, and Nicolas Usunier. 2017. Parseval Networks: Improving Robustness to Adversarial Examples. arXiv:1704.08847 [stat.ML]

[17] Jeremy Cohen, Elan Rosenfeld, and J. Zico Kolter. 2019. Certified Adversarial Robustness via Randomized Smoothing. *CoRR* abs/1902.02918 (2019). arXiv:1902.02918 http://arxiv.org/abs/1902.02918

[18] Corinna Cortes and Vladimir Naumovich Vapnik. 2004. Support-vector networks. *Machine Learning* 20 (2004), 273–297. https://api.semanticscholar.org/CorpusID:206787478

[19] Vincent Dumoulin and Francesco Visin. 2018. A guide to convolution arithmetic for deep learning. arXiv:1603.07285 [stat.ML]

[20] Eleazar Eskin, Andrew Arnold, Michael Prerau, Leonid Portnoy, and Sal Stolfo. 2002. *A Geometric Framework for Unsupervised Anomaly Detection*. Springer US, Boston, MA, 77–101. https://doi.org/10.1007/978-1-4615-0953-0_4

[21] Christian Etmann, Sebastian Lunz, Peter Maass, and Carola-Bibiane Schönlieb. 2019. On the Connection Between Adversarial Robustness and Saliency Map Interpretability. arXiv:1905.04172 [stat.ML]

[22] Kunihiko Fukushima. 1969. Visual Feature Extraction by a Multilayered Network of Analog Threshold Elements. *IEEE Transactions on Systems Science and Cybernetics* 5, 4 (1969), 322–333. https://doi.org/10.1109/TSSC.1969.300225

[23] Gene H. Golub and Henk A. van der Vorst. 2000. Eigenvalue computation in the 20th century. *J. Comput. Appl. Math.* 123, 1-2 (Nov. 2000), 35–65. https://doi.org/10.1016/S0377-0427(00)00413-1

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[25] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative Adversarial Networks. arXiv:1406.2661 [stat.ML]

[26] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. arXiv:1412.6572 [stat.ML]

[27] Adam Goodge, Bryan Hooi, See Kiong Ng, and Wee Siong Ng. 2020. Robustness of Autoencoders for Anomaly Detection Under Adversarial Impact. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, Christian Bessiere (Ed.). International Joint Conferences on Artificial Intelligence Organization, 1244–1250. https://doi.org/10.24963/ijcai.2020/173 Main track.

[28] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. 2017. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *CoRR* abs/1708.06733 (2017). arXiv:1708.06733 http://arxiv.org/abs/1708.06733

[29] Matthias Hein and Maksym Andriushchenko. 2017. Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation. *CoRR* abs/1705.08475 (2017). arXiv:1705.08475 http://arxiv.org/abs/1705.08475

[30] Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2 (1989), 359–366. https://api.semanticscholar.org/CorpusID:2757547

[31] Byungho Hwang and Sungzoon Cho. 1999. Characteristics of auto-associative MLP as a novelty detector. In *IJCNN'99. International Joint*

Conference on Neural Networks. Proceedings (Cat. No.99CH36339), Vol. 5. 3086–3091 vol.5. https://doi.org/10.1109/IJCNN.1999.836051

[32] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *CoRR* abs/1502.03167 (2015). arXiv:1502.03167 http://arxiv.org/abs/1502.03167

[33] Nathalie Japkowicz, Catherine Myers, and Mark Gluck. 1995. A novelty detection approach to classification. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 1* (Montreal, Quebec, Canada) *(IJCAI'95)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 518–523.

[34] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

[35] M. A. Kramer. [n. d.]. Autoassociative neural networks. , 313-328 pages. Issue 4. https://doi.org/10.1016/0098-1354(92)80051-A

[36] Mark A. Kramer. 1991. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal* 37, 2 (1991), 233–243. https://doi.org/10.1002/aic.690370209 arXiv:https://aiche.onlinelibrary.wiley.com/doi/pdf/10.1002/aic.690370209

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (Eds.), Vol. 25. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[38] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2016. Adversarial examples in the physical world. *CoRR* abs/1607.02533 (2016). arXiv:1607.02533 http://arxiv.org/abs/1607.02533

[39] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324. https://doi.org/10.1109/5.726791

[40] Qiyang Li, Saminul Haque, Cem Anil, James Lucas, Roger B. Grosse, and Jörn-Henrik Jacobsen. 2019. Preventing Gradient Attenuation in Lipschitz Constrained Convolutional Networks. *CoRR* abs/1911.00937 (2019). arXiv:1911.00937 http://arxiv.org/abs/1911.00937

[41] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. 2017. Understanding the Effective Receptive Field in Deep Convolutional Neural Networks. arXiv:1701.04128 [cs.CV]

[42] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2019. Towards Deep Learning Models Resistant to Adversarial Attacks. arXiv:1706.06083 [stat.ML]

[43] Jonathan Masci, Ueli Meier, Dan Ciresan, and Jürgen Schmidhuber. 2011. Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction. 52–59. https://doi.org/10.1007/978-3-642-21735-7_7

[44] Federico Di Mattia, Paolo Galeone, Michele De Simoni, and Emanuele Ghelfi. 2019. A Survey on GANs for Anomaly Detection. *CoRR* abs/1906.11632 (2019). arXiv:1906.11632 http://arxiv.org/abs/1906.11632

[45] Luke Metz, Ben Poole, David Pfau, and Jascha Sohl-Dickstein. 2016. Unrolled Generative Adversarial Networks. *CoRR* abs/1611.02163 (2016). arXiv:1611.02163 http://arxiv.org/abs/1611.02163

[46] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. 2018. Spectral Normalization for Generative Adversarial Networks. *CoRR* abs/1802.05957 (2018). arXiv:1802.05957 http://arxiv.org/abs/1802.05957

[47] Christoph Molnar. 2022. *Interpretable Machine Learning* (2 ed.). https://christophm.github.io/interpretable-ml-book

[48] M M Moya, M W Koch, and L D Hostetler. 1993. One-class classifier networks for target recognition applications. (1 1993). https://www.osti.gov/biblio/6755553

[49] Hajime Ono, Tsubasa Takahashi, and Kazuya Kakizaki. 2018. Lightweight Lipschitz Margin Training for Certified Defense against Adversarial Examples. *CoRR* abs/1811.08080 (2018). arXiv:1811.08080 http://arxiv.org/abs/1811.08080

[50] Martin J. Osborne and Ariel Rubinstein. 1994. *A course in game theory*. The MIT Press, Cambridge, USA. electronic edition.

[51] Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. 2016. Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples. *CoRR* abs/1602.02697 (2016). arXiv:1602.02697 http://arxiv.org/abs/1602.02697

[52] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. 2015. The Limitations of Deep Learning in Adversarial Settings. *CoRR* abs/1511.07528 (2015). arXiv:1511.07528 http://arxiv.org/abs/1511.07528

[53] Lucas Pinheiro Cinelli, Matheus Araújo Marins, Eduardo Antúnio Barros da Silva, and Sérgio Lima Netto. 2021. *Variational Autoencoder.* Springer International Publishing, Cham, 111–149. https://doi.org/10.1007/978-3-030-70679-1_5

[54] Alec Radford, Luke Metz, and Soumith Chintala. 2016. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. arXiv:1511.06434 [cs.LG]

[55] Russell D. Reed and Robert J. Marks. 1998. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks.* MIT Press, Cambridge, MA, USA.

[56] Lukas Ruff, Robert Vandermeulen, Nico Goernitz, Lucas Deecke, Shoaib Ahmed Siddiqui, Alexander Binder, Emmanuel Müller, and Marius Kloft. 2018. Deep One-Class Classification. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 4393–4402. https://proceedings.mlr.press/v80/ruff18a.html

[57] Mayu Sakurada and Takehisa Yairi. 2014. Anomaly Detection Using Autoencoders with Nonlinear Dimensionality Reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis* (Gold Coast, Australia QLD, Australia) *(MLSDA'14)*. Association for Computing Machinery, New York, NY, USA, 4–11. https://doi.org/10.1145/2689746.2689747

[58] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved Techniques for Training GANs. *CoRR* abs/1606.03498 (2016). arXiv:1606.03498 http://arxiv.org/abs/1606.03498

[59] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. 2017. Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery. *CoRR* abs/1703.05921 (2017). arXiv:1703.05921 http://arxiv.org/abs/1703.05921

[60] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. 2018. Adversarially Robust Generalization Requires More Data. *CoRR* abs/1804.11285 (2018). arXiv:1804.11285 http://arxiv.org/abs/1804.11285

[61] Bernhard Schölkopf, John Platt, John Shawe-Taylor, Alexander Smola, and Robert Williamson. 2001. Estimating Support of a High-Dimensional Distribution. *Neural Computation* 13 (07 2001), 1443–1471. https://doi.org/10.1162/089976601750264965

[62] Mathieu Serrurier, Franck Mamalet, Alberto González-Sanz, Thibaut Boissin, Jean-Michel Loubes, and Eustasio del Barrio. 2020. Achieving robustness in classification using optimal transport with hinge regularization. *CoRR* abs/2006.06520 (2020). arXiv:2006.06520 https://arxiv.org/abs/2006.06520

[63] Hidetoshi Shimodaira. 2000. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference* 90 (10 2000), 227–244. https://doi.org/10.1016/S0378-3758(00)00115-4

[64] Irwin Sobel. 2014. An Isotropic 3x3 Image Gradient Operator. *Presentation at Stanford A.I. Project 1968* (02 2014).

[65] Jure Sokolic, Raja Giryes, Guillermo Sapiro, and Miguel R. D. Rodrigues. 2017. Robust Large Margin Deep Neural Networks. *IEEE Transactions on Signal Processing* 65, 16 (Aug. 2017), 4265–4280. https://doi.org/10.1109/tsp.2017.2708039

[66] C. Spence, L. Parra, and P. Sajda. 2001. Detection, synthesis and compression in mammographic image analysis with a hierarchical image probability model. In *Proceedings IEEE Workshop on Mathematical Methods in Biomedical Image Analysis (MMBIA 2001)*. 3–10. https://doi.org/10.1109/MMBIA.2001.991693

[67] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2015. Striving for Simplicity: The All Convolutional Net. arXiv:1412.6806 [cs.LG]

[68] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing properties of neural networks. arXiv:1312.6199 [cs.CV]

[69] Marc Tanti. 2023. Generative Adversarial Network illustration.svg. https://en.wikipedia.org/wiki/File:Generative_Adversarial_Network_illustration.svg Figure by Marc Tanti from the Wikimedia Commons under the public creative commons license (https://creativecommons.org/licenses/by-sa/4.0/deed.en).

[70] David Tax and Robert Duin. 2004. Support Vector Data Description. *Machine Learning* 54 (01 2004), 45–66. https://doi.org/10.1023/B:MACH.0000008084.60811.49

[71] Hoang Thanh-Tung, Truyen Tran, and Svetha Venkatesh. 2018. On catastrophic forgetting and mode collapse in Generative Adversarial Networks. *CoRR* abs/1807.04015 (2018). arXiv:1807.04015 http://arxiv.org/abs/1807.04015

[72] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. 2019. Robustness may be at odds with accuracy. *arXiv.org* (Sep 2019). https://arxiv.org/abs/1805.12152

[73] Yusuke Tsuzuku, Issei Sato, and Masashi Sugiyama. 2018. Lipschitz-Margin Training: Scalable Certification of Perturbation Invariance for Deep Neural Networks. *CoRR* abs/1802.04034 (2018). arXiv:1802.04034 http://arxiv.org/abs/1802.04034

[74] Laurens van der Maaten, Eric Postma, and H. Herik. 2007. Dimensionality Reduction: A Comparative Review. *Journal of Machine Learning Research - JMLR* 10 (01 2007).

[75] Aladin Virmaux and Kevin Scaman. 2018. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2018/file/d54e99a6c03704e95e6965532dec148b-Paper.pdf

[76] Zifan Wang, Matt Fredrikson, and Anupam Datta. 2021. Boundary Attributions Provide Normal (Vector) Explanations. *CoRR* abs/2103.11257 (2021). arXiv:2103.11257 https://arxiv.org/abs/2103.11257

[77] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. 2018. Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. arXiv:1801.10578 [stat.ML]

[78] Lechao Xiao, Yasaman Bahri, Jascha Sohl-Dickstein, Samuel S. Schoenholz, and Jeffrey Pennington. 2018. Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks. arXiv:1806.05393 [stat.ML]

[79] Yao-Yuan Yang, Cyrus Rashtchian, Hongyang Zhang, Ruslan Salakhutdinov, and Kamalika Chaudhuri. 2020. Adversarial Robustness Through Local Lipschitzness. *CoRR* abs/2003.02460 (2020). arXiv:2003.02460 https://arxiv.org/abs/2003.02460

[80] Yuichi Yoshida and Takeru Miyato. 2017. Spectral Norm Regularization for Improving the Generalizability of Deep Learning. arXiv:1705.10941 [stat.ML]

[81] Houssam Zenati, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. 2018. Efficient GAN-Based Anomaly Detection. *CoRR* abs/1802.06222 (2018). arXiv:1802.06222 http://arxiv.org/abs/1802.06222

Eidesstattliche Versicherung

(Affidavit)

Wickes, Justin
Name, Vorname
(surname, first name)

222987
Matrikelnummer
(student ID number)

☑ Bachelorarbeit
(Bachelor's thesis)

☐ Masterarbeit
(Master's thesis)

Titel
(Title)

Surveying the effects of Lipschitz continuity on the Robustness of Anomaly Detection Algorithms

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Pine Beach NJ, 12.06.24
Ort, Datum
(place, date)

Justin Wickes
Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Pine Beach NJ, 12.06.24
Ort, Datum
(place, date)

Justin Wickes
Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**