eTRainer
Trier University
of Applied Sciences

HOCHSCHULE TRIER

Bachelor-Teamprojekt

Analyse, Design und Implementierung von unterschiedlichen Generative Adversarial Network (GAN) Architekturen im Bereich der Bildverarbeitung

Analysis, design and implementation of different Generative Adversarial Network (GAN) architectures in the field of image processing

Elisa Du, Marcel Hoffmann

Mat.Nr.: 976090, 973043

Betreuer:

Prof. Dr. rer. nat. E.-G. Haffner

Datum:

01. Dezember 2023

Eidesstattliche Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht. Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen. Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Ort, Datum

Unterschrift

Abstract

This is a summary of all the important points and achivements of this work.

Zusammenfassung

Hierbei handelt es sich um eine Zusammenfassung aller wichtigne Punkte und Errungenschaften dieser Arbeit.

Abkürzungsverzeichnis

GAN Generative Adversarial Network

Inhaltsverzeichnis

1 Einleitung

Hier wird in die Arbeit eingeleitet.

2 Grundlagen

2.1. Generatuve Adversarial Netowrks

Generative Adversarial Networks, kurz GANs, sind eine aufstrebende Technologie im Bereich des maschinellen Lernens und der künstlichen Intelligenz. Inspiriert von Ian Goodfellow und seinen Kollegen im Jahr 2014, bieten GANs eine effiziente Möglichkeit, tiefe Repräsentationen von Daten zu erlernen, ohne dass große Mengen an annotierten Trainingsdaten benötigt werden(CWD+18). Dies wird durch die Verwendung von Backpropagation und den Wettbewerb zwischen zwei neuronalen Netzen - dem Generator und dem Diskriminator - erreicht. Daraus ergeben sich zahlreiche neue Ansätze zur Generierung realistischer Inhalte. Die Anwendungen reichen von der Bildgenerierung bis hin zur Superauflösung und Textgenerierung (AMB21).

Funktionsweise

Der Generator und der Diskriminator sind die Hauptkomponenten eines GAN. Die beiden neuronalen Netze werden gleichzeitig trainiert und konkurrieren miteinander, wobei der Generator versucht, den Diskriminator zu täuschen, indem er synthetische Inhalte erzeugt. Um die Glaubwürdigkeit des Generators zu erhöhen, so dass der Diskriminator nicht mehr zwischen den Eingaben unterscheiden kann, wird das gesamte Netz trainiert. Die Netze werden in der Regel als mehrschichtige Netzwerke implementiert, die aus Convolutional und Fully-Connected Layers bestehen(CWD+18).

Generator

Der Generator dient zur Erzeugung künstlicher Daten wie Bilder und Texte. Der Generator ist nicht mit dem realen Datensatz verbunden und lernt daher nur durch die Interaktion mit dem Diskriminator. Wenn der Diskriminator nur noch 50% der Eingaben richtig vorhersagt, gilt der Generator als optimal(CWD+18).

Diskriminator

Die Unterscheidung zwischen echten und unechten Eingaben ist Aufgabe des Diskriminators. Der Diskriminator kann sowohl künstliche als auch reale Daten ver-

wenden. Wenn der Diskriminator nicht mehr richtig unterscheiden kann, wird er als konvergierend bezeichnet(AMB21). Andernfalls wird er als optimal bezeichnet, wenn seine Klassifizierungsgenauigkeit maximiert ist. Im Falle eines optimalen Diskriminators wird das Training des Diskriminators gestoppt und der Generator trainiert alleine weiter, um die Genauigkeit des Diskriminators wieder zu verbessern(CWD+18).

Training

Durch das Finden von Parametern für beide Netze wird das Training durchgeführt. Ziel ist die Optimierung beider Netze durch Anwendung von Backpropagation zur Verbesserung der Parameter. Das Training wird oft als schwierig und instabil beschrieben, da es einerseits schwierig erscheint, beide Modelle konvergieren zu lassen. Auf der anderen Seite kann der Generator sehr ähnliche Muster für verschiedene Eingaben erzeugen (MODE COLLAPSE), und der Diskriminatorverlust kann schnell gegen Null konvergieren, so dass es keinen zuverlässigen Weg für die Gradientenaktualisierung zum Generator gibt. Zur Lösung dieser Probleme wurden verschiedene Ansätze vorgeschlagen, wie z.B. die Verwendung heuristischer Verlustfunktionen. Eine weitere Möglichkeit, die von Sonderby et al. vorgeschlagen wurde, besteht darin, den Datensatz vor der Verwendung zu verrauschen(CWD+18).

Adversarial Loss

Der Erfolg von GAN liegt zum Einen an dem verwendeten Adverarsial Verlustfunktion. Dieser
// TODO: PAPER ABBOUT THAT

Anwendungen

GAN wurde ursprünglich für unüberwachtes maschinelles Lernen entwickelt. Die Architektur liefert jedoch ebenso gute Ergebnisse beim halbüberwachten Lernen und beim Reinforcement Learning(AMB21). Aus diesem Grund wird sie in einer Vielzahl von Bereichen wie dem Gesundheitswesen, dem Maschinenbau und dem Bankwesen eingesetzt. Beispielsweise wird GAN in der Medizin zur Erkennung und Behandlung chronischer Krankheiten eingesetzt. Aber auch die Identifikation von 3D-Objekten und die Generierung von realen Bildern und Texten ist durch den Einsatz von GANs möglich.

Limitationen

Die Tatsache, dass ein Generative Adversial Network in der Lage ist, Inhalte zu erzeugen, die nahezu identisch mit realen Inhalten sind, kann in der realen Welt zu Problemen führen, insbesondere bei der menschlichen Bildsynthese. Diese Bilder können von Betrügern verwendet werden, um falsche Profile in sozialen Medien

zu erstellen. Auch dies kann durch den Einsatz von GANs verhindert werden, indem einzigartige und pragmatische Bilder von nicht existierenden Personen erzeugt werden(AMB21). // TODO MODE COLLAPSE NACHLESEN

2.2. Pix2Pix

Isola et al., hat sich als zentrales Framework für Bild-zu-Bild-Übersetzungen auf der Basis von bedingten generativen adversariellen Netzwerken (cGANs) etabliert. Es ermöglicht die Erstellung einer abstrakten Abbildung von einem Eingangsbild zu einem korrespondierenden Ausgangsbild und bewältigt dabei eine vielfältige Palette an Bildübersetzungsaufgaben, wie die Transformation von Skizzen in realistische Bilder oder die Konvertierung von Tages- zu Nachtaufnahmen.
Pix2Pix fungiert hier als Generative Adversarial Network (GAN), spezialisiert auf diverse Formen der Bildübersetzung. Darunter fallen die Umwandlung von Schwarz-Weiß-Fotos in Farbbilder, die Transformation von Skizzen in realistische Bilder, und relevant für diese Arbeit, die Konvertierung von Satellitenbildern in kartographische Darstellungen, ähnlich den Visualisierungen von Google Maps.
Die Architektur von Pix2Pix besteht aus einem Generator und einem Diskriminator. Der Generator, der eine U-Net-Architektur verwendet, besteht aus einem Encoder und einem Decoder. Der Encoder komprimiert das Eingangsbild schrittweise zu einer niedrigdimensionalen Repräsentation, während der Decoder diese dazu nutzt, das Ausgangsbild zu rekonstruieren. Skip-Verbindungen zwischen Encoder und Decoder helfen dabei, sowohl globale als auch lokale Informationen im generierten Bild zu bewahren.
Der Diskriminator nimmt die Form eines PatchGAN-Modells an und bewertet Patches des Bildes, indem er die Wahrscheinlichkeit für die Echtheit jedes Patches ausgibt. Dies ermöglicht die Anwendung des Diskriminators auf Bilder unterschiedlicher Größen. Im Zuge des adversariellen Trainingsprozesses passen sowohl der Generator als auch der Diskriminator ihre Fähigkeiten fortlaufend an. Während der Generator lernt, immer realistischere Übersetzungen zu erzeugen, wird der Diskriminator stetig besser darin, zwischen echten und generierten Bildern zu unterscheiden.

2.2.1. Pix2Pix-Kernkonzepte

Generator

Die Bildverarbeitung hat in den letzten Jahren durch den Einsatz tiefer neuronaler Netzwerke erhebliche Fortschritte gemacht. Im Mittelpunkt vieler dieser Fortschritte steht die U-Net-Architektur, die speziell für die Bildsegmentierung entwickelt wurde. Diese Architektur zeichnet sich durch ihre angeklügelte Kombination aus Encoder- und Decoder- Strukturen sowie durch den Einsatz von Skip-Verbindungen aus (PJTA).
Bei der Encoder-Decoder-Struktur handelt es sich um einen Ansatz, bei dem das

Eingangsbild zunächst durch den Encoder schrittweise reduziert wird. Dieser Prozess dient dazu, wesentliche Merkmale des Bildes zu erfassen. Anschließend wird das Bild durch den Decoder wiederhergestellt, indem die zuvor extrahierten Merkmale verwendet werden. Während dieser Prozesse besteht jedoch das Risiko des Informationsverlustes, insbesondere in den tieferen Schichten des Netzwerks. Um dieses Problem zu adressieren, führt die U-Net-Architektur Skip-Verbindungen ein. Diese direkten Verbindungen zwischen korrespondierenden Schichten des Encoders und Decoders sorgen dafür, dass Detailinformationen nicht verloren gehen. Genauer gesagt, ermöglichen diese Verbindungen den direkten Informationsfluss zwischen jeweils äquivalenten Schichten, wodurch die Rekonstruktion des Bildes im Decoder mit einer höheren Genauigkeit erfolgt(PJTA).
Die Bedeutung von Skip-Verbindungen zeigt sich insbesondere in Anwendungen wie der Bild-zu-Bild-Übersetzung. Hier muss oft ein Bild mit niedriger Auflösung in ein Bild mit hoher Auflösung überführt werden, ohne dass Details verloren gehen. Die U-Net-Architektur, die angereichert mit diesen Verbindungen ist, ermöglicht daher eine feinere Rekonstruktion, die sowohl globale als auch lokale Informationen berücksichtigt (PJTA).
Somit kann die U-Net-Architektur durch ihre Kombination aus Encoder-Decoder-Struktur und Skip-Verbindungen ein effektives Werkzeug für die Bildsegemtierung darstellen. Ihre Fähigkeit, sowohl globale Muster als auch feine Details zu berücksichtigen, macht sie zu einer bevorzugten Wahl für viele Bildverarbeitungsaufgaben (PJTA).
In Abbildung 2.1 ist die typische U-Net-Architektur dargestellt. Die linke Seite des ”U” repräsentiert den Encoder-Teil, der das Eingangsbild schrittweise reduziert und wesentliche Merkmale extrahiert. Die rechte Seite repräsentiert den Decoder-Teil, der das Bild mithilfe der extrahierten Merkmale rekonstruiert. Die horizontalen Linien repräsentieren die Skip-Verbindungen, die sicherstellen, dass Detailinformationen zwischen den korrespondierenden Schichten des Encoder und Decoders direkt übertragen werden (PJTA).
In der Pix2Pix Technologie dient diese U-Net-Architektur als Generator. Er ist das zentrale Element, das für die Bild-zu-Bild-Übersetzung verantwortlich ist. Die Wahl der U-Net-Struktur für den Generator liegt in ihrer Fähigkeit, feinere Details und Kontextinformationen aus dem Eingangsbild beizubehalten, was für die Bild-zu-Bild-Übersetzung von entscheidender Bedeutung ist. Die Encoder-Decoder-Struktur des U-Net ermöglicht es dem Generator, den globalen Kontext des Bildes zu erfassen, während die Skip-Verbindungen sicherstelle, dass auch lokale Details im resultierenden Bild berücksichtigt werden (PJTA).

Diskriminator

Im adversariellen Lernprozess spielen Generatoren und Diskriminatoren eine entscheidende Rolle. Während der Generator versucht, Daten zu erzeugen, die von echten Daten kaum zu unterscheiden sind, evaluiert der Diskriminator die vom Generator erzeugten Daten und versucht, zwischen echten und gefälschten Daten zu unterscheiden (PJTA).

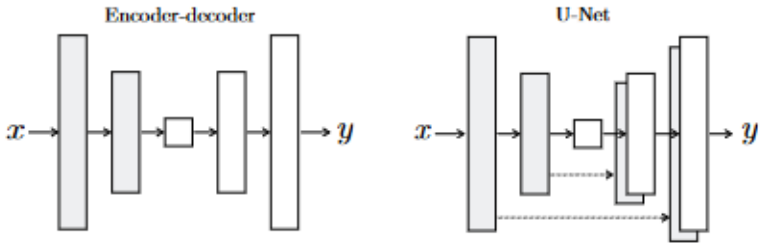

Abbildung 2.1.: Schematische Darstellung der U-Net-Architektur. Die Architektur besteht aus einem Encoder-Teil (links), einem Decoder-Teil (rechts) und Skip-Verbindungen zwischen korrespondierenden Schichten.

Im Kontext von Generative Adverserial Networks (GANs), insbesondere im speziellen Fall des Pix2Pix GANs, spielt der PatchGAN-Diskriminator eine besonders wichtige Rolle. Der zentrale Unterschied dieses Diskriminators zu traditionellen Diskriminatoren liegt in der Art und Weise, wie er Bilder bewertet. Statt das gesamte Bild zu beurteilen, zerlegt der PatchGAN-Diskriminator das Bild in mehrere kleinere Bildabschnitte oder Patches und bewertet jeden Patch einzeln auf seine Echtheit (PJTA).
Ein solches Vorgehen hat den klaren Vorteil, dass feinere Strukturen und Details in den Bildern erkannt und beurteilt werden können. Durch diese segmentierte Beurteilung kann der Diskriminator besser einschätzen, ob die Struktur und Beschaffenheit eines bestimmten Bildteils realistisch ist. Dies ist besonders nützlich, da kleinere Unstimmigkeiten in den Bildern, die ein traditioneller Diskriminator möglicherweise übersieht, vom PatchGAN erfasst werden können.
Ein weiterer Vorteil des PatchGAN-Diskriminators ist seine Skalierbarkeit. Da er auf festen Patchgrößen basiert, kann er flexibel auf Bilder unterschiedlicher Größen angewendet werden, ohne dass das zugrunde liegende Modell geändert werden muss. Dies führt nicht nur zu einer schnelleren Bildverarbeitung, sondern ermöglicht auch eine effiziente Ausführung auf großen Bildern. Darüber hinaus reduziert es potenzielle Kachelartefakte, die bei traditionellen Diskriminatoren auftreten können (PJTA).
Eine Metrik, die oft verwendet wird, um die Leistung des Diskriminator zu beurteilen, ist der FCN-Score. Dieser bewertet die Qualität der vom Generator erzeugten Bilder. Ein hoher FCN-Score zeigt, dass der Diskriminator erfolgreich echte von gefälschten Bildern unterscheiden kann.
Der PatchGAN-Diskriminator kann wenn er effektiv eingesetzt wird, zu besseren und realistischeren Bildern im adverseriellen Lernprozess beitragen. Seine Fähigkeit, lokale Bildinformationen zu bewerten, ermöglicht es auch subtile Unterschiede in den Bildern zu erkennen, was zu einer verbesserten Qualität der generierten Bilder führt (PJTA).

L1-Verlustfunktion

Die L1-Verlustfunktion, auch bekannt als Mean Absolute Error (MAE), spielt eine entscheidende Rolle im Pix2Pix-Modell, einem bedingten Generative Adversari-

al Network (cGAN) für Bild-zu-Bild-Übersetzungen. Diese Verlustfunktion misst den durchschnittlichen absoluten Unterschied zwischen den vorhergesagten und den tatsächlichen Werten, wodurch sie die Genauigkeit der generierten Bilder verbessert, insbesondere im Hinblick auf die niedrigen Frequenzen im Bild. Die L1-Verlustfunktion trägt somit maßgeblich zur Bewahrung der strukturellen Integrität und des Kontexts des Bildes bei (PJTA).
Die Verwendung des L1-Verlusts zusätzlich zum adversariellen Verlust im Pix2Pix-Modell ist von entscheidender Bedeutung. Während der adversarielle Verlust darauf abzielt, die generierten Bilder realistisch erscheinen zu lassen, konzentriert sich der L1-Verlust auf die Genauigkeit der niedrigen Frequenzen, um die strukturelle Integrität und den Kontext des Bildes zu bewahren. Diese Kombination ermöglicht es, sowohl die niedrigen als auch die hohen Frequenzen im Bild effektiv zu erfassen, was zu generierten Bildern führt, die sowohl strukturell korrekt als auch visuell ansprechend sind (PJTA).
Die L1-Verlustfunktion neigt jedoch dazu, bei den hohen Frequenzen unscharfe Ergebnisse zu liefern. Dies liegt daran, dass der L1-Verlust den Median der möglichen Werte bevorzugt, was zu einer Glättung der Bildtexturen führen kann. Um dieses Problem zu adressieren und scharfe, hochfrequente Details im Bild zu erhalten, wird der L1-Verlust im Pix2Pix-Modell mit einem adversariellen Verlust kombiniert. Diese synergetische Kombination von Verlustfunktionen ermöglicht es dem Pix2Pix-Modell, hochwertige Bild-zu-Bild-Übersetzungen durchzuführen, die sowohl visuell ansprechend als auch strukturell korrekt sind (PJTA).
Darüber hinaus hat sich die Kombination von L1-Verlust und adversariellen Verlust im Pix2Pix-Modell als nützlich für eine Vielzahl von Bild-zu-Bild-Übersetzungsproblemen erwiesen, einschließlich semantischer Segmentierung und Farbgebung. Durch die effektive Erfassung sowohl der niedrigen als auch der hohen Frequenzen im Bild trägt das Pix2Pix-Modell dazu bei, die Qualität der generierten Bilder zu verbessern und ihre Anwendbarkeit auf verschiedene Probleme zu erweitern (PJTA).

Training

Der Trainingsprozess von Pix2Pix-Generative Adversarial Networks in der Bild-zu-Bild-Übersetzung geht über die bloße Erlernung der Abbildung von Eingabe- zu Ausgabebildern hinaus. Er umfasst auch das Entwickeln einer maßgeschneiderten Verlustfunktion, die speziell auf diese Art der Bildtransformation abgestimmt ist. Dieser umfassende Ansatz ermöglicht es Pix2Pix, sich flexibel an eine Vielzahl von Problemen anzupassen, die in der Vergangenheit unterschiedliche und spezialisierte Ansätze für die Verlustfunktion erforderten. Dadurch wird die breite Anwendbarkeit und Effektivität des Pix2Pix-Modells in verschiedenen Bildübersetzungsaufgaben deutlich. Pix2Pix benötigt eine spezifische Art von Trainingsdaten, um effektiv zu funktionieren. Die Trainingsdaten bestehen aus Paaren von Bildern, wobei jedes Paar ein Eingabebild und das entsprechende Ausgabebild enthält. Diese Bilder können 1-3 Kanäle aufweisen, was bedeutet, dass das Modell sowohl mit monochromatischen (Graustufen) als auch mit farbigen Bildern (RGB) arbeiten kann. Diese Flexibilität in der Kanalverarbeitung ermöglicht eine breite-

re Anwendung des Pix2Pix-Modells auf verschiedene Bildtypen. Die Bilder müssen nicht in einer bestimmten Weise vorverarbeitet werden, da das Modell direkt auf den Rohpixeln arbeitet. Diese Flexibilität in der Kanalverarbeitung und die Fähigkeit, direkt auf Rohpixeln zu arbeiten, unterstreichen die Vielseitigkeit des Pix2Pix-Modells. Dies wird weiter durch die sorgfältige Auswahl spezifischer Trainingsparameter und Hyperparameter-Optimierungsstrategien ergänzt. Der Adam-Grandientenoptimierungsalgorithmus ist eine Methode zur Optimierung des maschinellen Lernens, die auf adaptiven Schätzungen niedrigerer Ordnung basiert. Er passt automatisch die Lernrate während des Trainings an und eignet sich besonders gut für Probleme mit großen Datensätzen und/oder vielen Parametern. Durch empirische Tests hat sich herausgestellt das eine initiale Lernrate von 0.0002 und die Momentum-Parameter von $\beta 1 = 0.5$ und $\beta = 0.999$ optimal sind, um die Balance zwischen Lerngeschwindigkeit und Stabilität des Trainingsprozesses zu optimieren. Auch die Wahl einer kleinen Batchgröße, typischerweise 1, spielt eine entscheidende Rolle, um die Trainingseffizienz zu maximieren und qualitativ hochwertige Ergebnisse zu erzielen. Diese spezifischen Einstellungen der Trainingsparameter tragen maßgeblich dazu bei, das Potenzial des Pix2Pix-Modells voll auszuschöpfen.(PJTA).
Im Rahmen des Trainingsprozesses von Pix2Pix wird eine iterative Methode verwendet, bei der der Generator und der Diskriminator abwechselnd trainiert werden. Zunächst werden die Trainingsdaten vorbereitet, die aus Paaren von Eingabe- und Zielbildern besteht. Diese Bilder werden auf einen Wertebereich von -1 bis +1 skaliert. Diese Normalisierung ist wichtig, da sie zur Stabilisierung des Trainingsprozesses beiträgt. Sie ermöglicht es dem Modell, mit einem standardisierten Datensatz zu arbeiten, was die Lernrate und die Konvergenzgeschwindigkeit verbessert. Während des Tranings versucht der Generator, Bilder zu erzeugen, die vom Diskriminator nicht von realen Bildern unterschieden werden können. Der Diskriminator hingegen lernt, zwischen echten und vom Generator erzeugten Bildern zu unterscheiden. Eine Schlüsselkomponente dieses Prozesses ist die Verwendung einer zusammengesetzten Verlustfunktion, die sowohl den adverseriellen Verlust (bewertet vom Diskriminator) als auch den L1-Verlust(mittlerer absoluter Fehler zwischen generiertem Bild und Zielbild) umfasst. Dadurch wird der Generator dazu angehalten, realistische Übersetzungen der Eingabebilder zu generieren. Dieses Gleichgewicht zwischen Generator und Diskriminator ist entscheidend für die Effektivität des Pix2Pix-Modells (Haz21).

2.2.2. Anwendungen von Pix2Pix

Hier können Sie über die Anwendungen von Pix2Pix schreiben.

2.3. CycleGAN

CycleGAN, das 2017 von Jun-Yan Zhu et al. vorgestellt wurde, stellt eine neue Entwicklung im Bereich des maschinellen Lernens und insbesondere der Bildüber-

setzung zwischen unpaaren Domänen dar. Es erweitert die Pix2Pix-Architektur durch die Einführung einer Cycle Consistency Loss-Funktion, die sicherstellt, dass das Originalbild nach einem Übersetzungs- und Rückübersetzungszyklus erhalten bleibt. Der Generator G transformiert Bilder aus der Domäne X in die Domäne Y, während der Generator F den umgekehrten Prozess durchführt. Diese Transformationen werden ohne gepaarte Trainingsdaten durchgeführt.

2.3.1. CycleGAN - Kernkonzepte

Architektur der Generatoren

Die Architektur der Generatoren in CycleGAN spielt eine entscheidende Rolle bei der erfolgreichen Durchführung von Bildübersetzungen zwischen verschiedenen Domänen. Typischerweise basieren die Generatoren auf dem ResNet-Ansatz, der für seine Fähigkeit bekannt ist, tiefe neuronale Netze zu trainieren.
ResNet-Blöcke bestehen aus Faltungsschichten, Normalisierungsschichten und Aktivierungsfunktionen. Sie ermöglichen Generatoren, komplexe Transformationen durchzuführen. Die Verwendung von ResNet-Blöcken erleichtert auch das Training tiefer Netze, was für qualitativ hochwertige Übersetzungen wichtig ist.

Architektur der Diskriminatoren

Wie bei Pix2Pix ist die übliche Architektur für Diskriminatoren in CycleGAN PatchGAN, wobei das Bild in kleine Patches aufgeteilt wird und jeder Patch separat klassifiziert wird. Diese Methode ermöglicht eine feine Unterscheidung zwischen echten und generierten Bildern auf lokaler Ebene (ZPIE).
Die Diskriminatoren bestehen in der Regel aus Convolutional Layer, gefolgt von Normalization Layer und Activation Function. In einigen Implementierungen von CycleGAN wird die instanzielle Normalisierung der üblichen Batch-Normalisierung vorgezogen ().
Die instanzielle Normalisierung ist eine Variante der Normalisierung, die auf Instanzebene durchgeführt wird. Im Gegensatz zur Batch-Normalisierung, bei der die Normalisierung über die gesamte Batch-Dimension durchgeführt wird, wird bei der Instanz-Normalisierung jede Instanz bzw. jedes Bild einzeln betrachtet. Dies kann besonders vorteilhaft sein, wenn die statistischen Eigenschaften der einzelnen Instanzen variieren.
Die Verwendung der Instanznormalisierung in den Diskriminatoren von CycleGAN kann helfen, eine stabilere und konsistentere Konvergenz während des Trainings zu erreichen.

Training

Das Training von CycleGAN erfolgt nach einem kompetitiven Verfahren. Die Generatoren $G : X \to Y$ und $F : Y \to X$ konkurrieren mit den entsprechenden Diskriminatoren D_X und D_Y. D_X versucht, die von F erzeugten Bilder von den

echten Bildern aus X zu unterscheiden, während D_Y versucht, die von G erzeugten Bilder von den echten Bildern aus der Domäne Y zu unterscheiden. Die adversen Verluste sind so optimiert, dass die erzeugten Bilder für die Diskriminatoren kaum von den echten Bildern zu unterscheiden sind. (ZPIE).

Cycle - Konsistenz

CycleGAN führt zusätzlich eine Cycle-Konsistenz ein. Diese stellt sicher, dass die Übersetzungen zwischen den Domänen sowohl vorwärts (X nach Y) als auch rückwärts (Y nach X) konsistent sind.
Die Kernidee besteht darin, dass nach der Übersetzung von X nach Y und zurück nach X das resultierende Bild dem ursprünglichen X entsprechen sollte. Um dies zu erreichen, wird die Differenz zwischen dem Originalbild x und dem zyklisch übersetzten Bild $F(G(x))$ mit Hilfe der L1-Verlust minimiert.
Durch die Einführung dieser zyklischen Konsistenz wird das Problem des Modekollapses gelöst. Die weitere Verlustfunktion stellt sicher, dass die generierten Bilder mehr Strukturen enthalten und somit konsistentere Übersetzungen zwischen den Domänen liefern (ZPIE).

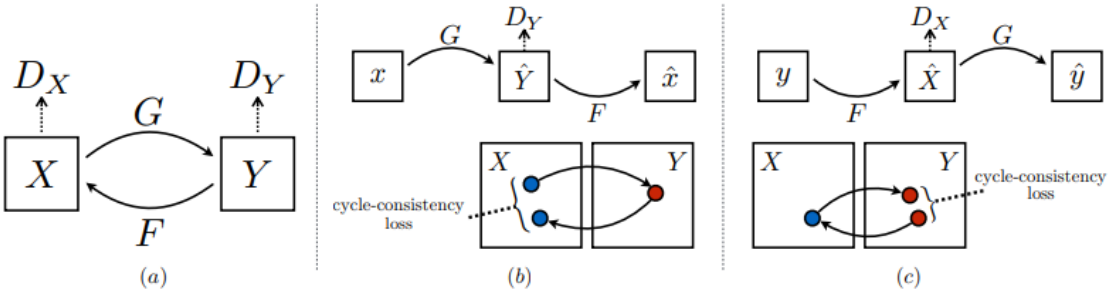

Abbildung 2.2.: (a) Modell des CycleGANs, bestehend aus zwei Generatoren $F : Y \to X$ und $G : X \to Y$ und zugehörige adversarielle Diskriminatoren D_X und D_Y, (b) Cycle-Konsistenz $F(G(x)) \approx x$, (c) Cycle-Konsistenz $G(F(y)) \approx y$(ZPIE)

Identity - Loss

Zusätzlich zu den adversariellen und zyklischen Verlusten kann ein Identitätsverlust in die Gesamtverlustfunktion integriert werden, um sicherzustellen, dass die Farbkomposition des Eingabebildes beibehalten wird, während es ins Ausgabebild übersetzt wird. Insbesondere bei der Erstellung von Fotografien aus Gemälden hat sich diese Methode bewährt. Wenn der Generator G ein Bild aus dem Bereich Y erhält, darf es sich aufgrund seiner bereits vorhandenen Zugehörigkeit zu diesem Bereich nicht mehr verändern. Der Verlust wird dabei mittels des L1-Verlust ermittelt, bei dem die Differenz zwischen den Pixeln des generierten Bildes $G(y)$ und dem Referenzbild $y \in Y$ erfasst wird. Das gleiche Verfahren wird für den anderen Generator F angewendet (ZPIE)

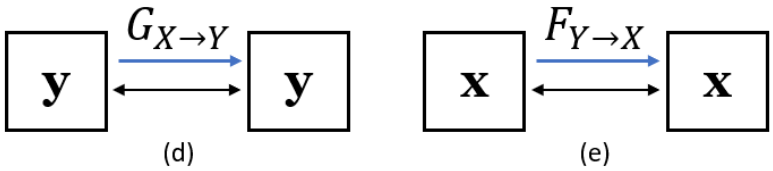

Abbildung 2.3.: Identity-Mapping für (d) Generator G und (e) Generator F

2.3.2. Anwendungen von CycleGAN

CycleGAN hat sich als äußerst vielseitiges Modell erwiesen und wird in verschiedenen Anwendungsbereichen eingesetzt. Die Fähigkeit, Bildübersetzungen zwischen unpaaren Domänen durchzuführen, hat zu zahlreichen innovativen Anwendungen geführt.
Eine der prominentesten Anwendungen von CycleGAN ist die Bild-zu-Bild-Übersetzung. Dies beinhaltet die Transformation von Bildern zwischen verschiedenen Stilen, Szenarien oder Kunstwerken. Beispielsweise kann CycleGAN verwendet werden, um Fotos in den Stil berühmter Kunstwerke zu transformieren, was einen einzigartigen und kreativen Ansatz für die Bildbearbeitung bietet. Ein weiteres Anwendungsgebiet von CycleGAN ist die Stilübertragung. Hier können Stile von einem Bild auf ein anderes übertragen werden, ohne dass gepaarte Trainingsdaten benötigt werden. So ist es möglich, den Stil eines Gemäldes auf ein fotografisches Bild zu übertragen oder umgekehrt. CycleGAN kann auch für die Übersetzung zwischen verschiedenen Farbbereichen verwendet werden. Zum Beispiel kann es verwendet werden, um Schwarz-Weiß-Bilder in Farbversionen zu übersetzen oder den Farbton von Bildern anzupassen, ohne dass gepaarte Trainingsdaten benötigt werden.

2.4. Layers

// TODO

2.5. Bibliotheken

// TODO

2.5.1. Tensorflow

Tensorflow ist ein Open-Source-Framework,

2.5.2. Keras

3
Literaturreview

Hier wird der Inhalt der Arbeit präsentiert.

4
Problembeschreibung

4.1. Datenbeschaffung und -vorverarbeitung

4.2. Architekturentwurf

Pix2Pix

CycleGAN

5 Lösungsbeschreibung

5.1. Bewertungskriterien

5.2. Pix2Pix: Ergebnisse und objektive Bewertung

5.3. CycleGAN: Ergebnisse und objekte Bewertung

5.4. Vergleich und Bewertung von Pix2Pix und CycleGAN

5.5. Implementierung der Pix2PixGAN-Architektur

5.5.1. Generator

Die Struktur des Generator in der Pix2Pix-Implementierung ist ein wesentlicher Aspekt, der die Leistungsfähigkeit des Modells bestimmt. Der Generator ist als U-Net-Architektur aufgebaut, die aus dem Encoder und Decoder bestehen die wiederum aufeinanderfolgende Downsampling- und Upsampling-Schritten beinhalten. Im Encoder-Teil des Generators wird das Downsampling durch eine Reihe von Convolutional Neuronal Network (CNN) Schichten realisiert, die durch die downsample-Funktion innerhalb des downstack definiert sind.

Die downsample-Funktion erstellt eine Downsampling-Schicht, die mittels einer Conv2D-Schicht mit spezifischen Filtern und Kernel-Größen die räumlichen Dimensionen der Eingabebilder reduziert. Zur Verbesserung der Stabilität und Leistungsfähigkeit des Modells integriert die Funktion optional eine Batch-Normalisierung. Diese Normalisierung reguliert und standardisiert die Ausgabe der Conv2D-Schicht, was dazu beiträgt, das Training effizienter zu gestalten. Darüber hinaus beinhaltet die Funktion eine LeakyReLU-Aktivierung, eine Variation der herkömmlichen ReLU-Aktivierungsfunktion. LeakyReLU ermöglicht es, dass auch für negative

Eingabewerte ein kleiner Gradient erhalten bleibt, wodurch das Problem der inaktiven Neuronen, bekannt als ßterbende ReLUs", vermieden wird.
Im Anschluss daran erfolgt das Upsampling im Decoder-Teil des Generators, das durch die upsample-Funktion innerhalb des upstack repräsentiert wird. Diese Schichten arbeiten daran, die Merkmale auf ein höher ausgelöstes Format zu projizieren und die Bildgröße wiederherzustellen.
Die upsampling-Funktion verwendet eine spezielle Art von konvolutioneller Schicht, die Conv2DTranspose-Schicht, um die Bildgröße zu erhöhen. Diese Schicht kehrt den Prozess einer konvolutionellen Schicht um, indem sie die Eingabedaten expandiert, was für das Wiederherstellen einer größerem Bildgröße im Generator unerlässlich ist. Zusätzlich zur Conv2DTranspose-Schicht integriert die upsample-Funktion eine Batch-Normalisierung, die zur Stabilisierung des Lernprozesses beiträgt, indem sie die Ausgaben der Conv2DTranspose-Schicht normalisiert. Dies ist ein wichtiger Schritt, um die interne Kovariantenverschiebung zu reduzieren und die Leistung des Modells zu verbessern. Ein weiteres wichtiges Merkmal der Funktion ist die optional Anwendung von Dropout. Wenn diese aktiviert ist, hilft Dropout, Überanpassungen(Overfitting) zu vermeiden, indem zufällig eine bestimmte Anzahl von Neuronen während des Trainingsprozesses ausgeschaltet werden. Dies trägt dazu bei dass das Modell robustere und generalisierbare Merkmale lernt. Schließlich wird eine ReLU-Aktivierungsfunktion angewendet, die dafür sorgt, dass das Modell nicht-lineare Zusammenhänge lernt.
Die Skip-Verbindungen werden im Generator durch die Speicherung und spätere Verwendung der Ausgaben der Downsampling-Schichten in der skips-Liste realisiert. Nach dem Downsampling-Prozess werden diese gespeicherten Ausgaben in umgekehrter Reihenfolge durchlaufen und mit den Ausgaben der Upsampling-Schichten mittels einer Concatenate-Operation verbunden. Diese Kombination von hoch- und niedrigstufigen Merkmalen führt zu einer detaillierteren und genaueren Bildrekonstruktion.
Schließlich wird das endgültige Bild durch die letzte Schicht des Generators erzeugt, eine Conv2DTranspose-Schicht, die die Ausgabe des Generators darstellt. Diese letzte Schicht spielt eine entscheidende Rolle bei der Erzeugung des endgültigen Bildes, das die kombinierten Merkmale aus dem gesamten Netzwerk nutzt.

```
down_stack = [
downsample(64, 4, apply_batchnorm=False), # (batch_size, 128, 128, 64)
downsample(128, 4), # (batch_size, 64, 64, 128)
downsample(256, 4), # (batch_size, 32, 32, 256)
downsample(512, 4), # (batch_size, 16, 16, 512)
downsample(512, 4), # (batch_size, 8, 8, 512)
downsample(512, 4), # (batch_size, 4, 4, 512)
downsample(512, 4), # (batch_size, 2, 2, 512)
downsample(512, 4), # (batch_size, 1, 1, 512)
]
```

Code-Auszug 5.1: Downsampling-Schritt


```
def downsample(filters, size, apply_batchnorm=True):
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
  tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
  kernel_initializer=initializer, use_bias=False))

  if apply_batchnorm:
      result.add(tf.keras.layers.BatchNormalization())

  result.add(tf.keras.layers.LeakyReLU())

  return result
```

Code-Auszug 5.2: Downsampling-Schicht

```
up_stack = [
upsample(512, 4, apply_dropout=True), # (batch_size, 2, 2, 1024)
upsample(512, 4, apply_dropout=True), # (batch_size, 4, 4, 1024)
upsample(512, 4, apply_dropout=True), # (batch_size, 8, 8, 1024)
upsample(512, 4),  # (batch_size, 16, 16, 1024)
upsample(256, 4),  # (batch_size, 32, 32, 512)
upsample(128, 4),  # (batch_size, 64, 64, 256)
upsample(64, 4),  # (batch_size, 128, 128, 128)
]
```

Code-Auszug 5.3: Upsampling-Schritt

```
def upsample(filters, size, apply_dropout=True):
 initializer = tf.random_normal_initializer(0., 0.02)

 result = tf.keras.Sequential()
 result.add(
 tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
 padding='same',
 kernel_initializer = initializer,
 use_bias=False))

 result.add(tf.keras.layers.BatchNormalization())

 if apply_dropout:
 result.add(tf.keras.layers.Dropout(0.5))

 result.add(tf.keras.layers.ReLU())

 return result
 ]
```

Code-Auszug 5.4: Upsampling-Schritt

```
1           initializer   = tf.random_normal_initializer(0., 0.02)
2           last  = tf.keras.layers.Conv2DTranspose(
                OUTPUT_CHANNELS, 4,
3           strides=2,
4           padding='same',
5           kernel_initializer = initializer,
6           activation='tanh')  # (batch_size, 256, 256, 3)
7
8           x = inputs
9
10          # Downsampling through the model
11          skips = []
12          for down in down_stack:
13                  x = down(x)
14                  skips.append(x)
15
16          skips = reversed(skips[:-1])
17
18          # Upsampling and establishing the skip connections
19          for up, skip in zip(up_stack, skips):
20                  x = up(x)
21                  x = tf.keras.layers.Concatenate()([x, skip])
22
23          x = last(x)
24
25          return tf.keras.Model(inputs=inputs, outputs=x)
26          ]
```

Code-Auszug 5.5: Skip Verbindungen

5.6. Training- und Testdaten

Das Training und die Evaluierung von Generative Adversarial Networks (GANs) erfordern die klare Definition von Trainings- und Testdatensätzen. Der entscheidende Unterschied zwischen diesen Datensätzen besteht darin, dass das Modell während des Trainings auf den Trainingsdaten optimiert wird, während die Testdaten verwendet werden, um die Leistung und die Generalisierungsfähigkeiten des Modells zu bewerten.

5.6.1. Datenladung für GAN-Training

Für das effektive Training von GANs ist der Zugriff auf qualitativ hochwertige Datensätze von entscheidender Bedeutung. In diesem Kontext bietet TensorFlow eine umfassende Sammlung öffentlich verfügbarer Datensätze. Die verwendeten Datensätze werden von der Quelle `https://efrosgans.eecs.berkeley.edu` heruntergeladen und lokal extrahiert.

```
1 path_to_zip = tf.keras.utils.get_file(fname=f"{dataset_name}.tar.gz",
2 origin=_URL, extract=True)
```

Code-Auszug 5.6: Laden eines Datensatzes von einer URL

Die Transformation und Vorverarbeitung der Bilddaten erfolgt durch die TensorFlow-Datensatz-API. Diese API bietet eine effiziente Datenpipeline für das Laden und Verarbeiten von Daten, insbesondere für den Einsatz in Machine-Learning-Modellen. In den folgenden Implementierungen wird ein Dataset durch eine Liste von Dateipfaden als Zeichenketten erzeugt.

```
1 train_horses = tf.data.Dataset.list_files(str(PATH / 'trainA/*.jpg'))
```

Code-Auszug 5.7: Erzeugung eines Tensorflow-Dataset aus CycleGAN Implementierung

5.6.2. Vorverarbeitung des Datensatzes

Um die Leistung der GAN-Modelle zu verbessern, werden vor dem Training Variationen in den Trainingsdaten eingeführt. Dieser Prozess, der Datenjittering und Normalisierung umfasst, trägt dazu bei, das Modell robuster zu machen und eine bessere Konvergenz während des Trainings zu erreichen.

```
1 def normalize(image):
2     image = (image / 127.5) - 1
3     return image
```

Code-Auszug 5.8: Vorverarbeitung des Datensatzes: Normalisierung

Datenjittering wird durchgeführt, indem die heruntergeladenen Bilder (256x256) auf eine größere Größe (286x286) mit der Nearest-Neighbor-Methode skaliert. Bei dieser Methode wird für jedes Pixel im skalierten Bild der Farbwert des nächstgelegenen Pixels im Originalbild übernommen, um eine einfache und effiziente Skalierung zu ermöglichen. Das Bilder werden daraufhin zufällig zugeschnitten. Darüber hinaus wird eine zufällige Spiegelung angewendet.

```
1 def random_crop(image):
2     cropped_image = tf.image.random_crop(image, size=(
          IMG_HEIGHT, IMG_WIDTH, 3))
3     return cropped_image
4
5 def random_jitter(image):
6     image = tf.image.resize(image, [286, 286], method=tf.image.
          ResizeMethod.NEAREST_NEIGHBOR)
7     image = random_crop(image)
8     image = tf.image.random_flip_left_right(image)
9     return image
```

Code-Auszug 5.9: Vorverarbeitung des Datensatzes: Jittering

Zusätzlich dazu werden die Bildpfade geladen und in das resultierende JPEG-Format decodiert.

```
1 def load_image(image_path):
2     image = tf.io.read_file(image_path)
3     image = tf.io.decode_jpeg(image, channels=3)
4     image = tf.cast(image, tf.float32)
5     return image
```

Code-Auszug 5.10: Vorverarbeitung des Datensatzes: Jittering

Die vorverarbeiteten Bilder werden dann in TensorFlow-Datasets integriert. Nachfolgend wird der Trainingsdatensatz zufällig gemischt und in Batches gruppiert, wodurch sichergestellt wird, dass das Modell nicht von der Reihenfolge der Datenpunkte beeinflusst wird.

```
1  def preprocess_image_train(image_path):
2      image = load_image(image_path)
3      image = random_jitter(image)
4      image = normalize(image) if not tf.reduce_all(tf.math.logical_and(
           image >= 0.0, image <= 1.0)) else image
5      return image
6
7  def preprocess_image_test(image_path):
8      image = load_image(image_path)
9      image = normalize(image) if not tf.reduce_all(tf.math.logical_and(
           image >= 0.0, image <= 1.0)) else image
10     return image
11
12 train_dataset = tf.data.Dataset. list_files (str(PATH / 'train/*.jpg'))
13 train_dataset = train_dataset.map(preprocess_image_train,
       num_parallel_calls=tf.data.AUTOTUNE)
14 train_dataset = train_dataset.shuffle(BUFFER_SIZE)
15 train_dataset = train_dataset.batch(BATCH_SIZE)
```

Code-Auszug 5.11: Integration der vorverarbeiteten Bilder in Tensorflow-Datasets

Diese umfassende Vorverarbeitung stellt sicher, dass das GAN-Modell auf optimal vorbereiteten Daten trainiert wird, um eine maximale Leistung und Generalisierungsfähigkeit zu erreichen.

5.7. Implementierung der CycleGAN-Architektur

Nach den theoretischen Grundlagen der CycleGAN-Architektur und der Datenvorverarbeitung in den vorherigen Abschnitten, wird nun die konkrete Implementierung der Architekturen unter Verwendung von TensorFlow und Keras vorgestellt Die Generator- und Diskriminatorarchitekturen wurden entsprechend der im Grundlagenkapitel erklärten theoretischen Grundlagen umgesetzt, insbesondere den Empfehlungen von Zhu et al. (2017)(ZPIE) folgend. Der Generator setzt sich aus einem Enkoder-Block, mehreren Residualblöcken und einem Dekoder-Block zusammen. Der Diskriminator wurde als sequentielles Modell implementiert und umfasst mehrere Convolutional-Schichten, konzipiert als PatchGAN.
Die spezifischen Implementierungsdetails sind im beigefügten Code zu finden.

5.7.1. Verlustfunktion

Während des Trainings werden verschiedene Verlustfunktionen verwendet, um sicherzustellen, dass der Generator qualitativ hochwertige Bilder generiert und dass

die Transformationen zwischen den Domänen konsistent sind. Die zentralen Verlustfunktionen, insbesondere die Gesamtverlustfunktionen für den Generator und den Diskriminator, werden im Folgenden erläutert.
Für die Klassifizierung, ob es sich um echte oder generierte Bilder handelt, wird in der Implementierung der *BinaryCrossentropy*-Verlustfunktion aus TensorFlow/Keras verwendet. Dieser berechnet den binären Kreuzentropieverlust zwischen den Zielwerten und den Vorhersagen [1].

```
1  loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

Code-Auszug 5.12: Vorverarbeitung des Datensatzes: Jittering

Gesamtverlust des Generators

Der Gesamtverlust des Generators setzt sich aus dem adversariellen Verlust und dem Zykluskonsistenz-Verlust zusammen. Optional kann der Identitätsverlust berücksichtigt werden, was in der Implementierung erfolgte. Die Integration des Identitätsverlusts stellt sicher, dass das Modell die Struktur des Originalbildes beibehält, was zu einem konsistenteren Transformationsprozess führt. Die spezifischen Implementierungen der adversariellen Verlustfunktion des Generators, des Cycle Consistency Loss und des Identitätsverlusts sind im Code-Anhang zu finden.

```
1  def generator_loss(real_x, cycled_x, real_y, cycled_y, identity,
       discriminator_generated, step):
2      gan_loss = generator_adversarial_loss(discriminator_generated)
3      cycle_loss = cycle_consistency_loss(real_x, cycled_x) +
           cycle_consistency_loss(real_y, cycled_y)
4      id_loss = identity_loss(real_x, identity)
5      total_loss = gan_loss + cycle_loss + id_loss
6      return total_loss
```

Code-Auszug 5.13: Vorverarbeitung des Datensatzes: Jittering

Gesamtverlust des Diskriminators

Die Gesamtverlustfunktion des Diskriminators setzt sich aus dem adversariellen Verlust für echte und generierte Bilder zusammen. Der Diskriminator wird entsprechend trainiert, echte Bilder als "1ünd generierte Beispiele als "0ßu klassifizieren. Die Multiplikation des Gesamtverlustes mit dem Wert 0.5 bildet den Durchschnitt des Gesamtverlustes und stellt sicher, dass die Gradientenaktualisierung während des Prozesses angemessen skaliert wird.

[1] https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy

5.7.2. Training und Hyperparameter

Das Training wurde über 300 Epochen durchgeführt, wobei die Modelle durch spezifische Optimierer optimiert wurden. In jedem Durchlauf wurde ein Bild aus den Trainingsdaten ausgewählt, und der Generator wurde auf dieses angewendet, um inkrementell die Verbesserung zu verfolgen. Die generierten Bilder wurden mittels *plt.savefig()* aus der Matplotlib-Bibliothek gespeichert, um auch nach dem Training darauf zugreifen zu können.
Diese Vorgehensweise ermöglicht eine systematische Überprüfung der Modellleistung und eine visuelle Darstellung der Fortschritte während des Trainingsprozesses. Die Wahl der Hyperparameter, einschließlich Lernraten und des Lambda-Werts für den Cycle Consistency Loss, wurde durch abgestimmte Experimente ermittelt. Die Optimierer wurden sorgfältig ausgewählt, um eine effiziente Konvergenz während des Trainings zu gewährleisten.

```
1 def discriminator_adversarial_loss(real, generated):
2     real_loss = loss_obj(tf.ones_like(real), real)
3     generated_loss = loss_obj(tf.zeros_like(generated), generated)
4     total_disc_loss = real_loss + generated_loss
5     return total_disc_loss * 0.5
```

Code-Auszug 5.14: Vorverarbeitung des Datensatzes: Jittering

```python
def Generator():
    inputs = tf.keras.layers.Input(shape=[None,None,3])

    # Layer 1
    x = layers.Conv2D(64, 7, strides=1, padding='same')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    # Layer 2+ 3
    x = convolutional_layer(x, 128, 3,2)
    x = convolutional_layer(x, 256, 3, 2)

    # Residual Blocks
    for _ in range(6):
        x = residual_block(x, 256)

    # Layer 10 + 11
    x = t_convolutional_layer(x, 128, 3, 2)
    x = t_convolutional_layer(x, 64, 3, 2)

    # Layer 12
    outputs = layers.Conv2D(3, 7, strides=1, padding='same', activation
        ='tanh')(x)

    return tf.keras.Model(inputs=inputs, outputs=outputs)
```

Code-Auszug 5.15: CycleGAN Generator in Tensorflow

```
1  def Discriminator():
2      model = tf.keras.Sequential()
3
4      # Layer 1
5      model.add(layers.Conv2D(64, 4, strides=2, padding='same',
           input_shape=(256,256,3)))
6      model.add(layers.BatchNormalization())
7      model.add(layers.Activation('relu'))
8
9      # Layer 2
10     model.add(layers.Conv2D(128, 4, strides=2, padding='same'))
11     model.add(layers.BatchNormalization())
12     model.add(layers.Activation('relu'))
13
14     # Layer 3
15     model.add(layers.Conv2D(256, 4, strides=2, padding='same'))
16     model.add(layers.BatchNormalization())
17     model.add(layers.Activation('relu'))
18
19     # Layer 4
20     model.add(layers.Conv2D(512, 4, strides=2, padding='same'))
21     model.add(layers.BatchNormalization())
22     model.add(layers.Activation('relu'))
23
24     # Layer 5
25     model.add(layers.Conv2D(1, 4, strides=1, padding='same'))
26     model.add(layers.BatchNormalization())
27     model.add(layers.Activation('sigmoid'))
28
29     return model
```

Code-Auszug 5.16: CycleGAN Diskriminator in Tensorflow

6 Evaluation

6.1. Vergleich von Pix2Pix und CycleGAN

- Matching Paare von Bildern sind ebenfalls für das Training nicht nötig (crewall)
- Macht die Datenvorbereitung einfacher und öffnet neue Techniken für Applikationen (crewall)

7 Fazit und Ausblick

Hier wird ein Fazit und ein Ausblick gegeben.

7.1. Fazit

Fazit.

7.2. Ausblick

Ausblick.

A
Anhang - Code

Hier sehen Sie den gesamten Quellcode!

B

Anhang - Dokumentationen

Hier sehen Sie die gesamten Dokumentationen zu den erstellten Programmen.

Literaturverzeichnis

[.2019] *Proceedings of The 7th International Conference on Intelligent Systems and Image Processing 2019*. The Institute of Industrial Application Engineers, 2019 . – ISBN 9784907220198

[AMB21] Aggarwal, Alankrita ; Mittal, Mamta ; Battineni, Gopi: Generative adversarial network: An overview of theory and applications. In: *International Journal of Information Management Data Insights* 1 (2021), Nr. 1, S. 100004. http://dx.doi.org/10.1016/j.jjimei.2020.100004. – DOI 10.1016/j.jjimei.2020.100004. – ISSN 26670968

[CWD+18] Creswell, Antonia ; White, Tom ; Dumoulin, Vincent ; Arulkumaran, Kai ; Sengupta, Biswa ; Bharath, Anil A.: Generative Adversarial Networks: An Overview. In: *IEEE Signal Processing Magazine* 35 (2018), Nr. 1, S. 53–65. http://dx.doi.org/10.1109/MSP.2017.2765202. – DOI 10.1109/MSP.2017.2765202. – ISSN 1053–5888

[Haz21] Hazem Abdelmotaal, Ahmed A. Abdou, Ahmed F. Omar, Dalia Mohamed El-Sebaity, Khaled Abdelazeem: Pix2pix Conditional Generative Adversarial Networks forScheimpflug Camera Color-Coded Corneal TomographyImage Generation. (2021)

[PJTA] Phillip Isola ; Jun-Yan Zhu ; Tinghui Zhou ; Alexei A. Efros: Image-to-Image Translation with Conditional Adversarial Networks

[ZGQZ19] Zhu, Miao M. ; Gong, Shengrong ; Qian, Zhenjiang ; Zhang, Lifeng: A Brief Review on Cycle Generative Adversarial Networks. In: *Proceedings of The 7th International Conference on Intelligent Systems and Image Processing 2019*, The Institute of Industrial Application Engineers, 2019. – ISBN 9784907220198, S. 235–242

[ZPIE] Zhu, Jun-Yan ; Park, Taesung ; Isola, Phillip ; Efros, Alexei A.: *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*

Abbildungsverzeichnis

Tabellenverzeichnis

Code-Auszugs-Verzeichnis

Glossar

- **C++**:
  Hardwarenahe, objektorientierte Programmiersprache.
- **HTML**:
  Hypertext Markup Language - textbasierte Auszeichnungssprache zur Strukturierung elektronischer Dokumente.
- **HTTP**:
  Hypertext Transfer Protocol - Protokoll zur Übertragung von Daten auf der Anwendungssicht über ein Rechnernetz.
- **iARS**:
  innovative Audio Response System - System mit zwei Applikationen (iARS-master-App; iARS-student-App), dass sich zum Einsetzten von e-TR-ainer-Inhalten in Vorlesungen eignet.
- **ISO**:
  Internationale Vereinigung von Normungsorganisationen.
- **JavaScript**:
  Skriptsprache zu Auswertung von Benutzerinteraktionen.
- **Konstruktor**:
  Beim Erzeugen einer Objektinstanz aufgerufene Methode zum Initialisieren von Eigenschaften.
- **MySQL**:
  Relationales Datenbankverwaltungssystem.
- **OLAT**:
  Online Learning and Training - Lernplattform für verschiedene Formen von webbasiertem Lernen.
- **OOP**:
  Objektorientierte Programmierung - Programmierparadigma, nach dem sich die Architektur eine Software an realen Objekten orientiert.
- **Open Source**:
  Software, die öffentlich von Dritten eingesehen, geändert und genutzt werden kann.
- **PHP**:
  Skriptsprache zur Erstellung von Webanwendungen.
- **Python**:
  Skript- und Programmiersprache, die unter Anderem objektorientiertes Programmieren ermöglicht.

- **Shell**:
  Shell oder auch Unix-Shell - traditionelle Benutzerschnittstelle von Unix-Betriebssystemen

- **Spyder**:
  Entwicklungsumgebung für wissenschaftliche Programmierung in der Programmiersprache Python.

- **SymPy**:
  Python-Bibliothek für symbolische Mathematik.