Master's Thesis

Deep Support Vector Data Description

Author
Lukas Ruff

Supervisors
Prof. Dr. Marius Kloft
Prof. Dr. Klaus-Robert Müller

A thesis presented in partial fulfillment of the requirements for the degree of Master of Science in Statistics

Joint Master's Program in Statistics in Berlin

Humboldt University of Berlin
Technical University of Berlin
Charité
Free University of Berlin

30th September, 2017

Declaration of Authorship

Selbständigkeitserklärung

Ich, Lukas Ruff, erkläre hiermit, dass ich die vorliegende Masterarbeit mit dem Titel "Deep Support Vector Data Description" selbständig verfasst und noch nicht für andere Prüfungen eingereicht habe. Sämtliche Quellen einschließlich Internetquellen, die unverändert oder abgewandelt wiedergegeben werden, insbesondere Quellen für Texte, Grafiken, Tabellen und Bilder, sind als solche kenntlich gemacht. Mir ist bekannt, dass bei Verstößen gegen diese Grundsätze ein Verfahren wegen Täuschungsversuchs bzw. Täuschung eingeleitet wird.

Berlin, den 30. September 2017 .....

Acknowledgments

I would like to thank Marius for his idea to investigate an approach to deep anomaly detection based on the one-class classification loss and moreover for his trust in supervising my thesis although being partly on parental leave. Thanks to Robert for all the helpful discussions we had and the suggestions you made for improving the approach. Also thanks to Nico for early discussions on kernel-based one-class classification and anomaly detection in general. And thanks to both, Marius and Prof. Dr. Klaus-Robert Müller for sparking the joy for machine learning research with their contagious enthusiasm shown for the subject in their lectures and seminars.

Finally, I would like to thank my parents for their unconditional support throughout my studies and Nicole who shared the path to graduation with me.

Contents

List of Figures

List of Tables

List of Abbreviations

AD	Anomaly Detection
AE	Autoencoder
AUC	Area Under the Receiver Operating Characteristic
BPTT	Backpropagation Through Time
CAE	Convolutional Autoencoder
CCCP	Concave-Convex Procedure
CDBN	Convolutional Deep Belief Net
CNN	Convolutional Neural Network
DAE	Denoising Autoencoder
DBM	Deep Boltzmann Machine
DBN	Deep Belief Net
DC	Difference-of-Convex
DCGAN	Deep Convolutional Generative Adversarial Network
DL	Deep Learning
ERM	Empirical Risk Minimization
FFN	Feed-forward Neural Network
GAN	Generative Adversarial Network
i.i.d.	Independent and identically distributed
IoT	Internet of Things
KKT	Karush Kuhn Tucker
LSTM	Long short-term memory
MLP	Multilayer Perceptron
OC-SVM	One-Class Support Vector Machine
PCA	Principal Component Analysis
PL-CNN	Piecewise-Linear Convolutional Neural Networks
QP	Quadratic Programming
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit

RKHS	Reproducing Kernel Hilbert Space
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
SGD	Stochastic Gradient Descent
SVDD	Support Vector Data Description
SVM	Support Vector Machine
VAE	Variational Autoencoder

Notation

$\boldsymbol{x} \geq 0$	Inequalities for vectors $\boldsymbol{x}$ are defined element-wise
$\boldsymbol{x}$	Vector notation in bold
$\boldsymbol{x}^T$	Transpose of vector $\boldsymbol{x}$
$\forall$	Quantifier "for all"
$\langle \boldsymbol{x}, \boldsymbol{y} \rangle$	Scalar product between vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
$\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\mathcal{F}}$	Scalar product between vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{F}$ in feature space $\mathcal{F}$
$\log$	The natural logarithm
$\mathbb{E}[X]$	Expected value of random variable X
$\mathbb{N}$	The natural numbers
$\mathbb{R}$	The real numbers
$\mathcal{D}_n$	A set of $n \in \mathbb{N}$ data points, that is $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$
$\mathcal{X}$	Data input space, that is $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ (e.g. $\mathcal{X} \subseteq \mathbb{R}^d$)
$\xrightarrow{P}$	Convergence in probability with probability measure P
$\phi(\cdot)$	Feature map implicitly defined by some kernel k, that is $k(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle_{\mathcal{F}}$ for vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ and feature space $\mathcal{F}$
$\mathrm{Var}(X)$	Variance of random variable X
$\|\boldsymbol{A}\|_F$	The Frobenius norm of a matrix $\boldsymbol{A}$
$\|\boldsymbol{x}\|$	Norm of vector $\boldsymbol{x}$
$\|\boldsymbol{x}\|_{\mathcal{F}}$	Norm of vector $\boldsymbol{x} \in \mathcal{F}$ in feature space induced by the scalar product $\langle \cdot\,, \cdot \rangle_{\mathcal{F}}$

$\|\cdot\|_p$	p-norm
$k(\boldsymbol{x}, \boldsymbol{y})$	Kernel applied to vectors $\boldsymbol{x}$ and $\boldsymbol{y}$
L^p	L^p space with corresponding p-norm $\|\cdot\|_p$
P	Probability measure
X	A random variable

1 Introduction

Anomaly detection is the task of uncovering unusual or atypical patterns in data. This task is posed in a great variety of application domains such as intrusion detection for cybersecurity, fraud detection in finance and insurance, fault and damage detection in industrial machines to enable predictive maintenance for example, or medical anomaly detection. In most of the domains, more and more data becomes available and the number of variables observed is increasing. Prices for sensors of different kinds are dropping and the availability of low-cost storage mediums as well as fast connectivity enable the accumulation of huge amounts of (usually unlabeled) data which often is of high-dimensionality. Event detection in an Internet of Things (IoT) with ubiquitous sensing capabilities (Gubbi et al., 2013), detecting novel situations autonomous vehicles with visual guiding systems face (Richter and Roy, 2017), or detecting diseases from medical imaging Schlegl et al. (2017) are all examples of the rising need for effective and efficient methods for the task of (unsupervised) high-dimensional anomaly detection for applications of large-scale.

The information contained in such large and complex data sets is of great potential, yet extracting that information can be very challenging. High-dimensionality causes data to be sparse due to the exponential increase in volume in the number of features which is known as the *curse of dimensionality*. The key problem for anomaly detection caused by high dimensionality of the data is given by the presence of irrelevant features which add noise that can mask the relevant factors of variation for detecting anomalies (Zimek et al., 2012). Furthermore, the unsupervised anomaly detection task is challenging in itself as there is no discriminative information given on the anomalies. Thus, only information from typical examples can be used for a description of a "normal" class. This problem of *one-class classification* or *data description* therefore has the aim to give a most accurate characterization of the normal class in order to detect anomalies.

The popular kernel methods of Support Vector Data Description (SVDD) (Tax and Duin, 2004) and the One-Class Support Vector Machine (OC-SVM) (Schölkopf et al., 2001) were introduced for the one-class classification task which allow to incorporate non-linear structures into the description of the data by the use of kernels. Those methods have nice practical properties, such as allowing to incorporate a prior assumption on the number of anomalies being present in the training data, and theoretical ones as

they are asymptotically consistent density level set estimators (Vert and Vert, 2006). However, they do not scale well as their complexity grows quadratically in the training set size (Vempati et al., 2010) which makes them impractical for large-scale applications. Moreover, the kernel-induced feature transformations are fixed and the generalization error of kernel methods increases in the number of irrelevant features (Huang and LeCun, 2006, Liu et al., 2013).

Deep learning (LeCun et al., 2015) takes the step from manual or fixed feature transformations that are applied subsequently to a learning objective to *learning* complex data representations jointly with the objective. By using hierarchical model architectures with multiple processing layers, deep models learn data representations with multiple levels of abstraction. This allows to represent a rich space of feature representations in a compact form which helps alleviating the curse of dimensionality (Bengio and Delalleau, 2011). Moreover, deep methods scale well with large data set sizes as they are optimized via stochastic gradient descent algorithms which scale linearly and allow distributed parallel processing (Bengio, LeCun et al., 2007).

Objective and contributions of this thesis

The objective of this thesis is to find a new approach utilizing the potential of deep representations for the task of *high-dimensional and large-scale anomaly detection*. For this, we introduce the novel method of *Deep Support Vector Data Description*. The idea of the approach is to use the classification-based objectives introduced by kernel one-class classification methods, but to transfer from fixed kernel feature representations to the learning of representations. The deep SVDD method is based on the primal objective of SVDD which is to find a data-enclosing hypersphere of minimum volume. However, we extend the objective to jointly train a deep network for learning representations that capture the factors of variation underlying the data. We derive theoretical propositions of the method such as that the so-called ν-property of the respective kernel variants is preserved. To put the method to the test, experiments where we compare deep SVDD to other (shallow and deep) anomaly detection methods are performed in which we find deep SVDD to give competitive results and moreover seems to generalize well even without additional regularization. Thus, the thesis contributes a novel deep anomaly detection method for the task of high-dimensional and large-scale anomaly detection.

Related Work

Using hierarchical representations learned via deep networks has not yet been explored much for the task of anomaly detection, at least compared to the general attraction the field of deep learning has generated in recent years. This is somewhat surprising, though the focus in deep learning has been mainly on supervised objectives (LeCun et al., 2015, Schmidhuber, 2015) and defining unsupervised deep learning objectives is less straightforward (Bengio et al., 2013). Approaches to anomaly detection using deep

representations can be separated into *hybrid models*, which use unsupervised deep learning methods only for extracting features on which classical anomaly detection techniques are then applied on top, and *fully deep* approaches that learn the representations directly on the objective defined for detecting anomalies. The almost exclusively employed objective for deep anomaly detection is the reconstruction error, that is deep autoencoders (Hinton and Salakhutdinov, 2006) (of various kinds) are the dominantly used approach. We will give the detailed review on deep anomaly detection approaches in section 4.3, where we will see that there essentially is only one other approach deviating from the reconstruction objective.

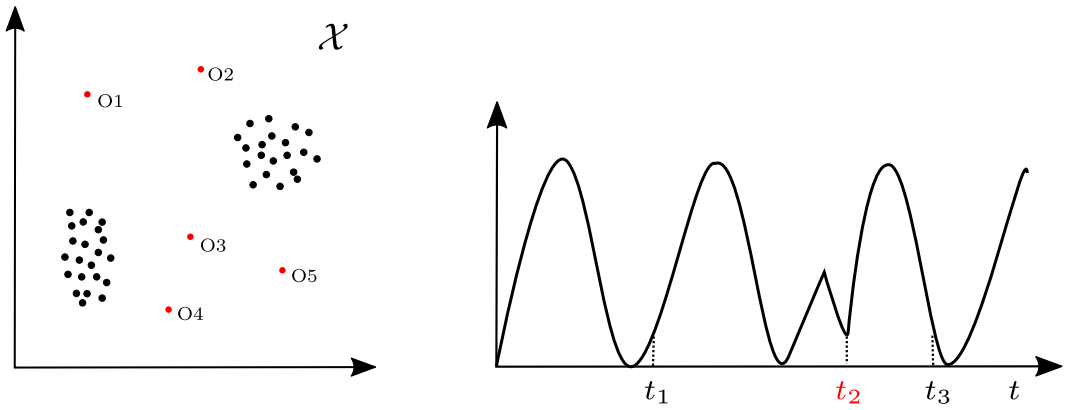
Organization of this Thesis

This thesis is structured as follows: section 2 introduces the task of anomaly detection and the one-class classification setting, where we explain the challenges arising especially for high-dimensional data due to the curse of dimensionality. The section closes with the concept of density level set estimation which is the theory underlying the kernel-based one-class classification methods. In section 3 the two kernel-based one-class classification methods of Support Vector Data Description and One-Class Support Vector Machine are presented. For both methods we derive the kernelized dual problems starting from their primal problem formulations and show that for a certain class of kernels the two approaches in fact solve the equivalent problem. We finish the section with theoretical results. Section 4 gives an overview on deep learning where we explain the fundamentals of hierarchical representation learning and show open research challenges in the field. Moreover, other approaches to deep anomaly detection are thoroughly reviewed in this section. After explaining and reviewing the ingredients, section 5 presents the novel method of deep SVDD for which we give a soft- and a hard-boundary objective. Theoretical properties of deep SVDD will be given and discussed. Experiments on the MNIST, CIFAR-10 and the Bedroom data sets are presented in section 6. Finally, section 7 reviews, concludes and gives direction for further research.

2 Anomaly Detection

What is anomaly detection? *Anomaly detection (AD)* defines the task of identifying patterns in data which qualify as *unusual*, *unexpected*, or *atypical* in relation to the overall data set or in context of a domain-specific application. Such instances or subsets of the data are called *anomalous*. Another term for anomaly detection, often used interchangeably, is *outlier detection*. Hawkins (1980) gives the classic definition of an *outlier* as an observation that deviates so much from other observations as to arouse suspicion that it was generated by a different mechanism. Related fields of study to anomaly detection are *novelty detection* (also often interchangeably in meaning), *change detection*, *noise removal*, *extreme value theory*, *density (level set) estimation*, *one-class classification*, and *minimum volume set estimation*. Applications of anomaly detection are widely spread across domains. Examples include fraud detection (e.g. in finance and insurance), intrusion detection for cybersecurity, industrial fault and damage detection, system-health monitoring, event detection, military surveillance, image processing, textual anomaly detection (e.g. novel topics) and medical anomaly detection (e.g. disease detection). Two extensive reviews on anomaly detection in recent years are given by Chandola et al. (2009) and Pimentel et al. (2014). Another resource to explore the width of anomaly detection techniques is the monograph on outlier analysis by Aggarwal (2013) which provides a rich collection.

Aspects to consider Different anomaly detection techniques are differently well-suited given the task at hand. Aspects of the problem that play an essential role in determining suitable methods are the *nature of the data* (categorical and continuous; univariate and multivariate; temporal data; spatial data), the *availability of labels* (i.e. known anomalous examples), and the possible *types of anomalies* suspected. If examples of anomalies are known, supervised or semi-supervised methods can be considered, depending on whether labels are given in full or are only partially available. Anomalies themselves can be simple point anomalies or contextual-based. The latter case is given if the data records have contextual attributes (e.g. time or space), which must be accounted for to characterize anomalies.



(a) Point anomalies.

(b) Contextual anomalies: the value of the time series at t_2 is anomalous based on the temporal context whereas the same value at t_1 and t_3 are considered normal.

Figure 2.1: Types of anomalies illustrated (Figures adapted from Chandola et al. (2009)).

Anomaly detection techniques Methods of anomaly detection can be domain-specific or more generic. Table 2.1 shows how anomaly detection techniques can be grouped based on their underlying approaches. To measure the degree of "anomalousness" of

Classification based	e.g. One-Class SVM, Support Vector Data Description
Nearest neighbor based	e.g. k-Nearest Neighbor, Local Outlier Factor (LOF)
Clustering based	e.g. k-Means
Statistical	parametric (Maximum Likelihood Estimation) and non-parametric (e.g. Kernel Density Estimation)
Information theoretic	e.g. Local Search Algorithm (LSA)
Spectral	PCA-based

Table 2.1: Groups of anomaly detection techniques (Chandola et al., 2009).

some input $\boldsymbol{x} \in \mathcal{X}$, anomaly detection techniques usually define an *anomaly score* $s(\boldsymbol{x})$ which implies a ranking of examples. From such a score, by defining an appropriate threshold, samples can be flagged to be anomalous or normal and from the order a prioritization for possible expert analysis is provided.

Below, the setting of one-class classification will be introduced, which is the framework this work focuses on. Afterwards, the special challenges in detecting anomalies in high-

dimensional spaces are investigated. Finally, the general concept of density level set estimation will be established before the next section examines Support Vector Data Description and One-Class SVM, which are examples of density level set estimators, in great detail.

2.1 One-Class Classification

The one-class setting Most commonly, data sets in anomaly detection are strongly imbalanced including only a few (known) anomalies or none (known) anomalies at all. The task tackling the latter setting, where only records of the "normal" or "typical" class (and potentially unknown anomalies) are given, is called *one-class classification*. The term originates from the work of Moya et al. (1993) and is an unsupervised learning task, that is no labels are accessible. The assumption is that the data set predominantly includes samples from the normal class, which we also call the *target class*, but contamination with outliers often is suspected. Such data sets could for example be generated from monitoring a system in its normal state such as sensor data from an ordinarily running industrial machine or network activity logs from common users. Generating examples from the target class often is cheap whereas collecting or simulating anomalous data records on the other hand can often be costly and moreover characteristics of anomalies are usually unknown. This motivates the need for one-class classification techniques.

Data desciption As methods in the one-class classification setting only can utilize information from one unlabeled data set in trying to discriminate against *all* (potentially infinite) classes of unseen anomalies, the objective is to give a most accurate description of the target class. This is why the task of one-class classification also is called *data description*. In comparison, regular supervised binary or multi-class classification problems, in which ground truth labels are given for all the classes involved, can utilize information from all of the classes to distinguish between them and the number of classes is limited. In one-class classification, in order to detect anomalies, common patterns and properties amongst the target class thus must be characterized — a "least common denominator" of the target examples has to be specified. Sufficiently large deviations from the characterization of the "normal" are consequently potential outliers. Such a characterization or description of the target class usually translates back into estimating a region in input space representing the target class as we will see below in the approach of density level set estimation.

Challenges The challenges of this simple idea to determine a bounded region where target class lives are manifold:

- The distribution of the target class may overlap with different anomalous distributions (i.e. there is no precise boundary between the target class and anomalies).

- The distribution of the target class in some applications may not be stationary (i.e. the description of the target class has to be adapted over time).
- There could be undetected anomalies in the training set that distort the estimation of a boundary.
- The presence of noisy samples or features make it harder to deduce a tight description to distinguish from anomalies as relevant variations are masked by noise.
- Adversarial anomalies (e.g. in network intrusion detection) are especially designed to pretend to be of the target class.

Besides those challenges, high-dimensional data itself, that is data having many (hundreds or thousands) of features, poses additional challenges to the task of unsupervised anomaly detection.

2.2 The Curse of Dimensionality

The *curse of dimensionality* is a general term describing various phenomena caused from the volume of space increasing exponentially in its dimensionality. Bellman (1961) originally introduced the term studying problems in dynamic optimization. To get an intuition on how dimensionality affects measuring distances in space, consider the volume of a hypersphere centered at some point in a d-dimensional space with radius r:

$$V_d(r) = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)}\, r^d, \tag{2.1}$$

where $\Gamma(\cdot)$ is the Gamma function. Let's now compare the fraction of a small shell to the volume of the entire ball, that is for any $\varepsilon > 0$ consider

$$\begin{aligned} \frac{V_d(r) - V_d(r-\varepsilon)}{V_d(r)} &= \frac{r^d - (r-\varepsilon)^d}{r^d} \\ &= 1 - \left(\frac{r-\varepsilon}{r}\right)^d \longrightarrow 1 \quad \text{for} \quad d \longrightarrow \infty. \end{aligned} \tag{2.2}$$

This means that as the dimensionality increases, volume is pushed further away from any fixed point in space given a notion for the "vastness" of high-dimensional spaces. Put differently, when exploring the neighborhood of some point, the volume of space covered increases to the power of dimensionality d when increasing the search radius r of the neighborhood. Informally, almost all volume is "far away". Given some n data samples, the implication is that the higher the dimensionality of the data is, that is the more features are given, the more sparse the data becomes in input space. This poses several challenges not just to the task of anomaly detection, but to machine learning

and statistical estimation in general as the required number of samples (depending on the correlation between the features) often also has to increase exponentially in order to obtain statistically reliable results. This also means that methods that are based on the assumption of independent features are not applicable in high-dimensional settings as the sample size required explodes. Zimek et al. (2012) analyze the different effects arising from the curse of dimensionality thoroughly and moreover state important consequences for anomaly detection.

Concentration of distances One effect is that distances between data points become more and more similar with increasing dimensionality and therefore carry less meaning. This *distance concentration effect* also was investigated by Beyer et al. (1999) which state the effect as follows: Given an i.i.d. sample of random variables $X_{1,d}, \ldots, X_{n,d}$ of dimensionality d, then if

$$\lim_{d\to\infty} \operatorname{Var}\left(\frac{\|X_{1,d}\|}{\mathbb{E}[\|X_{1,d}\|]}\right) = 0 \tag{2.3}$$

for some norm $\|\cdot\|$, that is the variance of the ratio of the length of any random vector to the mean vector length vanishes in dimensionality, we have that

$$\frac{\mathrm{dist}_{\max} - \mathrm{dist}_{\min}}{\mathrm{dist}_{\min}} \xrightarrow{P} 0, \tag{2.4}$$

where $\mathrm{dist}_{\max}$ is the maximum and $\mathrm{dist}_{\min}$ the minimum distance to some other (independently) drawn vector $\tilde{X}_d$, that is

$$\mathrm{dist}_{\max} = \max_i \|X_{i,d} - \tilde{X}_d\| \quad \text{and} \quad \mathrm{dist}_{\min} = \min_i \|X_{i,d} - \tilde{X}_d\|. \tag{2.5}$$

Hence, if assumption (2.3) is satisfied, the proportional difference between the greatest-point distance and the closest-point distance vanishes as well and therefore distances get concentrated with dimensionality. Condition (2.3) is fulfilled by a broad range of norms (most important, all L^p-norms with $p \geq 1$) and distributions, the simplest setting being data that is i.i.d. in all feature dimensions $X_{1,d}^{(1)}, \ldots, X_{1,d}^{(d)}$ and distances measured in the L^2-norm since

$$\lim_{d\to\infty} \operatorname{Var}\left(\frac{\|X_{1,d}\|_2^2}{\mathbb{E}[\|X_{1,d}\|_2^2]}\right) = \frac{d \operatorname{Var}((X_{1,d}^{(1)})^2)}{(d\,\mathbb{E}[(X_{1,d}^{(1)})^2])^2} = \frac{\operatorname{Var}((X_{1,d}^{(1)})^2)}{d\,\mathbb{E}[(X_{1,d}^{(1)})^2]^2} = 0, \tag{2.6}$$

assuming the respective moments are finite. The consequence from concentrating distances is that distance-based anomaly scores get numerically close making it hard to differentiate true outliers.

Higher dimensionality can simplify AD The distance concentration effect often is given as the explanation for worse performances of anomaly detection techniques in spaces of higher dimensionality as "all data points become indistinguishable". But this is only a simplified explanation. In fact, the task of detecting an outlier becomes easier with greater dimensionality if the outlier follows a different distribution than the target class in each of the features added. In this case, the mean pair-wise distance of the outlier to all other data points will be significantly different from the mean pair-wise distance of all the data points within the target class as each additional feature will add to the discriminative power (cf. section 2.1 in Zimek et al., 2012). As an illustrative example, imagine the target class being generated from identical normal distributions in every dimension and an outlier following a normal distribution in each dimension as well, but with different mean. With overlapping tails, it could be hard to tell from observing only one dimension if a point in question is indeed an anomaly or rather an atypical point from the target class. But as the dimensionality increases, the overlapping region becomes smaller and smaller in relation to the supports of the target and the anomaly distribution. Thus, as more features are added, anomalous examples become extremely unlikely to have been generated from the target class and therefore separability improves.

The key problem: irrelevant features The actual problem for the task of anomaly detection caused by the curse of dimensionality is given, if only a few of the many features are indicative for detecting anomalies. In this case, the presence of *irrelevant features* (or *noise attributes*) starts to cloak the differences in variation of those features that are relevant. Therefore, the crucial factor for success in high-dimensional anomaly detection is the proportion of outlier-relevant to outlier-irrelevant features. The challenge thus lies in finding appropriate subspaces or feature transformations that have reduced noise but exhibit information from the features relevant to uncover anomalies, that is to find subspaces or transformations that maximize the signal-to-noise ratio.

The data-snooping bias Another effect that must be considered for defining outliers is the *data-snooping bias*. Due to the combinatorial explosion of feature subspaces with increasing dimensionality, the chance of finding a subspace for a given point to appear anomalous increases. To illustrate the effect, consider again i.i.d. data with independent normal features. If we define outliers simply by having at least one feature value outside the 3σ-range, the probability for a point from the target class to be declared anomalous is given by $1 - 0.9973^d$ and converges to 1 in dimensionality. Thus, anomaly scores and thresholds must be chosen carefully.

The effects in summary To summarize, the challenges posed by the curse of dimensionality on anomaly detection are (cf. section 2.6 in Zimek et al., 2012):

- *Exponential search space* causing the data to be sparse.

- *Concentration of scores* due to the concentration of distances potentially causing numerical issues.
- *Irrelevant features* cloaking anomalies.
- *Data-snooping bias* enabling every data point to appear anomalous given enough feature subspaces and uncareful anomaly interpretation.

The effects from the curse of dimensionality given above might not be as severe, however, if the data has strong correlations amongst the features. In that case, the data lies close to a manifold of lower dimensionality embedded into the high-dimensional space. Therefore the effective dimensionality of a problem could be much lower compared to its representational dimensionality. The extreme exemplification of this would be perfectly correlated features, i.e. duplicate features. In this scenario, the data would be on a one-dimensional line and the effective dimensionality one.

Such correlations amongst the features could be of non-linear nature. One approach to detect non-linear patterns in the data are kernel methods, which measure similarities between data instances by means of a kernel function. In order to analyze kernel-based approaches to one-class classification, we will first introduce the concept of density level set estimation in general.

2.3 Density Level Set Estimation

Assume some sample of the target class $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ living in some (potentially high-dimensional) input space $\mathcal{X}$ was generated from i.i.d. random variables $X_1, \ldots, X_n \sim P$ with probability measure P potentially unknown. Records sampled from random variables following probability distributions different from P are considered to be anomalous as they do not follow the same data-generating process. As mentioned above, one of the main challenges in unsupervised anomaly detection is that also the anomalous distributions are unknown and that they potentially have overlaps with the target distribution. This motivates to introduce a concept for estimating subsets of the target distribution support in which samples under the measure P most likely occur. That is, the goal is to find a set containing the "most typical" samples from the target class. Such a concept can be expressed within the framework of *density level set estimation* (Tsybakov, 1997).

Definition 2.1 (Density level set)**.** Assume probability measure P has density p (e.g. P absolutely continuous w.r.t. the Lebesgue-measure) and let $X \sim P$ be another independent copy from the random variables $X_1, \ldots, X_n$. Then, the set defined by

$$C_\nu = \{\boldsymbol{x} \in \mathcal{X} \,|\, p(\boldsymbol{x}) \geq b_\nu\} \tag{2.7}$$

with $\nu \in (0, 1]$ and $b_\nu > 0$ such that

$$P(X \in C_\nu) = 1 - \nu \tag{2.8}$$

is called the *ν-density level set*.

That is, the ν-density level set is the region in input space $\mathcal{X}$ which contains the most likely inputs under p with overall probability mass of $1-\nu$. Conversely, inputs under p fall outside of C_ν with probability ν.

The role of the ν-parameter The parameter ν is user-specified in advance and the aim in context of anomaly detection is to select ν such that C_ν contains the typical samples from the target class while rare samples or anomalies should fall outside of it. Hence, ν controls the boundary between what is considered to be normal and what is considered to be anomalous under P. We will discuss the selection of ν in section 3.4 in more detail.

Density level set estimation The aim now is to estimate the true density level set C_ν from the data $\mathcal{D}_n$, that is to find a function $f : \mathcal{X} \to \mathbb{R}$ such that

$$\hat{C}_\nu = \{\boldsymbol{x} \in \mathcal{X} \mid f(\boldsymbol{x}) \geq 0\} \tag{2.9}$$

is a close estimation of C_ν. The threshold of 0 is chosen arbitrary as f could always be shifted. The desired property of a density level set estimator $\hat{C}_\nu$ would be to asymptotically have probability mass $1-\nu$ under P, i.e.

$$P(X \in \hat{C}_\nu) \to 1-\nu \quad \text{for} \quad n \to \infty, \tag{2.10}$$

that is

$$P(X \in (\hat{C}_\nu \setminus C_\nu) \cup (C_\nu \setminus \hat{C}_\nu)) \to 0 \quad \text{for} \quad n \to \infty. \tag{2.11}$$

As we will see, Support Vector Data Description and the One-Class SVM are density level set estimators which indeed asymptotically converge to the true density level set in probability.

Density level set and density estimation At this point, it is worth pointing out the differences between *density level set estimation* and the task of *density estimation*. In classic density estimation the goal is to estimate a function $\hat{p} : \mathcal{X} \to \mathbb{R}$ such that $\hat{p}$ approximates the true density, i.e. $\hat{p}(\boldsymbol{x}) \approx p(\boldsymbol{x})$ should hold on the overall support of P and thus the aim is to estimate the specific distribution. In comparison, the function f estimated to get a density level set estimation $\hat{C}_\nu$ could for example only be a binary function indicating whether some point lies inside or outside the specified level set. The problem in this case is reduced to finding a boundary separating the most likely regions that have some pre-specified probability mass.

In the next section, we will introduce the two most prominent kernel-based one-class classification methods for density level set estimation.

3 Kernel-Based One-Class Classification

One classic approach to incorporate non-linear structures of the data into learning machines are kernel-based feature representations (Schölkopf et al., 1999, Müller et al., 2001, Schölkopf and Smola, 2002). Given an input space $\mathcal{X} \subset \mathbb{R}^d$, for every Mercer kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, according to the *representer theorem*, there exists a (possibly infinite-dimensional) Hilbert space $\mathcal{F}$, called *Reproducing Kernel Hilbert Space (RKHS)* or simply *feature space*, such that the dot product in Hilbert space $\mathcal{F}$ is given by kernel k (see Definition B.1 and Theorem B.2 in the appendix). That is, kernel k implicitly defines a mapping $\phi : \mathcal{X} \to \mathcal{F}$ such that

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \langle \phi(\boldsymbol{x}), \phi(\tilde{\boldsymbol{x}}) \rangle_{\mathcal{F}}, \quad \text{for all } \boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X}. \tag{3.1}$$

The mapping ϕ is the so-called *feature map*. This property often is referred to as the "kernel trick". The kernel trick allows for non-linear transformations of the data in algorithms that only depend on pairwise dot products between samples and do not require the specific representation of each point. Thus, kernel methods enable to incorporate non-linear dependency structures into a model by implicitly mapping the data into a high-dimensional feature space, in which in turn a linear separation of the data often is much easier. The kernel trick gives the most prominent class of kernel methods, the support vector machine, its power by enabling to fit the maximum-margin hyperplane in feature space $\mathcal{F}$. Besides, many other classical linear methods allow to be "kernelized" resulting in models such as kernel PCA (Schölkopf et al., 1997), kernel canonical correlation analysis (Akaho, 2001) or kernel ridge regression (e.g. in Murphy, 2012).

The two most prominent kernel-based approaches to one-class classification are the *One-Class Support Vector Machine (OC-SVM)* (Schölkopf et al., 2001) and *Support Vector Data Description (SVDD)* (Tax and Duin, 2004). Both methods are approaches to density level set estimation as outlined in the previous section, the difference in the methods being, that OC-SVM separates the data via *hyperplane* whereas SVDD separates the data by an data-enclosing *hypersphere*. If the kernel trick is applied, the separation of the data and estimation of the decision boundary is again carried out implicitly in feature space $\mathcal{F}$. As we will see, the two approaches are equivalent if kernel k fulfills a certain property. This property is satisfied by the most frequently used Gaussian kernel.

The decision boundary approach of kernel-based one-class classification to anomaly detection is in line with Vapnik's argument, that in order to solve a problem, one should

not try to solve a more general problem as an intermediate step (Vapnik, 1998). In the context of one-class classification this means, that the estimation of a level set, which is a simpler problem compared to general density estimation, is the sufficient problem to solve.

In this section, we will first investigate the hypersphere-based approach of SVDD. After that, we will analyze the hyperplane-based OC-SVM and eventually show in which case they solve the equivalent problem. Finally, we will finish this section by giving some theoretical results.

3.1 Support Vector Data Description

The method of Support Vector Data Description (SVDD) was introduced by Tax and Duin (2004). The aim of data domain description in general is to characterize a data set. As mentioned before, data domain description is an equivalently used term to one-class classification. A perfect description of a data class would tell for every (seen and unseen) data point whether this data point belongs to the target class or not. Naturally, data description and one-class classification methods can be applied to anomaly detection by identifying data points which are uncharacteristic from the target class. A thorough revisit of SVDD, correcting some minor theoretical inaccuracies of the original paper, was carried out by Chang et al. (2013).

Applications of SVDD are given in a variety of domains. Examples of successful applications include face recognition (Lee et al., 2006), pattern denoising (Park et al., 2007) and of course anomaly detection (Banerjee et al., 2007).

The idea The basic idea of SVDD to determine a description of the data set is to find a hypersphere with minimum volume which covers all data from the target class. The method is an example of the more general class of *minimum volume estimators* (Schölkopf et al., 2001). The approach to minimize a data-enclosing hypersphere also has roots in statistical learning theory: Schölkopf et al. (1995) show that the VC-dimension of a classifier, which in part controls an upper bound on the structural risk for a class of estimators (the other part being the empirical risk), is bounded by the squared radius of the smallest sphere enclosing all training data. By drawing a spherically shaped boundary around the target data and minimizing the volume of the sphere, SVDD reduces the chance of falsely identifying an outlier as part of the target class. As we will see, the description does not have to be made in the original input space. From the dual of its objective, SVDD can be kernelized and thereby allows to deduce a description of the data mapped to feature space enabling to account for non-linear structures of the data in the description. Let's first have a look at the primal objective of SVDD.

Primal problem

Let $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a set of training data that we would like to characterize living in some (potentially high-dimensional) input space $\mathcal{X} \subset \mathbb{R}^d$. We assume that the data has variance in every feature dimension (otherwise the feature without any variation between the data points could be utilized to derive a trivial perfect description of the whole data set solely by one point). Next, a hypersphere in $\mathcal{X}$ is uniquely characterized by its center $\boldsymbol{c} \in \mathcal{X}$ and radius $R > 0$ and therefore can be identified by the pair $(R, \boldsymbol{c})$. The objective of SVDD now is to minimize the volume of the sphere including all data points from the normal class, that is

$$\begin{aligned} \min_{R,\boldsymbol{c}} \quad & R^2 \\ \text{s.t.} \quad & \|\boldsymbol{x}_i - \boldsymbol{c}\|^2 \leq R^2, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{3.2}$$

In this hard-boundary setting a testing point $\boldsymbol{x} \in \mathcal{X}$ is detected as an outlier if $\|\boldsymbol{x} - \boldsymbol{c}\|^2 > R^2$. To account for the possibility of outliers in the training data, the hard constraints can be softened by introducing slack variables $\xi_i \geq 0$. This is analogous to the transition from hard-margin to soft-margin support vector machines, where errors in classification are allowed but get penalized. The penalties are given exactly by the slack variables ξ_i. Therefore, the soft-boundary objective, constituting the SVDD primal, is formulated as

Problem 3.1 (SVDD primal)**.**

$$\begin{aligned} \min_{R,\boldsymbol{c},\boldsymbol{\xi}} \quad & R^2 + \frac{1}{\nu n} \sum_i \xi_i \\ \text{s.t.} \quad & \|\boldsymbol{x}_i - \boldsymbol{c}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, n, \end{aligned} \tag{3.3}$$

where hyperparameter $\nu \in (0, 1]$ controls the trade-off between volume and penalty. We will later see, that ν in fact controls the fraction of outliers. Note that the penalty on the slack variables is a L^1-penalty favoring sparse penalization.

Dual problem

In their revisit of SVDD, Chang et al. (2013) correctly indicate that the primal Problem 3.1 in fact is not a convex optimization problem since the inequality constraints $(R, \boldsymbol{c}, \boldsymbol{\xi}) \mapsto \|\boldsymbol{x}_i - \boldsymbol{c}\|^2 - R^2 - \xi_i$ are concave in R (see Definition A.1 in the appendix for the characterization of a convex optimization problem). This is not pointed out or discussed in the derivation of the dual objective in the original introduction of SVDD by Tax and Duin (2004). Moreover, Tax and Duin (2004) did not verify the constraint qualifications that must be satisfied in order for strong duality to hold.

Reparameterized SVDD problem The correct formulation of the primal such that it is convex requires a reparameterization of the problem. By defining $\tilde{R} := R^2$, we get the following reparameterized primal:

$$\begin{aligned} \min_{\tilde{R},\boldsymbol{c},\boldsymbol{\xi}} \quad & \tilde{R} + \frac{1}{\nu n}\sum_i \xi_i \\ \text{s.t.} \quad & \|\boldsymbol{x}_i - \boldsymbol{c}\|^2 \leq \tilde{R} + \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1,\ldots,n, \\ & \tilde{R} \geq 0 \end{aligned} \tag{3.4}$$

Now, the resulting inequality constraints $(\tilde{R}, \boldsymbol{c}, \boldsymbol{\xi}) \mapsto \|\boldsymbol{x}_i - \boldsymbol{c}\|^2 - \tilde{R} - \xi_i$ are convex as the sum of convex and linear functions and the objective function is linear, thus making the problem convex. The reparameterized primal (3.4) also satisfies Slater's condition (see Theorem A.4 in the appendix) as for example for $\boldsymbol{c} = \boldsymbol{0}$ one can simply find $\tilde{R} > 0$ sufficiently large and also for every $\boldsymbol{x}_i \in \mathcal{D}_n$ sufficiently large $\xi_i > 0$ such that

$$\|\boldsymbol{x}_i\|^2 - \tilde{R} - \xi_i < 0, \tag{3.5}$$

for every $i = 1,\ldots,n$, that is there always can be found an interior point in the parameter space such that strict inequalities hold. Thus, strong duality holds and the optimum of the dual coincides with optimum of the primal. Since the formally correct derivation of the dual has identical implications on the resulting dual problem and its optimality conditions, we will stick to the original parameterization of Tax and Duin (2004) in the derivation of the dual below because it is the predominant formulation found in the literature on SVDD.

Derivation of the SVDD dual Applying the Lagrange duality principle (see also Theorem A.2 in the appendix), we can incorporate the inequality constraints in (3.3) into the Lagrangian $\mathcal{L}$:

$$\begin{aligned} \mathcal{L}(R, \boldsymbol{c}, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = R^2 + \frac{1}{\nu n}\sum_i \xi_i \\ - \sum_i \alpha_i \left(R^2 + \xi_i - \|\boldsymbol{x}_i - \boldsymbol{c}\|^2\right) - \sum_i \beta_i \xi_i, \end{aligned} \tag{3.6}$$

with Lagrange multipliers $\alpha_i \geq 0$ and $\beta_i \geq 0$. Setting the partial derivatives of the Lagrangian with respect to the primal variables to zero, gives the following optimality conditions:

$$\frac{\partial \mathcal{L}}{\partial R} = 2R - 2R\sum_i \alpha_i = 0 \qquad \Longrightarrow \qquad \sum_i \alpha_i = 1 \tag{3.7}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{c}} = 2\sum_i \alpha_i(\boldsymbol{x}_i - c) = 0 \qquad \overset{(3.7)}{\Longrightarrow} \qquad \boldsymbol{c} = \sum_i \alpha_i \boldsymbol{x}_i \tag{3.8}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = \frac{1}{\nu n} - \alpha_i - \beta_i = 0 \qquad \Longrightarrow \qquad \alpha_i = \frac{1}{\nu n} - \beta_i \tag{3.9}$$

Since the Lagrangian multipliers must be non-negative, the last equations implies that $0 \leq \alpha_i \leq \frac{1}{\nu n}$ must hold. From those boundaries on the elements of $\boldsymbol{\alpha}$, we can also see why we have the restriction $\nu \leq 1$ since otherwise (3.7) cannot be fulfilled and then the dual problem would not be feasible. Substituting optimality conditions (3.7)–(3.9) back into the Lagrangian (3.6) gives

$$\begin{aligned}
\mathcal{L} = \quad & R^2 + \frac{1}{\nu n}\sum_i \xi_i - R^2 \underbrace{\sum_i \alpha_i}_{=1} - \sum_i \underbrace{(\alpha_i + \beta_i)}_{=\frac{1}{\nu n}} \xi_i \\
& + \sum_i \alpha_i \|\boldsymbol{x}_i - \sum_j \alpha_j \boldsymbol{x}_j\|^2 \\
= \quad & \sum_i \alpha_i \|\boldsymbol{x}_i\|^2 - 2\sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \underbrace{\left(\sum_i \alpha_i\right)}_{=1} \\
= \quad & \sum_i \alpha_i \|\boldsymbol{x}_i\|^2 - \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle
\end{aligned} \tag{3.10}$$

In total, the dual objective therefore is given by

Problem 3.2 (SVDD dual)**.**

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & \sum_i \alpha_i \|\boldsymbol{x}_i\|^2 - \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\
\text{s.t.} \quad & \sum_i \alpha_i = 1, \\
& 0 \leq \alpha_i \leq \frac{1}{\nu n}, \quad \forall i = 1, \ldots, n.
\end{aligned} \tag{3.11}$$

Support vectors in SVDD In the optimum, the necessary Karush-Kuhn-Tucker (KKT) conditions (see Theorem A.3 in the appendix) for optimality must hold. From the KKT conditions, we can deduce the optimal Lagrange multipliers for different cases of inputs $\boldsymbol{x}_i \in \mathcal{X}$. The KKT slackness condition (A.8) implies that for all $i = 1, \ldots, n$ the following must hold for the Lagrange multipliers in the optimum $(R^*, \boldsymbol{c}^*, \boldsymbol{\xi}^*)$:

$$\alpha_i \left(\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 - R^{*2} - \xi_i^*\right) = 0 \tag{3.12}$$

$$\beta_i \, \xi_i^* = 0 \tag{3.13}$$

With those additional conditions we can infer the range of the Lagrange multipliers for the three different cases a data point $\boldsymbol{x}_i \in \mathcal{X}$ can be in:

(1) Inside of the sphere: $\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 < R^{*2}$

(2) On the boundary of the sphere: $\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 = R^{*2}$

(3) Outside of the sphere: $\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 > R^{*2}$

If $\boldsymbol{x}_i$ lies within the sphere, this implies that $(\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 - R^{*2} - \xi_i^*) < 0$ because ξ_i^* is non-negative. Thus, for (3.12) to be valid, $\alpha_i = 0$ must hold. By (3.9) this means that $\beta_i = \frac{1}{\nu n} > 0$ and therefore $\xi_i^* = 0$ in order for (3.13) to be satisfied.

In the case that $\boldsymbol{x}_i$ is on the boundary of the sphere, we get that $\xi_i^* = 0$ must hold as well because otherwise for (3.12) and (3.13) to be satisfied, both, α_i and β_i would have to be 0. But then $\alpha_i + \beta_i \neq \frac{1}{\nu n}$ and thus (3.9) would be violated. Therefore, if $\boldsymbol{x}_i$ lies on the boundary, we only know the Lagrange multipliers to be within the general bounds, i.e. $0 \leq \alpha_i \leq \frac{1}{\nu n}$ and $\beta_i = \frac{1}{\nu n} - \alpha_i$.

Finally, if $\boldsymbol{x}_i$ lies outside the sphere, for the primal inequality constraint to hold there must be a positive slack penalty, i.e. $\xi_i^* > 0$. This means that $\beta_i = 0$ by slackness condition (3.13) and hence $\alpha_i = \frac{1}{\nu n}$ in order for (3.9) to hold.

Conversely, if we look at the implications on the location of some data point $\boldsymbol{x}_i$ making assumptions on the Lagrange multipliers, we get that if $0 < \alpha_i < \frac{1}{\nu n}$ then $\beta_i = \frac{1}{\nu n} - \alpha_i > 0$ and therefore $\xi_i^* = 0$ by slackness condition (3.13) which in turn implies that $\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 = R^{*2}$ in order for (3.12) to hold, that is, $\boldsymbol{x}_i$ must lie on the boundary of the sphere. This allows to recover R^* from any $\boldsymbol{x}_i$ with $0 < \alpha_i < \frac{1}{\nu n}$ by

$$R^{*2} = \|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 = \|\boldsymbol{x}_i\|^2 - 2\sum_j \alpha_j \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle + \sum_{j,k} \alpha_j \alpha_k \langle \boldsymbol{x}_j, \boldsymbol{x}_k \rangle, \tag{3.14}$$

Furthermore, having $\alpha_i = \frac{1}{\nu n}$ then $\beta_i = 0$ by (3.9) and thus, because by (3.12) $\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 = R^{*2} + \xi_i^*$ must hold, $\boldsymbol{x}_i$ either is on the boundary of the sphere or outside of it (depending on whether ξ^* is zero or positive).

The implications above are important for understanding the characterization of the optimal sphere $(R^*, \boldsymbol{c}^*)$. As the optimal center is given by the linear combination $c^* = \sum_i \alpha_i \boldsymbol{x}_i$ and R^* can be recovered from any $\boldsymbol{x}_i$ with $0 < \alpha_i < \frac{1}{\nu n}$, the optimal description solely depends on data points with $\alpha_i > 0$. Also, the value of the dual objective (3.11) only depends on examples with $\alpha_i > 0$ as well. This is the reason why every $\boldsymbol{x}_i$ with $\alpha_i > 0$ is called a *support vector*, analogous to the classic SVM. The support vectors are the only examples required to infer the data description. As we examined above, every vector that lies outside the sphere is a support vector and examples on the boundary can be support vectors. Conversely, every support vector must either lie on the boundary or outside the sphere. As we will see below, those two states a support vector can have are important to deduce the role of the ν-parameter.

Anomaly detection in SVDD As examples that lie outside the sphere are considered to be anomalous, the natural anomaly score for some input $\boldsymbol{x} \in \mathcal{X}$ to define from the

dual is

$$\begin{aligned} s(\boldsymbol{x}) &= \|\boldsymbol{x} - \boldsymbol{c}^*\|^2 - R^{*2} = \|\boldsymbol{x} - \sum_i \alpha_i \boldsymbol{x}_i\|^2 - R^{*2} \\ &= \|\boldsymbol{x}\|^2 - 2\sum_i \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}\rangle + \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle - R^{*2} \\ &= \|\boldsymbol{x}\|^2 - 2\sum_i \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x}\rangle + C, \end{aligned} \tag{3.15}$$

which means the greater the score, the more anomalous an example is to be suspected. $C \in \mathbb{R}$ only must be computed once or does not have to be computed at all if only the ranking, but not the exact scores (with a threshold on anomalies), is of interest. The threshold or decision function for some input $\boldsymbol{x} \in \mathcal{X}$ can then defined as

$$f(\boldsymbol{x}) = \operatorname{sign}(R^{*2} - \|\boldsymbol{x} - \boldsymbol{c}^*\|^2) = \operatorname{sign}(-s(\boldsymbol{x})), \tag{3.16}$$

i.e. outliers are flagged by -1 and examples from the target class by 1 with the convention $\operatorname{sign}(0) = 1$, i.e. examples on the boundary are considered to be part of the target class. For an outlier $\boldsymbol{x}_i$ in the training set $\mathcal{D}_n$, the anomaly score therefore is given exactly by its slack penalty $\xi_i > 0$ as the optimal value of ξ_i is given by the distance to the boundary as it is the minimal value such that the inequality constraint is satisfied, i.e.

$$\xi_i^* = \max\{\|\boldsymbol{x}_i - \boldsymbol{c}^*\|^2 - R^{*2}, 0\}. \tag{3.17}$$

Kernel SVDD

As we can see, solving the dual (3.11) and computing the anomaly score (3.15) only requires information on the dot products between data points. Hence, the kernel trick can be applied. Let $\boldsymbol{K}$ denote the kernel matrix on training data $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ for some (Mercer) kernel k with corresponding, implicitly defined feature map $\phi : \mathcal{X} \to \mathcal{F}$, that is

$$\boldsymbol{K} = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1,\ldots,n} = (\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j)\rangle_{\mathcal{F}})_{i,j=1,\ldots,n}\,. \tag{3.18}$$

Then, the kernel SVDD dual objective can be formulated as

Problem 3.3 (Kernel SVDD dual)**.**

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_i \alpha_i K_{ii} - \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_i \alpha_i = 1, \\ & 0 \le \alpha_i \le \frac{1}{\nu n}, \quad \forall i = 1, \ldots, n. \end{aligned} \tag{3.19}$$

The corresponding kernel SVDD primal is given by

Problem 3.4 (Kernel SVDD primal)**.**

$$\begin{aligned} \min_{R,\boldsymbol{c},\boldsymbol{\xi}} \quad & R^2 + \frac{1}{\nu n}\sum_i \xi_i \\ \text{s.t.} \quad & \|\phi(\boldsymbol{x}_i) - \boldsymbol{c}\|_{\mathcal{F}}^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, n, \end{aligned} \tag{3.20}$$

with the data-enclosing hypersphere fit in feature space $\mathcal{F}$.

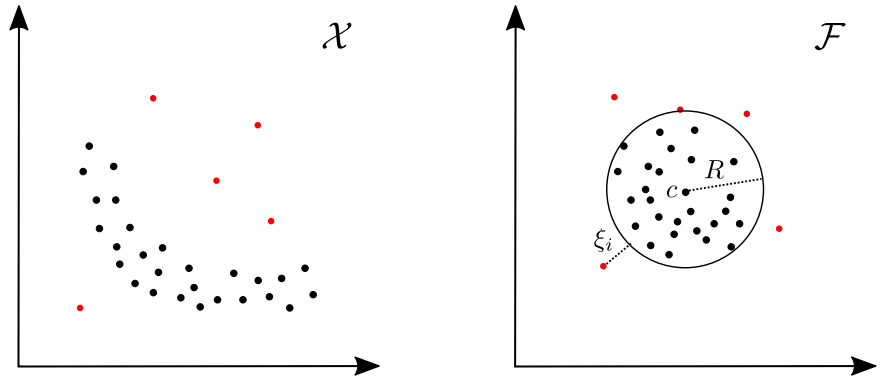


Figure 3.1: Illustration of kernel SVDD: Feature map $\phi : \mathcal{X} \to \mathcal{F}$ maps the data to feature space where the hypersphere is fit.

From (3.8), it follows that the optimal center $\boldsymbol{c}^*$ of the hypersphere in feature space is given by

$$\boldsymbol{c}^* = \sum_i \alpha_i \phi(\boldsymbol{x}_i) \tag{3.21}$$

and the optimal radius R^* can be recovered analogously as in (3.14) from any support vector $\boldsymbol{x}_i$ with $0 < \alpha_i < \frac{1}{\nu n}$ by

$$R^{*2} = \|\phi(\boldsymbol{x}_i) - \boldsymbol{c}^*\|_{\mathcal{F}}^2 = K_{ii} - 2\,\boldsymbol{\alpha}^T \boldsymbol{K}_i + \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}, \tag{3.22}$$

where $\boldsymbol{K}_i$ is the i-th column (and the i-th row, as $\boldsymbol{K}$ is symmetric) of kernel matrix $\boldsymbol{K}$. To compute the anomaly score of some point $\boldsymbol{x} \in \mathcal{X}$, we get from (3.15)

$$\begin{aligned} s(\boldsymbol{x}) &= \|\phi(\boldsymbol{x}) - \boldsymbol{c}^*\|_{\mathcal{F}}^2 - R^{*2} = \|\phi(\boldsymbol{x}) - \sum_i \alpha_i \phi(\boldsymbol{x}_i)\|_{\mathcal{F}}^2 - R^{*2} \\ &= k(\boldsymbol{x}, \boldsymbol{x}) - 2\sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + \alpha^T \boldsymbol{K} \alpha - R^{*2} \\ &= k(\boldsymbol{x}, \boldsymbol{x}) - 2\sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + C, \end{aligned} \tag{3.23}$$

with $C = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} - R^{*2} \in \mathbb{R}$ independent of the data point $\boldsymbol{x} \in \mathcal{X}$ to evaluate. The term $\boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha}$ can be expensive to compute as it has order $\mathcal{O}(n^2)$ of complexity, but it only has to be computed once in training and then stored to be used for testing. Moreover, the complexity is more specifically quadratic in the number of support vectors as examples with $\alpha_i = 0$ are not required for computation, which makes computation cheaper.

The most prominent kernel applied in kernel methods is the Gaussian kernel (or radial base function (RBF) kernel) given by

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \exp\left(-\gamma \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2\right), \tag{3.24}$$

with smoothing hyperparameter $\gamma > 0$. We will investigate the case of the Gaussian kernel and its geometrical implications in more detail in section 3.3, where we will see that in case of the Gaussian kernel, SVDD and OC-SVM solve the equivalent problem.

Optimization

From Problem 3.3, we can see that the kernel SVDD dual is a convex quadratic objective with linear constraints, since kernel matrix $\boldsymbol{K}$ is a real symmetric, positive semi-definite matrix for a kernel k fulfilling Mercer's condition. Thus, it can be solved by using any quadratic programming optimization method, such as interior-point methods or subgradient-based methods (see e.g. Nocedal and Wright, 2006). The implementation in `LIBSVM` (Chang and Lin, 2011), a popular open source support vector machine library, on which `scikit-learn` (Pedregosa et al., 2011) provides an interface in the Python programming language, uses the sequential minimal optimization (SMO) algorithm (Platt, 1998) to solve the quadratic program.

3.2 One-Class Support Vector Machine

In comparison to the hyperspherical approach of SVDD, the objective in OC-SVM introduced by Schölkopf et al. (2001) is to find a maximum-margin separating *hyperplane*, which is more comparable to the traditional Support Vector Classifier by Vapnik (1998). But as we will see below, the two approaches are in fact identical when using certain kernels, most importantly when using the Gaussian kernel.

Primal objective

The key idea of OC-SVM is to separate the target class by hyperplane with maximum-margin from the origin, i.e. samples from the target class should be linearly separated with a hyperplane that has maximum distance from the origin. It may not seem intuitive or clear, why anomalies should be found on the side of the hyperplane, that is closer to the origin. Here, it is important to remind that the goal in mind is to perform the separation via hyperplane again in some feature space induced by some kernel.

To introduce the primal problem of OC-SVM, let again $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a training set without labels on some input space $\mathcal{X} \subset \mathbb{R}^d$ from which we would like to deduce a description or one-class classification of the target class. For a hyperplane in the input space characterized by $\boldsymbol{w} \in \mathcal{X}$, which separates a data point $\boldsymbol{x}_i \in \mathcal{D}_n$ at least by a margin of $\rho > 0$, it must hold that

$$\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq \rho. \tag{3.25}$$

Analogously to SVDD and SVMs in general, we soften the hard-margin again by allowing violations through introducing slack variables ξ_i, that is we ask for

$$\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq \rho - \xi_i \tag{3.26}$$

to hold for all $i = 1, \ldots, n$ with $\xi_i \geq 0$ and the goal will be to minimize slack violations. As in classical hyperplane SVMs for binary classification, the structural error in OC-SVM is measured by $\|\boldsymbol{w}\|$. This motivates to formulate the primal problem of OC-SVM as follows:

Problem 3.5 (OC-SVM primal)**.**

$$\begin{aligned} \min_{\boldsymbol{w},\rho,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 - \rho + \frac{1}{\nu n}\sum_i \xi_i \\ \text{s.t.} \quad & \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, n, \end{aligned} \tag{3.27}$$

again with hyperparameter $\nu \in (0, 1]$ which controls the weight of the slack penalization and thereby gives control on the fraction of outliers and the fraction of support vectors in the model as we will show in section 3.4. A similar formulation of the primal would be

$$\begin{aligned} \max_{\boldsymbol{w},\rho,\boldsymbol{\xi}} \quad & \rho - \frac{1}{\nu n}\sum_i \xi_i \\ \text{s.t.} \quad & \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, n, \\ & \|\boldsymbol{w}\| = 1, \end{aligned} \tag{3.28}$$

where $\boldsymbol{w}$ is constraint to have unit length.

Dual objective

Derivation of the OC-SVM dual As the primal Problem 3.5 has a quadratic objective and affine inequality constraints, that is the primal is a quadratic problem and satisfies the linearity constraint qualification, strong duality holds. (cf. again appendix A) The Lagrangian is given by

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}, \rho, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = & \frac{1}{2}\|\boldsymbol{w}\|^2 - \rho + \frac{1}{\nu n}\sum_i \xi_i \\ & - \sum_i \alpha_i \left(\langle \boldsymbol{w}, \boldsymbol{x}_i \rangle - \rho + \xi_i\right) - \sum_i \beta_i \xi_i, \end{aligned} \tag{3.29}$$

with Lagrange multipliers $\alpha_i \geq 0$ and $\beta_i \geq 0$. Taking the partial derivatives with respect to the primal variables and setting them to zero yields the following optimality conditions:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_i \alpha_i \boldsymbol{x}_i = 0 \qquad \Longrightarrow \qquad \boldsymbol{w} = \sum_i \alpha_i \boldsymbol{x}_i \tag{3.30}$$

$$\frac{\partial \mathcal{L}}{\partial \rho} = -1 + \sum_i \alpha_i = 0 \qquad \Longrightarrow \qquad \sum_i \alpha_i = 1 \tag{3.31}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = \frac{1}{\nu n} - \alpha_i - \beta_i = 0 \qquad \Longrightarrow \qquad \alpha_i = \frac{1}{\nu n} - \beta_i \tag{3.32}$$

Analogous to SVDD, the third condition (3.32) implies the bounds $0 \leq \alpha_i \leq \frac{1}{\nu n}$ and as the center of the hypersphere in SVDD, the characterization of the data-separating hyperplane only requires data points with $\alpha_i > 0$ as the optimal hyperplane is given by $\boldsymbol{w}^* = \sum_i \alpha_i \boldsymbol{x}_i$.

Substituting back the optimality conditions (3.30)–(3.32) into the Lagrangian (3.29) gives

$$\begin{aligned}
\mathcal{L} = \; & \frac{1}{2}\|\boldsymbol{w}\|^2 - \rho + \frac{1}{\nu n}\sum_i \xi_i - \sum_i \alpha_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \\
& + \rho \underbrace{\sum_i \alpha_i}_{=1} - \sum_i \underbrace{(\alpha_i + \beta_i)}_{=\frac{1}{\nu n}} \xi_i \\
= \; & \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i \langle \boldsymbol{w}, \boldsymbol{x}_i \rangle \\
= \; & \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\
= \; & -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle
\end{aligned} \tag{3.33}$$

Hence, the OC-SVM dual problem is given by

Problem 3.6 (OC-SVM dual)**.**

$$\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2}\sum_{i,j} \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \\
\text{s.t.} \quad & \sum_i \alpha_i = 1, \\
& 0 \leq \alpha_i \leq \frac{1}{\nu n}, \quad \forall i = 1, \ldots, n.
\end{aligned} \tag{3.34}$$

Support vectors in OC-SVM The set of vectors $\{\boldsymbol{x}_i \in \mathcal{D}_n \,|\, \alpha_i > 0\}$ are again called the *support vectors*. For the hyperplane-based OC-SVM, there are again three cases for

a data point $\boldsymbol{x}_i \in \mathcal{D}_n$ to be in:

(1) Above the hyperplane: $\langle \boldsymbol{x}_i, \boldsymbol{w}^* \rangle > \rho^*$
(2) On the hyperplane: $\langle \boldsymbol{x}_i, \boldsymbol{w}^* \rangle = \rho^*$
(3) Below the hyperplane: $\langle \boldsymbol{x}_i, \boldsymbol{w}^* \rangle < \rho^*$

Completely analogous to the derivation in SVDD, it follows from the KKT conditions for optimality that every support vector must either lie below or on the separating hyperplane. Likewise, if for a support vector $\boldsymbol{x}_i$ it holds that $0 < \alpha_i < \frac{1}{\nu n}$ (and therefore also $\beta_i > 0$), then the corresponding penalty must be zero, i.e. $\xi_i^* = 0$, and therefore $\boldsymbol{x}_i$ lies on the hyperplane. This means that the optimal ρ^* can be recovered from any support vector $\boldsymbol{x}_i$ with $0 < \alpha_i < \frac{1}{\nu n}$ via

$$\rho^* = \langle \boldsymbol{w}^*, \boldsymbol{x}_i \rangle = \sum_j \alpha_j \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle. \tag{3.35}$$

Anomaly detection in OC-SVM As outliers lie below the separating hyperplane, that is closer to the origin, the OC-SVM anomaly score function is given by

$$s(\boldsymbol{x}) = \rho^* - \langle \boldsymbol{w}^*, \boldsymbol{x} \rangle = \rho^* - \sum_i \alpha_i \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle, \tag{3.36}$$

and the corresponding threshold or decision function f for some input $\boldsymbol{x} \in \mathcal{X}$ is defined as

$$f(\boldsymbol{x}) = \operatorname{sign}(\langle \boldsymbol{w}^*, \boldsymbol{x} \rangle - \rho^*) = \operatorname{sign}(-s(\boldsymbol{x})), \tag{3.37}$$

again with examples on the boundary considered to be part of the target class.

Kernel OC-SVM

As solving the OC-SVM dual (3.34) again only involves scalar products, it can be kernelized leading to the following kernel OC-SVM dual:

Problem 3.7 (Kernel OC-SVM dual)**.**

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \\ \text{s.t.} \quad & \sum_i \alpha_i = 1, \\ & 0 \leq \alpha_i \leq \frac{1}{\nu n}, \quad \forall i = 1, \ldots, n, \end{aligned} \tag{3.38}$$

with kernel matrix

$$\boldsymbol{K} = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1,\ldots,n} = (\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle_{\mathcal{F}})_{i,j=1,\ldots,n} \tag{3.39}$$

corresponding to some kernel k which implicitly defines the feature map $\phi : \mathcal{X} \to \mathcal{F}$. The corresponding kernelized primal problem is

Problem 3.8 (Kernel OC-SVM dual)**.**

$$\begin{aligned} \min_{\boldsymbol{w},\rho,\boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|_{\mathcal{F}}^2 - \rho + \frac{1}{\nu n}\sum_i \xi_i \\ \text{s.t.} \quad & \langle \boldsymbol{w}, \phi(\boldsymbol{x}_i)\rangle_{\mathcal{F}} \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad \forall i = 1, \ldots, n, \end{aligned} \tag{3.40}$$

with separating hyperplane $\boldsymbol{w} \in \mathcal{F}$ fit in feature space.

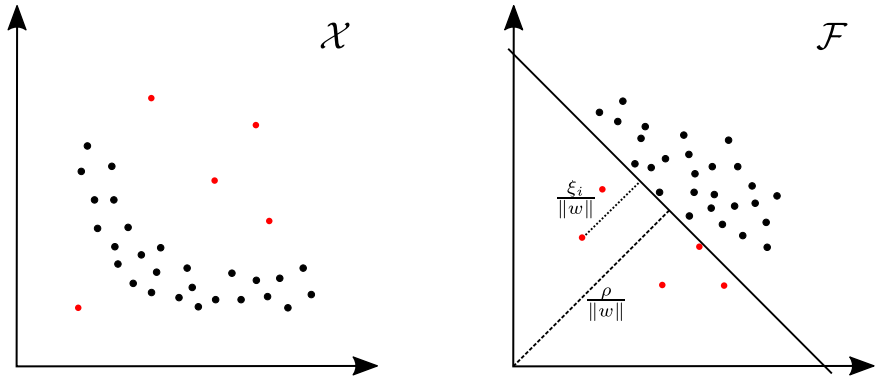


Figure 3.2: Illustration of kernel OC-SVM: Feature map $\phi : \mathcal{X} \to \mathcal{F}$ maps the data to feature space where the hyperplane is fit.

From (3.30) the optimal separating hyperplane $\boldsymbol{w}^*$ in kernel OC-SVM then is given by

$$\boldsymbol{w}^* = \sum_i \alpha_i \phi(\boldsymbol{x}_i) \tag{3.41}$$

and from a support vector $\boldsymbol{x}_i$ with $0 < \alpha_i < \frac{1}{\nu n}$ lying on the hyperplane, the optimal ρ can again be recovered by

$$\rho^* = \langle \boldsymbol{w}^*, \boldsymbol{x}_i\rangle_{\mathcal{F}} = \sum_j \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_i). \tag{3.42}$$

Finally, anomaly score (3.36) can be computed in the kernelized version by

$$s(\boldsymbol{x}) = \rho^* - \langle \boldsymbol{w}^*, \phi(\boldsymbol{x})\rangle_{\mathcal{F}} = \rho^* - \sum_i \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) \tag{3.43}$$

for some sample $\boldsymbol{x} \in \mathcal{X}$.

Optimization

The kernel OC-SVM dual problem (3.38) also formulates a convex quadratic program and therefore it can be solved by using any off-the-shelf QP solver such as discussed

in the optimization of the kernel SVDD dual. As pointed out in the original OC-SVM introduction by Schölkopf et al. (2001), the simplicity of the constraints, however, can be exploited to improve the convergence of the SMO algorithm (see section 4 in Schölkopf et al., 2001, for details).

3.3 When SVDD and OC-SVM are equivalent

As already indicated, SVDD and OC-SVM solve similar problems. If we compare the kernelized dual problems of SVDD (3.19) and OC-SVM (3.38), we can see that the problems have identical constraints on $\boldsymbol{\alpha}$. The only difference is given in the cost functions, as the cost function in SVDD has an additional term:

$$J_{\mathrm{SVDD}}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} - \sum_i \alpha_i K_{ii} \tag{3.44}$$

$$J_{\mathrm{OCSVM}}(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^T \boldsymbol{K} \boldsymbol{\alpha} \tag{3.45}$$

where $\boldsymbol{K}$ is again the kernel matrix of some kernel k. The additional term shows that in SVDD, for the optimal weights α_i of the support vectors, the lengths of the support vectors in feature space $K_{ii} = \|\phi(\boldsymbol{x}_i)\|^2_{\mathcal{F}}$ are also taken into account, where $\|\cdot\|_{\mathcal{F}}$ is the norm induced by the dot product defined by kernel k.

From this we can deduce the property a kernel function k must have in order for SVDD and OC-SVM to be equivalent, namely the feature map $\phi : \mathcal{X} \to \mathcal{F}$ must map all data points onto a sphere centered at the origin, that is every mapping $\phi(\boldsymbol{x})$ of an input $\boldsymbol{x}$ must have the same norm in feature space:

$$\|\phi(\boldsymbol{x})\|^2_{\mathcal{F}} = k(\boldsymbol{x}, \boldsymbol{x}) = C > 0 \quad \text{for every } \boldsymbol{x} \in \mathcal{X}. \tag{3.46}$$

If the norms of all training examples are equal in feature space, the additional term in the SVDD objective becomes constant

$$\sum_i \alpha_i K_{ii} = C \sum_i \alpha_i = C \tag{3.47}$$

and thus can be omitted from the SVDD dual optimization problem.

Equivalence for the Gaussian kernel Property (3.46) is fulfilled by the most prominently used kernel, the Gaussian kernel, given by

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \exp\left(-\gamma \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2\right), \quad \gamma > 0, \quad \boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X}. \tag{3.48}$$

Since

$$\|\phi(\boldsymbol{x})\|^2_{\mathcal{F}} = k(\boldsymbol{x}, \boldsymbol{x}) = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{x}\|^2\right) = 1, \tag{3.49}$$

for every $\boldsymbol{x} \in \mathcal{X}$, the feature map ϕ implicitly defined by the Gaussian kernel maps every input onto the unit sphere in feature space $\mathcal{F}$. More specifically, because

$$\langle \boldsymbol{x}, \tilde{\boldsymbol{x}} \rangle_{\mathcal{F}} = k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \exp\left(-\gamma \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2\right) > 0 \tag{3.50}$$

for all $\boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X}$, that is all scalar products are strictly positive, the feature map ϕ maps all inputs onto the segment of the unit sphere in feature space that lies in the positive orthant.

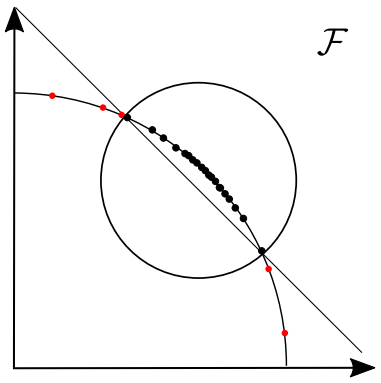


Figure 3.3: Equivalence of SVDD and OC-SVM illustrated.

The equivalence of a minimum-volume enclosing hypersphere and a maximum-margin (w.r.t. the origin) hyperplane in feature space, if all data lies on the unit sphere, is also geometrically plausible: to find the data-enclosing sphere of minimum volume, the smallest segment of the sphere the data lies on has to be determined. But such a segment is exactly found by intersecting the sphere with a hyperplane and from all the hyperplanes that cut off a segment that has all data on it, the hyperplane with maximal distance from the origin is the one cutting off the smallest segment.

3.4 The ν-property and Theoretical Results

In this section, we provide the justification for why the ν-parameter with $\nu \in (0, 1]$ is a nice choice of parameterization in the context of kernel-based one-class classification for controlling the weight of the slack penalty in both, the formulation of SVDD and OC-SVM, as both (kernelized) primal problems (3.20) and (3.40) incorporate the slack penalization term as

$$\frac{1}{\nu n} \sum_i \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n. \tag{3.51}$$

In addition, we will give main theoretical results from the literature on kernel-based one-class classification.

The ν-property We formulate the property of the ν hyperparameter used in our formulations of SVDD and OC-SVM in the following theorem:

Theorem 3.1 (ν-property)**.** *Assume the solutions of the (kernel) problems* (3.20) *and* (3.40) *satisfy $R^* > 0$ and $\rho^* \neq 0$ respectively (i.e. the solutions are not degenerated into a single point). Then, the following statements on $\nu \in (0, 1]$ hold:*

(i) ν is an upper bound on the fraction of outliers.

(ii) ν is a lower bound on the fraction of support vectors.

(iii) Suppose the training data $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ was generated independently from the identical distribution P that has zero probability mass on discrete components. Moreover, suppose that kernel k is analytic and not constant. Then, with probability 1, asymptotically, ν equals both the fraction of outliers and the fraction of support vectors.

Proof. A training example $\boldsymbol{x}_i \in \mathcal{D}_n$ is an outlier, if it lies outside the data-enclosing hypersphere or below the data-separating hyperplane respectively, that is

$$\|\phi(\boldsymbol{x}_i) - c^*\|_{\mathcal{F}}^2 > R^{*2} \qquad \text{or} \qquad \langle w^*, \phi(\boldsymbol{x}_i)\rangle_{\mathcal{F}} < \rho^*. \tag{3.52}$$

For the KKT optimality conditions to be satisfied, precisely the feasibility of the primal constraints, the corresponding slack penalty ξ_i of an outlier must be positive, that is the set of outliers in the training set is given by

$$S_{\text{out}} = \{\boldsymbol{x}_i \in \mathcal{D}_n \mid \xi_i > 0\}. \tag{3.53}$$

By the slackness KKT condition (3.13), this implies for the Lagrange multipliers of an outlier $\boldsymbol{x}_i \in S_{\text{out}}$ that $\beta_i = 0$ and thus $\alpha_i = \frac{1}{\nu n}$. Therefore we get

$$\begin{aligned} 1 = \sum_i \alpha_i &= \sum_{i:\boldsymbol{x}_i \in S_{\text{out}}} \alpha_i + \sum_{i:\boldsymbol{x}_i \in S_{\text{out}}^{\complement}} \alpha_i \\ &\geq \sum_{i:\boldsymbol{x}_i \in S_{\text{out}}} \alpha_i = n_{\text{out}} \cdot \frac{1}{\nu n} \end{aligned} \tag{3.54}$$

as the Lagrange multipliers are non-negative which is implies

$$\frac{n_{\text{out}}}{n} \leq \nu, \tag{3.55}$$

where $n_{\text{out}} = |S_{\text{out}}|$ is the number of outliers and thus concludes (i).

The set of support vectors is defined as

$$S_{\text{sv}} = \{\boldsymbol{x}_i \in \mathcal{D}_n \mid \alpha_i > 0\}, \tag{3.56}$$

where each support vector either is an outlier or lies exactly on the decision boundary. As non-support vectors have $\alpha_i = 0$, we get that

$$1 = \sum_i \alpha_i = \sum_{i:\boldsymbol{x}_i \in S_{\text{sv}}} \alpha_i \leq n_{\text{sv}} \cdot \frac{1}{\nu n}, \tag{3.57}$$

since $0 < \alpha_i \leq \frac{1}{\nu n}$ for every support vector and thus

$$\frac{n_{\text{sv}}}{n} \geq \nu, \tag{3.58}$$

where $n_{\text{sv}} = |S_{\text{sv}}|$ which concludes statement (ii)

Statement (iii) is conducted by a uniform convergence argument showing that the fraction of points lying exactly on the boundary is asymptotically negligible under the assumption that P has no probability mass on countable sets. That is, the support vectors on the boundary asymptotically have zero probability mass and therefore all the probability mass of support vectors is given by outlier-support vectors. Thus, from statements (i) and (ii) the probability mass for support vectors and outliers is exactly given by ν. The proof uses that the covering numbers of the kernel expansions are well behaved under the assumptions made. See Schölkopf et al. (2001) and section 7 in Schölkopf et al. (2000) for further details. □

The ν-property is quite powerful in the context of anomaly detection as it allows to a priori incorporate a belief on the fraction of outliers assumed to be in the data. Therefore, the sensitivity of the learned description to anomalies but also to atypical examples of the target class itself can be adjusted. For example in applications that are more conservative, for instance if not detecting an anomaly is costly, ν can be set to a greater value (e.g. $\nu = 0.5$) which gives a tight description of the "most typical" examples an thus outliers will be detected with great probability. That is, for larger values of ν very few or zero false negatives (i.e. true anomalies that are not recognized as such) are to be expected of course at the cost of a greater number of false positives (i.e. true examples of the target class predicted to be anomalous). The concrete selection of ν thus depends on the specific goal and setting of an application as well as on the assumptions about the possible distributions anomalies could have. If strongly overlapping tails are suspected, a tighter description (i.e. larger ν) should be applied. If anomalies (or novelties) are expected to exhibit distributions of greater difference, a smaller value can be used. A commonly used starting point is $\nu = 0.1$, but because of the reasons given above, the selection always has to be made with the application in mind and is, because of its dependence on the unknown distributions of anomalies, generally not an easy task.

The two extreme special cases of ν are given by $\nu < \frac{1}{n}$ (i.e. no example in the training data is modeled to be an outlier) and $\nu = 1$ (i.e. all examples are support vectors). The first case which assumes that there are no outliers in the training data, implies that $\xi_i = 0$ must hold for every $\boldsymbol{x}_i \in \mathcal{D}_n$. (for $\nu < \frac{1}{n}$, no Lagrange multiplier α_i reaches the

upper bound of $\frac{1}{\nu n} > 1$ as $\alpha_i \leq 1$ by condition $\sum_i \alpha_i = 1$) Thus, SVDD and OC-SVM fall back in this case to its hard-boundary and hard-margin formulations respectively, that is all data points of the training set must lie within or on the data-enclosing hypersphere and above or on the data-separating hyperplane.

In the second case of $\nu = 1$, since $0 \leq \alpha_i \leq \frac{1}{\nu n} = \frac{1}{n}$ and because $\sum_i \alpha_i = 1$ must hold, the only feasible solution is given by $\alpha_i = \frac{1}{n}$ for all $i = 1, \ldots, n$. This means that all examples are support vectors and by (3.21) and (3.41) the optimal center of the hypersphere as well as the hyperplane in feature space are given by the mean over all samples mapped into feature space, that is

$$c^* = w^* = \frac{1}{n} \sum_{i=1}^{n} \phi(\boldsymbol{x}_i). \tag{3.59}$$

Moreover, if k is the Gaussian kernel with hyperparameter $\gamma > 0$, we get for the anomaly score of some input $\boldsymbol{x} \in \mathcal{X}$ for SVDD from (3.23)

$$\begin{aligned} s_{\text{svdd}}(\boldsymbol{x}) &= k(\boldsymbol{x}, \boldsymbol{x}) - 2 \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) + \boldsymbol{\alpha}^T K \boldsymbol{\alpha} - R^{*2} \\ &= -\frac{2}{n} \sum_{i=1}^{n} \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}_i\|^2) + C_1 \end{aligned} \tag{3.60}$$

and for OC-SVM from (3.43)

$$\begin{aligned} s_{\text{ocsvm}}(\boldsymbol{x}) &= \rho^* - \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}) \\ &= -\frac{1}{n} \sum_{i=1}^{n} \exp(-\gamma \|\boldsymbol{x} - \boldsymbol{x}_i\|^2) + C_2 \end{aligned} \tag{3.61}$$

for constants $C_1, C_2 \in \mathbb{R}$.

Relation of SVDD and OC-SVM to KDE Now, if we have a look at the Parzen windows estimator (Rosenblatt et al., 1956, Parzen, 1962), that is the kernel density estimator with a Gaussian kernel, which is given by

$$\hat{f}(\boldsymbol{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h} \right), \quad \boldsymbol{x} \in \mathcal{X}, \tag{3.62}$$

where d is the dimensionality of $\mathcal{X}$ and $h > 0$ is the bandwidth, then we can see that by naturally defining the anomaly score of the kernel density estimator for $\boldsymbol{x} \in \mathcal{X}$ to be

$$s_{\text{kde}}(\boldsymbol{x}) = -\hat{f}(\boldsymbol{x}) = -\frac{1}{nh^d} \sum_{i=1}^{n} \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h} \right), \tag{3.63}$$

that is the smaller the estimated density value the more likely $\boldsymbol{x} \in \mathcal{X}$ is suspected to be anomalous, then all three scores (3.60), (3.61), and (3.63) imply exactly the same anomaly *ranking* for inputs $\boldsymbol{x} \in \mathcal{X}$ if $\gamma = \frac{1}{h}$ as they are translated and scaled versions of the basic Gaussian kernel expansion

$$s(\boldsymbol{x}) = -\frac{1}{n}\sum_{i=1}^{n} \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}_i\|^2}{h}), \quad \boldsymbol{x} \in \mathcal{X} \tag{3.64}$$

on the training data set $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. Thus, in view of the task of anomaly detection, KDE corresponds to the special case of SVDD or OC-SVM with $\nu = 1$ and Gaussian kernel the only difference being in the choice of the anomaly threshold.

The following result of Munoz and Moguerza (2004) states the exact relation between SVDD or OC-SVM and kernel density estimation.

Theorem 3.2 (Precise relation of KDE to SVDD or OC-SVM)**.** *Consider the SVDD or OC-SVM with Gaussian kernel. Then, asymptotically, the points obtained as non-support vectors (i.e. inliers) in SVDD or OC-SVM are those closest to the mode estimator calculated on the set of $(1-\nu)n$ non-support vectors, using a kernel density estimator with Gaussian kernel (i.e. Parzen windows estimator).*

Proof. The proof is given in Munoz and Moguerza (2004). □

Density level set estimation To recognize SVDD and OC-SVM as density level set estimators as introduced in section 2.3, consider again the decision functions each method defines for $\boldsymbol{x} \in \mathcal{X}$:

$$f_{\text{svdd}}(\boldsymbol{x}) = \text{sign}(R^{*2} - \|\phi(\boldsymbol{x}) - c^*\|_{\mathcal{F}}^2) \tag{3.65}$$

and

$$f_{\text{ocsvm}}(\boldsymbol{x}) = \text{sign}(\langle w^*, \phi(\boldsymbol{x})\rangle_{\mathcal{F}} - \rho^*). \tag{3.66}$$

Under the assumption that the data is realized from i.i.d. random variables $X_1, \ldots, X_n$ following some distribution P with density p (that has zero probability mass on countable sets) we get from Theorem 3.1 (iii) (under the additional smoothness assumptions on kernel k given) that the estimator defined by

$$\hat{C}_\nu = \{\boldsymbol{x} \in \mathcal{X} \,|\, f(\boldsymbol{x}) \geq 0\}, \tag{3.67}$$

where f is either the decision function of SVDD or OC-SVM, does have probability mass $1-\nu$ under P asymptotically, as the set of outliers asymptotically has probability mass ν. However, the asymptotic convergence of $\hat{C}_\nu$ to the desired probability of $1-\nu$ does not already imply consistency as a density level set estimator, that is that $\hat{C}_\nu$ converges in probability to the density level set

$$C_\nu = \{\boldsymbol{x} \in \mathcal{X} \,|\, p(\boldsymbol{x}) \geq b_\nu\}, \tag{3.68}$$

with $b_\nu > 0$ such that $P(X \in C_\nu) = 1 - \nu$. Consistency is only given if $\hat{C}_\nu$ asymptotically has the desired probability of $1 - \nu$ and in addition is the set of minimum volume (in the Lebesgue-sense) having the probability mass of $1 - \nu$ asymptotically. Vert and Vert (2006) show that $\hat{C}_\nu$ indeed is a consistent density level set estimator when the Gaussian kernel is used with the bandwidth (sufficiently fast) converging to zero. They also give results on the convergence rate achieved. The proof is based on results in minimum volume set estimation (Scott and Nowak, 2006), as the ν-density level set is exactly characterized as the set with probability mass $1 - \nu$ that has smallest volume.

Further theoretical results in the literature on SVDD and the OC-SVM are the relation to boosting (Rätsch et al., 2000), a classification framework (Steinwart et al., 2005), and the OC-SVM solution path (Hastie et al., 2004, Lee and Scott, 2007).

3.5 Summary and Discussion

In the above, the two kernel-based approaches to density level set estimation for the task of anomaly detection in the one-class classification setting have been introduced. We have seen that the two methods are equivalent for the Gaussian kernel, which is the kernel having sound theoretical properties as shown in section 3.4 and is also the kernel most relevant in application. Moreover, the justification and the benefits for the task of anomaly detection in specifying SVDD and the OC-SVM in the ν-hyperparameter formulation have been shown.

The good The two kernel-based approaches to one-class classification have several nice properties. First, both are convex optimization problems and therefore have globally optimal solutions. Furthermore, they incorporate the ν-property given in Theorem 3.1 which allows to account for anomalies present in the training set. Employing kernels moreover allows non-linear feature transformations capturing non-linear patterns in the data. For the case of the Gaussian kernel, for which SVDD and OC-SVM are equivalent, the kernel-based approaches are asymptotically consistent density level set estimators.

The bad On the other hand, one of the major drawbacks of the kernel methods is that they scale quadratically in training set size n if non-linear kernels are used limiting their applicability to large-scale or online anomaly detection tasks. Moreover, they have bad memory complexity in the sense that all support vectors of a model are needed for making predictions. In addition, the kernel-induced feature representations are fixed and cannot be learned. Thus, for the methods to perform and generalize well, manual feature engineering and selection with expert and domain knowledge still often is required as an additional pre-processing step (Pal and Foody, 2010). Moreover, kernel methods are also sensitive to the curse of dimensionality outlined in section 2.2, more precisely if the data does not lie close to a smooth manifold but on a manifold with many variations (i.e. of

high curvature) local smoothing kernels such as the Gaussian kernel require a number of training examples proportional to the number of such variations (cf. Bengio et al., 2006). Also, local smoothing methods do not generalize well far from the training data. However, if the smooth manifold assumption holds, kernel-based methods are generally the methods of choice in high-dimensional settings with small sample sizes where they show good generalization performance.

In order to tackle those drawbacks given above and develop a method for the task of large-scale and high-dimensional anomaly detection, the idea of deep SVDD is to move from fixed kernel mappings to learnable, hierarchical representations via deep networks. For this purpose, we will give an overview on deep learning in the next section where we also review the different approaches to deep anomaly detection taken in the literature.

4 Deep Learning

Deep Learning is the field in representation learning utilizing model architectures with multiple processing layers to learn representations of data with multiple layers of abstraction (Bengio et al., 2013, Schmidhuber, 2015, LeCun et al., 2015, Goodfellow et al., 2016). Another term for deep learning is *hierarchical learning*. There is no standard definition of how many layers a model architecture must have to be considered "deep". Due to the attention deep learning currently is attracting in research, the term to some degree also serves scientific marketing purposes. Many authors define "deep" simply by model architectures having more than one computational layer, whereas models with just one layer are coherently called "shallow", but usually authors have architectures with numerous hidden layers in mind when calling them "deep" (Schmidhuber, 2015). The most basic deep learning model architecture would be a simple feed-forward artificial neural network with multiple hidden layers, also called Multilayer Perceptron (MLP). Methods of deep learning improved state-of-the-art results in numerous areas of research, with record-breaking leaps especially in computer vision (Krizhevsky et al., 2012, Farabet et al., 2013, Simonyan and Zisserman, 2014, Taigman et al., 2014, Szegedy et al., 2015, He et al., 2016), speech and audio processing (Mikolov et al., 2011, Hinton et al., 2012, Sainath et al., 2013), and natural language processing (Collobert, Weston, Bottou, Karlen, Kavukcuoglu and Kuksa, 2011, Sutskever et al., 2014, Jean et al., 2015).

What follows is a an overview of the field of deep learning explaining the fundamentals, touching on various aspects, and describing open challenges in the field. Convolutional neural networks are reviewed in a separate subsection, as they are the architecture used with deep SVDD in the experiments given in section 6. The section closes with a review on deep learning approaches in anomaly detection.

4.1 Deep Learning: an Overview

Learning representations The performance of machine learning models depends on the data representations (i.e. features) properly exhibiting the different explanatory factors of variation behind the data (Bengio et al., 2013). Domain knowledge of experts can be used for manually designing and extracting good features for a given objective, which then can be fed into a model. The more general-purpose approach of *representation learning* is

to go one step further, that is, besides learning a final model on the data representations, also to learn the representations themselves. Hence, the aim in representation learning is to disentangle the factors of variation underlying the data automatically by learning a feature transformation as opposed to designing features by hand. Thereby, learning a good feature transformation involves both, feature extraction and feature engineering.

Deep learning is a specific way of representation learning, adopting the concept of *hierarchical* or *distributed* representations. This concept is realized by artificial neural network architectures composed from multiple hidden layers inducing different *levels of abstraction* in representation. The lowest layers extract and learn the simplest of features from the data. Going up, the composition of such simple features over multiple layers enable very complex feature representations in the higher levels of the network. A concrete example of such features with different levels of abstraction from learning hierarchical representations of images would be to go from pixels (raw input) to edges to simple patterns to parts of objects to objects and finally to the composition of multiple objects, where each higher level feature is composed from lower level features beneath.

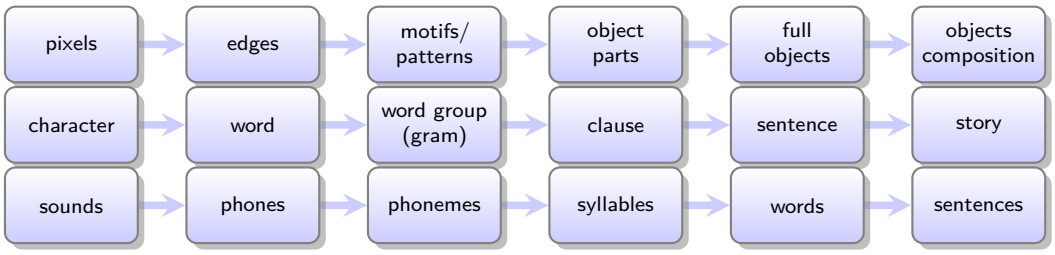


Figure 4.1: Examples of data that is composed hierarchically: images, text, and speech.

Neural networks Artificial neural network models are the basis for deep learning and are inspired by and loosely connected to our understanding of how the biological brain works (Rosenblatt, 1958). Let $\mathcal{X} \subset \mathbb{R}^n$ be some input space, then a neural network $f(\cdot\,; \mathcal{W}) : \mathcal{X} \to \mathbb{R}^p$ is a function depending on a set of parameters $\mathcal{W}$ that is composed from multiple simple functions, where each intermediate output of such a simple function in the composition constitutes a so-called *layer* and each element in the layer is a so-called *unit*. To empower artificial neural networks to approximate complex non-linear functions, each unit in a network is composed of a linear function and a (usually) non-linear activation function. This basic composition dates back to the 1960s and the ADALINE (Adaptive Linear Element) (Widrow and Hoff, 1960). Considering a neural network with $L \in \mathbb{N}$ layers, the output of some layer $\ell \in \{1, \ldots, L\}$ is given by

$$\boldsymbol{z}^{\ell} = \sigma^{\ell}(\boldsymbol{W}^{\ell} \cdot \boldsymbol{z}^{\ell-1}), \tag{4.1}$$

where vector $\boldsymbol{z}^{\ell-1}$ denotes the output of the previous layer (with $\boldsymbol{z}^0 \in \mathcal{X}$ being the data input layer and $\boldsymbol{z}^L \in \mathbb{R}^p$ being the output of the network), matrix $\boldsymbol{W}^\ell$ the parameters or *weights* of layer ℓ, and $\sigma^\ell(\cdot)$ the non-linear activation function of layer ℓ applied element-wise. Operator "$\cdot$" denotes the linear operation, usually a simple matrix multiplication, but for example a convolution in the case of convolutional neural networks. If "$\cdot$" is a matrix multiplication, each row of $\boldsymbol{W}^\ell$ corresponds to the weights of one unit in layer ℓ and layer ℓ is said to be *fully-connected* or *dense* as each unit has one weight for each output unit of the previous layer. If layer ℓ also has trainable bias terms $\boldsymbol{b}^\ell$, capable of shifting the input to the activation function in each unit, the output of layer ℓ is given by

$$\boldsymbol{z}^\ell = \sigma^\ell(\boldsymbol{W}^\ell \cdot \boldsymbol{z}^{\ell-1} + \boldsymbol{b}^\ell). \tag{4.2}$$

Finally, defining $f^\ell(\boldsymbol{z}) = \sigma^\ell(\boldsymbol{W}^\ell \cdot \boldsymbol{z} + \boldsymbol{b}^\ell)$ for $\ell \in \{1, \dots, L\}$, the overall network is given by the composition

$$f(\boldsymbol{x}; \mathcal{W}) = f^L \circ \cdots \circ f^1(\boldsymbol{x}), \quad \boldsymbol{x} \in \mathcal{X}, \tag{4.3}$$

with the set of network parameters $\mathcal{W} = \{\boldsymbol{W}^1, \dots, \boldsymbol{W}^L\}$ given by the weights of all layers. Thus, a neural network transforms an original input $\boldsymbol{x} \in \mathcal{X}$ in multiple stages where in each layer an intermediate representation is given. As the representation of the next layer depends on the representation of the previous one, a neural network forms a hierarchy of representations with the level of abstraction increasing successively in every layer. To optimize the weights $\mathcal{W}$ of the network such that it learns useful representations capturing the factors of variation of some data distribution, usually (stochastic) gradient descent (or some variant of it) via the backpropagation algorithm is used on the objective of the estimation task (e.g. some empirical loss). We will elaborate on optimization in deep learning further below.

Activation functions The composition of such simple non-linear functions in neural networks is very powerful and capable of approximating very complex functions as the *universal approximation theorem* tells (Cybenko, 1989, Hornik et al., 1989, Csáji, 2001). Classic examples of non-linear activation functions are the *Sigmoid* function

$$\sigma(x) = \frac{1}{1 + \exp(-x)}, \quad x \in \mathbb{R}, \tag{4.4}$$

or the *tanh* function

$$\sigma(x) = \tanh(x) = \frac{2}{1 + \exp(-2x)} - 1, \quad x \in \mathbb{R}, \tag{4.5}$$

but the currently most used activation function in deep networks is the piecewise linear *Rectified Linear Unit (ReLU)* given by

$$\sigma(x) = \max\{0, x\}, \quad x \in \mathbb{R}, \tag{4.6}$$

since it shows fast convergence in optimizing deep networks with many layers, favors sparse unit activations, and moreover alleviates the need for unsupervised pre-training of network parameters in supervised tasks (Glorot et al., 2011). A recent extension of the ReLU by Maas et al. (2013) is the *leaky ReLU* activation function given by

$$\sigma(x) = \begin{cases} \alpha\, x & \text{for } x < 0, \\ x & \text{for } x \geq 0, \end{cases} \quad x \in \mathbb{R}, \tag{4.7}$$

with "leakiness-parameter" $\alpha \in (0, 1]$, which usually is a small positive number (e.g. $\alpha = 0.01$). By allowing a small non-zero gradient for a negative input, the leaky ReLU helps convergence as it reduces the "dying ReLU" problem, which occurs when a ReLU unit is inactive (i.e. zero) for most of the inputs, for example due to an unfortunate parameter initialization or a learning rate in gradient descent that was set too high. With zero gradient for negative inputs, a regular ReLU unit cannot be recovered from such a state in backpropagation and "dies" whereas a small nonzero gradient allows recovering the unit.

Network architectures The simplest deep architecture is a multi-layered *feed-forward neural network (FFN)* with fully-connected layers. The class of architectures that achieved breakthroughs and are now standard in computer vision are *convolutional neural networks (CNNs)*, which introduce parameter sharing across layer inputs and the pooling of features, accounting for local correlations in images and the invariance of local patches to their global position in an image. We will introduce CNNs more closely in subsection 4.2.

A further important class of architectures, which we will not focus on in this work, are *recurrent neural networks (RNNs)* which incorporate directed cycles (i.e. loops) into the network architecture in addition to strict feed-forward connections as given in standard networks. Such networks are suited to sequential data, such as time series or text and speech, as they keep information from the history of a sequence. To train RNNs with backpropagation, they have to be "unfolded in time". The resulting training algorithm is called backpropagation through time (BPTT) (Werbos, 1988, Mozer, 1989). Basic RNNs suffer from the problem of exploding or vanishing gradients in training over many time steps (Hochreiter, 1991) which can be tackled with gradient-clipping strategies. A major leap forward in RNNs to also detect long-range dependencies of sequences came from utilizing the *Long short-term memory (LSTM)* architecture introduced by Hochreiter and Schmidhuber (1997). LSTM architectures are currently state-of-the-art in many applications facing sequential data, such as machine translation (Sutskever et al., 2014) or language modeling (Jozefowicz et al., 2016).

Learning objectives Objectives in deep learning can be of all learning paradigms: supervised, semi-supervised or unsupervised. With the breakthrough applications of deep models in supervised learning in the recent years, deep learning research mainly focused

on supervised learning and employed objectives of unsupervised learning mostly as means to pre-train networks before training and fine-tuning them on supervised classification or regression objectives (Erhan et al., 2010). In this day and age, due to improvements such as the ReLU activation, unsupervised pre-training of supervised networks is not deemed necessary for the success of supervised deep models but rather considered a sophisticated weight initialization strategy (Glorot et al., 2011). In binary classification tasks, a final output layer consisting only of one single sigmoid neuron, modeling the probability of the input belonging to either one of the two classes, followed by the cross-entropy loss is commonly the objective. Analogously, in multi-class classification tasks, a softmax final layer, modeling the probabilities of the input belonging to each one of the multiple classes, followed by the categorical cross-entropy commonly constitutes the objective. In regression tasks, a linear activation followed by the mean squared error loss is common, though supervised deep learning applications are mainly classification tasks.

In unsupervised learning tasks, such as the one-class classification setting in anomaly detection, choosing an objective is less clear. If there are no ground truth labels available, the question of how to evaluate learned representations is more difficult (Bengio et al., 2013). One first probabilistic approach, still with the goal of pre-training networks for supervised tasks in mind, are *Deep Belief Nets (DBNs)* introduced by Hinton et al. (2006). DBNs learn hierarchical representations one layer at a time in a greedy fashion by considering each pair of successive connected layers as a Restricted Boltzmann Machine (RBM), where the lower layer constitutes the visible variables and the subsequent layer the latent variables, starting with the pair of the input and first hidden layer. The objective then is to maximize the joint likelihood over the visible and the latent variables given the training data which can be achieved with the contrastive divergence algorithm (Hinton et al., 2006) or stochastic maximum likelihood (Tieleman, 2008) for example. Thus, generative DBNs are essentially stacked RBMs. DBNs are studied by Bengio, Lamblin, Popovici and Larochelle (2007) and Salakhutdinov and Hinton (2009) extend the model to *Deep Boltzmann Machines (DBMs)* using a learning algorithm based on variational inference. Lee et al. (2009) present the extension to deep convolutional architectures called *Convolutional Deep Belief Network (CDBN)*. Another class of unsupervised models are based on reconstruction error objectives, that is the output of the network is of the same dimension as and should be as close to its original input. Such models are called *Deep Autoencoders* (Hinton and Salakhutdinov, 2006) and in order to learn the factors of variation underlying the data, autoencoders compress or regularize the flow of information in the hidden layers of the network, which can be achieved in various ways. We will investigate deep autoencoders in section 4.3 as they are the most frequently applied model for deep anomaly detection. A novel unsupervised objective is given in *Generative Adversarial Networks (GANs)* (Goodfellow et al., 2014) which is a system of two networks playing a minimax game. We will introduce GANs in section 4.3 in more detail as well and show how they can be used for the task of anomaly detection.

The power of distributed representations Complex data samples that exhibit variations in many dimensions are the result from the interaction of many distributions, the *factors of variation* underlying the data. Imagine the scenario of taking a picture of a cat. The overall distribution of cat images then is composed from a great deal of underlying factors such as the angle or viewpoint from which the photo is taken, the intensity and location of different light sources, different background environments, and the position and size of the cat in the image. In addition, attributes of the cat itself vary by breed, for example the color and patterns in the fur, but also physical expressions in the moment can be diverse. Thus, to separate signal from noise with respect to the class of cats, a useful feature representation would express the different underlying factors of variation.

Bengio et al. (2013) describe good representations to be *distributed* (i.e. compositional and sparse), *expressive*, and *invariant* to various transformations of the data such that the factors of variation underlying the data distribution are disentangled. Deep networks learning hierarchical features, where the higher-level features are composed from the lower-level ones, enable such distributed representations. The compositional structure of deep networks allows features in the hierarchy to be reused for representing multiple inputs (e.g. simple edges or basic shapes like triangles etc. from which more complex objects are composed). The crucial property of deep networks now is, that through the combination of hierarchical features the representational configurations possible grow exponentially with the depth of the network! That is, deep networks enable an exponential gain in representational power leading to a rich "expressiveness" which means that multi-layered networks compactly represent very rich feature representation spaces (Bengio and Delalleau, 2011). Mathematically speaking, the composition of many simple non-linear functions allows to approximate highly non-linear and highly-varying functions. In addition, multiple levels of abstraction in feature representation induced by multiple layers in the model imply different invariances in representation (such as translational or rotational) depending on the network architecture, which we will see for convolutional neural networks in section 4.2. This key property of distributed features in deep networks enabling an exponential gain in representations also tackles the curse of dimensionality. Whereas classical non-parametric methods, such as nearest-neighbor-based or clustering-based techniques, would require an exponentially increasing number of data samples in increasing dimensionality to achieve statistically sound estimations (if applied in the original input space), distributed representations have the potential to generalize well to unseen regions in input space if the overall distribution can be factorized and the hierarchical features in the network learned to disentangle those factors of variation. (see also Montufar et al., 2014, Montúfar and Morton, 2015, for a detailed analysis) In simple terms, not all possible configurations have to be seen if the underlying sources of variation are known. Finally it must be emphasized that despite the potential of deep networks to have an exponential gain in representational power, the success of the concept of distributed representations relies on the assumption that the data-generating distribution can be factorized into different sources of variation, that is the underlying distribution

has a compositional structure.

To conclude this overview, we will below review the basics and current status in optimization and generalization in deep learning, which are both still active areas of research.

Optimization

The difficulty in training deep models is given by the fact, that the objective functions employed are generally non-convex in the parameters of the deep network making the optimization problem to solve non-convex one. Furthermore, the theoretical objective usually is defined by finding the network which minimizes the expected loss (i.e. risk) of some chosen loss function and by the principle of *empirical risk minimization (ERM)* this theoretical objective is approximated with the empirical risk (i.e. the average loss over training examples). Therefore, since the class of networks chosen in a deep model usually is very rich (i.e. deep networks have many parameters), the problem of overfitting the model on the training data must be paid particular attention to.

Backpropagation and gradient descent The standard algorithm to train neural networks is the *backpropagation* algorithm (Rumelhart et al., 1988, LeCun et al., 2012) which can be applied to every network that employs (sub-)differentiable activation functions in its layers and has a (sub-)differentiable learning objective. In essence, the backpropagation algorithm applies the chain rule to compute the gradient of the learning objective with respect to every parameter in the network. The key observation of backprop is that the gradient of the objective with respect to the input of some layer can be calculated by going backwards from the gradient with respect to to the output of that layer. Hence, starting from the final output of the network, the gradients of the parameters can be computed iteratively going backwards through the network. Thus, the backpropagation algorithm enables to use gradient descent for the optimization of the weights of a network.

Let $f(\cdot\,;\mathcal{W}) : \mathcal{X} \to \mathbb{R}^p$ be a neural network with $L \in \mathbb{N}$ layers and set of parameters $\mathcal{W} = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L\}$. Furthermore, let $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be a training data set and $l : \mathbb{R}^p \to [0, \infty)$ be a (sub-)differentiable loss function (which in supervised learning also has an argument for the label as in this case the loss of a prediction usually is measured by some distance to the ground truth). Then, the *full-batch gradient descent* update rule for the k-th network weight w_{jk}^{ℓ} of neuron j in some layer ℓ with the empirical loss objective

$$J(\mathcal{W}; \mathcal{D}_n) = \frac{1}{n} \sum_{i=1}^{n} l(f(\boldsymbol{x}_i; \mathcal{W})) \tag{4.8}$$

is given by

$$w_{jk}^{\ell} \;\leftarrow\; w_{jk}^{\ell} - \eta \, \frac{\partial J(\mathcal{W}; \mathcal{D}_n)}{\partial w_{jk}^{\ell}} = w_{jk}^{\ell} - \eta \, \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_{jk}^{\ell}} \, l(f(\boldsymbol{x}_i; \mathcal{W})), \tag{4.9}$$

where $\eta > 0$ is the learning rate and the gradient for every $\boldsymbol{x}_i \in \mathcal{D}_n$,

$$\frac{\partial}{\partial w_{jk}^{\ell}} l(f(\boldsymbol{x}_i; \mathcal{W})), \tag{4.10}$$

is computed via backpropagation, that is at each update iteration first a forward pass through the network is performed to calculate the error of the respective input $\boldsymbol{x}_i$ which then is propagated back through the network to receive the gradient for every weight. In practice, since backpropagation is a special case of reverse mode *automatic differentiation*, for every network with (sub-)differentiable non-linearities and (sub-)differentiable objective, the numerical (sub-)gradients can be computed automatically (Rall, 1981). Automatic differentiation is implemented in most of the popular deep learning software libraries such as `TensorFlow` (Abadi et al., 2015), `Theano` (Theano Development Team, 2016), or `Torch` (Collobert, Kavukcuoglu and Farabet, 2011).

As indicated above, updating the network weights on the whole data set at every iteration though is not advisable due to the risk of overfitting and moreover for large-scale applications and huge networks, updates on the overall training data are often simply not computationally feasible. *Stochastic gradient descent (SGD)* (Bottou, 1998) is on the opposite end of the spectrum, where each update iteration is only performed on one randomly drawn data sample $\tilde{\boldsymbol{x}} \in \mathcal{D}_n$, that is

$$w_{jk}^{\ell} \leftarrow w_{jk}^{\ell} - \eta \frac{\partial}{\partial w_{jk}^{\ell}} l(f(\tilde{\boldsymbol{x}}; \mathcal{W})), \quad \eta > 0. \tag{4.11}$$

The randomness introduced by SGD improves generalization and avoids getting stuck in poor local solutions, though pure SGD takes long time to converge as the gradients strongly oscillate. For this reason, *mini-batch gradient descent* is a reasonable trade-off between the two ends of the spectrum, where every update is performed on a mini-batch $\mathcal{B} \subset \mathcal{D}_n$ of size $b = |\mathcal{B}|$, that is

$$w_{jk}^{\ell} \leftarrow w_{jk}^{\ell} - \eta \frac{1}{b} \sum_{\boldsymbol{x}_i \in \mathcal{B}} \frac{\partial}{\partial w_{jk}^{\ell}} l(f(\boldsymbol{x}_i; \mathcal{W})), \quad \eta > 0, \tag{4.12}$$

which has faster and more stable convergence than pure SGD but still provides better generalization performance through stochasticity (Bousquet and Bottou, 2008). Moreover, in very large data sets many examples are similar and therefore the benefit of including more samples is decreasing. Therefore, mini-batch gradient descent is most of the time the optimization algorithm of choice for large-scale problems in deep learning. In practice, when referring to SGD commonly mini-batch gradient descent is meant and training is performed in *epochs*, where in each epoch the training data is shuffled randomly and partitioned into mini-batches of equal size, that is after every epoch each training example has been used in one of the mini-batch updates. Furthermore, by the additive nature of the empirical error (to which usually a regularization term is added) the computation of

the gradients of each individual error with respect to each parameter can be parallelized and Graphics Processing Units (GPUs) enable an efficient parallel computation of the tensor operations involved in every mini-batch update (Raina et al., 2009, Srivastava et al., 2014) which facilitates a great boost in speed for computation in practice.

Challenges As the learning objective is generally non-convex in the network parameters, the challenge in deep learning optimization is given in avoiding bad local solutions in parameter space. Since the higher the dimensionality of the parameter space (i.e. the greater the number of parameters in the network which is generally high in deep learning), the more likely saddle points, instead of local minima, become (the greater the number of directions, the less likely it becomes that all directions go either up or down), the challenge of avoiding bad local solutions mostly translates into avoiding getting stuck in saddle points of high altitude although plateaus in the loss landscape, if step sizes are too small, can also be an issue. If a local solution has been found, however, recent analysis indicates that the locality of the solution often is not as big of an issue as previously thought since there are many redundant solution configurations in the parameter space (Dauphin et al., 2014, Choromanska et al., 2015).

If SGD is used for optimization the question on how to initialize the parameters is posed. A poor initialization ("on the wrong side of a mountain") could cause gradient descent to get stuck in a bad region in parameter space from which it cannot easily escape. Greedy layer-wise pre-training (Hinton et al., 2006, Bengio, Lamblin, Popovici and Larochelle, 2007, Erhan et al., 2010), as mentioned before, was considered a necessary step before starting to jointly train the weights of a network via SGD, but Glorot et al. (2011) showed that when ReLU activations are employed, the effect of pre-training is negligible. Nowadays, network parameters are usually initialized randomly from the Uniform or the Gaussian distribution by *Glorot* (a.k.a. *Xavier*) *initialization* (Glorot and Bengio, 2010), which adapts the range of the weights in each neuron to the number of its inputs such that units are not saturated in the beginning, or *He initialization* (He et al., 2015) which particularly takes the rectifier non-linearity into account. However, there are indications that pre-training still helps for good generalization if sample-sizes are small (Bengio et al., 2013) and generally the understanding of how initialization affects generalization performance is still primitive (section 8.4 in Goodfellow et al., 2016).

Besides initialization, a number of hyperparameters have to be selected in choosing and training a deep model such as the size (number of layers and units) and the architecture of the network and its non-linearities, regularization hyperparameters, the size of the mini-batch, and the SGD learning rate, which is considered the most important hyperparameter in gradient-based optimization (Bengio, 2012). This is why several adaptive optimization methods have been proposed as we will see below. Tuning the hyperparameters is generally not an easy task and can be computationally costly. Random search should be preferred to grid search when evaluating different configurations (Bergstra and Bengio, 2012). Practical recommendations for hyperparameter selection are given by Bengio

(2012) and Montavon et al. (2012).

SGD extensions The difficulty in choosing a learning rate in gradient descent is that if it is set too high, the gradient steps taken can strongly oscillate and the algorithm could even diverge. If it is set too low on the other hand, progress on the loss surface is only small.

A first and generally recommended (Sutskever et al., 2013) extension to SGD to tackle strong oscillations and slow progress is to add *momentum* (Polyak, 1964). Momentum adds inertia to the gradient descent update rule such that the current update step is a weighted linear combination of all previous gradient steps where the importance of each previous step is decreasing in the number of iterations. A physical metaphor for this extension would be a ball rolling down the landscape which builds up momentum due to inertia. The momentum update rule for the k-th network weight w_{jk}^{ℓ} of neuron j in some layer ℓ is given by

$$\begin{aligned} v_{jk}^{\ell} &\leftarrow \alpha\, v_{jk}^{\ell} - \eta\, \frac{\partial J(\mathcal{W}; \mathcal{D}_n)}{\partial w_{jk}^{\ell}} \\ w_{jk}^{\ell} &\leftarrow w_{jk}^{\ell} + v_{jk}^{\ell} = w_{jk}^{\ell} - \eta\, \frac{\partial J(\mathcal{W}; \mathcal{D}_n)}{\partial w_{jk}^{\ell}} + \alpha\, v_{jk}^{\ell} \end{aligned} \tag{4.13}$$

with v_{jk}^{ℓ} being the corresponding *velocity* of the update and where $\alpha \in (0, 1)$ is the momentum parameter determining the importance of the previous steps in the current update (usually $\alpha \in \{0.9, 0.99\}$) and $\eta > 0$ is again the learning rate. Momentum softens oscillations in regions of high curvature as gradients of opposite directions even out in the linear combination. Moreover, momentum builds up speed which helps avoiding getting stuck in bad local solutions and helps traversing plateaus faster. An adjustment to momentum is given by Nesterov (1983) which proposed to first take the step of the accumulated gradients before determining the new direction added to the accumulation, which can lead to even faster convergence.

A problem of momentum is that due to the speed build up final convergence can take longer which is the motivation behind a *learning rate schedule* which decays the learning rate over time to decrease step sizes and fine-tune convergence. Adaptive learning rate algorithms go one step further by varying the learning rate across layers and parameters as well as over time. *AdaGrad (Adaptive Gradient Algorithm)* (Duchi et al., 2011) individually adapts learning rates from a base rate specified by taking into account past updates to increase the learning rates for parameters that are activated sparsely by the data in updating while decreasing the learning rates for parameters that are activated more often. The idea behind this strategy is the assumption that sparse features can be informative but due to their scarcity equal learning rates would lose that information. However, AdaGrad has been shown to lead to premature convergence due to the decreases in the learning rates being too strong (cf. section 8.5 in Goodfellow

et al., 2016). *RMSProp (Root Mean Square Propagation)* (Tieleman and Hinton, 2012) tries too alleviate this problem by dividing the individual learning rate of a weight by the moving average of the root *mean* squared (RMS) gradients instead of the root of the *accumulated* squared gradients which is the divisor in AdaGrad. *ADADELTA* (Zeiler, 2012) provides another strategy for preventing the premature convergence problem of AdaGrad by restricting the size of accumulated past gradients thus also making the decay of the learning rate slower. Another approach is the *Adam* (adaptive moments) optimizer (Kingma and Ba, 2014) which effectively is extending RMSProp with momentum.

There is no one go-to optimizer and often the methods introduced above give similar (good or bad) solutions with similar convergence behavior. A comparison of optimizers is given by (Schaul et al., 2013). For this reason, optimization in deep learning is still an active and important area of research.

Generalization

The principal goal of every statistical estimation or machine learning procedure is to have a good generalization performance, that is a model estimated from training data should perform well on new, unseen examples. This is why model performance always is measured on a test set which remains untouched while training the model. The generalization error depends on the complexity of the model chosen, that is the capacity of the class of functions selected for estimation to approximate an unknown general function. The greater the capacity, the easier it is for the model to have a good fit on the training data, but also the greater the tendency towards overfitting is. This is why model capacity is limited by choosing a subset from the class of all functions in order to achieve a low generalization error. If the model capacity is too limited on the other hand (e.g. choosing a linear model when the data has non-linear patterns), it is possibly underfitting. This balance in model complexity which has to be found is the well-known *bias-variance tradeoff*.

Generalization in deep learning is not yet well understood Although deep neural networks are massive in their number of parameters, that is they have great model complexity, they seem to generalize remarkably well. Experiments by Zhang et al. (2016) suggest that the fundamental reasons for good generalization of deep neural networks are not yet well-understood and a theoretical framework still has to be formulated. Classical complexity measures from statistical learning theory, such as VC dimension (Vapnik, 1998) or Rademacher complexity (Bartlett and Mendelson, 2002), that are controlled to achieve good generalization, struggle to explain the low generalization errors attained by deep neural networks. In their experiments, Zhang et al. (2016) show that state-of-the-art deep networks are also capable of fitting completely random labels, that is they are powerful enough to shatter the training data, but nevertheless they show small generalization errors when trained on the ground truth labels — even without any explicit

form of regularization. Hence, classical measures of complexity do not seem to apply for explaining generalization in deep learning.

Generalization due to the concept of distributed representations To some degree, good generalization of deep hierarchical models, can be explained by their characteristic property of distributed representations (Bengio et al., 2013) as was mentioned before. Using a deep architecture with multiple layers means that input representations are given by different activation paths in the network, that is, the final representation is a composition from representations of different levels of abstraction. This combinatorial nature of representation allows to reuse the learned features for new activation paths not realized in training, that are potentially useful for representing previously unseen examples effectively, given the unseen data has a similar composition of underlying factors of variation. For example, low-level features such as strokes and small motifs, or even object parts, learned from a set of images can be reused to represent previously unseen images. In the depth of the network, such possible configurations increase exponentially. (e.g. k binary units allow for 2^k input region representations) This potentially exponential increase in "expressiveness" in part explains the good generalization performance of deep networks (see also Montufar et al., 2014, Montúfar and Morton, 2015).

Regularization techniques Despite not being the fundamental reason for the good generalization of deep networks as the above indicates, regularization techniques still help to improve generalization performance. We will briefly describe the different options available for regularization in deep learning.

The simplest way to achieve better generalization performance of course is to train the model on more data. Although this is not a form of regularization, it explains good generalization performance in many large-scale applications, for which deep neural networks are the go-to models since they scale well due to parallelization and mini-batch training. A related form of "regularization" is *data augmentation* (e.g. used by Krizhevsky et al., 2012, Cui et al., 2015), where the size of the training data is increased artificially by applying transformations (e.g. translations and rotations) or by adding noise to original samples. Data augmentation can drastically improve model performance, which is why results that use data augmentation are usually excluded from benchmarks or are listed separately.

Next, the most basic genuine regularization technique is penalizing the norms of the network parameters. This is also called *weight decay* and typical choices are L^2-(*Ridge* or *Tikhonov regularization*) or L^1-penalties (*Lasso regularization*), where the latter favors parameter sparsity. Parameter norm penalizations prevent the network weights from exploding and implicitly restrict the parameter solution space. This prevents the model from over-adapting to the training data and therefore decreases the tendency for overfitting.

Another cheap and commonly used implicit form of regularization for neural networks

is *early stopping*, which simply means to stop training, when the error on a held-out validation set has been not decreasing or increasing successively for some iterations. After stopping, the weights with lowest validation error are chosen. Assuming that the gradient of the objective is bounded, early stopping also imposes a restriction on the volume of the solution space by limiting the number of gradient steps with some learning rate η when updating the parameters (cf. section 7.8 in Goodfellow et al., 2016).

More computationally expensive, but very effective meta-techniques to decrease the generalization error are *ensemble methods*. Ensemble methods can essentially be applied to any machine learning models involving hyperparameters or local optima. The idea of ensemble methods is to train several models on different configurations and finally to combine the results (usually by averaging or voting). This drastically decreases the generalization error by reducing the variance portion of the error from the basic bias-variance decomposition of the error (Montavon et al., 2012). For neural networks, the possible configurations to average over are manifold. *Bagging* (bootstrap aggregating) (Breiman, 1996) can be applied, where the difference in models is that they are trained on different training sets, which are created by drawing from the original data set with replacement. Other differences that can be utilized are varying the hyperparameters and differences in weight initialization. As meta-techniques that are very powerful, ensemble methods are also usually excluded from benchmarks and only single models are compared.

A more recent regularization technique, that quickly got adapted broadly due to its effectiveness, is *dropout* (Srivastava et al., 2014). The idea behind dropout is to approximate an ensemble average over exponentially many architectures by randomly dropping units and their connections from the network in each training iteration. The probability p for dropping a unit is set a priori and per layer (usually $p = 0.5$). Essentially, by dropping units randomly per iteration, many subnetworks of the original architecture are trained, which explains the approximation of an ensemble. In order for the subnetworks to have similar capacity as the original network, the overall size of the original network usually is increased. (e.g. twice the number of parameters in layers with $p = 0.5$) This also means that the computational cost of training a network with dropout is generally greater, also because more training epochs are required until convergence.

Finally, a regularization method used in one of the most influential deep architectures, convolutional neural networks, is *parameter sharing*. Parameter sharing forces network parameters to be equal across multiple sets of features of an input, which effectively reduces the number of weights in a network. We will introduce convolutional neural networks in more detail next.

4.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were introduced with the spatial structure of image data in mind (LeCun et al., 1990, 1998) and essentially revolutionized the computer vision domain with breakthroughs that almost halved existing error rates of the best

competing models (Krizhevsky et al., 2012). CNNs account for the spatial structure of images and local correlations of nearby pixels by introducing *parameter sharing* across local patches in an image allowing translational invariant local features to be learned. However, the convolutional architecture is not limited to vision applications. They are, for example, also applied in natural language processing (Collobert and Weston, 2008). Below we will briefly explain how the convolutional architecture works. This quick review is mainly based on section 9 in Goodfellow et al. (2016).

Parameter sharing Considered by raw pixels, images are of very high dimensionality as the number of features are given by width $\times$ height $\times$ number of channels (e.g. 3 channels for RGB color images). Using fully-connected dense layers, the number of parameters in a network for image data therefore would be huge and impractical to train. Moreover, nearby pixels in images are often highly correlated and local patches can be invariant to their global location, that is similar or equal edges, motifs, and color blobs could appear in different locations within and across images. Thus, fully-connected layers would waste many parameters. This is why *parameter sharing* is introduced which accounts for *translation* or *shift invariance* as well as *local correlations* in images and greatly reduces the number of parameters in a network.

Parameters in convolutional layers have a spatial arrangement as images have, that is the parameters are arranged in width $\times$ height $\times$ depth. They are given by a set of *filters*, where the overall number of weights in a layer is given by the number of filters multiplied by their spatial extend (i.e. width $\times$ height $\times$ depth). For this reason, the parameters of a convolutional layer are also called *filter bank*. For RGB-color images, for example, every filter has a depth of 3, that is the depth of each filter is equal to the number of channels given in the input. The width and height of the filters are hyperparameters to select and determines the size of the *receptive field* of each filter which means the size of the local patches in an image each filter looks at. For instance specifying filters of size 5×5 implies the filter to be applied to local patches of the input image of size 5×5. By sharing the parameters of each filter across all patches in an input image that have the size of the filter, local correlations and the translational invariance of nearby pixels is accounted for. Visually, each filter is slided over the input image and by computing the dot product between the filter and every local patch, the patch is scanned for the feature the filter has learned to detect (e.g. an edge with specific orientation). This results in a map of scores which gives for every patch a score with respect to the local feature learned by the filter. The sliding operation with fixed filter weights producing such a map mathematically is a discrete convolution which is where the layer and network architecture got its name from.

Stacked modules The resulting map of scores is then fed element-wise into an activation function, usually the ReLU activation, which results in an activation map showing the response of each local patch of an input image. Depending on the architecture, the

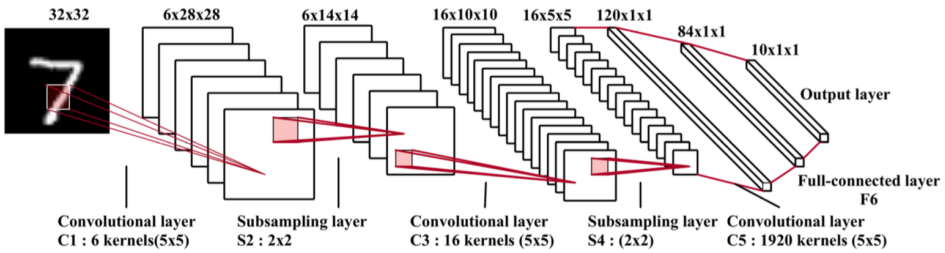


Figure 4.2: LeNet-5 network (Figure taken from Gu et al. (2015))

activation map is followed by another convolution-activation-pair, for example in *AlexNet*-type architectures (Krizhevsky et al., 2012), or immediately by a *pooling layer* which spatially downsamples the activation map. Usually 2×2 pooling is performed which maps four units spatially arranged in a 2×2 patch in the activation map to one output thus reducing each dimension of the activation map by a factor of 2. Pooling operations are for example the average but max-pooling is commonly used where only the highest activation of each 2×2 patch is passed to the next layer, i.e. 75% of activations are discarded, which favors sparse activations and induces invariance with respect to small shifts and distortions (Scherer et al., 2010). Pooling increases the level of abstraction as merged activation maps of the filters from the previous layer are the input to the next layer in the hierarchy. Stacking convolution-activation-pooling modules constitutes the convolutional network architecture in which the final layer often are fully-connected dense layers again which allow the abstract features detected by the convolutional stacks to be combined. Figure 4.2 shows the classic LeNet-5 architecture where convolutional stacks are followed by fully-connected layers in the final hidden layers.

Optimization Convolutional neural networks can be trained with SGD and its variants using the backpropagation algorithm (LeCun et al., 1990) as the convolution is a linear operation and the commonly used activation and pooling functions are (sub-)differentiable. However, for the case that ReLU activations and max-pooling is employed in the network, the resulting composition is in fact a piecewise-linear function which can be exploited. For instance,Berrada et al. (2016) introduce a layer-wise optimization algorithm for Piecewise-Linear Convolutional Neural Networks (PL-CNN) which have a SVM classifier as a final layer, formulating the estimation of the parameters in each layer as a difference-of-convex (DC) program that can be optimized using the concave-convex procedure (CCCP) (Yuille and Rangarajan, 2002) which allows the derivation of an optimal learning rate.[1]

[1]The work of Berrada et al. (2016) served as an inspiration to investigate a one-class SVM loss for deep anomaly detection and sparked the topic for this thesis.

4.3 Deep Learning in Anomaly Detection

Deep methods for learning representations are yet not much explored for the task of unsupervised anomaly detection or one-class classification. Approaches can be separated into "hybrid" models, that use deep learning methods in a first step of learning representations independently of the anomaly detection technique that is subsequently applied to the representations learned, and "fully deep" methods, which learn the representations directly on an objective that is suited for the task of anomaly detection. The first approach is more similar to the classical pipeline of (manual) feature engineering and extraction and applying a machine learning model on top. The latter is in line with the fundamental idea of deep learning: to learn feature representations and optimize an objective jointly.

Deep autoencoders

The predominantly used (almost exclusively it seems) models in the literature of unsupervised anomaly detection using deep learning techniques are *autoencoders*. They are utilized in both approaches, either to only learn and extract features in a hybrid system, or the objective of the autoencoder itself is directly employed for the purpose of anomaly detection.

Autoencoders are neural networks which aim to learn two transformations of the data: first, a transformation that encodes the information of an input into a latent representation, and second, a transformation that reconstructs the input again from this hidden representation. Hence, for an input $\boldsymbol{x} \in \mathcal{X} \subset \mathbb{R}^d$ the autoencoder network learns a concatenation $(\psi \circ \phi)(\cdot\,; \mathcal{W}) : \mathcal{X} \to \mathcal{X}$, where $\phi(\cdot\,; \mathcal{W}_1) : \mathcal{X} \to \mathcal{F}$ is the *encoder*, mapping the input to some latent space $\mathcal{F}$ (e.g. $\mathcal{F} \subset \mathbb{R}^p$ for a dense hidden layer with p units), and $\psi(\cdot\,; \mathcal{W}_2) : \mathcal{F} \to \mathcal{X}$ is the *decoder*, trying to reconstruct the original input from the latent representation, where $\mathcal{W}_1$ is the set of the network parameters of the encoder and $\mathcal{W}_2$ the set of the parameters of the decoder part of the network with $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$ denoting the set of all network parameters. The objective of an autoencoder is defined by some reconstruction loss, for example the mean squared reconstruction error:

$$\min_{\mathcal{W}} \frac{1}{n} \sum_i \|\boldsymbol{x}_i - (\psi \circ \phi)(\boldsymbol{x}_i; \mathcal{W})\|^2. \tag{4.14}$$

for some training set $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$. Usually, autoencoder architectures are symmetric in the sense that the same hidden layers (in size and activation type) that lead to the encoding are also employed in reverse order to get back to the reconstruction. Therefore, autoencoders generally have an odd number of layers with the encoding layer in the middle.

The *code* $\phi(\boldsymbol{x}; \mathcal{W}_1)$ can be regarded as a compressed representation of the original input $\boldsymbol{x} \in \mathcal{X}$. The idea of autoencoders is to extract the common factors of variation from the data by forcing the information given by the data to be compressed in the encoding

process. Otherwise, the trivial solution for the autoencoder network $\psi \circ \phi$ would be to learn the identity function. Controlling this compression, that is the proper regularization of the autoencoder to prevent overfitting due to rich architectures or small sample sizes, is the key challenge in designing and training autoencoders. Several different techniques have been proposed in the literature to address this challenge which will be given and explained in the following.

Using autoencoders for anomaly detection As training the autoencoder on some target class causes the network to learn parameters extracting common factors of variation from that class, the reconstruction error of the trained autoencoder for some test point $\boldsymbol{x} \in \mathcal{X}$ can directly be used as an anomaly score, for example in case of the quadratic reconstruction error:

$$s(\boldsymbol{x}) = \|\boldsymbol{x} - (\psi \circ \phi)(\boldsymbol{x}; \mathcal{W}^*)\|^2, \tag{4.15}$$

where $\mathcal{W}^*$ are the network weights of the trained autoencoder. The higher the reconstruction error, the less typical factors of variation the tested example exhibits with respect to the target class, that is, the trained autoencoder should give poorer reconstructions for atypical examples and anomalies. Alternatively, the code $\phi(\boldsymbol{x}; \mathcal{W}_1)$ representation of an input $\boldsymbol{x} \in \mathcal{X}$ (or also other intermediate hidden layers) of the autoencoder can be used as an input for another anomaly detection method in a hybrid system.

"Bottleneck" autoencoders Hawkins et al. (2002) where amongst the first to use a neural network based model with reconstruction error objective for the task of outlier detection. To detect anomalies, they use an autoencoder with 3 hidden layers what they then called a "Replicator Neural Network". To enforce a compression of information, they use intermediate layers with a lower number of units than the input dimensionality. This "bottleneck"-approach in fact makes such an autoencoder a method of hierarchical, non-linear dimensionality reduction, if non-linear activations are used. A more recent investigation using the bottleneck representation of an autoencoder as the input for a OC-SVM to tackle high-dimensional and large-scale anomaly detection was conducted by Erfani et al. (2016). The performance results of their hybrid approach look promising and moreover the results between applying a linear or Gaussian kernel in the OC-SVM used on top of the learned representations do not differ significantly in the experiments carried out. Thus, more efficient linear programming algorithms can be applied to solve the OC-SVM which enable better scaling. However, since the reported anomaly detection performance of the deep autoencoder itself is similar, the benefit of additionally applying a OC-SVM on top could be questioned.

Denoising autoencoders Another approach to regularize autoencoders, that is to extract the most important factors of variation, is by, instead of using the original inputs in training, feeding the algorithm corrupted versions with added noise. The

reconstruction then again is compared to the undistorted original input. Such autoencoders are called *denoising autoencoders (DAE)* (Vincent et al., 2008, 2010) and essentially apply the strategy of data augmentation for regularization. Xu et al. (2015) use denoising autoencoders in their system for anomalous event detection in video scenes. In their system, they do not use the reconstruction error as an anomaly score, but use the DAE latent representations learned from appearance and motion features to train multiple OC-SVMs, thus utilizing the DAE also in a hybrid model.

Sparse autoencoders *Sparse autoencoders* are another variant of autoencoders that are, in contrast to bottleneck-architectures, usually overcomplete in the sense that the number of units in the hidden layers exceeds the input dimensionality. The idea of regularization then is to introduce penalties that encourage sparsity in the hidden unit activations, which is motivated from the concept of *sparse coding* or *dictionary learning* (Lee et al., 2007, 2008) where the idea is to learn a set of basic elements (the dictionary) from which the input can be reconstructed in a sparse linear combination. Thus, sparsity regularization causes the network to learn hierarchical features that allow reconstruction by sparse combination and therefore encourages the network to learn features that can be reused amongst inputs and capture again the different factors of variation of the underlying class. There are different ways of penalization to induce sparsity, such as encouraging the mean activation rate to be close to some user-specified fraction (Ng, 2011), simple L^1 penalization on the activations, or keeping only the k largest activations in the hidden layers (k-*sparse autoencoders*, Makhzani and Frey, 2013). Sparse autoencoders for anomaly detection are for example investigated by Andrews et al. (2016) who also use the features learned by the autoencoder as an input to a OC-SVM but their approach is not unsupervised as anomaly labels are used in cross-validation for selecting the hyperparameters of the OC-SVM. An application in video anomaly detection using sparse autoencoders is given by Sabokrou et al. (2016).

Variational autoencoders A probabilistic Bayesian autoencoder model was introduced by Kingma and Welling (2013) in which assumptions on the distributions of the latent variables are made. Estimation of the intractable posterior in the model is done by variational inference approximation which is why such an autoencoder is called *variational autoencoder (VAE)*. The resulting training algorithm is called *Stochastic Gradient Variational Bayes*. Utilizing VAEs for anomaly detection is considered by An and Cho (2015) which propose anomaly scoring based on the VAE reconstruction probability.

Convolutional autoencoders In image or video anomaly detection applications, *convolutional autoencoders (CAEs)* (Masci et al., 2011, Makhzani and Frey, 2015) are an appropriate model of choice incorporating the benefits of convolutional architectures for computer vision tasks as outlined in the previous section. Ribeiro et al. (2017) recently published a study on deep CAEs for anomaly detection in videos in which the

reconstruction error in each frame directly is used as an anomaly score. An application in medical imaging using deep CAEs for feature extraction is conducted by Seeböck et al. (2016). To detect markers relevant for treatment guidance they also employ a (linear) OC-SVM on top of the features learned by a CAE trained only on healthy data. Another computer vision novelty detection application is given by Richter and Roy (2017), who use a deep autoencoder in addition to a supervised deep network trained for autonomous visual navigation. Their idea is to use the deep autoencoder to detect novel inputs compared to the body of training data used in learning the supervised network. If a novel situation is detected by the CAE, the vehicle reverts back to a safe prior behavior as the trust in the supervised network is reduced.

Autoencoder ensembles Chen et al. (2017) investigated the use of autoencoder ensembles for unsupervised anomaly detection. To tackle overfitting and sensitivity to noise, Chen et al. (2017) vary the connectivity in the autoencoder architecture randomly (random edge sampling) to train an ensemble of independent autoencoders of which the outputs are combined. They call such an ensemble model "RandNet" (Randomized Neural Network for Outlier Detection).

Generative Adversarial Networks

Besides deep autoencoders, another unsupervised deep model has been employed for the task of anomaly detection only recently. A *Generative Adversarial Network (GAN)* (Goodfellow et al., 2014) is a system consisting of two models having adversarial objectives that are trained simultaneously. The first of the two models is the generator G, the other the discriminator D. The generator is a mapping $G : \mathcal{Z} \to \mathcal{X}, \boldsymbol{z} \mapsto G(\boldsymbol{z})$ from some latent space $\mathcal{Z}$ to the space $\mathcal{X}$ where the original data lives. Samples in latent space $\boldsymbol{z} \in \mathcal{Z}$ are generated from noise (usually uniform or Gaussian) and the objective is to learn the generator G such that it approximates the underlying distribution $P_{\boldsymbol{x}}$ from which the data $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ was generated, that is to learn G such that $P_{G(\boldsymbol{z})} \approx P_{\boldsymbol{x}}$. The discriminator $D : \mathcal{X} \to [0, 1]$ then tries to distinguish between samples $G(\boldsymbol{z})$ from the generator and original inputs $\boldsymbol{x}_i \in \mathcal{D}_n$ assigning each of its inputs a value between 0 and 1 which can be interpreted as the probability the discriminator assumes its input to be real or fake. The objective is given by a two-player minimax game with value function $V(G, D)$ defined by

$$\min_G \max_D V(D, G) = \mathbb{E}_{\boldsymbol{x} \sim P_{\boldsymbol{x}}}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z} \sim P_{\boldsymbol{z}}}[\log(1 - D(G(\boldsymbol{z})))], \tag{4.16}$$

that is, the objective of the discriminator D is to maximize the probability of detecting real and fake examples whereas the generator G simultaneously tries to fool the discriminator by minimizing $V(G) = \log(1 - D(G(\boldsymbol{z})))$. If G and D are neural networks, they can be trained iteratively via backpropagation. In case of deep convolutional neural networks being used, we speak of *Deep Convolutional Generative Adversarial Networks (DCGANs)*

(Radford et al., 2015). For image data, the trained generator G from optimizing the value function of the game is capable of generating realistic-looking fake images which is the application responsible for GANs attracting a lot of attention even outside the research community.

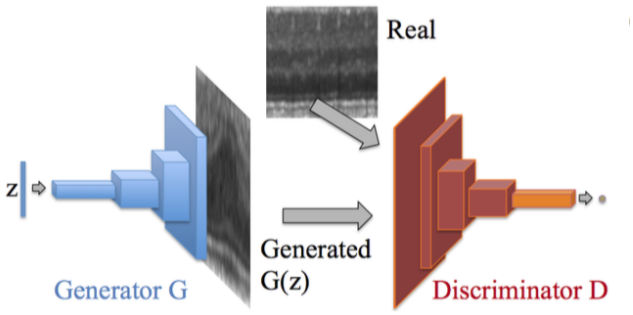


Figure 4.3: Illustration of a DCGAN (Figure taken from Schlegl et al. (2017)).

Using Generative Adversarial Networks for anomaly detection Schlegl et al. (2017) propose an anomaly detection method called *AnoGAN* based on deep convolutional generative adversarial networks thus introducing another deep approach to large-scale and high-dimensional unsupervised anomaly detection apart from deep autoencoders. After training the DCGAN on data from the target class, the idea of Schlegl et al. (2017) is to find for a test example $\boldsymbol{x} \in \mathcal{X}$ the vector $\boldsymbol{z} \in \mathcal{Z}$ in latent space generating a sample $G(\boldsymbol{z})$ that is closest to $\boldsymbol{x}$, that is for each $\boldsymbol{x} \in \mathcal{X}$ to find $\boldsymbol{z} \in \mathcal{Z}$ such that the approximation $G(\boldsymbol{z}) \approx \boldsymbol{x}$ is the best possible given the trained generator G. In that sense they try to find the inverse of the trained generator mapping from input space $\mathcal{X}$ back into latent space $\mathcal{Z}$. To find such a value $\boldsymbol{z}$ resulting in a good approximation, Schlegl et al. (2017) perform gradient descent in latent space with the network weights of generator G held fixed. As the generator has learned the variability of the target class, the reasoning is that approximations $G(\boldsymbol{z})$ for test examples $\boldsymbol{x}$ of the target class should be better. Hence, the difference between $\boldsymbol{x}$ and the optimized corresponding generation $G(\boldsymbol{z})$ can be used as an anomaly score. Schlegl et al. (2017) train AnoGAN on healthy data in medical imaging in order to capture markers relevant for disease progression and treatment monitoring and observe promising performance of the *AnoGAN*.

4.4 Summary and Discussion

In this section, we have explained the fundamentals of deep learning, reviewed the current status and identified open challenges in the field. The essential idea behind deep learning is to learn representations with multiple layers of abstraction by the concept of distributed or hierarchical features realized through models with multiple processing

layers. The aim of hierarchical models with multiple layers is to disentangle the factors of variation underlying a data distribution and by using deep architectures to have a potentially exponential increase in representational expressiveness. We reviewed the de facto standard way to train neural networks, which is by (mini-batch) SGD and its variants realized by the backpropagation algorithm, but have also seen that new approaches are proposed as in the case of PL-CNNs. State-of-the-art techniques such as the (leaky) rectifier, adaptive learning rate optimizers, or dropout regularization were presented. Important research directions in deep learning are given by further improving the training of deep networks as well as formulating a sound theory of generalization.

Finally, we reviewed how deep learning is employed for the task of unsupervised anomaly detection and have seen that the major approach is based on the reconstruction error objective, which is either used directly to detect anomalies or used as a feature extractor to learn representations which then can be fed into traditional anomaly detectors in hybrid systems. Another recent deep approach to anomaly detection using GANs was examined.

In the following section, a novel "fully deep" one-class classification objective will be proposed which is the major contribution of this work, providing a new direction for deep anomaly detection and the task of high-dimensional and large-scale anomaly detection in general.

5 Deep Support Vector Data Description

> *"An ideal kernel function would map the target data onto a bounded, spherically shaped area in the feature space and outlier objects outside this area."*
>
> (Tax and Duin, 2004)

Deep Support Vector Data Description is based on kernel SVDD but incorporates hierarchical feature transformations that are learned jointly with the objective via deep networks into the model instead of fixed kernel mappings. This constitutes the key idea of Deep SVDD: to move from fixed kernel representations to the learning of hierarchical representations by using deep architectures thus allowing more flexible feature representations. First, the model will be introduced in two variants and afterwards various properties of deep SVDD will be given and discussed.

5.1 The Model

In the following, let $\phi(\cdot\,;\mathcal{W}) : \mathcal{X} \to \mathcal{F} \subset \mathbb{R}^p$ be a network with $L \in \mathbb{N}$ hidden layers and set of weights $\mathcal{W} = \{\boldsymbol{W}^1, \ldots, \boldsymbol{W}^L\}$ with $\boldsymbol{W}^\ell$ being the weights of layer $\ell \in \{1, \ldots, L\}$, that is $\phi(\boldsymbol{x}; \mathcal{W}) \in \mathcal{F}$ is the feature representation of $\boldsymbol{x} \in \mathcal{X}$ given by the network. Furthermore, denote by $\mathcal{W}_0$ the set of all-zero network weights that is $\boldsymbol{W}^\ell = \boldsymbol{0}$ for every $\boldsymbol{W}^\ell \in \mathcal{W}_0$. There will be two variants of the deep SVDD objective introduced below: a soft-boundary and a hard-boundary objective.

Soft-boundary objective

Define the Deep Support Vector Data Description *soft-boundary* problem given the data $\mathcal{D}_n = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ as follows:

Problem 5.1 (Deep SVDD soft-boundary objective)**.**

$$\min_{\mathcal{W}, R} \quad \frac{\lambda}{2} \sum_{\ell=1}^{L} \|\boldsymbol{W}^\ell\|_F^2 + R^2 + \frac{1}{\nu n} \sum_{i=1}^{n} \max\{0, \|\phi(\boldsymbol{x}_i; \mathcal{W}) - \boldsymbol{c}\|_2^2 - R^2\} \tag{5.1}$$

with weight decay hyperparameter $\lambda > 0$, hyperparameter $\nu \in (0, 1]$ indicating the fraction of outliers and some fixed center $\boldsymbol{c} \neq \phi(\boldsymbol{x}; \mathcal{W}_0)$ of the hypersphere in output space. $\|\cdot\|_F$ denotes the Frobenius norm that is the regularization term is the sum of all squared network weights.

We require $\boldsymbol{c} \neq \phi(\boldsymbol{x}; \mathcal{W}_0)$ in order to avoid a trivial solution in which the network representation of every input is given by the same vector in output space causing the hypersphere to collapse into one point of zero volume. Note that $\phi(\boldsymbol{x}; \mathcal{W}_0)$ is constant as every input $\boldsymbol{x} \in \mathcal{X}$ is transformed to zero in the linear transformation of the first layer and the specific value depends on the activation function applied in the final layer of the network (e.g. $\phi(\boldsymbol{x}; \mathcal{W}_0) = \mathbf{0}$ in a network exclusively employing ReLUs). We will explain the trivial solution and strategies to choose center $\boldsymbol{c}$ in the next section, where properties of deep SVDD are discussed.

Comparison to kernel SVDD If we compare the deep SVDD objective (5.1) with the primal of kernel SVDD (3.20) the objective is unchanged in the sense that the goal still is to find the minimum-volume data-enclosing hypersphere, but implicit fixed kernel feature representations are now replaced with feature representations of a deep network that is learned jointly with the objective. The constraints of the kernel primal are incorporated directly into the deep SVDD objective and a weight decay term is added for regularization of the network weights in addition. Again, there is penalization if $\|\phi(\boldsymbol{x}_i; \mathcal{W}) - \boldsymbol{c}\|_2^2 > R^2$, i.e. if the feature representation for some input $\boldsymbol{x}_i \in \mathcal{D}_n$ lies outside the hypersphere. Hence, the network in deep SVDD must be adapted such that most of the representations fall within the hypersphere. Simultaneously, the volume of the hypersphere is minimized which results in an iterative adaptation of the network weights such that the network representations are close to the hypersphere center $\boldsymbol{c}$. Training is carried out until convergence. The objective is generally non-convex in the network parameters so the solution found is a local minimum. This means the network ends up in a parameter configuration from which further minimization of the hypersphere-volume does not seem to be possible using the optimization procedure employed. This is in contrast to the convex optimization problem of the kernel SVDD dual (3.19) that has a global minimum. The role of the ν-parameter in deep SVDD is similar as in kernel SVDD, which we will see below, that is ν again allows to control the fraction of outliers in the model.

Anomaly Detection The anomaly score in deep SVDD for a test point $\boldsymbol{x} \in \mathcal{X}$ is defined analogously as in kernel SVDD by

$$s(\boldsymbol{x}) = \|\phi(\boldsymbol{x}; \mathcal{W}^*) - \boldsymbol{c}\|_2^2 - R^{*2}, \tag{5.2}$$

that is the score is given by the distance from the center of the hypersphere corrected by the radius R^{*2} of the converged model with network parameters $\mathcal{W}^*$. Examples $\boldsymbol{x} \in \mathcal{X}$

with positive scores are again flagged to be anomalous, which are the points having representations outside the hypersphere. As the network adapts to the factors of variation given in the target class, typical examples should have network representations close to the center of the hypersphere as the objective is to map most of the data as close to $\boldsymbol{c}$ as possible which causes the network to learn a transformation that tries to fit most of the data to this one point. Hence, points that are unable to be mapped close to the center $\boldsymbol{c}$ are unlike the rest of the data which is why atypical examples and outliers should have representations further away from the center or outside of the hypersphere.

Hard-boundary objective

If no outliers or anomalies are assumed to be within the training set, a simpler *hard-boundary* objective can be adopted:

Problem 5.2 (Deep SVDD hard-boundary objective)**.**

$$\min_{\mathcal{W}} \quad \frac{\lambda}{2}\sum_{\ell=1}^{L}\|\boldsymbol{W}^{\ell}\|_F^2 + \frac{1}{n}\sum_{i=1}^{n}\|\phi(\boldsymbol{x}_i;\mathcal{W})-\boldsymbol{c}\|_2^2, \tag{5.3}$$

where $\lambda > 0$ is again the weight decay hyperparameter and $\boldsymbol{c} \neq \phi(\boldsymbol{x};\mathcal{W}_0)$ is some fixed center of the hypersphere in the output space of the network.

Compared to the soft-boundary objective (5.1), the radius parameter R and hyperparameter $\nu \in (0,1]$ have been removed. This has the effect that the balance of the objective is not given in a configuration that allows some representations to be outside the sphere. To explain, consider the empirical error term of the soft-boundary objective (5.1):

$$\frac{1}{\nu n}\sum_{i=1}^{n}\max\{0, \|\phi(\boldsymbol{x}_i;\mathcal{W})-\boldsymbol{c}\|_2^2 - R^2\} \tag{5.4}$$

Assuming the network parameters $\mathcal{W}$ are held fixed, changing radius R changes the empirical error proportionally to the number of current outliers, that is the samples where the maximum is positive. This results in the objective for fixed network parameters $\mathcal{W}$ to be adjusted in R according to

$$R^2 - \frac{n_{\text{out}}}{\nu n}R^2 = \left(1 - \frac{n_{\text{out}}}{\nu n}\right)R^2, \tag{5.5}$$

where n_{out} denotes the number of outliers in the current configuration. Therefore, if $n_{\text{out}} > \nu n$ radius R is increased to lower the objective by reducing the penalties, if $n_{\text{out}} < \nu n$ the volume of the hypersphere dominates the objective and radius R is decreased in order to lower the objective. Balance in R is reached for $n_{\text{out}} = \nu n$. In fact this already explains why the ν-property is sustained in deep SVDD as for the state in

which the network parameters have converged to a local minimum, the objective can always be improved by adjusting R such that the number of outliers is equal to νn. In the hard-boundary objective (5.3) this property has been removed and the aim is instead to minimize the volume of an all-data-enclosing hypersphere with radius R^* given by the largest distance to the center, that is

$$R^* = \max_{i=1,\ldots,n} \|\phi(\boldsymbol{x}_i; \mathcal{W}^*) - \boldsymbol{c}\|_2^2, \tag{5.6}$$

where $\mathcal{W}^*$ are the network weights of the converged model. The anomaly score for some input $\boldsymbol{x} \in \mathcal{X}$ of the hard-boundary objective can be just defined as the distance from the center

$$s(\boldsymbol{x}) = \|\phi(\boldsymbol{x}; \mathcal{W}^*) - \boldsymbol{c}\|_2^2, \tag{5.7}$$

and a threshold to flag outliers could be determined from the distribution of scores (e.g. the 95%-quantile).

Optimization

Both of the objectives (5.1) and (5.3) being composed of regularization terms and an empirical loss can be trained via mini-batch or stochastic gradient descent and its different flavors (e.g. (Nesterov-)momentum, adam, adadelta, etc.) as explained in section 4.1 by using automatic differentiation, that is the backpropagation algorithm. Hence, training the model has linear complexity in the number of batches where every batch is used to perform one gradient update step. This allows for better scaling in training set sizes and enables large-scale applications as well as iterative or online learning, though the complexity of each batch update has to be considered which depends on the richness of the deep network (i.e. the number of network parameters to be updated) and the size of the batch and the dimensionality of the inputs. To tune the hyperparameters of the model (number of layers and units in the network architecture, learning rate, regularization hyperparameters) the value of the objective on a validation set can be considered. Observing the objective on a validation set can also serve to implement an early stopping routine that aborts training if the validation objective does not improve over some window of previous iterations or in the case that the validation objective even starts to increase, which would indicate the model starting to overfit on the training data. Another aspect to consider, as was mentioned above, is the initialization of the network parameters where various options are available.

For the implementation of the method and its objective to run the experiments reported in section 6, the `Theano` Python library, which allows automatic differentiation and supports GPUs for training, in conjunction with the `Lasagne` Python framework (Dieleman et al., 2015) are used.

5.2 Properties of Deep SVDD

Denote the cost functions of the soft-boundary and hard-boundary deep SVDD problems by J_{soft} and J_{hard} respectively, that is

$$J_{\text{soft}}(R, \mathcal{W}) = \frac{\lambda}{2}\sum_{\ell=1}^{L}\|\boldsymbol{W}^{\ell}\|_F^2 + R^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\max\{0, \|\phi(\boldsymbol{x}_i;\mathcal{W}) - \boldsymbol{c}\|_2^2 - R^2\} \quad (5.8)$$

and

$$J_{\text{hard}}(\mathcal{W}) = \frac{\lambda}{2}\sum_{\ell=1}^{L}\|\boldsymbol{W}^{\ell}\|_F^2 + \frac{1}{n}\sum_{i=1}^{n}\|\phi(\boldsymbol{x}_i;\mathcal{W}) - \boldsymbol{c}\|_2^2. \quad (5.9)$$

The ν-property

The first proposition on the (soft-boundary) deep SVDD we have already mentioned above, which is that the ν-property of kernel SVDD is preserved in the following form:

Proposition 5.1 (ν-property)**.** *The fraction of outliers in the soft-boundary deep SVDD model is given by $\nu \in (0, 1]$ which is specified a priori.*

Proof. Consider the derivative of the soft-boundary objective function J_{soft} with respect to R,

$$\frac{\partial J_{\text{soft}}}{\partial R} = 2R - 2\frac{n_{\text{out}}}{\nu n}R = 0, \quad (5.10)$$

where n_{out} is the number of positive maxima, that is the number of outliers. As $R > 0$ for a non-trivial solution, we get that $\nu = \frac{n_{\text{out}}}{n}$ for the optimal radius R^*. □

Avoiding trivial solutions

The following three propositions characterize situations in which deep SVDD is likely to converge to a trivial solution which is given if for every input $\boldsymbol{x} \in \mathcal{X}$ the trained network returns exactly the same representation equal to center $\boldsymbol{c}$ in which case the hypersphere collapses into one point of zero volume.

Choosing a hypersphere center $\boldsymbol{c}$ First, we state which choice for the center of the hypersphere should be avoided.

Proposition 5.2 (All-zero-weights network solution)**.** *If $\boldsymbol{c} = \phi(\boldsymbol{x}; \mathcal{W}_0)$, where $\mathcal{W}_0$ is the set of all-zero network parameters, the optimal solution of deep SVDD is given by $\mathcal{W}^* = \mathcal{W}_0$ and $R^* = 0$ which implies the hypersphere to collapse into its center.*

Proof. For every configuration $(R, \mathcal{W})$ we have that $J_{\text{soft}}(R, \mathcal{W}) \geq 0$ and $J_{\text{hard}}(\mathcal{W}) \geq 0$ respectively. As the output of the all-zero-weights network $\phi(\boldsymbol{x}; \mathcal{W}_0)$ is identical for every input $\boldsymbol{x} \in \mathcal{X}$, since the linear transformation in the first layer puts every input to zero (and again in every subsequent layer), and the center of the hypersphere is given by $\boldsymbol{c} = \phi(\boldsymbol{x}; \mathcal{W}_0)$, all penalties in the empirical term of the objective become zero. Thus, $R^* = 0$ and $\mathcal{W}^* = \mathcal{W}_0$ are the optimal solutions yielding regularization terms of zero resulting in $J_{\text{soft}}(R^*, \mathcal{W}^*) = 0$ and $J_{\text{hard}}(\mathcal{W}^*) = 0$ respectively. □

Proposition 5.2 is the reason why $\boldsymbol{c} \neq \phi(\boldsymbol{x}; \mathcal{W}_0)$ is demanded in fixing the center of the hypersphere in the deep SVDD objectives (5.1) and (5.3) as to avoid a trivial "black hole"-solution. The specific output $\phi(\boldsymbol{x}; \mathcal{W}_0)$ is determined by the activation function used in the final layer of the network since $\phi(\boldsymbol{x}; \mathcal{W}_0) = (\sigma(0), \ldots, \sigma(0))^T$ where σ is the activation function in the final layer. Hence, if for example a convolutional neural network with ReLU activations is applied, $\boldsymbol{c} > \boldsymbol{0}$ should be demanded as the Rectifier has zero value in the origin and its image is given by the positive orthant. To avoid numerical issues, the center should take a value far enough away from the origin in this case. If the features of the data are rescaled to $[0, 1]$ before being fed to the network, for example, choices could be $\boldsymbol{c} = \frac{1}{2}(1, \ldots, 1)^T \in \mathcal{F}$ or $\boldsymbol{c} = \frac{1}{\sqrt{p}}(1, \ldots, 1)^T \in \mathcal{F}$ lying on the unit sphere with network output space $\mathcal{F} \subset \mathbb{R}^p$. Another way to fix $\boldsymbol{c}$ would be to perform a forward pass on (all or a fraction of) the data on the initialized network before starting to train and fix $\boldsymbol{c}$ to be the mean of the network representations using some tolerance criterion to check whether the mean is not too close to the "black hole". Such an initialization increases numerical stability as well as training efficiency as $\boldsymbol{c}$ is initialized in the neighborhood of the initial network outputs.

The case of the trivial all-zero-parameters hypersphere-imploding solution is also the reason why $\boldsymbol{c}$ has to be fixed in the network output space in the first place as if $\boldsymbol{c}$ would be a free optimization parameter convergence to the trivial solution is likely.

Networks with bias terms Next, we will explain why including bias terms in the network again can lead to the trivial solution of the hypersphere collapsing into its center.

Proposition 5.3 (Network with bias term solution)**.** *Let $\boldsymbol{c}$ be any fixed center in the output space of network ϕ. If there is a hidden layer with bias term present in network ϕ, there exists an optimal solution $(R^*, \mathcal{W}^*)$ of the (unregularized) deep SVDD objectives* (5.1) *and* (5.3) *with $R^* = 0$ and $\phi(\boldsymbol{x}; \mathcal{W}^*) = \boldsymbol{c}$ for every $\boldsymbol{x} \in \mathcal{X}$.*

Proof. Assume a bias term $\boldsymbol{b}^\ell$ is given in some layer $\ell \in \{1, \ldots, L\}$ of the network. For any input $\boldsymbol{x} \in \mathcal{X}$ the output of layer ℓ is given by

$$\boldsymbol{z}^\ell(\boldsymbol{x}) = \sigma^\ell(\boldsymbol{W}^\ell \cdot \boldsymbol{z}^{\ell-1}(\boldsymbol{x}) + \boldsymbol{b}^\ell), \tag{5.11}$$

where "$\cdot$" is again a linear operator (e.g. matrix multiplication or convolution) and the input to layer ℓ, $\boldsymbol{z}^{\ell-1}$, depends on input $\boldsymbol{x}$ via concatenation of the previous layers. Then,

by setting $\boldsymbol{W}^\ell = \mathbf{0}$, we have that $\boldsymbol{z}^\ell(\boldsymbol{x}) = \sigma^\ell(\boldsymbol{b}^\ell)$, i.e. the output of layer ℓ is constant for every input $\boldsymbol{x} \in \mathcal{X}$. Thus, the bias term $\boldsymbol{b}^\ell$ (and the weights of the subsequent layers) can be chosen such, that $\phi(\boldsymbol{x}; \mathcal{W}^*) = \boldsymbol{c}$ for every $\boldsymbol{x} \in \mathcal{X}$ (assuming $\boldsymbol{c}$ lies in the image of the network as a function of $\boldsymbol{b}^\ell$ and the subsequent parameters $\boldsymbol{W}^{\ell+1}, \ldots, \boldsymbol{W}^L$). Hence, the optimal solution is given by such a choice of parameters $\mathcal{W}^*$, resulting in the empirical term being zero, and $R^* = 0$ (ignoring the network weight decay regularization terms in the objectives) □

This implies that if there is at least one bias term present in the network, the network potentially can replace the information given by any input $\boldsymbol{x} \in \mathcal{X}$ with an artificial representation of one point that can be scaled towards the center of the hypersphere allowing again the trivial solution of a zero-volume hypersphere. This is also the reason why strictly speaking we would have to make the assumption that there is variability in every input dimension of the data since otherwise the weight in the first layer corresponding to a constant (non-zero) feature effectively is a bias term. The presence of the network parameter regularization term could prevent the trivial solution even if bias terms are employed, yet the proper selection of λ and ν to obtain stable results we found to be a delicate process. To increase robustness of the deep SVDD method, no bias terms should therefore be used in the deep network chosen and thus inputs should be normalized in pre-processing. Moreover, if features are rescaled in pre-processing, they should not be centered at 0 as signals close to 0 are hard to recover if no bias terms are given in the network, that is rescaling to the unit interval $[0, 1]$ should be preferred opposed to $[-1, 1]$ for example.

Bounded activation functions As outlined above, bias terms should not be employed as to avoid convergence to a trivial solution and for this reason, features should not be centered at zero but rather translated and rescaled on the positive (or negative) axis (e.g. the unit interval). However, if there are input features given that take exclusively positive (or exclusively negative) values, the choice of activation functions in the hidden layer must be considered.

Proposition 5.4 (Bounded activation functions)**.** *Assume a network unit has a monotonic increasing activation function σ that has an upper or lower bound (or both) with $\sup_x \sigma(x) \neq 0$ or $\inf_x \sigma(x) \neq 0$ (or both). Then, for a set of inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n$ which have at least one feature that is positive (or negative) for every input, the non-zero supremum or infimum can be approximated arbitrarily close uniformly on the set of inputs.*

Proof. Without loss of generality, consider the case of σ being upper bounded with $B := \sup_x \sigma(x) \neq 0$ and feature k being positive for all inputs, i.e. $z_i^{(k)} > 0$ for every $i = 1, \ldots, n$. Then, for every $\varepsilon > 0$ one can always choose the weight of feature k, w_k, large enough (while setting all other weights to zero) such that

$$\sup_{i=1,\ldots,n} \left| \sigma\left(w_k \, z_i^{(k)}\right) - B \right| < \varepsilon. \tag{5.12}$$

□

Proposition 5.4 is a rather trivial statement and simply says that if we have a bounded activation function (with supremum or infimum different from zero) and one feature that is positive (or negative) for all inputs, the respective unit can effectively be saturated (to a value different from zero) for every input. However, a unit that is effectively saturated for every input can be problematic for deep SVDD as again a constant unit in a hidden layer again the subsequent layer to have effectively a bias term. Hence, using bounded activation functions in the hidden layers (such as the sigmoid activation) enables the network to learn a bias term on its own which allows the algorithm again to converge to the trivial one-point solution illustrated above. As for the presence of bias terms above, using adequate regularization on the network parameters could prevent the parameters from exploding and saturating units for all inputs, but this is again a sensitive task and solutions are not robust. In conclusion, using ReLU activations (which anyway alleviate the vanishing gradient problem) or leaky ReLU activations (which alleviate the "dying ReLU" problem) should be preferred compared to bounded activation functions in the hidden layers.

6 Experiments and Application

In this section, we will put the novel method of deep SVDD to the test with the aim to evaluate its performance quantitatively as well as qualitatively. As deep SVDD is designed for the task of large-scale and high-dimensional anomaly detection, we will perform experiments on image data and choose data sets which are well-known in the machine learning literature, namely MNIST, CIFAR-10, and the bedrooms data set from the Large-scale Scene Understanding Challenge. Image data has the additional advantage of easy qualitative evaluation by just looking at the respective pictures, that is we can check visually the images that are considered to be typical or atypical within a certain target class according to the method used. In general, the field of (high-dimensional) unsupervised anomaly detection lacks commonly used benchmark data sets (see e.g. Emmott et al., 2016, for an attempt to change this situation) in comparison to the general domain of supervised learning for instance, which is why classification data sets (as the ones we are using) are often employed for one-class classification by training the models just on one of the classes. Having ground truth labels available (one-class vs. all other classes which are considered to be various kinds of anomalies) for testing also allows the quantitative evaluation by accuracy measures, where the *Area Under the Receiver Operating Characteristic (ROC) curve* (*AUROC* or *AUC* in short) (Hanley and McNeil, 1982) is the commonly used metric in anomaly detection (cf. the works mentioned in section 4.3). But again, we emphasize that training is carried out only on sets that include samples from one target class, that is training is performed in the unsupervised one-class classification setting.

6.1 MNIST

Experimental Setup

Data set The MNIST data set of handwritten digits (LeCun et al., 2010) has a training set of 60,000 examples, and a test set of 10,000 examples where each example is a grayscale image of size 28×28 and belongs to one of the 10 digit classes 0–9. Figure 6.1 shows 10 randomly selected examples from each class.

We perform global contrast normalization with the L^1-norm as a pre-processing

Figure 6.1: Ten random samples from each class in MNIST.

step to normalize the input signals, that is for each image the mean across its pixels is subtracted (which implies every image to have an equal mean signal) and the contrast (i.e. variation of pixels) is normalized by pixel intensity (i.e. the L^1-norm). Global contrast normalization is an important pre-processing step in computer vision to account for different settings of light and background fillings, though more relevant for color than grayscale images. Normalizing the signals moreover assures the method not to pick up on pure signal strength but on form and shape as well as location instead. Finally, min-max rescaling is applied to shift the pixels back to the unit interval $[0, 1]$ using the minimum and maximum value over all pixels and examples in the training set.

To create one-class classification settings, we extract from the training set of 60,000 images the examples of each class leading to 10 setups where each training set includes about 6,000 samples of one class. The training sets are shuffled and reduced to the largest sample size resulting in batches of equal size giving the training set sizes listed in Table 6.1. For testing, the original test set of 10,000 examples is used which has about 1,000 samples from each class. Hence, in each experiment there are 9 anomalous classes present in the test set.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
n	5800	6600	5800	6000	5800	5400	5800	6200	5800	5800

Table 6.1: MNIST training set sizes in each one-class classification setup.

Deep SVDD The network architecture used in deep SVDD on MNIST is given in Figure 6.2. No bias terms are present in the network for the reasons explained in section 5.2. We train the network via mini-batch gradient descent in two ways, once with plain momentum and once with the adaptive Adam optimizer. For momentum we use momentum term $\alpha = 0.9$ and learning rate $\eta = 0.0001$ and train for 100 epochs which shows smooth convergence. For Adam we can use a higher learning rate of $\eta = 0.001$ which leads to a smooth convergence of the objective as well but earlier which is why we only train for 20 epochs and otherwise use the recommended hyperparameters given in the original paper (Kingma and Ba, 2014). The batch size is set to 200 in both cases and we apply weight decay with $\lambda = 0.001$ though we will see that using no weight decay at all leads to similar results. To initialize the weights of the network we use Uniform Glorot weight initialization (Glorot and Bengio, 2010) as introduced in section 4.1. For the initialization of the filter bank in the first layer, we also try a more sophisticated strategy by training a sparse dictionary (Mairal et al., 2009) on 5×5 patches extracted from 500 randomly selected images using the implementation given in `scikit-learn`. For the soft-boundary objective, we set $\nu = 0.1$.

Figure 6.2: Network architecture for MNIST.

Baselines To compare the results of deep SVDD we also train a OC-SVM with Gaussian kernel (i.e. perform SVDD with Gaussian kernel), a kernel density estimator, and a convolutional autoencoder as another deep anomaly detection method. The OC-SVM and KDE are the implemented with `scikit-learn` and the CAE also with `Lasagne` and `Theano`. We choose the hyperparameter of the Gaussian kernel to be the maximum Euclidean distance found between samples in the training set as proposed in Munoz and Moguerza (2004) in order to minimize the numerical errors arising from the use of the exponential. In the OC-SVM we also set $\nu = 0.1$ as in soft-boundary deep SVDD. For the encoder of the CAE, we use the same architecture as in deep SVDD given in Figure 6.2 and the decoder is build exactly symmetrical where the 2×2 max-pooling layers are replaced with simple 2×2 upsampling layers. The mean squared error is used as the reconstruction error and training is performed with the Adam optimizer again with a learning rate of $\eta = 0.0001$ to obtain smooth convergence behavior. The CAE is also trained with batch size 200 but for 150 epochs as the objective takes longer to converge.

Results

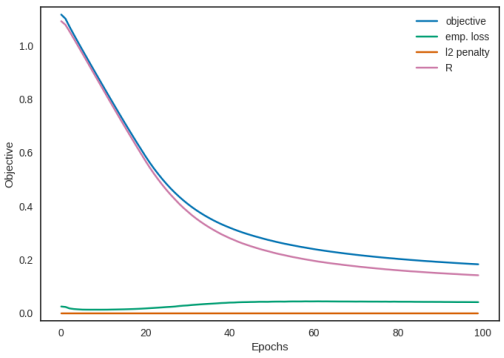
The results in each of the 10 one-class classification setups for the baselines and hard- as well as soft-boundary deep SVDD (trained with plain momentum) are given in Table 6.2.

	OC-SVM/SVDD	KDE	CAE	hard-boundary Deep SVDD	soft-boundary Deep SVDD
0 vs. all	0.9858	0.9712	**0.9876**	0.9662	0.9599
1 vs. all	**0.9947**	0.9889	0.9934	0.9866	0.9445
2 vs. all	0.8235	0.7898	**0.9165**	0.8692	0.8535
3 vs. all	0.8821	0.8622	0.8845	**0.8902**	0.8209
4 vs. all	**0.9490**	0.8788	0.8621	0.9311	0.8895
5 vs. all	0.7705	0.7384	**0.8575**	0.7880	0.7782
6 vs. all	**0.9646**	0.8757	0.9540	0.9431	0.9500
7 vs. all	0.9368	0.9145	**0.9397**	0.9119	0.8571
8 vs. all	0.8879	0.7922	0.8233	**0.8935**	0.8831
9 vs. all	0.9314	0.8817	**0.9648**	0.9343	0.9266

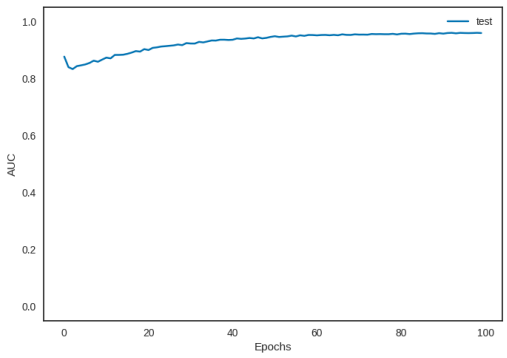
Table 6.2: AUCs per method on MNIST.

As we can see, deep SVDD gives competitive results. Comparing the two deep SVDD performances suggests that hard-boundary deep SVDD performs better which is an indication that the joint optimization of the network weights and radius R of the hypersphere needs further investigation and fine-tuning. Figure 6.3 shows diagnostics for soft-boundary deep SVDD exemplary for the 0 vs. all experiment. In (a) we see that from around epoch 20 to 80 the empirical error term increases again on the train set and flattens as radius R starts to converge which is the ν-property in action. The increase of the empirical term on the test set in (c) shows that anomalies are placed outside the hypersphere as it contracts and the subsequent decrease shows that the network weights are finally updated such that inputs in general are mapped closer the center of the sphere as the volume begins to balance. Moreover we see in (b) that AUC improves over time as the transformation learned by the network adapts to the factors of variation in images of zeros. In (d) we see the median deep SVDD anomaly score over time for the samples of the target and the anomalous class in the test set with 5%- and 95%-quantile confidence bands as well as minimum and maximum bounds which shows that almost all examples of the target class lie within the hypersphere but close to its boundary as the volume balances whereas the large portion of the anomalies are placed outside the hypersphere.

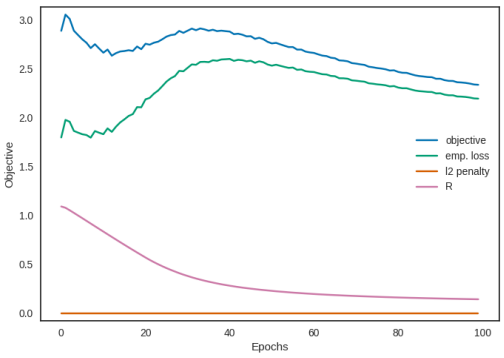
In Table 6.3 we compare the effect of the more sophisticated dictionary initialization strategy for the first layer filters, but can see that the initialization does not lead to a clear improvement as the results without initialization in 4 out of 10 experiments give a better performance for an otherwise equal Adam optimization configuration. To see the difference in performance between the two optimizers, we put the results of momentum training to the left and recognize that both solvers lead to similar results. To examine the potential of hierarchical representations of deep networks, the last column shows the result if we initialize the weights from network parameters obtained by training the network in Figure 6.2 on a supervised 10-class categorical cross-entropy loss preceded by a softmax layer, which enables deep SVDD to almost perfectly enclose the target class in each setup.



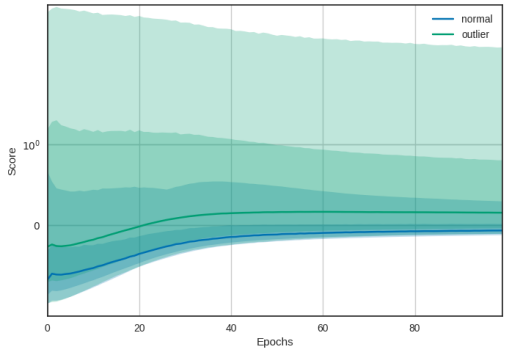
(a) Train objective with its parts.



(b) AUC on test set.



(c) Test objective with its parts.



(d) AD score distribution on test set.

Figure 6.3: Example of diagnostics for soft-boundary deep SVDD with $\nu = 0.1$ for the 0 vs. all MNIST experiment.

	Momentum (w/ dict. init)	Adam (w/ dict. init)	Adam (wo dict. init)	Adam (supervised init)
0 vs. all	0.9662	**0.9854**	0.9780	0.9997
1 vs. all	**0.9866**	0.9805	0.9804	0.9996
2 vs. all	0.8692	0.8700	**0.8911**	0.9977
3 vs. all	0.8902	0.8561	**0.8903**	0.9979
4 vs. all	**0.9311**	0.8602	0.9235	0.9998
5 vs. all	0.7880	**0.8221**	0.7886	0.9977
6 vs. all	0.9431	**0.9622**	0.9044	0.9991
7 vs. all	0.9119	**0.9296**	0.8949	0.9989
8 vs. all	**0.8935**	0.8774	0.8820	0.9940
9 vs. all	0.9343	**0.9363**	0.9095	0.9971

Table 6.3: AUCs for hard-boundary deep SVDD with different initializations on MNIST.

In order to see the effect of weight decay regularization, Table 6.4 shows the results with ($\lambda = 0.001$) and without weight decay where we can see that the results are very similar thus indicating a good generalization performance of deep SVDD even without

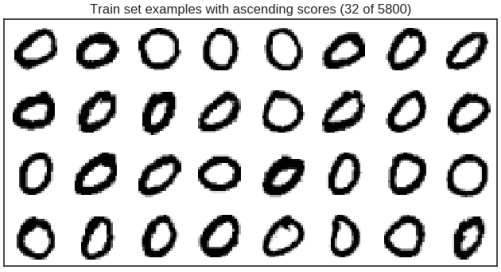
additional regularization.

	Adam (w/ wd)	Adam (wo wd)
0 vs. all	0.9854	**0.9858**
1 vs. all	**0.9805**	0.9753
2 vs. all	0.8700	**0.8718**
3 vs. all	**0.8561**	0.8556
4 vs. all	0.8602	**0.8701**
5 vs. all	**0.8221**	0.8185
6 vs. all	**0.9622**	0.9603
7 vs. all	**0.9296**	0.9272
8 vs. all	**0.8774**	0.8767
9 vs. all	**0.9363**	0.9285

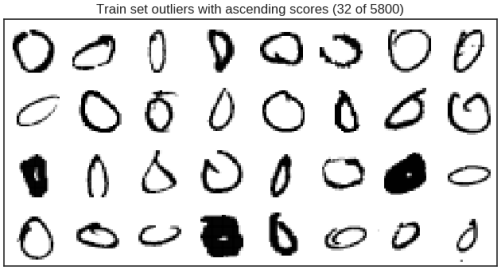
Table 6.4: AUCs for hard-boundary deep SVDD with and without weight decay on MNIST.

Finally, to examine the results qualitatively, Figure 6.4 shows the examples of the target class with the lowest and highest AD scores in each setup for the hard-boundary deep SVDD trained with the Adam optimizer, where we can clearly see that samples with atypical shapes are mapped further away from the center of the hypersphere by the deep network.

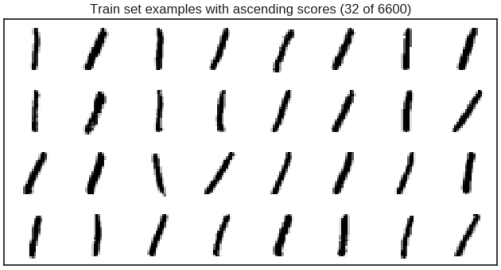
Train set examples with ascending scores (32 of 5800)



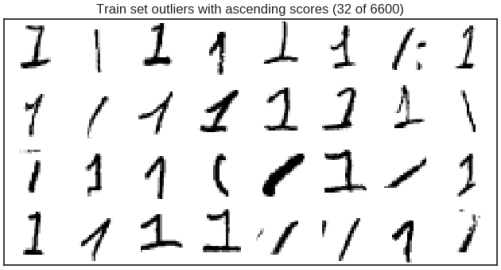
Train set outliers with ascending scores (32 of 5800)



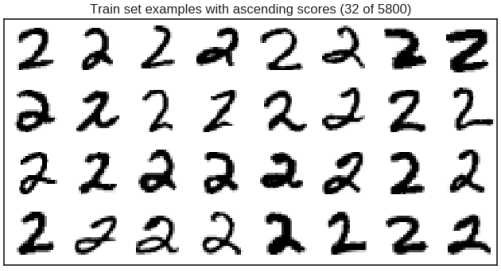
Train set examples with ascending scores (32 of 6600)



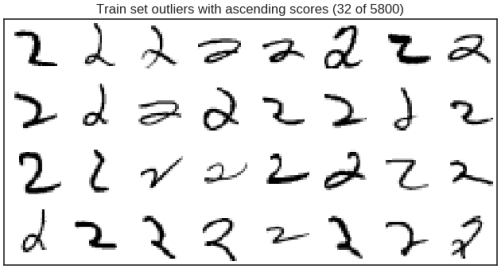
Train set outliers with ascending scores (32 of 6600)



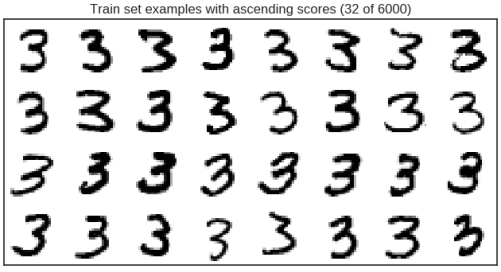
Train set examples with ascending scores (32 of 5800)



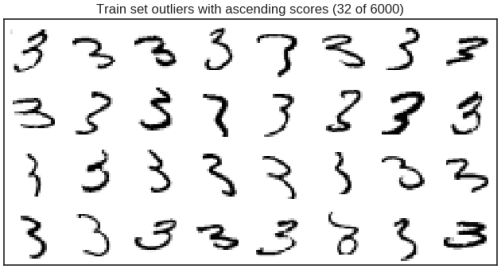
Train set outliers with ascending scores (32 of 5800)



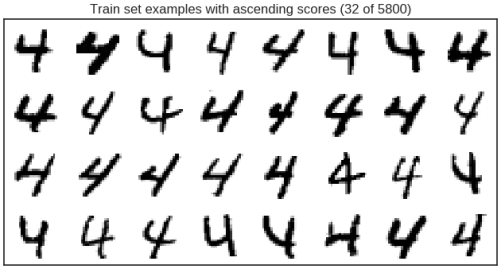
Train set examples with ascending scores (32 of 6000)



Train set outliers with ascending scores (32 of 6000)



Train set examples with ascending scores (32 of 5800)



Train set outliers with ascending scores (32 of 5800)

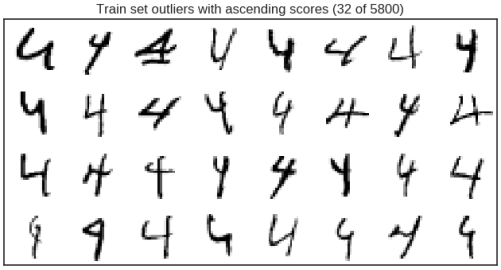


Figure 6.4: Examples of the MNIST one-class training sets with the lowest (top left) and highest (bottom right) deep SVDD anomaly scores in each experiment.

6.2 CIFAR-10

Experimental Setup

Data set CIFAR-10 (Krizhevsky and Hinton, 2009) is a data set of 32×32 RGB-color images from 10 classes with a training set of 50,000 images and a test set of 10,000 images where the training set includes exactly 5,000 and the test set exactly 1,000 examples per class. Figure 6.5 shows 10 randomly selected images from each class.

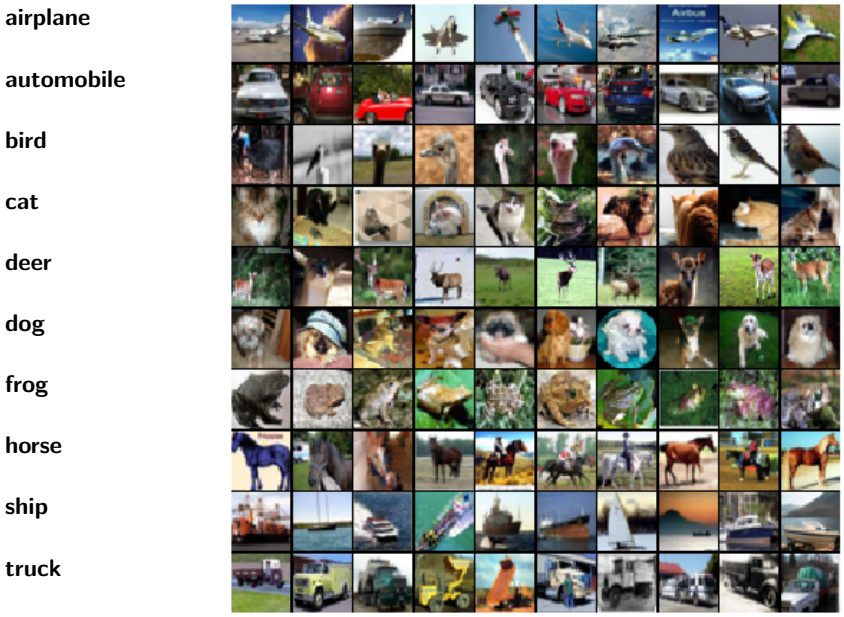


Figure 6.5: Ten random samples from each class in CIFAR-10.

Again, we perform global contrast normalization with the L^1-norm as a pre-processing step to normalize the images and do a final min-max rescale of the pixels to the unit interval $[0, 1]$ using the minimum and maximum value over all pixels and examples in the train set. The one-class classification setups are created analogously to MNIST from the 10-class training set by extracting training sets that only contain the samples of one class. This gives in turn one setup per class where each training set includes exactly $n = 5{,}000$ images which are shuffled before training. For the test set in each setup the original test set of 10,000 examples is used causing again 9 different classes of outliers to exist in testing.

Deep SVDD Figure 6.6 shows the network architecture used in deep SVDD on CIFAR-10 which again does not include bias terms. Training is performed with mini-batch gradient

descent again once with plain momentum and once with the Adam optimizer. For plain momentum we use $\alpha = 0.9$ and learning rate $\eta = 0.00001$ and train for 100 epochs giving smooth convergence. For Adam we can set the learning rate again one order of magnitude higher to $\eta = 0.0001$ train only for 50 epochs as convergence is faster. As the network is much richer than the one used on MNIST we choose $\lambda = 10^{-6}$ though again we will see that using no weight decay regularization at all gives similar results indicating good generalization performance of deep SVDD. The batch size is set to 200 and for initialization of the network weights, we again do Uniform Glorot weight initialization except for the filters in the first layer which we also initialize with the elements of a sparse dictionary trained on 5×5 patches extracted from 500 randomly selected images of the respective training set. In soft-boundary deep SVDD we fix $\nu = 0.1$.

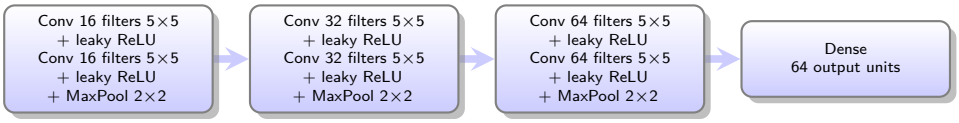


Figure 6.6: Network architecture for CIFAR-10.

Baselines We exercise the same baseline methods as previously, that is a OC-SVM with Gaussian kernel (i.e. SVDD with Gaussian kernel), KDE, and a CAE. To make training feasible for the OC-SVM and KDE, we apply PCA to reduce the dimensionality as an additional step in the pre-processing pipeline where the number of eigenvectors is selected such that 95% of the variance is retained. The hyperparameter of the Gaussian kernel is again selected as proposed in Munoz and Moguerza (2004). In the OC-SVM we also set $\nu = 0.1$. The CAE is build symmetrically from the architecture given in Figure 6.6 where max-pooling is again substituted with upsampling in the decoder part. The objective of the CAE is equally chosen to be the quadratic reconstruction error as before and training is performed with Adam with learning rate $\eta = 0.0001$ for 150 epochs with batches of size 200.

Results

Tables 6.5–6.7 show the results for CIFAR-10 of the analogous experiments as performed on MNIST. As objects in CIFAR-10 are not centered as in MNIST, a worse performance for vanilla OC-SVM and KDE should be expected as the images are compared globally though KDE does give good results on some of the sets caused by certain background properties being characteristic for a lot of examples in some of the target classes (e.g. the horizontal division of the background into sea and sky for ships) which can be seen from the most typical and atypical examples according to KDE which is provided in the supplementary material (cf. Figure C.1 in appendix C). In fairness, the full potential of the smoothing methods (i.e. OC-SVM and KDE) is not shown here as they could also be

trained on locally extracted features such as the SIFT (Scale-invariant feature transform) features (Lowe, 1999). Table 6.5 shows that the hard-boundary deep SVDD performs best on CIFAR-10 for most of the target classes. Moreover, the results for soft-boundary deep SVDD confirm the suspicion, that the joint gradient-descent optimization of the network weights and radius R needs further investigation. Notably, the hard-boundary deep SVDD performs generally better than the deep competitor which is the CAE in our experiments.

		OC-SVM/SVDD	KDE	CAE	hard-boundary Deep SVDD	soft-boundary Deep SVDD
airplane	vs. all	0.5311	0.6467	0.5806	**0.6560**	0.4707
automobile	vs. all	0.6265	0.6293	0.5239	**0.6355**	0.6207
bird	vs. all	0.4388	0.5029	0.4493	**0.5518**	0.4946
cat	vs. all	0.5285	0.5653	**0.5774**	0.4634	0.4784
deer	vs. all	0.5244	0.6346	0.4903	**0.6356**	0.3884
dog	vs. all	0.5397	0.6160	0.5970	**0.6258**	0.5584
frog	vs. all	0.6873	**0.7180**	0.4204	0.7146	0.4368
horse	vs. all	0.5935	0.6376	0.5539	**0.6924**	0.4928
ship	vs. all	0.6783	**0.7776**	0.7492	0.7166	0.5662
truck	vs. all	0.7359	**0.7551**	0.6207	0.7245	0.6829

Table 6.5: AUCs per method on CIFAR-10.

Table 6.6 shows that the results using plain momentum or the Adam optimizer are not as consistent as for MNIST for some of the target class (e.g. bird, cat, or horse). Moreover, there is not clear benefit evident for using the more sophisticated first layer filter initialization via dictionary learning. Looking at the last column shows again the potential of hierarchical representations though the level of AUCs generally is low on CIFAR-10. The results in Table 6.7 are another indication, that weight decay is not essential for the generalization performance of deep SVDD as the performances are similar with and without including weight penalties into the objective.

		Momentum (w/ dict. init)	Adam (w/ dict. init)	Adam (wo dict. init)	Adam (supervised init)
airplane	vs. all	0.6560	**0.6667**	0.6228	0.6807
automobile	vs. all	**0.6355**	0.6191	0.6041	0.6645
bird	vs. all	**0.5518**	0.4934	0.4530	0.5753
cat	vs. all	0.4634	0.5748	**0.5862**	0.6966
deer	vs. all	**0.6356**	0.5904	0.5210	0.6953
dog	vs. all	**0.6258**	0.5857	0.5954	0.7156
frog	vs. all	**0.7146**	0.6666	0.5878	0.7823
horse	vs. all	**0.6924**	0.6201	0.6338	0.7428
ship	vs. all	0.7166	**0.7749**	0.7680	0.8078
truck	vs. all	**0.7245**	0.6991	0.7035	0.6969

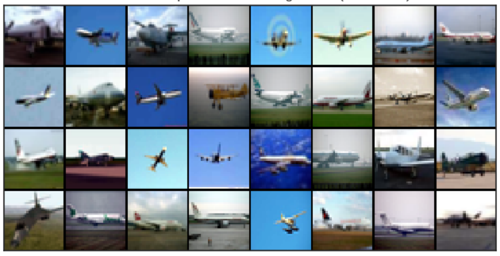
Table 6.6: AUCs for hard-boundary deep SVDD with different initializations on CIFAR-10.

		Adam (w/ wd)	Adam (wo wd)
airplane	vs. all	**0.6667**	0.6456
automobile	vs. all	0.6191	**0.6237**
bird	vs. all	0.4934	**0.4981**
cat	vs. all	**0.5748**	0.5680
deer	vs. all	**0.5904**	0.5871
dog	vs. all	**0.5857**	0.5827
frog	vs. all	**0.6666**	0.5930
horse	vs. all	0.6201	**0.6455**
ship	vs. all	**0.7749**	0.7685
truck	vs. all	**0.6991**	0.6958

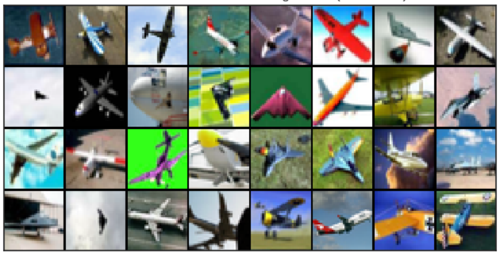
Table 6.7: AUCs for hard-boundary deep SVDD with and without weight decay on CIFAR-10.

At last, Figure 6.7 allows again a qualitative visual assessment of what is considered to be typical and atypical within each of the respective target classes. We can see that deep SVDD picks up on unusual colors as well as on atypical shapes and angles with respect to each target class. Comparing the findings visually with the AD ranking according to KDE (cf. Figure C.1 in appendix C) suggests that deep SVDD finds the visually more interesting anomalies on CIFAR-10.

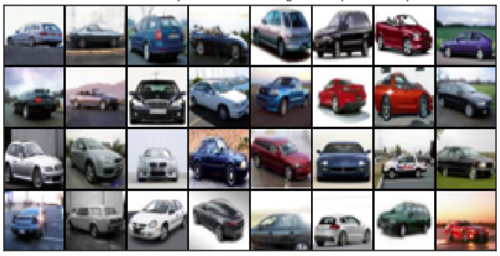
Train set examples with ascending scores (32 of 5000)



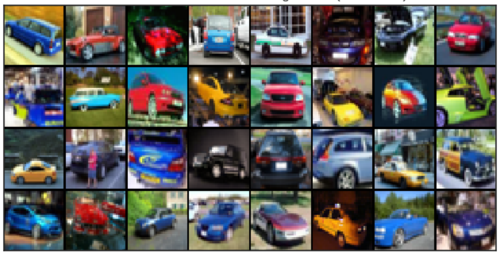
Train set outliers with descending scores (32 of 5000)



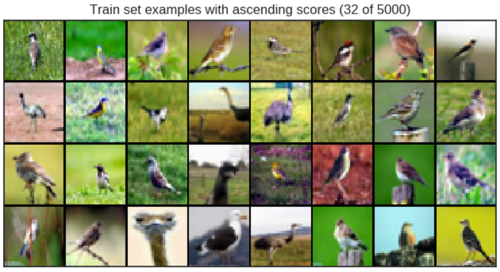
Train set examples with ascending scores (32 of 5000)



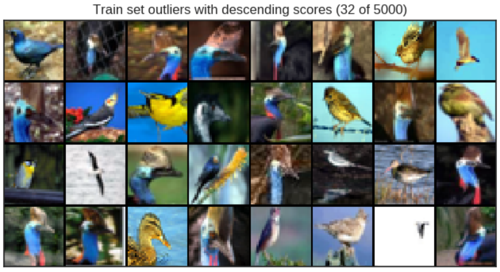
Train set outliers with descending scores (32 of 5000)



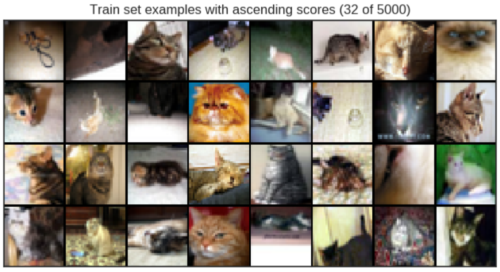
Train set examples with ascending scores (32 of 5000)



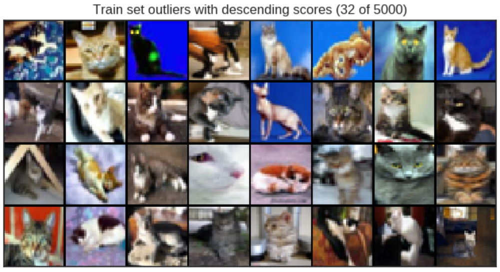
Train set outliers with descending scores (32 of 5000)



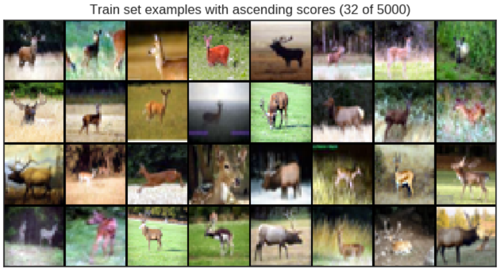
Train set examples with ascending scores (32 of 5000)



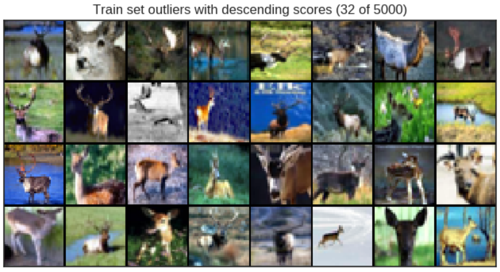
Train set outliers with descending scores (32 of 5000)



Train set examples with ascending scores (32 of 5000)



Train set outliers with descending scores (32 of 5000)



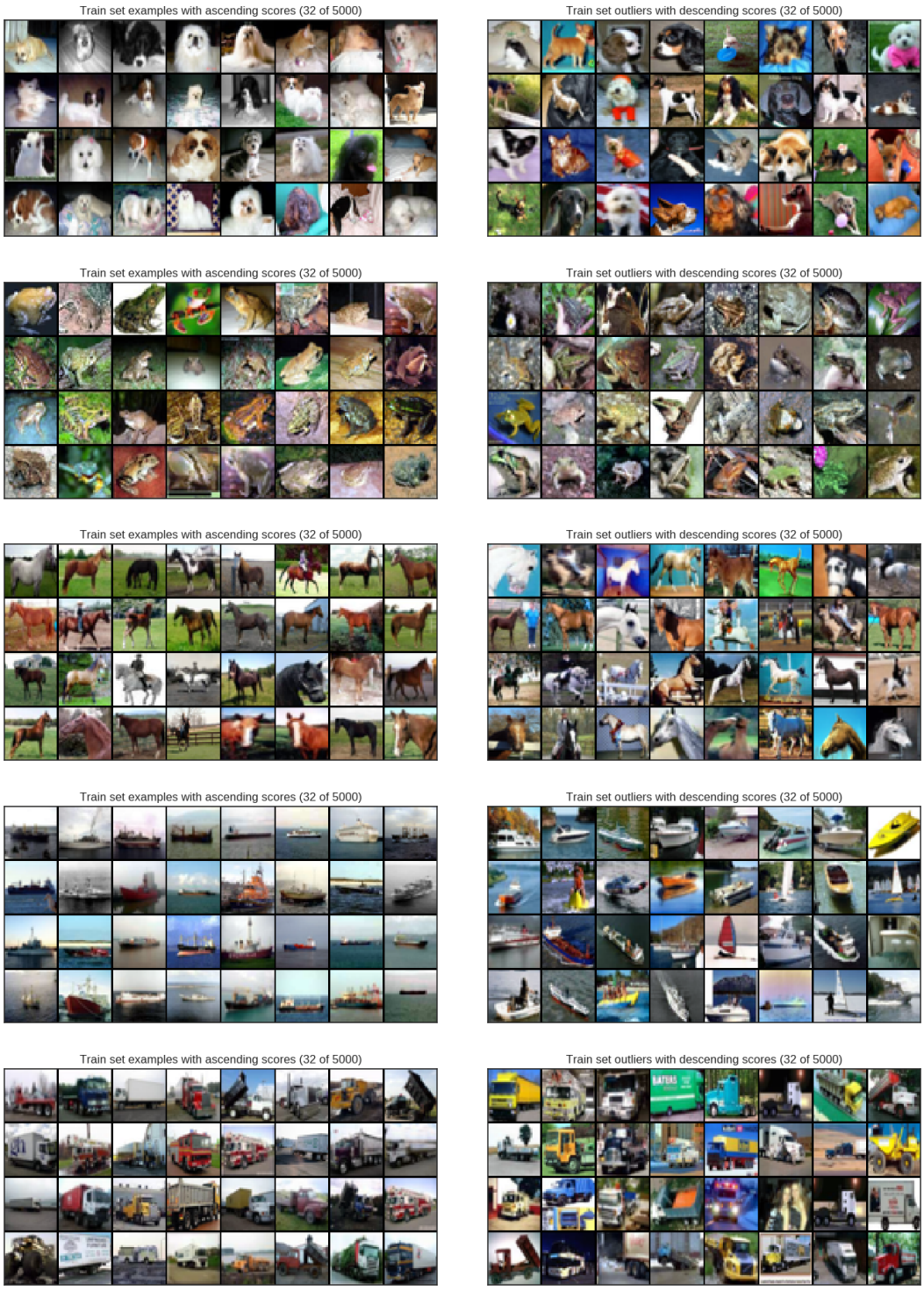


Figure 6.7: Examples of the CIFAR-10 one-class training sets with the lowest (top left) and highest (bottom right) deep SVDD anomaly scores in each experiment.

6.3 Large-scale Scene Understanding Challenge: Bedrooms

Data set We performed a final experiment on the data set from the large-scale scene understanding challenge (LSUN) (Zhang et al., 2015) to demonstrate the applicability of deep SVDD to large-scale applications. We chose bedrooms as the target class from overall 10 scene categories (Bedroom, Bridge, Church Outdoor, Classroom, Conference Room, Dining Room, Kitchen, Living Room, Restaurant, Tower). The bedrooms training set contains 3,033,042 RGB-color images of varying sizes but where the shorter side (i.e. either width or height) has 256 pixels. To feed the images into a CNN we cut out a centered 256x256 square from each image (cf. Yu et al., 2015) and downscale to 64x64 pixels afterwards to reduce the computational burden. Moreover, global contrast normalization with the L^1-norm and final rescaling to the unit interval is performed in pre-processing. As there is no public test set available in the challenge, we built a test set from the validation sets of all of the 10 classes each containing 300 images giving us a test set with 3,000 examples with again 9 anomalous classes.

Deep SVDD We choose hard-boundary deep SVDD as the model because of the optimization issues of soft-boundary deep SVDD indicated above. For the network architecture we take exactly the same architecture as for CIFAR-10 (cf. Figure 6.6) again without bias terms. To make mini-batch online learning feasible (due to memory restrictions), we load batches iteratively from the database into memory. Training was performed on a stream of $n = 3{,}000{,}000$ images drawn in random order (without replacement) from the database and the mini-batches are set to a lower size of 100 to increase stochasticity in order to lower the importance of early samples. We used a learning rate of $\eta = 0.00001$ with the adaptive Adam optimizer and weight decay regularization again with $\lambda = 10^{-6}$. The network parameters were initialized with Uniform Glorot weight initialization.

Results The resulting AUC on the test set is **0.5762**. We show the most typical and most unusual bedrooms (from the 300 validation bedrooms) in Figure 6.8 and Figure 6.9 respectively. The visual assessment of the bedrooms is less clear as on CIFAR-10 although maybe the bedrooms with higher scores appear to be slightly more colorful. However, there are also plain white bedrooms given with high scores. In conclusion, this experiment rather demonstrated the scalability of the method. To improve the qualitative results, further investigation of the optimization seems necessary.

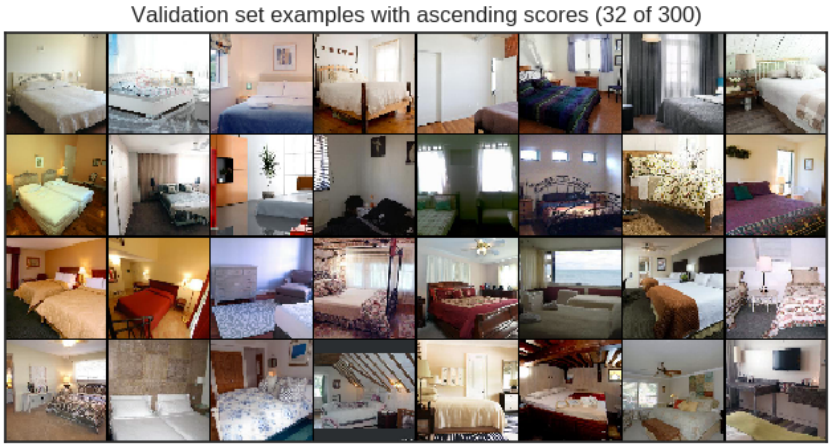


Figure 6.8: Examples from the bedroom validation set with the lowest deep SVDD anomaly scores.

Figure 6.9: Examples from the bedroom validation set with the highest deep SVDD anomaly scores.

7 Conclusion and Outlook

Review and Conclusion This thesis introduced the novel method of Deep Support Vector Data Description for the task of high-dimensional anomaly detection in large-scale applications. Deep SVDD combines the idea of kernel-based one-class classification methods to find a data-enclosing hypersphere of minimum volume with the concept of learning hierarchical representations in deep learning. Moving from fixed kernels to deep neural networks enables to *jointly* learn representations and optimize the objective on those feature representations. In optimizing the objective, the network learns the factors of variations underlying the data which in the one-class classification setting are the characteristics of the target class. Thus, deep SVDD extends the small set of unsupervised deep anomaly detection techniques with a new objective.

The idea of deep SVDD is simple and the method is "fully deep", that is learning the representation is not an independent step in a hybrid system, but the representation is learned directly on the unsupervised anomaly detection objective. The challenges of high-dimensional data summarized in whats called the curse of dimensionality are approached with the concept of distributed representations which potentially enables an exponential gain in representational expressiveness if the data examined exhibits a factorized structure, that is the factors of variation underlying the data have a hierarchical structure. As deep SVDD can be trained with (mini-batch) SGD, its complexity is linear in the number of batches and since the objective is composed of a regularizer and an empirical error term, each batch-update can be processed in parallel, that is special hardware such as GPUs and fast network connectivity can be utilized for parallel distributed processing. This enables applications of large-scale. Compared to kernel-based one-class classification, the deep SVDD model has lower memory complexity in the sense that the model is conserved in the network parameters whereas kernel methods require the support vectors for prediction. In conjunction with SGD optimization, this also enables online learning from data streams. Like in the kernel-based one-class classification methods, the ν-parameter in deep SVDD allows to control the fraction of outliers in the training set a priori. Aspects to consider for avoiding trivial solutions when applying deep SVDD have been stated, which are cautious initialization of the hypersphere center (sufficiently far away from the all-zero-weights network solution) and avoiding bias terms as well as bounded activation functions in the network architectures. In the experiments carried

out we saw that deep SVDD works and gives competitive results which are qualitatively sound. Moreover, the generalization performance of deep SVDD seems to be good even without the use of additional regularization. However, our experiments also revealed that there is room for improvement in optimizing deep SVDD, which generally is a non-convex problem to solve. Especially the optimization of the soft-boundary objective needs further investigation in how it can be optimized more efficiently.

Outlook

As was indicated, a first opportunity for improving deep SVDD lies in a thorough investigation of its optimization. This includes studying the relationship of the radius parameter R to the weights of the network in greater detail — also with respect to the hyperparameters $\lambda > 0$ and $\nu \in (0, 1]$ employed in the objective. To achieve progress in this direction, a theoretical study of the objective's solution path could be undertaken. Moreover, trying and examining different strategies of weight initialization and pre-training also could help the network in finding better configurations. Experiments with greedy layer-wise pre-training, network sparsity regularization or the initialization from weights pre-trained by a deep autoencoder could be insightful. Considering experiments, observing the performance for different choices of ν when anomalies are mixed into the training set in different fractions (e.g. 1%, 5%, 10%, 20%, etc.) would be interesting. An extension of deep SVDD to the semi-supervised setting analogously to kernel SVDD, where known anomalies are incorporated into training such that they are penalized for being located *within* the hypersphere, is possible and could be studied. Considering the deep networks employed, the model could be applied to different kinds of architectures (e.g. dynamic architectures like RNNs).

In general, we expect deep SVDD to benefit from the vivid advances in deep learning research, that is from improvements in architectures, optimization, and generalization. Deep learning provides untapped potential for anomaly detection and approaches are yet sparse which leaves much space for further exploration.

A Convex Optimization

Definition A.1 (Convex optimization problem, Boyd and Vandenberghe (2004))**.** A *convex optimization problem* with $m \in \mathbb{N}$ inequality and $p \in \mathbb{N}$ equality constraints is of the form

$$\begin{aligned} \min_{w \in \mathcal{C}} \quad & f_0(w) \\ \text{s.t.} \quad & f_i(w) \leq 0, \quad i = 1, \ldots, m, \\ & h_j(w) = 0, \quad j = 1, \ldots, p, \end{aligned} \tag{A.1}$$

where $\mathcal{C} \subseteq \mathbb{R}^d$ is a convex set, $f_0, f_1, \ldots, f_m : \mathcal{C} \to \mathbb{R}$ are convex functions, and $h_1, \ldots, h_p : \mathcal{C} \to \mathbb{R}$ are affine linear functions.

The convex optimization problem as given in (A.1) has no general analytic solution, but there exist very effective numerical optimization methods for solving it. For example interior-point methods, methods based on the subgradient-projection method by Polyak (1969) or limited-memory quasi-Newton methods (see e.g. Nocedal and Wright, 2006). Often, simpler constraints or special forms of the objective function f_0 can be exploited to construct more efficient solvers (e.g. for quadratic programs, where the objective is quadratic and the constraints are linear).

Theorem A.2 (Strong duality, Boyd and Vandenberghe (2004))**.** *If the primal problem is a convex optimization problem as in* (A.1) *and either has exclusively linear constraints or satisfies some constraint qualifications such as Slater's condition (Theorem A.4), and the Lagrange dual problem is defined as*

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} \; g(\boldsymbol{\alpha}, \boldsymbol{\beta}) \tag{A.2}$$

with Lagrange dual *objective function*

$$g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \inf_{w \in \mathcal{C}} \; \mathcal{L}(w, \boldsymbol{\alpha}, \boldsymbol{\beta}), \tag{A.3}$$

where the Lagrangian $\mathcal{L} : \mathcal{C} \times [0, \infty)^m \times \mathbb{R}^p \to \mathbb{R}$ *is given by*

$$\mathcal{L}(w, \boldsymbol{\alpha}, \boldsymbol{\beta}) = f_0(w) + \sum_{i=1}^{m} \alpha_i f_i(w) + \sum_{j=1}^{p} \beta_j h_j(w), \tag{A.4}$$

then strong duality *holds. This means, that the duality gap is zero, i.e. the optimum of the dual coincides with the optimum of the primal:*

$$\max_{\boldsymbol{\alpha} \geq 0, \boldsymbol{\beta}} g(\boldsymbol{\alpha}, \boldsymbol{\beta}) = p^*, \tag{A.5}$$

where p^* denotes the optimal value of problem (A.1)*.*

In the optimum, the following necessary conditions must be satisfied, which generally must hold for non-linear optimization problems subject to inequality and equality constraints.

Theorem A.3 (Karush-Kuhn-Tucker (KKT) conditions)**.** *Let w^* be the optimum corresponding to the optimal value p^* of problem* (A.1)*. If the convex optimization problem* (A.1) *satisfies some (mild) regularity conditions (or constraint qualifications), such as the linearity constraint qualification or Slater's condition (Theorem A.4), then the optimum w^* and Lagrange multipliers $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ must satisfy the following conditions, called the* Karush-Kuhn-Tucker (KKT) conditions*:*

Primal feasibility

$$\begin{aligned} f_i(w^*) &\leq 0, \quad \textit{for } i = 1, \ldots, m, \\ h_j(w^*) &= 0, \quad \textit{for } j = 1, \ldots, p. \end{aligned} \tag{A.6}$$

Dual feasibility

$$\alpha_i \leq 0, \quad \textit{for } i = 1, \ldots, m. \tag{A.7}$$

Complementary slackness

$$\alpha_i f_i(w^*) = 0, \quad \textit{for } i = 1, \ldots, m. \tag{A.8}$$

One constraint qualification often fulfilled in the more general context of kernel methods and support vector machine problems is Slater's condition:

Theorem A.4 (Slater's condition, Boyd and Vandenberghe (2004))**.** *For any function f_i and any set S, let $dom(f_i)$ be the domain of f_i and define the* relative interior *of S by*

$$\mathit{relint}(S) \equiv \{w \in S \,|\, \exists\, r > 0, B_r(w) \cap \mathit{aff}(S) \subset S\}, \tag{A.9}$$

where $B_r(w)$ is a ball centered at w with radius r and $\mathit{aff}(S)$ is the affine hull of S. Consider problem (A.1)*, if there exists w such that*

$$\begin{aligned} w &\in \mathit{relint}\left(\bigcap_{i=0}^{m} dom(f_i)\right), \\ f_i(w) &< 0, \quad i = 1, \ldots, m, \\ h_j(w) &= 0, \quad j = 1, \ldots, p, \end{aligned} \tag{A.10}$$

then strong duality for (A.1) *holds.*

More informally, Slater's condition says that the feasible region must include an interior point and the inequality constraints must be strict.

B Kernology

Definition B.1 (Mercer's condition)**.** Let $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a kernel function with $\mathcal{X} \subset \mathbb{R}^d$. Then, kernel k fulfills *Mercer's condition* if the kernel matrix K defined by k is positive semi-definite, that is for any $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n \in \mathcal{X}$ and any $\boldsymbol{\alpha} \in \mathbb{R}^n$ we have that

$$\boldsymbol{\alpha}^T K \boldsymbol{\alpha} = \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j k(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0, \tag{B.1}$$

with kernel matrix $K = (k(\boldsymbol{x}_i, \boldsymbol{x}_j))_{i,j=1,\ldots,n}$. In this case, k is also called *Mercer kernel* or simply a positive semi-definite kernel.

Theorem B.2 (Representer theorem)**.** *If $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a Mercer kernel on $\mathcal{X} \subset \mathbb{R}^d$, then there exists a (possibly infinite-dimensional) Hilbert space $\mathcal{F}$, the so-called* Reproducing Kernel Hilbert Space (RKHS) *or simply* feature space*, and a continuous mapping $\phi : \mathcal{X} \to \mathcal{F}$, the so-called* feature map*, such that*

$$k(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \langle \phi(\boldsymbol{x}), \phi(\tilde{\boldsymbol{x}}) \rangle_{\mathcal{F}} \quad \textit{for all } \boldsymbol{x}, \tilde{\boldsymbol{x}} \in \mathcal{X}. \tag{B.2}$$

That is kernel k implicitly allows to compute the scalar products of the feature representations $\phi(\boldsymbol{x})$ in feature $\mathcal{F}$.

Definition B.3 (Common kernel functions)**.**

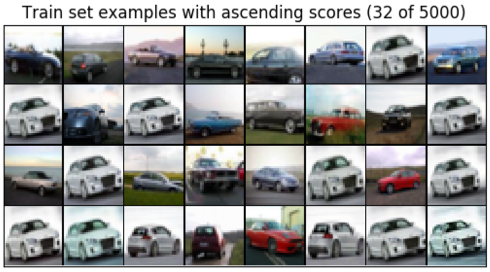
Linear Kernel	$k(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{x}, \boldsymbol{y} \rangle + c, \quad c \in \mathbb{R}.$	(B.3)
Polynomial Kernel	$k(\boldsymbol{x}, \boldsymbol{y}) = (\alpha \langle \boldsymbol{x}, \boldsymbol{y} \rangle + c)^d, \quad \alpha, c > 0, d \in \mathbb{N}.$	(B.4)
Gaussian Kernel	$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|^2}{2\sigma^2}\right), \quad \sigma > 0.$	(B.5)
(or RBF Kernel)	$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\gamma \|\boldsymbol{x} - \boldsymbol{y}\|^2\right), \quad \gamma > 0.$	(B.6)
Exponential Kernel	$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|}{2\sigma^2}\right), \quad \sigma > 0.$	(B.7)
(or Laplacian Kernel)	$k(\boldsymbol{x}, \boldsymbol{y}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{y}\|}{\sigma}\right), \quad \sigma > 0.$	(B.8)

C Supplementary Material of Experiments

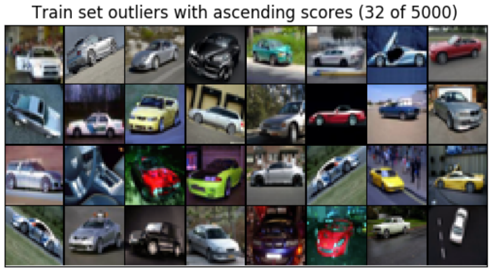
Train set examples with ascending scores (32 of 5000)

Train set outliers with ascending scores (32 of 5000)

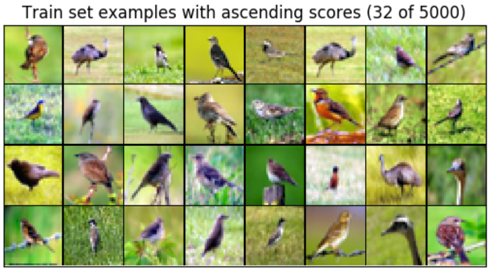
Train set examples with ascending scores (32 of 5000)



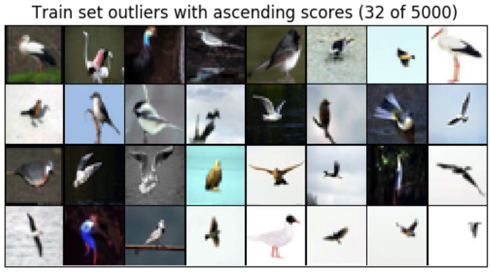
Train set outliers with ascending scores (32 of 5000)



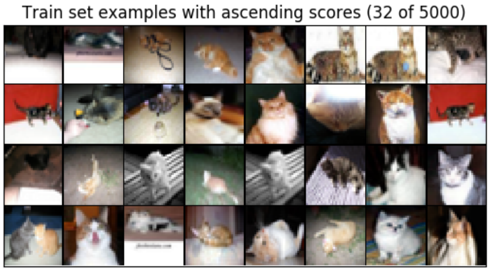
Train set examples with ascending scores (32 of 5000)



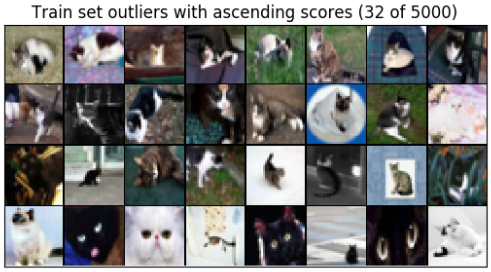
Train set outliers with ascending scores (32 of 5000)



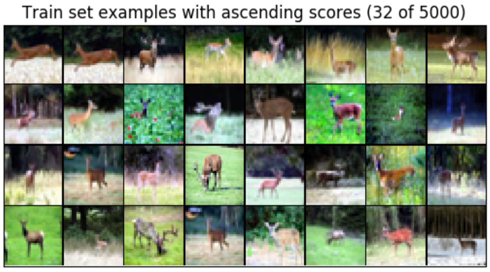
Train set examples with ascending scores (32 of 5000)



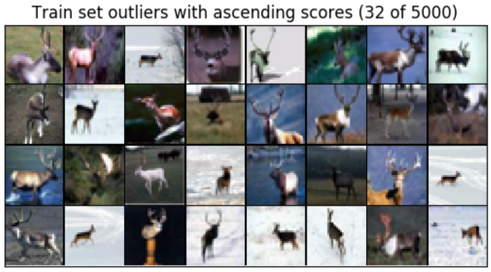
Train set outliers with ascending scores (32 of 5000)



Train set examples with ascending scores (32 of 5000)



Train set outliers with ascending scores (32 of 5000)



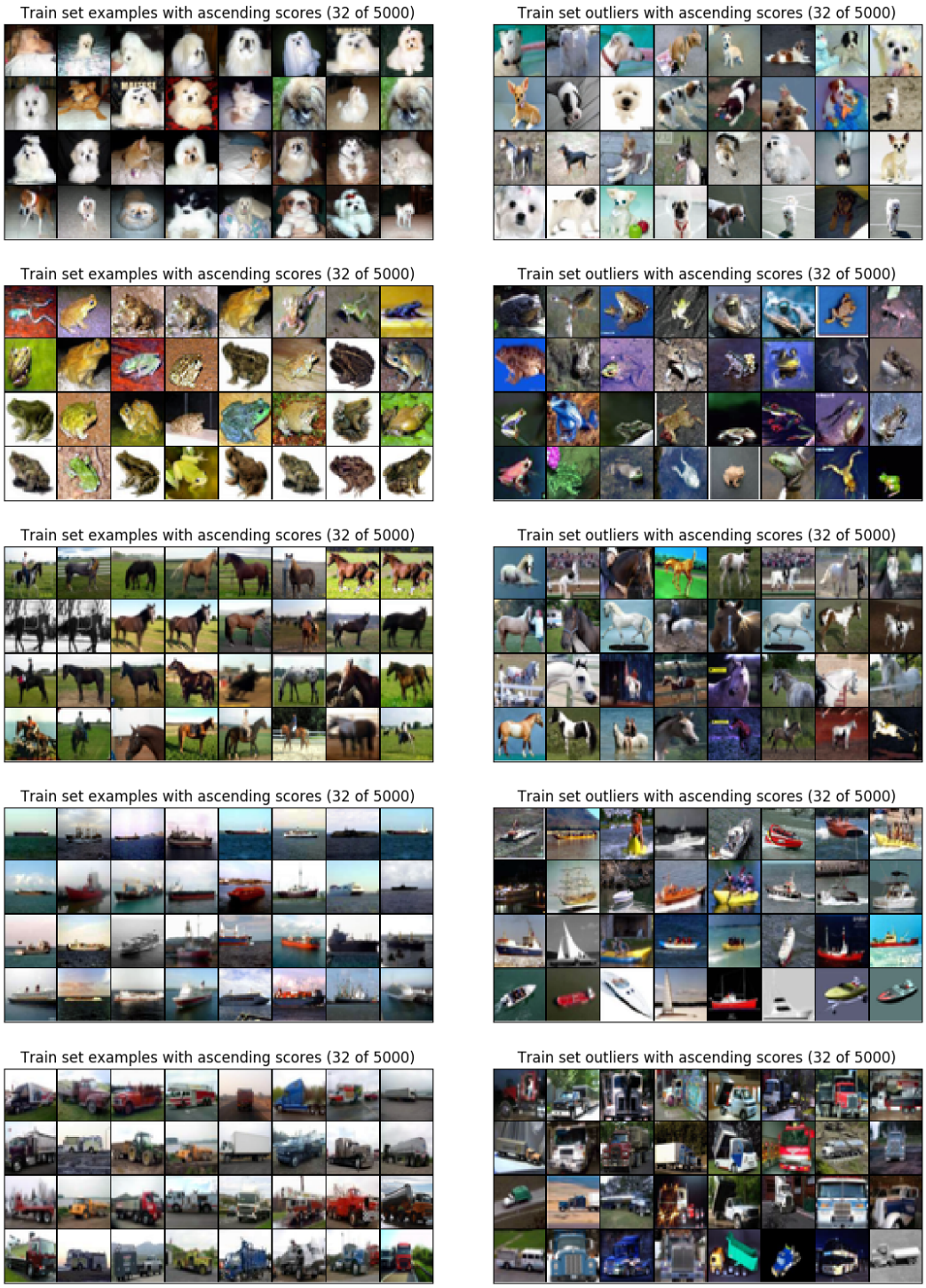


Figure C.1: Examples of the CIFAR-10 one-class training sets with the lowest (top left) and highest (bottom right) KDE anomaly scores in each experiment.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from `https://www.tensorflow.org/`.

Aggarwal, C. C. (2013). *Outlier analysis*, Springer New York.

Akaho, S. (2001). A kernel method for canonical correlation analysis, *Proceedings of the International Meeting of the Psychometric Society*, Citeseer.

An, J. and Cho, S. (2015). Variational autoencoder based anomaly detection using reconstruction probability, *Technical report*, Technical Report.

Andrews, J. T., Morton, E. J. and Griffin, L. D. (2016). Detecting anomalous data using auto-encoders, *International Journal of Machine Learning and Computing* **6**(1): 21.

Banerjee, A., Burlina, P. and Meth, R. (2007). Fast hyperspectral anomaly detection via svdd, *Proceedings of IEEE International Conference on Image Processing*, Vol. 4, IEEE, pp. IV–101.

Bartlett, P. L. and Mendelson, S. (2002). Rademacher and gaussian complexities: Risk bounds and structural results, *Journal of Machine Learning Research* **3**(Nov): 463–482.

Bellman, R. (1961). Adaptive control processes: a guided tour.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures, *Neural networks: Tricks of the trade*, Springer, pp. 437–478.

Bengio, Y., Courville, A. and Vincent, P. (2013). Representation learning: A review and new perspectives, *IEEE transactions on pattern analysis and machine intelligence* **35**(8): 1798–1828.

Bengio, Y. and Delalleau, O. (2011). On the expressive power of deep architectures, *Algorithmic Learning Theory*, Springer, pp. 18–36.

Bengio, Y., Delalleau, O. and Roux, N. L. (2006). The curse of highly variable functions for local kernel machines, *Advances in neural information processing systems*, pp. 107–114.

Bengio, Y., Lamblin, P., Popovici, D. and Larochelle, H. (2007). Greedy layer-wise training of deep networks, *Proceedings Advances in Neural Information Processing Systems*, pp. 153–160.

Bengio, Y., LeCun, Y. et al. (2007). Scaling learning algorithms towards ai, *Large-scale kernel machines* **34**(5): 1–41.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization, *Journal of Machine Learning Research* **13**(Feb): 281–305.

Berrada, L., Zisserman, A. and Kumar, M. P. (2016). Trusting svm for piecewise linear cnns, *arXiv preprint arXiv:1611.02185* .

Beyer, K., Goldstein, J., Ramakrishnan, R. and Shaft, U. (1999). When is "nearest neighbor" meaningful?, *Proceedings International conference on database theory*, Springer, pp. 217–235.

Bottou, L. (1998). Online learning and stochastic approximations, *Online learning in neural networks* **17**(9): 142.

Bousquet, O. and Bottou, L. (2008). The tradeoffs of large scale learning, *Advances in neural information processing systems*, pp. 161–168.

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*, Cambridge University Press.

Breiman, L. (1996). Bagging predictors, *Machine learning* **24**(2): 123–140.

Chandola, V., Banerjee, A. and Kumar, V. (2009). Anomaly detection: a survey, *ACM Computing Surveys* **41**(3): 1–58.

Chang, C.-C. and Lin, C.-J. (2011). LIBSVM: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology* **2**: 27:1–27:27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Chang, W.-C., Lee, C.-P. and Lin, C.-J. (2013). A revisit to support vector data description (svdd), *Technical report*, National Taiwan University, Taipei, Taiwan.

Chen, J., Sathe, S., Aggarwal, C. and Turaga, D. (2017). Outlier detection with autoencoder ensembles, *Proceedings of the 2017 SIAM International Conference on Data Mining*, SIAM, pp. 90–98.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B. and LeCun, Y. (2015). The loss surfaces of multilayer networks, *Artificial Intelligence and Statistics*, pp. 192–204.

Collobert, R., Kavukcuoglu, K. and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning, *BigLearn, NIPS Workshop*. Software available from `http://torch.ch/`.

Collobert, R. and Weston, J. (2008). A unified architecture for natural language processing: Deep neural networks with multitask learning, *Proceedings of the 25th international conference on Machine learning*, ACM, pp. 160–167.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011). Natural language processing (almost) from scratch, *Journal of Machine Learning Research* **12**(Aug): 2493–2537.

Csáji, B. C. (2001). Approximation with artificial neural networks, *Faculty of Sciences, Etvs Lornd University, Hungary* **24**: 48.

Cui, X., Goel, V. and Kingsbury, B. (2015). Data augmentation for deep neural network acoustic modeling, *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* **23**(9): 1469–1477.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function, *Mathematics of Control, Signals, and Systems (MCSS)* **2**(4): 303–314.

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S. and Bengio, Y. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization, *Advances in neural information processing systems*, pp. 2933–2941.

Dieleman, S., Schlüter, J., Raffel, C., Olson, E., Sønderby, S. K., Nouri, D. et al. (2015). Lasagne: First release. Software available at `http://dx.doi.org/10.5281/zenodo.27878`.

Duchi, J., Hazan, E. and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization, *Journal of Machine Learning Research* **12**(Jul): 2121–2159.

Emmott, A., Das, S., Dietterich, T., Fern, A. and Wong, W.-K. (2016). Anomaly detection meta-analysis benchmarks.

Erfani, S. M., Rajasegarar, S., Karunasekera, S. and Leckie, C. (2016). High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning, *Pattern Recognition* **58**: 121–134.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P. and Bengio, S. (2010). Why does unsupervised pre-training help deep learning?, *Journal of Machine Learning Research* **11**(Feb): 625–660.

Farabet, C., Couprie, C., Najman, L. and LeCun, Y. (2013). Learning hierarchical features for scene labeling, *IEEE transactions on pattern analysis and machine intelligence* **35**(8): 1915–1929.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pp. 249–256.

Glorot, X., Bordes, A. and Bengio, Y. (2011). Deep sparse rectifier neural networks, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 315–323.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*, MIT Press. `http://www.deeplearningbook.org`.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. (2014). Generative adversarial nets, *Proceedings Advances in Neural Information Processing Systems*, pp. 2672–2680.

Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X. and Wang, G. (2015). Recent advances in convolutional neural networks, *arXiv preprint arXiv:1512.07108* .

Gubbi, J., Buyya, R., Marusic, S. and Palaniswami, M. (2013). Internet of things (iot): A vision, architectural elements, and future directions, *Future generation computer systems* **29**(7): 1645–1660.

Hanley, J. A. and McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (roc) curve., *Radiology* **143**(1): 29–36.

Hastie, T., Rosset, S., Tibshirani, R. and Zhu, J. (2004). The entire regularization path for the support vector machine, *Journal of Machine Learning Research* **5**(Oct): 1391–1415.

Hawkins, D. M. (1980). *Identification of outliers*, Vol. 11, Springer.

Hawkins, S., He, H., Williams, G. and Baxter, R. (2002). Outlier detection using replicator neural networks, *DaWaK*, Vol. 2454, Springer, pp. 170–180.

He, K., Zhang, X., Ren, S. and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034.

He, K., Zhang, X., Ren, S. and Sun, J. (2016). Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*.

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N. et al. (2012). Deep neural networks for acoustic modeling in speech recognition, *IEEE Signal Processing Magazine* **29**(6): 82–97.

Hinton, G. E., Osindero, S. and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets, *Neural computation* **18**(7): 1527–1554.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks, *science* **313**(5786): 504–507.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen, *Diploma, Technische Universität München* **91**.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory, *Neural computation* **9**(8): 1735–1780.

Hornik, K., Stinchcombe, M. and White, H. (1989). Multilayer feedforward networks are universal approximators, *Neural networks* **2**(5): 359–366.

Huang, F. J. and LeCun, Y. (2006). Large-scale learning with svm and convolutional for generic object categorization, *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, Vol. 1, IEEE, pp. 284–291.

Jean, S., Cho, K., Memisevic, R. and Bengio, Y. (2015). On using very large target vocabulary for neural machine translation, *arXiv preprint arXiv:1412.2007* .

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N. and Wu, Y. (2016). Exploring the limits of language modeling, *arXiv preprint arXiv:1602.02410* .

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980* .

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes, *arXiv preprint arXiv:1312.6114* .

Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images. Available at `https://www.cs.toronto.edu/~kriz/cifar.html`.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks, *Proceedings Advances in neural information processing systems 25*, pp. 1090–1098.

LeCun, Y. A., Bottou, L., Orr, G. B. and Müller, K.-R. (2012). Efficient backprop, *Neural networks: Tricks of the trade*, Springer, pp. 9–48.

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning, *Nature* **521**(7553): 436–444.

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E. and Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network, *Advances in neural information processing systems*, pp. 396–404.

LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11): 2278–2324.

LeCun, Y., Cortes, C. and Burges, C. J. (2010). MNIST handwritten digit database, *AT&T Labs* **2**. Available at `http://yann.lecun.com/exdb/mnist`.

Lee, G. and Scott, C. D. (2007). The one class support vector machine solution path, *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Vol. 2, IEEE, pp. II–521–II–524.

Lee, H., Battle, A., Raina, R. and Ng, A. Y. (2007). Efficient sparse coding algorithms, *Proceedings Advances in neural information processing systems*, pp. 801–808.

Lee, H., Ekanadham, C. and Ng, A. Y. (2008). Sparse deep belief net model for visual area v2, *Proceedings Advances in neural information processing systems*, pp. 873–880.

Lee, H., Grosse, R., Ranganath, R. and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations, *Proceedings of the 26th annual international conference on machine learning*, ACM, pp. 609–616.

Lee, S.-W., Park, J. and Lee, S.-W. (2006). Low resolution face recognition based on support vector data description, *Pattern Recognition* **39**(9): 1809–1812.

Liu, D., Qian, H., Dai, G. and Zhang, Z. (2013). An iterative svm approach to feature selection and classification in high-dimensional datasets, *Pattern Recognition* **46**(9): 2531–2537.

Lowe, D. G. (1999). Object recognition from local scale-invariant features, *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, Vol. 2, IEEE, pp. 1150–1157.

Maas, A. L., Hannun, A. Y. and Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models, *Proceedings ICML*, Vol. 30.

Mairal, J., Bach, F., Ponce, J. and Sapiro, G. (2009). Online dictionary learning for sparse coding, *Proceedings of the 26th annual international conference on machine learning*, ACM, pp. 689–696.

Makhzani, A. and Frey, B. (2013). K-sparse autoencoders, *arXiv preprint arXiv:1312.5663* .

Makhzani, A. and Frey, B. J. (2015). Winner-take-all autoencoders, *Proceedings Advances in Neural Information Processing Systems*, pp. 2791–2799.

Masci, J., Meier, U., Cireșan, D. and Schmidhuber, J. (2011). Stacked convolutional auto-encoders for hierarchical feature extraction, *Artificial Neural Networks and Machine Learning–ICANN 2011* pp. 52–59.

Mikolov, T., Deoras, A., Povey, D., Burget, L. and Černockỳ, J. (2011). Strategies for training large scale neural network language models, *Proceedings Automatic Speech Recognition and Understanding (ASRU)*, IEEE, pp. 196–201.

Montavon, G., Orr, G. B. and Müller, K.-R. (eds) (2012). *Neural Networks: Tricks of the Trade, Reloaded*, Vol. 7700 of *Lecture Notes in Computer Science (LNCS)*, 2nd edn edn, Springer. URL: `http://www.springer.com/computer/theoretical+computer+science/book/978-3-642-35288-1`.

Montúfar, G. F. and Morton, J. (2015). When does a mixture of products contain a product of mixtures?, *SIAM Journal on Discrete Mathematics* **29**(1): 321–347.

Montufar, G. F., Pascanu, R., Cho, K. and Bengio, Y. (2014). On the number of linear regions of deep neural networks, *Advances in neural information processing systems*, pp. 2924–2932.

Moya, M. M., Koch, M. W. and Hostetler, L. D. (1993). One-class classifier networks for target recognition applications, *Proceedings World Congress on Neural Networks*, International Neural Network Society, Portland, OR, pp. 797–801.

Mozer, M. C. (1989). A focused back-propagation algorithm for temporal pattern recognition, *Complex systems* **3**(4): 349–381.

Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K. and Schölkopf, B. (2001). An introduction to kernel-based learning algorithms, *IEEE transactions on neural networks* **12**(2): 181–201.

Munoz, A. and Moguerza, J. M. (2004). One-class support vector machines and density estimation: The precise relation, *CIARP*, Springer, pp. 216–223.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*, MIT press.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$, *Soviet Mathematics Doklady*, Vol. 27, pp. 372–376.

Ng, A. (2011). Sparse autoencoder, *CS294A Lecture notes* **72**(2011): 1–19.

Nocedal, J. and Wright, S. J. (2006). *Numerical optimization*, Springer.

Pal, M. and Foody, G. M. (2010). Feature selection for classification of hyperspectral data by svm, *IEEE Transactions on Geoscience and Remote Sensing* **48**(5): 2297–2307.

Park, J., Kang, D., Kim, J., Kwok, J. T. and Tsang, I. W. (2007). Svdd-based pattern denoising, *Neural computation* **19**(7): 1919–1938.

Parzen, E. (1962). On estimation of a probability density function and mode, *The annals of mathematical statistics* **33**(3): 1065–1076.

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

Pimentel, M. A., Clifton, D. A., Clifton, L. and Tarassenko, L. (2014). A review of novelty detection, *Signal Processing* **99**: 215–249.

Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods, *USSR Computational Mathematics and Mathematical Physics* **4**(5): 1–17.

Polyak, B. T. (1969). Minimization of unsmooth functionals, *USSR Computational Mathematics and Mathematical Physics* **9**(3): 14–29.

Radford, A., Metz, L. and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks, *arXiv preprint arXiv:1511.06434* .

Raina, R., Madhavan, A. and Ng, A. Y. (2009). Large-scale deep unsupervised learning using graphics processors, *Proceedings of the 26th annual international conference on machine learning*, ACM, pp. 873–880.

Rall, L. B. (1981). Automatic differentiation: Techniques and applications.

Rätsch, G., Schölkopf, B., Mika, S. and Müller, K.-R. (2000). *SVM and boosting: One class*, GMD-Forschungszentrum Informationstechnik.

Ribeiro, M., Lazzaretti, A. E. and Lopes, H. S. (2017). A study of deep convolutional auto-encoders for anomaly detection in videos, *Pattern Recognition Letters* .

Richter, C. and Roy, N. (2017). Safe visual navigation via deep learning and novelty detection, *Proc. of the Robotics: Science and Systems Conference*.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological review* **65**(6): 386.

Rosenblatt, M. et al. (1956). Remarks on some nonparametric estimates of a density function, *The Annals of Mathematical Statistics* **27**(3): 832–837.

Rumelhart, D. E., Hinton, G. E., Williams, R. J. et al. (1988). Learning representations by back-propagating errors, *Cognitive modeling* **5**(3): 1.

Sabokrou, M., Fayyaz, M., Fathy, M. et al. (2016). Fully convolutional neural network for fast anomaly detection in crowded scenes, *arXiv preprint arXiv:1609.00866* .

Sainath, T., Mohamed, A.-R., Kingsbury, B. and Ramabhadran, B. (2013). Deep convolutional neural networks for lvcsr, *Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp. 8614–8618.

Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines, *Proceedings International Conference on Artificial Intelligence and Statistics*, pp. 448–455.

Schaul, T., Antonoglou, I. and Silver, D. (2013). Unit tests for stochastic optimization, *arXiv preprint arXiv:1312.6055* .

Scherer, D., Müller, A. and Behnke, S. (2010). Evaluation of pooling operations in convolutional architectures for object recognition, *Artificial Neural Networks–ICANN 2010* pp. 92–101.

Schlegl, T., Seeböck, P., Waldstein, S. M., Schmidt-Erfurth, U. and Langs, G. (2017). Unsupervised anomaly detection with generative adversarial networks to guide marker discovery, *Proceedings International Conference on Information Processing in Medical Imaging*, Springer, pp. 146–157.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview, *Neural networks* **61**: 85–117.

Schölkopf, B., Burges, C. and Smola, A. J. (1999). *Advances in kernel methods: support vector learning*, MIT press.

Schölkopf, B., Burges, C. and Vapnik, V. (1995). Extracting support data for a given task, *Proceedings, First International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, CA, pp. 252–257.

Schölkopf, B., Platt, J. C., Shawe-Taylor, J., Smola, A. J. and Williamson, R. C. (2001). Estimating the support of a high-dimensional distribution, *Neural computation* **13**(7): 1443–1471.

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*, MIT press.

Schölkopf, B., Smola, A. J., Williamson, R. C. and Bartlett, P. L. (2000). New support vector algorithms, *Neural computation* **12**(5): 1207–1245.

Schölkopf, B., Smola, A. and Müller, K.-R. (1997). Kernel principal component analysis, *Artificial Neural Networks—ICANN'97*, Springer Berlin Heidelberg, pp. 583–588.

Scott, C. D. and Nowak, R. D. (2006). Learning minimum volume sets, *Journal of Machine Learning Research* **7**(Apr): 665–704.

Seeböck, P., Waldstein, S., Klimscha, S., Gerendas, B. S., Donner, R., Schlegl, T., Schmidt-Erfurth, U. and Langs, G. (2016). Identifying and categorizing anomalies in retinal imaging data, *arXiv preprint arXiv:1612.00686* .

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556* .

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting, *Journal of machine learning research* **15**(1): 1929–1958.

Steinwart, I., Hush, D. and Scovel, C. (2005). A classification framework for anomaly detection, *Journal of Machine Learning Research* **6**(Feb): 211–232.

Sutskever, I., Martens, J., Dahl, G. and Hinton, G. (2013). On the importance of initialization and momentum in deep learning, *International conference on machine learning*, pp. 1139–1147.

Sutskever, I., Vinyals, O. and Le, Q. V. (2014). Sequence to sequence learning with neural networks, *Advances in neural information processing systems*, pp. 3104–3112.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A. (2015). Going deeper with convolutions, *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1–9.

Taigman, Y., Yang, M., Ranzato, M. and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification, *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708.

Tax, D. M. and Duin, R. P. (2004). Support vector data description, *Machine learning* **54**(1): 45–66.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions, *arXiv e-prints* **abs/1605.02688**. URL: `http://arxiv.org/abs/1605.02688`.

Tieleman, T. (2008). Training restricted boltzmann machines using approximations to the likelihood gradient, *Proceedings of the 25th international conference on Machine learning*, ACM, pp. 1064–1071.

Tieleman, T. and Hinton, G. (2012). Lecture 6.5-RMSProp: Divide the gradient by a running average of its recent magnitude, *COURSERA: Neural networks for machine learning* **4**(2): 26–31.

Tsybakov, A. B. (1997). On nonparametric estimation of density level sets, *The Annals of Statistics* **25**(3): 948–969.

Vapnik, V. (1998). *Statistical learning theory*, Vol. 1, Wiley.

Vempati, S., Vedaldi, A., Zisserman, A. and Jawahar, C. (2010). Generalized rbf feature maps for efficient detection, *21st British Machine Vision Conference*, pp. 1–11.

Vert, R. and Vert, J.-P. (2006). Consistency and convergence rates of one-class svms and related algorithms, *Journal of Machine Learning Research* **7**(May): 817–854.

Vincent, P., Larochelle, H., Bengio, Y. and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders, *Proceedings of the 25th International Conference on Machine Learning*, ACM, pp. 1096–1103.

Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y. and Manzagol, P.-A. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *Journal of Machine Learning Research* **11**(Dec): 3371–3408.

Werbos, P. J. (1988). Generalization of backpropagation with application to a recurrent gas market model, *Neural networks* **1**(4): 339–356.

Widrow, B. and Hoff, M. E. (1960). Adaptive switching circuits, *Technical report*, Stanford electronics labs, Stanford University, CA.

Xu, D., Ricci, E., Yan, Y., Song, J. and Sebe, N. (2015). Learning deep representations of appearance and motion for anomalous event detection, *arXiv preprint arXiv:1510.01553* .

Yu, F., Seff, A., Zhang, Y., Song, S., Funkhouser, T. and Xiao, J. (2015). LSUN: Construction of a large-scale image dataset using deep learning with humans in the loop, *arXiv preprint arXiv:1506.03365* .

Yuille, A. L. and Rangarajan, A. (2002). The concave-convex procedure (cccp), *Advances in neural information processing systems*, pp. 1033–1040.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method, *arXiv preprint arXiv:1212.5701* .

Zhang, C., Bengio, S., Hardt, M., Recht, B. and Vinyals, O. (2016). Understanding deep learning requires rethinking generalization, *arXiv preprint arXiv:1611.03530* .

Zhang, Y., Yu, F., Song, S., Xu, P., Seff, A. and Xiao, J. (2015). Large-scale scene understanding challenge, *CVPR Workshop*. Available at `http://lsun.cs.princeton.edu`.

Zimek, A., Schubert, E. and Kriegel, H.-P. (2012). A survey on unsupervised outlier detection in high-dimensional numerical data, *Statistical Analysis and Data Mining: The ASA Data Science Journal* **5**(5): 363–387.