



HOCHSCHULE MÜNCHEN

FAKULTÄT 07 - FAKULTÄT FÜR INFORMATIK UND MATHEMATIK

MASTERARBEIT ZUM THEMA:

**Verbesserung einer modernen,
Semi-Supervised
Anomalieerkennungsmethode, durch die
Zugabe von synthetisch hergestellten
Datenpunkten**

Improvement of a modern semi-supervised anomaly detection method by adding
synthetically generated data points

VORGELEGT VON:

Philipp Tomac
Einsteinstraße 121
81675 München
E-Mail: tomacph@gmail.com
Matrikelnummer: 04634519
Fachsemester 4
Studiengang: Master Informatik - Visual Computing and Machine Learning
Abgabe: 15.09.2021

ERSTPRÜFERIN: FRAU PROF. DR. VOGL

ZWEITPRÜFER: HERR PROF. DR. SPIELER

BETREUER: HERR SCHULZE

Die folgende Erklärung ist in jedes Exemplar der Masterarbeit einzubinden und jeweils persönlich zu unterschreiben.

Tomac, Philipp (Familienname, Vorname)	13.09.2021 (Ort, Datum)
04.01.1993 (Geburtsdatum)	4 (Studiengruppe) / SS 2021 / WS/SS

Erklärung
Gemäß § 40 Abs. 1 i. V. m. § 31 Abs. 7 RaPO

Hiermit erkläre ich, dass ich die Masterarbeit selbstständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

(Unterschrift)

Kurzbeschreibung

Die Semi-Supervised Anomalieerkennung ist ein komplexes Themengebiet, welches mit vollständig bekannten Labels der normalen Datenpunkte und ein paar wenigen, bekannten Klassenlabels der anomalen Samples arbeitet. Bei dieser Art von Anomalieerkennung treten verschiedene Einschränkungen auf. In vielen Fällen sind keine bzw. nur wenige Daten, für das Training der Modelle, vorhanden. Das Sammeln von Daten für Trainingsdatensätze ist kostspielig und benötigt viel Zeit. Hinzu kommt, dass Anomalien nicht an ein bestimmtes Verhalten gebunden sind, sondern verschiedene Formen annehmen können. Dadurch ist es schwierig alle Anomalien mit Hilfe von Algorithmen oder Experten zu finden. Eine weitere Einschränkung ist das vorliegende Ungleichgewicht in den Trainingsdaten. In der Semi-Supervised Anomalieerkennung sind sehr wenige Anomalien bekannt, jedoch viele normale Datenpunkte. Dieses Ungleichgewicht im Datensatz erschwert das Erlernen von anomalen Merkmalen. In dieser Arbeit wird erarbeitet, wie diese Einschränkungen und Probleme gelöst werden können.

Im ersten Teil der Arbeit werden ein Supervised, Unsupervised und Semi-Supervised Adversarial Autoencoder implementiert und durch verschiedene Dateneinteilungen des MNIST-Datensatzes getestet. Dabei wird der latente Raum analysiert, um deren Erkennungsleistung zu überprüfen. Der Unsupervised Adversarial Autoencoder liefert hierbei die besten Ergebnisse, weshalb er für die Generierung von neuen Datenpunkten verwendet wird.

Im zweiten Teil werden in dieser Arbeit verschiedene Datensätze mit Hilfe des Unsupervised Adversarial Autoencoders erstellt und in die Anomalieerkennung A^3 integriert. Mit der Analyse des zweidimensionalen latenten Raums, der durch den Encoder erzeugt wird, können gezielt bestimmte Arten von Datenpunkten generiert werden. Der Bereich, zur Generierung der Samples, im latenten Raum lässt sich mit Unterstützung der Normalverteilung und dessen Parametern bestimmen. Durch das Anreichern der bekannten anomalen Trainingsdaten mit generierten Datenpunkten können die Einschränkungen überwunden und die Leistung der Anomalieerkennung gesteigert werden. Dabei spielt die Anzahl der vorhanden Trainingsdaten eine wichtige Rolle und aus welchem Bereich des latenten Raums die Daten erzeugt werden. Bereits das Austauschen der originalen Anomalien mit generierten Daten führt zu einer Verbesserung der Klassifizierung. Besonders gut eignen sich die generierten Anomalien zum Erkennen von unbekannten Anomalien, also Datenpunkte, die dem Modell während des Trainings vorenthalten wurden.

Keywords— Anomalieerkennung, A^3 , Autoencoder, GAN, Adversarial Autoencoder, Machine Learning, IT security, Deep Learning, Intrusion Detection

Inhaltsverzeichnis

1 Einleitung	1
2 Related Work	2
3 Grundlagen	6
3.1 Anomalieerkennung - Anomaly Detection	6
3.1.1 Schwierigkeiten bei der Anomalieerkennung	8
3.1.2 Supervised Anomaly Detection	9
3.1.3 Unsupervised Anomaly Detection	9
3.1.4 Semi-Supervised Anomaly Detection	9
3.1.5 Ausgabe der Anomalieerkennungsmethoden	10
3.2 Generative Machine Learning Modelle	10
3.2.1 Generatives Lernen	10
3.2.2 Autoencoder	11
3.2.3 Untervollständige Autoencoder	12
3.2.4 Adversarial Autoencoder	15
3.2.4.1 Unsupervised Adversarial Autoencoder	17
3.2.4.2 Supervised Adversarial Autoencoder	17
3.2.4.3 Semi-Supervised Adversarial Autoencoder	19
3.2.5 Generative Adversarial Networks	20
4 Implementierung und Analyse der Adversarial Autoencoder	23
4.1 Grundidee der Arbeit	23
4.2 Explorative Datenanalyse	23
4.2.1 Der Datensatz	23
4.2.2 Datenvorverarbeitung und Datenanalyse	24
4.2.3 Verarbeiten der Daten für Adversarial Autoencoder	25
4.3 Implementierung und Auswertung der Adversarial Autoencoder	27
4.3.1 Implementierung der Modelle	28
4.3.1.1 Aufbau der Adversarial Autoencoder	28
4.3.1.2 Trainingsphase	32
4.3.2 Ergebnisanalyse der Adversarial Autoencoder	36
4.3.2.1 Variablen und Hyperparameter	36
4.3.2.2 Versuchsaufbau	36
4.3.2.3 Experimente mit bekannten Datenpunkten	37
4.3.2.4 Experimente mit bekannten, normalen und unbekannten, anomalen Datenpunkten	40
4.3.2.5 Experimente mit Semi-Supervised Anomalieerkennungscharakter	42
4.3.2.6 Experimente mit vollständigem Datensatz	45
4.3.2.7 Erkenntnisse aus den Experimenten	49

5 Anwendung generierter Datenpunkte in einer Semi-Supervised Anomalieerkennung	51
5.1 Generieren der Datenpunkte	52
5.2 Beschreibung und Auswertung der Experimente	56
5.2.1 Parameter für die Auswertung	57
5.2.2 Experiment 1: Ersetzten der Daten	58
5.2.2.1 Ergebnisse	59
5.2.2.2 Learnings	61
5.2.3 Experiment 2: Hinzufügen von synthetischen Anomalien	62
5.2.3.1 Ergebnisse	62
5.2.3.2 Learnings	64
6 Fazit	65
7 Ausblick auf zukünftige Arbeiten	67
Abkürzungsverzeichnis	68
Abbildungsverzeichnis	69

Vorwort und Danksagung

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum bei personenbezogenen Substantiven und Pronomen verwendet. Dies impliziert jedoch keine Benachteiligung anderer Geschlechteridentitäten, die mit eingeschlossen sind, sondern soll im Sinne der sprachlichen Vereinfachung als geschlechtsneutral verstanden werden.

Diese Masterarbeit wurde selbstständig, in dem Zeitraum vom 16.03.2021 bis zum 15.09.2021, implementiert und geschrieben.

An dieser Stelle möchte ich mich bei allen Personen bedanken, die mich während der Fertigstellung dieser Masterarbeit motiviert und unterstützt haben.

Besonderer Dank gilt Frau Prof. Dr. Vogl und Herrn Schulze, die meine Arbeit betreuten. Für die hilfreichen Anregungen und die konstruktive Kritik möchte ich mich herzlich bedanken.

Außerdem möchte ich Caroline Drexel, Frederike Bernhöft, Tobias Oberpaul, Christine und Sebastian Pittrich für das Korrekturlesen meiner Masterarbeit danken.

1 Einleitung

Die Digitalisierung dominiert unser alltägliches Leben, die Industrie und die sich daraus entwickelnden neuen Arbeitsprozesse. Dabei spielen Daten eine immer wichtigere Rolle. Daten werden bei jedem Schritt in unserem Leben erzeugt und gespeichert egal, ob wir mit dem Fitnesstracker am Handgelenk joggen gehen, mit dem Auto fahren, oder online einkaufen. Diese Datenströme werden über das Internet an verschiedene Anbieter verschickt und durch deren Dienste verarbeitet. Dadurch entstehen Risiken bezüglich der Sicherheit der Privatsphäre von Usern. Dasselbe gilt in der Industrie. In der Produktion liefern Sensoren kontinuierlich Daten an Steuergeräte. Fehlerhafte oder manipulierte Datenpunkte, sogenannten Anomalien, können den Produktionsprozess stören und in besonderen Fällen können diese eine Attacke auf das System auslösen, was zu der Veröffentlichung sensibler Daten führen kann. Dadurch gewinnt in den letzten Jahren das Forschungsgebiet Anomalieerkennung immer mehr an Bedeutung. Die Anomalieerkennung beschäftigt sich mit der Identifikation von Datenpunkten, welche sich in ihrem Verhalten von der Mehrheit der Stichproben unterscheiden. In der Praxis untersuchen automatisierte Algorithmen und in wenigen Fällen Experten manuell Daten nach unbekannten Parametern oder Ausschlägen in den Daten, um Anomalien zu erkennen. Auch die Analyse von Netzwerkdaten, Bilder oder Sensordaten handeln. Da die manuelle Analyse sehr kostspielig und zeitintensiv ist, wird immer öfter versucht, diese mit Hilfe von Algorithmen durchzuführen. Hierfür wurden in der Vergangenheit verschiedene Methoden entwickelt.

Beispielsweise nehmen die Zahlungen durch Kreditkarten stark zu, weshalb es immer wichtiger wird, diesen Vorgang sicher zu gestalten. Es wurden verschiedene Methoden entwickelt, um anomale Daten zu identifizieren und so Sicherheitsrisiken zu minimieren [1] [2] wie z.B.: die Analyse von Netzwerkdaten, um Angriffe von außen, aber auch interne Datenlecks von innen zu identifizieren und zu beseitigen. Auch in der Medizin finden Anomalieerkennungsmethoden neue Anwendungsgebiete. Bei der Analyse von Röntgen- oder MRT-Bildern können mit Hilfe solcher Methoden Anomalien, beispielsweise Tumore, schneller automatisch erkannt werden [3].

Die zu analysierenden Datensätze können in drei verschiedene Kategorien unterteilt werden [3]:

1. **Supervised:** Alle Labels (y) zu den Datenpunkten (x) sind bekannt
2. **Unsupervised:** Kein Label (y) zu den Datenpunkten (x) ist bekannt
3. **Semi-Supervised:** Teilweise sind Labels (y) zu den Datenpunkten (x) bekannt

Supervised bezeichnet Datensätzen bei denen vollständig bekannt ist, bei welchem Datenpunkt es sich um einen normalen bzw. einem anomalen Datenpunkt handelt. Dieses Szenario ist in den wenigsten Fällen relevant, da in der realen Umgebung derlei Art von Datensätze kaum bis gar nicht vorhanden sind [4]. Dies liegt daran, dass die Suche nach allen Anomalien sehr kostspielig ist.

Viele Lösungsansätze gehen davon aus, dass kein Label vorhanden ist und es sich somit

um einen Unsupervised Datensatz handelt. Dies bedeutet, dass es keine Klassen gibt, denen die Datenpunkte zugeordnet werden können. Die Kategorisierung in Normal und Anomal findet durch das neuronale Netz bzw. den Algorithmus statt [5] [6].

Die dritte Kategorie, der Semi-Supervised Fall, spiegelt das realistischste Szenario wider, da durch Expertenwissen bzw. Algorithmen bereits Anomalien gefunden werden können. Hier ist ein Großteil der Labels bekannt. Die normalen Datenpunkte sind vollständig klassifiziert und hinzu kommen außerdem eine begrenzte Anzahl von bekannten Anomalien. Die Algorithmen versuchen anhand der bekannten Daten neue, unbekannte Anomalien zu erkennen [3]. In der Industrie sind Semi-Supervised Datensätze die Regel, da durch Expertenwissen bereits Anomalien identifiziert wurden [7] [8] [9]. In dieser Arbeit wurden verschiedene Variationen des Adversarial Autoencoder (AAE) implementiert und anhand verschiedener Experimente mit dem Modified National Institute of Standards and Technology (MNIST)-Datensatz die Darstellung im latenten Raum betrachtet. Anschließend wurden eigene Datensätze erstellt, bestehend aus Datenpunkten, welche durch die Adversarial Autoencoder (AAE) generiert wurden. Diese generierten Datensätze wurden im letzten Abschnitt der Arbeit verwendet, um die state of the art Methoden der Anomalieerkennung A^3 zu optimieren.

Zusammengefasst werden folgende Punkte in dieser Abschlussarbeit erarbeitet:

1. Analyse des MNIST-Datensatzes
2. Aufbereitung der Grundlagen der Anomalieerkennung und generativer Modelle
3. Implementierung von einem Supervised, Unsupervised und Semi-Supervised AAE
4. Generieren eines neuen Datensatzes bestehend aus zuvor durch die AAE generierten Datenpunkten
5. Einfügen der generierten Daten in einer der technisch aktuellen Methoden, um eine Erkennungsleistung der Anomalieerkennungen zu erzielen

2 Related Work

In diesem Kapitel wird auf die Literatur und die aktuellen Forschungsthemen im Bereich der Anomalieerkennung sowie der Generativen Netzwerke eingegangen. Zunächst wird sich mit den Grundlagen beschäftigt. Ian Goodfellow beschreibt in seinem Buch „Deep Learning“ [10], welches zu den Standardwerken im Bereich Machine Learning zählt, die Theorie hinter Neuronalen Netzen, sowie deren Variationen beispielsweise Rekurrente Neuronale Netze (RNN), Autoencoder (AE) oder auch generative Modelle. Dabei bietet das Buch eine Einführung in alle Bereiche des Deep Learning (DL), sowie mathematisches und konzeptionelles Wissen zu verwendeten Techniken in der Industrie und Forschung. Das Buch bildet die Grundlage für diese Arbeit. Auf der „Conference on Neural Information Processing Systems (NIPS)“ 2016 hielt Ian Goodfellow einen Tutorial-Vortrag über Generative Adversarial Networks (GAN). Das dazugehörige Paper „*NIPS 2016 Tutorial: Generative Adversarial Networks*“ [11] behandelt die Frage,

warum es sich lohnt, GANs zu erforschen und zu verwenden. Generative Netzwerke spiegeln die Funktionsweise des menschlichen Gehirns sehr gut wider, da sie in der Lage sind, sich Dinge „vorzustellen“. Sie kombinieren erlernte Merkmale zu neuen, unbekannten Daten. Das Tutorial beschäftigt sich ebenfalls mit der Funktionsweise von GANs und vergleicht diese mit weiteren, generativen Modellen. Außerdem werden die Grenzen der Forschung thematisiert, sowie die Möglichkeit GANs mit anderen Modellen zu kombinieren. Als Grundlage für die Anomalieerkennung dient das Paper „Anomaly Detection: a Survey“ [3]. In diesem wird eine Übersicht der Forschung im Gebiet der Anomaly Detection (AD) gegeben, indem verschiedene Techniken erläutert werden. Das Paper kategorisiert die beschriebenen Techniken nach den zugrundeliegenden Ansätzen. Dabei wird bei jeder Methodik beschrieben, wie Anomalien erkannt werden können und welche weiteren Varianten der Methodik existieren. Abschließend gehen die Autoren auf die Komplexität der vorgestellten Techniken ein, um deren Anwendbarkeit in einer realen Umgebung zu testen.

Für das Problem der Anomalieerkennung existieren in der Literatur verschiedene Lösungsmethoden. Die Methoden werden in drei verschiedene Ausgangsszenarien eingeteilt: dem Supervised, Unsupervised oder Semi-Supervised Szenario. Während der Literaturrecherche wurde deshalb in diese drei Kategorien untergliedert, was sich in diesem Kapitel widerspiegelt.

In allen drei Varianten ist das Generieren von neuen Daten zur Verbesserung der Anomalieerkennung ein wichtiges Thema. Die Forscher Chaoyue Wang, Chang Xu, Xin Yao und Dacheng Tao verwenden einen Supervised-Datensatz und generieren mit Hilfe eines GANs, in ihrer Arbeit „Evolutionary Generative Adversarial Network“ [4], qualitativ bessere Samples als die des Originaldatensatzes. Die Idee in ihrer Arbeit ist es den Evolutionsalgorithmus und das Generieren von neuen Datenpunkten zu kombinieren. Der evolutionäre Algorithmus besteht aus drei Schritten: Variation, Evolution und Selektion. In der Variation wird der Generator des GANs erzeugt und es werden mehrere Kopien erstellt. Die erstellten Generatoren werden durch die Mutation verändert. Dabei wurden drei mögliche Mutationen implementiert, welche aber nur die Zielfunktionen des Generators betreffen. Jeder Generator wird auch als Kind des ursprünglichen Generators bezeichnet. Während der Evolution, dem nächsten Prozess, wird die Performance der Generatoren mit Hilfe einer Fitnessfunktion evaluiert, welche durch den Diskriminator definiert wird. Im dritten Schritt der Selektion, werden die Generatoren anhand einer Fitnessfunktion bewertet und die besten werden in die neue Population übernommen. Diese Generatoren erzeugen qualitativ hochwertigere Samples als ihre Kontrahenten, da sie den Diskriminator besser täuschen können. Der Durchlauf dieser drei Phasen wird als Evolutionsschritt bezeichnet. Mit dieser Methode konnten die Forscher ein stabileres Training des GANs, sowie eine bessere generative Leistung von Bildern erreichen [4].

Das Verwenden von verschiedenen Generatoren für das Erstellen von Samples wird sich hier in der Arbeit als wichtig erweisen, da durch synthetisch hergestellte Anomalien die Klassifizierung der Anomalieerkennung verbessert werden soll. Außerdem kann durch

mehrere Generatoren eine größere Vielfalt in den Samples erreicht werden, was einen positiven Einfluss auf die Anomalieerkennung haben kann.

Im Fall eines Unsupervised Datensatzes entwickelte das Team um Fabio Carrara das Modell CBiGAN [5], welches ein GAN mit einem AE kombiniert. Als Grundlage für das Modell dient das Bidirectional GAN [12]. Es verbessert die Modellierung des latenten Raums, indem ein Encoder hinzugefügt wird, der reale Proben in dem entsprechenden latenten Raum abbildet und zusammen mit dem Generator trainiert. Dabei entscheidet der Diskriminatator, ob der erzeugte Datenpunkt real oder generiert ist. Somit handelt es sich bei diesem Modell um einen One-Class-Klassifikator. Bei CBiGAN handelt es sich um ein Modell, welches Anomalien in Bildern findet, indem bei dem Encoder und Decoder des BiGAN-Modells ein Regularisierungsterm hinzugefügt wird. Die jeweilige Fehlerfunktion (eng. Loss-Function) des Encoders und Generators werden zu einer linearen Fehlerfunktion kombiniert. Hinzu kommt ein Parameter α , der bestimmt, wie stark der Einfluss der jeweiligen Komponente ist. Durch den Regularisierungsterm erreicht das Modell gute Rekonstruktionskonsistenz sowie gute generative Leistungen. Bei dem GAN handelt es sich um die Variante Wasserstein GAN. Diese unterscheidet sich darin, dass der Abstand zwischen echten und generierten Datenpunkten mit dem namensgebenden Wassersteinabstand berechnet wird [13]. Daraus resultieren stabilere Gradienten.

Die Analyse des latenten Raums von einem AE ermöglicht es gezielt Datenpunkte auszuwählen und zu erzeugen. So ist erkennbar, wo das Modell, Anomalien und normale Datenpunkte abbildet. Diese Idee wurde in dieser Arbeit weiterverfolgt und diente als Grundidee für die Analyse des zweidimensionalen latenten Raums.

Um einen Überblick über das Arbeiten mit Semi-Supervised Daten zu erhalten, wurde das Paper „Semi-Supervised Learning with Generative Adversarial Networks“ [8] analysiert. Das Paper beschäftigt sich mit der Implementierung eines GANs für das Lernen von Semi-Supervised Datensätzen. Hierfür hat der Forscher Augustus Odena das GAN so erweitert, dass der Diskriminatator gezwungen wird, die Klassenlabel vorherzusagen. Dies wird durch das Austauschen der Sigmoid-Einheit mit einer Softmax-Layer erreicht. So fungiert der Diskriminatator als Klassifikator und kann die Inputdaten in $n+1$ Klassen einteilen, wobei die $n+1$ Klasse für die generierten Daten steht. Mit diesem Modell wird eine dateneffizientere Klassifikation ermöglicht, die zusätzlich eine bessere Qualität von generierten Daten liefert.

Im nächsten Schritt wird aufgezeigt, wie Anomalieerkennung mit generierten Daten kombinierbar ist. Das Forschungsteam um Zhe Li entwickelte für Semi-Supervised Daten das „RCC-Dual-GAN“ [7] Modell. Das Modell bearbeitet den Fall mit nur wenigen, bekannten vorhandenen Anomalien. Der Rest der Daten ist nicht gelabelt. Die Forscher erweitern das Dual-GAN Modell, indem sie im ersten Schritt des Trainings den Datensatz mit Hilfe eines Klassifizierungsalgorithmus namens „Robust Continuous Clustering (RCC)“ [14] in mehrere Gruppen von Anomalien und normalen

Datenpunkten einteilen. Dual-GAN besteht aus zwei GAN-Gruppen und einem zentralen Diskriminator. Die erste GAN-Gruppe erhält alle unbekannten Datenpunkte als Input. Die zweite Gruppe erhält die bekannten Anomalien. Die jeweiligen Generatoren in den Gruppen erzeugen neue Datenpunkte anhand des Inputs, die eine sehr ähnliche Verteilung wie die Inputdaten besitzen. Die erzeugten Daten und der ursprüngliche Datensatz werden an den zentralen Diskriminator übergeben, welcher den Anomaliescore des eingelesenen Datenpunkts berechnet. Dabei steht ein Score nahe 0 für eine Anomalie und nahe 1 für einen normalen Datenpunkt. Das RCC unterstützt das Dual-GAN, durch eine bessere Vorverarbeitung der Daten. Eine Vorsortierung der Datenpunkte hilft dabei, eine genauere Trennlinie zwischen Anomalien und normalen Datenpunkten zu finden. Die gruppierten Datenpunkte werden, wie beim Dual-GAN, an die jeweiligen Sub-GANs übergeben und erzeugen neue Datenpunkte, welche sich im Bereich des hervorgehenden Clusters befinden. Somit erzeugt jeder Sub-Generator einen neuen Datensatz, anstatt wie bei Dual-GANs einen gemeinsamen. Diese einzelnen Datensätze und der originale Datensatz fließen als Input in den zentralen Diskriminator, der einen Anomaliescore berechnet [14].

Während der Recherche konnte eine Grundidee bei dem Generieren von qualitativ hochwertigen Samples erkannt werden. Um gute Samples nachzubilden, darf der Generator nicht optimal funktionieren, da dieser die Leistung des Diskriminators senkt. Das liegt an der verwendeten MinMax-Funktion von GANs. Sie besagt, dass beide Komponenten nicht zur gleichen Zeit optimal sein können [15]. Die Forscher des Department of Mechanical and Industrial Engineering der Universität Toronto entwickelten hierfür ein Modell names „Vanishing Twin GAN“ [15]. Das Modell besteht aus zwei Generatoren und einem Diskriminator. Der erste Generator ist dem Diskriminator zugehörig, der die Entscheidung trifft. Hinzu kommt der zweite Generator Vanishing Twin, dessen Leistung durch Dropout-Schichten gezielt reduziert wird. Der Diskriminator erhält die erzeugten Datenpunkte des ersten Generators, des Vanishing Twins sowie den originalen Datensatz. Einen ähnlichen Ansatz wie in Vanishing Twin GAN verfolgen die Forscher Chongxuan Li, Kun Xu, Jun Zhu und Bo Zhang mit ihrer Methode Tripple GAN [9]. Sie haben ebenfalls festgestellt, dass die zwei Spieler, Generator und Diskriminator, nicht zum selben Zeitpunkt optimal sein können. Die Idee ist es, einen dritten Spieler in das System zu integrieren. Dem GAN wird ein Klassifikator hinzugefügt. Der Generator und der Klassifikator bestimmen gemeinsam die bedingten Verteilungen zwischen den Datenpunkten und den Labels. Der Klassifikator erstellt für einen Datenpunkt das zugehörige Label, indem es den Datenpunkt kategorisiert. Die erzeugten Paare werden an den Diskriminator übergeben, der die Einteilung in erzeugte und echte Daten vornimmt. Dabei erhält der Diskriminator ebenfalls Datenpunkte mit zugehörigem Label aus dem originalen Datensatz [9].

Anhand dieser Paper ist die Idee entstanden verschiedene Datensätze mit unterschiedlich guten Samples zu generieren und diese einer Anomalieerkennung zu übergeben. Es soll herausgefunden werden, ob die Qualität der generierten Samples einen positiven Einfluss auf die Klassifizierungsleistung der Anomalieerkennungsmethode nimmt.

3 Grundlagen

3.1 Anomalieerkennung - Anomaly Detection

Die in diesem Kapitel beschriebenen Eigenschaften und Methoden wurden hauptsächlich aus dem Paper „Anomaly Detection: A Survey“ von v. Chandola, A. Banerjee und V Kumar [3] entnommen. Das Paper liefert einen guten Überblick über die Anomalieerkennung und deren Eigenschaften.

Die Anomalieerkennung gewinnt immer mehr an Bedeutung in den verschiedensten Branchen und beschreibt das Auffinden von ungewöhnlichen Datenpunkten in einem Datensatz. Beispielsweise findet Anomalieerkennung in der Cyber-Security, Betrugserkennung bei Kreditkarten und Geldtransaktionen und Medizin statt. In der Medizin können z.B.: schwarze Flecken in Röntgenbildern Anomalien darstellen oder in der Betrugserkennung unbefugte Zahlungen. Durch die Digitalisierung und den daraus resultierenden, großen Datenmengen ist es sehr zeitaufwendig und kostspielig, Daten nach Ausreißern zu untersuchen [3] [16].

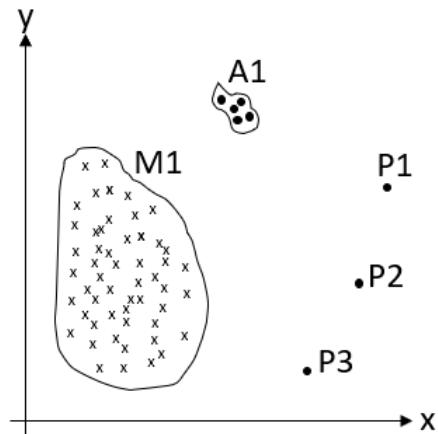


Abbildung 1: Beispielhafte Darstellung von Anomalien

Abbildung 1 verdeutlicht die Definition einer Anomalie. Bei den Datenpunkten in der Menge M_1 handelt es sich um normale Daten. Somit werden Datenpunkte, welche neu in das System kommen und sich in diesem Bereich eingliedern, auch als normal klassifiziert. Punkte P_1 , P_2 und P_3 hingegen befinden sich signifikant weit entfernt von der Menge M_1 und werden durch das System als Anomalie eingestuft. Bei Anomalien muss es sich nicht um einzelne Punkte handeln, es kann ebenfalls eine Menge von Anomalien geben, wie zum Beispiel die Menge A_1 . Es wird zwischen drei Arten von Anomalien unterschieden, der Point Anomaly, der Conditional Anomaly und der Collective Anomaly.

Point Anomaly Wie der Name schon verrät, handelt es sich bei einer Point Anomaly um punktuell auftretende Anomalien. In Abbildung 1 sind die Punkte P_1 , P_2 und P_3 Beispiele für diese Art von Anomalie. Somit wird eine Punktanomalie danach definiert, ob sie sich in der natürlichen Menge der normalen Datenpunkte befindet oder nicht [3].

Werden beispielsweise Netzwerkdaten anhand des Parameters der Übertragungsmenge betrachtet, könnte eine Punktanomalie z.B. daran erkannt werden, dass die Anzahl an heruntergeladen MB im Vergleich zu den restlichen Zugriffen sehr groß ist.

Conditional Anomaly Eine Conditional Anomaly beschreibt eine Dateninstanz abhängig vom Kontext als normal oder anomal [17]. Als Beispiel kann die Belastung von Kreditkarten betrachtet werden. Angenommen Person A gibt monatlich 500 Euro aus, jedoch betragen die Ausgaben im Dezember 2000 Euro. Die Ausgaben im Dezember werden als Conditional Anomaly klassifiziert, da sie an die Bedingung gebunden sind, dass es sich im Dezember um Weihnachtseinkäufe handelt [3]. Diese Art von Anomalien wird durch zwei Attribute charakterisiert. Einerseits werden kontextbezogenen Attribute betrachtet, welche die direkte Nachbarschaft für diesen Datenpunkt beschreiben. Bei Daten mit einer Zeitreihe wäre die Zeit ein solches Attribut, da sie die Position einer Dateninstanz auf der vollständigen Zeitreihe bestimmt. Das zweite Verhaltensattribut ist kontextfrei und beschreibt das Verhalten eines Datenpunktes [3]. Wird beispielsweise ein Datensatz mit allen Kreditkartenzahlungen der deutschen Bürger betrachtet, so ist eine Kreditkartenzahlung eines Bürgers ein Verhaltensattribut. Um eine Conditional Anomaly zu erkennen, muss das Verhaltensattribut in Kombination mit dem kontextbezogenen Attribut betrachtet werden, da sich die Klassifizierung in normal und anomalous je nach Kontext ändern kann. Wird das vorherige Beispiel der Kreditkartenzahlungen betrachtet, so ist die deutlich höhere Summe im Dezember bezüglich des Kontextes „Weihnachten“ keine Anomalie. Tritt diese Erhöhung der Ausgaben jedoch im Juli ein, so würde dieser Punkt als Anomalie bewertet werden.

Collective Anomaly Collective Anomaly bezeichnet Anomalien, welche durch ihre Menge definiert werden. Das muss jedoch nicht bedeuten, dass die einzelnen Datenpunkte der Menge eine Anomalie darstellen. Das Auftreten der Punkte als Gruppe definiert eine Anomalie. Diese Art von Anomalien können nur in Datensätzen auftreten,

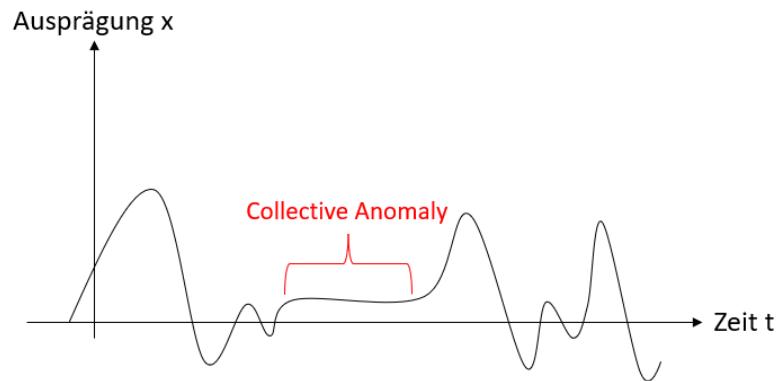


Abbildung 2: Beispielhafte Darstellung einer Collective Anomaly

bei denen die Datenpunkte zueinander abhängig sind, wie zum Beispiel bei sequenziellen und räumlichen Daten. In Abbildung 2 wird eine Collective Anomaly dargestellt. Der

gekennzeichnete Bereich ist eine Anomalie, da der Wert über einen längeren Zeitraum gehalten wird, was innerhalb des Datensatz nicht üblich ist. Es ist zu beachten, dass der niedrige Wert an sich keine Anomalie darstellt [3].

3.1.1 Schwierigkeiten bei der Anomalieerkennung

In dem vorhergehenden Abschnitt wurde die Frage geklärt, was eine Anomalie definiert, jedoch ist die Erkennung solcher anomalen Datenpunkte komplex. Prinzipiell müssen Datenpunkte gefunden werden, welche nicht in die normale Datenverteilung passen. Dabei gibt es jedoch mehrere Probleme. In der Realität weisen Datensätze Rauschen auf, was die Anomalieerkennung erschwert. Oft findet vor der Analyse eine Rauschreduzierung statt, welche auch Anomalien entfernen kann. Somit werden Anomalien gar nicht in die Analyse miteinbezogen und Attacken auf ein System bleiben gegebenenfalls unbemerkt. Außerdem sind nur sehr wenige Anomalien bekannt. In Kombination mit dem Problem des Datenrauschen bleibt für das Training eines neuronalen Netzes nur sehr wenige Anomalien zur Verfügung, was sich negativ auf die Klassifizierungsleistung des Netzes auswirkt [3].

Eine Anomalie ist so definiert, dass ein Datenpunkt nicht der normalen Datenverteilung entspricht. Es ist jedoch schwierig, eine genaue Trennlinie zwischen den Bereichen für normale und anomale Punkte festzulegen. So können normale Dateninstanzen nahe, jedoch außerhalb des normalen Bereichs liegen und so fälschlicherweise als Anomalie klassifiziert werden.

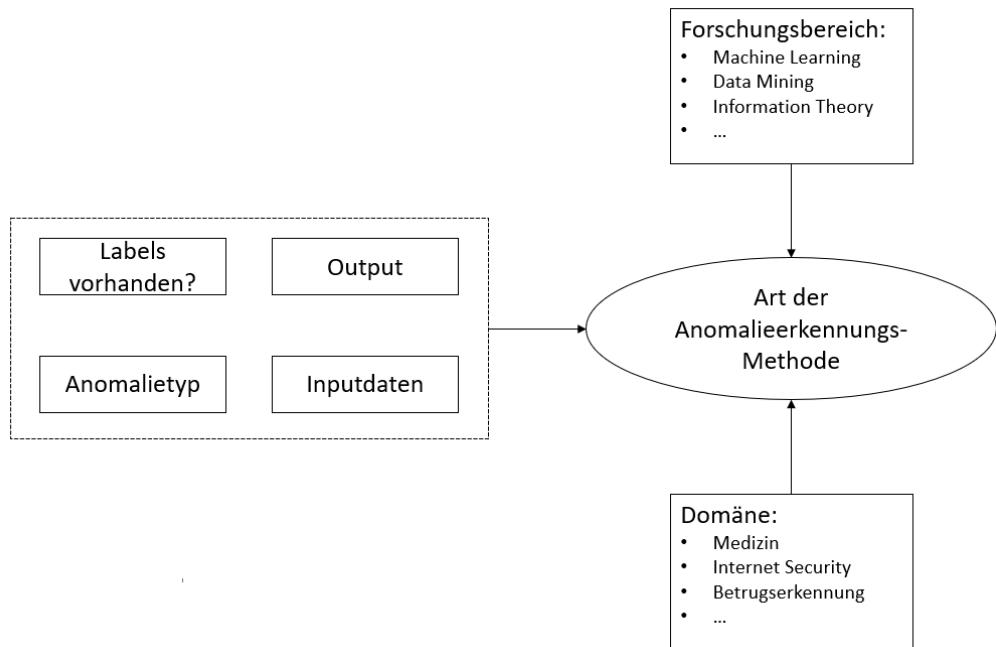


Abbildung 3: Kriterien für Anomalieerkennungsmethode

Umgekehrt können Anomalien nahe dem Bereich der normalen Datenpunkte liegen und so fälschlicherweise als normale Instanz klassifiziert werden. In vielen Bereichen findet eine stetige Veränderung der Daten statt. Daher verändert sich der Datensatz mit der

Zeit und neue normale Datenpunkte könnten durch die alte Definition von Anomalien fälschlicherweise als solche erkannt werden. Daher muss auch die Definition von normal und anomalo stetig angepasst werden.

Erschwerend kommt dazu, dass die Anomalieerkennung sehr domänenspezifisch ist. Jede Domäne arbeitet mit verschiedenen Datensätzen, wie zum Beispiel Bilddaten, Textdaten oder Audiospuren. Unabhängig vom Datentyp können Schwankungen verschiedene Auswirkungen haben. Beispielsweise sind kleinste Abweichungen von normalen Werten wie z.B.: bei der Körpertemperatur sehr gefährlich wohingegen im Finanzmarkt eine hohe Volatilität normal ist [3]. Mit diesem Wissen muss je nach Datentyp, Anomalieart, Forschungsbereich und Domäne die passende Anomalieerkennungsmethode gefunden werden. Die Auswahlkriterien werden in Abbildung 3 dargestellt.

3.1.2 Supervised Anomaly Detection

Die Voraussetzung für Supervised Anomaly Detection ist ein vollständig bekannter Datensatz. Das bedeutet, alle normalen sowie anomalen Daten sind bekannt. Bei dieser Art von Detektion handelt es sich in den meisten Fällen um ein Vorhersagemodell. Das Modell wird mit dem bekannten Datensatz trainiert. Der neue Datenpunkt wird mit den vorhandenen Daten verglichen und der ähnlichsten Klasse zugewiesen. Ein Problem dabei ist, dass die Klassen ungleichmäßig verteilt sind, da es viel mehr normale als anomale Datenpunkte gibt. Dies beeinträchtigt die Leistung solcher Implementierungen. Außerdem ist es in der Realität sehr schwierig, Datensätze mit vollständigen Klassifizierungen zu erhalten. Es ist meist nur eine begrenzte Anzahl von Anomalien bekannt [3].

3.1.3 Unsupervised Anomaly Detection

Bei der unüberwachten Anomalieerkennung wird angenommen, dass es deutlich mehr normale, ohne die exakte Labels zu kennen, als anomale Daten gibt. Das ist notwendig, da keine Klassen für den Datensatz existieren. Sollte die Annahme nicht zutreffen, kommt es zu einer hohen Fehlerrate [3]. Dies bedeutet, dass normale Datenpunkte sehr oft als Anomalie klassifiziert werden. Der Vorteil dieser Methode ist, dass sie in vielen Bereichen eingesetzt werden kann, da die Art des Datensatzes keine große Rolle spielt. Die Methode funktioniert allerdings nur, wenn die Annahme der deutlich höheren Präsenz von normalen Daten zutrifft, da ansonsten viele normale Datenpunkte als Anomalie klassifiziert werden. Das bedeutet es kommt zu vielen False Positiv Ergebnissen [18].

3.1.4 Semi-Supervised Anomaly Detection

Eine Möglichkeit die Supervised Anomalieerkennung relevanter für echte Szenarien zu gestalten, ist die Umwandlung in ein Semi-Supervised Anomaly-Detection-System. Hierfür muss nur ein Teil der Daten bekannt sein, meistens alle normalen Daten. In manchen

Fällen sind ebenfalls ein paar bekannte Anomalien vorhanden. Die Modelle erlernen das normale Verhalten der Daten und können anhand dieses Wissens neue Daten in normal oder anomalo unterteilen [3].

3.1.5 Ausgabe der Anomalieerkennungsmethoden

Zum Schluss muss die Frage geklärt werden, wie der Output eines solchen Modells aussieht. Dabei gibt es zwei verschiedene Möglichkeiten das Ergebnis auszugeben. Es kann wie in dem Paper „ A^3 Activation Anomaly Analysis“ von den Forschern Philip Sperl, Jan-Philipp Schulze und Konstantin Böttinger [16] ein Anomaliescore berechnet werden, welcher aussagt, mit welcher Wahrscheinlichkeit ein Datenpunkt eine Anomalie bzw. ein normaler Datenpunkt ist. Meist wird hier ein Wert zwischen Null und Eins vergeben. Die zweite Möglichkeit ist die Ausgabe eines Labels anstatt numerischer Werte. Beispielsweise können hier die Labels *Normal* und *Anomalie* verwendet werden [3].

3.2 Generative Machine Learning Modelle

3.2.1 Generatives Lernen

Im Bereich des maschinellen Lernens wird zwischen überwachtem und unüberwachtem Lernen, sowie diskriminativen und generativen Modellen unterschieden. Generative Modelle berechnen eine Wahrscheinlichkeitsverteilung über die Variablen eines Systems, wohingegen die diskriminativen Modelle ein diskretes Mapping zwischen den Input- und Outputvariablen durchführen [19]. Zu den diskriminativen Modellen gehören auch beispielsweise Regressionsmodelle. Durch diese Definition unterscheiden sich diskriminative von generativen Modellen genauso wie unüberwachtes von überwachtem Lernen. Allerdings entsprechen alle überwachten Modelle diskriminativen Modellen. Bei unüberwachten Modellen kann in diskriminativ und generativ unterteilt werden. Sobald die Wahrscheinlichkeitsverteilung über alle Variablen dazu verwendet wird, neue Samples aus den Inputdaten zu generieren, handelt es sich um ein generatives Modell. Diese Art von Modellen sind in den meisten Fällen unüberwacht. Lineare Regression, Scalar Vector Machine (SVM) oder auch traditionelle Neuronale Netze sind Beispiele für diskriminative Modelle. Beispiele für generative Modelle sind Bayesian Networks, Hidden Markov Models, AE und GAN. In den letzten Jahren haben generative Modelle an Popularität



Abbildung 4: Fortschritt der Generierung menschlicher Gesichter [20]

gewonnen, was an verschiedenen Faktoren liegt. Das Ziel im Machine Learning ist es nicht nur die Daten zu klassifizieren, sondern gleichzeitig ein besseres Verständnis der Daten zu erhalten. Dazu zählt die Nachvollziehbarkeit der Datenerzeugung. Hierbei können tiefe, generative Modelle helfen. Des Weiteren haben erfolgreiche Projekte wie zum Beispiel StyleGAN von NVIDIA [21], welches realistische Bilder von menschlichen Gesichtern generiert, oder auch das GPT-2 Sprachmodell von OpenAI [22], welches Textpassagen in kurze Paragraphen zusammenfasst, zu dem Aufschwung der generativen Modelle beigetragen. In Abbildung 4 wird der Fortschritt der Generierung von Bildern mit menschlichen Gesichtern dargestellt. Abgesehen von den technischen Aspekten kommen generative Modelle dem menschlichen Gehirn am nächsten. Menschen sind in der Lage, sich verschiedene existierende und nichtexistierende Objekte vorzustellen. Generative Modelle können dabei helfen, ein besseres Verständnis über die Funktionsweise des menschlichen Gehirns zu erlangen und so die Forschung im Bereich Artificial Intelligence (AI) voranzutreiben [23]. Wie bereits beschrieben, wird ein Generatives Modell verwendet, um neue Datenpunkte anhand einer Datenverteilung $p_{(data)}$ zu erstellen. Hierfür kann ein Framework verfasst werden, welches die Aufgaben und Ziele der Generativen Modelle beschreibt. Als Input wird ein Datensatz x benötigt, dessen Datenpunkte einer unbekannten Wahrscheinlichkeitsverteilung $p_{(data)}$ unterliegen. Das Generative Modell lernt eine Wahrscheinlichkeitsverteilung $p_{(Modell)}$ und versucht damit die Verteilung $p_{(data)}$ so gut wie möglich zu lernen und zu kopieren. Im Laufe des Lernprozesses soll das Generative Modell Datenpunkte erzeugen können, welche nicht von den Inputdatenpunkten x unterschieden werden können. Dabei ist es wichtig, dass das Generative Modell nicht die Datenpunkte aus x kopiert, sondern neue, den Inputdaten ähnliche Datenpunkte generiert [23].

3.2.2 Autoencoder

Der Autoencoder (AE) (Abb. 5) ist ein Neuronales Netz, welches lernt, den Input nachzubilden. Genauer gesagt, lernen diese Netze eine effiziente Repräsentation der Eingabedaten [24]. Da das Modell nicht die Inputdaten exakte Nachbilden soll, wird

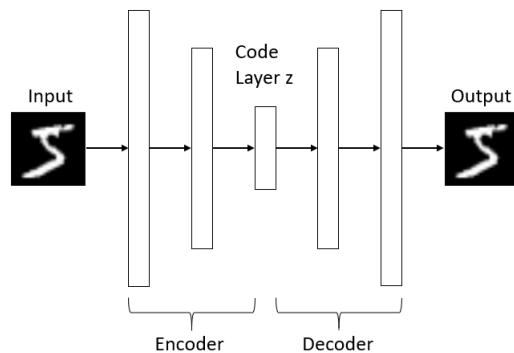


Abbildung 5: Aufbau eines Autoencoders

durch Restriktionen aktiv versucht, den AE zu zwingen, Merkmale und Features zu lernen und aus diesen neuen Daten zu generieren. Aufgrund dieser Eigenschaft werden Autoencoder zur Feature-Extraktion, sowie zur Anomalieerkennung verwendet. Um die Features zu extrahieren, wird die Dimension der Inputdaten reduziert. Beispielsweise können Bilder der Größe 28x28 Pixel auf eine Dimension von zwei reduziert werden, was wiederum bedeutet, dass hier eine Destillation von zwei Merkmalen geschieht. Diese Einschränkung führt dazu, dass die Kopie des Inputs nur teilweise möglich ist. Das Modell muss die Features kopieren, welche den Trainingsdaten am meisten ähneln. Dadurch muss der AE Prioritäten setzen und die wichtigsten Merkmale der Daten kopieren und erlernen. Im Anschluss wird die Dimension wieder auf die ursprüngliche Inputdimension vergrößert. Daraus entsteht ein spezifischer, symmetrischer Aufbau, der in Abbildung 5 skizziert ist. Der AE besteht aus zwei Hauptkomponenten. Encoder $h = f(x)$ führt die Dimensionsreduktion durch und Decoder $r = g(h)$ rekonstruiert die Daten. Der Encoder wird als Recognition Network und der Decoder als Generative Network bezeichnet [10]. AE gehören zu den Feedforward Networks, sie können deshalb mit denselben Algorithmen trainiert werden. Heutzutage können die Autoencoder mit deterministischen Funktionen auf die stochastische Abbildung $p_{Encoder}(r|x)$ und $p_{Decoder}(x|r)$ verallgemeinert werden [10].

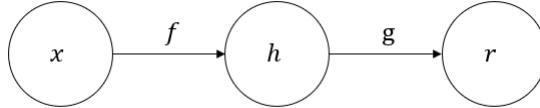


Abbildung 6: Struktur eines Autoencoders

In Abbildung 6 wird die generelle Struktur eines Autoencoders aufgezeigt. Es wird ein Input x zu einem Output r gemappt. Dabei wird der Input zuerst auf die Hidden Layer h abgebildet und danach rekonstruiert durch das Mapping von h zu r [10]. Die Grundidee des Autoencoders wurde im Laufe der Zeit weiterentwickelt und kann in vielen verschiedenen Aufgabengebieten eingesetzt werden. Beispielsweise gibt es Denoising Autoencoder, welche den strukturellen Aufbau der Daten lernen und so Unreinheiten wie z.B.: beschädigte oder verzerrte Datenpunkte im Datensatz entfernen [10].

3.2.3 Untervollständige Autoencoder

Autoencoder mit dem Aufbau aus Abbildung 5 heißen Untervollständige Autoencoder. Hierbei liegt das Hauptaugenmerk nicht auf dem Output, sondern auf den Merkmalen, welches das Netz innerhalb seiner Hidden-Layers lernt. Für diese Aufgabe ist es wichtig, dass die Hidden-Layers eine kleinere Dimension besitzen als der Input. Wird die Dimension nicht oder nicht ausreichend verkleinert so lernt das Netz den Input zu kopieren, ohne die Merkmale zu extrahieren. Dabei kann der Lernprozess als Minimierung der Verlustfunktion $L(x, g(f(x)))$ beschrieben werden. Hier „bestraft“ die Verlustfunktion L die Abweichung des Outputs $g(f(x))$ vom Input x [10]. Um die

Funktionsweise eines AE zu veranschaulichen wurde in einem Versuch ein AE mit linearer Aktivierungsfunktion mit der Hauptkomponentenanalyse (HKA) verglichen. Beide Modelle führen eine Dimensionsreduzierung durch und filtern dadurch klassenspezifische Merkmale. Die Hauptkomponentenanalyse (HKA) ist ein Verfahren zur Dimensionsreduzierung von hochdimensionalen Daten. Das Verfahren filtert aus den Daten die wichtigsten Merkmale und bündelt diese in Hauptkomponenten, um so eine Klassifizierung durchzuführen [24]. Für den Fall, dass der Decoder linear und die Verlustfunktion die mittlere quadratische Abweichung ist, stecken beide den gleichen latenten Raum ab [24]. Um dies zu veranschaulichen, wurde ein unvollständiger Autoencoder, sowie eine HKA implementiert und die Ergebnisse miteinander verglichen. Die HKA erreicht einen Rekonstruktionsfehler von 0.056. Dementsprechend müsste der AE ein sehr ähnliches Ergebnis liefern wie die HKA. Hierfür wurde ein AE mit linearer Aktivierungsfunktion implementiert:

```
m = Sequential()
m.add(Dense(784, activation='linear', input_shape=(784,)))
m.add(Dense(512, activation='linear'))
m.add(Dense(256, activation='linear'))
m.add(Dense(128, activation='linear'))
m.add(Dense(2, activation='linear', name="middleHiddenLayer"))
m.add(Dense(128, activation='linear'))
m.add(Dense(256, activation='linear'))
m.add(Dense(512, activation='linear'))
m.add(Dense(784, activation='sigmoid'))
m.compile(loss='mean_squared_error', optimizer = Adam())
history = m.fit(x_train, x_train, batch_size=128, epochs=5, verbose=1,
                  validation_data=(x_test, y_test))
encoder = Model(m.input, m.get_layer('middleHiddenLayer').output)
Zenc = encoder.predict(x_train)
Renc = m.predict(x_train)
```

Code 1: Linearer, unvollständiger Autoencoder mit mittlerer, quadratische Abweichung

Trainiert wurde der AE über eine Länge von fünf Epochen mit einer Batchgröße von 128. Beim Trainieren und Validieren wurde derselbe Datensatz wie bei der HKA verwendet. Der Datensatz beinhaltet handgeschriebene Zahlen zwischen null und neun. Dieser



Abbildung 7: Vergleich der Klassifizierung

sogenannte MNIST-Datensatz wurde zuvor in Trainings- und Testdaten unterteilt und die Pixelwerte normiert. Der AE erreichte einen Loss von 0.0553 und ist damit nicht wesentlich besser als die HKA mit einem Loss von 0.056. Abbildung 7 zeigt die Klassifikation der HKA im Vergleich zur Klassifizierung des linearen Autoencoders. Jede Farbe stellt eine Klasse dar, also eine Zahl. Das Muster, anhand dessen Datenpunkte klassifiziert werden ist ähnlich und die Klassen können gut erkannt werden. Dennoch gibt es einen großen Bereich von Daten, den beide Algorithmen nicht gut voneinander trennen können.

Um das Beispiel konkreter zu visualisieren, wurden die Outputs der Algorithmen in Abbildung 8 dargestellt. In der ersten Reihe werden zehn Zahlen aus dem Input

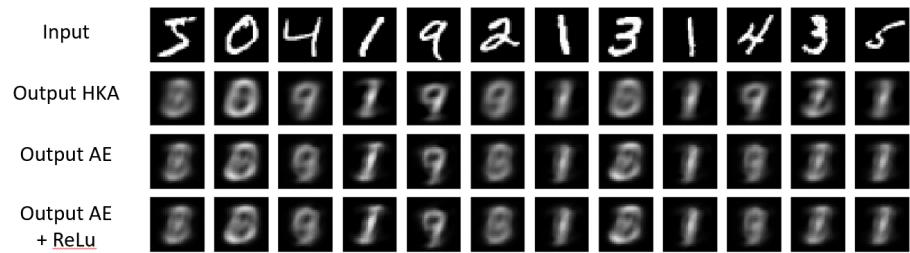


Abbildung 8: Rekonstruierte Ergebnisse der HKA und des Autoencoders

angezeigt. In der zweiten Reihe wird der rekonstruierte Output der HKA dargestellt. Die dritte Zeile bildet das Ergebnis des AE ab. Der Unterschied zwischen der HKA und dem AE mit linearer Aktivierungsfunktion ist sehr klein. Sie haben beide Probleme, Zahlen genau einem Input zuzuordnen. Die letzte Zeile stellt das Ergebnis eines Autoencoders mit einer Rectified Linear Unit (ReLU)-Aktivierungsfunktion und einer Trainingslänge von 15 Epochen dar.

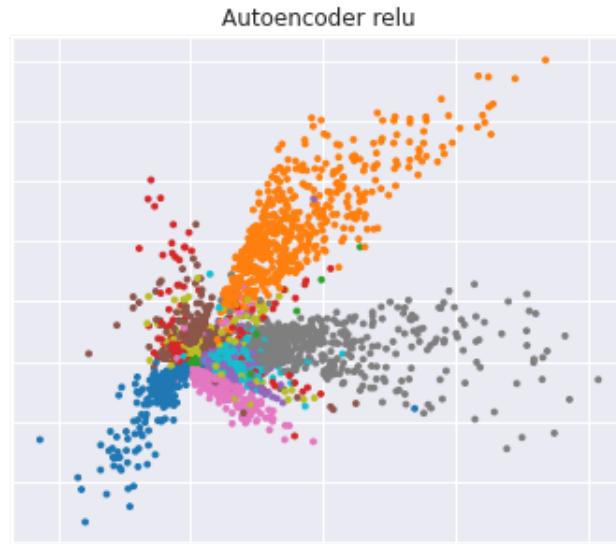


Abbildung 9: Klassifizierungsergebnis des AE mit ReLU-Aktivierungsfunktion. Datenpunkte werden in bestimmte Bereiche gruppiert. Die Veränderung der Aktivierungsfunktion steigert die Erkennungsleistung deutlich gegenüber der einfach linearen Aktivierungsfunktion

Dieser Autoencoder wurde zum besseren Vergleich implementiert. Die Architektur unterscheidet sich nicht von der des linearen AE, abgesehen von der Aktivierungsfunktion. Bei der Rekonstruktion können keine wesentlichen Verbesserungen erzielt werden, jedoch ist die Klassifizierung deutlich besser (siehe Abb. 9). Die Klassen sind sichtlich besser getrennt als bei der HKA oder des AE mit linearer Aktivierungsfunktion. Das Gegenteil eines unvollständigen Autoencoders ist der übervollständige Autoencoder. In diesem Fall ist die Dimension der Hidden-Layers größer als die des Inputs. Diese Architektur ermöglicht ebenfalls den Output an den Input anzulegen, jedoch werden hierbei keine Features erlernt. Die Funktionsweise und die Lernfähigkeit von AE wird jedoch nicht nur durch die Dimension bestimmt. Mithilfe der Verlustfunktion kann ein Autoencoder weitere Aufgaben erfüllen. Diese Arten von AE heißen Regularisierte AE. Zu dessen Aufgaben gehören zum Beispiel die Robustheit gegenüber Rauschen, die Ableitung der Darstellung, oder die Sparse Representation.

3.2.4 Adversarial Autoencoder

Ein AAE ist ein Modell, welches ein AE in ein generatives Modell verwandelt. Dabei verfolgt der AAE zwei verschiedene Kriterien: das Rekonstruktionsfehler-Kriterium und das Adversarial Training-Kriterium [25]. Das Adversarial Training-Kriterium bewirkt, dass sich die aggregierte Posterior-Verteilung der latenten Repräsentation, also der komprimierten Codeschicht des Autoencoders, an eine beliebige Prior-Verteilung anpasst.

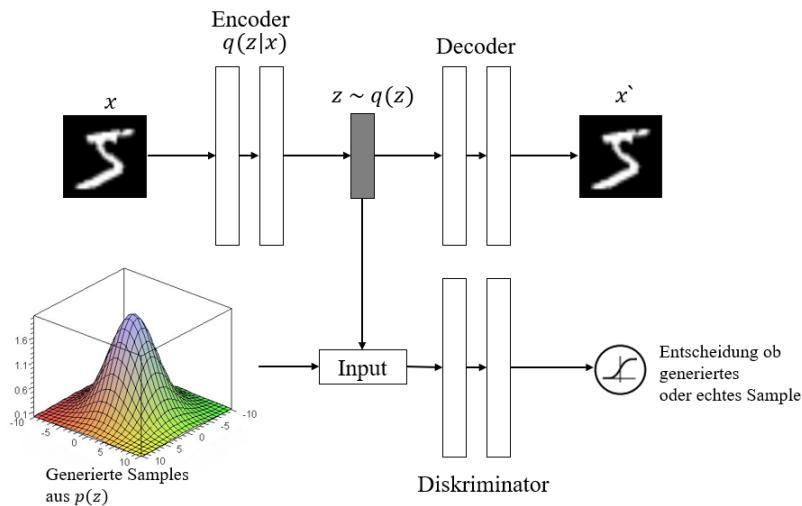


Abbildung 10: Skizzierte Darstellung eines AAE

Durch diese Trainingskriterien lernt der Encoder, die Datenverteilung in eine Prior-Verteilung umzuwandeln und der Decoder lernt ein generatives Modell, welches die auferlegte Prior-Verteilung auf die Datenverteilung abbildet [25]. Abbildung 10 zeigt die Architektur eines AAE. Im oberen Teil des Modells ist der Aufbau eines klassischen Autoencoders zu sehen, welcher als Input die Datenpunkte x erhält. Der Encoder komprimiert die Informationen bis hin zur Code-Layer z . Er trägt ebenfalls die Rolle des

Generators und versucht das diskriminative Netz zu täuschen, dass es nicht zwischen generierten und echten Daten unterscheiden kann. Der Decoder benutzt diese komprimierten Informationen und rekonstruiert das Inputbild bestmöglich. Der darunter liegende Teil der Abbildung bildet das diskriminative Modell, welches als Input durch Rauschen erzeugte Bilder, sowie die Informationen aus der Code-Layer Schicht erhält. Der Diskriminator entscheidet im nächsten Schritt, ob die Daten generiert oder echt sind. Die aggregierte Posterior-Verteilung $q(z)$ wird durch die Kodierungsfunktion des Autoencoders $q(z|x)$ definiert. Sie beschreibt die Code-Layer z des Autoencoders wie folgt [25]:

$$q(z) = \int_x q(z|x)p_d(x)dx \quad (1)$$

x steht hier für den Input des Netzwerks und z für den latenten Code-Layer Vektor, der sich aus der Code-Layer-Schicht ergibt. Der Autoencoder kann mit den Wahrscheinlichkeiten $q(z|x)$ für die Encoderverteilung und $p(x|z)$ für die Decoderverteilung beschrieben werden. Die Wahrscheinlichkeit $p_d(x)$ definiert die Datenverteilung des Inputdatensatzes. Hinzu kommt die Wahrscheinlichkeitsverteilung $p(z)$, welche dem Netz auferlegt werden soll. Aus dieser Verteilung werden die Samples generiert, da bei dem AAE versucht wird, dem Modell diese aufzulegen. Die Modellverteilung wird durch $p(x)$ definiert [25]. Der AE im AAE erfüllt die klassische Aufgabe, den Rekonstruktionsfehler zu minimieren. Das diskriminative Modell ist für die Regularisierung zuständig, indem es die aggregierte Posterior-Verteilung $q(z)$ an die Prior-Verteilung $p(z)$ angleicht. Das bedeutet, sie versucht die Verteilung der Code Layer an die auferlegte Verteilung anzupassen. [25].

Da der Encoder ebenfalls die Aufgabe des Generators im AAE übernimmt, gibt es verschiedene Optionen für die Verteilung $q(z|x)$. Die einfachste Variante ist, dass die Verteilung $q(z|x)$ eine deterministische Funktion von x ist. Dies bedeutet, die Wahrscheinlichkeit in $q(z)$ stammt nur von der Datenverteilung $p_d(x)$ ab. Eine weitere Möglichkeit ist die Gaußian Posterior-Verteilung.

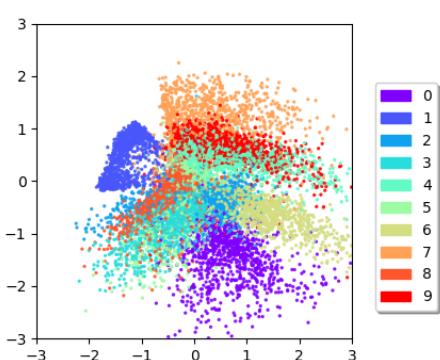


Abbildung 11: Latenter Raum eines deterministischen Unsupervised AAE auf dem MNIST Datensatz

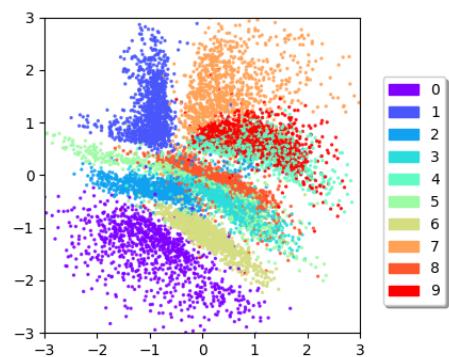


Abbildung 12: Latenter Raum eines deterministischen Unsupervised AAE mit CNN-Aufbau auf dem MNIST Datensatz

Hier wird eine Gaußsche Normalverteilung $\mathcal{N}(\mu_i(x), \sigma_i(x))$ angenommen, deren Varianz und Mittelwert vom Encoder vorhergesagt werden. Anders als bei der deterministischen Variante besteht die Wahrscheinlichkeit in $q(z)$ aus zwei Verteilungen: der Datenverteilung und der Normalverteilung $z \sim \mathcal{N}(\mu_i(x), \sigma_i(x))$ [25]. In Abbildung 11 wird die Trennung der Klassen des MNIST-Datensatz dargestellt. Der Unsupervised, deterministische AAE, also die simpelste Struktur, kann die Datenpunkte bereits gut voneinander trennen und in bestimmte Bereiche einteilen. Wird der Aufbau dieses AAE durch eine tiefere und komplexere Architektur verbessert, so können bereits sehr gute Ergebnisse erzielt werden (Abb. 12). Es besteht ebenfalls die Möglichkeit einen universellen Approximator zu verwenden. Der Vorteil hier ist, dass der Encoder $q(z|x)$ jede beliebige Posterior-Verteilung, anhand der Inputdaten x , annehmen kann. Um dies zu ermöglichen, bildet der Encoder mit Hilfe des Inputs x und eines Rauschens n mit fester Verteilung, z.B.: Gauß, die Funktion $f(x, n)$. Im nächsten Schritt kann eine Stichprobe der Posterior-Verteilung $q(z|x)$ erstellt werden, indem die Encoderfunktion $f(x, n)$ mit unterschiedlichen Inputdaten evaluiert wird [25]:

$$q(z|x, n) = \delta(z - f(x, n)) \quad (2)$$

Dabei werden $q(z|x)$ und $q(z)$ wie folgt definiert: [25]

$$q(z|x) = \int_n q(z|x, n)p_n(n)dn \quad (3)$$

$$q(z) = \int_x \int_n q(z|x, n)p_d(x)p_n(n)dndx \quad (4)$$

Bei der Implementierung können weitere Anpassungen am Modell vorgenommen werden. Beispielsweise können verschiedene Fehlerfunktionen z.B.: WassersteinGAN verwendet, oder die Netzarchitektur des Encoders und Decoders ähnlich eines tiefen Convolutional Neural Network (CNN) aufgebaut werden.

Der AAE kann in drei Unterkategorien eingeteilt werden, da das Modell alle Varianten von Datensätzen verarbeiten kann. Somit gibt es einen Unsupervised, Supervised und Semi-Supervised AAE. Prinzipiell besitzen alle drei Arten des AAE dieselbe Funktionsweise.

3.2.4.1 Unsupervised Adversarial Autoencoder

Unsupervised bedeutet, dass keine Labelinformationen vorhanden sind, sondern nur der Datenpunkt. Die Architektur eines Unsupervised AAE wird in Abbildung 10 dargestellt. Hier werden die Inputdaten eingelesen und verarbeitet wie im Abschnitt 4.2.4 beschrieben.

3.2.4.2 Supervised Adversarial Autoencoder

Beim Supervised AAE liegen Daten vor, welche vollständig gelabelt sind, d.h. sie bestehen jeweils aus einem Datenpunkt und der dazugehörigen Klasse.

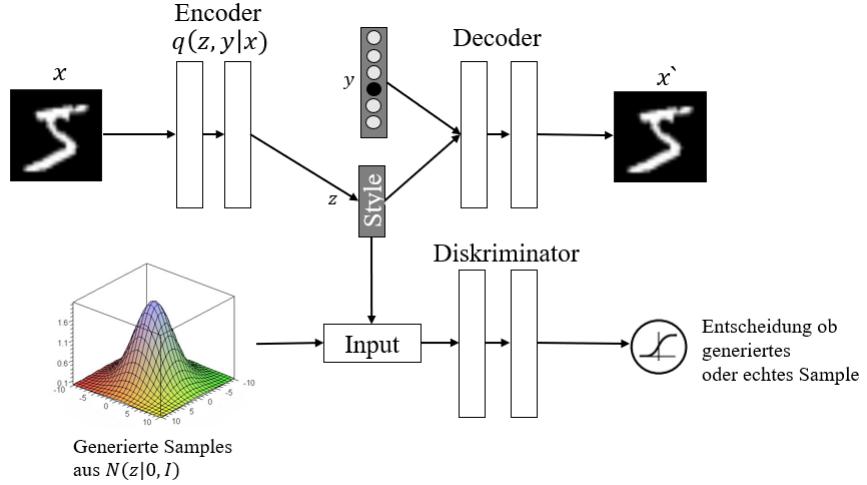


Abbildung 13: Architektur eines Supervised Adversarial Autoencoder (AAE)

Dieses Wissen kann verwendet werden, um die Style-Informationen der Code-Layer von den Label-Informationen zu trennen. Der schematische Aufbau dieses AAE wird in Abbildung 13 skizziert. Als Erweiterung zu der Architektur des Unsupervised AAE werden die Labels als One-Hot-Vektor-Kodierung mit den Style-Informationen der latenten Code-Layer an den Decoder übergeben [25]. Dadurch benutzt der Decoder beide

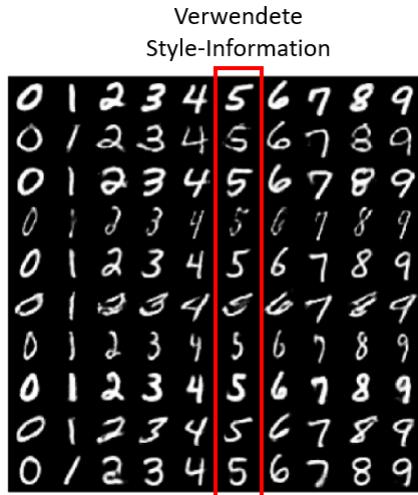


Abbildung 14: Generieren von Daten mit vorgegebenem Style [25]

Informationen für die Rekonstruktion des Inputs und behält so alle Informationen unabhängig vom Label. Mit Hilfe dieser Fähigkeit können gezielt Datenpunkte erstellt werden, die einem bestimmten Aussehen (Style) entsprechen. In dem Paper „Adversarial Autoencoders“ führten Alireza Makhzaniein et al. Experimente durch, in dem Sie dem Autoencoder einen bestimmten Style-Vorgaben und die Labels änderten. In Abbildung 14 wird das Ergebnis dargestellt. Dabei werden die Ziffern aus dem MNIST-Datensatz abgebildet, wobei für jede Zeile die Style-Information der Ziffer Fünf verwendet wurde. Es ist gut zu erkennen, dass der Style auf die generierten Datenpunkte übertragen wurde.

3.2.4.3 Semi-Supervised Adversarial Autoencoder

Die Architektur des Supervised AAE dient als Basis für das Semi-Supervised Modell. Hier werden Daten verarbeitet, die sich aus Datenpunkten und begrenzter Anzahl dazugehöriger Labels zusammensetzen.

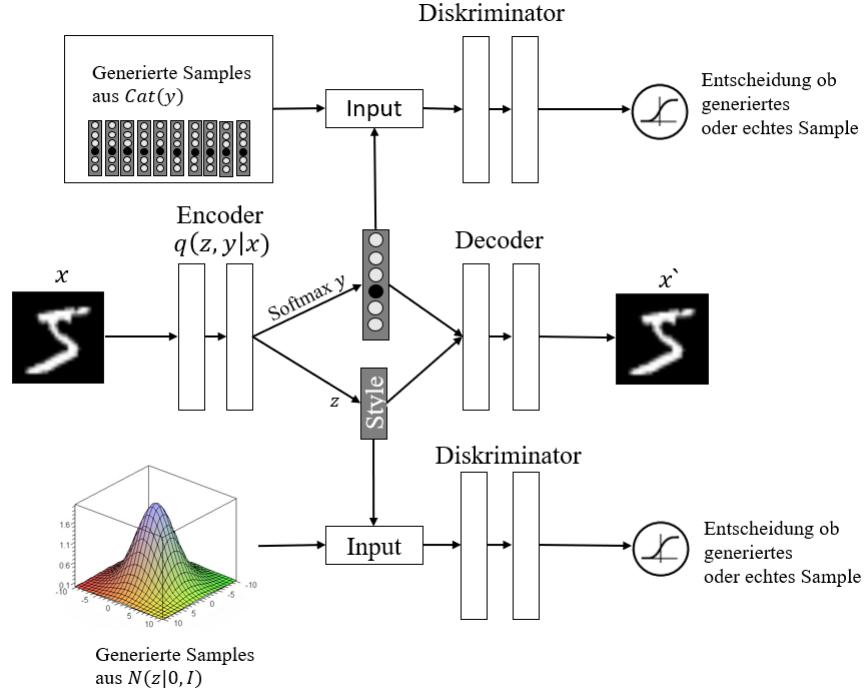


Abbildung 15: Architektur eines Semi-Supervised AAE

Das bedeutet, dass nicht jeder Datenpunkt ein dazugehöriges Label besitzt. Die Architektur eines solchen Netzwerks wird in Abbildung 15 dargestellt. Dem Modell wird ein weiterer Diskriminator hinzugefügt, welcher für die Labels verantwortlich ist. Hierfür wird dem Encoder, welcher auch als Generator fungiert, der Input ohne Label übergeben, welcher mit Hilfe einer Softmax-Layer die latenten Labelinformation bestimmt. Zudem wird die Style-Information in der Code-Layer z komprimiert. Der Encoder folgt dementsprechend der Verteilung $p(z, y|x)$. Für die Rekonstruktion des Inputs erhält der Decoder die latente Labelinformation als One-Hot-Vektor-Kodierung sowie den latenten Hidden-Code z . Die Style-Information wird wie bei den Unsupervised und Supervised Modellen verarbeitet, indem der Diskriminator entscheiden muss, ob es sich um Daten handelt, die durch gaußsches Rauschen erzeugt wurden, oder, der vom Encoder komprimierten Information z , handelt.

$$p(z) = \mathcal{N}(z|0, I) \quad (5)$$

Das zweite, neue adversarial Netzwerk erzwingt eine Multinomialverteilung für die Repräsentation der Label. Ähnlich wie bei der latenten Style-Information z , wird hier angenommen, dass die latente Klassenvariable von einer Multinomialverteilung y abstammt.

$$p(y) = \text{Cat}(y) \quad (6)$$

Ebenso wie bei dem ersten Adversarial Netzwerk versucht das Netzwerk die aggregierte Posterior-Verteilung von y mit der Multinomialverteilung des Encoders zu vereinen. Der Diskriminator versucht zwischen erzeugten und echten Labels zu unterscheiden [25].

Anders als bei einem Supervised bzw. Unsupervised AAE besitzt der Semi-Supervised AAE zusätzlich zur *Rekonstruktions- und Regularisierungsphase* eine dritte Trainingsphase, die sogenannten *Semi-Supervised Klassifizierungsphase*. Das Training wird mit Hilfe von Stochastic Gradient Descent (SGD) durchgeführt. In der ersten Phase, der Rekonstruktionsphase, wird der Rekonstruktionsfehler anhand der unbekannten Inputdaten minimiert. Hierfür wird der Encoder $q(z, y|x)$ und der Decoder aktualisiert. Darauf folgt die Regularisierungsphase, in der die Adversarial Netzwerke und somit die Diskriminatoren aktualisiert werden. Die zwei Diskriminatoren werden trainiert, um die mit Multinomialverteilung und Gauß-Verteilung erstellten, echten Samples von den durch den Autoencoder erstellten Samples der Code-Layer, zu unterscheiden. Danach werden die Generatoren verbessert, um die Diskriminatoren täuschen zu können. Final wird in der Semi-Supervised Klassifizierungsphase ein weiteres Mal der Encoder $q(y|x)$ trainiert, welcher nun als Klassifikator dient. Als Loss-Funktion wird die Cross-Entropy anhand eines nun bekannten Inputbatches minimiert [25].

3.2.5 Generative Adversarial Networks

GAN gehören ebenso wie die AAE zu der Klasse der Generativen Modelle. Der Outputschicht wird eine Datenverteilung $p_d(x)$ auf Pixelebene auferlegt [26]. Der Unterschied ist, dass die Datenverteilung bei AAE durch den Autoencoder festgelegt wird [25]. In Abbildung 16 wird der Aufbau eines GAN schematisch skizziert. Es besteht

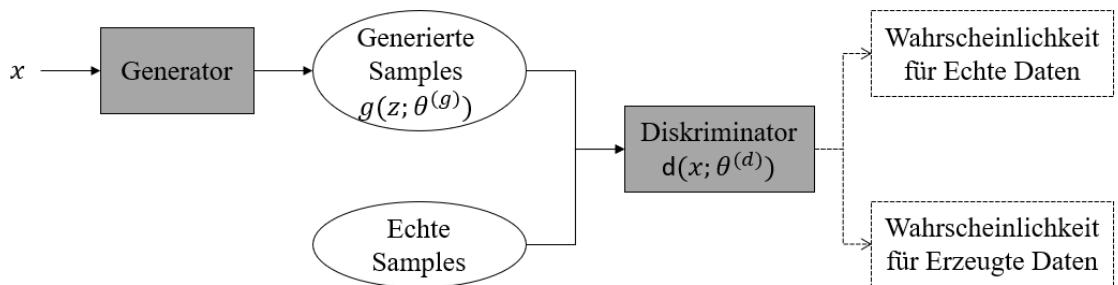


Abbildung 16: Skizzierter Aufbau eines Generative Adversarial Network

aus zwei neuronalen Netzen, welche gleichzeitig trainiert werden. Dabei übernimmt ein Netz die Aufgabe des Generators und das zweite die des Diskriminators. G bildet die Datenverteilung $p_d(x)$ ab und generiert dadurch ein neues Sample $x = g(z; \theta^{(g)})$. Der Diskriminator fungiert als Gegenspieler und versucht zwischen den erzeugten Samples und den echten Datenpunkten zu unterscheiden. Der Diskriminator berechnet mit $d(x; \theta^{(d)})$ wie wahrscheinlich es ist, dass es sich bei x um ein reales Sample handelt. Die Funktion $d(x; \theta^{(d)})$ gibt einen skalaren Wert aus. Daraus ergibt sich ein Spiel mit zwei Akteuren, die sich beide optimieren wollen. Der Generator versucht besser zu werden, um den Diskriminator zu täuschen. Im Gegensatz dazu versucht der Diskriminator, sich

nicht täuschen zu lassen. Mathematisch formuliert ergibt sich daraus ein MinMax-Spiel mit einer Value-Funktion $V(g, d)$ [10] [25] [26]:

$$\min_g \max_d V(d, g) = E_{x \sim p_{data}(x)}[\log(d(x))] + E_{z \sim p_z(z)}[\log(1 - d(g(z)))] \quad (7)$$

Die Wahrscheinlichkeit $p_z(z)$ gehört zu den Eingangsvariablen inklusive des Rauschens, welche mit $g(z; \theta^{(g)})$ abgebildet werden. Maximiert wird die Wahrscheinlichkeit, dass der Diskriminator den Samples das richtige Label zuordnet und der Generator wird gleichzeitig trainiert, um den Logarithmus der invertierten Wahrscheinlichkeiten der gefälschten Bilder $\log(1 - d(g(z)))$ zu minimieren [26]. Der Vorteil von GANs gegenüber anderen generativen Modellen wie zum Beispiel Markov Ketten ist, dass für die Gradientenberechnung Backpropagation verwendet werden kann und keine Inferenz während des Trainings benötigt wird. Des weiteren besitzen GANs die Fähigkeit, diskrete Verteilungen darzustellen [26]. Der Trainingsprozess besteht aus zwei Schritten. Im ersten Schritt wird der Diskriminator aktualisiert, was dazu führt, dass er sehr gut zwischen echten und generierten Datenpunkten unterscheiden kann. Anfangs wird es der Diskriminator bei der Unterscheidung der Daten sehr einfach haben, da die erzeugten Daten schlecht sind. Das führt zu einer Sättigung von $\log(1 - G(z))$, was zu sehr kleinen Gradienten führen kann und der Generator sich nicht verbessern kann. Um dies zu verhindern, kann der Generator nicht darauf trainiert werden $\log(1 - d(g(z)))$ zu minimieren, sondern $\log d(g(z))$ zu maximieren [26]. Im zweiten Schritt wird der Generator aktualisiert, um bessere, für den Diskriminator schwerer zu unterscheidende Daten zu generieren. Diese Änderung nimmt keinen Einfluss auf die Zwei-Spieler-Dynamik, bietet jedoch bessere Gradienten zu Beginn der Trainingsphase. Sollte der Diskriminator einen Wert von $\frac{1}{2}$ für jeden Datenpunkt ausgeben, so ist das Modell konvergiert und es kann nicht mehr zwischen generierten und realen Datenpunkten unterscheiden. Ist das der Fall, arbeitet der Diskriminator nicht optimal [26].

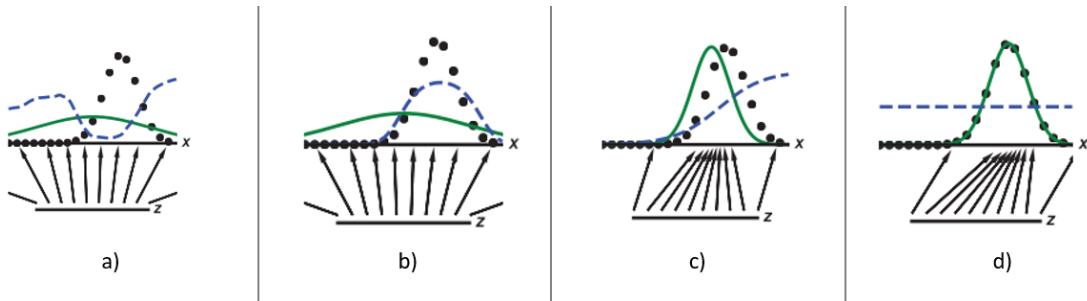


Abbildung 17: Darstellung des Trainingsprozesses von GANs [26] [20]

In Abbildung 17 wird der Trainingsprozess von GANs illustriert. Die blau gestrichelte Linie stellt die Diskriminatorenfunktion, die schwarz gepunktete Linie die Generatorfunktion und die grüne Linie die Modellverteilung p_{Modell} dar. Gefüllt wird eine 1-D Gaußsche Verteilung. In diesem Beispiel hat der Generator das Ziel, eine Skalierung der inversen Kummulativverteilungsfunktion der datenerzeugenden Verteilung

zu erlernen. Das wird erzielt, indem das GAN bei dem Training die Diskriminatorfunktion und die Generatorfunktion gleichzeitig aktualisiert, sodass der Diskriminator zwischen den Daten p_x , welche durch die datenerzeugende Verteilung generiert wurden, von denen der Modellverteilung p_{Modell} unterscheiden kann. Die untere, horizontale Linie stellt den Bereich dar, aus dem z erzeugt wird. Die obere Linie stellt einen Teil des Bereichs von x dar. Die gerichteten Pfeile zeigen das Mapping $x = G(z)$, welches die ungleichmäßige Verteilung p_{Modell} auf die transformierten Samples erzwingt. Die Generatorfunktion schlägt bei einer hohen Dichte von Samples aus und bei einer niedrigen Dichte verläuft die Kurve flacher. Die Kurve *a*) stellt das Modell bei der Initialisierung dar. Der Diskriminator ist zufällig initialisiert. Das Bild *b*) stellt den Fall dar, wenn die Diskriminatorfunktion konvergiert und der Generator auf einem fixen Wert gehalten wird. Dies ist in der Praxis nicht der Fall. Bei fixiertem Generator würde der Diskriminator zu $D(x) = \frac{p_{Data}(x)}{p_{Data}(x)+p_{Modell}(x)}$ konvergieren. Der dritte Zustand *c*) spiegelt das GAN, nachdem es eine Zeit lang trainiert wurde. Dabei wurden der Diskriminator sowie der Generator trainiert. Die durch den Generator erzeugten Datenpunkte befinden sich im Ausschlag der Diskriminatorfunktion. Das bedeutet, dass die erzeugten Daten mit höherer Wahrscheinlichkeit als echte Daten klassifiziert werden. Im letzten Zustand *d*) befindet sich das GAN im Nash Equilibrium und somit im Gleichgewicht. Dies führt dazu, dass der Diskriminator nicht mehr zwischen erzeugten und echten Daten unterscheiden kann und somit der Generator sich beim Generieren von Samples nicht verbessern kann. [20]. Der Begriff Nash Equilibrium stammt aus der Spieletheorie und beschreibt ein System, in dem sich beide Spieler durch eine einseitige Strategieveränderung nicht verbessern können [27]. Tritt ein Nash-Gleichgewicht ein, so konvergieren alle Spieler zu dem Zustand, der durch das Gleichgewicht repräsentiert wird. Folglich versucht jeder Spieler seine Nutzenfunktion zu maximieren [28].

4 Implementierung und Analyse der Adversarial Autoencoder

4.1 Grundidee der Arbeit

In der Anomalieerkennung gibt es verschiedene Ansätze der Problemlösung. Dabei beziehen sich die meisten Arbeiten auf Unsupervised Datensätze, jedoch sind in der Industrie Datensätze mit Semi-Supervised Eigenschaften üblich. In den letzten Jahren hat das Gebiet der Generativen Netzwerke an Popularität gewonnen. Generative Modelle ermöglichen es, Daten anhand bereits existierender Daten nachzubilden. Diese Eigenschaft kann in der Anomalieerkennung einen sehr positiven Einfluss haben, da im Semi-Supervised Fall bereits normale Datenpunkte und ein paar wenige Anomalien bekannt sind. Anhand der bereits vorhandenen Anomalien können neue Anomalien mit Hilfe eines GAN oder AAE erzeugt werden. Dies kann dabei helfen, die Anomalieerkennungsmethode robuster zu gestalten. Die Arbeit behandelt die Frage, ob eine Semi-Supervised Anomalieerkennung durch generative Netze verbessert werden kann und kann in vier Teile aufgeteilt werden:

1. Explorative Datenanalyse
2. Vergleich der drei AAE: Unsupervised, Supervised und Semi-Supervised
3. Generieren von Datenpunkten
4. Einfügen der generierten Daten in eine Anomalieerkennungsmethode

Im ersten Schritt wurde eine explorative Datenanalyse, anhand des MNIST-Datensatz durchgeführt. Die Daten wurden ausgewertet und auf ihre Eigenschaften untersucht. Anschließend wurde mit Hilfe von AAE-Modellen die Verteilung der Datenpunkte analysiert, um festzustellen in welchen Bereichen das Generieren von Daten sinnvoll ist und welche Variante der AAE diese Aufgabe am besten löst. Mit diesem Wissen werden im dritten Schritt Daten aus bestimmten Bereichen des latenten Raums generiert. Im letzten Schritt werden die generierten Daten in eine Anomalieerkennung eingefügt, um deren Auswirkung auf die Klassifizierung zu messen.

4.2 Explorative Datenanalyse

4.2.1 Der Datensatz

Die Daten für diese Arbeit liefert der MNIST-Datensatz. Dieser besteht aus handgeschriebenen Zahlen zwischen Null und Neun (siehe Abbildung 18) und besitzt somit zehn Klassen. Der MNIST-Datensatz besteht aus Datenpunkten des größeren



Abbildung 18: Beispiel jeder Zahl im MNIST-Datensatz [29]

Datensatzes NIST. MNIST enthält 70000 Bilder, von denen 60000 für das Training und 10000 für das Testen verwendet werden. Der MNIST-Testdatensatz besteht aus den

ersten 5000 Datenpunkten des NIST-Trainingdatensatzes und aus den letzten 5000 Datenpunkten des NIST-Testdatensatzes. Für die Validierung werden dem MNIST-Trainingsdatensatz 3000 Bilder entnommen und damit der Validierungsdatensatz erstellt. Dadurch ergibt sich die neue Aufteilung mit 57000 Trainingsdaten, 10000 Testdaten und 3000 Validierungsdaten. Die Datenpunkte haben eine feste Größe von 28x28 Pixeln mit Pixelwerten zwischen 0 und 255 [30].

4.2.2 Datenvorverarbeitung und Datenanalyse

Der Datensatz muss für die Verwendung vorverarbeitet werden. Nach dem Herunterladen des Datensatzes wird ein Bild durch ein Array der Größe (1, 28, 28) dargestellt. Die Labels bestehen aus einstelligen, binären Vektoren der Größe Zehn, die den Klassen Null bis Neun entsprechen. Für die AAE werden die Daten mit Hilfe der *reshape* Funktion aus der numpy-Bibliothek, in das Format (1, 784) konvertiert. Somit wird das Bild in ein 1d-Array umgewandelt und der Datensatz wird von einem Array der Form (60000, 28, 28) zu einem Array der Form (60000, 784) transformiert. Nun wird jeder Datenpunkt durch einen Vektor, welcher einer Zeile entspricht, repräsentiert. Dieser Schritt ist ebenfalls für die Matrixmultiplikation in den vollständig vernetzten Modellen wichtig. Anschließend werden die Pixelwerte in einen Bereich zwischen 0 und 1 normalisiert, indem jeder Wert durch 255 dividiert wird.

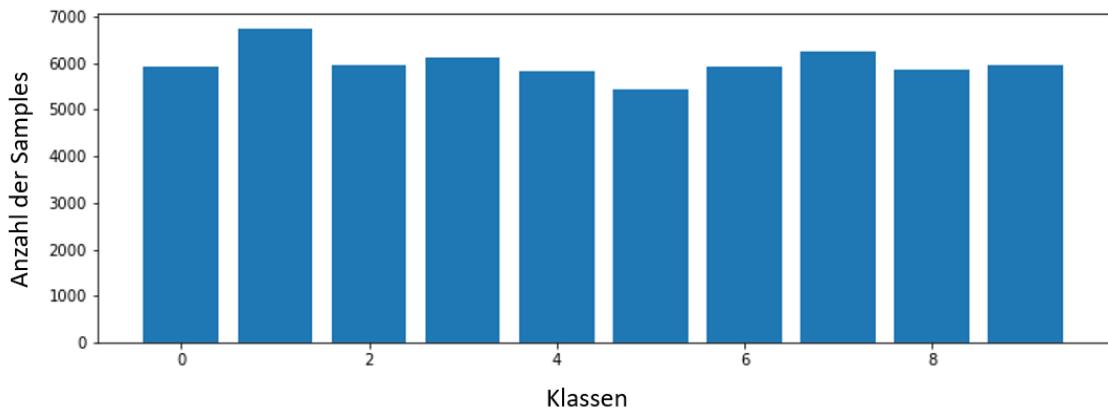


Abbildung 19: Verteilung der Daten für jede Klasse

In der Abbildung 19 wird die Datenverteilung der Klassen im Trainingsdatensatz dargestellt. Die genaue Anzahl von Datenpunkten der jeweiligen Klassen ist in Tabelle 1 gelistet. Zu beachten ist hierbei, dass nicht jede Klasse dieselbe Anzahl von Samples besitzt.

Dieselbe Analyse wurde für den Testdatensatz durchgeführt. Auch hier stehen nicht von jeder Klasse gleich viele Samples zur Verfügung. Die exakte Anzahl wird in der untersten Reihe der Tabelle 1 aufgelistet.

Klasse	0	1	2	3	4	5	6	7	8	9
Anzahl der Samples im Trainingdatensatz	5923	6742	5958	6131	5842	5421	5918	6265	5851	5949
Anzahl der Samples im Testdatensatz	980	1135	1032	982	892	958	1028	6265	974	1009

Tabelle 1: Verteilung des Training- und Testdatensatzes

4.2.3 Verarbeiten der Daten für Adversarial Autoencoder

In dieser Arbeit wurden drei Varianten des AAE implementiert: Unsupervised, Supervised und Semi-Supervised. Das hat zur Folge, dass der Datensatz in einem weiteren Schritt für das jeweilige Szenario verarbeitet werden muss. Als Grundlage hierfür diente der Data-Handler der Implementierung von $A^{(3)}$ von Philip Sperl, Jan-Philipp Schulze und Konstantin Böttinger [16].

Für den Unsupervised AAE sind keine Labels notwendig, weshalb für das Laden der Daten die Funktion `get_data_unsupervised()` (siehe Codesnippet 2) verwendet wurde. Hierbei werden die Daten geladen und vorverarbeitet. Die Labels werden jedoch vernachlässigt.

```

def get_data_unsupervised(self, data_split: str,
                           drop_classes: Union[List[int], List[str]] = None,
                           include_classes: Union[List[int], List[str]] = None
                           ) -> Tuple[np.ndarray, np.ndarray]:
    this_data = self._get_data_set(data_split=data_split)
    if include_classes:
        drop_classes = self.include_to_drop(include_classes)
    this_x = np.delete(this_data[0], np.where(np.isin(this_data[1],
                                                    drop_classes)), axis=0)
    return this_x, this_x

```

Code 2: Laden des Datensatzes für den Unsupervised AAE [16]

Die Funktion besitzt drei Parameter, welche für die Ausführung notwendig sind. Der erste Parameter `data_split` ist eine String-Variable, welche die Ausprägungen „train“, „test“ oder „val“ annehmen kann. Je nach Eingabe wird entweder der Trainings-, Test- oder Validierungsdatensatz bearbeitet. Der Parameter `drop_classes` kann verwendet werden, um nicht benötigte Klassen aus dem Datensatz zu löschen. Die Klassen müssen anhand einer Liste übergeben werden. Diese Möglichkeit ist für die später beschriebenen Experimente wichtig. Der Inputparameter `include_classes` besteht ebenfalls aus einer Liste von Integern und enthält alle Klassen, welche in dem Datensatz vorhanden sein sollen. Dieser Parameter hat Vorrang vor `drop_classes` und überschreibt diese, falls sich Klassen in beiden Parametern befinden. Als Output werden zwei Arrays mit den notwendigen Samples der Klassen ohne Labels zurückgegeben. Mit Hilfe dieser Funktion ist es möglich, den Datensatz mit den benötigten Klassen zu erstellen. Für den Fall des

Supervised AAE wird die obige Funktion leicht verändert. Die hierfür implementierte Funktion `get_data_supervised()` besitzt dieselben Inputparameter wie `get_data_unsupervised()`, gibt jedoch die gewünschten Datenpunkte sowie deren Labels aus. Zusätzlich müssen nicht nur die ungewünschten Datenpunkte, sondern auch die dazugehörigen Labels gelöscht werden.

```
if include_classes:
    drop_classes = self.include_to_drop(include_classes)
this_x = np.delete(this_data[0], np.where(np.isin(this_data[1], drop_classes)),
                  axis=0)
this_y = np.delete(this_data[1], np.where(np.isin(this_data[1], drop_classes)),
                  axis=0)
return this_x, this_y
```

Code 3: Änderungen für das Laden des Datensatzes für Supervised AAE [16]

Die Änderungen werden in Code 3 aufgezeigt. Mit diesen beiden Funktionen ist es möglich, Supervised und Unsupervised Datensätze mit den gewünschten Klassen zu erstellen. Für die Datenvorverarbeitung für den Semi-Supervised AAE wurde die Funktion `get_semisupervised_data()` implementiert. Diese erweitert die Funktionalität der Funktion `get_data_supervised()`, welche es ermöglicht eine bestimmte Anzahl von Anomalien in den Datensatz zu laden. Die restlichen, anomalen Datenpunkte werden entfernt. Das wird durch den Inputparameter `n_anomaly_samples` ermöglicht. Die Idee dahinter ist es, den Trainingsdatensatz mit einer bestimmten Anzahl von Anomalien zu füllen, jedoch beim Validieren die vollständige Anzahl von Anomalien zu verwenden. Die Funktionalität wird im Code 4 beschrieben.

```
if n_anomaly_samples is not None:
    idx_anom = np.where(this_y == 1)[0]
    idx_original = np.where(y_original == anomaly_classes)[0]
    n_delete = len(idx_anom) - n_anomaly_samples
    idx_delete = np.random.choice(idx_anom, size=n_delete, replace=False)
    n_delete_original = len(idx_original) - n_anomaly_samples
    original_delete = np.random.choice(idx_original, size=n_delete_original,
                                         replace=False)

    this_x = np.delete(this_x, idx_delete, axis=0)
    this_y = np.delete(this_y, idx_delete, axis=0)

    y_original = np.delete(y_original, original_delete, axis=0)
    assert np.sum(this_y) == n_anomaly_samples
```

Code 4: Limitieren der Anzahl von Anomalien [16]

Um diese Abfrage der Anomaliereduzierung ausführen zu können, müssen im Vorfeld die Labels der Daten zu normal und anomalous verändert werden. Hierfür wurde der

Inputparameter `anomaly_classes` eingeführt. Dieser bestimmt, welche Klassen als Anomalie definiert werden sollen. Entsprechend besitzen alle Anomalien das Label 1 und normale Datenpunkte das Label 0. Die Konvertierung in binäre Labels wird im Codeausschnitt 5 skizziert.

```
this_y[np.where(~np.isin(this_y, anomaly_classes))] = -1
this_y[np.where(np.isin(this_y, anomaly_classes))] = 0
this_y += 1
this_y = this_y.astype("uint8")
```

Code 5: Konvertieren der Labels in binäre Klassen [16]

Dieser Codeausschnitt stammt aus dem Paper A^3 [16] und wurde in diese Implementierung übernommen, da es die gewünschte Funktionalität mitbringt. Im ersten Schritt werden alle Labels, welche nicht einem Integer aus der Liste `anomaly_classes` entsprechen, auf -1 gesetzt. Danach werden die gewünschten anomalen Datenpunkten mit dem Label 0 versehen. Zum Schluss werden alle Labels mit eins addiert, wodurch die gewünschte Klassifizierung vollzogen wird.

Aus diesen drei Funktionen hat sich im Laufe des Projekts eine neue Funktion entwickelt, welche universell einsetzbar ist. Die Funktion erledigt folgende Aufgaben:

1. Laden der Daten in den gewünschten Split (Training, Test, Validieren)
2. Aussortieren der nicht benötigten Klassen
3. Umwandeln der Klassen in binäre Labels (Normal, Anomalie)
4. Speichern der Originallabels für eine bessere Visualisierung und Nachvollziehbarkeit
5. Reduzieren der Anzahl von Anomalien
6. Löschen von bestimmten Klassen und/oder Labels

Der sechste Punkt wurde implementiert, um bereits während des Trainings die vollständige Anzahl von anomalen Datenpunkten zu übergeben, ohne den Semi-Supervised Charakter zu verlieren. Beispielsweise wird ein Datensatz mit normalen Datenpunkten der Klasse 0 bis 5 und anomalen Datenpunkten 6 bis 9 erstellt. Mit der Funktionalität aus Punkt sechs ist es möglich, die Labels der Datenpunkte 8 und 9 zu entfernen. Diese repräsentieren unbekannte Anomalien in dem gewünschten Szenario.

Mit den oben beschriebenen Eigenschaften und der implementierten Methoden ist der Grundstein für das Entwickeln der AAE gelegt. Im nächsten Abschnitt wird die Implementierung der Modelle genauer beleuchtet.

4.3 Implementierung und Auswertung der Adversarial Autoencoder

Um zu überprüfen, ob sich die Verwendung generierter Daten positiv auf die Anomalieerkennung auswirkt, werden im letzten Schritt die erzeugten Daten in ein bestehendes Anomaly Detection Network eingefügt. Hierfür wurde A^3 von Philip Sperl, Jan-Philipp Schulze und Konstantin Böttinger [16] verwendet.

Die Modelle und Funktionen wurden alle in einem Python-Projekt mit folgenden Anforderungen implementiert:

- Tensorflow / Tensorflow-GPU 2.4.1
- Python 3.8 oder höher
- Numpy
- Matplotlib
- Keras
- Pandas
- scikit-learn

4.3.1 Implementierung der Modelle

Insgesamt wurden drei Modelle implementiert: Unsupervised, Supervised und Semi-Supervised Adversarial Autoencoder. Die Architektur der Modelle wurde dem Paper „Adversarial Autoencoders“ der Autoren Alireza Makhzani et al. [25] nachempfunden.

4.3.1.1 Aufbau der Adversarial Autoencoder

Die grundlegende Architektur des Unsupervised AAE wird in Abbildung 10 skizziert.

```
def create_encoder(self):  
    inputs = tf.keras.Input(shape=(self.image_size,))  
    x = tf.keras.layers.Dense(self.h_dim)(inputs)  
    x = tf.keras.layers.LeakyReLU()(x)  
    x = tf.keras.layers.Dropout(0.5)(x)  
  
    encoded = tf.keras.layers.Dense(self.z_dim)(x)  
    model = tf.keras.Model(inputs=inputs, outputs=encoded)  
    return model
```

Code 6: Encoder des Unsupervised und Supervised AAE

Für den Input des Encoders werden die Datenpunkte ohne Labels und mit der Inputgröße benötigt, welche dem Array des Bildes entspricht. Somit ist die Form des Inputs 784. Der gewählte Aufbau des Netzwerks ist simpel und besteht aus einer Dense-Layer, also einem vollständig verbundenem Hidden-Layer. Dieser Schicht folgt eine LeakyRectified Linear Unit (Leaky ReLu)-Aktivierungslayer und eine Dropoutschicht welche 50% der Neuronen ausschaltet. Die Leaky ReLu stammt von der ReLu-Aktivierungsfunktion ab. Sie ist jedoch für negative Werte nicht flach, sondern besitzt eine geringfügige Steigung. Der Vorteil der Leaky ReLu ist, dass kleine Gradienten erlaubt werden, auch wenn das Neuron nicht aktiv ist [31].

```
f(x) = alpha * x if x < 0
f(x) = x if x >= 0
```

Code 7: Funktionsweise LeakyReLU

Die Funktionsweise wird in Code 7 dargestellt. Wie der Name bereits verrät, kann durch α Information durchsickern und geht nicht wie bei der normalen ReLU verloren [31]. Grafisch werden die zwei Aktivierungsfunktionen in der Abbildung 20 verglichen.

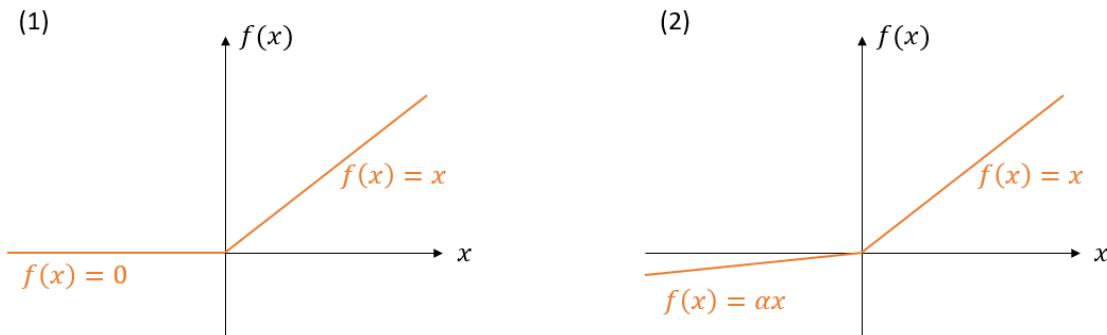


Abbildung 20: (1) Klassische ReLU-Aktivierungsfunktion,
(2) LeakyReLU-Aktivierungsfunktion

Bei der ReLU-Aktivierungsfunktion werden negative Gradienten gleich Null gesetzt, was zu dem Problem führt, dass sich diese Neuronen nicht erholen können. Somit ist das betroffene Neuron dauerhaft ausgeschaltet. Dieses Problem ist auch als *Dying ReLU Problem* bekannt [32]. Der Parameter α wurde in der Implementierung nicht angepasst, weshalb er standardmäßig den Wert 0.3 besitzt. Dieser Block, bestehend aus der Dense-, LeakyReLU- und Dropoutschicht, wird ein weiteres Mal wiederholt. Zum Schluss wird der Output durch eine Denselayer mit zwei Neuronen erzeugt. Dadurch wird die Datendimension auf 2D komprimiert, was für die Darstellung der Datenpunkte im latenten Raum notwendig ist. Durch die Dimensionsreduzierung lernt das Modell die Bilder anhand von zwei Feature einzuteilen. Der Autoencoder wird durch den Decoder vervollständigt. Dieser muss die gespiegelte, sonst jedoch identische Architektur des Encoders besitzen. Daraus resultiert folgender Aufbau wie in Code 8 dargestellt.

```
def create_decoder(self):
    encoded = tf.keras.Input(shape=(self.z_dim,))
    x = tf.keras.layers.Dense(self.h_dim)(encoded)
    x = tf.keras.layers.LeakyReLU()(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    reconstruction = tf.keras.layers.Dense(self.image_size,
                                           activation='sigmoid')(x)
    model = tf.keras.Model(inputs=encoded, outputs=reconstruction)
    return model
```

Code 8: Decoder des Unsupervised AAE

Bei dem Encoder wurden nur Datenpunkte ohne deren Labels verarbeitet, da diese bei einem Unsupervised-Modell nicht benötigt werden. Diese Information wird an ein Denselayer mit 1000 Neuronen übergeben und so die Dimension wieder vergrößert. Wie beim Encoder besteht ein Block aus Dense-, LeakyReLU- und Dropoutlayer. Die Hyperparameter wurden nicht verändert. Der Output des Decoders ist ein generiertes Bild, welches den Originaldatenpunkten idealerweise, sehr ähnlich ist. Aufgrund dessen muss der letzten Denselayer, welche den Output generiert, die Form des Originalbildes besitzen. Als Aktivierungsfunktion wurde die Sigmoidfunktion gewählt, welche Werte zwischen 0 und 1 annimmt. Da es nur zwei Klassen gibt, kann durch die Sigmoidfunktion die Wahrscheinlichkeit angegeben werden, mit der das generierte Bild zu der zugeordneten Klasse gehört. Die Kurve der Sigmoidfunktion $\delta(x) = \frac{1}{1+e^{-x}}$ wird in Abbildung 21 skizziert.

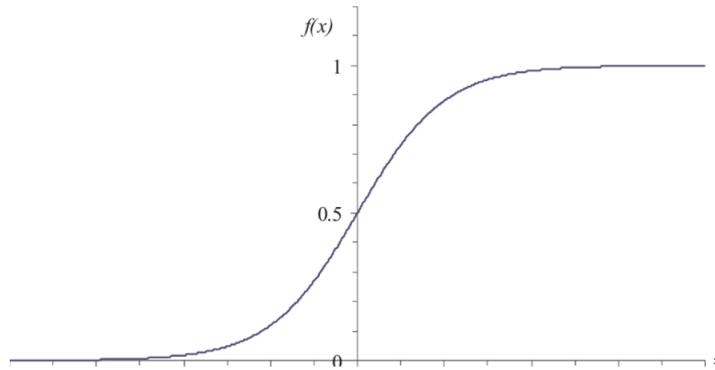


Abbildung 21: Sigmoidkurve

Damit aus dem bestehenden AE ein AAE entwickelt werden kann, muss der Diskriminatator implementiert und hinzugefügt werden.

```
def create_discriminator_style(self):
    encoded = tf.keras.Input(shape=(self.z_dim,))
    x = tf.keras.layers.Dense(self.h_dim)(encoded)
    x = tf.keras.layers.LeakyReLU()(x)
    x = tf.keras.layers.Dropout(0.5)(x)

    prediction = tf.keras.layers.Dense(1)(x)
    model = tf.keras.Model(inputs=encoded, outputs=prediction, name='disc_style')
    return model
```

Code 9: Diskriminatator des Unsupervised AAE

Bei dem Diskriminatator handelt es sich um ein vollständig verbundenes, neuronales Netz, welches als Input eine Wahrscheinlichkeitsverteilung erhält. Während des Trainings wird dem Diskriminatator ein zufällig ausgewählter Punkt aus einer Normalverteilung und der Output des Encoders übergeben. Der Aufbau ähnelt dem des Encoders stark, da es sich um die Struktur des Hauptblocks handelt. Die Anzahl der Neuronen und die Hyperparameter können frei angepasst werden. Als Output wird ein Zahlenwert

ausgegeben, welcher für die Verlustfunktion und das Training benötigt wird. Bevor erläutert wird, wie die einzelnen Komponenten zusammenarbeiten und der genaue Trainingsprozess aussieht, werden die Unterschiede zwischen dem Unsupervised Adversarial Autoencoders und den zwei Variationen Supervised und Semi-Supervised herausgearbeitet.

Für den Supervised Adversarial Autoencoder muss die Labelinformation in das System miteinbezogen werden. Der Input des Decoders wird hierfür um die Labelinformation erweitert, welcher als One-Hot-Encoded Vektor übergeben wird. Somit wird Zeile 2 von Code 8 durch die Zeile in Code 10 ersetzt.

```
encoded = tf.keras.Input(shape=(self.z_dim + self.labels,))
```

Code 10: Veränderung im Decoder für Supervised AAE

Somit erhält der Decoder die Informationen des Encoders sowie die der Labels, woraus sich in dieser Implementation die eindimensionale Inputarray (4,) ergibt. Der Grund hierfür ist, dass in dieser Arbeit mit zwei Klassen, Anomalie und Normal, gearbeitet wird. Würden stattdessen die Originallabels vom MNIST-Datensatz verwendet werden, addiert sich der Input aus dem Encoderoutput mit den zehn originalen Labels der Klassen zu dem eindimensionale Array (12,). Der restliche Aufbau ist identisch zu dem des Unsupervised Modells.

Für den Semi-Supervised AAE werden weitere Veränderungen benötigt. Die Architektur wird in der Abbildung 15 dargestellt. Diese zeigt, dass nur der Decoder des Supervised AAE wiederverwendet werden kann. Der Encoder (Code 11) wird bei dieser Variante zwei Mal für zwei unterschiedliche Aufgaben benötigt. Zuerst wird die klassische Aufgabe des AE erfüllt, die Minimierung des Rekonstruktionsfehlers. Anders als bei den beiden vorherigen Modellen, erzeugt der Encoder neben der Dimensionsreduzierung auch Labels für die erhaltenen Inputdaten. Hierfür wird eine Softmax-Schicht benötigt, welche je nach Anzahl der vorhandenen Klassen k, einen Vektor mit k-Ausgaben besitzt. Die Summe der k-Werte im Vektor ergibt immer 1 und die einzelnen Werte der Ausgabe können zwischen 0 und 1 liegen.

```
def create_encoder_semi(self):
    inputs = tf.keras.Input(shape=(self.image_size))
    x = tf.keras.layers.Dense(self.h_dim)(inputs)
    x = tf.keras.layers.LeakyReLU()(x)
    x = tf.keras.layers.Dropout(0.5)(x)

    encoded = tf.keras.layers.Dense(self.z_dim, name='output_z')(x)
    encoded_labels = tf.keras.layers.Dense(self.labels, name='output_label')(x)
    encoded_labels_softmax = tf.nn.softmax(encoded_labels)
    model = tf.keras.Model(inputs=inputs, outputs=[encoded, encoded_labels,
                                                   encoded_labels_softmax], name='Encoder')
    return model
```

Code 11: Encoder des Semi-Supervised AAE

Daraus ergeben sich zwei Outputs: die komprimierte Bildinformation z und die durch die Softmax-Schicht generierten Wahrscheinlichkeiten für das dazugehörige Label. Diese Informationen dienen als Input für den Decoder, welcher identisch mit dem des Supervised Modells ist. Anders als bei den bereits beschriebenen Modellen wird der Encoder hier auch für die Klassifizierung verwendet, weshalb es einen weiteren Output gibt. Der Encoder erhält für die Klassifizierung als Input die ursprünglichen Datenpunkte ohne Labels und teilt sie in die Klassen Normal oder Anomalie ein. Diese Einteilung wird im Training mit den Original-Labels verglichen, um den Klassifikationsfehler zu minimieren.

Der Semi-Supervised AAE besteht aus zwei Diskriminatoren, dem Diskriminator des Supervised und Unsupervised AAE, der für das Anpassen der Wahrscheinlichkeitsverteilung im latenten Raum zuständig ist und einem neuen Diskriminator, welcher für die Labels verantwortlich ist.

```
def create_discriminator_label(self, n_labels):
    encoded = tf.keras.Input(shape=(n_labels,))
    x = tf.keras.layers.Dense(self.h_dim)(encoded)
    x = tf.keras.layers.LeakyReLU()(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    x = tf.keras.layers.Dense(self.h_dim)(x)
    x = tf.keras.layers.LeakyReLU()(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    prediction = tf.keras.layers.Dense(1)(x)
    model = tf.keras.Model(inputs=encoded, outputs=prediction,
                           name='disc_labels')
    return model
```

Code 12: Diskriminator zur Verarbeitung der Labelinformation

Der Unterschied zu dem Diskriminator für die Style-Information liegt in der Form des Inputs. Die Style-Variable wird durch den zweidimensionalen latenten Raum dargestellt, somit besitzt der Input für den Diskriminator einen zweidimensionalen Vektor. Bei dem Diskriminator für Labels bildet sich der Input aus der Anzahl der Klassen k . Als Input wird ein k -dimensionaler Vektor übergeben, welcher für den Fall der Anomalieerkennung zweidimensional ist, da zwei Klassen existieren. Das bedeutet, dass die Inputform je nach Anzahl der Klassen angepasst werden muss.

4.3.1.2 Trainingsphase

Die Trainingsphase ist bei allen drei Modellen, bis auf kleine Unterschiede, identisch. Aufgrund dessen wird die Implementierung des Trainingsprozesses allgemein anhand des Unsupervised AAE erläutert und an den passenden Stellen auf die Unterschiede zu den zwei weiteren Modellen hingewiesen. Die einzelnen Modelkomponenten werden gemeinsam mit Stochastic Gradient Descent (SGD) in drei Schritten trainiert.

1. Rekonstruktionsphase

2. Regularisierungsphase
3. Semi-Supervised Klassifizierungsphase

Die dritte Phase findet nur bei den Semi-Supervised AAE Anwendung. Alle drei Abschnitte werden in der Funktion `training()` implementiert. Die Funktion besitzt als Input bei dem Unsupervised AAE nur den Batch mit den Bilddaten, wohingegen die Supervised und Semi-Supervised AAE als Input den Batch mit Bilddaten sowie einen weiteren Batch mit Labeldaten erhalten. Für den SGD wurde aus der TensorFlow-Bibliothek die Operation `GradientTape` verwendet. In der Rekonstruktionsphase wird der AE trainiert (Codebeispiel 13).

```
with tf.GradientTape() as ae_tape:
    encoder_output = encoder(batch_x, training=True)
    decoder_output = decoder(encoder_output, training=True)

    ae_loss = autoencoder_loss(batch_x, decoder_output, ae_loss_weight)

ae_grads = ae_tape.gradient(ae_loss, encoder.trainable_variables +
    decoder.trainable_variables)
ae_optimizer.apply_gradients(zip(ae_grads, encoder.trainable_variables +
    decoder.trainable_variables))
```

Code 13: Rekonstruktionsphase

Der zuvor erstellte Encoder erhält als Input die Datenpunkte und komprimiert die Bildinformation in einen zweidimensionalen, latenten Raum. Der Decoder übernimmt den erzeugten Output und verwendet diese Information ein neues Bild zu generieren. Das originale Inputbild wird mit dem generierten Bild verglichen, wobei das Modell versucht den Unterschied zu minimieren. Dadurch wird der Encoder gezwungen, immer bessere Merkmale zu extrahieren und der Decoder generiert daraus qualitativ hochwertigere Bilder. Der Verlust wird mit Hilfe der Mean squared error-Funktion der Keras Bibliothek berechnet.

```
def autoencoder_loss(inputs, reconstruction, loss_weight):
    return loss_weight * mse(inputs, reconstruction)
```

Code 14: Berechnung des Rekonstruktionsfehlers

Die Verwendung eines Gewichts in der Berechnung des Fehlers ermöglicht dessen Bedeutsamkeit während des Trainings zu beeinflussen und während der Hyperparameteroptimierung bessere Ergebnisse zu erzielen. Bei dem Supervised-Modell benötigt der Decoder die Informationen der Labels, weshalb der Encoder-Output mit den Originallabels zusammengeführt wird und anschließend an den Decoder übergeben wird. Der nächste Schritt ist das Trainieren des Diskriminators in der Regularisierungsphase (Code 15).

```

with tf.GradientTape() as dc_tape:
    real_distribution = tf.random.normal([batch_x.shape[0], z_dim], mean=0.0,
                                         stddev=1.0)
    encoder_output = encoder(batch_x, training=False)

    dc_real = discriminator(real_distribution, training=True)
    dc_fake = discriminator(encoder_output, training=True)

    dc_loss = discriminator_loss(dc_real, dc_fake, dc_loss_weight)

    dc_acc = accuracy(tf.concat([tf.ones_like(dc_real), tf.zeros_like(dc_fake)], axis=0),
                      tf.concat([dc_real, dc_fake], axis=0))

dc_grads = dc_tape.gradient(dc_loss, discriminator.trainable_variables)
dc_optimizer.apply_gradients(zip(dc_grads, discriminator.trainable_variables))

```

Code 15: Regularisierungsphase

Zunächst muss ein zufälliger Punkt im zweidimensionalen, latenten Raum generiert werden. Dieser Punkt wird mit Hilfe einer Normalverteilung erzeugt und dem Diskriminatator übergeben. Annahme ist hierbei, dass es sich hierbei um die echte Datenverteilung handelt. Der Encoder erstellt aus den Bilddaten die komprimierte Information, welche ebenfalls an den Diskriminatator übergeben wird. Die hier verwendete Normalverteilung kann mit verschiedenen Verteilungen ausgetauscht werden z.B.: mit einer Gaußschen Posterior-Verteilung. Die zwei Vorhersagen des Diskriminators werden miteinander verglichen und das Modell versucht den Unterschied der beiden Vorhersagen zu minimieren. So wird der latente Raum des Encoders an die Normalverteilung angeglichen. Dies geschieht durch die Funktion *discriminator_loss*.

```

def discriminator_loss(real_output, fake_output, loss_weight):
    loss_real = cross_entropy(tf.ones_like(real_output), real_output)
    loss_fake = cross_entropy(tf.zeros_like(fake_output), fake_output)
    return loss_weight * (loss_fake + loss_real)

```

Code 16: Berechnung des Regularisierungsfelhlers

Als Input benötigt diese Funktion beide Vorhersagen. Zuerst wird die Cross Entropy des generierten, latenten Raums und anschließend die des Encoders berechnet. Beide Werte werden zuerst addiert und danach mit einem Gewichtsparameter multipliziert.

Für das Verarbeiten von Supervised-Daten wird derselbe Trainingsschritt wie in Code 15 verwendet. Beim Trainingsschritt des Semi-Supervised Modells werden zwei anstatt eines Diskriminators benötigt. Zunächst muss eine kategoriale Verteilung erstellt werden, welche Zahlen zwischen 0 und 1 generiert, die für die bestehenden Klassen steht. Das Array mit den jeweils zufällig erstellten Klassen muss für die Übergabe in den Diskriminatator in einen One-Hot-Vektor konvertiert werden.

```

real_label_distribution = tf.experimental.numpy.random.randint(low=0,
    high=n_labels, size=batch_size)
real_label_distribution = tf.one_hot(real_label_distribution, n_labels)

dc_y_fake = discriminator_labels(encoder_softmax)
dc_y_real = discriminator_labels(real_label_distribution)

dc_y_loss = discriminator_loss(dc_y_real, dc_y_fake, dc_loss_weight)

dc_y_acc = accuracy(tf.concat([tf.ones_like(dc_y_real),
    tf.zeros_like(dc_y_fake)], axis=0),
    tf.concat([dc_y_real, dc_y_fake], axis=0))

dc_y_grads = dc_tape1.gradient(dc_y_loss,
    discriminator_labels.trainable_variables)
dc_optimizer.apply_gradients(zip(dc_y_grads,
    discriminator_labels.trainable_variables))

```

Code 17: Training des Diskriminators für die Labels

Die vom Encoder erstellten, sowie die generierten Labels werden an den Diskriminator übergeben und die Vorhersagen miteinander verglichen. Hierfür wird die Funktion aus Code 16 verwendet.

Zum Schluss muss der Encoder ein weiteres Mal trainiert werden, da er in jedem Modell auch als Generator fungiert. Hierfür wird in den Modellen mit Hilfe der Funktion 18 der Verlust berechnet.

```

def generator_loss(fake_output, loss_weight):
    return loss_weight * cross_entropy(tf.ones_like(fake_output), fake_output)

```

Code 18: Berechnung der Crossentropy des Generators

Dieser Schritt ist bei allen Variationen des AAE notwendig, jedoch enthält das Training des Semi-Supervised AAE zusätzlich den Semi-Supervised Klassifizierungsschritt (Code 19). Hierfür wird der Encoder mit den Original-Bilddaten nochmals trainiert. Die dabei generierten Labels werden mit den echten Labels der Bilder verglichen und die Softmax-Crossentropy zwischen den Labels berechnet. Das Modell versucht diesen Fehler zu minimieren.

```

with tf.GradientTape() as label_tape:
    encoder_z, encoder_y, _ = encoder_ae(batch_x, training=True)
    labels = tf.one_hot(batch_y, n_labels)
    l_loss = label_loss(labels, encoder_y, label_loss_weight)
label_grads = label_tape.gradient(l_loss, encoder_ae.trainable_variables)
label_optimizer.apply_gradients(zip(label_grads, encoder_ae.trainable_variables))

```

Code 19: Berechnen des Klassifikationsfehlers

4.3.2 Ergebnisanalyse der Adversarial Autoencoder

In diesem Abschnitt werden die Ergebnisse der Experimente ausgewertet. Für die Experimente wurde der MNIST-Datensatz und die vorher beschriebenen Variationen des AAAs verwendet. Die Versuche wurden auf einem Computer mit einer NVIDIA GeForce RTX 3070X und der AMD Ryzen 7 3700 CPU ausgeführt, sowie auf dem Server des Fraunhofer AISEC. Dieser besitzt fünf Nvidia Titan X Grafikkarten mit jeweils 12 GB RAM und zwei Xeon E5-2660 v3 CPUs.

4.3.2.1 Variablen und Hyperparameter

Für die Versuche wurden in allen drei Modellen dieselben Parameter sowie der gleiche Random Seed verwendet. Dies steigert die Vergleichbarkeit zwischen den Ergebnissen, da das Einteilen des Trainingsdatensatzes, sowie das Training gleich ablaufen. Für den Random Seed wurde der Wert 1993 gewählt. Der Batch wird aus 256 Datenpunkten erstellt. Die Buffergröße ist abhängig von der Anzahl der Datenpunkte im Trainingsdatensatz x_train . Die beiden Parameter, Batchgröße und Buffergröße, werden für das Mischen und Einteilen des Datensatzes bei dem Training der Modelle verwendet. Im ersten Schritt wird ein Datensatz aus den Inputdaten erstellt. Dabei wird der Datensatz durch TensorFlow transformiert, damit die Daten vorverarbeitet werden können. Zum Schluss iteriert die Funktion über den Datensatz und verarbeitet so alle Daten. Anschließend wird der erstellte Datensatz gemischt und in verschiedene Batches eingeteilt. Die gewählte Lernrate beträgt 0.00025 und darf maximal bis zum Wert 0.0025 steigen. Der Lernschritt wird anhand der Samples und der Batchgröße mit folgender Formel berechnet:

$$2 * np.ceil(n_samples/batch_size) \quad (8)$$

Dabei enthält der Parameter **n_samples** die Anzahl der Daten, die für das Training verwendet wurden. Trainiert wird über eine Länge von 500 Epochen.

4.3.2.2 Versuchsaufbau

Im ersten Schritt wurden die implementierten AAE verwendet, um deren Leistung nachvollziehbar zu messen und um zu veranschaulichen, in welchen Bereichen die Daten klassifiziert werden. Hierfür wurde der Semi-Supervised und der Unsupervised AAE verwendet. Der Supervised AAE wurde nach den ersten Testläufen verworfen, da die Ergebnisse schlechter als die des Unsupervised und Semi-Supervised AAE waren. Außerdem kommen in der Anomalieerkennung vollständig bekannte Datensätze selten vor. Falls der Supervised-Fall benötigt wird, kann durch die Zugabe von allen vorhandenen Labels aus der Semi-Supervised-Variante eine Supervised-Variante erstellt werden.

4.3.2.3 Experimente mit bekannten Datenpunkten

In der ersten Testreihe (Tabelle 2) werden jeweils nur zwei Klassen aus dem Datensatz verwendet. Hier wird die generelle Klassifizierungsleistung der Modelle geprüft und zugleich die Annahme überprüft, ob die Modelle stark unterschiedliche Datenpunkte besser voneinander unterscheiden können als sich sehr ähnelnde Punkte. Dieses Wissen kann später für das Generieren von Datenpunkten genutzt werden. Es gibt Auskunft, aus welchen Bereich des latenten Raums es eventuell sinnvoll ist Datenpunkte zu generieren. Für die Einteilung der Daten wird das implementierte Python Skript DataHandler.py verwendet. Mit dessen Hilfe werden die zuvor bestimmten Klassen extrahiert und als Anomalie bzw. normaler Datenpunkt gekennzeichnet. Da hier keine Datenpunkte

	Test 1	Test 2	Test 3	Referenz 1	Referenz 2
Normal	2	3	4	1	2
Anomalie	7	8	9	9	4
Unbekannt	-	-	-	-	-

Tabelle 2: Testreihe mit zwei bekannten Klassen

unbekannt sind, wird aus dem Semi-Supervised ein Supervised AAE. Um dieses Problem zu umgehen, wurden nur 50 und 1000 Anomalien für das Training bereitgestellt. Zur besseren Vergleichbarkeit wurden bei dem Unsupervised AAE ebenfalls nur 50 und 1000 Anomalien während des Trainingsprozesses verwendet. Für die Tests wurden zwei Klassen ausgesucht, welche sich in ihrer Darstellung ähneln. Als Referenz wurden zwei Klassen gewählt, die sich stark unterscheiden. Die Datenpunkte der Klasse 3 und 8 besitzen eine große Ähnlichkeit in ihrer Darstellung, was eine Klassifizierung erschwert. In der Abbildung 22 wird beispielhaft ein Datenpunkt der jeweiligen Klassen aus Test 2 abgebildet.



Abbildung 22: Vergleich der Datenpunkte 3 und 8

Da der Encoder die Daten in den zweidimensionalen, latenten Raum komprimiert, lernen die Modelle zwei Features. Bei den Klassen aus Abbildung 22 besteht z.B. ein Feature aus zwei übereinanderliegenden Kreisen. Es wird angenommen, dass diese zwei Merkmale von dem Modell gelernt werden, was zu einer schlechten Unterscheidbarkeit der Punkte führt. Dasselbe gilt für den Test 1 und 3. Die Klassen 4 und 9 bestehen beide aus einer runden, geschlossenen Form oben und einer Linie welche von der rechten Seite der geschlossenen Form nach unten geht. Für Referenzversuche wurden zwei vermeidlich

unterschiedliche Klassen gewählt. Bei dem ersten Referenztest wurde die Datenpunkte der Klassen 1 und 9 ausgewählt, da die 9 eine geschlossene Form besitzt und die 1 im Gegensatz dazu nur aus einem vertikalen Strich besteht. Der zweite Referenztest wird mit den Klassen 2 und 9 ausgeführt. Es wird erwartet, dass beide Modelle keine Schwierigkeiten bei der Klassifizierung der Datenpunkte haben. Durch die zusätzliche Information, um welche Klasse es sich während des Trainings handelt, wird eine bessere Performance vom Semi-Supervised Modell erwartet. Die Referenzversuche, also die Klassen mit großem, strukturellem Unterschied, sollten durch beide Modellen sehr gut klassifiziert und im latenten Raum getrennt werden. Bei den Tests 1 bis 3 wird erwartet, dass beide keine gute Trennbarkeit der Datenpunkte erzielen.

Analyse der ersten Testreihe

Für die Auswertung der Ergebnisse wurde das trainierte Modell anhand des Validierungdatensatzes getestet. Es wird der latente Raum des Encoders der Modelle analysiert. In der Analyse werden ein Referenztest und ein normaler Test ausgewertet. Zunächst wird das Ergebnis des ersten Referenztests „Referenz 1“ betrachtet. In der Abbildung 23 wird der latente Raum des Unsupervised AAE mit 50 Anomalien während des Trainings dargestellt.

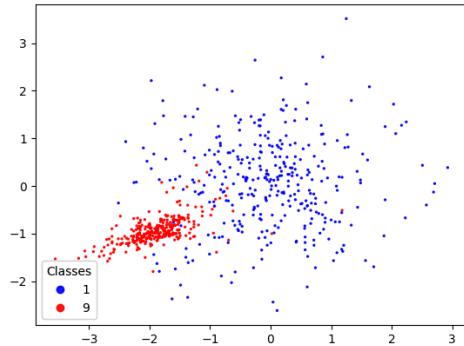


Abbildung 23: Referenz 1 - Latenter Raum eines Unsupervised AAE mit 50 Anomalien

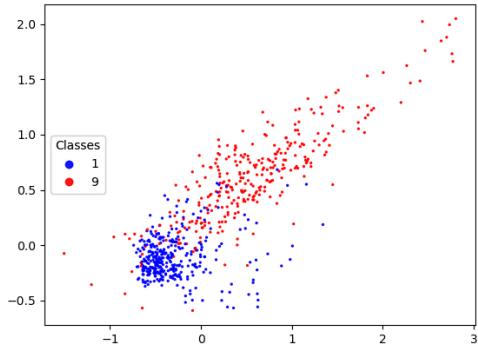


Abbildung 24: Referenz 1 - Latenter Raum eines Semi-Supervised AAE mit 50 Anomalien

Bereits mit einer kleinen Menge von anomalen Datenpunkten lernt das Modell gut zwischen den zwei Klassen zu unterscheiden. Im Vergleich dazu wird in Abbildung 24 das Ergebnis des Semi-Supervised AAE abgebildet. Hier werden ebenfalls die Datenpunkte gut voneinander getrennt, jedoch gibt es eine Überschneidung im Bereich $x = 0$ und $y = 0$. Auffällig ist, dass in dem latenten Raum (Abb. 24) die Punkte entlang einer diagonalen Linie liegen und nicht die vorgegebene Normalverteilung annehmen. Ein Grund hierfür könnte das Komprimieren der Informationen auf zwei Merkmale sein. Diese haben eventuell einen linearen Zusammenhang, welcher stärker ist als die Normalverteilung des Diskriminators. Diese Beobachtung trifft nicht bei dem Unsupervised AAE zu. Hier verteilen sich die Daten stärker um einen Mittelpunkt. Eine

Erklärung dafür könnte das Lernen von unterschiedlichen Merkmalen sein, da dies nicht von außen beeinflusst werden kann. Ebenfalls kann anhand des latenten Raums beobachtet werden, dass die bekannten Punkte der Klasse 1 des Unsupervised AAE die aufgezwungene Normalverteilung annimmt. Die unbekannten Punkte der Klasse 9 hingegen folgen visuell keiner Normalverteilung.

Um die Leistung bei sich ähnelnden Klassen zu untersuchen, wird der Test 2 betrachtet. Es ist deutlich erkennbar, dass beide Modelle nicht zwischen den zwei Klassen unterscheiden können. Bei der geringeren Anzahl an anomalen Datenpunkten, hier

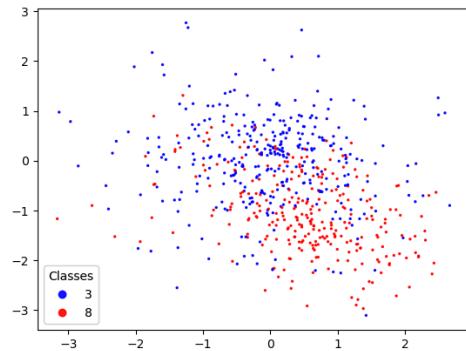


Abbildung 25: Test 2 - Latenter Raum eines Unsupervised AAE mit 50 Anomalien

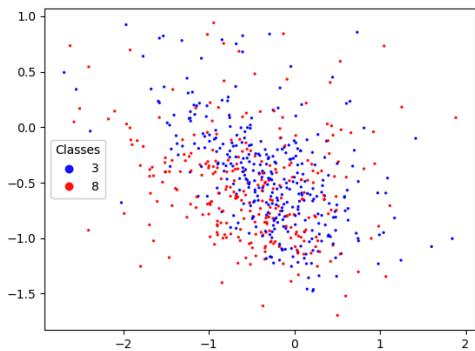


Abbildung 26: Test 2 - Latenter Raum eines Semi-Supervised AAE mit 50 Anomalien

repräsentiert durch die Klasse 8, können beide Modelle nicht die klassenspezifischen Merkmale erlernen. Die Ähnlichkeit der Klassenmerkmale ist zu stark. Entgegen der Erwartung liefert das Semi-Supervised AAE ebenfalls keine gute Klassifizierung, obwohl die Labels der anomalen Klasse vorhanden sind.

Um zu kontrollieren, ob die Leistung durch die Zugabe von mehr Anomalien während des Trainings steigt, wurden die Modelle mit 1000 Anomalien trainiert.

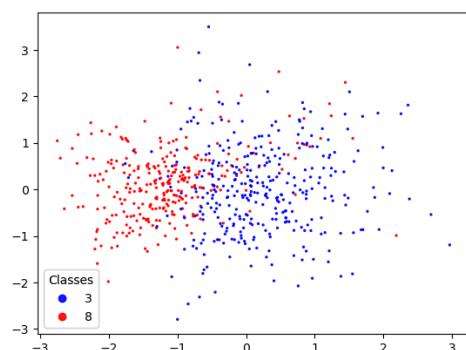


Abbildung 27: Test 2 - Latenter Raum eines Unsupervised AAE mit 1000 Anomalien

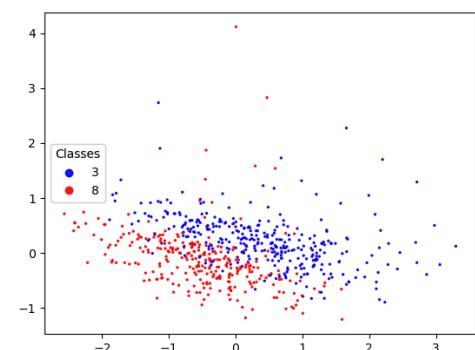


Abbildung 28: Test 2 - Latenter Raum eines Semi-Supervised AAE mit 1000 Anomalien

Es ist zu erwarten, dass mit einer größeren Menge von Daten die Samples besser voneinander getrennt werden. Durch mehr Daten können die Modelle bessere Merkmale extrahieren und so unbekannte Punkte der richtigen Klasse zuordnen. Die Annahme wird durch beide Modelle bestätigt. Die Anomalien werden nun deutlich besser von den normalen Daten getrennt als bei dem Testlauf mit 50 anomalen Samples. Allerdings führt die Information der Labels im Semi-Supervised Modell (Abb. 28) zu keiner sichtlichen Verbesserung gegenüber des Unsupervised Modells (Abb. 27). Beiden Modellen wird wie in den anderen Tests, die Nominalverteilung durch den Diskriminatator aufgezwungen. Diese wird durch den Unsupervised AAE besser gelernt als bei dem Semi Supervised AAE. Die Samples liegen bei beiden Modellen um den vorgegebenen Mittelpunkt $x = 0$ und $y = 0$, jedoch nimmt bei dem Semi Supervised Modell die gelernte Verteilung der Labels einen Einfluss auf die Normalverteilung, was sich auf die Streuung der Punkte auswirkt. Bei dem Unsupervised AAE nehmen auch die Anomalien, die in Abbildung 25 und 27 dargestellt durch die Klasse 8 dargestellten Anomalien die Form der Normalverteilung an.

4.3.2.4 Experimente mit bekannten, normalen und unbekannten, anomalen Datenpunkten

Um die Performance der Modelle besser vergleichen zu können, werden im zweiten Experiment die Klassenpaare aus der ersten Testreihe übernommen. Der Unterschied ist, dass die als Anomalie gekennzeichneten Datenpunkte unbekannt sind. Das bedeutet, während des Trainings werden nur die Datenpunkte mit der Bezeichnung normal verwendet und bei dem Testen und Validieren werden die unbekannten Samples hinzugefügt. Die Modelle lernen nur die Merkmale der normalen Daten, was die Klassifizierung der Anomalien erschwert.

	Test 4	Test 5	Test 6	Referenz 3	Referenz 4
Normal	2	3	4	1	2
Anomalie	-	-	-	-	-
Unbekannt	7	8	9	9	4

Tabelle 3: Testreihe mit einer bekannten und einer unbekannten Klasse

Wie in der ersten Testreihe wird erwartet, dass die Referenztests besser klassifiziert werden, da die Struktur der Datenpunkte unterschiedlich ist. Bei dem Test 5 wird die Klasse 3 als normal bezeichnet und die unbekannten Anomalien gehören der Klasse 8 an. Durch die Ähnlichkeit der Merkmale ist zu erwarten, dass die Modelle keine sinnvolle Einteilung der Samples finden. Im Referenztest 3 sollten die Klassen besser voneinander getrennt werden, da sich die Merkmale deutlich unterscheiden. Weil im Training nur eine Klasse im Training vorhanden ist, wird hier ebenfalls eine Verschlechterung gegenüber der ersten Testreihe angenommen.

Analyse der zweiten Experimentreihe

Da keine Anomalien während des Trainings verwendet werden, wird die Anzahl der

Trainingssamples nicht reduziert. Der Grundaufbau der Versuche ist ein Semi-Supervised Szenario. Zunächst wird der Referenzversuch 3 betrachtet. In Abbildung 29 wird der

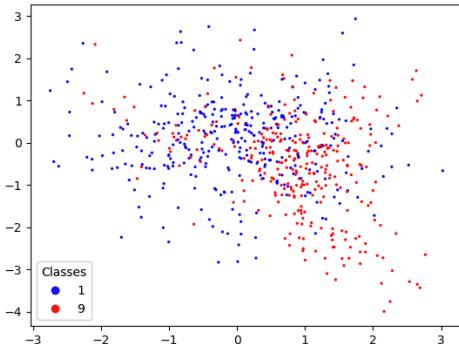


Abbildung 29: Referenz 3 - Latenter Raum des Unsupervised AAE

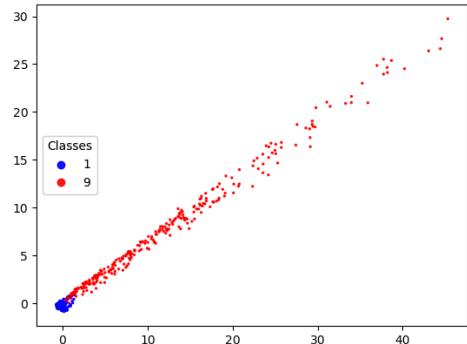


Abbildung 30: Referenz 3 - Latenter Raum des Semi-Supervised AAE

latente Raum des Unsupervised AAE dargestellt. Trotz der unterschiedlichen Struktur der Samples trennt der Encoder die Daten nicht gut voneinander. Die Klassen besitzen jeweils ein eigenes Zentrum, was dazu führt, dass sich die Samples der Klasse 1 auf x-Achse im negativen Bereich befinden und die der Klasse 9 auf der positiven Seite. Die Streuung der Punkte ist jedoch sehr groß, was zu der Vermischung der Samples führt. Im Gegensatz dazu, ist das Semi-Supervised AAE in der Lage, die Samples voneinander zu unterscheiden. In der Abbildung 30 liegen die Samples entlang einer diagonalen Linie. Die Datenpunkte der Klasse 1, also der normalen Datenpunkte, liegen gruppiert im Bereich von $x = 0$ und $y = 0$. Diese Klasse war während des Trainings bekannt. Das Modell lernt zwei Merkmale der Klasse 1 und bildet anhand dieser die Samples im zweidimensionalen Raum ab. Die Samples der Klasse 9 werden ebenfalls auf zwei Merkmale komprimiert. Anhand der linearen Verteilung der Punkte aus Klasse 9 ist davon auszugehen, dass diese einen linearen Zusammenhang mit den Merkmalen der Klasse 1 haben. Dadurch ist es dem Modell möglich, die Anomalien von den normalen Samples zu trennen, obwohl sie vollständig unbekannt sind.

Bei dem Test 5 konnten beide Modelle die Punkte nicht unterscheiden. Der Unsupervised AAE (siehe Abb. 31) bildet die unbekannten Anomalien in den Bereich der normalen Samples ab. Dasselbe gilt für den Semi-Supervised AAE (siehe Abb. 32). Dieser kann die Datenpunkte nicht wie bei dem Referenzversuch 3 trennen, sondern klassifiziert unbekannte Punkte zu den bereits bekannten Samples. Aufgrund der Verwandtschaft der klassenspezifischen Merkmale wird davon ausgegangen, dass die unbekannten Punkte zu der Klasse eingeteilt werden, welche dem Sample am ähnlichsten sind. In beiden Fällen liegen die Punkte um den vorgegebenen Mittelpunkt der Normalverteilung. Unterscheiden sich die Punkte jedoch stark voneinander, so kann das Semi-Supervised AAE die Klassen voneinander trennen.

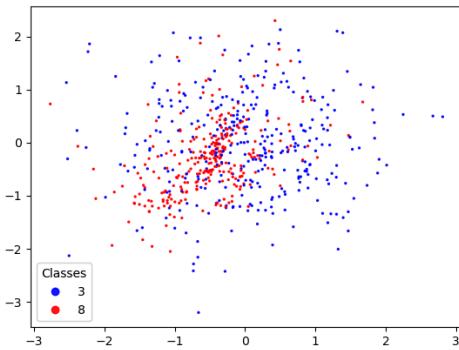


Abbildung 31: Test 5 - Latenter Raum des Unsupervised AAE

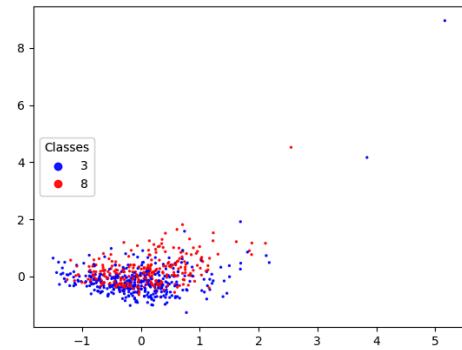


Abbildung 32: Test 5 - Latenter Raum des Semi-Supervised AAE

4.3.2.5 Experimente mit Semi-Supervised Anomalieerkennungscharakter

Um die aufgestellte These aus der zweiten Testreihe zu überprüfen, werden im nächsten Schritt die Semi-Supervised Eigenschaften mit reduziertem Datensatz getestet. Es wird davon ausgegangen, dass unbekannte Samples der sich am stärksten ähnelnden Klasse zugeordnet werden. Um dies zu überprüfen, werden vier Szenarien getestet. Die Klassen 1 und 9 werden in diesen Tests als Grundbaustein genutzt, da aus den vorherigen Tests deutlich wurde, dass die Klassen gut unterscheidbar sind. In den Tests werden die normalen Samples von der Klasse 1 und die Anomalien von der Klasse 9 repräsentiert. Als unbekannte Anomalie wurden die Klassen 8 und 9 definiert, da sie der anomalen Klasse ähnlich sind. Im Test 9 wird die unbekannte Anomalie durch die Klasse 7 dargestellt. Sie ähnelt eher der Klasse 1 da sie keine Rundungen besitzt. Die Einteilung wird in der Tabelle 4 zusammengefasst.

	Test 7	Test 8	Test 9
Normal	1	1	1
Anomalie	9	9	9
Unbekannt	8	4	7

Tabelle 4: Testreihe mit Semi-Supervised Datensatz

Anhand der vorherigen Tests wird erwartet, dass in den Tests 7 und 8 beide Modelle die unbekannten Samples als Anomalie klassifizieren. Die Merkmale ähneln sich stärker als die der normalen Klasse. Bei dem Test 9 wird vom umgekehrten Ergebnis ausgegangen, also dass die Modelle die unbekannte Klasse als normal klassifizieren. Da das Semi-Supervised Modell zusätzlich die Labels im Training bearbeitet, wird von diesem Modell die bessere Klassifizierung erwartet.

Analyse der reduzierten Semi-Supervised Experimente

Als erstes wird der Test 7 ausgewertet. Hierfür werden die zwei latenten Räume aus Abbildung 33 und 34 betrachtet. Sowohl das Unsupervised AAE als auch das Semi-Supervised AAE bestätigen die Hypothese, dass unbekannte Datenpunkte zu der

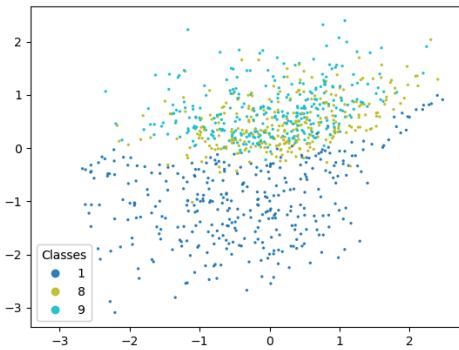


Abbildung 33: Test 7 - Latenter Raum des
Unsupervised AAE

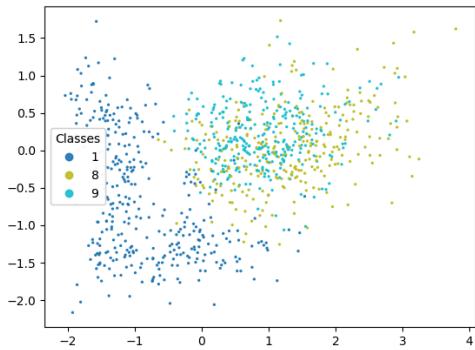


Abbildung 34: Test 7 - Latenter Raum des
Semi-Supervised AAE

sich am stärksten ähnelnden Klasse zugeordnet werden. In den Abbildungen werden die normalen Datenpunkte blau, die bekannte Anomalien türkis und die unbekannten Punkte Grün dargestellt. Die normalen Samples werden gut von den Anomalien getrennt. Unbekannte Punkte werden nicht als eigenes Cluster abgebildet, sondern, wie erwartet, in den gleichen Bereich eingeteilt. Somit wird die Hypothese in diesem Fall für beide Modelle bestätigt. Unbekannte Punkte, die einen großen Teil der Merkmale teilen, werden zu dieser Klasse sortiert. Ein signifikanter Unterschied zwischen der Performance beider Modelle kann anhand der Diagramme nicht erkannt werden.

Die erzielten Ergebnisse aus dem Test 7 werden ebenfalls für Test 8 erwartet. Die Erwartungen wurden durch beide Modelle bestätigt. In den Abbildungen 35 und 36 sind die normalen Samples blau, die bekannte Anomalien türkis und die unbekannten Datenpunkte Rot markiert. Der Unsupervised AAE trennt die normalen Samples von den Anomalien und klassifiziert die unbekannten Punkte in den Bereich der bekannten Anomalien.

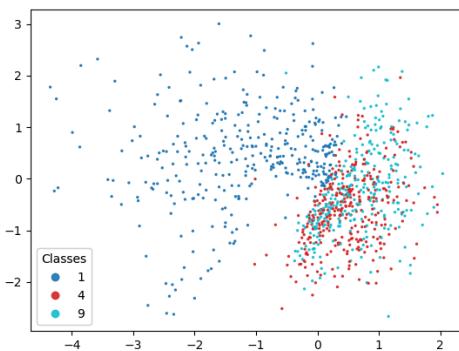


Abbildung 35: Test 8 - Latenter Raum
des Unsupervised AAE

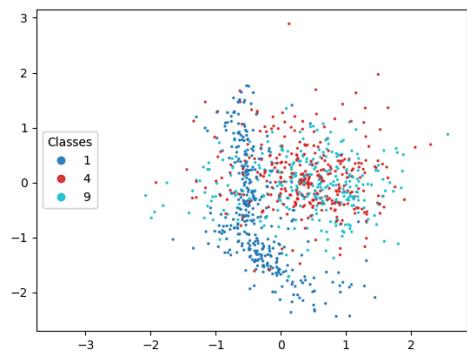


Abbildung 36: Test 8 - Latenter Raum
des Semi-Supervised AAE

Das Modell sortiert die unbekannten Punkte der sich am stärksten ähnelnden, bekannten Klasse zu, was die Hypothese bestätigt. Jedoch ist die Trennung zwischen anomalen und

normalen Samples schlechter geworden. Dasselbe konnte im Semi-Supervised Modell beobachtet werden. Das Semi-Supervised Modell führt die Klassifizierung deutlich schlechter durch. Zwar wird die Hypothese bestätigt, indem die unbekannte Klasse 4 der Klasse 9 zugeordnet wird, aber es gibt einen großen Bereich, um die Punkte $x = 0$ und $y = 0$, in dem alle drei Klassen vorkommen. Anders als bei der unbekannten Klasse 7, aus dem Test 7, leidet, unter der Zugabe der Samples, die generelle Klassifizierung beider Modelle. Das Cluster der Klasse 1 besitzt eine gleichartige Form wie in Test 7. Die bekannten Samples der Klasse 9 werden hingegen nicht mehr gut von den bekannten, normalen Datenpunkten unterschieden. Im Test 9 werden die normalen Samples blau und die anomalen Samples türkis abgebildet. Die unbekannten Punkte sind grau gekennzeichnet. Entgegen der Erwartung teilen beide Modelle die unbekannte Klasse der Anomalie zu (siehe Abb. 37 und 38). Der latente Raum des Unsupervised AAE trennt

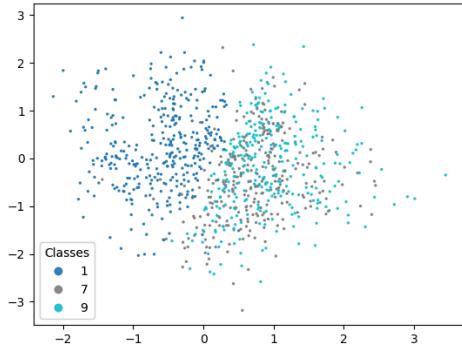


Abbildung 37: Test 9 - Latenter Raum des Unsupervised AAE

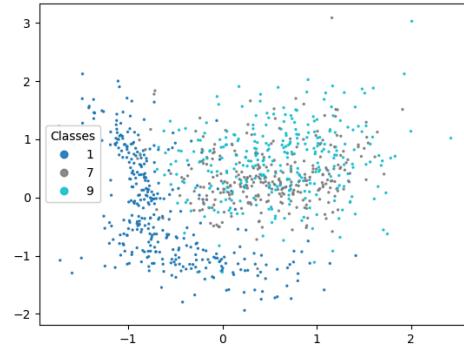


Abbildung 38: Test 9 - Latenter Raum des Semi-Supervised AAE

die Samples sehr sauber und es gibt im Gegensatz zum latenten Raum des Semi-Supervised Modells keine Ausreißer. Eine Erklärung für die Einteilung der Klasse 7 in die Klasse 9 kann die Struktur der Samples sein. Entgegen der Annahme, dass die Samples der Klasse 7 der Klasse 1 verwandt sind, kann durch Betrachten der einzelnen Datenpunkte auf das Gegenteil geschlossen werden. Die Samples der Klasse 1 (Abb. 39) bestehen aus einer diagonalen Linie von rechts oben nach links unten.

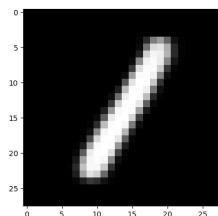


Abbildung 39: Samples der Klasse 1

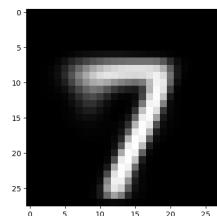


Abbildung 40: Samples der Klasse 7

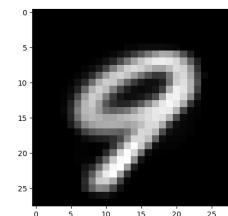


Abbildung 41: Samples der Klasse 9

Die Klassen 7 (Abb. 40) und 9 (Abb. 41) besitzen ebenfalls dieses Merkmal. Als zweites Merkmal gibt es bei den Klassen 7 und 9, am oberen Ende der Linie eine weitere Struktur, welche links von der linearen Struktur auftritt. Dieses Feature kommt bei der Klasse 1 nicht vor. Aufgrund dieser Eigenschaft wird angenommen, dass die Klasse 7 nicht den normalen Datenpunkten zugeordnet wird.

4.3.2.6 Experimente mit vollständigem Datensatz

Nach den Experimenten mit limitierter Klassenanzahl wird im nächsten Schritt der komplette Datensatz verwendet. Dieser bietet erschwerende Bedingungen, da deutlich mehr Klassen und Datenpunkte vorhanden sind. Die hier ausgewählten Tests spiegeln einen Semi-Supervised Fall am besten wider, da ein großer Teil des Datensatzes aus normalen, vollständig bekannten Samples besteht und dazu wenige bekannte Anomalien hinzugefügt werden. Insgesamt werden vier Tests durchgeführt. In den Tests 10 und 11 bestehen die normalen Samples aus den Klassen 0 bis 5. Als Anomalien wurden einmal die Klassen 6 und 7 definiert und als unbekannte Samples die Klassen 8 und 9. Im Test 11 werden die Unbekannten mit den Anomalien getauscht, um zu überprüfen, ob dies die Klassifizierung verändert. In den Tests 12 und 13 werden die Klassen 4 bis 9 als normal

	Test 10	Test 11	Test 12	Test 13
Normal	0 - 5	0 - 5	4 - 9	4 - 9
Anomalie	6, 7	8, 9	2, 3	0, 1
Unbekannt	8, 9	6, 7	0, 1	2, 3

Tabelle 5: Testreihe mit vollständigem Datensatz

definiert. Test 12 trainiert mit den Datenpunkten der Klassen 2 und 3 als Anomalien. Die Samples der Klassen 0 und 1 bleiben dem Trainingsprozess verborgen. Auch hier werden, wie in den anderen beiden Tests, die bekannten Anomalien und die unbekannten Samples gewechselt, um zu untersuchen, ob die unterschiedlichen Merkmale zu einer unterschiedlichen Klassifizierung führen. Damit die Hypothese bestätigt wird muss angenommen werden, dass die unbekannten Daten zu den Klassen zugeordnet werden, die sie sich am stärksten ähneln. Somit wird erwartet, dass die Datenpunkte der Klasse 8 nahe der Klasse 3 klassifiziert werden oder beispielsweise die Samples der Klasse 2 nahe des Bereichs der Klasse 7 liegen.

Analyse der Experimente mit vollständigem Datensatz

Wie in den vorherigen Tests gibt es zwei Klassen, anomalous und normal. Die originalen Labels wurden jedoch während des Trainings gespeichert und bei der Visualisierung verwendet, um nachvollziehen zu können, in welchem Bereich die jeweiligen Klassen liegen. Als erstes wird der Test 10 analysiert. Der Unsupervised AAE (Abb. 42) lernt die Merkmale von den bekannten Samples und bündelt jede Klasse gut voneinander getrennt, jedoch liegen die als normal gekennzeichneten Samples in einer großen Ballung zusammen. Im latenten Raum wird jede Klasse für sich abgebildet. Der Unsupervised AAE erhält zu keinem Zeitpunkt die Labelinformation und kann auf dieses Wissen nicht

beim Lernen von Merkmalen zurückgreifen. Somit ist eine Gruppierung der normalen und anomalen Samples nicht möglich. Die zwei unbekannten Klassen werden wie erwartet, einer gleichartigen Klasse zugeordnet. Samples der Klasse 9 liegen im Bereich der Klassen 7 und 4. In den vorherigen Versuchen konnte bereits analysiert werden, dass zwischen diesen Datenpunkten eine große Ähnlichkeit herrscht. Die Datenpunkte der Klasse 8 liegen wie prognostiziert, im Cluster der Klasse 3.

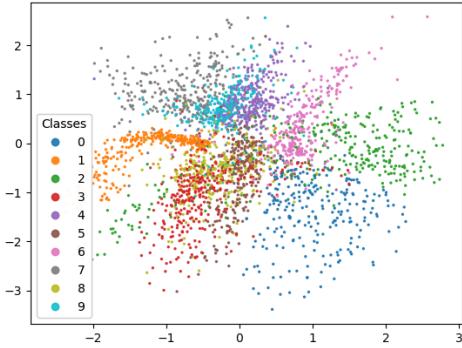


Abbildung 42: Test 10 - Latenter Raum des Unsupervised AAE

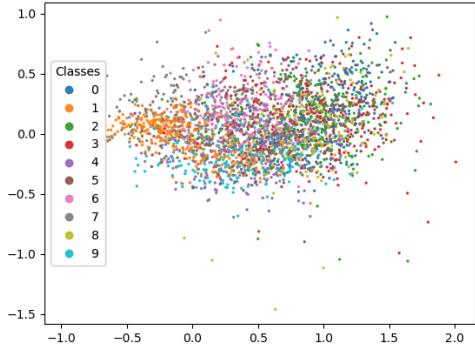


Abbildung 43: Test 10 - Latenter Raum des Semi-Supervised AAE

Demgemäß kann das Unsupervised AAE unbekannte Datenpunkte nicht der anomalen Klasse zuordnen, außer diese ähneln den bekannten Anomalien. Der Semi-Supervised

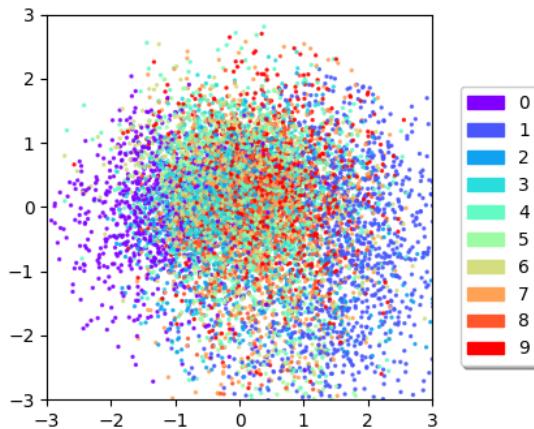


Abbildung 44: Zweidimensionaler latenter Raum eines Supervised AAE auf dem MNIST-Datensatz [25]

AAE (Abb. 43) liefert in dieser Implementierung keine sinnvollen Ergebnisse. Es wird ebenso wenig zwischen den Klassen Anomalie und normaler Datenpunkte unterschieden, wie zwischen den einzelnen Klassen. Trotz der Information zu welcher Klasse das jeweilige Sample gehört, wurde keine sinnvolle Klassifizierung erreicht. Der latente Raum des Semi-Supervised AAE ist vergleichbar mit dem des Supervised AAE (siehe Abb. 44). Es wird der Anschein erweckt, dass die Information, um welche Klasse es sich handelt, die Leistung der Adversarial Autoencoder bei großen Datenmengen verschlechtert. Der

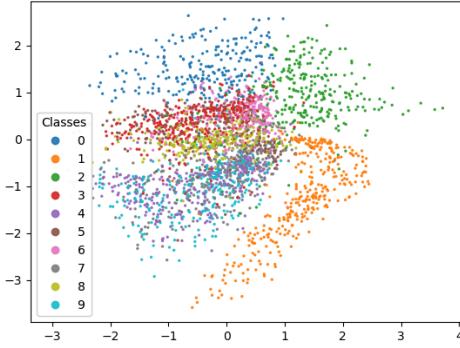


Abbildung 45: Test 11 - Latenter Raum des Unsupervised AAE

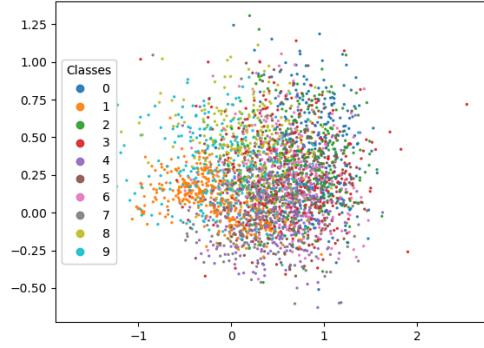


Abbildung 46: Test 11 - Latenter Raum des Semi-Supervised AAE

Test 11 liefert dieselben Ergebnisse. Beim Unsupervised AAE wird die unbekannte Klasse 7 zu der Gruppierung der Klassen 9 und 4 kategorisiert (siehe Abb. 45). Die Samples der Klasse 6 besitzen Überschneidungen mit den Samples der Klassen 2, 3 und 5. Augenscheinlich besitzt die Klasse 6 mehrere Merkmale, die sie sich mit mehreren Klassen teilt, was die eindeutige Eingliederung in eine bekannte Klasse erschwert. Das Semi-Supervised Modell liefert erneut keine saubere Kategorisierung. Es kann auch durch das Austauschen der unbekannten Samples keine Verbesserung erzielt werden. Keine Klasse wird im latenten Raum (Abb. 46) einem eindeutigen Bereich zugeordnet. Im Test 12 werden die normalen und anomalen Samples ausgetauscht, um die Hypothese auf verschiedenen Samples zu untersuchen. In der Abbildung 47 wird der latente Raum des Unsupervised AAE veranschaulicht. Die unbekannten Klassen 0 und 1 werden einer bereits bekannten Klasse zugeordnet. Jedoch ist die Zuordnung nicht eindeutig wie bei Test 10.

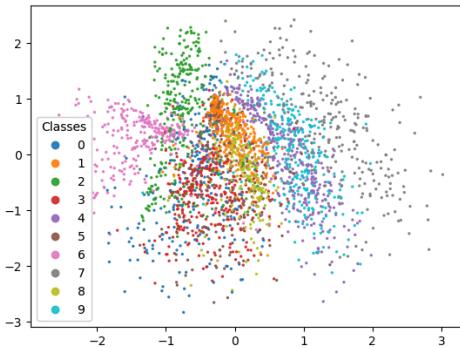


Abbildung 47: Test 12 - Latenter Raum des Unsupervised AAE

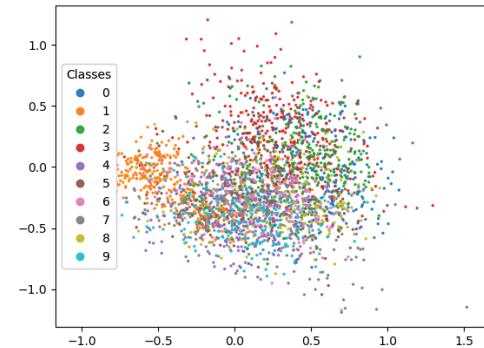


Abbildung 48: Test 12 - Latenter Raum des Semi-Supervised AAE

Die unbekannten Samples verfügen über mehrere Schnittmengen mit anderen Klassen. Samples der Klasse 0 besitzen eine hohe Übereinstimmung mit den Merkmalen der Klassen 3 und 5, wohingegen die Klasse 1 den Klassen 5 und 8 zu gegliedert wird. Auf den

ersten Blick widerlegt diese Klassifizierung die aufgestellte Hypothese. Mit der Analyse der erstellten Samples in Abbildung 49 kann die vorgenommene Einteilung erläutert werden. Der in blau umfasste Bereich zeigt generierte Samples, die den Datenpunkten der Klasse 0 ähnlich sind. Es kann zwischen den Klassen 3 und 5 eine Struktur ähnlich der einer Null erkannt werden, was die Einteilung zu diesen zwei Klassen rechtfertigt. Die Samples in dem markierten Bereich besitzen rundliche Außenlinien, welche der Form eines Kreises ähneln. Hinzu kommt, dass die Samples alle einen geschlossenen bzw. nahezu geschlossenen Kreis enthalten. Dieses Feature stimmt mit dem der Klasse 0 überein. Das orange Rechteck zeigt generierte Samples der Klassen 2, 3, 5, 8 und 9. Alle

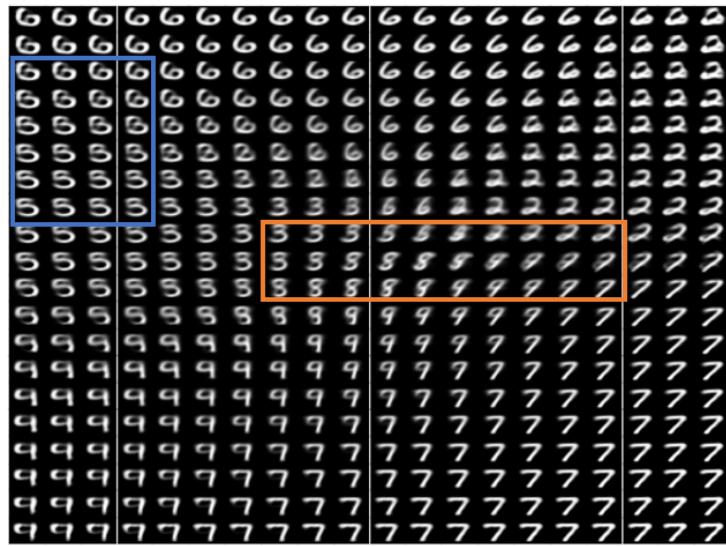


Abbildung 49: Test 12 - Durch den AAE generierte Datenpunkte

fünf Klassen besitzen eine Schnittmenge, in der sich die Merkmale vermischen. Im Übergang von der Klasse 8 zur Klasse 9 sind Strukturen der Samples 1 zu erkennen. In diesem Bereich teilen sich die Klassen dieselben Feature wie die Klasse 1. Der vertikale Strich ist in den generierten Samples stärker ausgeprägt als die die restlichen Pixel. Wegen dieser Merkmale werden die Samples der Klasse 1 in die Nähe bzw. in den Zwischenraum der oben genannten fünf Klassen gruppiert. Aufgrund der Analyse des latenten Raums, dargestellt in Abbildung 50, kann ein analoges Verhalten der Modelle zwischen Test 12 und Test 13 festgestellt werden. Die unbekannte Klasse 2 wird durch den Unsupervised AAE der bekannten Klasse 8 zugewiesen. Dasselbe gilt für die Klasse 3. Sie wird ebenfalls in den Bereich der Klasse 8 eingeteilt. Gleichermaßen kann wieder die Gruppierung der Samples der Klassen 4, 7 und 9 erkannt werden. Auffällig ist, dass die Klasse 1 isoliert von den anderen Samples liegt, sie aber näher zu den Klassen 8 und 9 liegt, was die obige Erklärung bestärkt. Die Klasse 0 teilt sich den latenten Raum mit den Samples der Klassen 5 und 6 was ebenfalls in der Abbildung 49 erklärt wird. Der Semi-Supervised AAE bewältigt die Aufgabe auch in den Testläufen 12 und 13, trotz anderer Datenaufteilung, schlecht. Wie bereits in den vorausgegangenen zwei Tests sind keine Muster im latenten Raum (Abb. 48 und 51) zu erkennen.

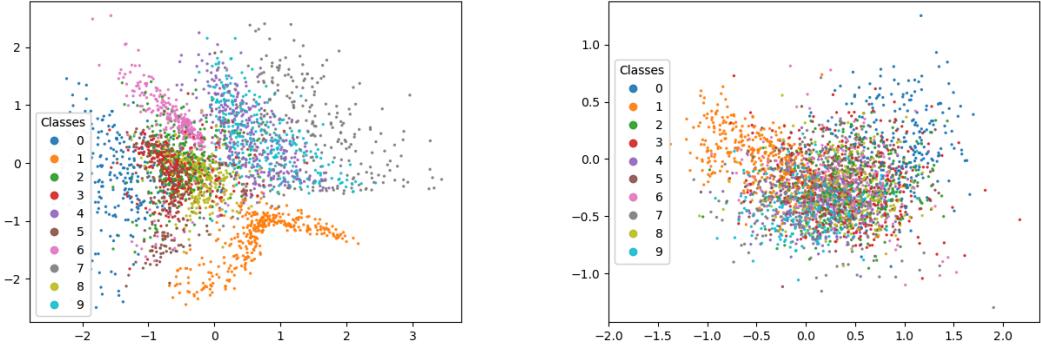


Abbildung 50: Test 13 - Latenter Raum des Unsupervised AAE

Abbildung 51: Test 13 - Latenter Raum des Semi-Supervised AAE

4.3.2.7 Erkenntnisse aus den Experimenten

Getestet wurde die Leistung des Unsupervised und Semi-Supervised AAE auf unterschiedlichen Aufteilungen des MNIST-Datensatzes. Der Datensatz wurden in den Tests in zwei Klassen eingeteilt, normal und anomalous. Es wurde der zweidimensionale latente Raum der Modelle analysiert, um Erkenntnisse über das Lernmuster von AAE zu erhalten. Diese Informationen sollen dabei helfen, gezielt neue Datenpunkte zu generieren, um Anomalieerkennungen zu verbessern. Entgegen der Erwartung klassifiziert der Unsupervised AAE die Samples besser als das Semi-Supervised Modell. Bei einer geringen Anzahl ist kein merklicher Unterschied der Performance zu erkennen, allerdings ist der Unterschied beim Verwenden des gesamten Datensatz groß. Der Semi-Supervised AAE gruppierter die Samples, trotz vorhandener Labels, ohne erkennbare Abtrennung zu den anderen Klassen. Der latente Raum der Unsupervised Variante ist auch bei einer großen Datenmenge aufgeräumt und die Cluster der einzelnen Klassen sind gut zu erkennen. In beiden Modellen werden die Daten nicht in die vorhandenen Klassen normal und anomalous klassifiziert, sondern anhand der gelernten Merkmale gruppiert. Es ist zu erkennen, dass sich ähnliche Punkte in demselben Bereich liegen. Dies führt dazu, dass unbekannte Punkte der Klasse zugeordnet werden, welche die meisten Übereinstimmungen besitzen. Dies gilt für beide Modelle. Aufgrund der Bestätigung der aufgestellten Hypothese durch die ausgewerteten Tests kann eine weitere Erkenntnis getroffen werden. Das Generieren von Anomalien zur Verbesserung einer Anomalieerkennung wird mit hoher Wahrscheinlichkeit positiven Einfluss nehmen, wenn die synthetisch hergestellten Samples den Anomalien ähneln. Besitzen die generierten Datenpunkte gleichartige Merkmale wie die normalen Daten, so kann es zu einer Verschlechterung der Performance führen, da unbekannte Punkte der normalen Klasse zugeordnet werden.

Zusammenfassend können drei Kernaussagen festgehalten werden:

- Unbekannte Datenpunkte werden der ähnlichsten Klasse zugeordnet.
- Semi-Supervised AAE klassifiziert schlechter als der Unsupervised AAE, trotz Labelinformation.

- Generieren von Anomalien ist sinnvoll, wenn sie sich deutlich von den normalen Samples unterscheiden.

Mit Hilfe dieser Erkenntnisse werden im nächsten Schritt Datenpunkte generiert, um durch deren Zugabe in eine bereits bestehende Anomalieerkennungsmethode die Klassifizierungsleistung zu verbessern.

5 Anwendung generierter Datenpunkte in einer Semi-Supervised Anomalieerkennung

Um eine konsistente Vergleichbarkeit aller Ergebnisse zu ermöglichen, wird bei diesen Experimenten ebenfalls ein Random-Seed von 2807 verwendet. Die Versuche werden auf dem Server des Fraunhofer AISEC ausgeführt. Letzterer besitzt fünf Nvidia Titan X Grafikkarten mit jeweils 12 GB RAM und zwei Exeon E5-2660 v3 CPUs. Als Datengrundlage dienen der original MNIST-Datensatz, sowie drei Datensätze, die mit Unterstützung des Unsupervised AAE generiert wurden. Von jeder Klasse des Datensatzes werden 6000 synthetische Punkte erstellt, somit besteht jeder hergestellte Datensatz aus 60000 Datenpunkten. Es gibt drei generierte Datensätze, da die Punkte aus verschiedenen Bereichen des latenten Raums gewählt werden.

Als Semi-Supervised Anomalieerkennung dient die Methode A^3 [16], entwickelt von Philip Sperl, Jan-Philipp Schulze und Konstantin Böttinger. Dieses Modell bietet die beste Grundstruktur für das Einbinden und Validieren der generierten Daten. Die Forscher analysieren die Aktivierungsenergie eines neuronalen Netzes, um zwischen Anomalien und normalen Daten zu unterscheiden. Hierfür implementieren Sie eine Modellarchitektur bestehend aus drei neuronalen Netzen [16]:

1. Das Target-Netzwerk führt eine Aufgabe, wie zum Beispiel Klassifizieren von Bildern, aus. Diese Aufgabe ist nicht Teil der Anomalieerkennung.
2. Das Anomaly-Netzwerk generiert Gegenbeispiele anhand der Samples, die dem Alarm Netzwerk im Trainingsprozess übergeben werden. In den Experimenten erzeugt dieses Netzwerk Datenpunkte durch einen Zufallszahlengenerator.
3. Das Alarm-Netzwerk klassifiziert die Samples in anomal und normal, indem es die Aktivierungsenergie in den Hidden-Layers des Target Netzwerks analysiert. Das Alarm Netzwerk analysiert während des Trainings ebenfalls die Aktivierungen der Datenpunkte, die durch das Netzwerk generiert wurden.

Das Alarm Netzwerk berechnet für das jeweilige Sample einen Anomalie-Score, welcher besagt, wie wahrscheinlich es sich bei dem Datenpunkt um eine Anomalie bzw. einen normalen Datenpunkt handelt. Die Labels der verwendeten Datensätze werden in A^3 auf binäre Labels umgewandelt, somit besitzen Anomalien das Label „1“ und normale Daten das Label „0“ [16]. Die hier generierten Daten werden in den nachfolgenden Experimenten während des Trainings in das Alarm-Netzwerk eingefügt, um so zu überprüfen, ob die Leistung des Netzwerks dadurch verbessert werden kann. Um eine Semi-Supervised Umgebung zu schaffen, wird A^3 auf 5, 25, 50 und 100 bekannten Anomalien trainiert. Das Experiment wird in zwei Versuche unterteilt. Im ersten Versuch werden die originalen Anomalien mit den generierten Anomalien ausgetauscht. Hiermit soll überprüft werden, ob generierte Punkte einen positiven bzw. negativen Einfluss auf die Anomalieerkennung haben. Im zweiten Versuch werden die originalen Anomalien mit den synthetisch hergestellten Anomalien angereichert. Hier wird angenommen, dass durch mehr Datenpunkte eine Steigerung der Klassifizierungsleistung stattfindet.

5.1 Generieren der Datenpunkte

In diesem Abschnitt wird darauf eingegangen, welche Punkte erstellt werden und welche Faktoren zu der vorgenommenen Einteilung beigetragen haben. Zunächst wird der, durch den Encoder erstellte, latente Raum betrachtet. Es wird eine Variante verwendet, die mehrere Klassen mit einem Unsupervised AAE trainiert. Dadurch werden im latenten Raum mehrere Klassen abgebildet, was ein Vorteil sein kann. Durch gezieltes Festlegen des Bereichs, aus welchem die Daten erstellt werden, können Mischformen der Datenpunkte generiert werden.

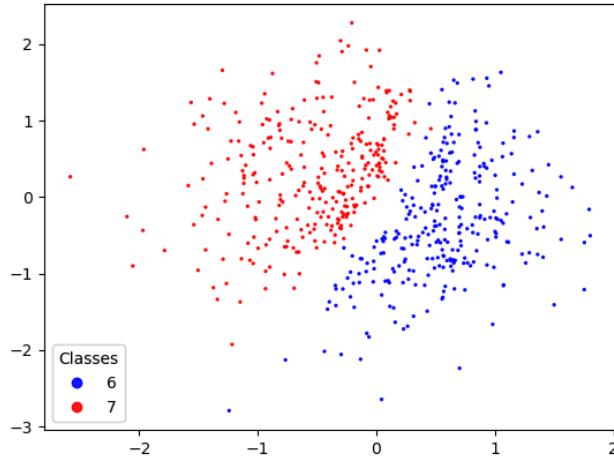


Abbildung 52: Zweidimensionaler, latenter Raum der Klassen 6 und 7

Wird der latente Raum (Abb. 52) eines Autoencoders betrachtet, welcher die Aufgabe hat zwischen der Klasse 6 und 7 zu unterscheiden, so ist merklich erkennbar, dass der Unsupervised AAE die Datenpunkte gut voneinander unterscheiden kann. Um einen gemischten Datenpunkt aus den Informationen der Klasse 6 und 7 zu generieren, muss der ausgewählte Bereich zwischen den beiden Punkteclustern liegen.

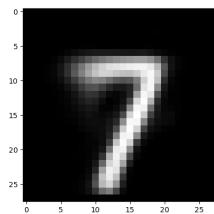


Abbildung 53: Rekonstruierte 7 des kombinierten latenten Raums 6 und 7

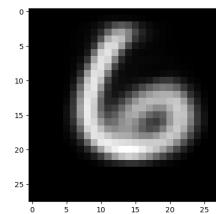


Abbildung 54: Rekonstruierte 6 des kombinierten latenten Raums 6 und 7

Mit der Normalverteilung wurden Punkte aus diesem Bereich generiert und mit Hilfe des Decoders erstellt. Die generierten Bilder nehmen jedoch leider nicht die gewünschte Form an, sondern ähneln immer einer der beiden Klassen (siehe Abb. 53 und 54). Um einen

weiteren Überblick der generierten Datenpunkte zu erhalten, werden 400 Datenpunkte in Abbildung 54 dargestellt. Hier wird ebenfalls ersichtlich, dass die Datenpunkte keine Mischformen zwischen der Klasse 6 und 7 bilden, sondern die Struktur klar einer der beiden Klassen zugeordnet werden kann.

7	7	6	7	6	6	7	7	7	7	6	6	7	6	6	6	7	6
7	7	6	7	7	6	6	6	7	7	6	6	7	7	6	7	7	7
7	7	6	7	6	6	7	6	7	7	7	7	6	6	7	7	6	6
6	7	6	7	6	6	7	7	7	6	7	6	7	6	7	7	6	7
6	7	7	7	6	6	7	7	7	6	7	6	7	6	7	7	6	6
6	7	7	7	6	7	7	7	7	7	7	6	6	7	7	7	6	6
7	7	6	7	6	7	7	7	7	7	7	6	6	6	7	7	7	6
7	7	7	6	7	7	7	7	6	6	7	6	6	7	7	7	6	7
7	6	7	6	7	6	6	7	7	7	7	7	6	7	6	6	6	6
6	6	7	7	6	6	7	6	6	7	6	7	6	6	7	6	7	7
7	7	7	6	6	7	7	6	6	7	6	6	7	6	7	7	7	7
7	7	6	6	7	7	7	7	7	7	6	6	6	7	7	7	7	7
4	6	7	6	6	7	6	6	6	7	7	6	7	7	6	6	6	6
4	7	7	6	7	7	7	6	7	7	7	6	7	6	7	6	7	7
6	7	7	6	7	7	7	6	7	6	7	7	7	6	6	7	6	7
6	6	7	6	7	7	7	6	7	7	7	7	6	6	7	6	7	7
6	6	7	6	7	6	7	6	7	7	6	7	6	7	7	6	7	6
6	6	7	6	6	7	7	7	6	6	6	7	6	7	7	7	6	6
4	6	7	6	7	6	7	7	6	7	7	7	6	7	7	6	7	7
6	6	6	7	7	7	7	6	6	6	6	6	6	7	7	6	7	6
6	6	6	6	7	6	7	6	7	7	7	7	7	6	7	7	7	6
6	6	6	6	7	6	7	6	7	7	7	7	7	7	6	7	7	7
6	6	7	6	6	7	6	7	7	7	7	7	7	7	6	7	7	7
6	7	6	7	7	6	6	7	6	6	7	6	6	7	7	6	7	7
6	6	7	6	6	7	7	6	7	7	6	7	6	7	7	6	7	7

Abbildung 55: 400 Samples der generierten Klasse 6 und 7

Aufgrund dieser Tatsache wurde die Variante verwendet, indem jede Klasse von einem Unsupervised AAE gelernt und generiert wird. Um Datenpunkte zu generieren, werden zehn verschiedene Unsupervised AAE trainiert und deren Gewichte gespeichert. Dieses Modell liefert bessere Ergebnisse als der Supervised und Semi-Supervised AAE.

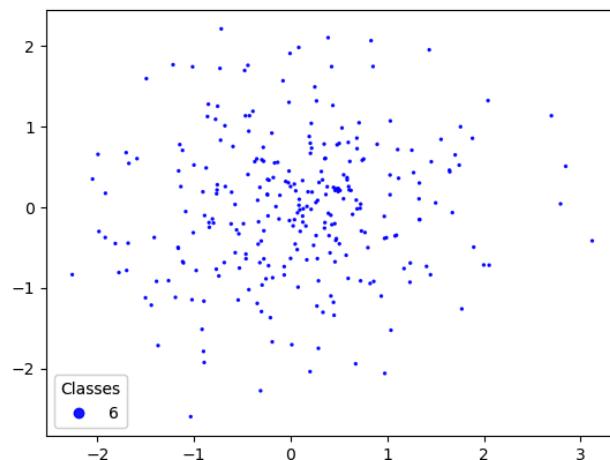


Abbildung 56: Zweidimensionaler, latenter Raum der Klasse 6

Jedes Netz ist für das Erlernen einer Klasse zuständig. Das bedeutet, für jede Klasse gibt es einen latenten Raum, welcher verwendet werden kann, um Datenpunkte zu generieren.

In der Abbildung 56 ist der zweidimensionale latente Raum der Klasse 6 dargestellt. Es ist gut zu erkennen, dass sich das Zentrum um den Punkt $(0, 0)$ bildet. Das bedeutet, die meisten Datenpunkte aus dem Trainingsdatensatz werden in diesem Bereich abgebildet. Dies lässt darauf schließen, dass die Punkte nahe de Zentrum eine hohen Informationsgewinn besitzen und die Features der Klasse am besten abbilden. Wird ein Punkt nahe de Zentrum für die Rekonstruktion der Klasse gewählt, so wird ein Bild ähnlich der gewünschten Klasse generiert. Je weiter der Punkt vom Erwartungswert entfernt ist, desto schlechter wird die Qualität des rekonstruierten Bildes, da in diesem Bereich nur wenige Testdaten liegen.

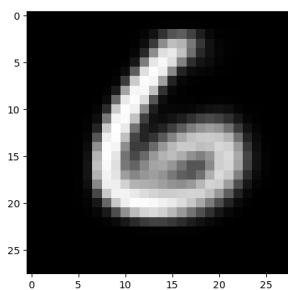


Abbildung 57: Rekonstruierter Punkt nahe des Zentrums

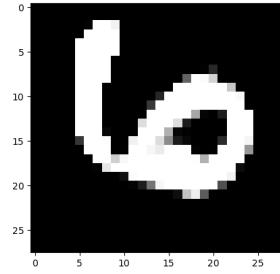


Abbildung 58: Rekonstruierter Punkt mit großem Abstand zum Zentrum

In der Abbildung 57 wird ein Punkt aus dem Zentrum dem Decoder übergeben. Der generierte Datenpunkt spiegelt die Klasse 6 gut wider. Wird ein Punkt außerhalb des Erwartungswerts gewählt, so wird beispielsweise der Datenpunkt aus Abbildung 58 erzeugt. Auf dem Bild ist ebenfalls eine Sechs erkennbar jedoch ist die Form und Struktur verschwommener und besitzt mehrere falsche Pixel. Des Weiteren wird die Zahl um 90 Grad rotiert, weshalb das System sie schwerer von einer Neun unterscheiden kann. Es wird angenommen, dass es sich bei der Datenverteilung im latenten Raum um eine Normalverteilung mit Erwartungswert $\mu = 0$ handelt [33], da diese durch den Diskriminatoren des AAE den Daten aufgezwungen wird.

$$X \sim \mathcal{N}(\mu, \sigma^2) \quad (9)$$

Für jede Klasse wird ein ähnlicher latenter Raum erzeugt (siehe Abb.56). Neben dem Erwartungswert enthält die Normalverteilung (Formel 9) den Parameter Standardabweichung σ^2 . Durch diesen lässt sich der Bereich begrenzen, aus dem die Punkte gewählt werden sollen. Die Leistung einer Anomalieerkennung durch generierte, anomale Daten zu verbessern war hierbei die Grundidee. Um herauszufinden welche Bereiche sich für eine Anomalieerkennung eignen, werden drei verschiedene Datensätze, mit jeweils verschiedenen Standardabweichungen σ^2 erzeugt. Als erstes wird die Normalverteilung $\mathcal{N}(\mu = (0, 0), \sigma^2 = 0.2)$ gewählt. Somit werden nur Punkte nahe des Erwartungswerts gewählt, um möglichst ideale Datenpunkte zu generieren. Diese werden

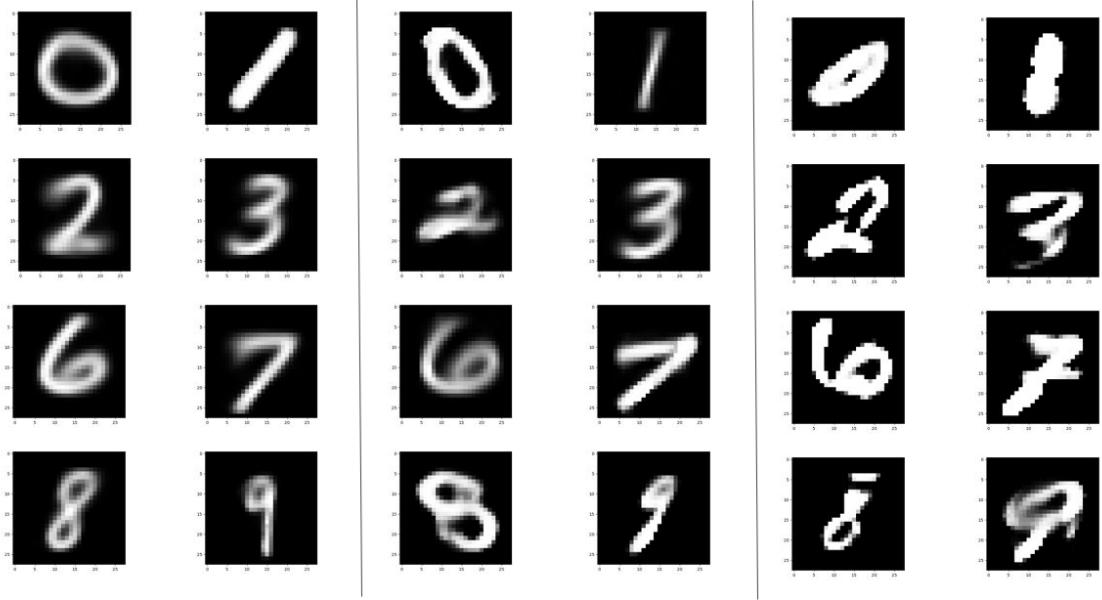


Abbildung 59: links: Generierte Bilder mit einer Standardabweichung von $\sigma^2 = 0.2$; mittre: Generierte Bilder mit einer Standardabweichung von $\sigma^2 = 3$; rechts: Generierte Bilder mit einer Standardabweichung von $\sigma^2 = 10$

in der linken Spalte der Abbildung 59 beispielhaft abgebildet. Die Klassenzugehörigkeit der jeweiligen Datenpunkte ist gut erkennbar. In der mittleren Zeile der Abbildung werden Datenpunkte mit dem Erwartungswert $\mu = 0$ und einer Standardabweichung von $\sigma^2 = 3$ gewählt. Diese Punkte spiegeln den MNIST-Datensatz am besten wider, da sie genau den Bereich abdecken indem sich die meisten Trainingsdatenpunkte befinden. Es herrscht eine höhere Varianz der Datenpunkte als in dem Datensatz mit einer Standardabweichung von 0.2. Da diese Standardabweichung ebenfalls die Punkte mit der Standardabweichung von 0.2 abdeckt, ist es wahrscheinlich, dass sich ebenfalls ideale Bilder in dem Datensatz befinden. Der dritte Datensatz wird mit Punkten befüllt, welche in dem Bereich $\mathcal{N}(\mu = (0, 0), \sigma^2 = 10)$ liegen. Diese Spanne enthält Punkte aus den ersten zwei Datensätzen, aber zusätzlich kommen Datenpunkte hinzu, die stark verformt sind und nicht eindeutig zu einer Klasse zugeordnet werden können. Das zugehörige Label wird den Punkten während dem Generieren übergeben. Somit werden drei veränderte Datensätze mit jeweils 60000 Datenpunkten generiert:

1. Datensatz aus generierten Punkten des Bereichs $\mathcal{N}(\mu = (0, 0), \sigma^2 = 0.2)$
2. Datensatz aus generierten Punkten des Bereichs $\mathcal{N}(\mu = (0, 0), \sigma^2 = 3)$
3. Datensatz aus generierten Punkten des Bereichs $\mathcal{N}(\mu = (0, 0), \sigma^2 = 10)$

Es wurde ebenfalls ein GAN implementiert, um Daten zu erzeugen. Der Generator erhält durch die Normalverteilung gewählte Punkte als Input und generiert daraus neue Bilder. Diese werden dem Diskriminatator übergeben, welcher versucht, zwischen den originalen Datenpunkten und den generierten Daten zu unterscheiden [10]. Die durch das GAN generierten Bilder sind in Abbildung 60 dargestellt. Eine Herausforderung bestand darin, gezielt Datenpunkte einer bestimmten Klasse zu erzeugen und diesen das richtige Label



Abbildung 60: Durch das GAN erzeugte Datenpunkte

mitzugeben. Aufgrund dieser Schwierigkeiten und der schlechteren Datenqualität als die des Unsupervised AAE, wurde diese Methode verworfen und für eine spätere Forschungsarbeit aufgehoben.

5.2 Beschreibung und Auswertung der Experimente

Es werden zwei verschiedene Experimente durchgeführt. Bei beiden wird die Datenaufteilung aus dem Paper *A³* [16] übernommen und jedes besteht aus vier Tests. Die Aufteilung des Datensatzes wird in der Tabelle 6 aufgelistet. Der Datensatz wird in

	Normal	Bekannte Anomalie	Validierungsdaten
Test a)	0, 1, 2, 3, 4, 5	6, 7	6, 7
Test b)	4, 5, 6, 7, 8, 9	0, 1	0, 1
Test c)	0, 1, 2, 3, 4, 5	6, 7	6, 7, 8, 9
Test d)	4, 5, 6, 7, 8, 9	0, 1	0, 1, 2, 3

Tabelle 6: Dateneinteilung für die Experimente

zwei Klassen eingeteilt, normal und Anomalie. Um eine Semi-Supervised Umgebung zu schaffen, wird die Anzahl der bekannten Anomalien auf 5, 25, 50 und 100 zufällig gewählte Datenpunkte der jeweiligen Klasse begrenzt. Durch die willkürliche Auswahl der Anomalien kann es vorkommen, dass eine Klasse im Training nicht vorkommt. Beispielsweise kann der Test a) keine Samples der Klasse 7 beinhalten, weil zufällig alle gewählten Samples der Klasse 6 entsprechen. In den Tests a) und b) wird die generelle Klassifizierungsleistung des Modells getestet. Das Alarm Netzwerk wird auf normalen und bekannten Anomalien trainiert. Die generierten Daten werden nur während des Trainings verwendet. Getestet und validiert werden die Modelle auf dem Originaldatensatz. Die Versuche c) und d) verwenden den vollständigen Datensatz und

enthalten bei der Validierung Datenpunkte, welche dem Netzwerk während des Trainings vorenthalten werden. Hier wird überprüft, ob das Netzwerk das gelernte Wissen von bekannten Anomalien auf unbekannte Datenpunkte übertragen kann [16]. Um eine Baseline zu schaffen, werden zunächst die Versuche aus A^3 wiederholt und evaluiert. Für die Evaluierung wird jeder Versuch dreimal durchgeführt und der Mittelwert der Ergebnisse berechnet.

5.2.1 Parameter für die Auswertung

Für die Auswertung der Modelle wird die Receiver Operating Characteristic Curve (ROC), sowie der Area Under The Curve (AUC)- und Average Precision (AP)-Wert betrachtet. Die ROC stellt die Werte der Konfusionsmatrix grafisch dar, indem sie die True Positiv Rate (TPR) gegen die False Positiv Rate (FPR) zeichnet. Auf der y-Achse befindet sich die TPR und auf der x-Achse die FPR. Die TPR ist auch unter den Namen Recall oder Sensitivität bekannt und wird wie folgt definiert [24]:

$$TPR = \frac{True\ Positiv(TP)}{TP + False\ Negativ(FN)} \quad (10)$$

Die Formel für FPR lautet:

$$FPR = \frac{FP}{TN + FP} \quad (11)$$

Die Definition der Konfusionsmatrix mit deren Inhalt wird in Tabelle 7 definiert. Somit besagt der True Positiv-Wert, wie viele Samples korrekterweise zur Klasse A klassifiziert werden. True Negativ wiederum definiert, wie viele Samples richtig zur Klasse B eingeteilt werden. Die Werte False Negative und False Positiv definieren die Anzahl der Samples, die irrtümlich der falschen Klasse zugewiesen werden. Der AUC-Wert berechnet

		Echte Klasse		100% positive Tests 100% negative Tests
		Klasse A	Klasse B	
Vorhersage	Test Positiv	True Positiv (TP)	False Positiv (FP)	
	Test Negativ	False Negativ (FN)	False Positiv (FP)	
		100% der Klasse A	100% der Klasse B	

Tabelle 7: Definition einer Konfusionsmatrix

die Fläche unter der ROC. Dabei ist das Modell besser, je näher der Wert bei 1 liegt. Das bedeutet, alle Samples werden richtig klassifiziert. Sollte der Wert bei 0 liegen, so werden alle Datenpunkte falsch klassifiziert. Bei einem AUC-Wert von 0.5 klassifiziert das Modell per Zufall die Validierungsdaten in die jeweiligen Klassen. Bei der AP handelt es sich um eine Kurve, welche die Precision gegen den Recall eines Modells plottet. Sie berechnet für jeden Schwellenwert die Precision und den Recall und stellt diese gegenüber. Auf der y-Achse sind die Precision-Werte gezeichnet und auf der x-Achse die des Recalls. Je höher der Wert der AP ist, desto besser ist das Modell. Die

Precision und der Recall werden wie folgt definiert [24]:

$$Precision = \frac{TP}{TP + FP} \quad (12)$$

$$Recall = \frac{TP}{TP + FN} \quad (13)$$

Um eine Baseline-Methode für eine besseren Vergleichbarkeit zu haben, wurde der Autoencoder Reconstruction Threshold berechnet und in der ROC eingezeichnet. Hierbei wird ein AE auf normale Datensamples trainiert und der Unterschied des Rekonstruktionsfehlers zwischen einer Anomalie und der normalen Samples berechnet. Der Unterschied wird mit Hilfe der mittleren quadratischen Abweichung berechnet [16].

$$\mathcal{L}(x, x') = \|x' - x\|_2^2 \quad (14)$$

5.2.2 Experiment 1: Ersetzen der Daten

Im ersten Experiment werden die anomalen Datenpunkte durch generierte Datenpunkte ausgetauscht. Die Anzahl von Anomalien bleibt jedoch bestehen. Hierbei soll überprüft werden, ob bei einer geringen Menge von generierten Daten eine Verbesserung bzw. Verschlechterung zu messen ist. Da der Versuch von Aufbau und Ablauf identisch mit dem Original-Experiment im Paper *A*³ ist, können die Ergebnisse gut miteinander verglichen werden.

Zuerst werden die generierten Daten mit der Standardabweichung von $\sigma^2 = 0.2$ verwendet. Es wird angenommen, dass diese Daten einen positiven Einfluss auf die Klassifizierungsleistung des Modells besitzen. Durch ihre saubere und qualitativ hochwertige Struktur kann das Alarm-Netzwerk trotz weniger Trainingssamples die klassendefinierenden Merkmale lernen. Bei dem Verwenden des Datensatzes mit der Standardabweichung von $\sigma^2 = 3$ werden ähnliche Ergebnisse wie mit den Originaldaten erwartet. Aufgrund der hohen Ähnlichkeit mit den Originaldaten wird keine Veränderung der Ergebnisse vermutet.

Im letzten Versuch werden die Anomalien mit den Daten des dritten Datensatzes ausgetauscht. Durch die Vielfalt an unterschiedlichen Datenpunkten und die reduzierte Anzahl von anomalen Datenpunkten, kann es vorkommen, dass sehr unterschiedliche Datenpunkte für das Training verwendet werden. Dies führt zu einer Verschlechterung der Klassifizierungsleistung, da das Modell nicht die wichtigen Merkmale der anomalen Klasse lernen kann. Die aus den Erwartungen abgeleitete Hypothese lässt wie folgt formulieren: Eine Verbesserung der Leistung kann mit Daten nahe des Erwartungswert verbessert werden, da die klassenspezifischen Merkmale gut dargestellt sind und so das Modell mit wenigen Samples gut zwischen Anomalien und normalen Daten unterscheiden kann. Einen Leistungsabfall wird es bei dem Verwenden des Datensatzes mit der Standardabweichung von $\sigma^2 = 10$ geben, da hier die Vielseitigkeit der Daten negativen Einfluss auf das Lernen der wichtigen Merkmale nimmt. Bei wenig bekannten Anomalien spielt die Qualität der Daten eine wichtigere Rolle als die Vielfalt von unterschiedlichen

Datenpunkten, weil das Modell schnellstmöglich die klassenspezifischen Merkmale extrahieren und lernen muss.

5.2.2.1 Ergebnisse

Dieses Experiment hat das Ziel herauszufinden, ob durch das bloße Austauschen der originalen, anomalen Datenpunkte durch generierte Samples bereits eine Veränderung der Klassifizierungsleistung zu erkennen ist. In der aufgestellten Hypothese wird erwartet, dass die generierten Daten nahe des Erwartungswerts, also dem Zentrum im latenten Raum, die besten Ergebnisse liefern. Das ist zurückzuführen auf die hohe Qualität der Daten. Dies hilft dem Alarm Netzwerk mit wenigen Daten die wichtigen Merkmale während des Trainings zu lernen.

Zunächst wird der Test a) betrachtet. Hier sind keine unbekannten Daten vorhanden. Die als Anomalie gekennzeichneten Klassen 6 und 7 werden mit den generierten Samples, während des Trainings, ausgetauscht. Die Ergebnisse in den nachfolgenden Tabellen beziehen sich auf den Validerungssplit des MNIST-Datensatzes.

Test a)	Original		Standardabw. 0.2		Standardabw. 3		Standardabw. 10	
	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP
100	0.991	0.984	0.985	0.970	0.969	0.935	0.673	0.494
50	0.982	0.971	0.986	0.970	0.965	0.921	0.826	0.717
25	0.978	0.960	0.980	0.962	0.913	0.850	0.632	0.488
5	0.874	0.821	0.817	0.755	0.788	0.660	0.747	0.631

Tabelle 8: Ergebnisse Test a)

In der Tabelle 8 sind die Ergebnisse der Testläufe zusammengefasst. Der originale Datensatz liefert bei 100 bekannten Anomalien das beste Ergebnis mit einem AUC-Wert von 0.991. Mit absteigender Anzahl von Datenpunkten während des Trainings, nimmt ebenfalls die Klassifizierungsleistung des Modells ab. Dasselbe kann bei dem Datensatz mit einer Standardabweichung von 3 beobachtet werden. Hier schneidet ebenfalls der Test mit 100 bekannten Samples am besten ab. Bei den Daten mit einer Standardabweichung von 0.2 werden bei 25 und 50 bekannten Datenpunkten bessere Ergebnisse erzielt als bei allen anderen Datensätzen. Jedoch ist der Unterschied zum Originaldatensatz sehr gering. Der Datensatz mit einer hohen Standardabweichung, also einer hohen Diversität innerhalb der Samples, liefert die schlechtesten Ergebnisse. In den Abbildungen 61, 62, 63 und 64 werden die ROC-Kurven für den jeweiligen Datensatz dargestellt. Die Graphen bestätigen die Ergebnisse der Tabelle 8. Der Originaldatensatz (Abb. 61), sowie der mit Standardabweichung 0.2 (Abb. 62), liefern die besten Ergebnisse. Dies kann am Kurvenverlauf abgelesen werden. Je stärker die Kurve in y-Richtung steigt, desto mehr Anomalien werden auch als solche klassifiziert.

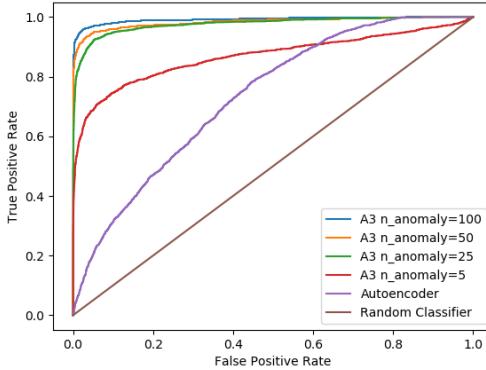


Abbildung 61: ROC,
Originaldatensatzes

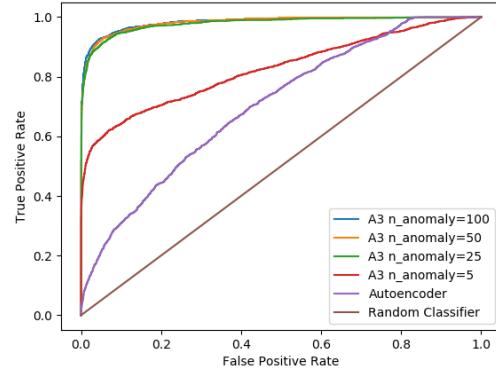


Abbildung 62: ROC,
Standardabweichung 0.2

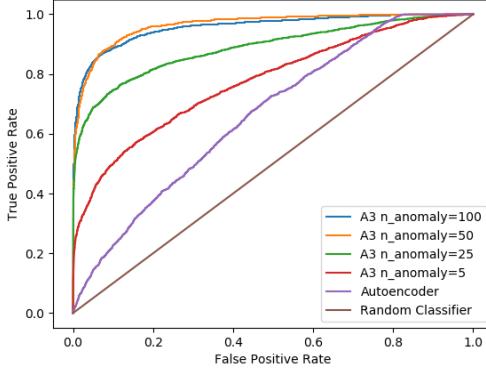


Abbildung 63: ROC,
Standardabweichung 3

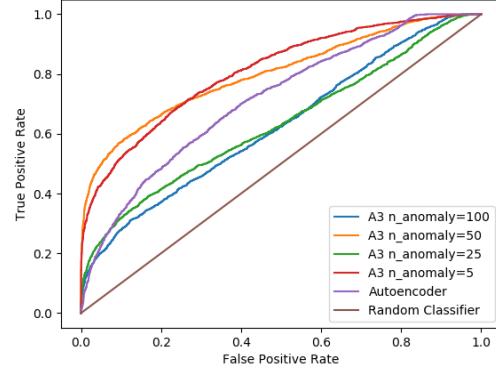


Abbildung 64: ROC,
Standardabweichung 10

Der Test b) besitzt ebenfalls keine vollständig unbekannten Datenpunkte. Die Ergebnisse sind der des Tests a) sehr ähnlich, jedoch sind sie allesamt besser. Beim Verwenden der Originaldaten werden bereits fast alle Anomalien als solche klassifiziert.

Test b)	Original		Standardabw. 0.2		Standardabw. 3		Standardabw. 10	
	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP
100	0.998	0.996	0.998	0.994	0.994	0.987	0.956	0.910
50	0.997	0.993	0.997	0.994	0.994	0.986	0.898	0.791
25	0.992	0.987	0.994	0.989	0.985	0.966	0.853	0.805
5	0.938	0.917	0.976	0.952	0.646	0.576	0.659	0.491

Tabelle 9: Ergebnisse Test b)

Aufgrund dessen ist es schwierig eine aussagekräftige Verbesserung zu erzielen. Der Datensatz mit der Standardabweichung von 0.2 erreicht die besten Ergebnisse mit einem AUC-Wert von 0.998. Die Datensätze mit einer Standardabweichung von 3 und 10 liefern vor allem bei fünf bekannten Anomalien schlechte Ergebnisse. Sie klassifizieren nur 65% der Samples richtig (siehe Tabelle 9).

Die Tests c) und d) überprüfen das Transferieren von Gelerntem auf unbekannte Datenpunkte. Im Test c) werden bei der Validierung die Klassen Acht und Neun hinzugefügt. Generell ist zu beobachten, dass die Ergebnisse schlechter sind als die der ersten zwei Tests. Jedoch ist ein ähnliches Muster zu beobachten. Die Datensätze mit

Test c)	Original		Standardabw. 0.2		Standardabw. 3		Standardabw. 10	
	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP
100	0.899	0.889	0.812	0.808	0.787	0.785	0.789	0.751
50	0.855	0.855	0.836	0.835	0.769	0.765	0.631	0.609
25	0.878	0.866	0.843	0.839	0.764	0.741	0.527	0.471
5	0.776	0.767	0.741	0.707	0.673	0.609	0.487	0.420

Tabelle 10: Ergebnisse Test c)

originalen und qualitativ hochwertigen Samples liefern auch hier die besseren Ergebnisse. Bei Test c) kann durch keinen der synthetischen Datensätze eine Verbesserung erzielt werden. Mit einer Standardabweichung von 0.2 werden jedoch durchschnittlich 80% der Daten richtig klassifiziert. Es ist ebenfalls zu beobachten, dass je größer die Standardabweichung ist, die Ergebnisse immer schlechter werden. Die Klassifizierung wird ebenfalls von der Anzahl der bekannten Trainingssamples beeinflusst. Der Test d) erzielt bessere Ergebnisse als der Test c). Dasselbe wurde bei den Tests a) und b) zu beobachtet. Es wird vermutet, dass die Struktur der Datenpunkte Null und Eins leichter zu lernen sind als die der Klassen 6 und 7. Aus der Tabelle 11 wird deutlich, dass der

Test d)	Original		Standardabw. 0.2		Standardabw. 3		Standardabw. 10	
	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP	AUC-ROC	AP
100	0.893	0.886	0.917	0.909	0.917	0.909	0.847	0.797
50	0.898	0.891	0.907	0.902	0.863	0.846	0.870	0.834
25	0.840	0.836	0.916	0.910	0.884	0.873	0.764	0.705
5	0.6148	0.658	0.787	0.794	0.614	0.601	0.621	0.586

Tabelle 11: Ergebnisse Test d)

Datensatz mit der kleinsten Standardabweichung die besten Ergebnisse erzielt. Er erreicht bei allen, außer dem Fall mit fünf bekannten Anomalien, eine Genauigkeit von über 90%.

5.2.2.2 Learnings

Anhand der Ergebnisse kann die Hypothese teilweise bestätigt werden. Je besser die Qualität der Samples ist, desto besser kann das Modell A^3 Anomalien als solche klassifizieren. Der Datensatz mit der Standardabweichung von 10 liefert bei jedem Test die schlechtesten Ergebnisse und erreicht bei Test c) mit fünf bekannten Anomalien, nur einen AUC von 0.487 und eine AP von 0.42. Es kann keine signifikante Verbesserung gegenüber dem originalen Datensatz beobachtet werden. Das kann an der bereits sehr guten Leistung von A^3 liegen. Einen großen Einfluss auf die Performance des Modells hat die Anzahl der vorhandenen Samples im Training. Mit einer hohen Anzahl von Samples klassifiziert A^3 die Anomalien besser.

5.2.3 Experiment 2: Hinzufügen von synthetischen Anomalien

Durch die Ergebnisse aus dem ersten Experiment wird deutlich, dass die Anzahl der bekannten Anomalien während des Trainings einen Einfluss auf die Performance des Alarm Netzwerks hat. Aufgrund dessen werden im zweiten Experiment die originalen Anomalien nicht ausgetauscht, sondern mit synthetischen Samples angereichert. Um zu überprüfen, ob dies zu einer Verbesserung führt, werden die Versuche aus Tabelle 6 jeweils mit 1000, 2500, 6000 und 12000 synthetischen Anomalien angereichert. Bei den ersten Testläufen ist aufgefallen, dass die Anzahl von originalen Datenpunkten keinen Einfluss auf die Ergebnisse nimmt. Das ist auf die hohe Anzahl von generierten Datenpunkten, welche während des Trainings den meisten Einfluss auf das Modell haben, zurückzuführen. Daher wurden als Basis 100 originale, anomale Samples ausgewählt. Somit stehen dem Alarm Netzwerk während des Trainings 100 Originale plus 1000, 2500, 6000 und 12000 generierte Anomalien zur Verfügung. Durch die zufällige Auswahl der anomalen Punkte kann es jedoch vorkommen, dass eine bestimmte Klasse nicht in dem Trainingsdatensatz enthalten ist. Wird beispielsweise der Versuch mit 1000 generierten Anomalien betrachtet, werden aus den übergebenen, anomalen Daten zufällig 1000 Punkte ausgewählt. Da jedoch von jeder Klasse 6000 Datenpunkte existieren, kann es vorkommen, dass beispielsweise in Test a) 1000-mal die Klasse 6 ausgewählt wird. Im Fall der 12000 synthetischen Anomalien wird garantiert jeder synthetisch generierte Datenpunkt verwendet. Durch das Hinzufügen von generierten Anomalien kann das Ungleichgewicht zwischen bekannten, normalen und anomalen Datenpunkten behoben werden.

Es wird erwartet, dass unabhängig vom verwendeten Datensatz, die Performance mit der Anzahl der Trainingssamples steigen wird. Anomale Daten sind nicht eingeschränkt auf ein spezielles Aussehen bzw. ein gemeinsames Verhalten. Sie können verschiedene Formen und Merkmale besitzen, was die Klassifizierung erschwert. Durch das Anreichern der Trainingsdaten durch generierte Anomalien wird versucht, diese Einschränkung zu umgehen. Deshalb wird, anders als bei dem ersten Experiment, die Hypothese aufgestellt, dass der Datensatz mit der Standardabweichung von 10 die besten Ergebnisse liefern wird. Durch die Vielseitigkeit der Datenpunkte lernt A^3 mehrere Features und kann neue Anomalien besser klassifizieren.

5.2.3.1 Ergebnisse

Die Tabellen zeigen die Ergebnisse der Testversuche mit den jeweiligen Datensätzen und der Anzahl von generierten Datenpunkten. In der untersten Reihe wird die Performance des A^3 mit 100 originalen Anomalien ohne die Zugabe von generierten Datenpunkten festgehalten. Die besten Ergebnisse sind gelb markiert und die des originalen Datensatzes grün.

In den Tests a) und b) erreichen alle Versuche einen sehr hohen AUC- und AP-Score. Jeder generierte Datensatz erzielt bessere Ergebnisse als der Original-Datensatz ohne zusätzliche Anomalien. Wie in der Hypothese formuliert, erreichen die Datensätze mit einer hohen Standardabweichung bessere Ergebnisse als mit geringer Abweichung. Bei

Test a) 100 Original Anomalien	Standardabweichung 0.2		Standardabweichung 3		Standardabweichung 10	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
1000	0.994	0.988	0.996	0.992	0.995	0.990
2500	0.994	0.988	0.995	0.990	0.996	0.991
6000	0.995	0.990	0.995	0.991	0.996	0.991
12000	0.995	0.990	0.996	0.991	0.995	0.989
Original	0.991	0.984				

dem Test a) kann eine Verbesserung von 0.05 erreicht werden und bei Test b) eine Verbesserung von 0.01. Aufgrund der sehr guten Performance des Modells ohne die Zugabe von generierten Anomalien kann keine aussagekräftige Schlussfolgerung getroffen werden. Bereits ohne die Zugabe von generierten Samples erreicht das Modell im Test a) eine durchschnittliche Genauigkeit von 0.996 und der AUC-Wert liegt bei 0.998. Im Test b) erreicht das A^3 ohne generierte Datenpunkte bereits einen AUC-Score von 0.998. Die fast perfekte Klassifikation in Test b) kann auf die einfache Struktur der Klasse Eins und Null zurückgeführt werden.

Test b) 100 original Anomalien	Standardabweichung 0.2		Standardabweichung 3		Standardabweichung 10	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
1000	0.998	0.997	0.999	0.997	0.998	0.996
2500	0.998	0.997	0.999	0.997	0.998	0.997
6000	0.999	0.997	0.998	0.998	0.998	0.997
12000	0.999	0.997	0.998	0.997	0.998	0.996
Original	0.998	0.996				

Die Tests c) und d) bieten aussagekräftigere Ergebnisse, da hier Wissenstransfer vom Modell notwendig ist. In beiden Tests werden in der Validierung zwei unbekannte Klassen hinzugefügt, um zu überprüfen, ob die Modelle gelernte Features der bekannten Anomalien auf die Unbekannten übertragen können. Hier kann beobachtet werden, dass mit steigender Anzahl von bekannten Anomalien während des Trainings die Leistung des Modells steigt. In beiden Tests liefert der Datensatz mit beiden Tests, mit einer Standardabweichung von 10 die besten Ergebnisse. Im Test c) kann bei der Verwendung von 6000 generierten Datenpunkten mit $\sigma^2 = 10$ der AUC-Wert um 0.011 verbessert werden. Der Datensatz mit der Standardabweichung $\sigma^2 = 0.2$ ist mit jeder Anzahl von zugefügten Anomalien schlechter als der Originalaufbau. Ein möglicher Grund hierfür ist der sogenannte Mode Collapse. Durch den sehr kleinen Bereich, welcher durch die Normalverteilung festgelegt wird, werden sehr ähnliche Datenpunkte erzeugt. Das neuronale Netz extrahiert dadurch nur bestimmte Features die sich sehr stark ähneln. Der Decoder bleibt durch das Overfitting in einem lokalen Minimum hängen und erkennt kleine Veränderungen der Features nicht, da es diese nicht gelernt hat. Bei Test d) ist der Effekt Mode Collapse nicht zu erkennen. Die Klassen Null und Eins besitzen eine geringe Komplexität mit wenigen verschiedenen Merkmalen, was trotz der sehr ähnlichen Datenpunkte eine Verbesserung bewirkt. Die aussagekräftigsten Ergebnisse liefert der Test d). Hier kann eine Verbesserung der Average Precision (AP) von 0.057 und des

Test c) 100 original Anomalien	Standardabweichung 0.2		Standardabweichung 3		Standardabweichung 10	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
1000	0.883	0.883	0.908	0.898	0.894	0.886
2500	0.881	0.879	0.895	0.892	0.916	0.905
6000	0.887	0.882	0.905	0.896	0.910	0.898
12000	0.892	0.882	0.901	0.889	0.910	0.897
Original	0.899	0.889				

AUC-Werts von 0.055 erzielt werden. Diese Leistungssteigerung wird beim Datensatz mit der Standardabweichung von 10 erreicht, aber die zwei Datensätze mit kleinerem Auswahlbereich liefern ebenfalls für jede Anzahl von synthetischen Anomalien eine Verbesserung der Performance von A^3 .

Test d) 100 original Anomalien	Standardabweichung 0.2		Standardabweichung 3		Standardabweichung 10	
	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR	AUC-ROC	AUC-PR
1000	0.901	0.895	0.935	0.933	0.936	0.927
2500	0.918	0.908	0.935	0.934	0.937	0.932
6000	0.925	0.918	0.940	0.9354	0.945	0.940
12000	0.938	0.931	0.914	0.910	0.948	0.943
Original	0.893	0.886				

5.2.3.2 Learnings

Die aufgestellte Hypothese kann besonders durch die Tests c) und d) bestätigt werden. Der Datensatz mit einem breiten Bereich an Daten liefert die besten Ergebnisse vor allem dann, wenn Transferleistung benötigt wird. Durch die verschiedenen Variationen der einzelnen Datenpunkte kann das Netzwerk viele Features erlernen und so unbekannte Datenpunkte als Anomalie klassifizieren. Die große Standardabweichung führt dazu, dass eine Mischung zwischen qualitativ hochwertigen und abstrakten Datenpunkten entsteht.

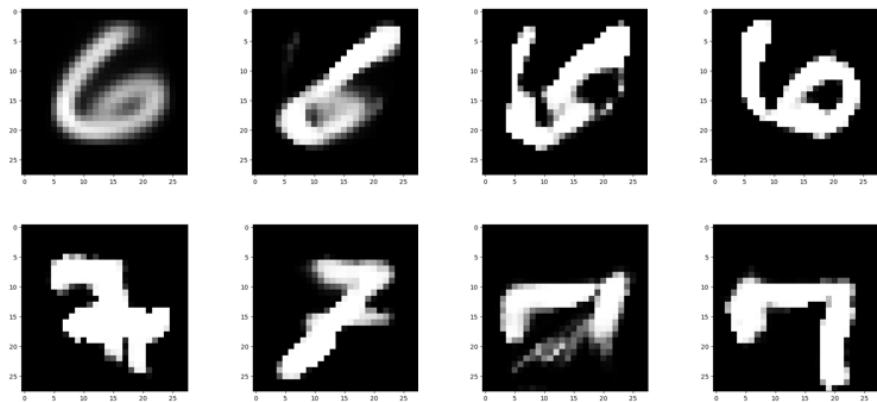


Abbildung 65: Datenpunkte der Klassen 6 und 7 aus dem Datensatz mit einer Standardabweichung von $\sigma^2 = 10$

Die zwei Restriktionen der Anomalieerkennung können durch das Einbringen von generierten Datenpunkten umgangen werden. Durch die unbegrenzte Anzahl von

bekannten Anomalien kann das vorherrschende Ungleichgewicht gelöst werden, aber es muss darauf geachtet werden, dass Ungleichgewicht nicht umzudrehen. Ansonsten stehen dem Modell zu viele Anomalien während des Trainings zur Verfügung und die Performance leidet unter dem Mangel an normalen Datenpunkten. Die zweite Restriktion besagt, dass Anomalien keinem speziellen Verhalten folgen. Sie können verschiedene Formen annehmen. Durch die Vielfältigkeit im generierten Datensatz (siehe Abb. 65) lernt das Modell verschiedene Merkmale, was bei der Klassifizierung unbekannter Punkte hilft. In der Abbildung 65 werden beispielhaft jeweils vier Datenpunkte der generierten Klasse 6 und 7 dargestellt. Die Datenpunkte stammen aus dem Datensatz mit der Standardabweichung von 10. Es ist gut zu erkennen, dass sowohl klare Strukturen der Klassen erkennbar sind, jedoch auch Verformungen welcher nicht den klassenspezifischen Mustern entsprechen. Diese Vielfalt nimmt positiven Einfluss auf die Transferleistung des Modells. Durch die hier ausgeführten Experimente kann festgestellt werden, dass die Hinzugabe von generierten Anomalien einen positiven Einfluss auf die hier verwendete Anomalie-Erkennungsmehtode A^3 .

6 Fazit

Das Ziel dieser Arbeit war es, eine bereits bestehende Anomalieerkennung durch die Zugabe von generierten Datenpunkten zu verbessern. Dafür wurden zunächst drei AAE implementiert. Einer für den Unsupervised-Fall, der zweite bearbeitete das Semi-Supervised Modell und der dritte benutzte einen Supervised-Datensatz. Die Experimente wurden nicht auf dem Supervised AAE durchgeführt, da in der Realität selten vollständig bekannte Datensätze vorliegen und der Fokus der Arbeit auf der Semi-Supervised Anomalieerkennung liegt.

Im ersten Experiment wurde der latente Raum der jeweiligen Modelle analysiert, um zu überprüfen, wie der jeweilige AAE, Daten voneinander trennen kann. Der hierbei verwendete MNIST-Datensatz wurde anhand der zehn Klassen in verschiedene Konstellationen aufgeteilt. Die originalen Klassenlabels wurden dafür in die binären Labels 0 und 1 umgewandelt. Das Label 1 steht für anomale und das Label 0 für normale Datenpunkte. Analysiert wurden die AAE in drei Schritten. Zuerst wurden zwei Klassen ausgewählt, die während des Trainings vollständig bekannt waren. Im zweiten Teil des Experiments wurde dieselbe Datenaufteilung wie im ersten Versuch übernommen, jedoch war, während des Trainings, die anomale Klasse vollständig unbekannt. Hier wurde überprüft, wie gut die Modelle das Gelernte auf unbekannte Daten transferieren können. Im dritten Teil wurde der Datensatz so aufgeteilt, dass er dem Semi-Supervised Szenario entspricht. Es wurden drei Klassen ausgewählt, wobei eine normale und eine anomale Klasse während des Trainings bekannt waren und die dritte Klasse erst zur Validierung hinzugefügt wurde. Insgesamt wurden vier verschiedene Aufteilungen dieser Art durchgeführt. Zum einen wurden Klassen ausgewählt, die sich stark voneinander unterscheiden und zum anderen die unbekannte Klasse, welche mal den normalen und mal den anomalen Datenpunkte ähnelte.

Ein weiteres Ziel, neben der Analyse der Performance, war es zu erfahren, ob die Datenstruktur einen Einfluss auf die Klassifizierung nimmt. Abschließend wurde der komplette Datensatz verwendet. Dabei wurden verschiedene Datenaufteilungen untersucht. Aus den Experimenten wurde deutlich, dass der Unsupervised AAE die Klassen am besten klassifiziert. Durch die Analyse des latenten Raums wurden drei Kernaussagen abgeleitet. Erstens wird vermutet, dass unbekannte Punkte der ähnlichsten, bekannten Klasse zugeordnet werden. Zweitens wird davon ausgegangen, dass generierte Datenpunkte eine Semi-Supervised Anomalieerkennung verbessern können, wenn die generierten Daten sich von den bekannten Datenpunkten unterscheiden bzw. ähnlich den bekannten Anomalien sind. Drittens liefert das Unsupervised AAE die besten Ergebnisse, trotz vorhandener Labels. Aufgrund dieser Resultate wird für das Generieren von Datenpunkten die Unsupervised Variante verwendet.

Im zweiten Teil der Arbeit werden neue MNIST-Datensätze erstellt und in die Anomalieerkennung A^3 integriert. Dafür werden zehn Unsupervised AAE genutzt, wovon jeder auf eine Klasse trainiert wird. Die Gewichte der trainierten Encoder und Decoder werden abgespeichert. Anhand des, durch den Encoder erstellten, zweidimensionalen, latenten Raums können mit Hilfe der Normalverteilung konkrete Bereiche zur Datengenerierung bestimmt werden. Dem Decoder werden anschließend Punkte aus dem zuvor bestimmten Bereich des latenten Raums übergeben. Der Parameter Erwartungswert und die Standardabweichung werden verwendet, um aus bestimmten Bereichen des latenten Raums Punkte auszuwählen. So wurden drei neue, vollständige MNIST-Datensätze generiert. Der erste besitzt eine Standardabweichung von 0.2. Das bedeutet, er beinhaltet qualitativ hochwertige Samples die sich alle sehr stark ähneln. Der zweite Datensatz spiegelt den originalen Datensatz wider. Hier wurde eine Standardabweichung von 3 gewählt, da mit dieser die meisten Punkte im latenten Raum abgedeckt werden. Der dritte Datensatz bietet die höchste Vielfalt an unterschiedlichen Datenpunkten, aufgrund einer sehr großen Standardabweichung. Anschließend werden die synthetischen Datensätze in die Anomalieerkennung A^3 hinzugefügt. A^3 verwendet für die Tests 5, 25, 50 und 100 bekannte Anomalien.

Im ersten Experiment wurden diese mit derselben Anzahl von generierten Datenpunkten ausgetauscht. Für jeden Datensatz wurde derselbe Versuch durchgeführt. Die Analyse der Ergebnisse zeigte, dass in bestimmten Fällen eine Verbesserung von A^3 bereits erzielt werden kann, wenn die bekannten, originalen Datenpunkte mit den generierten Datenpunkten aus dem Datensatz mit einer Standardabweichung von 0.2, ausgetauscht werden.

Im zweiten Experiment wird die Forschungsfrage, ob durch die Zugabe von generierten Anomalien die Semi-Supervised Anomalieerkennung verbessert werden kann, überprüft. Dafür wurden A^3 während des Trainings 1000, 2500, 6000 und 12000 generierte Datenpunkte, zusätzlich zu den 100 originalen Anomalien, übergeben. Die generierten Samples stammen immer aus einem Datensatz. Daher wurden keine Datenpunkte aus den drei Datensätzen gemischt. Die Ergebnisse zeigten, dass sich die Anzahl der im

Training vorhanden Datenpunkte auf die Performance des Modells auswirkt. Je mehr Samples vorhanden sind, desto besser kann das A^3 Anomalien erkennen. Auffallend ist, dass der Datensatz mit der Standardabweichung 10 die besten Ergebnisse erzielte, was auf die Vielseitigkeit der Datenpunkte zurückzuführen ist. Der Datensatz enthält sowohl qualitativ hochwertige Samples, als auch welche mit schlechter Qualität. Dadurch lernt A^3 verschiedene Merkmale und kann unbekannte Datenpunkte besser klassifizieren.

Durch die Ergebnisse lassen sich positive Erkenntnisse zur Verbesserung der Anomalieerkennung A^3 herausstellen. Diese Information kann relevant für jegliche Anomalieerkennung mit wenig bekannten anomalen Samples sein. Besonders in der Industrie könnten mit generierten Datenpunkten Fehler in der Produktion schnell erkannt werden, ohne vorher Expertenwissen zu besitzen. Auch eine Verbesserung der IT-Security oder des Onlinebankings könnte möglich sein. So könnten bekannte Netzwerkattacken verwendet werden, um neue Attacken zu generieren und dadurch die Modelle während des Trainings zu verbessern.

7 Ausblick auf zukünftige Arbeiten

Während der Arbeit wurden weitere interessante Experimente und Verbesserungsmöglichkeiten entdeckt. Das Modell A^3 liefert mit dem MNIST-Datensatz bereits sehr gute Ergebnisse. Daher wäre es interessant die Experimente mit weiteren Datensätzen zu testen. Eine zusätzliche Fragestellung ist, ob die Zugabe von normalen, generierten Daten, während des Trainings, ebenfalls zu einer Leistungssteigerung führt. Da es verschiedene Netzwerkarchitekturen gibt, die Daten generieren, wäre eine Möglichkeit die Datenpunkte mit einem GAN oder Variational Autoencoder (VAE) zu generieren. Um die Hypothese zu bekräftigen wäre es sinnvoll, weitere Anomalieerkennungsmethoden mit generierten Daten zu erweitern und die Klassifizierungsleistung zu messen.

Abkürzungsverzeichnis

- AAE** Adversarial Autoencoder 1, 2, 15–20, 23–29, 31–33, 35–38, 40, 42, 45, 48, 49, 51, 55, 65, 69, 70
- AD** Anomaly Detection 2
- AE** Autoencoder 2, 10–13, 15, 16, 29, 31, 33, 58, 69
- AI** Artificial Intelligence 10
- AP** Average Precision 57, 61, 62
- AUC** Area Under The Curve 57, 59–62
- CNN** Convolutional Neural Network 17
- DL** Deep Learning 2
- FPR** False Positiv Rate 57
- GAN** Generative Adversarial Networks 2, 10, 20, 21, 23, 55, 67
- HKA** Hauptkomponentenanalyse 12, 13, 15, 69
- Leaky ReLu** LeakyRectified Linear Unit 28
- MNIST** Modified National Institute of Standards and Technology 1, 13, 17, 23, 31, 36, 45, 48, 51, 54, 59, 65, 67, 70
- NIPS** Conference on Neural Information Processing Systems 2
- RCC** Robust Continuous Clustering 2
- ReLU** Rectified Linear Unit 13, 28, 29, 69
- RNN** Rekurrentes Neuronales Netz 2
- ROC** Receiver Operating Characteristic Curve 57, 58
- SGD** Stochastic Gradient Descent 19, 32, 33
- SVM** Scalar Vector Machine 10
- TPR** True Positiv Rate 57
- VAE** Variational Autoencoder 67

Abbildungsverzeichnis

1	Beispielhafte Darstellung von Anomalien	6
2	Beispielhafte Darstellung einer Collective Anomaly	7
3	Kriterien für Anomalieerkennungsmethode	8
4	Fortschritt der Generierung menschlicher Gesichter [20]	10
5	Aufbau eines Autoencoders	11
6	Struktur eines Autoencoders	12
7	Vergleich der Klassifizierung	13
8	Rekonstruierte Ergebnisse der HKA und des Autoencoders	14
9	Klassifizierungsergebnis des AE mit ReLu-Aktivierungsfunktion	14
10	Skizierte Darstellung eines AAE	15
11	Latenter Raum eines deterministischen Unsupervised AAE auf dem MNIST Datensatz	16
12	Latenter Raum eines deterministischen Unsupervised AAE mit CNN-Aufbau auf dem MNIST Datensatz	16
13	Architektur eines Supervised Adversarial Autoencoder (AAE)	18
14	Generieren von Daten mit vorgegebenem Style [25]	18
15	Architektur eines Semi-Supervised AAE	19
16	Skizzierter Aufbau eines Generative Adversarial Network	20
17	Darstellung des Trainingsprozesses von GANs [26] [20]	21
18	Beispiel jeder Zahl im MNIST-Datensatz [29]	23
19	Verteilung der Daten für jede Klasse	24
20	Vergleich der LeakyRelu- und ReLu-Aktivierungsfunktion	29
21	Sigmoidkurve	30
22	Vergleich der Datenpunkte 3 und 8	37
23	Referenz 1 - Latenter Raum eines Unsupervised AAE mit 50 Anomalien	38
24	Referenz 1 - Latenter Raum eines Semi-Supervised AAE mit 50 Anomalien	38
25	Test 2 - Latenter Raum eines Unsupervised AAE mit 50 Anomalien	39
26	Test 2 - Latenter Raum eines Semi-Supervised AAE mit 50 Anomalien	39
27	Test 2 - Latenter Raum eines Unsupervised AAE mit 1000 Anomalien	39
28	Test 2 - Latenter Raum eines Semi-Supervised AAE mit 1000 Anomalien	39
29	Referenz 3 - Latenter Raum des Unsupervised AAE	41
30	Referenz 3 - Latenter Raum des Semi-Supervised AAE	41
31	Test 5 - Latenter Raum des Unsupervised AAE	42
32	Test 5 - Latenter Raum des Semi-Supervised AAE	42
33	Test 7 - Latenter Raum des Unsupervised AAE	43
34	Test 7 - Latenter Raum des Semi-Supervised AAE	43
35	Test 8 - Latenter Raum des Unsupervised AAE	43
36	Test 8 - Latenter Raum des Semi-Supervised AAE	43
37	Test 9 - Latenter Raum des Unsupervised AAE	44

38	Test 9 - Latenter Raum des Semi-Supervised AAE	44
39	Samples der Klasse 1	44
40	Samples der Klasse 7	44
41	Samples der Klasse 9	44
42	Test 10 - Latenter Raum des Unsupervised AAE	46
43	Test 10 - Latenter Raum des Semi-Supervised AAE	46
44	Zweidimensionaler latenter Raum eines Supervised AAE auf dem MNIST-Datensatz [25]	46
45	Test 11 - Latenter Raum des Unsupervised AAE	47
46	Test 11 - Latenter Raum des Semi-Supervised AAE	47
47	Test 12 - Latenter Raum des Unsupervised AAE	47
48	Test 12 - Latenter Raum des Semi-Supervised AAE	47
49	Test 12 - Durch den AAE generierte Datenpunkte	48
50	Test 13 - Latenter Raum des Unsupervised AAE	49
51	Test 13 - Latenter Raum des Semi-Supervised AAE	49
52	Zweidimensionaler, latenter Raum der Klassen 6 und 7	52
53	Rekonstruierte 7 des kombinierten latenten Raums 6 und 7	52
54	Rekonstruierte 6 des kombinierten latenten Raums 6 und 7	52
55	400 Samples der generierten Klasse 6 und 7	53
56	Zweidimensionaler, latenter Raum der Klasse 6	53
57	Rekonstruierter Punkt nahe des Zentrums	54
58	Rekonstruierter Punkt mit großem Abstand zum Zentrum	54
59	Darstellung von Datenpunkten der drei generierten Datensätze	55
60	Durch das GAN erzeugte Datenpunkte	56
61	ROC Kurve des Originaldatensatzes	60
62	ROC Kurve des Datensatzes mit Standardabweichung 0.2	60
63	ROC Kurve des Datensatzes mit Standardabweichung 3	60
64	ROC Kurve des Datensatzes mit Standardabweichung 10	60
65	Datenpunkte der Klassen 6 und 7 aus dem Datensatz mit einer Standardabweichung von $\sigma^2 = 10$	64

List of Codes

1	Linearer, unvollständiger Autoencoder mit mittlerer, quadratische Abweichung	13
2	Laden des Datensatzes für den Unsupervised AAE [16]	25
3	Änderungen für das Laden des Datensatzes für Supervised AAE [16]	26
4	Limiteren der Anzahl von Anomalien [16]	26
5	Konvertieren der Labels in binäre Klassen [16]	27
6	Encoder des Unsupervised und Supervised AAE	28
7	Funktionsweise LeakyReLU	29
8	Decoder des Unsupervised AAE	29
9	Diskriminatior des Unsupervised AAE	30

10	Veränderung im Decoder für Supervised AAE	31
11	Encoder des Semi-Supervised AAE	31
12	Diskriminator zur Verarbeitung der Labelinformation	32
13	Rekonstruktionsphase	33
14	Berechnung des Rekonstruktionsfehlers	33
15	Regularisierungsphase	34
16	Berechnung des Regularisierungsfehlers	34
17	Training des Diskriminators für die Labels	34
18	Berechnung der Crossentropy des Generators	35
19	Berechnen des Klassifikationsfehlers	35

Literatur

- [1] Sara Makki, Zainab Assaghir, Yehia Taher, Rafiqul Haque, Mohand Saïd Hacid, and Hassan Zeineddine. An Experimental Study With Imbalanced Classification Approaches for Credit Card Fraud Detection. *IEEE Access*, 7:93010–93022, 2019.
- [2] Andrea Dal Pozzolo, Giacomo Boracchi, Olivier Caelen, Cesare Alippi, and Gianluca Bontempi. Credit card fraud detection: A realistic modeling and a novel learning strategy. *IEEE Transactions on Neural Networks and Learning Systems*, 29(8):3784–3797, aug 2018.
- [3] V Chandola, A Banerjee, and V Kumar. Anomaly detection: A survey. *ACM Reference Format*, 41(15), 2009.
- [4] Chaoyue Wang, Chang Xu, Xin Yao, and Dacheng Tao. Evolutionary generative adversarial networks. *CoRR*, abs/1803.00657, 2018.
- [5] Fabio Carrara, Giuseppe Amato, Luca Brombin, Fabrizio Falchi, and Claudio Gennaro. Combining GANs and autoencoders for efficient anomaly detection. *arXiv*, 2020.
- [6] Thomas Schlegl, Philipp Seeböck, Sebastian M. Waldstein, Ursula Schmidt-Erfurth, and Georg Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10265 LNCS:146–147, 2017.
- [7] Zhe Li, Chunhua Sun, Chunli Liu, Xiayu Chen, Meng Wang, and Yezheng Liu. RCC-Dual-GAN: An efficient approach for outlier detection with few identified anomalies, 2020.
- [8] Augustus Odena. Semi-Supervised Learning with Generative Adversarial Networks. jun 2016.
- [9] Chongxuan Li, Kun Xu, Jun Zhu, and Bo Zhang. Triple generative adversarial nets. *CoRR*, abs/1703.02291, 2017.
- [10] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. <http://www.deeplearningbook.org>.
- [11] Ian Goodfellow. NIPS 2016 tutorial: Generative adversarial networks. *arXiv*, 2016.
- [12] Jeff Donahue, Trevor Darrell, and Philipp Krähenbühl. Adversarial feature learning. In *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, may 2017.
- [13] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Gan. *arXiv*, 2017.

- [14] Sohil Atul Shah and Vladlen Koltun. Robust continuous clustering. *Proceedings of the National Academy of Sciences of the United States of America*, 114(37):9814–9819, sep 2017.
- [15] Saman Motamed and Farzad Khalvati. Vanishing twin GAN: how training a weak generative adversarial network can improve semi-supervised image classification. *CoRR*, abs/2103.02496, 2021.
- [16] Philip Sperl, Jan-Philipp Schulze, and Konstantin Böttinger. A³: Activation Anomaly Analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12458 LNAI:69–84, mar 2020.
- [17] Song Xiuyao, Wu Mingxi, Christopher Jermaine, and Sanjay Ranka. Conditional anomaly detection. *IEEE Transactions on Knowledge and Data Engineering*, 19(5):631–644, 2007.
- [18] Fabio Carrara, Giuseppe Amato, Luca Brombin, Fabrizio Falchi, and Claudio Gennaro. Combining GANs and autoencoders for efficient anomaly detection. *arXiv*, 2020.
- [19] Tony Jebara. *Machine Learning - Discriminative and Generative*. Springer Science+Business Media, LLC, 2004. Originally published by Kluwer Academic Publishers in 2004.
- [20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, oct 2020.
- [21] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.
- [22] A. Radford, Jeffrey Wu, R. Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.
- [23] David Foster. *Generative Deep Learning : Teaching Machines to Paint, Write, Compose, and Play*. O'Reilly Media, Incorporated, 2019. ProQuest Ebook Central, <https://ebookcentral.proquest.com/lib/hm-bib/detail.action?docID=5833992>.
- [24] Aurélien Géron. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Unsupervised learning techniques* -. O'Reilly Media, Incorporated, Sebastopol, California, 2019.
- [25] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.
- [26] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley,

Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

- [27] Ken Binmore, Volker Ellerbeck, and Ken Binmore. *Spieltheorie*. Reclam, 2013.
- [28] Sajal K. Das, Krishna Kant, and Nan Zhang. *Handbook on Securing Cyber-Physical Critical Infrastructure*. Elsevier Inc., United States, January 2012.
- [29] Toshifumi Kuga. More than 10x faster! you may have an access to the super-powered computers, too!, 2018.
- [30] The mnist database. <http://yann.lecun.com/exdb/mnist/>. Accessed: 2021-06-29.
- [31] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 28, 2013.
- [32] Kenneth Leung. The dying relu problem, clearly explained.
- [33] Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, USA, 2012.