HAW HAMBURG

MASTER THESIS
Daniel Eggert

Ein System für die Generierung von synthetischen fotorealistischen Bildern mit Prozessen des Deep Learning

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Daniel Eggert

Ein System für die Generierung von synthetischen fotorealistischen Bildern mit Prozessen des Deep Learning

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Dr. Susanne Draheim

Eingereicht am: 30. Mai 2023

Daniel Eggert

Thema der Arbeit

Ein System für die Generierung von synthetischen fotorealistischen Bildern mit Prozessen des Deep Learning

Stichworte

Diffusionsmodelle, Machine Learning Architektur, GANs

Kurzzusammenfassung

In dieser Arbeit wird eine Software Lösung für die Generierung von Bildern als Trainingsdaten vorgestellt. Dabei werden Methoden des Maschinellen Lernens verwendet, um anhand von einer übergebenen Bilddatenmenge neue Bilder zu generieren. Die Softwarelösung legt ihren Fokus auf die Abstraktion der Logik des maschinellen Lernens und eine leichte Erweiterbarkeit. Dadurch sollen auch Nutzer ohne Kenntnisse des maschinellen Lernens diese verwenden können und ohne größeren Mehraufwand alternative Methoden der Bildgenerierung bestehend aus neuronalen Netzen integriert werden können.

Daniel Eggert

Title of Thesis

A system for generating synthetic photorealistic images using deep learning processes

Keywords

Diffusion Models, Machine Learning Architecture, GANs

Abstract

In this paper a software solution for the generation of images as training data is presented. Machine learning methods are used to generate new images based on a given set of image data. The software solution focuses on the abstraction of the machine learning logic and an easy extensibility. Thus, even users without knowledge of machine learning should be

able to use it and alternative methods of image generation consisting of neural networks can be integrated without major additional effort.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Das Sammeln von guten und vielen Trainingsdaten ist eine der größten Herausforderungen des maschinellen Lernens.[9] Bei der Entwicklung von Modellen des maschinellen Lernens entfallen ca. 80-90% der Zeit für das Sammeln und Bearbeiten der Trainingsdaten.[74] Das ist ebenfalls bei Objekterkennern/-detektoren der Fall. Auch hier sind zu wenig annotierte Bilder als Trainingsdaten eine große Herausforderung beim Trainingsprozess. Dabei helfen viele vorhandene Bilder vor allem gegen das Problem, dass ein neuronales Netz sich zu stark auf die antrainierten Bilder fokussiert und neue nicht mehr erkennt. Manchmal ist auch nur ein kleiner Teil der Bilder für die Erkennung relevant, oder das Objekt, welches erkannt werden soll, ist an sich schon sehr klein. Des weiteren sind Objekte auch nicht immer in derselben Form.[28] Auch sind Bilder der zu erkennenden Objekte nicht immer leicht zu beschaffen, wie z.B. bei Krankheitssymptomen von Lungen.[27] Auch wird bei zu wenigen Daten, wenn diese häufig denselben Hintergrund verwenden, die erfolgreiche Erkennung eines Objekts abhängig von den anderen Objekten im Hintergrund.[39] Ein weiteres Problem ist das häufige Fehlen von Bildern unter schlechten Wetterbedingungen, da viele existente Datensätze lediglich Daten unter guten Wetterbedingungen enthalten.[69, 63] Dabei beeinträchtigen schlechte Wetterbedingungen, wie z.B. Nebel, Regen oder Schnee, die Erkennung im Außenbereich.[36] Die meisten Algorithmen der Bildverarbeitung sind folglich designt, um bei guten Wetterbedingungen zu funktionieren. Gerade beim Bereich des autonomen Fahrens wäre dies jedoch kritisch, da sich das Wetter auf der Straße häufig unterscheidet und der Algorithmus auch hier funktionieren muss.[63] Zwar könnte man mit Methoden der Bildverarbeitung das Wetter aus dem Bild entfernen, aber auch hier müsste man zunächst erkennen, welches Wetter sich auf den gesuchten Bildern befindet.[36] Die Möglichkeit, Simulationen in virtuellen Umgebungen zu nutzen, um Bilder mit den benötigten Wettereffekten zu generieren, sind hierbei nicht immer geeignet, da diese oft zu realitätsfern sind. Mit neuronalen Netzen,

z.B. GANs, könnten bereits existierende Datensätze genutzt werden, um damit Bilder mit den benötigten Wettereffekten zu generieren.[63]

Maschinelles Lernen kann für unterschiedliche Themenbereiche der Informatik eingesetzt werden, so z.B. auch für die Augmented Reality(AR), wenn man einen Objektdetektor für das Tracking verwenden will, um beispielsweise verschiedene Obst und Gemüsesorten zu erkennen.[20] Dafür benötigt man jedoch Kenntnisse in dem Bereich des maschinellen Lernens, um ein nützliches Netz trainieren und implementieren zu können. Um neuronale Netze ohne tiefere Kenntnisse des maschinellen Lernens verwenden zu können, wird eine Anwendung benötigt, welche möglichst viele Funktionen abstrahiert und für Nutzer verständlich darstellt. Ein Beispiel hierfür ist CreateML[2] von Xcode. Hier werden verschiedene vorimplementierte Netze bereitgestellt, welche lediglich ein vorgegebenes Format an Trainingsdaten benötigen, um ein ausgewähltes Netz zu generieren. So sind das bei Objekterkennern beispielsweise Bilder, weshalb deren Qualität hauptsächlich nur mittels Bilder verbessert werden kann. Gerade im Fall von Obst, Gemüse und Pflanzen im Allgemeinen sind viele Bilder wichtig, da diese selbst in derselben Gattung viele verschiedene Farben und Formen aufweisen. Auch wachsen viele Pflanzen nur in bestimmten Monaten, sodass der Zeitraum, Trainingsdaten zu diesen zu beschaffen nur eingegrenzt möglich ist. CreateML stellt jedoch keine Möglichkeit bereit, einen Bildgenerierer zu trainieren, weshalb dieser für eine Erweiterung des Trainingsdatensatzes selbst erstellt werden müsste. Dadurch müsste sich ein AR-Entwickler Kenntnisse über den Bereich des maschinellen Lernens aneignen.

Die Motivation dieser Arbeit besteht folglich daraus, eine Möglichkeit für das Erweitern eines Trainingsdatensatzes, bestehend aus Bildern, zu implementieren. Diese soll genau wie Xcode die tieferen Funktionen des maschinellen Lernens abstrahieren, um damit auch Entwicklern aus einer anderen Domain eine einfache Verwendbarkeit zu gewährleisten. Durch Speichern der Modelle soll zudem allen folgenden Entwicklern eine Möglichkeit zur Verfügung stehen, zu bereits existierenden Domänen Bilder generieren zu können.

1.2 Aufbau der Arbeit

Diese Arbeit ist in fünf aufeinander aufbauende Abschnitte eingeteilt. Zunächst beginnt die Arbeit mit der Analyse. Dafür werden zunächst die Ziele für die Arbeit festgelegt und definiert. Im Anschluss daran werden anhand von aktueller Forschungsliteratur Lösungsmöglichkeiten für die Umsetzung der Ziele erörtert und prägnant zusammengefasst.

Hierbei wird der verwendete Bereich des maschinellen Lernens, nämlich das unüberwachte Lernen, kurz erläutert. Danach werden die generativen GAN-Netze näher beschrieben. Dort wird vor allem auf die Funktionsweise, die Vorteile sowie die Herausforderungen und auf mögliche GAN Architekturen eingegangen. Anschließend wird die aktuelle Alternative zu GANs, namentlich das Diffusionsmodell, in einer ähnlichen Weise wie die GANs beschrieben. Eine weitere Alternative zu GANs und Diffusionsmodellen wird daraufhin kurz genannt. Hiernach folgt eine Zusammenfassung der aktuellen Konzepte für das Entwickeln von Software mit Bezug auf das maschinelle Lernen. Dabei spielt vor allem eine Rolle, wie eine Software aufgebaut werden muss, damit diese leicht erweitert werden kann. Hier soll eine Übersicht über die Frameworks, Architekturmöglichkeiten und gängige Pattern für Softwareanwendungen mit integrierten maschinellen Lernen geschaffen werden. Auch Möglichkeiten zum Deployment werden daraufhin kurz genannt. Die Analyse wird mit einer Evaluation der Erkenntnisse in Bezug auf die vorher gestellten Anforderungen abgeschlossen. Dabei wird betrachtet, welche Methoden sich mehr oder weniger gut für die Realisierung der Anforderungen eignen können.

Anhand der Erkenntnisse der Analyse wird im folgenden eine Anwendung entwickelt, welche die Anforderungen erfüllen kann. Dafür wird zunächst ein Gesamtüberblick über die Architektur der Software fassbar gemacht. Zusätzlich wird begründet, weshalb sich für spezifische Frameworks und Architekturmöglichkeiten entschieden wurde. Danach werden die einzelnen Bestandteile und Funktionen im Detail beschrieben.

Im Kapitel der Experimente wird die Effektivität der Anwendung in Bezug auf die Generierung von Bildern getestet. Dabei wird vor allem getestet, ob die Modelle mit wenig Trainingsbildern in der Lage sind, neue Bilder zu generieren, welche für weiteres Training verwendet werden können. Auch soll hier der Effekt von Data Augmentation auf die Modelle getestet werden. Kurz genannt wird hier zusätzlich, wie die Anwendung funktioniert hat und ob noch Probleme auftreten.

Im Anschluss an die Experimente werden die Ergebnisse evaluiert und deren Auswirkung auf die Anwendung betrachtet. Abschließend wird ein umfassendes Fazit gebildet und es werden mögliche zukünftige Arbeiten vorgestellt.

2 Analyse

2.1 Zielsetzung

Das primäre Ziel dieser Anwendung ist es, Nutzern spezifische Trainingsdaten in Form von Bildern zur Verfügung zu stellen. Dafür muss der Nutzer der Anwendung einen Grundbestand an Trainingsdaten übermitteln. Anhand des Grundbestandes soll die Anwendung lernen, Bilder derselben Domäne zu generieren. Die Anwendung soll dafür Methoden des maschinellen Lernens nutzen, um diese Funktion bereitzustellen. Einmal gelernt soll die Anwendung die Fähigkeit, Bilder dieser Domain zu generieren, für die weitere Nutzung erhalten. D.h. wenn die Anwendung einmal erlernt hat, beispielsweise Bilder von Katzen zu generieren, dann erhält die Anwendung diese Fähigkeit und jeder weitere Nutzer benötigt keinen Grundbestand an Trainingsdaten mehr, um Bilder von Katzen zu erhalten. Durch diese Funktion wird die Anwendung stetig erweitert und wird so bei jeder Nutzung mit einer neuen Zieldomäne erweitert.

Ein weiteres Ziel der Anwendung ist eine einfache Benutzbarkeit. Die Anwendung soll hierbei die Funktionen des maschinellen Lernens kapseln, sodass auch Nutzer, welche noch wenig Erfahrung mit den Funktionen des maschinellen Lernens haben, diese nutzen können.

Das letzte Ziel der Anwendung ist es, dass diese selbst leicht erweiterbar ist. So soll die Anwendung für den Fall, dass noch mehr Mechanismen für die Generierung von synthetischen Bildern verwendet werden sollen als diese bereits bereitstellt, leicht erweiterbar sein. Demzufolge sollen sich beispielsweise neuronale Netze in die Anwendung integrieren lassen, ohne dass diese nicht mehr funktioniert oder größtenteils umgebaut werden muss.

2.2 Kurzüberblick unüberwachtes Lernen

Unüberwachtes Lernen liegt vor, wenn neuronale Netze ohne Klassenbezeichnungen lernen. Das Model weiß also nicht, um was es sich bei den Daten handelt. Hierbei gibt es verschiedene Unterarten, wie z.B. das semi-supervised Learning, bei dem ein Teil der Daten eine solche Klassenbezeichnung mit sich bringen und ein anderer Teil diese nicht enthält.[66] In den Daten werden dann versteckte Strukturen identifiziert. Generative neuronale Netze sind ein Beispiel für das unüberwachte Lernen.[18] Sie können aber auch Methoden des semi-supervised Learnings verwenden.[42]

2.3 Overfitting

Der Zustand des Overfittings liegt vor, wenn ein neuronales Netz sehr gute Ergebnisse für die Trainingsdaten liefert, aber nur schlechte, wenn dieses auf neue, nicht im Trainingsdatensatz, vorhandene Daten angewendet wird. Damit ist das neuronale Netz zu stark auf die Trainingsdaten angepasst. Das Phänomen tritt sehr häufig bei einem kleinen Trainingsdatensatz auf, oder wenn die Trainingsdaten, am Beispiel von Bildern, nur aus einer Perspektive aufgenommen werden.[28]

Overfitting ist damit ein großes Problem, welches durch die Ergebnisse der Anwendung verhindert werden soll. Denn mit den generierten Bildern kann ein existierender Bilderdatensatz vergrößert werden.

2.4 Generative Adversial Networks

2.4.1 Funktionsweise

Generative Adversial Networks wurden 2014 von Goodfellow et al.[23] entwickelt. Sie basieren auf zwei neuronalen Netzen, dem Generator und dem Diskriminator. Der Generator erlernt anhand einer Menge von Trainingsdaten, der sogenannten Domain, neue Bilder derselben Domain zu generieren. Der Diskriminator versucht die Bilder zu erkennen, welche vom Generator erstellt wurden, indem er sie mit der ursprünglichen Bildmenge vergleicht. Bildlich sprechen Goodfellow et al. beim Generator von einem Fälscher, welcher z.B. Banknoten fälschen will und beim Diskriminator von der Polizei, welche

die gefälschten Banknoten erkennen soll. Der Diskriminator und der Generator treten in einem gegnerischen Minimax-Spiel gegeneinander an.[23]

GANs gehören zur Klasse der generativen Modelle. Ihr Ziel war es, die bisherigen Nachteile der damals existierenden generativen Modelle zu eliminieren. Ihre Relevanz in der Forschung ist groß, so wurden bis zum Jahr 2020 mehr als 28.500 verschiedene Arbeiten über GANs veröffentlicht.[26]

Seit der Entwicklung des Grundmodells wurde dieses stetig weiterentwickelt. Die Architektur variiert je nach Problemstellung. So gibt es GANs für den Stiltransfer, die Generierung von hochauflösenden Bildern und vielen weiteren. Eine detaillierte und aktuelle Zusammenfassung bieten Figueira und Vaz[22] in ihrer systematischen Literaturrecherche zum Thema GANs. GANs haben viele Anwendungsgebiete z.B. beim Generieren von Videodaten, Textdaten und Bilddaten.[26] Des Weiteren sind sie dazu geeignet, das Problem von unzureichenden Trainingsdaten zu lösen.[71]

2.4.2 Vorteile von GANs

Die Verwendung von GANs für die Synthese von Bildern bietet Vorteile. Ein Vorteil ist die flexible Gestaltung des Generators, so ist dieser anpassbar und es existieren allgemein wenig Einschränkungen, wie dieser zu gestalten ist.[26] So kann man z.B. verschiedene Loss-Funktionen im Generator verwenden, um das Training anzupassen.[71] Des Weiteren ist es möglich, den Prozess der Generierung beispielsweise bei großen Bildern zu parallelisieren. Sie besitzen auch deshalb keine Einschränkungen bei den Dimensionen von Bildern. Die Ergebnisse von GANs sind des Weiteren auch auf eine natürliche Art zu interpretieren und deshalb leicht zu verstehen.[26]

2.4.3 Herausforderungen von GANs

Die Entwicklung und Forschung von GANs ist seit längerer Zeit von ähnlichen Herausforderungen geprägt. Eine der größten Herausforderungen ist der Mode Collapse.[65, 48, 11] Mode Collapse kann durch unterschiedliche Arten entstehen. Eine Möglichkeit ist es, wenn der Diskriminator anfällig für eine spezifische Fälschung ist und die anderen Fälschungen zum größten Teil immer erkannt werden. Ist dies der Fall, fokussiert sich der Generator auf das Erzeugen des genau einen Falls und generiert keine anderen Bilder

mehr. Folglich wird immer dasselbe Bild mit minimalen Änderungen generiert. Ein anderer Fall von Mode Collapse liegt vor, wenn die wichtigen Eigenschaften von den Bildern in den Trainingsdaten nicht mehr in den generierten Bildern vorkommen.[65] D.h. Bilder, welche ähnlich der übergebenen Trainingsdaten sind, können nicht erzeugt werden. Dabei kann man zwischen dem partiellen Mode Collapse und dem kompletten Mode Collapse unterscheiden. Der partielle Mode Collapse liegt vor, wenn immer noch eine Teilmenge der Eigenschaften oder Modi der Trainingsdaten in den generierten Bildern berücksichtigt werden. Ein kompletter Mode Collapse wiederum liegt vor, wenn nur noch eine spezifische Eigenschaft (bzw. ein Modus) von Generator erzeugt wird.[48]

Als ein weiteres Problem von GANs wird in der Literatur der Konvergenzverlust genannt, welcher durch einen schwindenden Gradienten des Generators hervorgerufen wird. Dieser schwindende Gradient entsteht, wenn der Diskriminator wesentlich schneller lernt als der Generator. In diesem Fall wird der Diskriminator jedes Bild als Fälschung erkennen und der Gradient des Generators wird immer weiter minimiert, bis der Generator irgendwann kaum bis gar nicht mehr dazulernt.[48, 22, 65, 11] Dies ist besonders häufig der Fall, wenn die Lernrate des Diskriminators sehr hoch ist. [11]

Eine weitere große Herausforderung von GANs sind die fehlenden Evaluationsmetriken. So existieren für GANs keine standardisierten oder universellen Metriken. Die Qualität der erzeugten Daten wird häufig vom Menschen gemessen. Diese Bewertung ist jedoch subjektiv und fehleranfällig.[48] Des Weiteren ist eine menschliche Bewertung der Ergebnisse zeitaufwendig.[65] Eine Möglichkeit, welche häufig für die Bewertung von GANs in Betracht gezogen wird, ist es, die Auswirkung der generierten Bilder auf das Zielnetz zu bewerten. Wenn die synthetisierten Daten z.B. für einen Objekterkenner verwendet werden, wird verglichen, ob sich die Erkennung der Objekte durch die Daten verbessert hat.[48]

Auch wenn das GAN selbst dafür verwendet wird, synthetische Daten zu generieren, welche von weiteren neuronalen Netzen verwendet werden können, so brauchen auch diese einen ausreichenden Trainingsdatensatz für ihre Aufgabe. So können zu wenige Trainingsdaten auch bei GANs zu Problemen, wie Overfitting, führen.[48]

2.4.4 GAN Architekturen

In diesem Unterkapitel werden drei verschiedene GAN-Architekturen näher beschrieben. Es existieren jedoch noch viele weitere. Einen umfangreichen Überblick bekommt man in der Arbeit von Figueira und Vaz.[22]

DCGAN

Das Deep Convolutional Generative Adversial Network(DCGAN) wurde 2015 von Redford et al.[53] entwickelt. Zu dieser Zeit traten die ersten Teilerfolge beim Generieren von synthetischen Bildern auf. Das DCGAN soll die Probleme des normalen und ersten GANs beheben. Bei den Problemen handelt es sich vor allem um eine Instabilität beim Training und verrauschten Ergebnissen des Generators. So waren die generierten Bilder des GANs oft kaum verständlich und damit nutzlos. Die größten Änderungen am originalen GAN sind vor allem das Ersetzen von Pooling-Layern mit strided Convolutional-Layern und dem Nutzen der Batch-Normalization. Auch wurden die Fully-Connected-Layer entfernt und es wird die LeakyRelu-Aktivierungsfunktion für den Diskriminator und die Relu-Aktivierungsfunktion für den Generator verwendet.[53]

Auch wenn das DCGAN bereits 2015 entwickelt wurde, so wird es auch aktuell noch (Stand 2022) in der Forschung verwendet. Ein Beispiel ist die Schadenserkennung von Birnen. Da Bilder von beschädigten Bildern sehr selten sind, haben die Autoren ein GAN verwendet, um mithilfe eines originalen Datensatzes weitere Bilder zu generieren. Dabei haben sie das Training des GANs zu dem Zeitpunkt gestoppt, wenn ein Schwellenwert an Fehlklassifizierungen erreicht worden ist. Das ist der Fall, wenn die Ergebnisse des GANs zu häufig falsch erkannt werden.[77]

Einen ähnlichen Anwendungsfall haben Jin et al. [35]. Hier sollte das DCGAN Bilder mit Beschädigungen bei Klebestrukteren, generieren. Ein Beispiel hierfür sind Risse im Laminat. Da die Ergebnisse des DCGANs den Autoren in Ihrer Qualität nicht ausreichten, haben diese Änderungen am DCGAN vorgenommen. Eine ist beispielsweise jeden Convolutional mit einer Group Norm(GN) zu verbinden.[35]

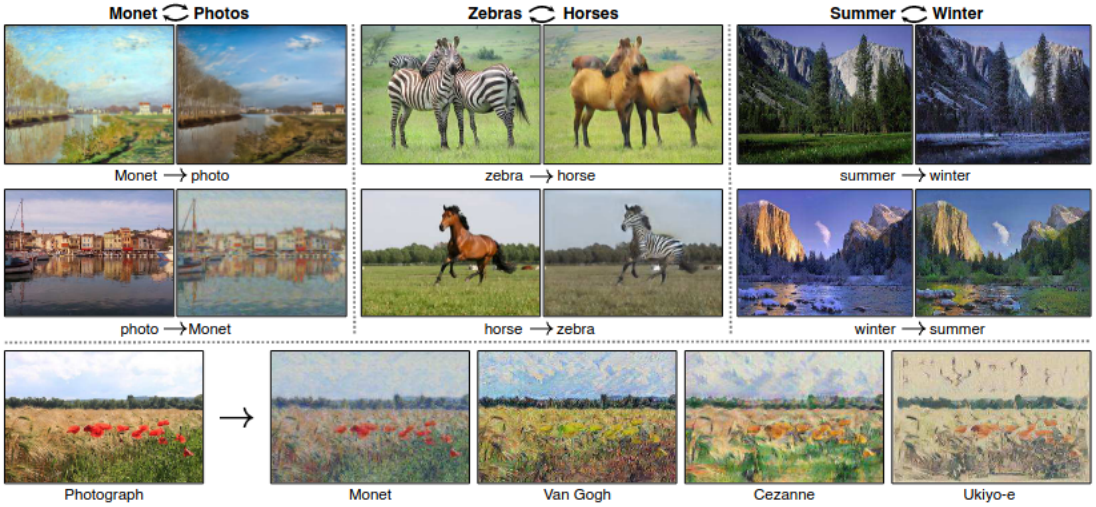

Abbildung 2.1: Beispiele der Verwendung eines CicleGAN von Zhu et al.[78]

CycleGAN

Bei dem CycleGAN handelt es sich um eine von Zhu et al.[78] entwickelte Architektur, welche einen Bild zu Bild Transfer ermöglichen soll. So sollen beispielsweise von einer Domäne Apfel die Äpfel auf dem Bild in die Domäne Orange übertragen werden. Wie die Ergebnisse eines solchen Bildtransfers aussehen, ist in 2.4 zu sehen. Der Name Cycle ergibt sich aus dem Umstand, dass diese Übersetzung auch in dem umgekehrten Fall möglich ist. Folglich können Orangen auch in Äpfel umgewandelt werden. Das CycleGAN wurde aus der Motivation heraus entwickelt, die bisherigen Bild zu Bild Übersetzungen, welche einen gepaarten Datensatz benötigen, auch ohne diesen zu ermöglichen. Dies begründet sich daraus, dass gepaarte Datensätze nur wenig vorhanden und auch schwer zu erstellen sind. Das CycleGAN benötigt folglich keinen gepaarten Datensatz, also wo ein Bild genau einem anderen Bild zugewiesen ist, sondern erlernt die Zusammenhänge aus den gemeinsamen Charakteristiken der Datensätze der verwendeten Domänen. Im Idealfall soll ein Bild, welches von der Domäne X in die Domäne Y übersetzt wird, wieder genauso aussehen wie das Originalbild, wenn das übersetzte Bild wieder in die Domäne X zurückübersetzt wird. Die Ergebnisse sind jedoch nicht ausschließlich positiv. Gute Ergebnisse erzielt das CycleGAN, wenn es für Textur und Farbveränderungen verwendet wird. Eine große Herausforderung stellen Transformationen der Geometrie und weitere größere Veränderungen dar.[78]

Viele weitere Arbeiten nutzen die Architektur in gleicher oder veränderter Form. So z.B. für den Stiltransfer von Gesichtern in andere Personen. Hierbei erhöhten die Autoren die Tiefe des Diskriminators, um damit qualitativere Ergebnisse zu erzielen. Des Weiteren nutzen sie zwei unterschiedliche Diskriminatoren auf beiden Seiten des Netzes. Wichtig war auch, den Hintergrund bei Personen zu entfernen, da das CycleGAN das ganze Bild als zu übersetzendes Objekt sieht.[34] Ein aktuelles Beispiel ist das Übersetzen eines Bildes von einer gesunden, radiologisch gescannten Brust in ein Bild, bei welchem die Brust Covid-19-Symptome aufweist. Das CycleGan wurde hierbei genutzt, um mehr Trainingsdaten für einen Objekterkenner zu generieren, welcher Covid-19 anhand radiologischer Bilder erkennen soll.[8]

AC-GAN

Das Auxilary Condidtional GAN(AC-GAN) baut auf dem Conditional GAN(CGAN) auf und wurde von den Autoren Odena et al. [47] eingeführt. Ziel des AC- und CGAN ist es, zusätzlich zu dem generierten Bild auch ein dazugehöriges Klassenlabel zu generieren. Die Architektur des AC-GAN unterscheidet sich nur leicht von den vergangenen Architekturen, nutzt aber einen auxiliaren Decoder, um die Klassenlabel selbst zu erkennen. Ein weiterer Unterschied ist die verwendete Bildgröße bei den Eingabedaten. So beträgt diese 128x128 und nicht wie üblich 32x32 Pixel. Damit wollen die Autoren dem Verlust der Klassenmerkmale entgegenwirken, welcher bei kleineren Bildern entstehen soll. In der Vergangenheit hatte ein GAN damit Probleme, hochauflösende Bilder zu generieren, wenn sich die Trainingsdaten stark voneinander unterscheiden, da ein Verkleinern der Bilder zu einer geringeren Diskriminierbarkeit der Klassen führt. Des weiteren unterscheiden sich die generierten Bilder stärker, je hochauflösender die Bilder sind. Den Einfluss der Bildauflösung haben sie mit einem Inceptionv3 Model gemessen. Dabei wurde dieses mit rein synthetischen Daten trainiert und die Genauigkeit zwischen hoch- und niedrig auflösenden Bildern verglichen.[47]

Doch auch das Training des AC-GAN wird herausfordernder, je höher die Anzahl der zu trainierenden Klassen wird, wie die Autoren Kang et al.[37] beschreiben. So unterscheiden sich die Bilder einer generierten Klasse meist kaum voneinander. Als Grund hierfür machen sie den zu Anfang schwachen Klassifizierer verantwortlich, da er zu diesem Zeitpunkt häufig falsche Klassenvorhersagen trifft. Dadurch sinkt auch die Wahrscheinlichkeit für jede Klasse. Während des Trainings konzentriert sich das AC-GAN somit mehr auf

die Klassenwahrscheinlichkeiten statt auf die Unterscheidung zwischen realen und synthetischen Bildern. Dies führt im Anschluss häufig zu einem explodierenden Gradienten. Die Autoren konnten dies mit einem Normalisieren der Merkmalseinbettung und einem neuen Data-to-Data-Crossentrophy-Loss lösen.[37]

2.4.5 Data Augmentation bei GANs

Trotz ihrer häufigen Verwendung zum Erweitern eines Trainingdatensatzes sind auch GANs abhängig von der Menge der Trainingsdaten. So nimmt die Qualität eines GANs zu, je größer der Trainingsdatensatz ist. Methoden der Data Augmentation führen bei GANs nicht in jedem Fall zu einer Steigerung der Performance. So können zum Beispiel umgedrehte Bilder auch dazu führen, dass die Bilder, die der Generator erstellt, anschließend auch umgedreht sind. Dieser Prozess wird Augmentation Leak genannt. Dies ist nicht in jedem Fall ein positiver Effekt.[38]

2.5 Diffusionsmodelle

2.5.1 Allgemeines

Eine weitere große Alternative, welche mit GANs auf einem ähnlichen Niveau konkurrieren, sind Diffusionsmodelle.[17] Diffusionsmodelle bestehen aus drei Unterkategorien, die Denoising Diffusion Probalistic Models(DDPM), die noise conditioned score networks (NCSNs) und die Stochastic-differential-equations (SDEs). Die Funktion des DDPMs wird im folgenden Unterkapitel genauer erläutert. Die Gemeinsamkeit der verschiedenen Modelle ist, dass das Diffusionsmodell erlernt, ein komplett verrauschtes Bild schrittweise zu entrauschen.[12] Eine aktuelle Übersicht über Diffusionsmodelle ist in[12] zu betrachten.

2.5.2 Denoising Diffusion Probalistic Model

Ein Denoising Diffusion Probalistic Model (DDPMs)(2.3) erlernt, ein komplett verrauschtes Bild schrittweise zu entrauschen. Für das Training wird dabei zunächst ein Trainingsbild in einer Markow-Kette schrittweise verrauscht, bis es ausschließlich aus purem Rauschen besteht. Anschließend wird ein neuronales Netz in einer parametrisierten

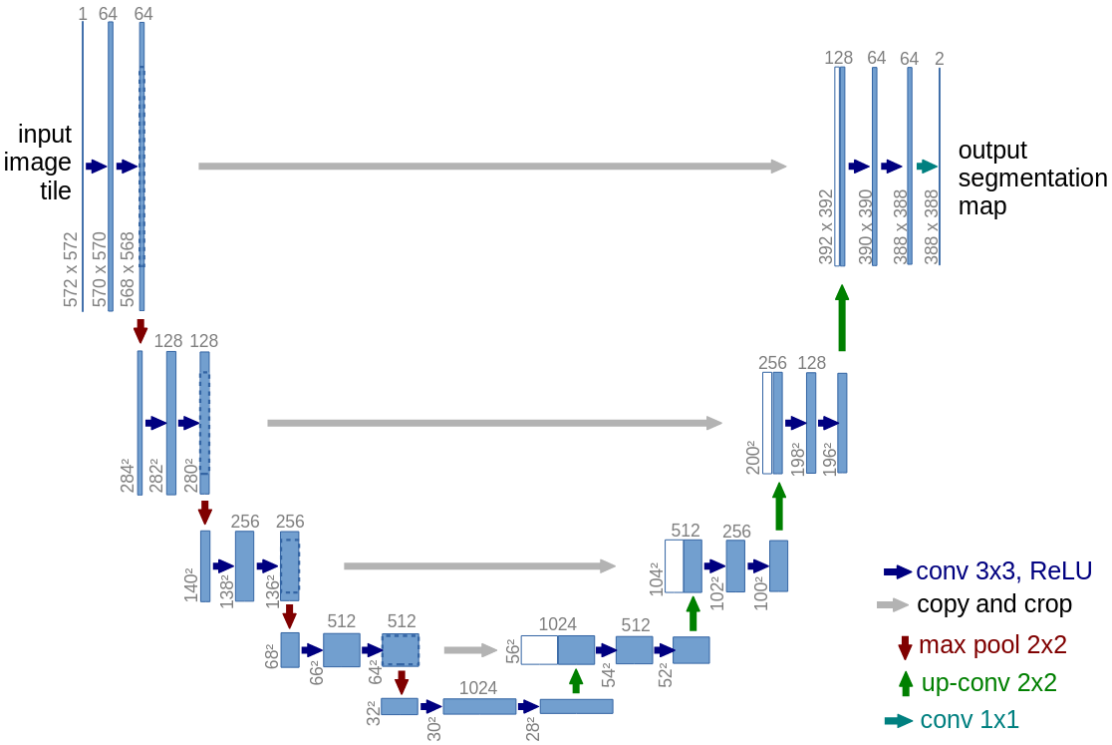

Abbildung 2.2: Die U-Net Architektur von Ronneberger et al. [62]

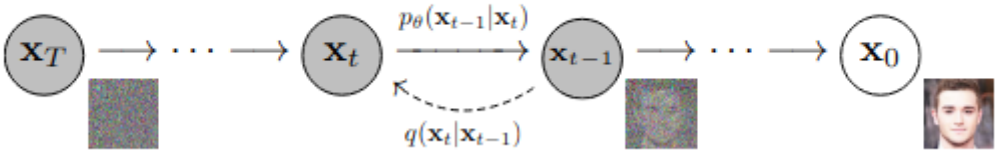

Abbildung 2.3: Funktionsweise eines DDPM von Ho et al. [30]

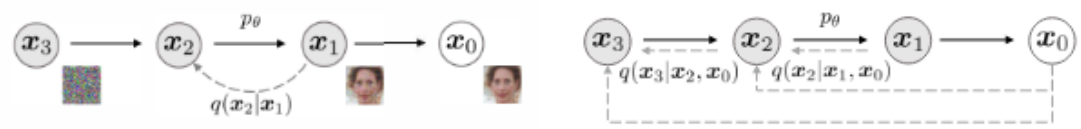

Abbildung 2.4: Unterschiede des DDIM zu Markow-Ketten basierten Diffusionsprozessen von Song et al.[68]

Markow-Kette verwendet, welches anhand der Vartiationsinferenz erlernt, dasselbe Bild wieder schrittweise zu entrauschen. Für die Architektur des Modells kann ein U-Net[62] verwendet werden, welches in 2.2 dargestellt ist.[30]

2.5.3 Denoising Diffusion Implicit Models

Denoising Diffusion Implicit Models (DDIMs) sind Modelle, welche sich an den DDPMs orientieren. Jedoch wird bei dem Prozess des Ver- und Entrauschens keine Markow-Kette mehr verwendet, um die Generierung von Bildern zu beschleunigen. Die Unterschiede finden sich in 2.4 wieder. So ist bei DDIMs der Prozess des Entrauschens nicht mehr ausschließlich vom vorherigen Zustand abhängig, sondern auch von einem geschätzten Anfangsbild.[68]

2.5.4 Abhängige vs unabhängige Diffusion

Bei der abhängigen Diffusion wird die Generierung der Bilder durch einen zusätzlichen Klassifizierer erweitert. Der Klassifizierer kann mittels Klassenbezeichnungen lernen, aber auch ohne. Ein Beispiel dafür ist es, den Klassifizierer mittels semi-supervised-learning zu trainieren. Dabei extrahiert der Klassifizierer die Features in einem Bild und clustert sie mit dem k-mean Algorithmus. Ein von Klassen abhängiges Diffusionsmodell hat eine bessere Qualität, da für jede Klasse die Menge an unterschiedlichen Modi verringert wird.[7] Bei der abhängigen Diffusion wird die Generierung durch den Klassifizierer geleitet. Der Klassifizierer lernt, die verrauschten Bilder einer Klasse zuzuordnen und steuert das Diffusionsmodell in diese Richtung.[17] Das Training eines solchen Klassifizierers verringert die Varianz der Ergebnisse, erhöht aber dessen Qualität. Einen Klassifizierer zu verwenden erhöht des Weiteren den Trainingsaufwand, da dieser zusätzlich zum Diffusionsmodell trainiert werden muss. Hier kann auch kein fertiger Klassifizierer verwendet werden, da der Klassifizierer auf den verrauschten Bildern trainiert wird.[31]

2.5.5 Text basierte Diffusionsmodelle

Text basierte Diffusionsmodelle wie das LDM[60] und das RDM [61] sind in der Lage mittels Texteingaben Bilder zu generieren. Das LDM macht sich dabei eine Repräsentation des Latenten-Raumes zunutze, um auch Bilder mit hoher Auflösung zu generieren. Ein bekannter Vertreter für textbasierte Diffusionsmodelle ist Stable Diffusion eine spezielle Version des LDM. Die Ergebnisse des LDM sind dabei nah an den Anforderungen in der Texteingabe des Nutzers.[12]

2.5.6 Vergleich mit GANs

Diffusionsmodelle generieren Daten, welche eine höhere Diversität als die von GANs aufweisen.[72] Die Forschung von Diffusionsmodellen ist jedoch noch in ihrem Anfangsstadium.[75] So schlagen sie GANs bereits in manchen Bereichen der Bildgenerierung, liegen aber noch bei komplexeren Generierungsprozessen hinter diesen zurück. Die Autoren Dhariwal et al. vermuten als Grund die gelieferte hohe Qualität der Ergebnisse von GANs, welche dafür jedoch Diversität der Ergebnisse opfern. Eine weitere Vermutung ist, dass GAN Architekturen durch intensivere Forschung bereits sehr verfeinert wurden und deshalb besser in komplexen Fällen funktionieren. Des Weiteren sind Diffusionsmodelle aktuell noch langsamer als GANs.[17] In einem Vergleich brauchte ein DDPM 20 Stunden um 50000 Bilder zu generieren, während ein GAN dafür nur eine Minute braucht. Allerdings wurde mit einem DDIM ein Modell geschaffen, welches Bilder 10- bis 20-mal so schnell generiert wie ein DDPM.[68]

2.6 Weitere Alternative

Ein Beispiel für eine weitere Alternative sind conditional invertable neural networks(iNNs), welche aufgrund ihrer Architektur von Natur aus die Fähigkeit haben, ein Bild auch wieder zurückzuübersetzen. D.h. sie sind bijektiv. Diese sollen vor allem das Problem des Mode Collapse von GANs lösen und auch mehr Unterscheidbarkeit der generierten synthetischen Daten liefern.[5] Des Weiteren ist das Training stabil und benötigt keinen gegnerischen Lernprozess wie z.B. GANs.[4] Ihre Garantie für die Inversität zwischen In- und Output erhalten sie durch die Verwendung von Coupling Layern. Diese sind invertierbar, da sie den Input in zwei Teile aufteilen. Ein Teil wird transformiert und der andere wird nicht verändert.[16]

2.7 Metriken für die Messung der Qualität von generativen Modellen

Borji[10] fasst in seiner Arbeit gängige Metriken für die Messung der Qualität der generierten Bilder von GANs, zusammen. Er macht deutlich, dass es hier keine eindeutige Metrik für die Messung gibt. Es gibt Metriken, welche darauf spezialisiert sind, die Qualität der Objekte auf den Bildern zu messen, aber auch Metriken welche bewerten, ob sich die generierten Bilder voneinander unterscheiden. Die richtige Metrik richtet sich folglich nach den Anforderungen, welche der Nutzer an das Generierungsmodell richtet.[10]

Auch wenn Borji sich in seiner Arbeit an die Generierung von Bildern mit GANs richtet, so lassen sich viele Metriken auch auf Diffusionsmodelle übertragen, da diese ebenfalls Bilder generieren und z.B. auch hier die Qualität der Objekte auf den Bildern wichtig sein kann.

2.7.1 Inception-Score

Der Inception-Score ist für die Messung der Qualität von GANs weit verbreitet. Dabei wird ein vortrainiertes Inception Netz verwendet. Das vortrainierte Inception Netz wurde auf den ImageNet Datensatz trainiert und soll sicherstellen, dass die generierten Bilder möglichst unterschiedlich und qualitativ sind. Dies soll es mit der Messung der durchschnittlichen Kullback-Leibler-Divergenz aus der abhängigen und marginalen Verteilung gewährleisten. Der Inception Score besitzt jedoch auch Schwächen, so ist er beispielsweise abhängig von der Auflösung der Bilder. Aufgrund seiner Eigenschaft, Objekte zu erkennen, eignet sich das Iception-Netz nur für die Bewertung der Qualität der Objekte auf den Bildern, jedoch nicht dafür, zu messen wie realitätsnah diese sind.[10]

2.7.2 GAN Quality Index

Der GAN Quality Index (GQI) ist eine weitere Möglichkeit, die Qualität von GANs zu messen. Dabei wird ein Klassifizierer mit den realen Daten erstellt. Dieser Klassifizierer wird zunächst dafür genutzt, Bilder mit schlechter Qualität bereits aus einem Datensatz mit generierten Bildern zu filtern. Mit den gefilterten Bildern wird anschließend ein weiterer Klassifizierer trainiert. Anschließend werden die beiden Klassifikatoren auf denselben Testdatensatz angewandt. Der Testdatensatz enthält eine Menge von realen

Bildern. Dabei wird die Genauigkeit einer richtigen Klassifizierung bei beiden Klassifikatoren berechnet. Im Abschluss wird die Genauigkeit des mit generierten Bildern trainierten Klassifikators durch die Genauigkeit des Klassifikators mit realen Bildern geteilt. Das Ergebnis wird danach mit hundert multipliziert und ergibt den GAN Quality Index.[76] Der GAN Quality Index ist jedoch nicht so gut darin, Overfitting eines Modells zu erkennen wie der Inception-Score.[10]

2.8 Softwarearchitektur

2.8.1 Allgemeines

Mit der steigenden Verbreitung des maschinellen Lernens bekommt die Softwarearchitektur eine neue Aufgabe. So muss neben der Erfüllung der nicht-funktionalen und funktionalen Anforderungen zusätzlich darauf geachtet werden, dass auch die Herausforderungen von maschinellem Lernen bewältigt werden.[46]

ML basierte Software ist meistens aufgeteilt in zwei Teilsysteme. Ein Teilsystem behandelt den Anteil des maschinellen Lernens, dazu gehören z.B. die Modelle oder die Datenvorbereitung. Das andere Teilsystem ist für die Komponenten, die Konnektoren und die Kommunikation der einzelnen Bestandteile zuständig. Das ML-Teilsystem generiert die Daten, welche vom Software-Teilsystem weiterverarbeitet werden. Der Softwarearchitekt muss das Zusammenspiel der beiden Teilsysteme koordinieren.[46]

Um eine Softwarearchitektur zu repräsentieren, werden häufig UML-Diagramme verwendet. Das ist bei maschinellem Lernen nicht immer möglich, da dieses mathematisches Verständnis erfordert. Die Softwarearchitektur fachfremden Stakeholdern zu erklären wie z.B. Geschäftspartnern, bringt folglich Probleme mit sich.[46]

2.8.2 Architektur-Pattern

Model-View-Controller Architektur

Die Model-View-Controller Architektur(MVC) teilt die Aufgaben einer Architektur in ein Model, eine View und einen Controller auf. Das Model ist ein unveränderlicher Bestandteil einer Architektur und besitzt keinerlei Kenntnisse über die anderen Bestandteile

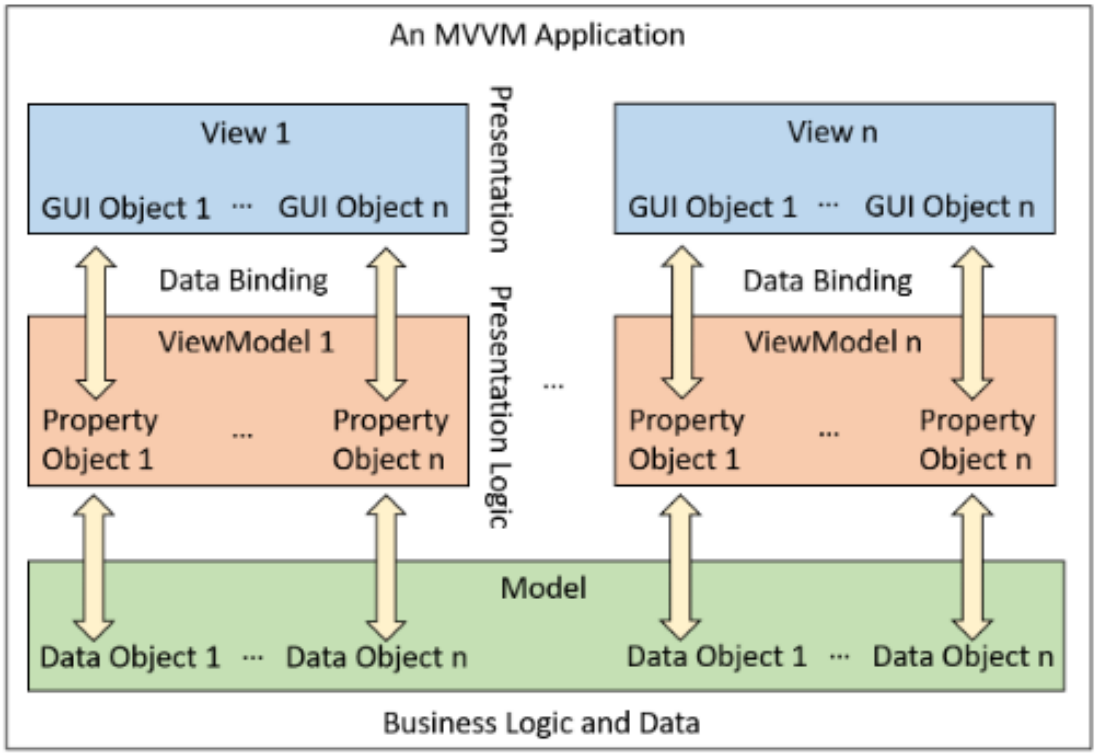

Abbildung 2.5: Das Model View View-Model Pattern von sheikh et al.[67]

der Anwendung. Das Model besteht aus einer Sammlung von verschiedenen Klassen. Die View ist für die Darstellung des Models nach außen zuständig. Sie weiß aber nur, dass ein bestimmtes Model existiert und von welcher Natur das Model ist. Der Controller manipuliert die View mit Events. Events entstehen, wenn sich z.B. Instanzvariablen einer Klasse im Model ändern. Die Events werden dann von der View behandelt, indem eine bestimmte Aktion ausgeführt wird.[14]

Model View View-Model Architektur

Die Model View View-Model(MVVM) Architektur besteht aus drei Komponenten, welche in 2.5 dargestellt sind. Der View, welche für die Interaktion mit dem Nutzer zuständig ist. Einem View-Model, welches Datenänderungen am Model verwaltet und einem Model, welches für die Datenlogik wie z.B. Lese- und Schreibzugriffe auf die Datenbank verantwortlich ist. Die View besitzt keinerlei Informationen über das Model und das Model keinerlei Informationen über die View. Die Kommunikation der View mit dem View-Model

ist automatisiert mittels Datenbindungen. So hält das View-Model einen aktuellen Zustand der Daten fest, welche mit der View verbunden sind. Ändert sich etwas an den Daten, wird automatisiert ein Prozess hervorgerufen. Das View-Model fungiert quasi als Brücke zwischen der View und dem Model.[67]

Client/Server Architektur

Im Falle der Client/Server Architektur besteht die Gesamtanwendung aus zwei über ein Netzwerk verbundene Bestandteile, dem Client und dem Server. Der Client schickt Anfragen an den Server, während der Server diese entgegennimmt bzw. auf diese wartet. Die Client/Server Architektur kann aus mehreren Schichten bestehen. Der Server Bestandteil kann hierbei mehrere Stufen haben. Z.B. kann ein Backend-Server, welcher die Logik enthält noch auf einen Datenbankserver zugreifen. In diesem Fall würde man von einer drei-stufigen(engl. three-tier) Client/Server Architektur sprechen. Für die Kommunikation der Bestandteile untereinander werden Netzwerkprotokolle wie z.B. HTTP oder HTTPS verwendet.[40]

Microservice Architektur

Bei der Microservice Architektur besteht die Anwendung aus vielen kleinen, voneinander unabhängigen Services. Dies ist ein starker Kontrast im Gegensatz zu einer Monolithischen Architektur bei welcher alle Komponenten in einer Anwendung zusammen und abhängig voneinander implementiert sind. Dabei hat ein sogenannter Microservice genau eine definierte Aufgabe bzw. Verantwortung. Auf die Microservices kann z.B. mittels Representational State Transfer(REST) zugegriffen werden. Dadurch ergeben sich einige Vorteile im Gegensatz zu einem System, welches alle Services in einem System implementiert hat. Beispielsweise sind Microservices leichter zu aktualisieren, da nur eine Komponente aktualisiert werden muss und diese bis auf die Schnittstellen keine Abhängigkeiten mit anderen Microservices teilt. Ein auf Microservices basierende Anwendung kann auch leicht erweitert werden, indem ein Microservice dem System hinzugefügt wird. Eine komplette Unabhängigkeit ist jedoch schwer zu realisieren, wenn Datenintegrität gewährleistet sein soll.[25]

Hierfür gibt es zwei verschiedene Pattern. Eines ist das shared Database Pattern, bei welchem sich die einzelnen Microservices dieselbe Datenbank teilen. Das geht jedoch nur, wenn die Microservices dieselbe Art von Datenbank verwenden. Also beispielsweise

eine NoSQL Datenbank wie MongoDB. Es ist von Vorteil dieses Pattern zu verwenden, wenn verschiedene Microservices dieselben Daten benötigen.[58] Eine andere Möglichkeit ist das Database per Service Pattern, bei dem jeder Microservice seine eigene Datenbank besitzt. Dadurch können Microservices den Datenbanktyp verwenden, welcher am besten für sie geeignet ist. Jedoch sind Datenverknüpfungen hier sehr schwer zu realisieren.[57]

Microkernel Architektur

Die Mikrokernel oder auch Pluginarchitektur besteht üblicherweise aus einem gemeinsamen Softwarekern, welcher von Plugins erweitert wird. Ein Beispiel hierfür ist, wenn z.B. verschiedene Geräteunterstützung angeboten wird. Dann erfolgt der Teil, der nicht abhängig von bestimmten Endgeräten ist im Primärsystem und der vom Endgerät abhängige Teil wird in Form von Plugins realisiert. Wie die Plugins dabei mit dem Primärsystem kommunizieren ist dabei nicht festgelegt. Dabei können Endpoint Verbindungen, aber auch REST-Schnittstellen verwendet werden. Der Vorteil einer Microkernel Architketur ist vor allem, dass sich diese Architektur leicht erweitern lässt, nämlich in der Form, dass Plugins hinzugefügt werden. Auch können die Plugins mehr oder weniger unabhängig von dem Primärsystem sein, dies wäre z.B. bei einer Anbindung mit dem REST der Fall.[56]

2.8.3 Architektur-Pattern für maschinelles Lernen

Allgemein

Washizaki et al.[73] haben in ihrer Arbeit einen ausführlichen Überblick über gängige Architiektur-Pattern geschaffen. Allgemein eignet sich für Anwendungen des maschinellen Lernens eine Microservice Architektur. Die einzelnen Microservices sollen hierbei aktiviert werden, wenn diese ihren benötigten Input übergeben bekommen. Auch unabhängig von einer Microservice Architektur wurden Pattern definiert. Eine für die Größe des Projekts relevante Auswahl,der Pattern von [73] findet sich in der folgenden Liste:

Distinguish Business Logic from ML-Models Der Prozess der Datenverarbeitung ist abgekoppelt von den anderen Prozessen der Anwendung.

Data-Algorithm-Serving-Evaluator Einzelne Bestandteile der Datenverarbeitung werden vom System verbunden.

Closed-Loop-Intelligence Interaktionen sollen klare Ausgaben liefern.

Design-Pattern

Ebenfalls liefern Washizake et al.[73] einen Überblick über verschiedene Design-Pattern für ML-Anwendungen. Eine Auswahl findet sich in der folgenden Auflistung wieder:

Pakages into Common-APIs Die Verwendung von Common-APIs verhindert eine Abhängigkeit der Anwendung von einzelnen Paketen, sodass die gesamte Anwendung nicht wegen der Änderung eines Paketes geändert werden muss.

Decouple Training Pipeline from Production Pipeline Verwenden von unterschiedlichen Maschinen für das Training und die Produktion.

Reuse Code between Training Pipeline and Serving Pipeline Wiederverwendung von Code welche beide Pipelines benötigen.

Seperation of Concerns and Modularization of ML Components Kleine Module mit so wenig Komplexität wie möglich erstellen, welche von Modulen mit höherer Komplexität verwendet werden.

Anti-Pattern

Des Weiteren beschreiben Washizake et al.[73] auch verschiedene Anti-Pattern. Eine Auswahl findet sich in der folgenden Liste wieder:

Big Ass Script Architekture Nicht alles in ein einzelnes Skript packen.

Multiple Language Smell Keine verschiedenen Programmiersprachen für die ML-Prozesse verwenden

Glue-Code Das Programm nicht von einzelnen Paketen abhängig machen.

2.8.4 Frontend-Frameworks

Allgemein

Die Verwendung eines Frontend-Frameworks erleichtert die Entwicklung. Gerade bekannte Frameworks besitzen durch eine große Community hilfreiche Quellen wie Tutorials, um damit schnell ein Frontend zu entwickeln. Am weitesten verbreitet sind React, Angular und Vue.js. Alle drei basieren auf JavaScript als Programmiersprache. Sie können jedoch auch in TypeScript implementiert werden. Angular ist sogar standardmäßig in TypeScript wird allerdings beim Kompilieren in JavaScript zurückübersetzt. [70]

Angular

Angular ist eine Plattform, mit welcher eine komponentenbasierte Entwicklung unterstützt wird. Sie eignet sich von kleinen Projekten bis zu großen Projekten. Angular bietet eine Menge an Bibliotheken, welche bei steigender Komplexität und Größe des Projektes hinzugefügt werden können.[24]

Blazor

Bei Blazor handelt es sich um Framework für die Entwicklung einer Web App, welche im Browser ausgeführt wird. Dabei wird Blazor von jedem Browser unterstützt. Die Programmiersprache von Blazor ist C# und bietet den Vorteil auch jede .Net Bibliothek in Blazor zu integrieren. Sie ist hierbei von Vorteil, wenn ein Entwickler bereits Erfahrungen im C# Backend Bereich hat und nicht auf JavaScript umsteigen möchte.[29]

React.js

Bei React.js handelt es sich um ein JavaScript Frontend-Framework, in welches es sich leicht einarbeiten lässt. Es nutzt das MVC Prinzip und ein virtuelles DOM. Das virtuelle DOM wird im Speicher und nicht im Browser gespeichert. Durch die Verwendung von diesem, soll die Performance erhöht werden, falls große Datenmengen geändert werden. React.js eignet sich dafür, wieder verwendbare Komponenten und modulare Nutzerinterfaces zu entwickeln.[1]

Vue.js

Vue.js ist ein in der Programmiersprache JavaScript geschriebenes Single-Page-Webapplication Framework(SPA). Es basiert auf dem MVVM-Prinzip(Model View View-Model). Dabei verbindet ein View-Model die Daten und die View, während diese selbst nicht untereinander kommunizieren können. Das Framework ist datengesteuert, komponentenbasiert und verfolgt eine bottom-up Strategie. Des Weiteren lässt sich Vue.js auch mit anderen Bibliotheken kombinieren. Ein großer Vorteil von Vue.js ist die Synchronität des DOM mit den Daten, denn sobald die Daten verändert werden, aktualisiert sich auch das DOM.[43]

2.8.5 Backend-Frameworks

Allgemein

Um maschinelles Lernen auf einem Server zu implementieren, eignet sich die Programmiersprache Python. Dies liegt vor allem daran, dass die meiste Forschung im Bereich des maschinellen Lernens in Python betrieben wird. Aber Python besitzt auch darüber hinaus weitere Vorteile. So wird Python von einer großen Community unterstützt, ist einfach zu erlernen und eignet sich außerdem für die Webentwicklung.[64]

Django

Django ist ein sehr schnelles Full-Stack Framework, welches in der Programmiersprache Python geschrieben ist. Es bietet damit Unterstützung für die Entwicklung einer ganzen App und beinhaltet damit UI-Elemente, Datenmanagement und zusätzlich auch Sicherheitsaspekte.[64] Bei Django handelt es sich um ein MVC Framework.[32]

Flask

Flask ist ein leichtes Non-Fullstack-Framework mit einem eingebauten Server für die Entwicklung. Bei diesem ist mehr eigene Arbeit und Organisation nötig als bei einem Framework wie Django.[64] Dafür überlässt einem Flask auch mehr Freiheiten als andere Python Frameworks. In Flask lassen sich auch Modelle des maschinellen Lernens integrieren, wie beispielsweise Menon et al. mit einem Modell der Diabetes Erkennung zeigen.[45]

Flask ist als Framework sogar noch schneller als Django. Aufgrund seiner Leichtgewichtigkeit hat es auch bei größeren Datenmengen weniger Probleme als Django.[32]

2.8.6 Datenbanken: MongoDB vs MySQL

Um Daten zu Speichern können MySQL Datenbanken oder NOSQL Datenbanken wie MongoDB verwendet werden. Im Fall von MySQL werden die Daten in tabellarischer Form gespeichert. Mithilfe von Verknüpfungen können Datenduplikate vermieden werden. Der Zugriff auf MySQL Datenbanken erfolgt mit spezifischen Schemas. MongoDB speichert die Daten in Textform, im sogenannten JSON-Format. Das JSON-Format wird auch häufig für den Austausch beim Datentransfer verwendet. MongoDB bietet gegenüber MySQL den Vorteil, dass Vorgänge wie Einfügen, Auswählen, Updaten und Löschen viel schneller durchgeführt werden. Allgemein sollte MongoDB verwendet werden, wenn in der Anwendung sehr viele Daten verarbeitet werden müssen.[19]

2.9 Verwandte Arbeiten und Abgrenzung

In den vorherigen Kapiteln, welche einen Überblick über die Möglichkeiten zur Generierung von Bildern mit Diffusions- und GAN Modellen lieferten, wurden bereits in diesem Kontext einige verwandte Arbeiten genannt. Schließlich ist auch ein GAN- oder Diffusionsmodell ein System, mit welchem man fotorealistische Bilder generieren kann. Hier unterscheidet sich die vorliegende Arbeit vor allem in der Bereitstellung der Bildgenerierung. Die Verwendung eines einzelnen Genrationsmodells benötigt Fachkenntnisse im Bereich des maschinellen Lernens, da der Code selbst verwendet wird. Des Weiteren konzentrieren sich die erwähnten Arbeiten meist darauf, bestehende Netze zu verfeinern oder neue Netze zu entwerfen. Die vorliegende Arbeit stellt eine Methode für die Bereitstellung von Bildgeneratoren dar, bei denen der Nutzer keine Fachkenntnisse im Bereich des maschinellen Lernens benötigt. So kann z.B. ein AR-Entwickler, welcher lediglich die Fachkenntnisse besitzt, einen Objektdetektierer für das Tracking zu entwickeln, die Anwendung verwenden, um damit Trainingsbilder für den Objekterkenner oder Objektdetektor zu generieren. Zusätzlich stellt die Anwendung zwei verschiedene Netze als Generationsmöglichkeit bereit und bietet die Möglichkeit weitere nach dem selben Schema funktionierende Netze der Anwendung, ohne großen Änderungsaufwand zur Verfügung zu stellen.

Die verwandten Arbeiten, welche mit dieser Arbeit am meisten Ähnlichkeiten aufweisen, sind mehr im Open-Source oder Geschäftsbereich zu finden. Ein Beispiel hierfür ist CreateML[2]. Wie bereits in [20] gezeigt wurde, lässt sich mit CreateML ohne Fachkenntnisse des maschinellen Lernens ein Objekterkenner entwickeln. Für das Training eines Objekterkenners müssen lediglich die Daten in einem richtigen Format an die CreateML UI übergeben werden, damit ein solches Netz generiert wird. CreateML beinhaltet jedoch keine Möglichkeit einen Bildgenerator zu entwickeln. Des Weiteren lassen sich die trainierten Netze auch nur in Apple-Anwendungen verwenden.[2] Es existieren jedoch auch Webseiten wie DeepAI[15] oder DALL-E-2 von OpenAI[49], mit welchen sich anhand von Text ein oder mehrere Bilder generieren lassen. OpenAI verwendet für die Generierung ein Constrative Modell. DALL-E-2 bietet zusätzlich auch Möglichkeiten des Stiltransfers, um damit beispielsweise ein realistisches Bild in ein künstlerisches umzuwandeln oder auch um Wettereffekte hinzuzufügen.[54] OpenAI ist deshalb sehr mächtig. Die Bildgenerierungswebseiten lassen sich auch ohne Fachkenntnisse des maschinellen Lernens verwenden.[49] Die Webseiten sind jedoch nicht frei von Kosten. Auch wenn DeepAI zwar die Möglichkeit bereitstellt, kostenlos einzelne Bilder zu generieren, so muss man hier z.B. für einen API-Zugriff bezahlen.[15] DALL-E-2 wiederum stellt dem Nutzer bei der Anmeldung 50 Credits zur Verfügung und pro Monat zusätzliche 15 Credits. Mit einem Credit können 4 Bilder pro Text generiert werden. Sind die gratis Credits aufgebraucht, müssen weitere gekauft werden oder es muss auf den nächsten Monat gewartet werden.[49] Die Verwendung der Bildgenerator Webseiten eignet sich somit hauptsächlich für den privaten Gebrauch und nur bedingt für das Aufbessern eines Trainingsdatensatzes, denn dafür würde der kostenlose Bereich schnell ausgeschöpft sein, da hier sehr viele Bilder benötigt werden. Ein weiteres Problem bei der Verwendung von Bildgenerator-Webseiten ist die nicht eindeutig festgelegte Domain der generierten Bilder. Es kann nur bedingt Einfluss auf die Art des generierten Bildes genommen werden. Dies ist in Abbildung 2.6 und 2.7 zu sehen. Hier wurde im Fall von DeepAI ein gezeichnetes Bild erstellt, obwohl dies gar nicht gewünscht war. Im Fall der Verwendung von OpenAI wurden zwar Bilder von Sanddorn generiert, jedoch aus einer sehr nahen Ansicht. Sollen z.B. Bilder generiert werden, welche den ganzen Baum anzeigen, so generiert OpenAI nur Bilder in der Nahsicht. Dies ist beim Training eines Objekterkenners nicht immer gewünscht.

Die vorliegende Arbeit unterscheidet sich folglich von den bisherigen Bildgeneratoren darin, dass der Nutzer die Domain der zu generierenden Bilder selbst festlegen kann und damit auch genau die gewünschten Bilder erhält.

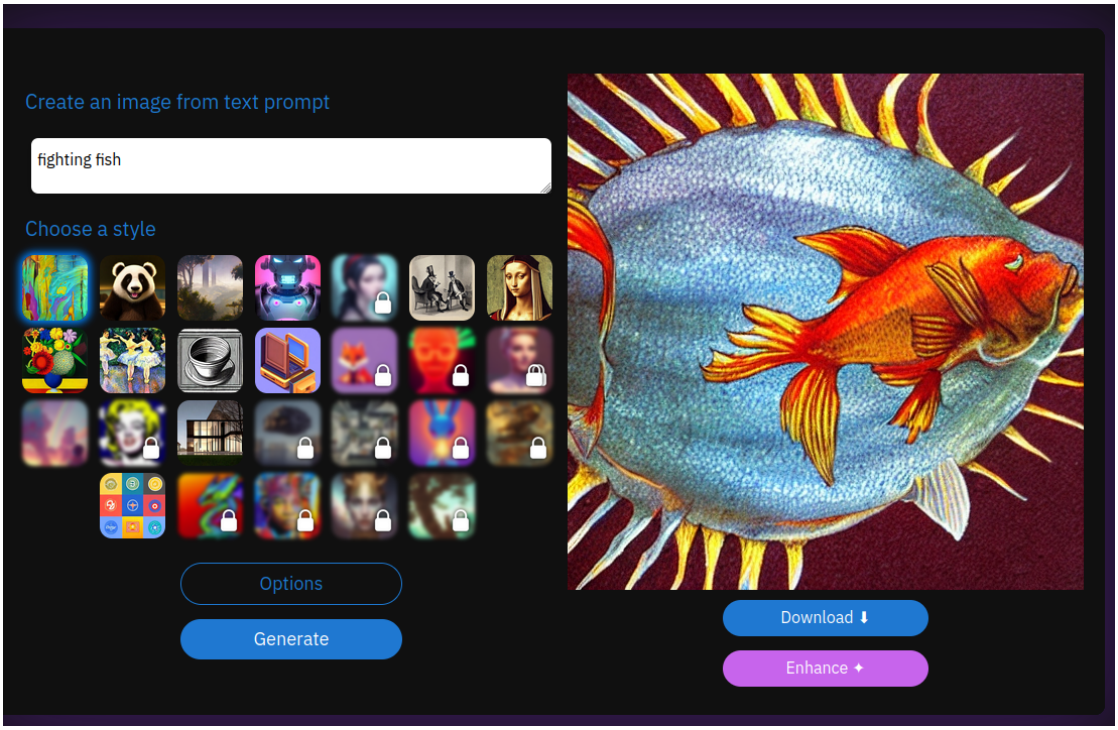

Abbildung 2.6: Ein mit DeepAI[15] generiertes Bild

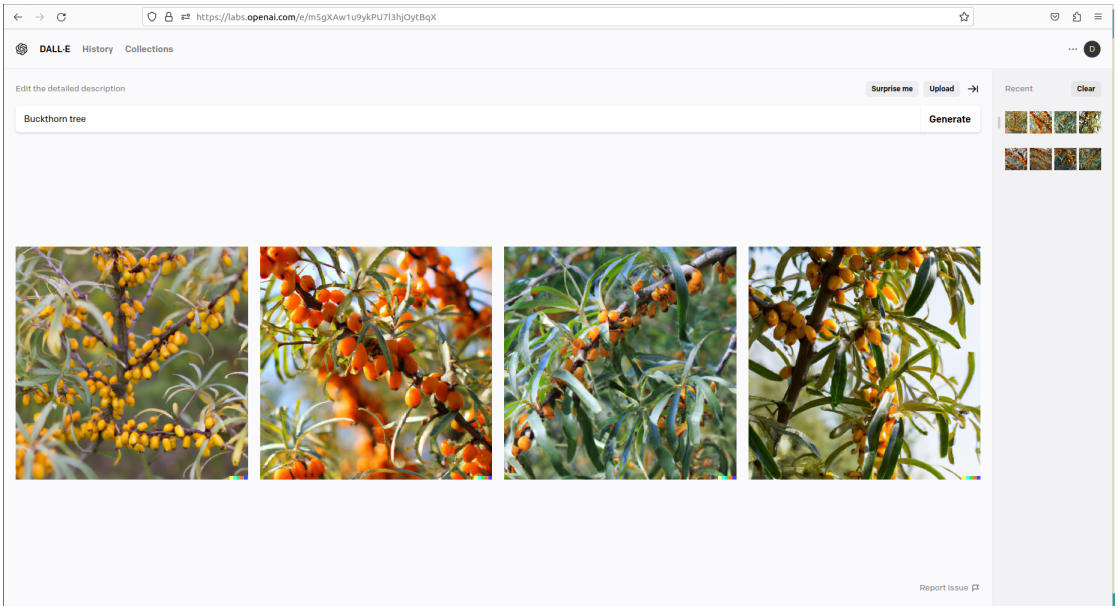

Abbildung 2.7: Ein mit DALL-E-2[49] generiertes Bild

2.10 Gesellschaftliche Auswirkungen

Generative AI ist ein präsentes Diskussionsthema in der Forschungsliteratur und in den Medien. Vor allem aufgrund ihrer Auswirkungen auf die Gesellschaft. Eine Auswirkung von generativer KI in Bezug auf Bilder sind Deepfakes. Hier ist es mit Diffusionsmodellen möglich, Bilder zu generieren, die mit dem menschlichen Auge kaum von Originalbildern zu unterscheiden sind. Böswillig können solche Bilder genutzt werden, um damit Desinformationskampangen zu starten, womit die öffentliche Meinung gesteuert werden könnte.[59] Fehlende rechtliche Rahmen und eine rasch angestiegene Entwicklung generativer AI führte dazu, dass über 1000 Unterzeichner einen Entwicklungsstop dieser fordern. Damit soll Regierungen genügend Zeit gegeben werden, die rechtlichen Rahmenbedingungen festzulegen.[3]

Ein Problem von Modellen des maschinellen Lernens beim Training ist ein hoher CO2-Fußabdruck durch die Verwendung großer Rechenpower. Dieser CO2-Abdruck kann jedoch durch einige Methoden, wie z.B. die Verwendung von effizienten Modell-Architekturen, die Nutzung einer Cloud als Trainingsumgebung oder erneuerbarer Energie für das Training gesenkt werden. Eine weitere Möglichkeit ist die Verwendung spezieller Hardware für das maschinelle Lernen, wie z.B. einer A100 oder V100 GPU. [50]

2.11 Zwischenfazit

2.11.1 Methoden zur Datengenerierung

Wie sich anhand der Analyse erkennen lässt, sind die Funktionen von GANs sehr vielfältig. So ist beispielsweise das DCGAN nicht abhängig davon, ob die Trainingsdaten gelabelt sind oder nicht. Es erstellt die Bilder anhand der Bilder des Trainingsdatensatzes. Um es zu verwenden, müsste folglich lediglich ein Trainingsdatensatz gesammelt und dann dem Model zum Training übergeben werden. Das DCGAN ist deshalb gut für die Erfüllung der Ziele der Anwednung geeignet. Das AC-GAN wiederum ist gut für den Fall, wenn man verschiedene Klassen der zu generierenden Objekte auswählen will. Da die Performance jedoch mit steigender Anzahl an Klassen abnimmt, wäre es nicht gut dafür geeignet, ein generisches Framework aufzubauen, welches um Objekte erweitert werden kann. Schließlich wäre es von erheblichem Nachteil, wenn das Modell um eine weitere Objektklasse erweitert wird, aber die Qualität aller Modelle anschließend abnimmt.

Mit dem CicleGAN kann man das Problem der wechselnden Jahreszeiten oder Wetterbedingungen umgehen. Sie sind allerdings nicht dazu geeignet, einen Trainingsdatensatz zu erweitern, welcher unabhängig von Jahreszeiten oder Wettereffekten ist. Die in dieser Arbeit aufgeführten GAN Architekturen sind folglich alle dazu geeignet, einen bestehenden Trainingsdatensatz zu erweitern, aber nicht jedes für die Integration in ein erweiterbares System. Hier muss allerdings noch überprüft werden, wie groß der verwendete Datensatz sein muss, damit das GAN hochwertige Bilder generiert. Eine besondere Herausforderung wird der Mode Collapse. Schließlich soll das GAN möglichst unterschiedliche Ergebnisse generieren. Ob das klappt, lässt sich jedoch nur nach dem Training beantworten. Das könnte zu einer Frustration der Nutzer führen.

Eine Option möglichst unterschiedliche Bilder zu generieren, bieten im Gegensatz zu GANs die Diffusionsmodelle. Die Diffusionsmodelle sind zwar noch recht neu und nicht so perfektioniert wie GANs, jedoch generieren sie Bilder mit größerer Unterscheidbarkeit untereinander. Allerdings haben sie ein großes Geschwindigkeitsproblem und generieren die Bilder viel langsamer als GANs.

Sich zwischen einem GAN und einem Diffusionsmodell zu entscheiden, ist schwierig. Gerade da Nutzer die Bildgenerierung aus unterschiedlichen Gründen mit unterschiedlich mitgebrachter Zeit nutzen werden. Je nach Anwendungsfall könnte mal ein GAN und mal ein Diffusionsmodell die richtige Wahl sein. Demzufolge lohnt sich die Implementierung beider Modelltypen. Dabei stechen vor allem das DCGAN sowie das DDPM und DDIM hervor. Beide Netztypen verwenden lediglich einen Datensatz an Bildern und könnten damit eine nur minimal veränderte, oder im besten Fall eine identische Trainingspipeline verwenden.

Durch die Verwendung eines DCGAN, DDPM oder DDIM werden auch negative gesellschaftliche Auswirkungen minimiert, da es sich bei den drei Modell-Architekturen, um unüberwachtes Lernen handelt. Die Ergebnisse der Modelle werden nur das generieren, was in ähnlicher Form bereits in den Trainingsdaten vorhanden ist. Dadurch ist die Generierung von Deep Fakes nur schwer möglich. Um die CO2 Emissionen so gering wie möglich zu halten, kann das rechenintensive Training der Modelle auf die speziell auf das maschinelle Lernen ausgerichtete Infrastruktur vom CSTI[13] ausgelagert werden. Lediglich die ersten Tests, ob das Modell auch funktioniert, müssten dadurch auf einem lokalen Rechner durchgeführt werden, sowie das Zusammenspiel der Anwendung selbst.

2.11.2 Architektur- und Designmöglichkeiten

Im Kapitel der Softwarearchitektur wurden verschiedene Pattern, Architekturstile und Frameworks für die Entwicklung einer Anwendung mit Bestandteilen des maschinellen Lernens beschrieben. Besonders sticht hier das Anti-Pattern des Glue-Codes hervor. Schließlich soll die zu entwickelnde Anwendung möglichst mit verschiedenen Modellen erweiterbar sein. Das ist aber gerade ein Problem, wenn Modelle mit einer Bibliothek wie z.B. TensorFlow entwickelt werden. Bei einem Monolithen wären die Modelle dann abhängig von einer spezifischen TensorFlow Version. Folglich ist es für diese Anwendung besonders wichtig, Glue-Code zu vermeiden. Dafür könnte sich auch die empfohlene Microservice Architektur eignen, da die einzelnen Modelle und so auch die verwendeten Bibliotheken in einem einzelnen Microservice kapseln lassen würden. Damit könnte ein Modell die TensorFlow Version x und das andere Modell TensorFlow Version y verwenden.

Die Wahl eines Frameworks für die Frontend Entwicklung ist recht frei. Alle bieten hier verschiedene Vorteile. Vue könnte sich hier aufgrund seiner gemischten Eigenschaften von React.js und Angular für die Entwicklung eignen. Da es ähnlich einfach ist wie React, würde auch eine nachträgliche Weiterentwicklung für neue Entwickler schneller möglich sein als bei Angular. Es bietet aber auch schon die klassische Architektur des Model View View-Models und damit kann diese leichter verwirklicht werden. Mit Blazor wiederum können Codebestandteile, in Unity, aufgrund derselben Programmiersprache, wiederverwendet werden.

Bei der Wahl des Backendframeworks kommt es darauf an, ob man eine Monolithen, Mikrokernel oder Microservice Architektur verwenden will. Django könnte im Falle eines Monolithen mehr Hilfe bieten, da es mehr Funktionen als Flask bereitstellt. Jedoch wäre Django für Microservices wahrscheinlich ein Overload, da diese nur einen Service bereitstellen sollen. Hierfür würde sich Flask aufgrund seiner Schnelligkeit und schnellen Implementierung besser eignen.

3 Design und Implementierung

3.1 Gesamtübersicht

Eine Gesamtübersicht des Systems findet sich in Abbildung 3.1 wieder. Diese Übersicht gibt den groben Aufbau des Systems wieder. Auf die Bestandteile und Komponenten wird in den folgenden Kapiteln näher eingegangen. In diesem Kapitel werden zunächst die grundlegenden Designentscheidungen erläutert. Das System gliedert sich in drei Abschnitte. Dem Frontend, welches für die Interaktion mit dem Nutzer zuständig ist und die benötigten Elemente des Backends ansteuert. Das Backend, bestehend aus vielen kleinen Komponenten, liefert die Logik, welche benötigt wird, um ein Diffusions- oder GAN-Modell zu trainieren. Des Weiteren beinhaltet es die Logik, mithilfe eines bestehenden Modells Bilder zu generieren. Die Datenbank speichert die Daten der Modelle. Für das Speichern der Modelle und der Trainingsdaten wird das Filessystem verwendet, da es sich hier um sehr große Dateien handelt.

In der folgenden Tabelle, werden die Technologien, Frameworks und Programmiersprachen der Anwendung hervorgehoben.

System			
Komponente	Technologie	Framework	Programmiersprache
Frontend	WebApp	Blazor	C# HTML CSS
Backend	Microservices	Flask	Python
Datenbank	NoSQL	MongoDB	Text

Tabelle 3.1: Eine Übersicht über die verwendeten Frameworks, Technologien und die Programmiersprachen

Das Frontend wird als Webapplikation realisiert. Es selbst beinhaltet keine Logik des maschinellen Lernens und kann deshalb in jedem Browser verwendet werden. Die Entscheidung fiel hier auf das Framework Blazor, da hier eine große Dokumentation mit

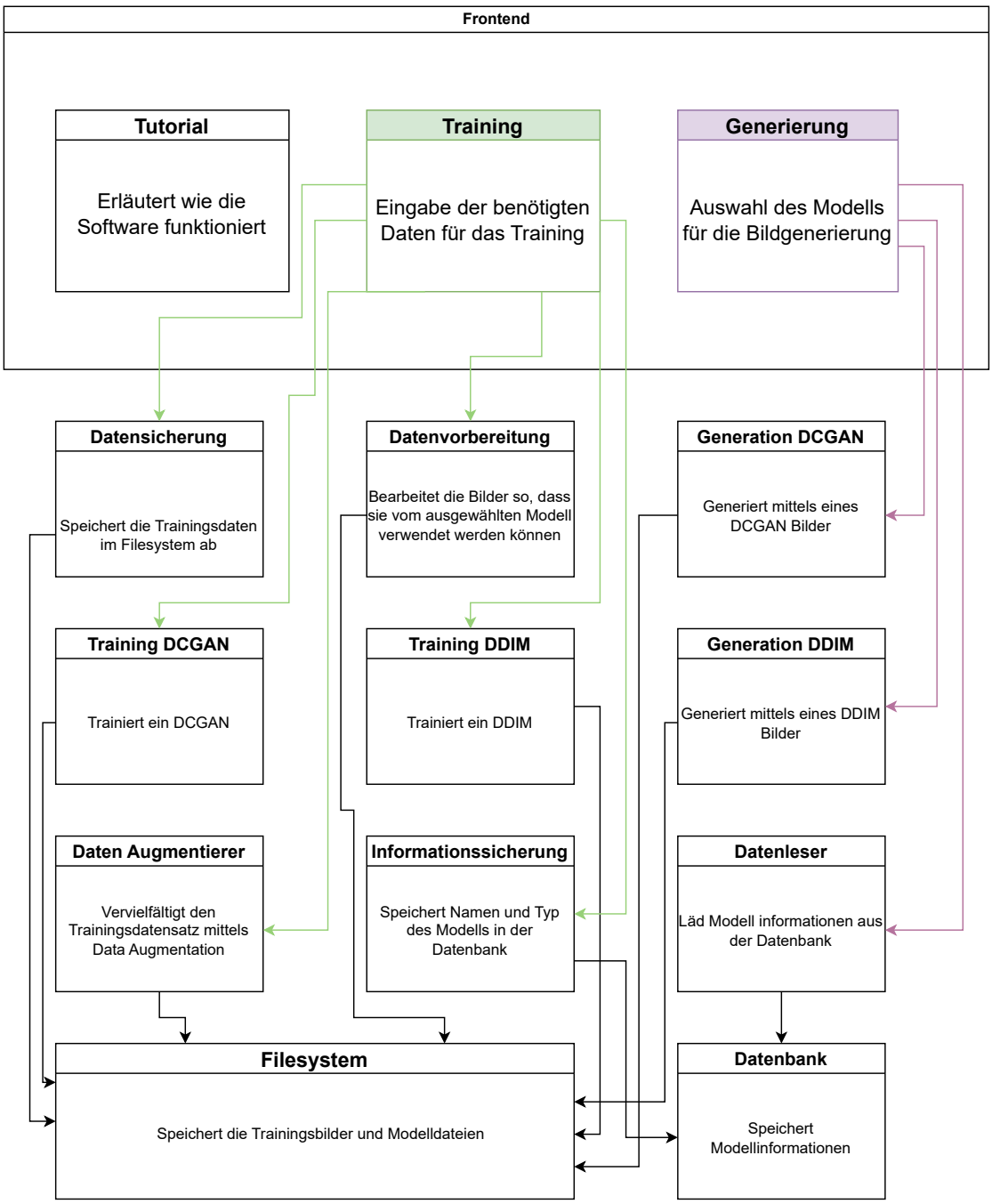

Abbildung 3.1: Die Gesamtübersicht des Systems

Anwendungsbeispielen bereits von Microsoft zur Verfügung gestellt wird. Der große Vorteil von Blazor ist hier die Verwendung von C#. Da das Framework Unity, mit welchem AR-Applikationen verwirklicht werden können, ebenfalls C# verwendet, könnten damit im Falle der Nutzung von Blazor Codebestandteile wiederverwendet werden.

Das Backend wiederum besteht aus vielen kleinen Microservice basierten Komponenten, welche jeweils eine möglichst kleine Aufgabe erfüllen. Hier wurde sich für eine Microservice basierte Architektur entschieden, da während der Analyse festgestellt wurde, dass sich Microservices dafür eignen, dass die Anwendung nicht speziell von einzelnen Bibliotheken abhängig ist. So können z.B. für das Training des GAN Modells und des Diffusionsmodells unterschiedliche Versionen derselben Bibliothek, wie z.B. TensorFlow, verwendet werden, ohne dass das jeweils andere Modell anschließend nicht mehr funktioniert. Hierbei wurde sich für das Prinzip der Shared Database entschieden, damit die Konsistenz der Daten gewährleistet wird. Die Trainingsdaten und fertig trainierte Modelle werden hierbei auf dem Filesystem gespeichert. Wie sich in der Analyse hervorgehoben hat, ist Flask als Framework sehr schnell und leichtgewichtig. Folglich eignet sich Flask am besten für die Entwicklung von vielen kleinen Services. Python ist wiederum die am meisten verwendete Programmiersprache für die Entwicklung von neuronalen Netzen und bietet auch die meisten existierenden Bibliotheken, welche die Entwicklung unterstützen.

Für die Datenbank wurde MongoDB verwendet. Der Vorteil gegenüber SQL ist wie in der Analyse beschrieben seine Geschwindigkeit. Des Weiteren wird JSON, wobei es sich ebenfalls um ein gespeichertes Textformat handelt, häufig für den Datenaustausch verwendet. Dadurch müssen die Daten nicht noch extra für SQL angepasst werden und können fast unverändert gespeichert werden.

3.2 Frontend

3.2.1 Startseite und Tutorial

Die Startseite, zu sehen in Abbildung 3.2, liefert dem Nutzer die Informationen, welche er benötigt, um nachzuvollziehen, wie die Anwendung verwendet wird. Bei dem Inhalt des Tutorials handelt es sich um statische Bilder, welche exemplarisch Ergebnisse der Anwendung aufzeigen und Beschreibungen zu den Parametern der Anwendung. Auf der linken Seite befindet sich die Navigation. Hier kann der Nutzer auswählen, ob er selbst

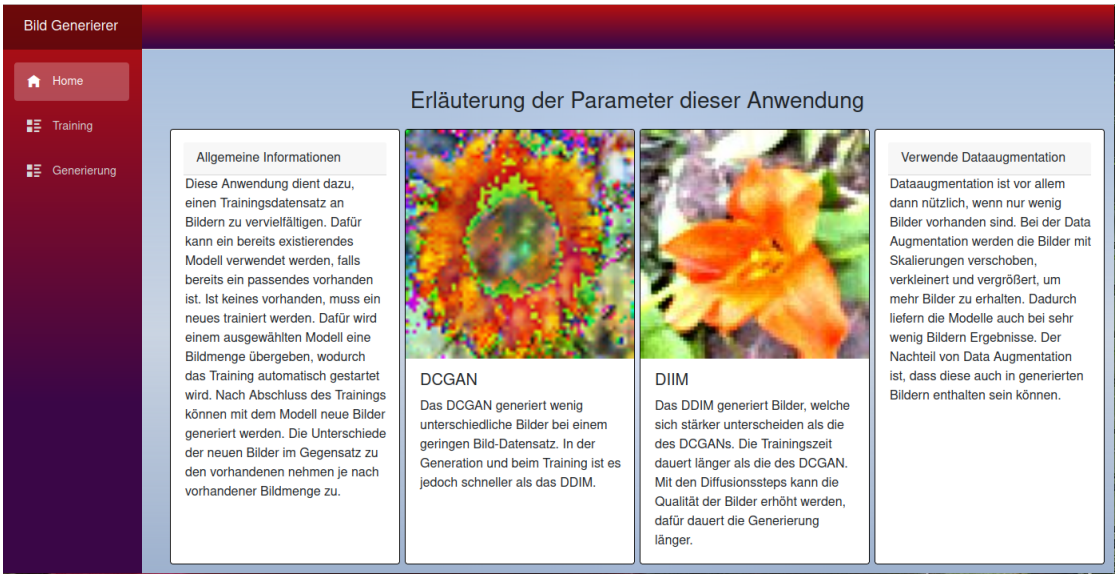

Abbildung 3.2: Die Startseite und das Tutorial der Anwendung

ein neues Netz generieren möchte, oder ob er Bilder von bereits bestehenden Modellen generieren will.

3.2.2 Training

Der Prozess des Trainings wird in dem folgendenden Aktivitätsdiagramm 3.3 verdeutlicht. So besteht dieser aus drei aufeinander folgenden Schritten.

Der erste Schritt des Trainings ist in Abbildung 3.4 aufgezeigt. Zu Beginn wird der Name des Modells vom Nutzer festgelegt. Des Weiteren wird im selben Schritt auch noch der Typ des Modells vom Nutzer festgelegt, also ob es sich um ein Diffusions oder GAN Modell handeln soll. Mit Betätigung des Submit Buttons wird Schritt zwei geladen.

Der zweite Schritt des Trainings findet sich in Abbildung 3.5 wieder. Dieser besteht aus einer Eingabezeile. Hier kann der Nutzer eine beliebige Anzahl an Bildern auf seinem lokalen Gerät auswählen. Die Auswahl ist dabei eingeschränkt auf den Bildtyp .png,jpg und jpeg. Andere Dateien können nicht ausgewählt werden. Damit wird ein Fehler im Backend vermieden, falls der Nutzer versehentlich eine Datei auswählen sollte, welche das Backend nicht verarbeiten kann. Der Nutzer kann zusätzlich noch festlegen, ob er Data Augmentation verwenden will. Sind alle Bilder ausgewählt, leitet das Frontend intern die

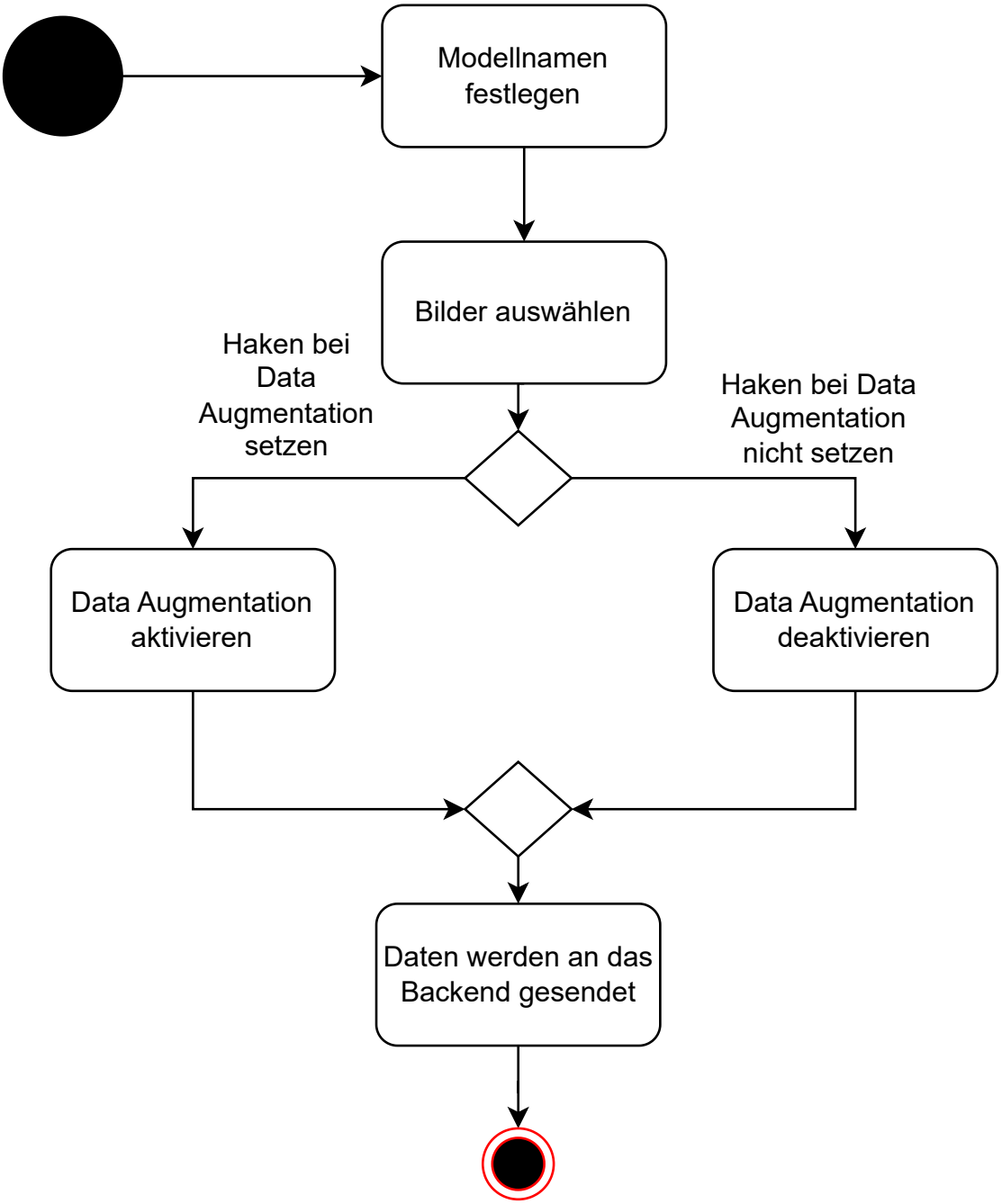

Abbildung 3.3: Aktivitätsdiagramm für die Trainingsseite

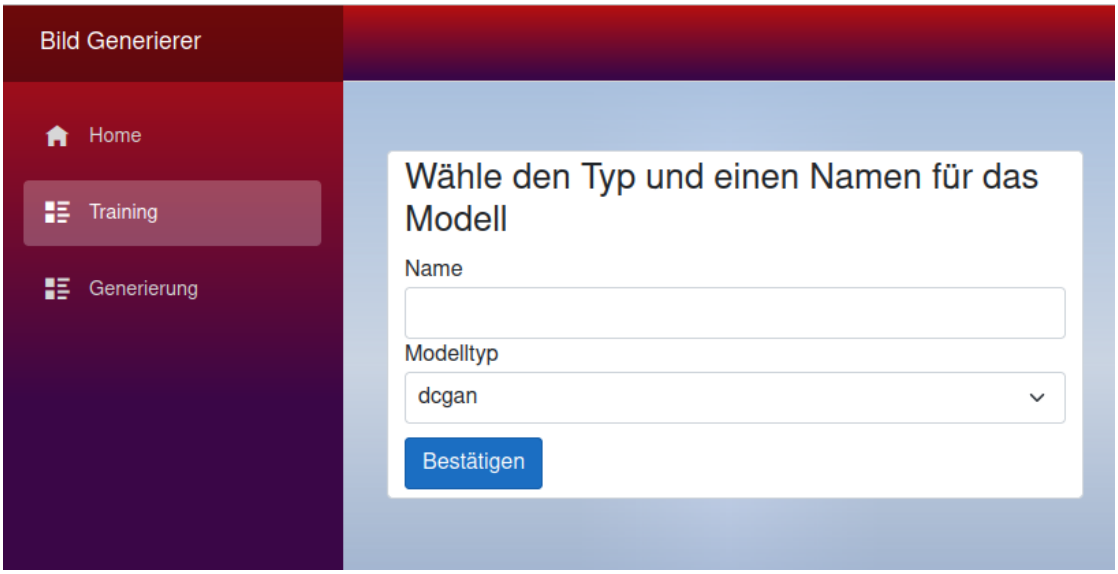

Abbildung 3.4: Der erste Schritt des Trainings

Schritte für das Training ein. Der Nutzer muss keine weiteren Schritte befolgen und nur noch warten, bis das Netz fertig trainiert wurde.

3.2.3 Generierung

Auch die Generierung erfolgt in drei Schritten, zu sehen in Abbildung 3.6. Im Gegensatz zum Trainingsprozess findet die Aktivitäten nicht mehr voneinander gekapselt statt. Es wird lediglich über eine Auswahlliste ein existierendes Modell ausgewählt. Danach wird noch die Anzahl der Bilder festgelegt, welche generiert werden sollen. Für die Wahl eines Diffusionsmodells wird zusätzlich die Möglichkeit die Diffusionsschritte festzulegen bereitgestellt. Nach einer Bestätigung werden die Bilder im Downloadordner verfügbar gemacht.

3.2.4 Orchestrierung des Trainingprozesses

In Abbildung 3.8 ist der Prozess des Trainings eines Modells, welches vom Frontend gesteuert wird, zu sehen. Bei den einzelnen Aktivitäten handelt es sich um die entsprechenden Services des Backends, welche von Orchestrierungsservice des Frontends aufgerufen werden. Zunächst werden die Trainingsbilder in die Größe formatiert, welche das jeweilige

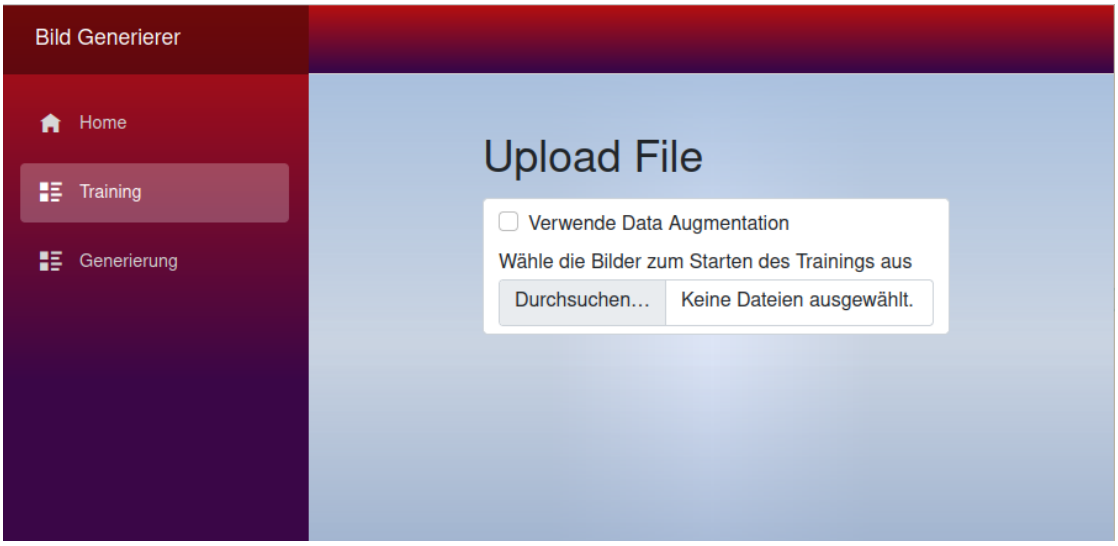

Abbildung 3.5: Der zweite Schritt des Trainings

Model benötigt. Wenn Data Augmentation ausgewählt wurde wird dieser Schritt daraufhin ausgeführt. Anschließend wird je nach Wahl des Modells das Training für das DDIM oder DCGAN initiiert. Nach dem Abschluss des Trainings wird der Name und der Typ des Modells abgespeichert. Dieser Prozess erfolgt erst nach dem Training, da das Modell erst im Frontend angezeigt werden soll, wenn es für die Bildgenerierung zur Verfügung steht. Abschließend werden die Trainingsdaten wieder gelöscht, um den Speicherplatz wieder freizugeben.

3.3 Backend

3.3.1 Allgemein

Das Backend ist verwirklicht in einem Microservice basiertem System. Es handelt sich hierbei jedoch nicht um vollständige Microservices, da die Microservices durch die Trainingsbilder und erstellten Modelle im gemeinsamen Filesystem untereinander abhängig sind. Diese Designentscheidung wurde getroffen, da die Bilddaten nicht mehrfach zwischen den verschiedenen Microservices und dem Frontend ausgetauscht werden sollen. Ansonsten würde der Trainingsprozess durch den ständigen Austausch der Bilder zwischen Frontend und Microservices erheblich mehr Zeit benötigen. Allerdings reicht die

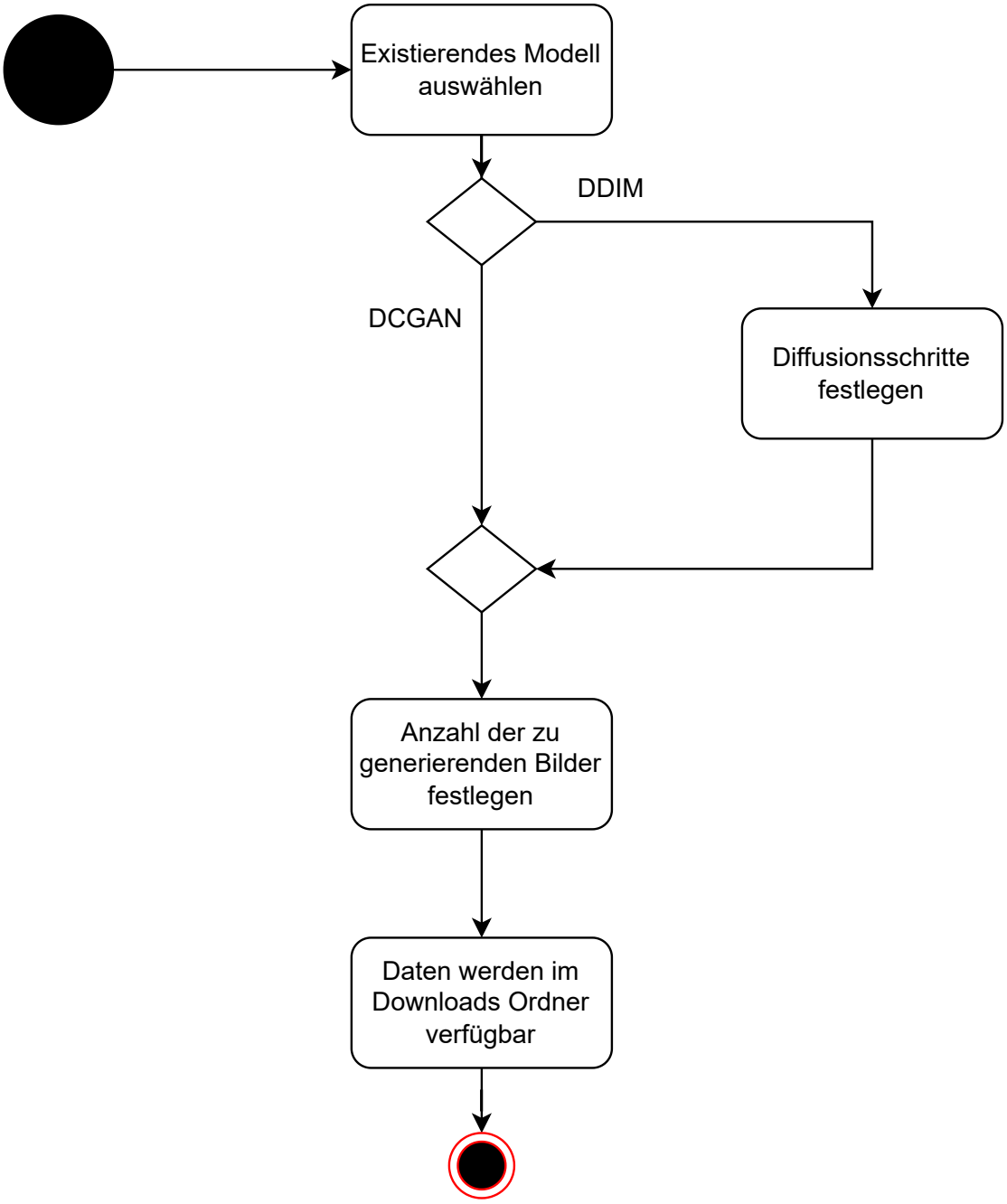

Abbildung 3.6: Aktivitätsdiagramm für die Generationsseite

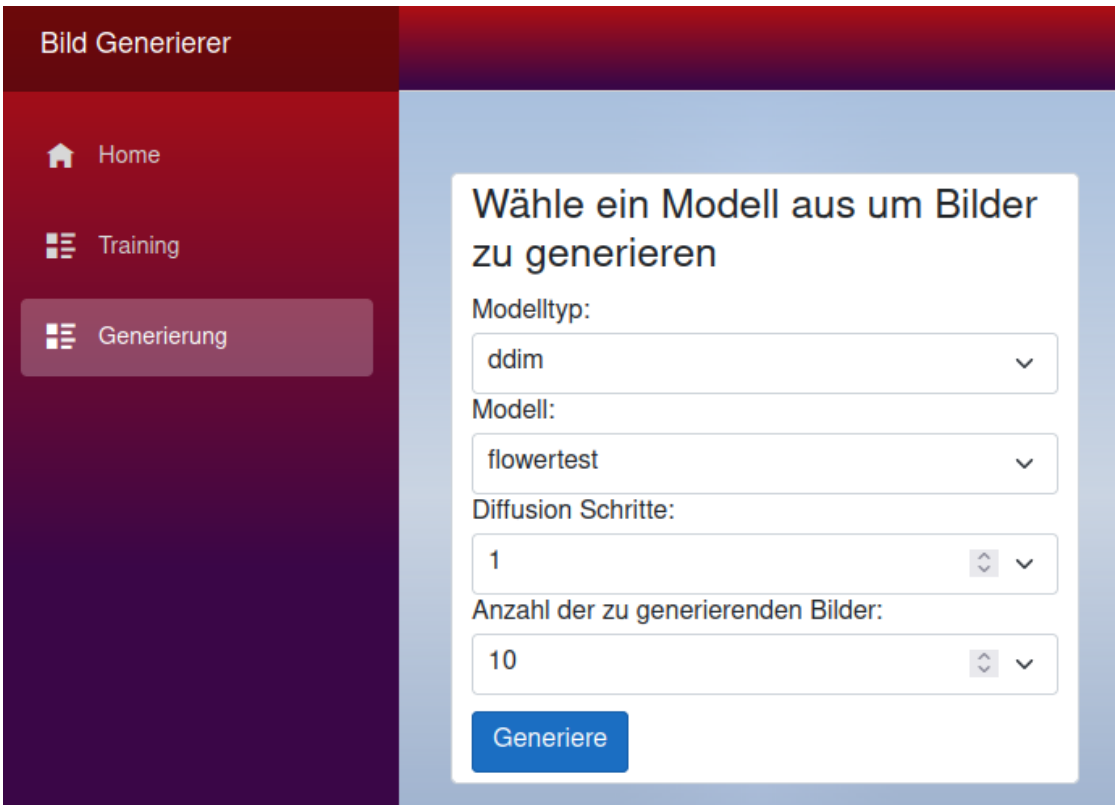

Abbildung 3.7: Die Seite für die Generierung

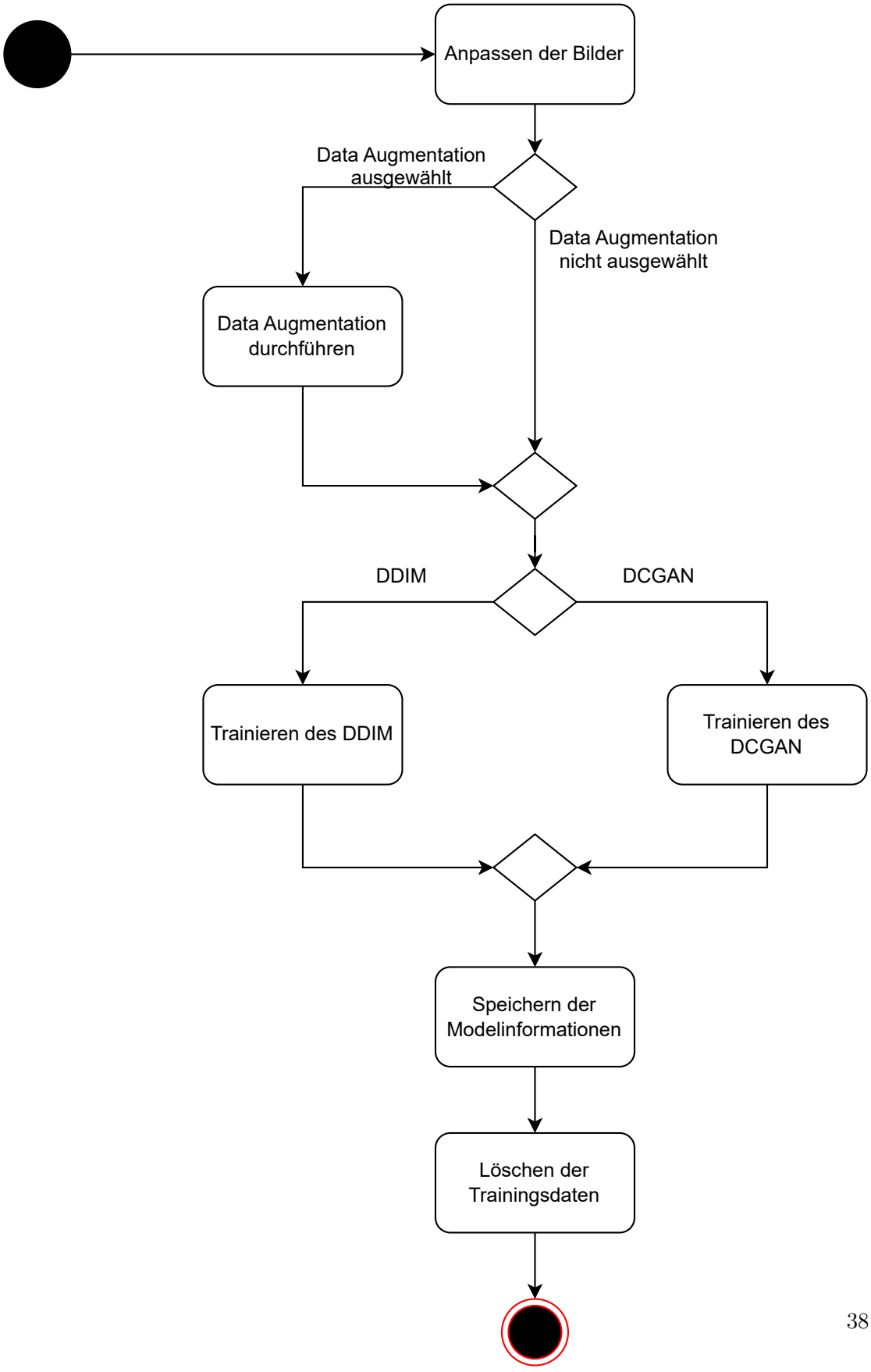

Abbildung 3.8: Aktivitätsdiagramm für den Prozess der Orchestrierung

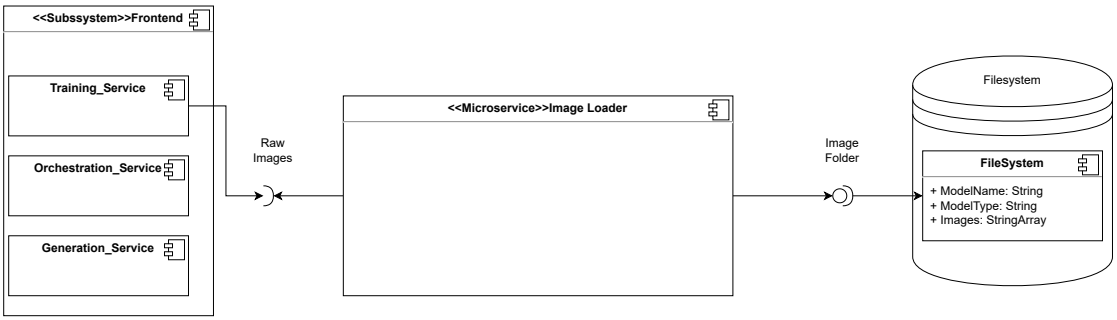

Abbildung 3.9: Komponentendiagramm für das Laden der Trainingsdaten in das Filesystem

Umsetzung, um Glue Code zu verhindern, da lediglich dieselben Ordner verwendet werden, jedoch unterschiedliche Bibliotheken genutzt werden können. In den folgenden Unterkapiteln werden die verschiedenen Services beschrieben.

3.3.2 Service für den Datenaustausch

Der erste Service, welcher während des Trainings verwendet wird, ist für das Laden der Trainingsbilder in das Filesystem zuständig. In Abbildung 3.9 wird der Service in einem Komponentendiagramm dargestellt. Der Service erhält die Trainingsdaten im Body des http Post Requests. Die Bilder sind hierbei im Base64 Format in einer JSON-Datei gespeichert. Um die Bilder im Filesystem abzuspeichern, werden diese wieder in ein Byte Array dekodiert und anschließend in einem festen Ordner abgespeichert.

3.3.3 Service für das Verändern der Bildgröße

Sind die Trainingsbilder in dem Filesystem der Anwendung, ist der nächste Schritt die Anpassung der Bilder an die für das jeweilige Modell benötigte Bildgröße. Der Microservice ist in Abbildung 3.10 dargestellt. Die Bilder werden in einer Schleife aus dem Ordner der unverarbeiteten Bilddaten entnommen und in einem neuen Ordner abgespeichert. Die Informationen der Bildgröße erhält der Microservice aus dem Body des Http-POST-Requests.

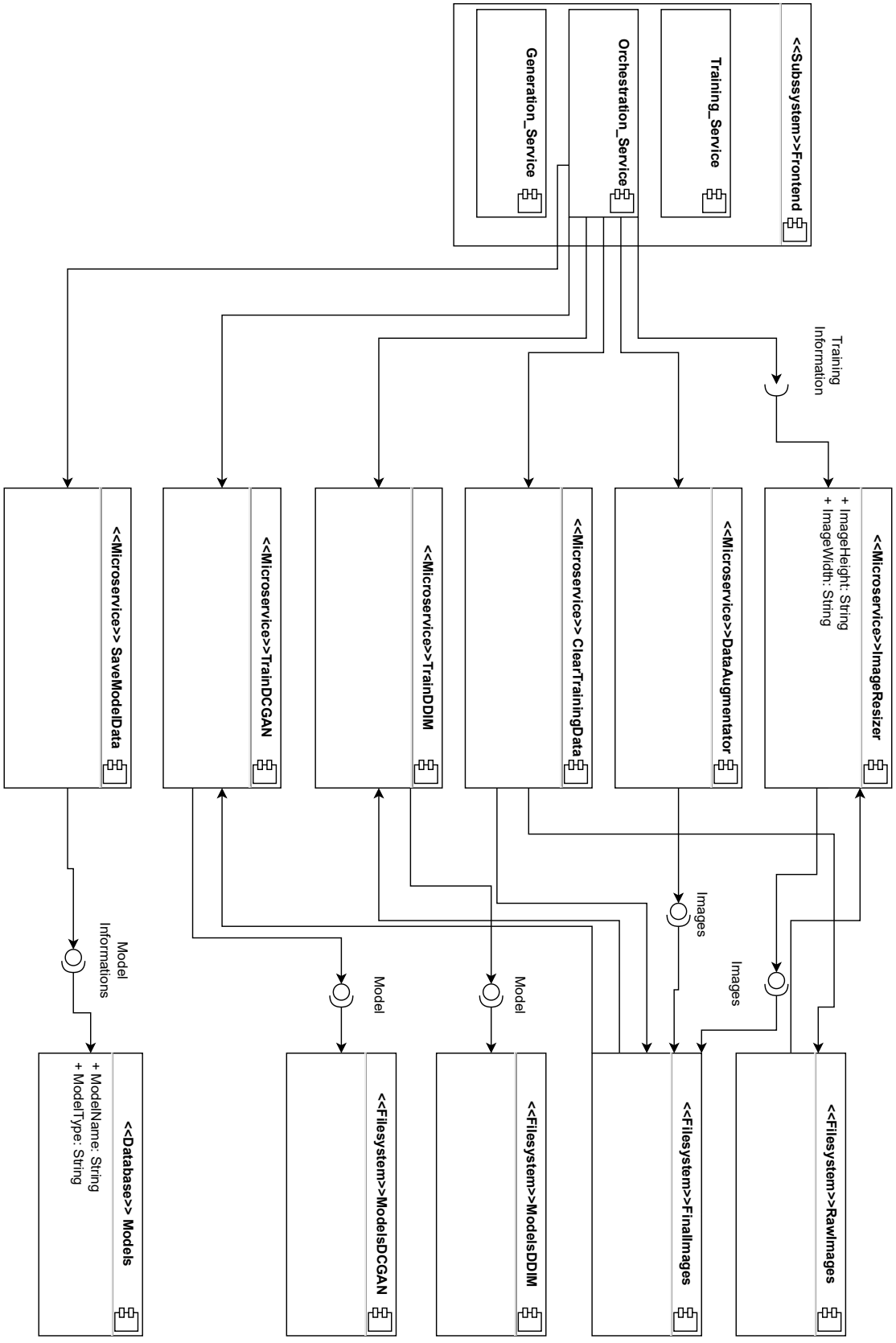

Abbildung 3.10: Die vom Orchestration Service angesteuerten Microservices für das Training der Modelle

3.3.4 Service für die Data Augmentation

Für den Service der Data Augmentation wurden stellenweise Methoden aus [21] übernommen. Weggelassen wurden Methoden, welche das Bild in hohem Maße verändern. Dies soll verhindern, dass die Generationsmodelle Bilder mit schlechter Qualität durch ein Augmentation-Leak hervorrufen. Je nach Größe der vorhandenen Trainingsdaten kann der Service für die Data Augmentation auch ganz weggelassen werden, um eine höhere Abbildung der Domain zu gewährleisten. Für den Fall, dass jedoch nur wenige Bilder vorhanden sind, ist dieser Service unerlässlich, um Ergebnisse hervorzurufen, da z.B. der Diskriminator des GANs zu schnell ein Overfitting hervorruft oder die Batch Size verändert werden muss. Um die Methode der Data Augmentation möglichst für jedes trainierbare Netz anwendbar zu machen, wird auf spezielle Methoden zur Minderung eines Augmentation-Leaks verzichtet. Dabei handelt es sich unter anderem um eine komplette Rotation des Bildes. Da die vorliegende Anwendung als primäres Ziel hat, Bilder für das Training von Objekterkennern zu generieren, ist in diesem Fall ein Augmentation-Leak jedoch nicht schlimm, solange dieses nicht zu stark ausfällt, wie z.B. wenn dem Bild zusätzliches Rauschen hinzugefügt wird.

3.3.5 Service für das Training des DCGAN

Der Service für das Training eines DCGAN ist von dem Pytorch Tutorial von Nathan Inkawich[33] übernommen und nur minimal verändert worden. Hierbei wurde ein Diskriminator implementiert, zu sehen in Abbildung 3.11. Der Diskriminator besteht aus mehreren aufeinander folgenden Blöcken von Conv Layern mit einer folgenden BatchNormalization und einer LeakyRelU Aktivierungsfunktion. Anhand dieser Layer-Architektur wird ein übergebenes Bild analysiert und anschließend durch die Sigmoid-Aktivierungsfunktion eine Wahrscheinlichkeit ausgegeben, ob es sich bei dem übergebenen Bild um ein reales Bild handelt.

Der Generator, zu sehen in Abbildung 3.12, wiederum versucht anhand des übergebenen Vektors ein Bild der Größe channelx64x64 zu generieren. Dies wird mit aufeinander folgenden Blöcken von Conv2d-Transpose-Layern mit anschließender Batch-Normalization und einer RelU-Aktivierungsfunktion ermöglicht.[33]

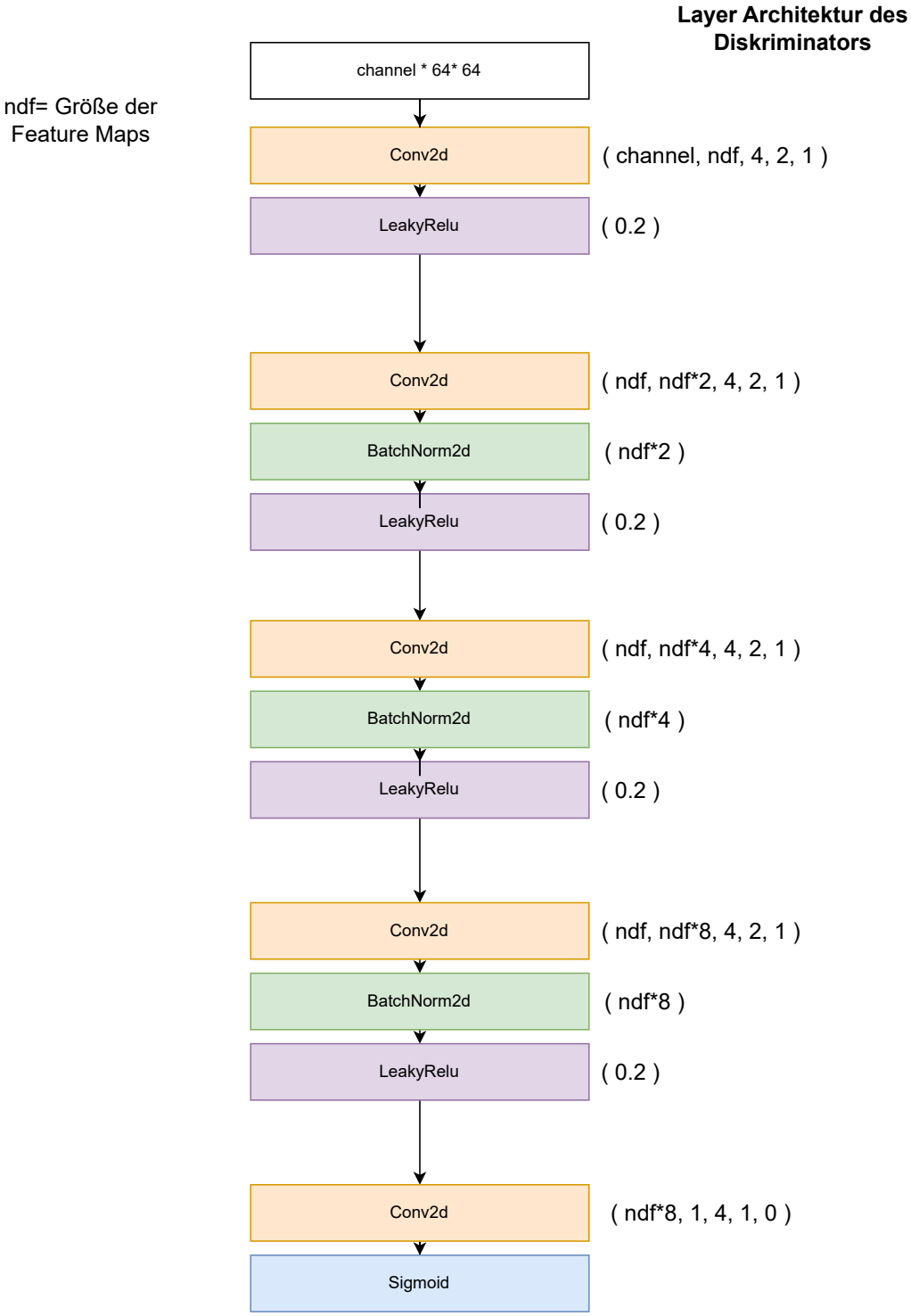

Abbildung 3.11: Diskriminator in Anlehnung an die Implementierung von [33]

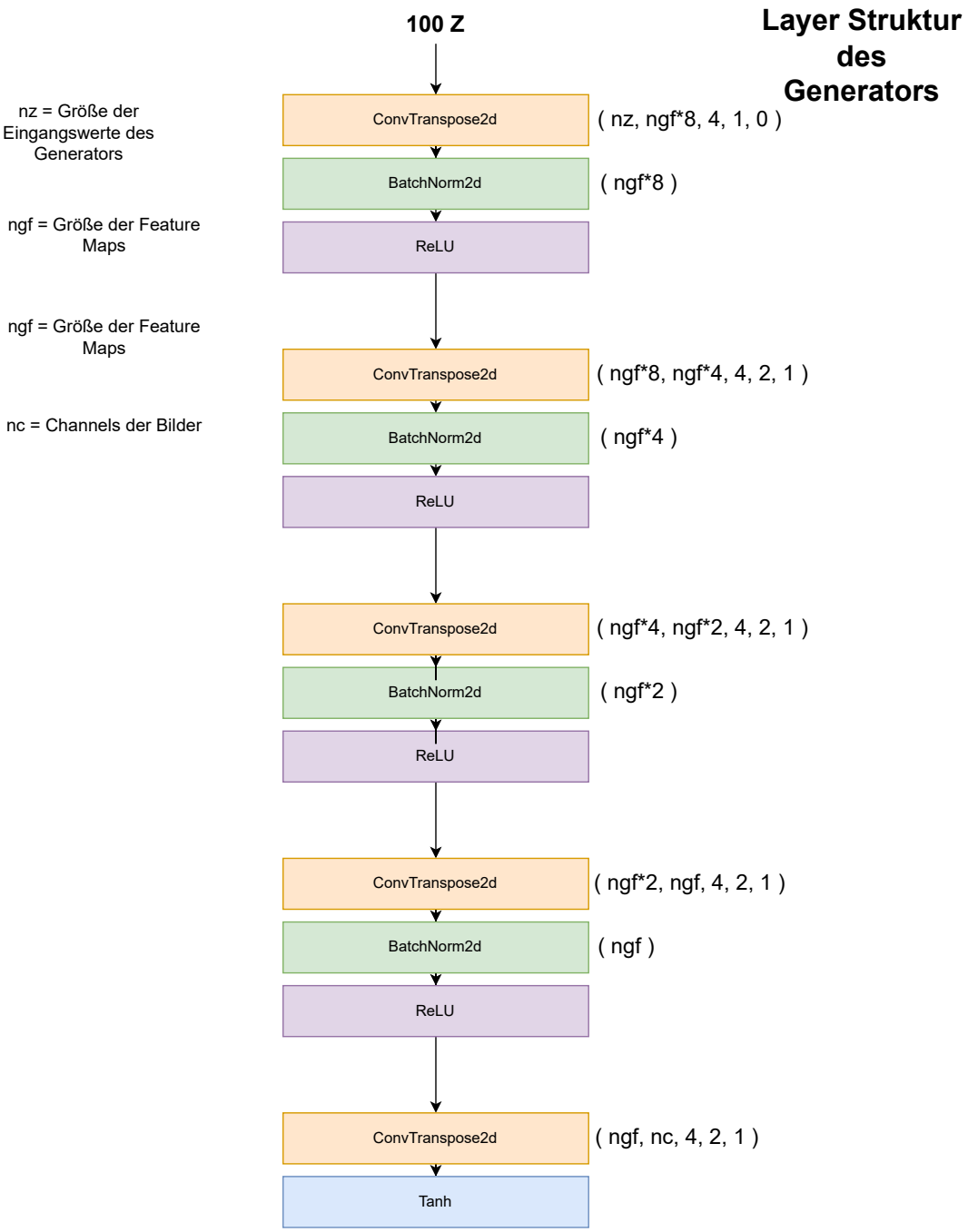

Abbildung 3.12: Generator-Architektur in Anlehnung an die Implementierung von [33]

Während dem Training des Diskriminators wird versucht, die Wahrscheinlichkeit zu maximieren, ein syntetisches Bild zu erkennen während beim Generator versucht wird, die Fälschungen zu verbessern, damit der Diskriminator sie nicht mehr erkennt.[33]

Das Netz selbst ist in einen Flask Server verpackt. Wird dieser aufgerufen, so werden die Bilder aus dem finalen Bildordner in die Anwendung geladen und anhand derer das Training gestartet. Nach dem Durchlauf aller Epochen wird das trainierte Netz mit dem Modellnamen in den Ordner der DCGAN Modelle abgespeichert.

3.3.6 Service für das Training des DDIM

Als DDIM Modell wurde die Implementierung des Modells von Asif [6] verwendet. Er verwendet dafür große Bestandteile der Implementierung von [55]. Asif macht in seiner Übersicht über nachimplementierte Paper deutlich, dass sich die Nachimplementierungen an Stellen vom original Paper unterscheiden können. Der Vorteil der Implementierung von Asif ist seine implementierte Methode für das Generieren von Bildern, da diese ein 64x64 vom DDIM Modell generiertes Bild auf 128x128 Pixel vergrößert. Dies ermöglicht eine bessere Untersuchung der Ergebnisse, da auf einem größeren Bild auch mögliche Generierungsfehler leichter zu erkennen sind.

Das DDIM Modell von Asif[6] besteht aus einem U-Net und Methoden für das Ver- und Entrauschen von Bildern. Ein weiterer Vorteil seiner Implementierung ist das Speichern von Checkpoints nach jeder Epoche und bereits existierender Methoden für die Wiederaufnahme des Trainings nach einem festgelegten Checkpoint. Um das Training mit seinem Modell zu starten, wird ein Ort für die Ablage der Checkpoint Modelle und der Beispielbilder benötigt. Des Weiteren wird der Pfad für die Trainingsbilder übergeben. Hier kommt der von den vorangegangenen Services erstellte Ordner mit den finalen Trainingsdaten ins Spiel. Im Gegensatz zu seinen Versuchen mit dem DDPM, welches 500 Diffusionsschritte benötigt, wird die Anzahl von Diffusionsschritten im Code auf lediglich 40 per Default gesetzt. In seiner Readme schreibt er zusätzlich, dass das DDIM weniger benötigt, aber nicht genau wie viele er für seine Experimente verwendet hat.

3.3.7 Service für das Speichern der Modellinformationen

Der Service für das Abspeichern von Modellinformationen soll in dem Fall des vorliegenden Frontend erst aufgerufen werden, wenn das ausgewählte Modell fertig trainiert

wurde. Dies liegt daran, dass das Modell im Frontend erst angezeigt werden soll, wenn sich damit auch Bilder generieren lassen und es auch existiert. Bei den Modellinformationen handelt es sich um den Namen des Modells und den Typ. Die Informationen werden benötigt, um bei der späteren Generierung von Bildern das richtige Modell auswählen zu können.

3.3.8 Service für die Speicherplatzfreigabe

Nach dem erfolgreichen Training eines Modells gilt es den Speicherplatz, den die Trainingsbilder in Anspruch genommen haben, wieder freizugeben. Auch wenn dies in den vorangegangenen Services im Anschluss geschehen könnte, wurde hier ein zusätzlicher Service erstellt, um die Funktionen der anderen Microservices weitestgehend zu kapseln. Hierfür wird jedes Trainingsbild in den von den vorangegangenen Services verwendeten Ordnern mittels Schleifen gelöscht.

3.3.9 Service für die Generierung von Bildern mit einem DCGAN

Bei der Generierung von Bildern mit einem DCGAN wird zunächst das Modell geladen, welches in dem Http-Body des Requests angegeben ist. Anschließend wird pro Bild, welches generiert werden soll, ein Vektor mit zufälligem Rauschen erzeugt und an das Modell übergeben. Die Ausgabe des Modells wird daraufhin in einen Byte-Array umgewandelt und anschließend als base64 String abgespeichert. Alle Bilder werden dann in einer JSON-Datei zusammengefasst und als Response zurückgesendet.

3.3.10 Service für die Generierung von Bildern mit einem DDIM

Der Prozess der Generierung von Bildern erfolgt für das DDIM anders als für das DCGAN, da hier die Generierung unterschiedlich funktioniert. Dies liegt daran, dass Asif[6] hier eine eigene Methode für die Generierung implementiert hat, bei welcher die Ergebnisse als Bild in einem Ordner abgelegt werden. Deshalb werden in diesem Service die Bilder von dem Ordner in eine JSON-Datei verpackt und anschließend der Ordner für die Bilder wieder geleert. Die JSON-Datei wird im Abschluss als Antwort auf den Request zurückgesendet.

3.3.11 Datenbank

Für die Aufbewahrung der Modellinformationen wird eine MongoDB Datenbank verwendet. Dort werden die Modellinformationen wie der Name des Modells und der Typ des Modells gespeichert. Bei den Modellinformationen handelt es sich folglich um die Informationen, die von dem Generierungsservice benötigt werden, um anhand des Namens und Typ des Modells das richtige Modell auszuwählen. Des Weiteren handelt es sich hier auch um die Informationen, welche im Frontend später dem Benutzer angezeigt werden sollen.

3.3.12 Deployment

In diesem Kapitel wird nun exemplarisch erläutert, wie man einzelne Microservices unabhängig voneinander in Betrieb nehmen kann. Zunächst wird hierfür ein Dockerfile und eine requirements.txt Datei angelegt. Die requirements.txt Datei beinhaltet die Namen der Python Bibliotheken, welche ein einzelner Microservice benötigt. Dies ist eine gängige Methode, um Bibliotheken zu installieren und festzuhalten, welche eine Python Anwendung benötigt. Für jeden Microservice werden die Bibliotheken in einer separaten requirements.txt Datei festgehalten. Anschließend wird ein dockerfile erstellt, welches zunächst ein passendes vordefiniertes Dockerimage verwendet. Da im Falle dieser Anwendung die Modelle in PyTorch geschrieben sind, kann dafür das offizielle PyTorch-Dockerimage[51] verwendet werden. Im Dockerfile werden anschließend noch die Dateien und Ordner festgelegt, welche in den Dockercontainer verschoben werden sollen. Danach werden anhand der requirements.txt Datei die Abhängigkeiten aufgelöst und abschließend wird der Befehl für das Starten der Anwendung festgehalten. Um die Daten wie die trainierten Modelle und verarbeiteten Trainingsdaten auch für andere Services festzuhalten, ist es essentiell, die verwendeten Ordner mit lokalen Ordnern zu verbinden. Dies kann beim Starten eines Dockercontainers mit dem -v Operator realisiert werden. Jeder Service, der nun denselben Ordner verwendet muss, kann anschließend den gemounteten Ordner selbst mit einem Ordner im Dockercontainer verbinden. Sollen die Microservices auf verschiedenen VM's oder Servern deployt werden, empfiehlt es sich, die Dateien auf einem geteilten Volume zu speichern. Dieser Vorgang wurde während des Trainings der Modelle mit der Infrastruktur des CSTI[13] anhand des Services für das Training des DCGAN und des DDIM getestet.

4 Experimente

4.1 Allgemeines

In diesem Kapitel soll die Fähigkeit der Modelle für die Generierung von fotorealistischen Daten überprüft werden. Dabei steht vor allem im Vordergrund, zu überprüfen, ob sich auch mit wenigen vorhandenen Trainingsdaten fotorealistische Bilder generieren lassen. Zunächst erfolgt die Überprüfung visuell. Der Grund hierfür ist, dass sich bei Bildern auch mit dem menschlichen Auge erkennen lässt, ob das gewünschte Objekt überhaupt ansatzweise im Bild vorhanden ist. Falls sich mit dem Modell Bilder generieren lassen, welche signifikante Ähnlichkeiten zu realen Bildern aufweisen, wird im nächsten Schritt ein Modell anhand der synthetischen Daten generiert, welches Objekte auf realen Bildern klassifizieren soll. Damit soll eine Eignung der Bilder für das Training eines Objektdetektors oder Objektklassifizerers evaluiert werden können.

4.2 Szenario 1 - Training eines Generators mit über 700 Bildern

4.2.1 Beschreibung

Dieses Szenario verwendet den Datensatz von Alexander Mamaev[44]. In diesem Szenario will der Nutzer einen Objekterkenner trainieren, welcher zwischen verschiedenen Blumen unterscheiden kann. Dabei hat er einen existierenden Datensatz gefunden und will die Menge der Trainingsdaten einander angleichen. Die Anzahl der Trainingsbilder von [44] ist wie folgt aufgeteilt:

Hierbei handelt es sich um einen relativ ausgeglichenen Datensatz, da Sonnenblumen, Gänseblümchen und Tulpen gleichmäßig vorhanden sind. Tulpen und Löwenzahn sind

Datensatz Szenario 1	
Blume	Anzahl Bilder
Sonnenblumen	733
Gänseblümchen	764
Rosen	784
Löwenzahn	1052
Tulpe	984

Tabelle 4.1: Der Datensatz von Mamaev [44]

hier stärker als die anderen repräsentiert. In diesem Szenario wird versucht, den Datensatz der Sonnenblumen zu vergrößern, um diesen ungefähr an die Datensätze des Löwenzahns und der Tulpen anzugleichen. Um die Qualität und den Nutzen der generierten Bilder zu testen, wird ein Objekterkenner trainiert, welcher zwischen Rosen, Tulpen und Löwenzahn trainiert. Dabei wird sich am im Analysekapitel beschriebenen GAN-Quality-Index orientiert. Es wird aber zusätzlich auch die Auswirkung einer Datensatzerweiterung anhand der synthetischen Daten ermitteln. Damit das Training des Objekterkenners besser auf die Realität verallgemeinert werden kann, wird für die Testbilder der Sonnenblumen ein anderer Datensatz von [41] verwendet. Für den Test des Löwenzahns und der Tulpen werden von dem Datensatz von [44] jeweils 100 Bilder entnommen und in einen Testordner verschoben. Die verschobenen Bilder werden nicht für das Training verwendet.

Der Code für den Objekterkenner wurde in großen Teilen von einem PyTorch Tutorial [52] übernommen. Da der Objekterkenner jedoch mit 32x32 Bildern arbeitet, wurde er um eine Layerschicht erweitert, um auch mit 64x64 Bildern zu funktionieren. Dies resultiert daraus, dass die Generatoren 64x64 Bilder generieren und diese unverändert genutzt werden sollen.

Für das Training des Diffusionsmodells und des GANs wurde der volle Sonnenblumendatensatz von [44] verwendet. Die Ergebnisse werden in den folgenden Kapiteln präsentiert.

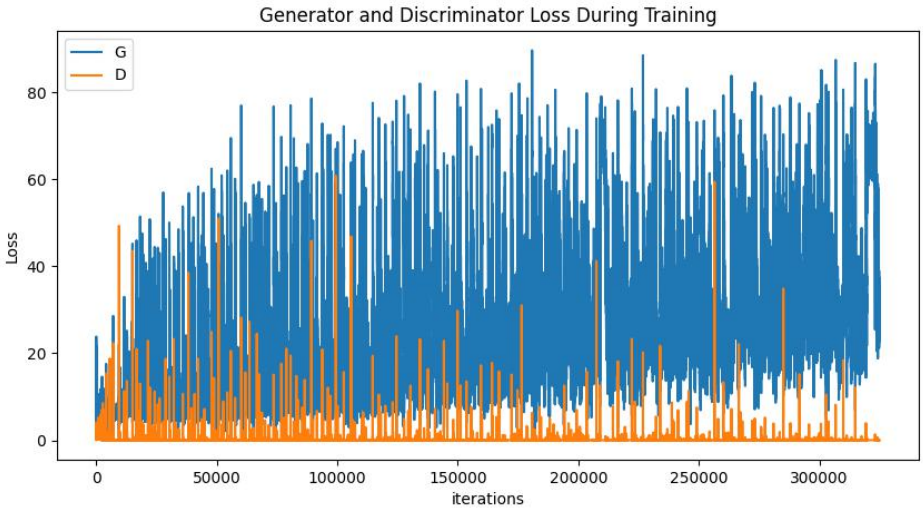

Abbildung 4.1: Der Trainingsverlauf des GANs mit Sonnenblumen

4.2.2 Training des GANs

Verlauf des Trainings

Der Verlauf des Trainings des GANs ist in Abbildung 4.1 zu sehen. Dem Verlauf nach zu urteilen funktioniert das Training des GANs, da sowohl der Diskriminator als auch der Generator kontinuierlich einen hohen Loss-Wert produzieren. Folglich erkennt der Diskriminator häufig die Fälschungen des Generators, jedoch schafft es auch der Generator Bilder zu generieren, welche der Diskriminator als Fälschung erkennt. Ein Blick auf die Ergebnisse des Trainings in Abbildung 4.2 zeigt jedoch, dass es sich mit hoher Wahrscheinlichkeit immer um ähnliche Bilder handelt. So wiederholen sich die einzelnen Bilder durchgängig. Es handelt sich hier um einen teilweise vorhandenen Mode-Collapse. Es werden zwar ein paar wenige Bilder derselben Domain generiert, jedoch lange nicht alle. Die Menge an neuen Bildern würde nicht reichen, einen Objekterkenner merklich zu verbessern. Bei einer Integration des trainierten GAN-Modells mit anschließender Generierung kamen die Bilder in Abbildung 4.3 heraus. Aufgrund der mangelnden Varianz an Sonnenblumenbildern werden die Ergebnisse des GANs nicht weiter mit einem Objekterkenner trainiert.

Abbildung 4.2: Ergebnisse des Trainings des GANs mit Sonnenblumen. Die realen Sonnenblumen stammen aus dem Trainingsdatensatz von [44]

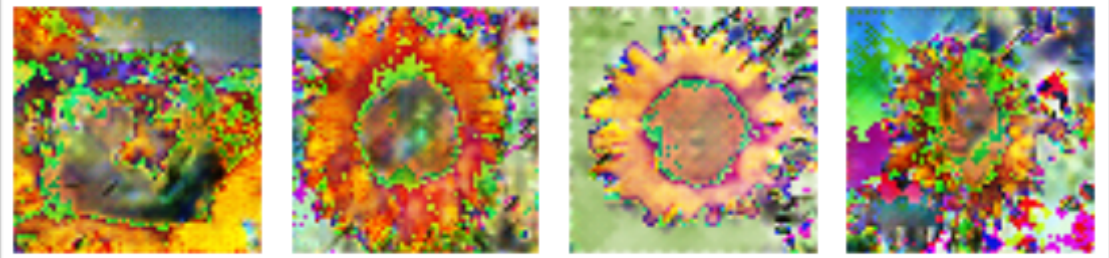

Abbildung 4.3: Die vier häufigsten Ergebnisse den Sonnenblumen GANs

4.2.3 Training des DDIM

Das Training des DDIM verläuft konstant. Man erkennt mit steigendem Verlauf der Epochen eine Verbesserung der Ergebnisse. Jedoch sind ab ca. der 400. Epoche beim Sonnenblumen-Datensatz keine Veränderungen anhand der zwischengespeicherten Bildern mehr erkennbar. Das fertige Modell wurde in der Anwendung integriert. Anschließend wurden mit dem Modell über die Anwendung Bilder generiert. Ein Beispiel für gute Bilder, welche mit dem DDIM generiert werden, ist in Abbildung 4.4 zu sehen. Es ist zu erkennen, dass hier teilweise sehr realitätsnahe Bilder generiert werden, jedoch können auch Bilder, auf welchen das Objekt kaum bis gar nicht mehr mit dem menschlichen Auge zu erkennen sind, wie in Abbildung 4.5, generiert werden. Die qualitativ hochwertigsten Bilder weisen auch Unterschiedlichkeiten untereinander auf. Sogar gezeichnete Sonnenblumen werden generiert, auch wenn diese nur teilweise im Trainingsdatensatz vorhanden waren. Aufgrund ausreichender Varianz der Bilder kann hierfür ein ähnlich großer Datensatz wie der reale Datensatz generiert werden, um damit die Qualität der Bilder bei der Verwendung als Trainingsdaten zu messen.

Training eines Objekterkenners

Für das Training des Objekterkenners wird zunächst mithilfe der generierten Daten ein synthetischer Sonnenblumen Trainingsdatensatz erstellt. Bilder, auf denen ausschließlich Rauschen zu erkennen ist, werden hierfür nicht berücksichtigt. Abbildung 4.4 ist ein Beispiel für die verwendeten Bilder. Hier wurden auch Bilder verwendet auf denen das Objekt nur schwach zu erkennen ist. Für den gemischten Datensatz wurden 100 synthetische

Abbildung 4.4: Beispiele generierter Bilder des DDIM, auf welchem Sonnenblumen zu erkennen sind

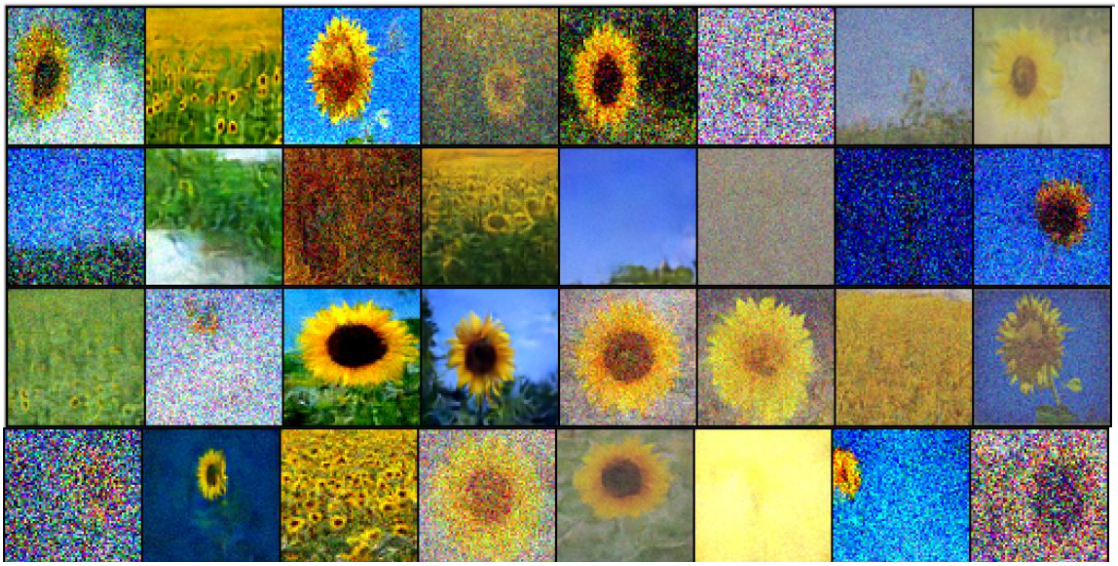

Abbildung 4.5: 32 aufeinander folgend generierte Bilder mit dem Sonnenblumen-DDIM

Bilder ausgewählt, auf denen eine Sonnenblume mit dem menschlichen Auge deutlich zu erkennen ist. Der Sonnenblumen Trainingsatensatz für den gemischten Datensatz beträgt anschließend 833 Bilder und ist damit dem Trainingsdatensatz der Tulpen mit 884 und des Löwenzahns mit 952 Bildern angenähert. Die trainierten Netze wurden jeweils einmal auf einem Testdatensatz mit jeweils 100 Bildern jeder Klasse und einem Testdatensatz mit fast nur realen Sonnenblumen angewandt. Lediglich ein Bild der anderen Klassen ist in dem Sonnenblumen Testdatensatz enthalten, um die gleiche Testpipeline verwenden zu können. Die Ergebnisse sind in Abbildung 4.2 und 4.3 zu sehen.

Objekterkenner mit gemischten Testdaten	
Model Trainings-daten	Genauigkeit
Real	87%
Synthetisch	84%
Gemischt	89%

Tabelle 4.2: Die Genauigkeit der Modelle nach dem Training mit realen, synthetischen und gemischten Daten jeweils 100 Bildern von Sonnenblumen, Tulpen und Löwenzahn

So ist die Genauigkeit des Modells, welches mit synthetischen Bildern von Sonnenblumen trainiert wurde, in beiden Fällen geringer als die der realen Daten. Wird jedoch der gemischte Datensatz für das Training des Modells verwendet, so kann das Modell

Objekterkenner Sonnenblumen als Testdaten	
Model Trainings-daten	Genauigkeit
Real	92%
Synthetisch	80%
Gemischt	95%

Tabelle 4.3: Test der Modelle auf einen Testdatensatz, welcher mit 1000 Bildern fast ausschließlich aus Sonnenblumen besteht

Sonnenblumen besser erkennen als das mit realen Bildern von Sonnenblumen trainierte Modell. Auch die Unterscheidung zwischen den Objekten ist beim gemischten Modell besser als die beim realen.

4.3 Szenario 2 - Training eines Generators mit 150 Bildern

4.3.1 Beschreibung

In diesem Szenario möchte der Nutzer eine neue Blume, von welcher noch keine Daten vorhanden sind, selbst hinzufügen. Hier entscheidet er sich für Tulpen, da diese bald anfangen zu wachsen und auch schon vereinzelt anzutreffen sind. In diesem Fall ist Beschaffung der Trainingsdaten nur begrenzt möglich. Zwar lassen sich Tulpensträuße schon vereinzelt im Supermarkt kaufen, jedoch ist dies keine natürliche Umgebung. Der Nutzer stellt mit den wenigen Tulpen, welche bereits draußen gewachsen sind und einem gekauften Strauß einen Trainingsdatensatz mit 150 Bildern zusammen. Um seinen Trainingsdatensatz zu erweitern, verwendet er die in dieser Arbeit entwickelte Anwendung.

Für dieses Szenario wird der gleiche Datensatz von [44] verwendet. Hiervon werden die Rosen und die Gänseblümchen genutzt. Gänseblümchen unterscheiden sich hier optisch stärker von den Tulpen als die Rosen. Das Szenario unterscheidet sich von dem vorherigen vor allem in der Anzahl der verwendeten Bilder. Während im vorangegangenen Szenario 733 Bilder für das Training der Generatornetze zur Verfügung standen, sind es diesmal nur 150. Des Weiteren ist der Unterschied in der Datenmenge zwischen Gänseblümchen, Rosen und den eigens erstellten Tulpen deutlich größer als im ersten Szenario. Vom Gänseblümchendatensatz sind 100 Bilder für den Testdatensatz entnommen worden. Der Trainingsdatensatz für Rosen bleibt unverändert, da hier 100 Rosenbilder von

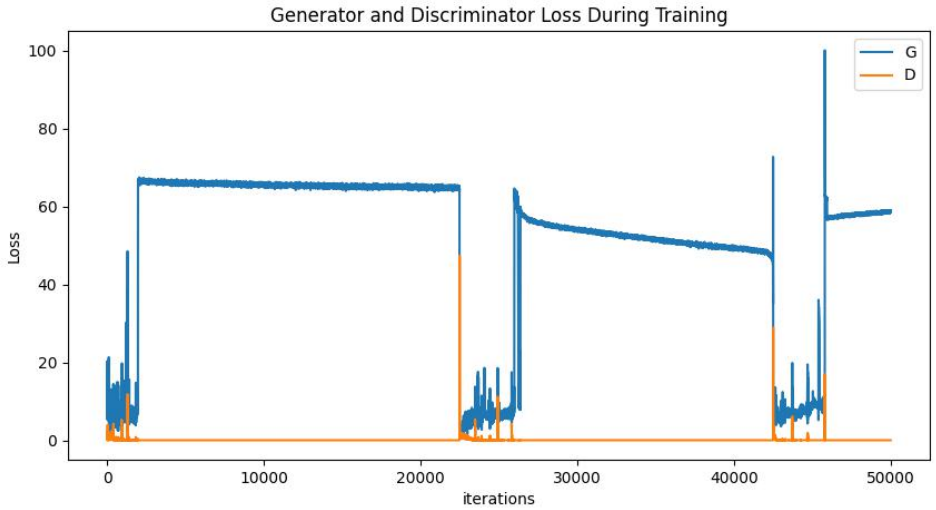

Abbildung 4.6: Der Trainingsverlauf des GANs mit Tulpen

[41] verwendet werden. Für den Testdatensatz mit 1000 Tulpen wurden Tulpenilder von [41] verwendet.

4.3.2 Training des DCGANs

Das Training des DCGANs verläuft sehr schnell in ein Overfitting des Diskriminators, wie in Abbildung 4.6 zu sehen ist. Der Generator schafft es zwar nach einer langen Zeit wieder ein Bild zu generieren, welches der Diskriminator nicht erkennt, fällt jedoch nach kurzer Zeit wieder in einen Zustand des Overfittings zurück. Dies wird auch durch die vom GAN generierten Bilder in Abbildung 4.7 bestätigt. Hier wird lediglich ein Bild generiert, auf welchem keine Tulpe zu sehen ist. Es handelt sich folglich um einen kompletten Mode Collapse. Da bei einem kompletten Mode Collapse keine Bilder generiert werden können, welche eine Tulpe abbilden, werden die Ergebnisse des GANs nicht für das Training des Objekterkenners verwendet.

4.3.3 Training eines DDIMs

Auch das Training eines DDIMs liefert weniger gute Ergebnisse als noch in Szenario 1. So ist unter den Ergebnissen von 32 Bildern in Abbildung 4.8 maximal ein verwendbares Bild zu finden. Dies ist zwar immer noch eine größere Teilmenge als unter den Ergebnissen des

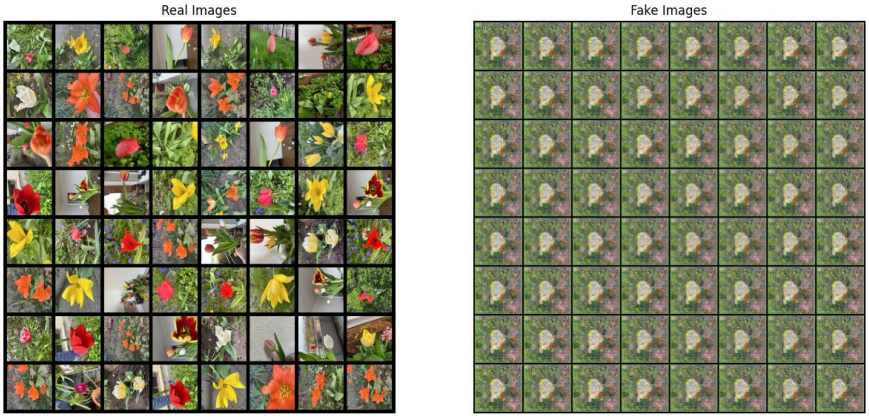

Abbildung 4.7: Die Ergebnisse des Trainings des GANs mit Tulpen

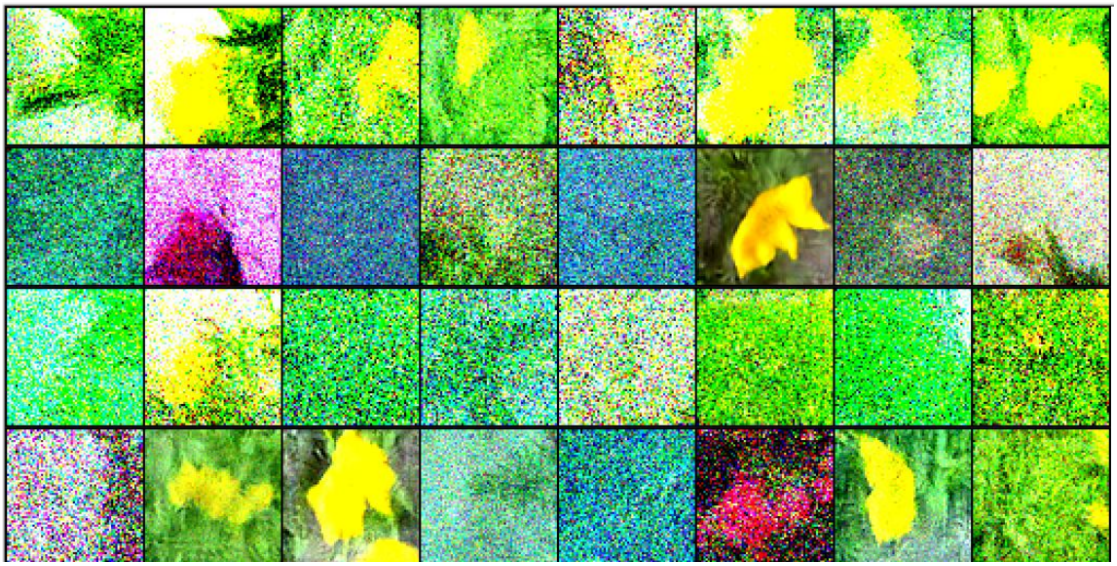

Abbildung 4.8: 32 aufeinander folgend generierte Bilder mit dem Tulpen-DDIM ohne Data Augmentation

GANs, jedoch ist es unter diesen Umständen nur mit großem Aufwand möglich, einen Trainingsdatensatz mit ausreichender Bildmenge zusammenzustellen.

4.4 Szenario 3 - Training eines Generators mit Data Augmentation

Auch hier werden dieselben 150 Bilder als Start wie in Szenario 2 verwendet. Die Bilder werden jedoch mit Methoden der Data Augmentation des Data Augmentation Services vervielfältigt, sodass der Trainingsdatensatz, für die Generationsmodelle, aus 2235 Bildern besteht. Die restlichen Bestandteile des zweiten Szenarios bleiben gleich.

4.4.1 Training des DCGANs

Der Trainingsverlauf des DCGANs, zu sehen in Abbildung 4.9, im Falle des mit Data Augmentation erweiterten Datensatzes, ist wieder in der gewünschten Form. So sind die Loss-Werte wie gewollt wiederholt am Steigen und Fallen, weshalb sich hier das minimax Spiel des GANs erkennen lässt. Der Generator schafft es folglich, wiederholt Bilder zu

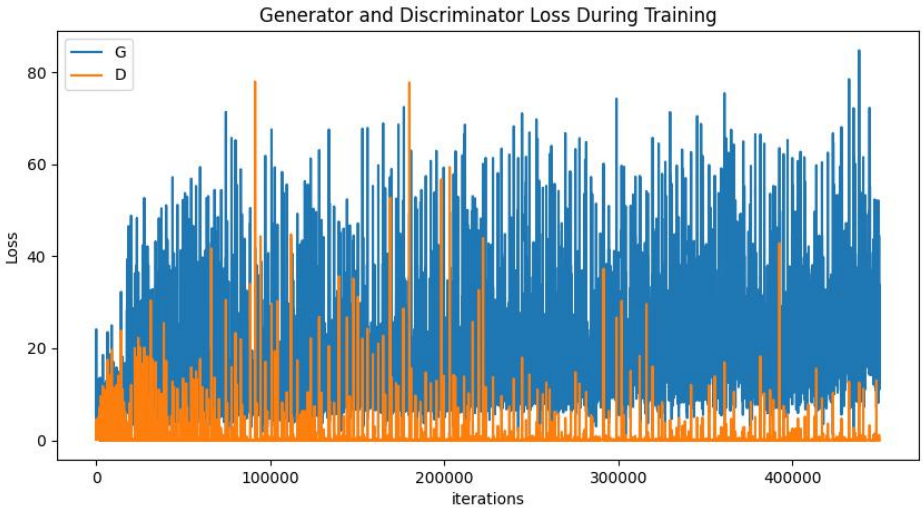

Abbildung 4.9: Der Trainingsverlauf des GANs mit Tulpen und Data Augmentation

generieren, welche der Diskriminator nicht erkennt. Jedoch zeigt ein Blick auf die Ergebnisse in Abbildung 4.10 der Generierung, dass auch in diesem Szenario die Ergebnisse auf wenige sehr gleich aussehende Bilder beschränkt sind. Dadurch reicht die Varianz nicht aus, einen eigenen erfolgversprechenden synthetischen Datensatz aufzubauen.

4.4.2 Training des DDIMs

Das Training des DDIM mit dem durch Data Augmentation erweiterten Datensatzes liefert kontinuierlich verwertbare Bilder, wie in Abbildung 4.11 zu sehen ist. Jedoch ist in Abbildung 4.12 deutlich, dass sich die generierten Bilder nur bei näherer Betrachtung von ihrem Original unterscheiden. Doch auch bei großer Ähnlichkeit handelt es sich hier immernoch um andere Bilder, da jedes Bild mal stärker und mal schwächere Unterschiede zu ihrem Original aufweisen. So ist das generierte Bild in der unteren Zeile dunkler als das Original, aber heller als das augmentierte Bild. Aufgrund ausreichender Qualität der Bilder einer genügenden Anzahl verwertbarer Bilder in 32 Bildern werden im nächsten Schritt die generierten Bilder verwendet, um damit einen Objekterkenner zu trainieren.

Abbildung 4.10: Die Ergebnisse des Trainings des GANs mit Tulpen und Data Augmentation

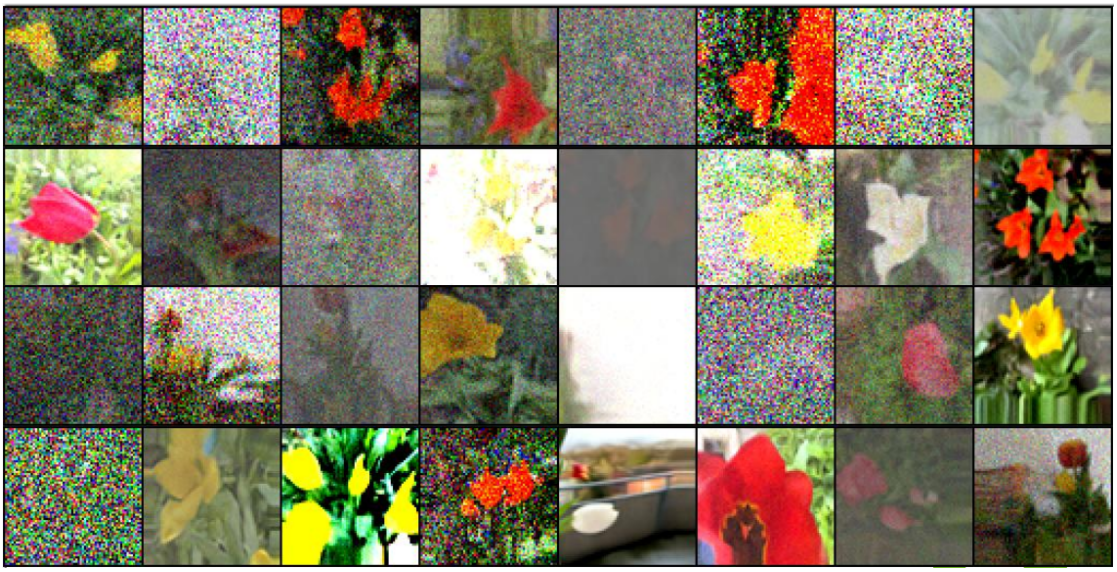

Abbildung 4.11: 32 aufeinander folgend generierte Bilder mit dem Tulpen-DDIM mit Methoden der Data Augmentation

4.4.3 Training eines Objekterkenners mit synthetischen Daten

Für das Training des Objekterkenners wurde einmal ein Objekterkenner mit den realen 150 Tulpendaten trainiert. Ein weiterer wurde mit 150 synthetischen Bildern des DDIM trainiert, dabei wurden von 1000 generierten Bilder die besten 150 ausgewählt. Ein dritter und letzter Objekterkenner wurde mit beiden Datensätzen in Kombination trainiert. Wie in Tabelle 4.4 zu erkennen ist, bringt ein Objekterkenner sowohl mit synthetischen als auch mit realen Daten ein Ergebnis von 57%. Die Erkennung funktioniert folglich gleich. Bei der Erkennung von beinahe ausschließlich Tulpen werden sogar 1% mehr Tulpen als solche erkannt, unter Verwendung des mit synthetischen Daten trainierten Modells. In beiden Fällen führte eine Kombination des realen und synthetischen Datensatzes zu einer deutlichen Steigerung der Erkennung. Bei der Erkennung von 1000 Tulpen, zu sehen in Tabelle 4.5, haben sich die Prozente sogar mehr als verdoppelt. Es bleibt jedoch anzumerken, dass auch hier der Datensatz der Tulpen im Vergleich zu den Gänseblümchen und Rosen mit weniger als die Hälfte an Bildern immer noch unterrepräsentiert ist.

Abbildung 4.12: Eine Gegenüberstellung von generierten Bildern(links), augmentierten Bildern(mitte) und Originalbildern(rechts)

Objekterkenner mit gemischten Testdaten	
Model Trainings-daten	Genauigkeit
Real	57%
Synthetisch	57%
Gemischt	65%

Tabelle 4.4: Die Genauigkeit der Objekterkenner nach dem Training mit realen, synthetischen und gemischten Daten jeweils 100 Bildern von Tulpen, Gänseblümchen und Rosen

Objekterkenner Tulpen als Testdaten	
Model Trainings-daten	Genauigkeit
Real	8%
Synthetisch	9%
Gemischt	22%

Tabelle 4.5: Test der Objekterkenner auf einen Testdatensatz, welcher mit 1000 Bildern fast ausschließlich aus Tulpen besteht

4.4.4 Verwendete Hardware

Für das Training der Modelle wurde die Rechnerinfrastruktur des FTZ-SMSY der HAW-Hamburg verwendet. Dort befinden sich leistungsstarke Rechner mit Grafikkarten wie unter anderem der A100.[13]

5 Evaluation

5.1 Auswertung der Experimente

5.1.1 Die Performance des DCGANs

Bei dem Training des DCGANs ist deutlich geworden, dass dieses bei der in den Experimenten verwendeten Trainingsdatenmenge schnell in die bekannten Probleme des Mode Collapse oder des Konvergenzverlusts hineinläuft. Die generierten Bilder sind dadurch wenig voneinander unterscheidbar und die Varianz reicht nicht aus, um damit einen eigenen synthetischen Datensatz zu erstellen. Maximal minimale Erweiterungen eines existierenden Datensatzes sind damit möglich. Eine Ursache ist, dass der Diskriminator zu schnell den Status des Overfitting fällt, dies ist anhand des Trainingsverlaufs erkennbar. Data Augmentation stellte sich als nützliche Funktion heraus, um das Overfitting des Diskriminators zu verlangsamen, um somit trotz eines kleinen Datensatzes Ergebnisse zu erzielen.

5.1.2 Die Performance des DDIM

Das DDIM schaffte es in jedem Experiment mindestens ein Bild des übergebenen Objektes zu generieren. Jedoch hat auch das DDIM bei 150 Bildern Probleme, so sind die meisten Bilder ausschließlich verrauscht. Data Augmentation wiederum führte dazu, dass sich die Anzahl an verwertbaren Bilder deutlich gesteigert hat. Das DDIM ist jedoch besonders anfällig für ein Augmentation-Leak, sodass diese oft 1 zu 1 auch in den generierten Bildern auftreten. Da sich die Bilder jedoch vor allem für das Training weiterer Netze eignen sollen, ist dies keine negative Eigenschaft. Die Generierung von Bildern des DDIMs dauerte, wie bereits in auch in den Experimenten länger als beim GAN, dies deckt sich mit den Erkenntnissen der Analyse. Durch die häufig auftretenden

stark und schwach verrauschten Bilder kann nicht jedes Bild des DDIM für weiteres Training verwendet werden, sondern nur ausgewählte. Des Weiteren war auffällig, dass sich die Bilder im Fall von 150 Originalbildern nur schwach unterschieden haben. Die Unterschiede reichten jedoch aus, um die Performance des Objekterkenners zu verbessern. Das DDIM erfüllt bei einer kleinen Trainingsdatenmenge folglich eine ähnliche Funktion wie die Data Augmentation. Trotzdem ist auch hier die Generierung mit DDIMs nicht mit Data Augmentation gleich zu setzen, da die Änderungen nicht gesteuert werden und die Änderungen mit normaler Data Augmentation nur mit erheblichen Aufwand zu erreichen sind.

5.1.3 Die Performance der Anwendung

Auch das Training mit der Anwendung wurde während der Experimente beiläufig getestet. Hierbei ist ein Bottleneck bei sehr großen Originalbildern aufgetreten. So kommt es gelegentlich zu einer Verzögerung im Browser, wenn die Bilder in das Base64 Format konvertiert werden, um es an den Server zu schicken. Dieser Schritt nimmt auch viel Zeit in Anspruch. Bei drei Bildern mit einer Pixelgröße von 3000x2400 Pixeln werden mehr als 10 Minuten für die Konvertierung benötigt. Die Bildgenerierung klappt jedoch ohne Verzögerung. Auch bei einer großen Anzahl von über 1000 generierten Bildern werden diese schnell und ohne sichtbare Verzögerung dem Nutzer zur Verfügung gestellt.

5.2 Einfluss auf die Anwendung

Da sich Data Augmentation in beiden Fällen als nützliche Möglichkeit erwiesen hat, um entweder die Dauer des Trainingserfolges zu verlängern oder die Anzahl an verwertbaren Bildern zu erhöhen, wird empfohlen nicht auf diese zu verzichten. Auch wenn sich das GAN in den Experimenten nicht dafür geeignet hat, den Trainingsdatensatz stärker zu erweitern, so kann sich dies bei einem großen Datensatz ändern. Die Verwendung eines Diffusionsmodells ist jedoch bei einer kleinen Menge an Trainingsdaten weitaus effizienter.

6 Fazit und zukünftige Arbeiten

Ziel dieser Arbeit war die Entwicklung einer Anwendung, welche Implementierungsdetails von generativen Netzen für den Nutzer abstrahiert. Dadurch sollen Entwickler auch ohne Kenntnisse über die verwendeten Methoden der Bildgenerierung Bilder als Trainingsdaten generieren können. Damit diese einen positiven Effekt auf das Training haben, müssen diese so fotorealistisch wie möglich sein. Mit GANs wurde das Ziel nur in einem sehr begrenzten Rahmen erreicht. Zwar konnten auch durch GANs Bilder mit Realitätsnähe generiert werden, jedoch nicht in ausreichender Menge. Diffusionsmodelle konnten die Bilder mit weit größerer Varianz generieren und die besten Ergebnisse der Diffusionsmodelle waren zusätzlich auch fotorealistischer als die der GANs. Unter Verwendung von Data Augmentation und einem Diffusionsmodell konnte sogar mit nur 150 Bildern ein Objekterkenner trainiert werden, welcher durch die synthetischen Daten die Erkennung der Testdaten auf eine Art und Weise verbessern konnte, dass damit die Genauigkeit der Erkennung des Objekts verdoppelt wurde. Die generierten Bilder unterscheiden sich jedoch im Fall von wenigen Trainingsdaten nicht stark von ihrem Original. Im Fall dieser Anwendung und unter Verwendung eines kleinen Trainingsdatensatzes funktionieren die generative Modelle wie eine weitere Form der Data Augmentation, bei welcher das Originalbild in leicht veränderter Form dupliziert wird. Die Funktion, das Training zu verbessern, ist jedoch in dem Versuchsaufbau der Arbeit in allen Fällen bei einem gemischten Trainingsdatensatz erfolgreich gewesen. Auch die Abstraktion der Methoden des maschinellen Lernens wurde erfolgreich umgesetzt. Lediglich die Bilder müssen der Anwendung übergeben werden, auch wenn es hier besser ist, die Bilder zumindest etwas zu verkleinern, falls sie sehr groß sind, um den Prozess der Datenübertragung zu verkürzen.

Ein weiteres Ziel war die leichte Erweiterung der Anwendung. Durch die implementierte Microservice Architektur ist dies ohne große Änderungen der Anwendung möglich für den Fall, dass unüberwachte Modelle verwendet werden. Lediglich die Schnittstellen im Frontend, primär im Orchestrierungsservice bei der Auswahl des generativen Modells

und der Bildgröße für die Bildskalierung, müssen dafür angepasst werden. Sekundär muss auch die Liste bei der Generierung, um das neue Modell, erweitert werden. Alle weiteren Methoden können so auch für das neue Modell verwendet werden. Dadurch können auch die neuen Modelle die Services für die Data Augmentation sowie der Bildskalierung verwenden. Des Weiteren ist die Anwendung auch nicht abhängig von verschiedenen Python Versionen, da diese nur in speziell einem Microservice verwendet werden.

In zukünftigen Arbeiten soll die Anwendung weiter ausgebaut werden. Dabei können vor allem noch weitere Methoden der Data Augmentation verwendet werden, damit die Bildgenerierer auch bei einer sehr geringen Anzahl an Trainingsdaten Bilder mit starken Unterschieden zu den Originalbildern enthalten. Fortführende Arbeiten können die Anwendung mit weiteren Generierungsmodellen erweitern. Eine Möglichkeit stellt z.B. die Möglichkeit des Transferlernens dar, mit welcher ein GAN oder DDIM bereits erlernt hat, Bilder einer anderen oder ähnlichen Domäne zu erlernen. Damit könnte die Varianz der generierten Bilder erhöht werden. Der Anteil nicht verwendbarer Bilder in den generierten Bildern ist sehr hoch. Hier könnte sich die Integration eines Objekterkenners eignen, welcher erkennen soll, ob das Objekt zumindest halbwegs auf den generierten Bildern zu erkennen ist. Im Gegensatz zu dem GAN Diskriminator würde er so nicht das Training beeinflussen, sodass die Varianz der generierten Daten weiterhin sehr hoch bleibt, jedoch die Ergebnisse filtern. Des Weiteren können noch zusätzliche Methoden der Data Augmentation verwendet werden, um den Trainingsdatensatz noch weiter zu vergrößern, damit noch mehr Variationen der Originalbilder vom generativen Netz erstellt werden können. Eine Möglichkeit, die Anwendung zu erweitern, ist die Integration einer textbasierten Generierung, womit der Nutzer seine eigenen Bilder mit Textbefehlen erweitern kann. Hierfür müssten jedoch größere Änderungen vorgenommen werden, damit das Training auch weiterhin von Nutzern ohne Kenntnisse des maschinellen Lernens durchgeführt werden können.

Literaturverzeichnis

[1] Aggarwal, Sanchit: Modern web-development using reactjs. In: *International Journal of Recent Research Aspects* 5 (2018), Nr. 1, S. 133–137

[2] Apple: *CreateML*. – URL https://developer.apple.com/documentation/createml. – Zugriffsdatum: 2023-02-08

[3] ARD: *Experten fordern Pause bei KI-Entwicklung*. – URL https://www.tagesschau.de/wissen/musk-tech-pause-ki-entwicklung-101.html. – Zugriffsdatum: 2023-04-17

[4] Ardizzone, Lynton ; Kruse, Jakob ; Lüth, Carsten ; Bracher, Niels ; Rother, Carsten ; Köthe, Ullrich: Conditional invertible neural networks for diverse image-to-image translation. In: *DAGM German Conference on Pattern Recognition* Springer (Veranst.), 2020, S. 373–387

[5] Ardizzone, Lynton ; Lüth, Carsten ; Kruse, Jakob ; Rother, Carsten ; Köthe, Ullrich: Guided image generation with conditional invertible neural networks. In: *arXiv preprint arXiv:1907.02392* (2019)

[6] Asif: *Pytorch Diffusion*. – URL https://github.com/quickgrid/pytorch-diffusion. – Zugriffsdatum: 2023-03-21

[7] Bao, Fan ; Li, Chongxuan ; Sun, Jiacheng ; Zhu, Jun: Why Are Conditional Generative Models Better Than Unconditional Ones? In: *arXiv preprint arXiv:2212.00362* (2022)

[8] Bargshady, Ghazal ; Zhou, Xujuan ; Barua, Prabal D. ; Gururajan, Raj ; Li, Yuefeng ; Acharya, U R.: Application of CycleGAN and transfer learning techniques for automated detection of COVID-19 using X-ray images. In: *Pattern Recognition Letters* 153 (2022), S. 67–74

[9] Bhatt, Dulari ; Patel, Chirag ; Talsania, Hardik ; Patel, Jigar ; Vaghela, Rasmika ; Pandya, Sharnil ; Modi, Kirit ; Ghayvat, Hemant: CNN variants for computer vision: History, architecture, application, challenges and future scope. In: *Electronics* 10 (2021), Nr. 20, S. 2470

[10] Borji, Ali: Pros and Cons of GAN Evaluation Measures. In: *CoRR* abs/1802.03446 (2018). – URL http://arxiv.org/abs/1802.03446

[11] Chen, Haiyang: Challenges and corresponding solutions of generative adversarial networks (GANs): a survey study. In: *Journal of Physics: Conference Series* Bd. 1827 IOP Publishing (Veranst.), 2021, S. 012066

[12] Croitoru, Florinel-Alin ; Hondru, Vlad ; Ionescu, Radu T. ; Shah, Mubarak: Diffusion models in vision: A survey. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023)

[13] CSTI: *Creative Space For Technical Innovations.* – URL https://rz.smsy.haw-hamburg.de/. – Zugriffsdatum: 2023-03-21

[14] Deacon, John: Model-view-controller (mvc) architecture. In: *Online][Citado em: 10 de março de 2006.] http://www. jdl. co. uk/briefings/MVC. pdf* 28 (2009)

[15] DeepAI: *DeepAI.* – URL https://deepai.org/. – Zugriffsdatum: 2023-02-08

[16] Denker, Alexander ; Schmidt, Maximilian ; Leuschner, Johannes ; Maass, Peter: Conditional Invertible Neural Networks for Medical Imaging. In: *Journal of Imaging* 7 (2021), Nr. 11, S. 243

[17] Dhariwal, Prafulla ; Nichol, Alexander: Diffusion models beat gans on image synthesis. In: *Advances in Neural Information Processing Systems* 34 (2021), S. 8780–8794

[18] Dike, Happiness U. ; Zhou, Yimin ; Deveerasetty, Kranthi K. ; Wu, Qingtian: Unsupervised learning based on artificial neural network: A review. In: *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)* IEEE (Veranst.), 2018, S. 322–327

[19] Dipina Damodaran, B ; Salim, Shirin ; Vargese, Surekha M.: Performance evaluation of MySQL and MongoDB databases. In: *Int. J. Cybern. Inform.(IJCI)* 5 (2016)

[20] Eggert, Daniel: *Entwicklung zweier Setups für eine Augmented Reality Anwendung zum Erkennen von Obst- und Gemüsesorten mit neuronalen Netzen.* 2022. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2022-proj/eggert_gp.pdf

[21] Eggert, Daniel: *Erweiterung eines Trainingdatensatzes für einen Objektdetektor im Kontext eines botanischen Gartens mit Methoden der Data Augmentation.* 2022. – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2022-proj/eggert_hp.pdf

[22] Figueira, Alvaro ; Vaz, Bruno: Survey on synthetic data generation, evaluation methods and GANs. In: *Mathematics* 10 (2022), Nr. 15, S. 2733

[23] Goodfellow, Ian J. ; Pouget-Abadie, Jean ; Mirza, Mehdi ; Xu, Bing ; Warde-Farley, David ; Ozair, Sherjil ; Courville, Aaron ; Bengio, Yoshua: *Generative Adversarial Networks.* 2014

[24] Google: *What is Angular?.* – URL https://angular.io/guide/what-is-angular. – Zugriffsdatum: 2023-01-17

[25] Gos, Konrad ; Zabierowski, Wojciech: The comparison of microservice and monolithic architecture. In: *2020 IEEE XVIth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH)* IEEE (Veranst.), 2020, S. 150–153

[26] Gui, Jie ; Sun, Zhenan ; Wen, Yonggang ; Tao, Dacheng ; Ye, Jieping: A review on generative adversarial networks: Algorithms, theory, and applications. In: *IEEE Transactions on Knowledge and Data Engineering* (2021)

[27] Han, Changhee ; Kitamura, Yoshiro ; Kudo, Akira ; Ichinose, Akimichi ; Rundo, Leonardo ; Furukawa, Yujiro ; Umemoto, Kazuki ; Li, Yuanzhong ; Nakayama, Hideki: Synthesizing diverse lung nodules wherever massively: 3D multi-conditional GAN-based CT image augmentation for object detection. In: *2019 International Conference on 3D Vision (3DV)* IEEE (Veranst.), 2019, S. 729–737

[28] Hesamian, Mohammad H. ; Jia, Wenjing ; He, Xiangjian ; Kennedy, Paul: Deep learning techniques for medical image segmentation: achievements and challenges. In: *Journal of digital imaging* 32 (2019), Nr. 4, S. 582–596

[29] Himschoot, Peter: *Blazor Revealed.* Springer, 2019

[30] Ho, Jonathan ; Jain, Ajay ; Abbeel, Pieter: Denoising diffusion probabilistic models. In: *Advances in Neural Information Processing Systems* 33 (2020), S. 6840–6851

[31] Ho, Jonathan ; Salimans, Tim: Classifier-free diffusion guidance. In: *arXiv preprint arXiv:2207.12598* (2022)

[32] Idris, Nuruldelmia ; Foozy, Cik Feresa M. ; Shamala, Palaniappan: A generic review of web technology: Django and flask. In: *International Journal of Advanced Science Computing and Engineering* 2 (2020), Nr. 1, S. 34–40

[33] Inkawich, Nathan: *DCGAN Tutorial*. – URL https://pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html. – Zugriffsdatum: 2023-03-21

[34] Jin, Xiaohan ; Qi, Ye ; Wu, Shangxuan: Cyclegan face-off. In: *arXiv preprint arXiv:1712.03451* (2017)

[35] Jin, Yong ; Gao, Huifang ; Fan, Xiaoliang ; Khan, Hassan ; Chen, Youxing: Defect Identification of Adhesive Structure Based on DCGAN and YOLOv5. In: *IEEE Access* 10 (2022), S. 79913–79924

[36] Kang, Li-Wei ; Chou, Ke-Lin ; Fu, Ru-Hong: Deep learning-based weather image recognition. In: *2018 International Symposium on Computer, Consumer and Control (IS3C)* IEEE (Veranst.), 2018, S. 384–387

[37] Kang, Minguk ; Shim, Woohyeon ; Cho, Minsu ; Park, Jaesik: Rebooting acgan: Auxiliary classifier gans with stable training. In: *Advances in Neural Information Processing Systems* 34 (2021), S. 23505–23518

[38] Karras, Tero ; Aittala, Miika ; Hellsten, Janne ; Laine, Samuli ; Lehtinen, Jaakko ; Aila, Timo: Training generative adversarial networks with limited data. In: *Advances in neural information processing systems* 33 (2020), S. 12104–12114

[39] Kaushik, Divyansh ; Hovy, Eduard ; Lipton, Zachary C.: Learning the difference that makes a difference with counterfactually-augmented data. In: *arXiv preprint arXiv:1909.12434* (2019)

[40] Kumar, Santosh: A Review on Client-Server based applications and research opportunity. In: *International Journal of Recent Scientific Research* 10 (2019), Nr. 7, S. 33857–3386

[41] L3LLFF: *Flowers*. – URL https://www.kaggle.com/datasets/l3llff/flowers. – Zugriffsdatum: 2023-03-21

[42] Lecouat, Bruno ; Foo, Chuan-Sheng ; Zenati, Houssam ; Chandrasekhar, Vijay R.: Semi-supervised learning with gans: Revisiting manifold regularization. In: *arXiv preprint arXiv:1805.08957* (2018)

[43] Li, Nian ; Zhang, Bo: The Research on Single Page Application Front-end development Based on Vue. In: *Journal of Physics: Conference Series* Bd. 1883 IOP Publishing (Veranst.), 2021, S. 012030

[44] Mamaev, Alexander: *Flowers Recognition*. – URL https://www.kaggle.com/datasets/alxmamaev/flowers-recognition/code. – Zugriffsdatum: 2023-03-21

[45] Menon, Ananya M. ; Jyothi, G N. ; Bhavana, K ; Saara, K: Flask Based Web App on Diabetes Prediction Using Machine Learning. In: *Proceedings of the 2nd International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications* Springer (Veranst.), 2022, S. 785–793

[46] Muccini, Henry ; Vaidhyanathan, Karthik: Software architecture for ml-based systems: what exists and what lies ahead. In: *2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN)* IEEE (Veranst.), 2021, S. 121–128

[47] Odena, Augustus ; Olah, Christopher ; Shlens, Jonathon: Conditional image synthesis with auxiliary classifier gans. In: *International conference on machine learning* PMLR (Veranst.), 2017, S. 2642–2651

[48] Olaniyi, Ebenezer ; Chen, Dong ; Lu, Yuzhen ; Huang, Yanbo: Generative adversarial networks for image augmentation in agriculture: a systematic review. In: *arXiv preprint arXiv:2204.04707* (2022)

[49] OpenAI: *OpenAI*. – URL https://openai.com/dall-e-2/. – Zugriffsdatum: 2023-02-08

[50] Patterson, David ; Gonzalez, Joseph ; Hölzle, Urs ; Le, Quoc ; Liang, Chen ; Munguia, Lluis-Miquel ; Rothchild, Daniel ; So, David R. ; Texier, Maud ; Dean, Jeff: The carbon footprint of machine learning training will plateau, then shrink. In: *Computer* 55 (2022), Nr. 7, S. 18–28

[51] Pytorch: *pytorch/pytorch*. – URL https://hub.docker.com/r/pytorch/pytorch. – Zugriffsdatum: 2023-03-21

[52] PyTorch: *Training a Classifier*. – URL https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html. – Zugriffsdatum: 2023-03-21

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

[54] Ramesh, Aditya ; Dhariwal, Prafulla ; Nichol, Alex ; Chu, Casey ; Chen, Mark: Hierarchical text-conditional image generation with clip latents. In: *arXiv preprint arXiv:2204.06125* (2022)

[55] Rampas, Dominic: *Diffusion Models*. – URL https://github.com/dome272/Diffusion-Models-pytorch. – Zugriffsdatum: 2023-03-21

[56] Richards, Mark ; Ford, Neal: *Handbuch moderner Softwareentwicklung*. 1. O'Reilly, 2021. – 155–165 S

[57] Richardson, Chris: *Pattern: Database per service*. – URL https://microservices.io/patterns/data/database-per-service.html. – Zugriffsdatum: 2023-01-17

[58] Richardson, Chris: *Pattern: Shared database*. – URL https://microservices.io/patterns/data/shared-database.html. – Zugriffsdatum: 2023-01-17

[59] Ricker, Jonas ; Damm, Simon ; Holz, Thorsten ; Fischer, Asja: Towards the Detection of Diffusion Model Deepfakes. In: *arXiv preprint arXiv:2210.14571* (2022)

[60] Rombach, Robin ; Blattmann, Andreas ; Lorenz, Dominik ; Esser, Patrick ; Ommer, Björn: High-resolution image synthesis with latent diffusion models. 2022 IEEE. In: *CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, S. 10674–10685

[61] Rombach, Robin ; Blattmann, Andreas ; Ommer, Björn: Text-guided synthesis of artistic images with retrieval-augmented diffusion models. In: *arXiv preprint arXiv:2207.13038* (2022)

[62] Ronneberger, Olaf ; Fischer, Philipp ; Brox, Thomas: U-net: Convolutional networks for biomedical image segmentation. In: *International Conference on Medical image computing and computer-assisted intervention* Springer (Veranst.), 2015, S. 234–241

[63] Rothmeier, Thomas ; Huber, Werner: Let it Snow: On the Synthesis of Adverse Weather Image Data. In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)* IEEE (Veranst.), 2021, S. 3300–3306

[64] Saabith, AL S. ; Fareez, MMM ; Vinothraj, T: Python current trend applications-an overview. In: *International Journal of Advance Engineering and Research Development* 6 (2019), Nr. 10

[65] Saxena, Divya ; Cao, Jiannong: Generative adversarial networks (GANs) challenges, solutions, and future directions. In: *ACM Computing Surveys (CSUR)* 54 (2021), Nr. 3, S. 1–42

[66] Schmarje, Lars ; Santarossa, Monty ; Schröder, Simon-Martin ; Koch, Reinhard: A survey on semi-, self-and unsupervised learning for image classification. In: *IEEE Access* 9 (2021), S. 82146–82168

[67] Sheikh, Waseem ; Sheikh, Nadeem: A Model-View-Viewmodel (MVVM) Application Framework For Hearing Impairment Diagnosis. In: *arXiv preprint arXiv:1911.08289* (2019)

[68] Song, Jiaming ; Meng, Chenlin ; Ermon, Stefano: Denoising diffusion implicit models. In: *arXiv preprint arXiv:2010.02502* (2020)

[69] Volk, Georg ; Müller, Stefan ; Von Bernuth, Alexander ; Hospach, Dennis ; Bringmann, Oliver: Towards robust CNN-based object detection through augmentation with synthetic rain variations. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)* IEEE (Veranst.), 2019, S. 285–292

[70] Vyas, Rishi: Comparative Analysis on Front-End Frameworks for Web Applications. In: *International Journal for Research in Applied Science and Engineering Technology* 10 (2022), Nr. 7, S. 298–307

[71] Wang, Kunfeng ; Gou, Chao ; Duan, Yanjie ; Lin, Yilun ; Zheng, Xinhu ; Wang, Fei-Yue: Generative adversarial networks: introduction and outlook. In: *IEEE/CAA Journal of Automatica Sinica* 4 (2017), Nr. 4, S. 588–598

[72] Wang, Weilun ; Bao, Jianmin ; Zhou, Wengang ; Chen, Dongdong ; Chen, Dong ; Yuan, Lu ; Li, Houqiang: Semantic image synthesis via diffusion models. In: *arXiv preprint arXiv:2207.00050* (2022)

[73] Washizaki, Hironori ; Uchida, Hiromu ; Khomh, Foutse ; Guéhéneuc, Yann-Gaël: Machine learning architecture and design patterns. In: *IEEE Software* 8 (2020)

[74] Whang, Steven E. ; Lee, Jae-Gil: Data collection and quality challenges for deep learning. In: *Proceedings of the VLDB Endowment* 13 (2020), Nr. 12, S. 3429–3432

[75] Yang, Ling ; Zhang, Zhilong ; Song, Yang ; Hong, Shenda ; Xu, Runsheng ; Zhao, Yue ; Shao, Yingxia ; Zhang, Wentao ; Cui, Bin ; Yang, Ming-Hsuan: Diffusion models: A comprehensive survey of methods and applications. In: *arXiv preprint arXiv:2209.00796* (2022)

[76] Ye, Yuancheng ; Wang, Lijuan ; Wu, Yue ; Chen, Yinpeng ; Tian, Yingli ; Liu, Zicheng ; Zhang, Zhengyou: GAN quality index (GQI) by GAN-induced classifier. (2018)

[77] Zhang, Yan ; Wa, Shiyun ; Sun, Pengshuo ; Wang, Yaojun: Pear Defect Detection Method Based on ResNet and DCGAN. In: *Information* 12 (2021), Nr. 10, S. 397

[78] Zhu, Jun-Yan ; Park, Taesung ; Isola, Phillip ; Efros, Alexei A.: Unpaired image-to-image translation using cycle-consistent adversarial networks. In: *Proceedings of the IEEE international conference on computer vision*, 2017, S. 2223–2232

A Anhang

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original