Bachelorarbeit Medientechnologie

Automatische Kolorierung von Grauwertbildern mithilfe von generativen Algorithmen des maschinellen Lernens und tiefer neuronaler Netze

vorgelegt von

Tobias Vossen

Matrikelnummer: 11098542

Erstgutachter: Prof. Dr. Lothar Thieling (TH Köln)

Zweitgutachter: Prof. Dr. Rainer Bartz (TH Köln)

Februar 2020

Fakultät für
Informations-, Medien-
und Elektrotechnik

Technology
Arts Sciences
TH Köln

Bachelorarbeit

Titel: Automatische Kolorierung von Grauwertbildern mithilfe von generativen Algorithmen des maschinellen Lernens und tiefer neuronaler Netze

Gutachter:

- Prof. Dr. Lothar Thieling (TH Köln)
- Prof. Dr. Rainer Bartz (TH Köln)

Zusammenfassung: Generative Modelle erschaffen – im Vergleich zu objekterkennenden Modellen – neue Inhalte, anstatt lediglich zwischen bekannten Instanzen zu differenzieren. Dadurch ergeben sich neuartige Möglichkeiten der Datensynthese, die überall dort eingesetzt werden, wo es nicht zielführend oder gar unmöglich ist, unbekannte Inhalte vorherzusagen. Die automatische Kolorierung von Grauwertbildern ist ein Paradebeispiel für den Einsatz generativer Modelle, wie das eines GAN (engl. Generative Adversarial Network), welches im Zuge dieser Arbeit eine farblose Vorlage mit plausiblen Farbinformationen versieht. Dabei kommen tiefe faltende neuronale Netze zum Einsatz, die mithilfe maschinellen Lernens die Kolorierung als Pixel-zu-Pixel-Transformation erlernen. Daraus resultierend lassen sich bspw. historische Schwarz-Weiß-Aufnahmen durch eine passende Farbrestauration aufwerten.

Stichwörter: Automatische Kolorierung, Generative Modelle, Maschinelles Lernen, Tiefe neuronale Netze

Datum: 3. Februar 2020

Bachelor thesis

Title: Automatic colorization of gray-value images based on generative algorithms of machine learning and deep neural networks

Reviewers:

- Prof. Dr. Lothar Thieling (TH Köln)
- Prof. Dr. Rainer Bartz (TH Köln)

Abstract: Generative models create - in contrast to object recognizing models – new content instead of just differentiating between familiar instances. This reveals novel possibilities of data synthesis, which are required whenever it is not useful or even impossible to predict unknown content. The automatic colorization of gray-value images is a prime example for the use of generative models, like a GAN (Generative Adversarial Network), which assigns as part of this work plausible color information to colorless patterns. Deep convolutional neural networks are used, which handle the colorization as a pixel-to-pixel transformation through machine learning. As a result, historical black and white images can be enhanced by applying a suitable color restoration.

Keywords: Automatic colorization, Generative models, Machine Learning, Deep neural networks

Date: 3 February 2020

Inhaltsverzeichnis

1 Einleitung

Es ist das Leitthema des vergangenen Wissenschaftsjahres 2019 [1], einer Initiative des Bundesministeriums für Bildung und Forschung: die künstliche Intelligenz (KI). Als informativer Austausch zwischen Öffentlichkeit und Wissenschaft zu aktuellen komplexen Forschungsfeldern soll das jährlich vergebene Wissenschaftsjahr diese Komplexität aufbrechen und für die Allgemeinheit transparent aufbereiten. Weiterhin liefert es einen entsprechenden Aufhänger für die nun folgende Bachelorarbeit über spezielle Anwendungsfälle dieser neuen Technologie und der Herangehensweise an eine Problemstellung, welche erst über die künstliche Intelligenz und weiterführende Konzepte, wie das maschinelle Lernen, ermöglicht wird.

Jene Konzepte sind aktuell gefragter denn je, denn während sich die Komplexität integrierter Schaltkreise und damit die prozessorbasierte Rechen- und Steuerleistung eines jeden Computers, nicht mehr jährlich verdoppelt, wie es seit 1965 über das moore'sche Gesetz [2] modelliert wird, lässt sich ein ähnlicher aber fortwährender Trend zu Forschungsarbeiten in der künstlichen Intelligenz beobachten. Als Leiter von *Google Brain*, dessen Abteilung jene Forschungsfelder bei dem Technologieriesen *Alphabet Inc.* vorantreibt, untermauert *Jeff Dean* mit dem folgenden Wachstumsdiagramm, die heutige Relevanz des maschinellen Lernens [3]. Denn der Trend aller themenbezogenen Veröffentlichungen auf dem Publikations-Archiv *arXiv*, übersteigt jene Verdopplungsrate des moore'schen Gesetzes, welche gerne für die Einschätzung aufkommender Forschungsfelder vergleichend herangezogen wird.

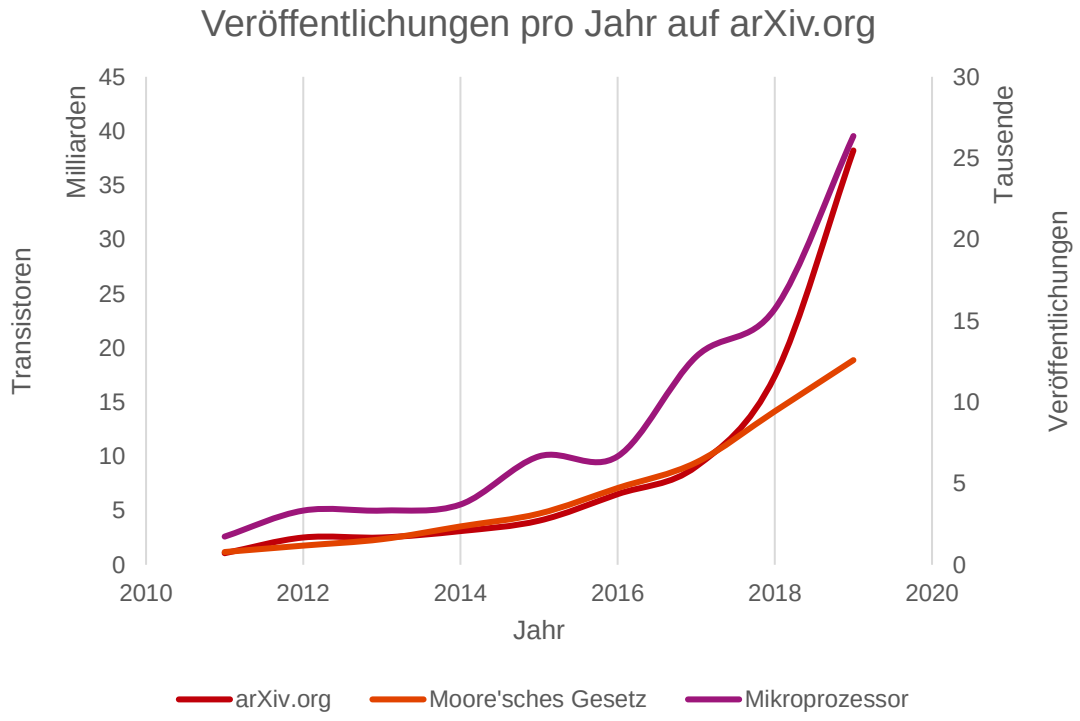


Abbildung 1: Themenbezogene Veröffentlichungen pro Jahr auf dem Publikations-Archiv arXiv [4]. Vergleichend das Moore'sche Gesetz bzw. die reale Anzahl an Transistoren pro Mikroprozessor [2]. Für [4] wurden folgende Kategorien ausgewertet: *cs.CV*, *cs.CL*, *cs.LG*, *cs.AI*, *cs.NE*, *stat.ML* (Stand: 31.01.2020)

Der Einfluss dieser Entwicklung ist längst spürbar, denn obwohl Begriffe der künstlichen Intelligenz, wie bspw. das maschinelle Lernen oder die tiefen neuronalen Netze, der breiten Masse weitestgehend fremd sind, finden ihre Konzepte bereits Einsatz in einer Vielzahl an Produkten, Dienstleistungen und Anwendungen der heutigen Zeit. Von der Spracherkennung in digitalen persönlichen Assistenten, über Bilderkennung in der Medizintechnik, bis hin zur Texterkennung in Suchmaschinen und automatischen Übersetzern, nichts bleibt mehr unberührt vom Einfluss intelligenter Systeme. Jene Dauerpräsenz in aktueller Forschungsliteratur erschien Grund genug, sich mit der KI als Lösungswerkzeug einer Aufgabenstellung zu beschäftigen, welche im Folgenden erstmals angerissen wird.

Das klassische Beispiel aus der bildverarbeitenden Maschinenintelligenz, dem sog. Computersehen (engl. Computer Vision, CV) ist ein diskriminierendes System, welches auf Basis der künstlichen Intelligenz mithilfe von maschinellem Lernen, eine Katze von einem Hund, zu unterscheiden weiß. Wenn das System also im Bilde darüber ist, wie eine Katze, bzw. ein Hund auszusehen hat, kann es dann nicht auch das Abbild einer Katze, in das eines Hundes verwandeln? Hier kommen generative Systeme ins Spiel, welche im Vergleich zum vorherigen Beispiel keine Unterscheidung treffen, sondern vielmehr neue Inhalte erschaffen, die auf den menschlichen Beobachter plausibel wirken.

Dabei werden mitunter spannende Anwendungen geboren, wie die Fertigstellung von unvollständigen Symphonien aus der historischen Klassik [5]. Die KI übernimmt hier die Arbeit des Komponisten und vollendet sein Werk autonom, aber unter Revision eines menschlichen Experten. Zugleich helfen generative Systeme der *FaceApp* [6] dabei, auf mobilen Endgeräten das eigene Aussehen zu verändern, Alterserscheinungen zu simulieren, oder den Gesichtsausdruck zu verfälschen (Abbildung 2). Neben klassischer Musik und dem eigenen Erscheinungsbild gibt es weitere Datengrundlagen, welche für die Verwendung in generativen Systemen in Frage kommen. Schwarz-Weiß-Aufnahmen aus früheren Zeiten, deren Aufbereitung durch einen Künstler einen beträchtlichen Aufwand nach sich ziehen, lassen sich ebenfalls mithilfe künstlicher Intelligenz kolorieren. Nicht wenige haben sich dieser Problemstellung bereits angenommen und selbst der erwähnte Technologieriese bewirbt seine quelloffene Softwarelösung für maschinelles Lernen mit einem solchen Anwendungsfall [7]. Auch im künstlerischen Bereich hält diese Technologie Einzug: ob im Falle der Animation von Cartoons und Animes oder der Restaurierung verwaschener Farbaufnahmen.

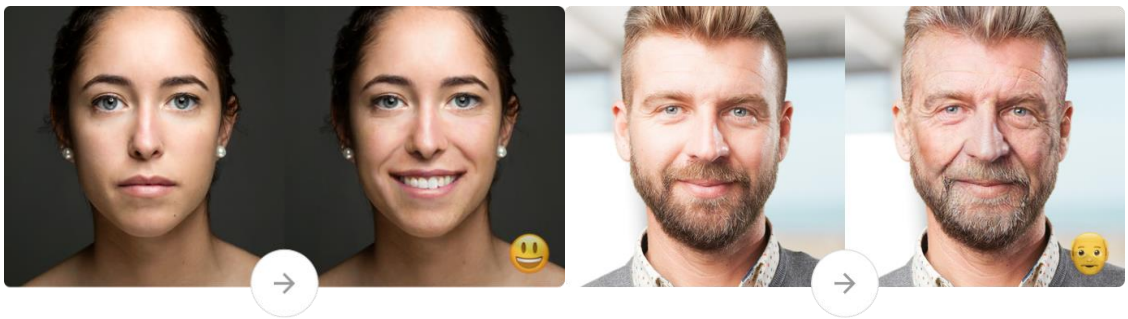


Abbildung 2: Verfälschung des Gesichtsausdruckes (li.) und Simulation von Alterserscheinungen (re.) mithilfe der FaceApp [6]

Dabei stellen sich eine Vielzahl an Fragen in Bezug auf die Umsetzbarkeit der angestrebten Kolorierung. Wie lässt sich generell ein farbloses Ausgangsbild über Methoden des maschi-

nellen Lernens mit Farbe füllen? Welchen Vorteil haben generative Systeme für jene Kolorierung im Vergleich zu bekannten diskriminierenden Systemen? Und wie lässt sich deren Leistung quantifizieren? Wie plausibel ist jene Kolorierung für den menschlichen Betrachter bzw. wie lässt sie sich qualitativ bewerten? Welche Rolle spielt die Modellierung der Farbe in diesem System? All diese Fragen dienen der nun folgenden Arbeit als Anreiz, ein System zu entwerfen, dessen Ziel die Kolorierung eines Grauwertbildes ist.

Jenes System versucht sich also im Rahmen dieses Projektziels an der Abbildung monochromer auf polychrome Inhalte. Auf ein farbloses Eingangsbild soll ein farbiges Ausgangsbild folgen. Das System versucht sich zudem an der Erzeugung plausibler Inhalte. Ein Ausgangsbild soll auf Basis geometrischer, wahrnehmbarer und semantischer Aspekte möglichst fotorealistisch erscheinen. Als Lösungsansatz wird das System als eine Bild-zu-Bild-Abbildung modelliert, dessen Ein- und Ausgänge in Bildform vorliegen. Das System soll zusätzlich zu dieser abbildenden Struktur eine Kontrollstruktur aufweisen, um eine möglichst realistische Kolorierung zu gewährleisten.

Nach der Definition von Projektziel und Lösungsansatz folgt die Strukturierung der Projektarbeit. Die folgende Projektplanung beinhaltet dabei chronologische Schritte zur Bearbeitung der Problemstellung, welche jeweils ein Kapitel der Arbeit abdecken (Abbildung 3): die ersten beiden Kapitel dienen dabei dem einleitenden und grundlegenden Verständnis der Projektarbeit und sind daher von theoretischer Natur. Kapitel drei leitet den praktischen Teil dieser Arbeit ein, welcher sich mit dem Lösungskonzept sowie dessen konkreter Implementierung im darauffolgenden Kapitel vier beschäftigt. Kapitel fünf präsentiert die praktischen Resultate dieser Arbeit, welche diskutiert und evaluiert werden, sodass im letzten Kapitel die Projektarbeit zusammenfassend abgerundet werden kann.

Nach der allgemeinen Planung erfordert der Einsatz künstlicher Intelligenz spezielle Arbeitsschritte, die es zu konkretisieren gilt und dessen Inhalte speziell auf die Anforderungen des maschinellen Lernens zugeschnitten sind. Man bewegt sich in den drei Domänen: *Daten*, *Modell* und *Verwendung*, welche den kommenden Kapiteln als kontextbezogene Unterüberschriften dienen. Bspw. erfordern Daten sowohl konzeptionelle als auch implementierende Schritte, während das Modell ebenso implementiert wie evaluiert werden will. Doch bevor jene Schritte eingeläutet werden, betrachtet das folgende Kapitel theoretische Grundlagen, welche für die darauffolgende Praxis vonnöten sind.

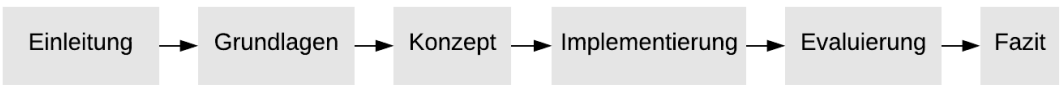


Abbildung 3: Projektplanung

2 Grundlagen

Eine Problemstellung, die sich an der Vorhersage von Farben versucht, tut gut daran, sich zu allererst mit ihrer Darstellung zu beschäftigen. An dieser Stelle kommen Farbräume ins Spiel, die als Abstraktion eine Farbe beschreiben, um sie zwischen Mensch und Maschine vergleichbar zu machen. Der erste Abschnitt dieser Grundlagenbetrachtung befasst sich daher mit drei gängigen Farbräumen der Bildverarbeitung.

Nach der Betrachtung verschiedener Farbsysteme steht die Frage im Raum, inwiefern diese automatisiert vorhergesagt werden können. An dieser Stelle folgt eine Einführung in die künstliche und maschinelle Intelligenz einschließlich tiefer neuronaler Netze. Darauf aufbauend werden Implementierungen dieser Konzepte erläutert, die als generative Modelle die Eigenschaft besitzen, nicht nur zu erkennen, sondern auch zu erschaffen. Abschließend werden bereits funktionierende Systeme automatischer Kolorierung vorgestellt, die als Basis für das Konzept, sowie die Implementierung der Problemlösung dienen.

2.1 Farbräume

Eine Farbempfindung lässt sich auf unterschiedlichste Weise modellieren. Die spektralen Eigenschaften des Auges mitsamt Empfindlichkeitskurven ergeben ein fast kontinuierliches, auf kleinsten Wellenlängenbereichen diskret abgetastetes Empfängersignal. Dabei können Wellen von unterschiedlicher spektraler Zusammensetzung dieselbe Farbvalenz besitzen und damit eine identische Farbwahrnehmung hervorrufen. Ein numerischer Vergleich dieser Farbsignale wäre nicht zielführend und so stellen Farbräume eine Vereinfachung dar, die auf Basis der trichromatischen Theorie, eine Farbe anhand von lediglich drei Parametern beschreibt. Der gängigste Farbraum ist dabei über die drei Typen vorhandener Rezeptoren im menschlichen Auge abgeleitet, er besteht aus den bekannten Primärfarben Rot, Grün und Blau [8, 2.1 - 2.2].

2.1.1 RGB

Der RGB-Farbraum bedient sich der drei Farbkanäle Rot, Grün und Blau, die als Basisvektoren (Achsen) den Farbraum anschaulich als Quader aufspannen (Abbildung 4). Die Gerade, für die alle drei Farbkanäle denselben Wert aufweisen, umfasst alle Grautöne von Schwarz bis Weiß und wird als Unbuntgerade bezeichnet. Üblicherweise liegen Rastergrafiken als Matrizen im RGB-Farbraum vor, sodass ein digitalisiertes Bild sich additiv aus den drei Farbkanälen zusammensetzt. Dieser Farbraum dient daher als übliche Repräsentationsform von Farbbildern in bildverarbeitenden Systemen. Leider fehlt ihm eine offensichtliche Trennung von Luminanz- und Chrominanzanteil eines Bildinhaltes, was vielmehr durch die alternative Repräsentation im folgenden Farbraum erreicht wird [8, 7].

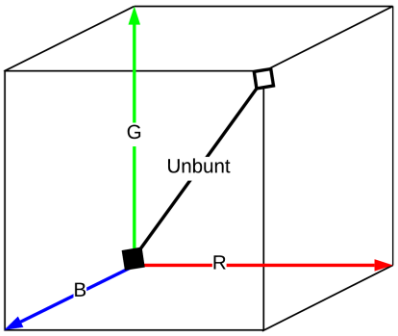


Abbildung 4: RGB-Farbraum als Quader

2.1.2 HSV

Der HSV-Farbraum verkörpert mithilfe seiner drei Farbkanäle Farbton (engl. Hue, H), Sättigung (engl. Saturation, S) und Helligkeit (engl. Value, V) den Farbraum anschaulich als Zylinder (Abbildung 5). Dabei wird ersterer als Farbtonwinkel in Radiant angegeben, auf dessen Radius die zweite Komponente als vollgesättigte Farbe liegt. Ein RGB-Farbtripel lässt sich über eine lineare Transformation in den HSV-Farbraum umwandeln, was die Trennung von Luminanz- und Chrominanzanteil des Inhaltes vereinfacht. Ersterer wird allein über den Helligkeitskanal repräsentiert, ohne von Änderungen des Farbtons bzw. der Sättigung beeinflusst zu werden. Da die Farbwahrnehmung des menschlichen visuellen Systems jedoch nichtlinear abläuft, wurde jene lineare Transformation des HSV-Farbraumes über die folgende Modellierung präzisiert [8, 9.1].

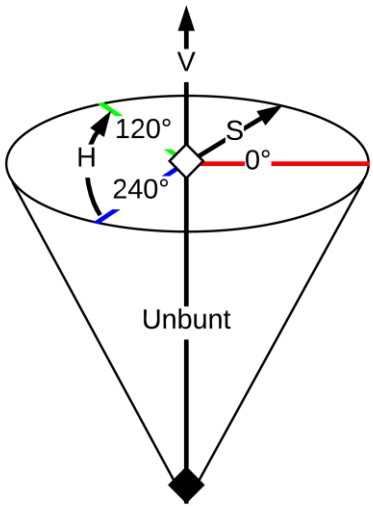


Abbildung 5: HSV-Farbraum als Zylinder

2.1.3 Lab

Der Lab-Farbraum verkörpert mithilfe der Helligkeit L* (engl. Lightness, L) und den zwei Farbebenen a* (Rot- und Grünanteil) und b* (Gelb- und Blauanteil) den Farbraum anschaulich als Kugel (Abbildung 6). Während die erste Komponente als Unbuntgerade die Luminanz eines Inhaltes abdeckt, definieren die Farbebenen seine Chrominanz. Dieser Far-

braum bedient sich im Vergleich zu den linearen RGB- bzw. HSV-Farbdefinitionen einer logarithmischen Modellierung basierend auf der nichtlinearen Reizempfindung des visuellen Systems. Eine Farbraumtransformation ist daher nichtlinear und abhängig vom Weißpunkt des verarbeitenden Systems. Der Vorteil des LAB-Farbraumes liegt in seiner realitätsnahen Darstellung einer Farbe. Während vorherige lineare Farbräume keine wahrnehmungsbezogenen Größen einbeziehen, basieren Berechnungen innerhalb des Lab-Farbraumes auf der visuellen Wahrnehmung menschlicher Testpersonen. Dies bringt im Vergleich zu theoretischen mathematischen Modellen, wie im Falle des HSV-Zylinders, einen praxisnahen Vorteil mit sich. Abweichungen zweier unterschiedlicher Farben lassen sich daher über den Lab-Farbraum wahrnehmungsbezogen berechnen, wohingegen der reine Abstand zweier RGB-Farbvektoren praktisch kaum aussagekräftig ist [8, 5.5].

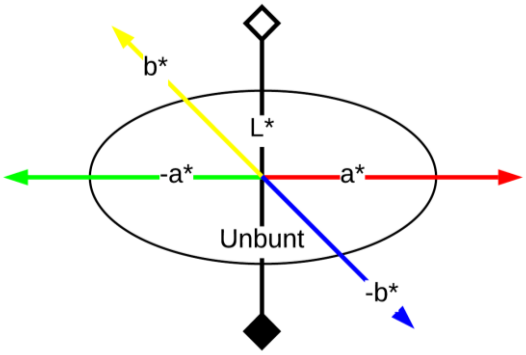


Abbildung 6: Lab-Farbraum als Kugel

2.2 Von künstlicher Intelligenz bis neuronale Netze

Nach der Modellierung des vorherzusagenden Inhaltes, also der Farbe, stellt sich die Frage nach einer Disziplin, mithilfe derer jene Vorhersage getätigt werden kann. An dieser Stelle erhält die künstliche Intelligenz Einzug, deren Technologieaufschwung neue Begrifflichkeiten mit sich bringt, welche jeweils eine Vielzahl an interdisziplinären und speziellen Methoden beinhalten. Man unterscheidet zunächst zwischen der KI, dem maschinellen Lernen, sowie dem tiefen Lernen. Dabei stellt letzteres ein Teilgebiet des maschinellen Lernens dar, während dieses sich selbst als Teilgebiet der KI wiederfindet, wie das folgende Kreisdiagramm verdeutlicht. Die künstliche Intelligenz schließt als Überbegriff diese weiterführenden Fachbereiche ein und wird im folgenden Abschnitt vorgestellt.

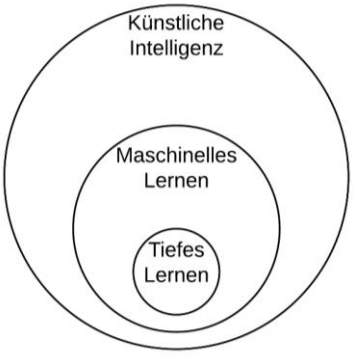


Abbildung 7: Künstliche Intelligenz als Überbegriff in Anlehnung an [9, Abb. 1.1]

2.2.1 Künstliche Intelligenz

Die künstliche Intelligenz erlebte ihre Geburtsstunde auf einer Fachkonferenz im Jahre 1956, dessen Antrag [10] den Begriff *Artificial Intelligence* enthielt. Als Definition dieses Fachgebietes lässt sich auf das renommierte *Oxford Dictionary* zurückgreifen, welches die KI als die Theorie und Entwicklung von Computer-Systemen sieht, um jene Aufgaben zu lösen, die normalerweise einer menschliche Intelligenz vorbehalten sind [11]. Damit verfolgt die KI den Versuch, einer Maschine das Denken näherzubringen, was sie über zwei konkurrierende Methoden, zu erreichen versucht: die symbolische und neuronale KI. Während erstere versucht die Verarbeitung von Daten über eine Definition von Regeln (bzw. Symbolen) zu erreichen und darauf basierend, intelligente Entscheidungen zu treffen, verfolgt die neuronale KI einen entgegengesetzten Ansatz, indem sie diese Regeln selbst zu bestimmen versucht. Während angewandte Regeln auf Daten in der symbolischen KI eine Antwort hervorbringen, soll die neuronale KI aus gegebenen Daten und bekannten Antworten Regeln erlernen, woraus sich das weiterführende Konzept des maschinellen Lernens entwickelte [9, 1.1.1]

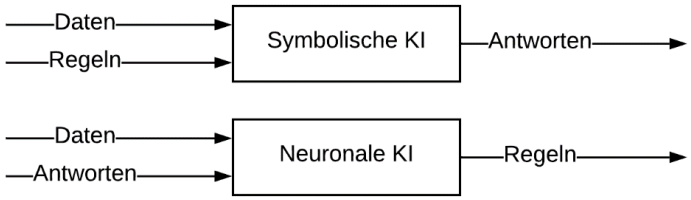


Abbildung 8: Symbolische und neuronale KI in Anlehnung an [9, Abb. 1.2]

2.2.2 Maschinelles Lernen

„Machine Learning is using data to answer questions“ (Y. Guo) [12]

Maschinelles Lernen lässt sich anhand dieses Zitates anreißen. Daten bestehend aus bekannten Frage-Antwort-Pärchen lassen Regeln erkennen, die letztendlich zum Beantworten neuer unbekannter Fragen verwendet werden. Das Erkennen dieser Regeln wird mit einem Trainings- bzw. Lernprozess verglichen, wonach das maschinelle Lernen bzw. die neuronale KI ihre Namensgebung hat. Denn nichts anderes passiert im menschlichen Gehirn bestehend aus Nervenzellen bzw. Neuronen: es lernt aus Bekanntem und Erfahrungen. Dieser Abstraktion bedient sich das maschinelle Lernen, welches im Grundsatz mithilfe von *Daten* ein *Modell* über ein *Training* optimiert, das für eine *Vorhersage* verwendet wird, wie die folgende Abbildung verdeutlicht [9, 1.1.2]:

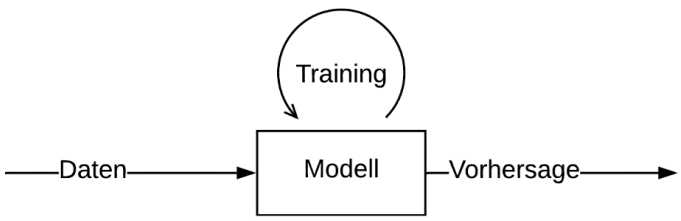


Abbildung 9: Maschinelles Lernen

Jene (Eingabe)Daten benötigen gezwungenermaßen eine passende Ausgabe, deren Beziehung zueinander durch das System erlernt werden soll. Ein simples Beispiel stellt die Umrechnung der Temperatur von Celsius T_C in Fahrenheit T_F dar. Dabei verkörpert die Funktion f die Beziehung zwischen Eingangs- und Ausgangswert, welche vom System erkannt werden soll, sodass letztendlich eine Vorhersage der Temperatur in Fahrenheit $T_F = f(T_C)$ getroffen werden kann.

In diesem Fall ist die Beziehung linear und auch die Ein- bzw. Ausgabe ein eindimensionaler Wert. Was passiert nun bei Bildern von zweidimensionaler Struktur, die im Falle einer Klassifizierung in eindimensionale Ausgangsdaten überführt werden sollen, die letztendlich eine binäre Entscheidung über das zu erkennende Objekt treffen? Eine lineare Beziehung ist hier nicht mehr gegeben und es wird versucht eine Vereinfachung zu finden, welche der Eingabe die Komplexität nimmt und es somit dem Modell erleichtert, die passende Ausgabe zu erlernen. Diese Vereinfachung wird als *Repräsentation* bezeichnet [9, 1.1.3].

Ein simples Beispiel für eine veränderte Repräsentation basiert auf einem System, das ein farbiges in ein Grauwertbild transformiert und dessen Eingangsdaten, also das Farbbild, im additiven RGB-Farbraum vorliegen (siehe 2.1.1). Um ein farbloses Bild zu erzeugen, wäre nun eine aufwändige Berechnung vonnöten, die sich durch eine alternative Repräsentation der Eingangsdaten im Lab- anstatt RGB-Farbraum, verhindert lässt. Das geforderte Grauwertbild lässt sich alternativ über die erste Komponente des dreidimensionalen Farbvektors extrahieren. Die veränderte Repräsentation der Eingangsdaten, die soeben manuell gewählt wurde, wird mithilfe von maschinellem Lernen automatisch gesucht und gefunden.

Im vorausgegangenen Beispiel ist mit der Änderung des Farbraumes lediglich eine passende Repräsentation vonnöten, um die Aufgabe zu lösen. Was passiert nun bei komplexeren Problemen, wie der bereits erwähnte Klassifizierung zwischen Hund und Katze, welche nicht mehr anhand einer einzelnen Repräsentation vereinfacht werden kann?

2.2.3 Tiefes Lernen

Im Vergleich zum bereits bekannten maschinellen Lernen, welches sich mit einer simplen Repräsentation zufriedengibt, bedient sich das tiefe Lernen (engl. Deep Learning, DL) dem Konzept der Kaskadierung. Eine einzelne Repräsentation ist hier nicht genug und man versucht mit zunehmender Anzahl an Repräsentationen, die Komplexität der Eingabe zu verringern. Ein Bildinhalt lässt sich bspw. neben der bereits behandelten Farbe anhand von Kanten, Ecken und Formen beschreiben. Eine einzelne Kante ist dabei kaum aussagekräftig, zwei orthogonale Kanten ergeben aber bereits eine Ecke und mehrere Ecken bilden letztendlich eine Form, die bspw. ein Objekt umreißt, das es zu klassifizieren gilt. Dabei verringert sich die Komplexität des Bildinhaltes mit jeder weiteren Repräsentationsstufe.

An dieser Stelle werden die jeweiligen Repräsentationsstufen als einzelne Schichten in einem Netzwerk zusammengefasst, was als neuronales Netz (engl. Neural Network, NN) bezeichnet wird. Dieses Modell setzt grundlegend das vorgestellte Konzept für tiefes Lernen um, indem es die Eingabe sequenziell durch verschiedene Schichten immer tiefer in das Netz schickt, bis diese eine hinreichend repräsentative Form angenommen hat, um sie in eine Ausgabe zu transferieren. Dabei bleibt im besten Fall der Informationsgehalt – analog zur verlustlosen Datenkompression – erhalten, nur die Komplexität sinkt [9, 1.1.4].

Nach Definition eines Modells wird nach Abbildung 9 noch das Training behandelt, aus dem die Schichten lernen, wie sie zu operieren haben. Hierzu besteht eine jede Schicht aus *Transformationen*, welche den Eingang einer Schicht, analog zu einer mathematischen Funktion, verarbeiten und verändern. Das Kaskadieren vieler Transformationen (bzw. Abbildungen) ergibt nun die Struktur einer Schicht und charakterisiert ihre Repräsentation. Über *Parameter* (bzw. Variablen) lässt sich eine jede Transformation steuern, was sich vom Trainingsprozess während der *Optimierung* des Netzes zu Nutze gemacht wird. Dieser strebt eine fehlerfreie Ausgabe des Netzes an und optimiert vorhandene Parameter dahingehend, dass der erzeugte Fehler minimal wird. Da ein Fehler immer auch numerisch dargestellt werden kann, lässt sich dieser über die Abweichung zwischen Soll- und Ist-Ausgabe quantifizieren. Im vorausgegangenen Beispiel der Temperatur-Umrechnung lässt sich diese Abweichung zwischen realer und vorhergesagter Temperatur als *Verlustfunktion* modellieren, um das Netz zu einer fehlerfreien Vorhersage zu bringen. Verringert sich die Verlustfunktion, trägt das Training Früchte und die Optimierung der Parameter verläuft in Zielrichtung. Vergrößert sich die Verlustfunktion, ist es umgekehrt und es bedarf einer Kehrtwende der Optimierung. Da sich bei einer sequenziell verbundenen Netzstruktur, die Anpassung eines Neurons als kleinster Knotenpunkt des Netzes auf alle anderen Neuronen auswirken kann, geschieht die Optimierung der Parameter rekursiv und rückläufig vom Netzaus- zum Netzeingang über die Fehlerrückführung (engl. Backpropagation), wie Abbildung 10 verdeutlicht [9, 1.1.5]. Nachdem vorausgehende Abschnitte die Entstehung des tiefen Lernens aus den Überbegriffen des maschinellen Lernens bzw. der KI abgeleitet haben, beschäftigt sich der kommende Abschnitt mit der konkreten Umsetzung jener Methoden.

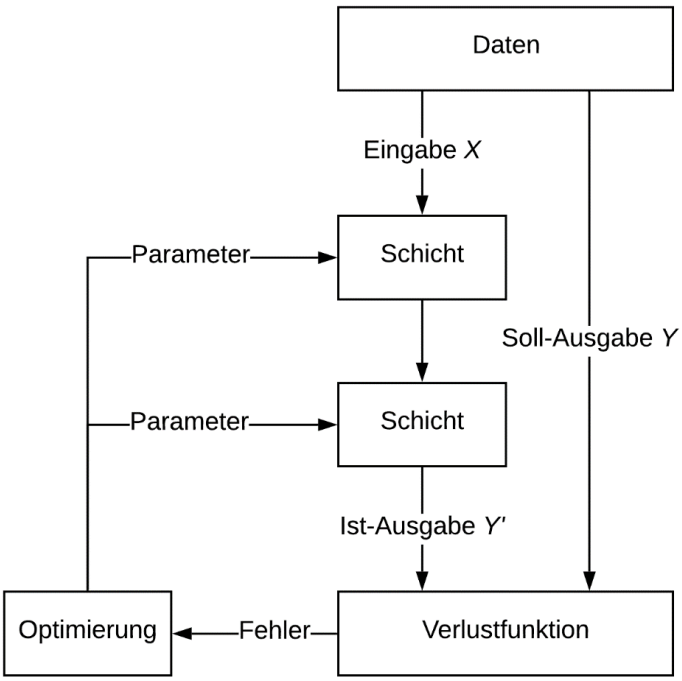


Abbildung 10: Fehlerrückführung in Anlehnung an [9, Abb. 1.9]

2.3 Generative Modelle

Mithilfe der bereits erläuterten neuronalen Netze lassen sich diskriminierende sowie generative Modelle umsetzen. Ein diskriminierendes Modell differenziert dabei bereits existierende Inhalte, während ein generatives Modell eine neue Instanz dieser Inhalte erzeugt. Mathematisch ausgedrückt modelliert ersteres die bedingte Wahrscheinlichkeit $p(Y|X)$, also die Wahrscheinlichkeit, dass Y von X abhängt, während sein generatives Gegenstück die Verbundswahrscheinlichkeit $p(X,Y)$ der beiden Variablen betrachtet. Die Vorhersage eines diskriminierenden Modells beschränkt sich daher lediglich auf die Wahrscheinlichkeit, dass es sich bei einer bekannten Instanz um ein passendes Objekt handelt, wohingegen die generative Vorhersage eine neue Instanz nach Vorlage des Objektes erzeugt [13]. Die folgende Abbildung überträgt diese theoretische Betrachtung auf das allgegenwärtige Beispiel aus der CV, wonach das diskriminierende Modell die Schätzung trifft, ob das Bild X zum Bildinhalt Y passt, während sein generierendes Gegenstück ein neues Bild X' vorhersagt.

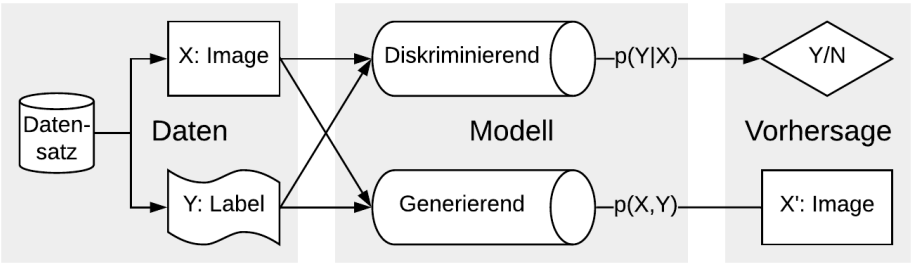


Abbildung 11: Modelle mit diskriminierender und generierender Funktion

Generative Modelle erlernen dabei die Struktur des Datensatzes, indem ihre Wahrscheinlichkeit maximiert wird, seiner Datenverteilung zu folgen, wohingegen diskriminierende Gegenstücke lediglich Daten klassifizieren und nicht imitieren. Um letztendlich die Komplexität von Farbinformationen zu modellieren, ist ein vereintes Modell vonnöten, dessen Kern auf einem generativen Netzwerk basiert, das gleichzeitig auf die Hilfe eines diskriminierenden Gegenspielers angewiesen ist. Es nennt sich GAN (engl. Generative Adversarial Network) [14] und wird im kommenden Abschnitt vorgestellt.

2.3.1 GAN

Ein GAN besteht in erster Linie aus zwei Subnetzen: einem *Diskriminator* D und einem namensgebenden *Generator* G, die eine gegensätzliche Intention verfolgen. Dabei versucht der Diskriminator die reale Datenverteilung von der synthetischen, vom Generator stammenden, zu unterscheiden [14, 1]. Es sind also zwei Grundlagen im Spiel, die reale Datenverteilung p_{data}, welche natürliche Daten x enthält, die es zu imitieren gilt, sowie die imaginäre Datenverteilung p_{noise}, die es als synthetisch erzeugte Daten z zu entlarven gilt, wie die folgende Übersicht verdeutlicht:

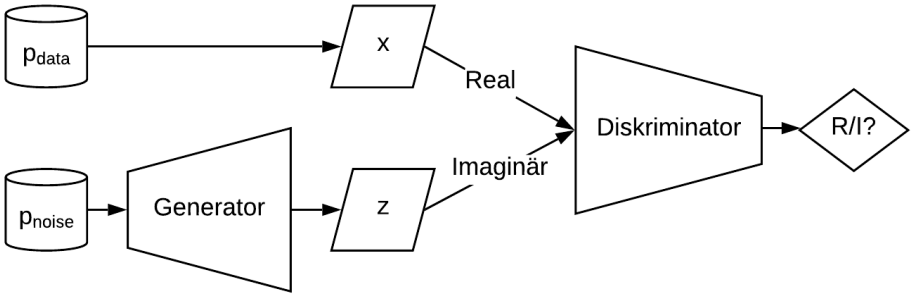


Abbildung 12: Generative Adversarial Network in Anlehnung an [15, Abb. 2]

Für ein besseres Verständnis der Konstellation lässt sich die Original-gegen-Fälschung-Problematik heranziehen. Während der Fälscher (als Generator) ein Objekt nachahmt und damit eine gefälschte Kopie (aus der imaginären Datenverteilung) in Umlauf bringt, muss der Experte (Diskriminator) diese Kopie von dem Original (aus der realen Datenverteilung) zu unterscheiden wissen. Während die Intention des Fälschers darin liegt, den Experten möglichst häufig zu täuschen, versucht dieser, möglichst erfolgreich die Werke des Fälschers zu entlarven. Dieser Gegensatz lässt sich über das Min-Max-Theorem als Nullsummen-Spiel modellieren, was zur folgenden mathematischen Grundlage eines GANs führt.

(1) $\max_D \mathbb{E}_{x\sim p_{data}}[\log \mathrm{D(x)}]$

(2) $\min_G \mathbb{E}_{\mathrm{z}\sim p_{noise}}\left[\log(1 - D\left(G(z)\right)\right]$

Dabei versucht der Diskriminator seine Erwartung $\mathbb{E}$ zu maximieren (Klassifikation als Original), sobald die Eingabe aus der realen Datenverteilung stammt (1). Gleichzeitig versucht er seine Erwartung zu minimieren (Klassifikation als Kopie), sobald die Eingabe aus dem Generator stammt, während jener die Erwartung des Diskriminators in diesem Fall umzudrehen, also zu maximieren, vermag (2).

Die Summe beider Gleichungen ergibt nun die *Zielfunktion* (3) eines GAN [14, 3]. Es handelt sich um die binäre Kreuzentropie (engl. Binary Crossentropy) [16], welches die typische Zielfunktion einer binären Klassifizierung verkörpert. Man spricht in diesem Fall bewusst nicht von einer Verlustfunktion, da die binäre Kreuzentropie keine Abweichung kalkuliert, wie es bspw. das Abweichungsquadrat tätigt. Bezogen auf die Optimierung des Netzes verkörpern Ziel- und Verlustfunktion jedoch dasselbe.

(3) $\min_G \max_D \mathcal{L}_{GAN}(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x)] + \mathbb{E}_{z\sim p_{noise}}[\log(1 - D\left(G(z)\right))]$

Werden beide Gegenspieler als neuronale Netze implementiert, so lässt sich diese Verlustfunktion $\mathcal{L}_{GAN}$ über die Fehlerrückführung optimieren. Da die Signalkette wie in Abbildung 12 vom Generator über den Diskriminator läuft, wäre ein paralleles Training nicht zielführend. Stattdessen wird ähnlich dem Grundsatz der Fehlerrückführung zunächst der Diskriminator trainiert, woraufhin dieser eingefroren wird und die Optimierung des Generators folgt. Diese Kette wird über eine einzelne Optimierung realisiert, sodass der Generator als zweites Kettenglied über dieselbe Rückmeldung wie der Diskriminator angepasst wird. Dieser Prozess wiederholt sich iterativ in einer Schleife, bis ein Gleichgewicht entsteht.

Dieses Gleichgewicht zwischen zwei Gegenspielern – auch bekannt als Nash-Equilibrium, welches ein optimales Spielergebnis hervorruft, wird erreicht, sobald sich beide Datenverteilungen angleichen. Lässt sich die imaginäre nicht mehr von der realen Struktur unterscheiden, wird der Diskriminator im Mittel eine 50 % Erwartung hervorrufen, was seine Einschätzung zufällig erscheinen lässt. Dieses Optimum wird durch die Jensen-Shannon-Divergenz $JSD(p_{data} \| p_{noise})$ sichergestellt, die es zu minimieren gilt, was letztendlich zu einer perfekten Rekonstruktion bzw. Imitierung der realen Datenverteilung durch den Generator führt [14, 4]. Jene Divergenzen werden in der Wahrscheinlichkeitstheorie dafür genutzt, die Abweichung zwischen zwei Datenverteilungen zu betrachten.

Nach dem Trainingsprozess erfolgt eine Vorhersage des Generators letztendlich in Vorwärtsrichtung (engl. Feed forward network) ohne rückführende Signalflüsse. Jene Vorhersage greift lediglich auf die imaginäre Datenverteilung als Eingabe zurück, welche in dieser Architektur als Zufallsvariable modelliert wird, die als normalverteiltes Rauschen den Generator füttert. Um die Vorhersage nun deterministischer zu gestalten, wird dieser Eingabe im nächsten Abschnitt eine weitere Variable beigefügt.

2.3.2 cGAN

Das Hinzufügen einer Bedingung y verwandelt das GAN in ein cGAN (engl. Conditional Generative Adversarial Network) [17]. Diese Kondition entstammt im Regelfall der realen Datenverteilung und reiht sich in die Netzeingabe des Generators, sowie des Diskriminators ein, wie die folgende Abbildung verdeutlicht:

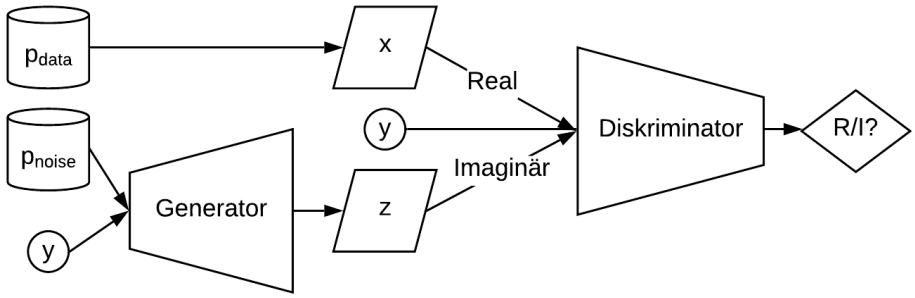


Abbildung 13: Conditional Generative Adversarial Network in Anlehnung an Abbildung 12

Dabei wird die Zielfunktion eines GAN (3) um die Bedingung erweitert (4).

(4) $\min_G \max_D \mathcal{L}_{cGAN}(D,G) = \mathbb{E}_{x\sim p_{data}}[\log D(x|y)] + \mathbb{E}_{z\sim p_{noise}}[\log(1 - D\big(G(z|y)\big))]$

Trotz zusätzlicher Bedingung beruht der Erfolg des Netzes auf zwei ausgeglichenen Subnetzen. Doch was passiert, wenn der Diskriminator zu undifferenziert agiert und die synthetisierte Kopie des Generators zu häufig als Original klassifiziert? In diesem Fall würde der Generator weder optimiert werden, noch könnte er die reale Datenverteilung adäquat abbilden. Abhilfe schafft das verstärkte Bestrafen falscher Klassifizierungen des Diskriminators.

2.3.3 LSGAN

Das Ersetzen der binären Kreuzentropie durch die quadratische Abweichung verwandelt das GAN in ein LSGAN (engl. Least Square GAN) [18]. Dabei wird das Gleichgewicht bei-

der Gegenspieler über die Pearson χ^2-Divergenz (anstatt JSD) erreicht, welche die Ist-Einschätzung des Diskriminators mit seiner Soll-Einschätzung s abgleicht. Letztere umfasst dabei lediglich die Information, ob der Eingang der realen (bzw. originalen), oder der imaginären (bzw. kopierten) Datenverteilung entstammt. Es entstehen zwei neue Verlustfunktionen, eine für den Diskriminator (5) und eine für den Generator (6).

(5) $$\min_D \mathcal{L}_{LSGAN}(D) = \mathbb{E}_{x\sim p_{data}}[(D(x) - s)^2] + \mathbb{E}_{z\sim p_{noise}}[(D(G(z)) - s)^2]$$

(6) $$\min_G \mathcal{L}_{LSGAN}(G) = \mathbb{E}_{z\sim p_{noise}}[(D(G(z)) - s)^2]$$

Neben der Einschätzung des Diskriminators ließe sich alternativ die Vorhersage des Generators sanktionieren, sollte diese wie im Falle des cGAN auf einer deterministischen Kondition als Eingabe fußen, dessen gewünschte Ausgabe bekannt ist. Nach der Vorstellung diverser Varianten eines GAN, welche in Tabelle 1 zusammengefasst sind, betrachtet der kommende Abschnitt über bereits implementierte Problemlösungen den aktuellen Stand der Wissenschaft.

Architektur	**Zielfunktion**	**Divergenz**
GAN	Binäre Kreuzentropie	JSD
cGAN	Binäre Kreuzentropie	JSD
LSGAN	Methode der kleinsten Quadrate	Pearson χ^2

Tabelle 1: Zielfunktionen unterschiedlicher Varianten eines GAN

2.4 Stand der Wissenschaft

Die vorliegende Problemstellung der intentionellen Kolorierung von Grauwertbildern lässt sich als 1-zu-2-Abbildung modellieren. Hierbei wird eine Luminanz-Komponente auf zwei Chrominanz-Komponenten abgebildet, sodass nach Vorlage bekannter Farbsysteme, wie HSV und Lab, der Zusammenschluss aller drei Anteile ein farbiges Bild ergibt, wie die folgende Abbildung verdeutlicht:

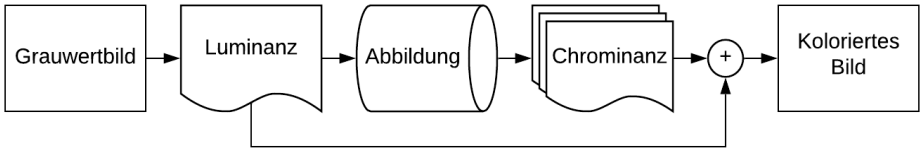


Abbildung 14: 1-zu2-Abbildung

Jene Abbildung wurde in der Vergangenheit durch unterschiedliche Systeme erreicht, welche von benutzergestützten Methoden der konventionellen Bildverarbeitung, über maschinelles Lernen, bis hin zu vollautomatischen Systemen auf Basis generativer Modelle reichen. Dieser Abschnitt befasst sich daher zum Abschluss der Grundlagenbetrachtung mit einer Fülle an Lösungsansätzen, sodass im nachfolgenden Kapitel ein passendes Lösungskonzept für die Problemstellung dieser Arbeit gefunden werden kann.

2.4.1 Halbautomatische Systeme

Jene Systeme beruhen auf benutzergestützten Hilfestellungen für die Abbildungsfunktion und agieren daher nicht vollautomatisch.

Entwurfsbasierte Methoden

Als Eingabe entwurfsbasierter Systeme dienen punktuelle Farbvorschläge zu definierten Bildbereichen. Ähnlich des Rohentwurfes eines Designers wird hier nur ein Bruchteil des Bildes vom Nutzer mit Farbe versehen, während unangetastete Bereiche über eine Klassifikation des nächsten Nachbarn (engl. Nearest neighbor algorithm) interpoliert werden [19]. Eine verbesserte Vorhersage unbekannter Farbwerte lässt sich in Kombination mit flächen- oder konturbasierter Segmentierung erreichen, deren erzeugte Objektgrenzen gleichzeitig als Übergänge zweier Farben dienen. Leider erfordern jene entwurfsbasierten Methoden einen hohen Grad an manuellem Eingriff, welchen es durch nachfolgende Lösungskonzepte zu verringern gilt.

Referenzbasierte Methoden

Als Eingabe referenzbasierter Systeme dienen natürliche Beispiele mit vergleichbarem Inhalt zum zu kolorierenden Bild. Über das Abgleichen der Luminanzverteilung eines farblosen Quell- zu der eines farbigen Zielbildes lässt sich die Farbinformation nach Vorlage der nutzerdefinierten Referenz überführen [20]. Auch dieser Ansatz bedarf einer manuellen Eingabe, auch wenn diese weniger aufwändig als noch im vorherigen Abschnitt erscheint.

2.4.2 Vollautomatische Systeme

Der nächste Schritt führte zu automatischen Systemen ohne jeden Benutzereingriff, deren Lösungskonzepte nicht selten auf Methoden des tiefen Lernens zurückgreifen. Als erste Implementierung wird von Cheng et al. [21] eine Transferfunktion erlernt, die einem Grauwertbild eine Farbkomponente hinzufügt. Diese Abbildung wird mithilfe neuronaler Netze nach der Methode der kleinsten Quadrate (engl. Least squares) optimiert, welche als Regressionsanalyse die Verteilungen von Luminanz- und Chrominanzanteil angleicht.

Überwachtes Lernen

Im Vergleich zur vorherigen Regression, nimmt Zhang et al. [22] eine Klassifikation zur Hilfe, welche aus einer erlernten Farbverteilung, die passende Farbe eines jeden Pixels differenziert. Der Lernprozess erfolgt in diesem Fall überwacht, da der Optimierung des Klassifikators neben der vorhergesagten Farbe weiterhin die echte Farbe aus der Datengrundlage übergeben wird. Da eine Klassifizierung im Vergleich zur Regression diskrete Ausgangswerte hervorruft, wird hier nur eine limitierte Anzahl möglicher Farben vorhergesagt, welche weit unter der Kapazität gängiger Farbsysteme liegt.

Unüberwachtes Lernen

Entflieht man dem überwachten Lernprozess eines gängigen neuronalen Netzes, optimieren sich generative Modelle, wie das bereits behandelte GAN, unkontrolliert bzw. unüberwacht. Tabelle 2 listet drei vergleichbare Veröffentlichungen, welche dieser Arbeit als Grundlage bereitstanden. Dabei seien Isola et al. [23] besonders hervorzuheben, welche

mit einer einzelnen Implementierung mehrere Anwendungsbereiche abdecken. Dabei realisiert ihr System mit Namen *Pix2Pix* bspw. eine Abbildung von realen Luftaufnahmen zu strukturellen Stadtkarten, koloriert analog zu dieser Arbeit das monochrome Eingangsbild und erzeugt aus einer skizzierten Vorlage ein fotorealistisches Abbild eines Objektes. Folgende konzeptionelle und implementierende Schritte fundieren daher auf jenem System, versuchen es zu reproduzieren und streben – im Einklang mit dem Projektziel – eine plausible Kolorierung an.

Jahr	Veröffentlichung	System	Herausgeber
2018	Image Colorization using Generative Adversarial Networks [24]		Nazeri et al.
2018	Image-to-Image Translation with Conditional Adversarial Networks [23]	Pix2Pix	Isola et al.
2019	ChromaGAN: An Adversarial Approach for Picture Colorization [25]	ChromaGAN	Vitoria et al.

Tabelle 2: Themenbezogene Veröffentlichungen zur Kolorierung mithilfe von GANs

3 Konzept

Aufbauend auf den Stand der Wissenschaft, lassen sich nun konzeptionelle Schritte abstecken, welche für die darauffolgende Implementierung des Systems benötigt werden. Das folgende Konzept bewegt sich in den drei Domänen Daten, Modell und Verwendung, welche jeweils kontextbezogene Unterthemen behandeln.

3.1 Datenkonzept

Bevor ein Modell erzeugt und verwendet werden kann, wird zunächst eine Datengrundlage benötigt. Jene Grundlage füttert im Hinblick auf Abbildung 9 das Modell mit Beispielen und nimmt damit ebenfalls Einfluss auf seine spätere Vorhersage. Es müssen daher spezielle Anforderungen gestellt, Vorbereitungen getroffen und Nutzen gezogen werden, die im weiteren Verlauf dieses Datenkonzeptes behandelt werden.

3.1.1 Anforderung an die Daten

Die Anforderungen an Rohdaten für maschinelles Lernen sind, außer, dass sie vielzahlig sein müssen, kaum zu beziffern. Abhilfe schaffen renommierte und hochdotierte Wettbewerbe im Forschungsumfeld, welche auf Basis eines ausgewählten Datensatzes eine Problemstellung, wie bspw. eine Objekterkennung, ausschreiben, die es möglichst genau zu bewältigen gilt. Als Aushängeschild jener Ausschreibungen gilt die *ILSVRC* (ImageNet Large Scale Visual Recognition Challenge) [26], welche auf millionenfache Bildbeispiele zurückgreift, die im *ImageNet* Datensatz gesammelt sind.

Weitere Eigenschaften eines Datensatzes lassen sich anhand der jeweiligen Problemstellung skizzieren, wonach im Allgemeinen die Datengrundlage repräsentativ auf das Projektziel zugeschnitten sein sollte. Historische Schwarz-Weiß-Aufnahmen sind dabei ebenso ungeeignet, wie Sammlungen himmelblauer Landschaftsbilder, welche entweder keine oder eine lediglich beschränkte Kolorierung zur Folge hätten. Glücklicherweise sind die Archive dieser Welt voll mit analogen Farbfotos und das Internet übersäht von digitalen Schnappschüssen, sodass es keineswegs an repräsentativen Daten mangelt. Inhaltliche Vielseitigkeit, eine gewisse Abbildungsqualität und die lizenzfreie Nutzung zählen abschließend zu den selbstverständlichen Anforderungen an eine Datengrundlage, die es im weiteren Datenkonzept vorzubereiten gilt.

3.1.2 Vorbereitung der Daten

Neben formalen Anforderungen an einen Datensatz, gehen einer späteren Nutzung diverse Maßnahmen voraus, damit sich dieser problemlos in das Projekt einfügt. Jene Maßnahmen unterziehen den Datensatz einer einmaligen Transformationskette, die ähnlich einer Kompression in der Nachrichtentechnik, jene Daten für die weitere Benutzung aufbereitet. Bildinhalte müssen dabei im Speziellen dem System zugeführt, normalisiert, skaliert und ggf. alternativ repräsentiert werden, was der folgenden Transformationskette auf Datenebene entspricht:

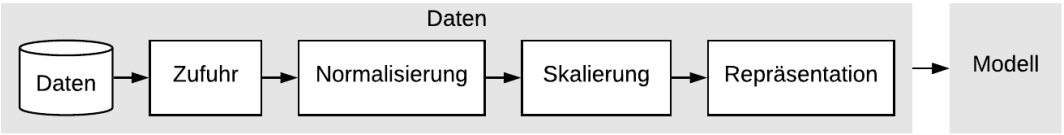


Abbildung 15: Transformationskette auf Daten- und vor der Modellebene

Da die aufbereiteten Daten letztendlich das Modell füttern, lässt sich eine solche Kette am besten rückwärtig betrachten, wo die Repräsentation der Daten steht. Diese sollten nicht wie üblich im RGB-Farbraum vorliegen, da dieser keine direkte Trennung von Luminanz- und Chrominanzanteil erlaubt. An dieser Stelle erfolgt eine Farbraumtransformation, welche wiederum vorausgehende Komponenten der Kette beeinflusst. So muss die Skalierung der Daten ein quadratisches Bildformat sicherstellen, welches für Modelle aus neuronalen Netzen unumgänglich ist. Zuvor werden die Daten einer Normalisierung unterzogen, welche wiederum gewährleistet, dass sich das nachfolgende Modell trainieren bzw. der Farbraum transformieren lässt. Zu Beginn der Transformationskette stellt die Zufuhr sicher, dass alle Rohdaten aus dem Datensatz in einem passenden Format für die weitere Verarbeitung vorliegen.

3.1.3 Aufteilung der Daten

Für die spätere Evaluierung des nachfolgenden Modells, welche im Verwendungskonzept betrachtet wird, muss dieses nicht nur erfolgreich trainiert, sondern ebenfalls validiert bzw. getestet werden, sodass auf die Vorbereitung eine Aufteilung der Daten folgt. Dafür wird die gesamte Datengrundlage für die Trainings- und Testphase zerlegt, wobei zweitere der abschließenden Beurteilung der Genauigkeit des Modells dient und daher abgegrenzt betrachtet werden sollte (Abbildung 16). Die Daten der Trainingsphase werden in einen Trainings- und einen Validierungssatz aufgeteilt, wobei ersterer die Aufgabe der Modelloptimierung und zweiterer die der Modellkontrolle besitzt.

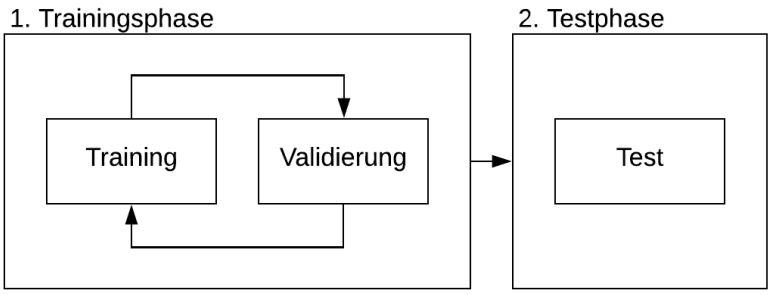


Abbildung 16: Trainings- und Testphase

Der Ablauf der Trainingsphase ist dabei immer gleich. Zu Beginn teilt man die Anzahl der Daten des Trainingssatzes in mehrere Stapel anhand einer bestimmten Stapelgröße. Jeder daraus resultierende Stapelsatz wird nun in das Modell geschickt und die resultierenden Fehler vermerkt. Nach Leeren des Stapelsatzes wird das Modell optimiert bzw. dessen Parameter angepasst. Dieser Prozess wiederholt sich nun bis alle Stapel abgearbeitet sind. Jetzt kommt der Validierungssatz ins Spiel, mithilfe dessen die Genauigkeit der Trainings-

phase berechnet wird. Die Sichtung des kompletten Trainings- und Validierungssatzes wird nun für die Anzahl bestimmter Trainingsepochen wiederholt. Das Modell sieht also in jeder Epoche den gesamten Trainings- und Validierungssatz, welche durch geschicktes Mischen in zeitliche und kontextuelle Unordnung gebracht werden. Nach Ablauf aller Epochen lässt sich die Genauigkeit des Modells in der Testphase auf einem unbekannten Testsatz verifizieren. Die vergangene Aufteilung der Daten für evaluierende Zwecke beschließt das Datenkonzept und zusätzliche Überlegungen werden nun auf Modellebene weitergeführt.

3.2 Modellkonzept

Das Modell versucht sich im Rahmen dieser Arbeit an der Vorhersage von Farbinformationen auf Basis eines Grauwertbildes. Dabei müssen dreidimensionale Eingangsdaten auf ebensolche Ausgangsdaten über eine $\mathbb{R}^3 \longrightarrow \mathbb{R}^3$ Abbildung transferiert werden. Es handelt sich um eine sog. Pixel-zu-Pixel-Transformation, welche die Bildbreite b und -höhe h erhält und lediglich jedes Eingangspixel auf ein Ausgangspixel abbildet. Damit ließe sich sowohl ein klassifizierendes als auch ein regressives neuronales Netz als abbildendes Modell implementieren. Ersteres würde einem Eingangspixel die wahrscheinlichste Farbe aus einer begrenzten Farbpallette zuordnen, während zweiteres einen diskreten Farbwert vorhersagt. Aufgrund der allgemeinen Anwendung dieses Modells, die jedweden natürlichen Bildinhalt als potenzielle Eingabe vorsieht, ist ein Zusammenschluss beider Konzepte nötig, um eine plausible Kolorierung zu erreichen. Dies lässt sich über das bereits behandelte GAN umsetzen, dessen Generator ein synthetisches Ausgangsbild auf Basis eines realen Eingangsbildes liefert und dessen Diskriminator diese Vorhersage dahingehend optimiert, dass sie vom Original nicht zu unterscheiden ist (siehe 2.3.1).

Dieses Modell wird im Folgenden gesondert als Generator bzw. Diskriminator konzipiert. Daraufhin folgt die Optimierung ihres Zusammenspiels mit Betrachtung eines instabilen bzw. stabilen Trainingsprozesses.

3.2.1 Konzeption des Generators

Die ursprüngliche Implementierung des GANs, füttert den Generator mit Rauschen, was im Falle der Imitierung niedrigauflösender Ziffern des *MNIST* Datensatzes (engl. Modified National Institute of Standards and Technology, MNIST) plausible Inhalte erzeugt, bei der Augmentation von Umweltszenen auf Basis der zehn Objektklassen in *CIFAR-10* (engl. Canadian Institute For Advanced Research, CIFAR) allerdings an seine Grenzen stößt [14, 5]. Als Hilfestellung für den Generator nutzt diese Arbeit daher das bekannte Grauwertbild als deterministische Eingabe. Aus einem GAN wird ein bedingtes cGAN, was auf der Luminanz-Komponente konditioniert werden soll (siehe 2.3.2).

Der Eingang des Generators ${X_{gen}}^{b \times h \times 1}$ erhält demnach den Luminanzanteil des realen Bildes RGB, der wiederum mit der erzeugten Chrominanz im Ausgang ${Y_{gen}}^{b \times h \times 2}$ kombiniert wird, um das kolorierte Ergebnis $R'G'B'$ zu erzeugen, wie die folgende Abbildung verdeutlicht:

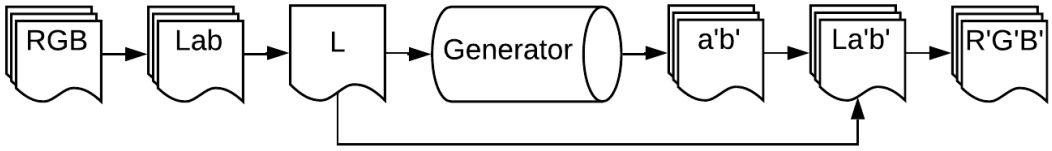


Abbildung 17: Konzeption des Generators

Als Pixel-zu-Pixel-Transformation behält der Generator die Dimensionalität der Eingangsdaten bei, was in der Netzarchitektur zu beachten ist. Die ursprüngliche Implementierung eines GAN muss lediglich das niedrigauflösende Eingangsrauschen über eine Überabtastung in ein hochauflösendes Ausgangsbild wandeln. Dabei ist die Struktur des Rauschens im Eingang verwerflich, da sie normalverteilt ist. Im Falle der realen Luminanz-Komponente als Eingabe muss das Netzwerk aber zunächst auf dessen Struktur ansprechen, da sie den jeweiligen Bildinhalt verbirgt. Dies wird in der CV über ein faltendes neuronales Netzwerk (engl. Convolutional Neural Network, CNN) umgesetzt, welches nach dem mathematischen Begriff der Faltung operiert. Faltungen extrahieren, analog zu bekannten Strukturfiltern der konventionellen Bildverarbeitung, bestimmte Inhalte des Bildes über den Faltungsoperator. Dieser ist aber nicht wie üblich auf einzelne Bildinhalte fokussiert (statische Faltungsmaske), sondern ist veränderlich, da seine Faltungsmaske über das Training dynamisch anpassbar ist. Ein CNN besteht zusammenfassend aus transformierenden faltenden Schichten, dessen Parameter die Maske der Faltung bilden, welche daher über die Fehlerrückführung optimiert werden kann.

Da unser Modell auf hochdimensionalen Bilddaten operiert, bietet sich das bereits vorgestellte Konzept des tiefen Lernens an, was aus dem CNN ein tiefes faltendes neuronales Netz (engl. Deep Convolutional Neural Network, DCNN) werden lässt. Es handelt sich um jenes Konzept, das in der CV in den vergangenen Jahren große Erfolge gefeiert hat, da es analog zu der Funktion eines Encoders, den hochdimensionalen Bildinhalt auf einen eindimensionalen Klassifikator reduziert, der letztendlich zwischen dem beispielhaft erwähnten Hund bzw. der Katze differenziert. Dreht sich diese Encoder-Architektur nun um, ergibt sich mit dem Decoder, das genaue Gegenstück dazu. Der Zusammenschluss beider Teilnetze verkörpert als Generator nach Abbildung 18 nun die Abbildung der Chromiannz: die Netzein- bzw. -ausgabe ist hochdimensional und das resultierende Gesamtnetz ist tief bzw. vielschichtig, was letztendlich den Bildinhalt adäquat erfasst und repräsentiert. Bewusst wird hier auf die Repräsentation des Netzes angespielt, da der Generator nur dann plausibel koloriert, wenn auch eine akkurate Repräsentation des Bildinhaltes herrscht. Wo keine Struktur erkannt wird, kann weder das diskriminierende System der Objekterkennung noch das generierende System der Objektimitierung zufriedenstellende Ergebnisse liefern.

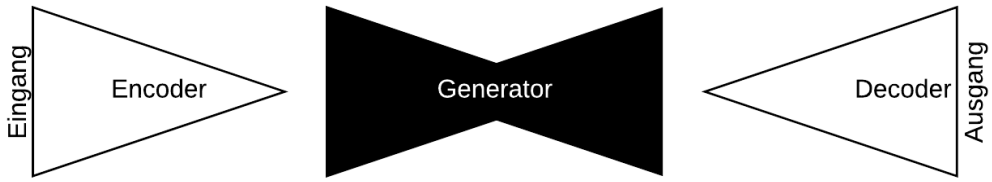


Abbildung 18: Generator als Zusammenschluss von Encoder- und Decoder-Architektur

Ein weiterer Aspekt des Modellkonzeptes beschäftigt sich mit der simultanen Vorhersage beider Farbebenen. Da ein einzelner Decoder vorliegt, wird der zweikanalige Chrominanzanteil über ein und dasselbe Netz abgebildet, was seine Vor- und Nachteile mit sich bringt. Oberflächlich betrachtet bildet eine Pixel-zu-Pixel-Transformation lediglich seinen festen Eingang pixelweise auf einen flexiblen Ausgang ab. Im Falle von strukturellen Stadtkarten ist dieser Ausgang einkanalig, eine Kolorierung wird zweikanalig vorhergesagt und eine Objektimitierung erzeugt ein dreikanaliges Ausgangsbild. Alle drei Anwendungsbereiche lassen sich über einen einzigen Generator realisieren, dessen transformierende Schichten sich mitunter auf abweichende Ausgangskanäle einstellen müssen. An dieser Stelle stellt sich die Frage nach einer eigenständigen Vorhersage beider Farbebenen, da diese im Falle des Lab-Farbraumes zwar Wertegleichheit, aber offensichtlich keine Inhaltsgleichheit besitzen. Ein nachvollziehbares Beispiel stellt ein Naturpanorama dar, welches sowohl aus grünen Wäldern als auch aus blauem Himmel besteht, deren Repräsentation auf zwei unterschiedlichen Farbebenen basiert. Fasst das Naturpanorama zusätzlich zu jenen globalen Flächen lokale Strukturen aus rot-gelben Farbtönen, müsste ein einzelner Decoder sowohl zwischen beiden Bildinhalten und zusätzlich zwischen beiden Farbebenen im Ausgang differenzieren. Ein schwieriges Unterfangen, da seine Entfaltungen auf denselben Faltungsmasken basieren. Abhilfe könnte eine Duplizierung des Decoders schaffen, welcher daraufhin aus zwei unabhängigen Decoder-Zweigen besteht, die von ein und demselben Encoder gespeist werden. Diese besitzen durch ihre Duplizierung zwar weiterhin dieselbe Repräsentationstiefe, bilden aber unabhängig voneinander ihre zugehörige Farbebene ab, was sowohl den Trainingsprozess beschleunigen als auch die Vorhersage verbessern sollte.

Da die gerade behandelte Encoder-Decoder-Architektur den generativen Teil des Modells übernimmt, ließe sich das Projektziel der Kolorierung allein über den Generator realisieren, da es mit dem realen Farbbild eine gewünschte Soll-Ausgabe am Ausgang gäbe. Solch ein System wäre zudem überwacht und dadurch stabiler im Optimierungsprozess. Doch mit dem Diskriminator wurde dem Generator gemäß der Theorie eine Kontrollstruktur zur Seite gestellt, dessen Architektur im Folgenden konzipiert werden soll.

3.2.2 Konzeption des Diskriminators

Der Diskriminator vergleicht die reale sowie generierte Farbinformation, um sich und den Generator über die Fehlerrückführung zu optimieren (Abbildung 19). Auch hier wird die Luminanz als Bedingung zugeführt, was den Eingang des Diskriminators ${X_{dis}}^{b\times h\times 3}$ auf drei Kanäle wachsen lässt. Ausgangsseitig ist der Diskriminator im ursprünglichen GAN als binärer Klassifikator implementiert, was den hochdimensionalen Eingang auf einen eindimensionalen Ausgang ${Y_{dis}}^{1\times 1\times 1}$ herunterbricht und im vorherigen Abschnitt als Encoder-Architektur betitelt wurde (siehe Abbildung 18). Jener Ausgang schätzt als einzelnes Neuron die Wahrscheinlichkeit, ob das eingangsseitige Bild real oder gefälscht ist.

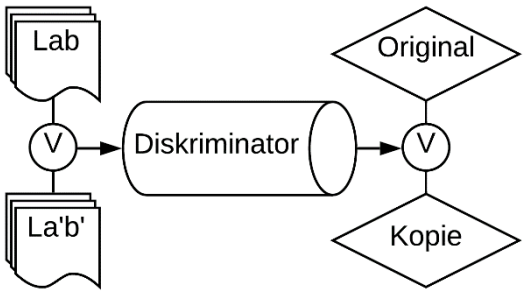


Abbildung 19: Konzeption des Diskriminators

An dieser Stelle kommt die Optimierung des Systems ins Spiel, welche durch die Kaskadierung von Generator und Diskriminator (siehe Abbildung 12) bei letzterem ansetzt. Offensichtlich ist die gewünschte Ausgabe des Diskriminators bekannt, da seine Eingabe entweder aus der realen oder der imaginären Datenverteilung stammt. Welche Probleme der serielle Optimierungsprozess mit sich bringt, wird im folgenden Abschnitt erörtert.

3.2.3 Gründe für ein instabiles Training

Der Optimierungsprozess beruht im ursprünglichen GAN lediglich auf dem Gegenspiel beider Subnetze, das bei optimalem Gleichgewicht im Mittel jedes zweite Bild als plausibel erklärt. An dieser Stelle konvergiert das Training, das Netz scheint optimiert und das Gegenspiel beider Subnetze findet seinen Ausgleich. Doch mitunter lässt sich jenes Optimum nicht erreichen, was an der gegensätzlichen Intention von Diskriminator und Generator liegt und in den folgenden Szenarien ausgeführt wird.

Differenziert der Diskriminator zu stark, geht seine Rückmeldung unter, weil er bereits akkurate, fehlerfreie Einschätzungen liefert. Da der Generator im Zuge der Fehlerrückführung auf jene Rückmeldung des Diskriminators angewiesen ist, wird er nicht weiter optimiert. Das Training konvergiert unter diesen Bedingungen selten und selbst wenn, verfehlt ein kaum verbesserter Generator die Abbildung der realen Datenverteilung. Man spricht von schwindenden Gradienten (engl. Vanishing gradient problem) in Anlehnung an die Fehlerrückführung, die über das Gradientenverfahren das Netz optimiert. [15, 2.3.1]

Differenziert der Diskriminator wiederum zu schwach, lässt sich dieser mit Leichtigkeit vom Generator täuschen. Innerhalb eines Trainingsschrittes lernt der Diskriminator nun die einseitige Abbildung des Generators zu durchschauen – dieser braucht für eine erneute Täuschung aber lediglich die Vorhersage im nachfolgenden Trainingsschritt deutlich zu variieren. Ein Optimierungsprozess, in dessen weiteren Verlauf der Generator durch die reale Datenverteilung traversiert und sie dadurch niemals komplett abbildet, bis er letztendlich kollabiert (engl. Mode collapse) [15, 2.3.2].

Ein drittes Problem lässt sich auf den seriellen Optimierungsprozess zurückführen, welcher zunächst den Diskriminator bei unverändertem Generator optimiert, um diesen im zweiten Schritt auf Basis des fixen Diskriminators anzupassen. In Kombination mit den gegensätzlichen Intentionen des Min-Max-Theorems verhindern alternierende Vorzeichen des Gradienten das Finden des kleinstmöglichen Fehlers, wodurch das Training ebenfalls zum Erliegen kommt [15, 2.3.2]..

Der folgende Abschnitt behandelt daher Maßnahmen, welche die obigen Probleme adressieren und beseitigen sollen, damit der Trainingsprozess des Modells stabilisiert und dessen Optimierung sichergestellt wird.

3.2.4 Maßnahmen für ein stabiles Training

Allen drei Problemfällen des vorherigen Abschnitts ist gemein, dass der Generator mitunter realitätsfern abbildet. Dies liegt in seiner Natur, da er einzig darauf aus ist, den Diskriminator zu täuschen. Abhilfe schafft eine Bestrafung des Generators für weniger natürliche Vorhersagen, sobald seine Abbildung von der realen Datenverteilung abweicht, was als eine Art LSGAN bereits angerissen wurde (siehe 2.3.3). Die folgende generatorspezifische Fehlerfunktion nimmt sich dieser Forderung an, indem sie die absolute Differenz aus realem und imaginärem Bild berechnet. Je höher diese Abweichung, desto größer ist offensichtlich die nötige Anpassung des Modells. Neben der hier vorgestellten kleinsten absoluten Abweichung (7), steht alternativ die Methode der kleinsten Quadrate (8) im Raum, welche jedoch durch die aggressivere Mittelwertbildung eine erhöhte Unschärfe erzeugt – ähnlich dem glättenden Effekt eines Binomialfilters erster und zweiter Ordnung. Da solch eine Fehlerfunktion zudem die Struktur des Bildes schützt, indem sie große Abweichungen zur bekannten Datenverteilung verhindert, wird diese Art von Zielfunktion im Folgenden als Rekonstruktionsverlust betitelt. Je höher dieser und mit ihm die Zielfunktion ausfällt, desto größer ist das Optimierungspotential des Generators.

(7) $\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|\mathrm{x} - \mathrm{G}(\mathrm{z}|\mathrm{y})\|]$

(8) $\mathcal{L}_{L2}(G) = \mathbb{E}_{x,y,z}[\|(\mathrm{x} - \mathrm{G}(\mathrm{z}|\mathrm{y}))^2\|]$

Schwindende Gradienten waren der Grund, dass ein optimaler Diskriminator das Training stilllegt. Abhilfe schaffen künstlich beschränkte Ist-Klassifikationen des Diskriminators. Eine fehlerfreie Einschätzung des Diskriminators wird hierfür absichtlich nach unten bzw. oben korrigiert, damit sich dennoch Gradienten berechnen lassen. Man spricht von geglätteten Zielwerten (engl. Label smoothing), die neben dem Rekonstruktionsverlust als stabilisierende Maßnahme eines Trainings verwendet werden [27, 3.4].

Jene Stabilisierungsmaßnahmen werden im folgenden Kapitel implementiert. Doch bevor dies geschieht, steht die Einschätzung eines erfolgreichen Trainings im Raum.

3.2.5 Indikatoren für ein erfolgreiches Training

Der folgende Abschnitt behandelt Indikatoren, die für ein erfolgreiches Training sprechen, indem er quantitative und qualitative Faktoren anreißt, welche für die Validierung neuronaler Netze gängig sind.

Quantitative Indikatoren

Dabei bedient sich eine numerische Auswertung neuronaler Netze der Abweichung eines aktuellen Ist-Wertes zu einem bekannten Soll-Wert, welche sich auch von gängigen Fehlerfunktionen zu Nutze gemacht wird. Jene Abweichung ist üblicherweise relativ zu sehen, da sie keinen Bezugspunkt besitzt, der zu einer Einheit führt. Geht man zu einer absoluten Abweichung über, kommt bspw. der Signalpegel in Dezibel ins Spiel, dessen Metrik des

Signal-Rauschabstandes im Kapitel der Implementierung konkretisiert wird. Der bekannteste quantitative Indikator eines neuronalen Netzes ist die Genauigkeit der Vorhersage. Werden 99 der 100 Objekte im beliebten Beispiel aus der CV richtig klassifiziert, so besitzt die Objekterkennung eine Genauigkeit von 99 %. Da unser Modell aber kontinuierliche Werte in Form eines Farbanteils ausgibt, müssten für solch eine Auswertung diskrete Wertebereiche aufgeteilt werden, welche durch ihre endliche Auflösung wiederum den Indikator verfälschen.

Qualitative Indikatoren

Nutzt man anstatt eines numerischen Indikators die Wahrnehmung eines Menschen, handelt es sich um einen sog. *Turing-Test*, dessen Auswertung nun auf qualitativen Faktoren basiert. Der menschliche Proband muss hierbei aus einem Original-Fälschungs-Paar das kolorierte Bild entlarven. Diese Prozedur steht analog zum Diskriminator, der ebenfalls ein Bild aus der realen, sowie eines aus der synthetisierten Datenverteilung zu Gesicht bekommt, um zwischen diesen zu differenzieren.

Die folgende Tabelle fasst die behandelten quantitativen und qualitativen Indikatoren zum Abschluss des Modellkonzeptes zusammen:

Auswertung	**Indikator**
Quantitativ	Relative Abweichung zwischen Ist- und Soll-Wert
Quantitativ	Absolute Abweichung zwischen Ist- und Soll-Wert
Qualitativ	Original vs. Fälschung

Tabelle 3: Quantitative und qualitative Indikatoren der Modellvalidierung

3.3 Verwendungskonzept

Nachdem das Modell im vorherigen Abschnitt skizziert wurde, beschließt dessen Verwendung den konzeptuellen Teil des Projektes. Das generative Netzwerk wird, wie in Abbildung 17 ersichtlich, dafür genutzt, aus der bekannten Luminanz und der synthetisierten Chrominanz ein farbiges Bild zusammenzusetzen. Für eine reine Kolorierung wird daher lediglich der Generator verwendet, sobald das GAN ausreichend optimiert ist. Es bietet sich also an, die Verwendung des Generators innerhalb des folgenden Kapitels von dessen Vorgängern, der Daten- und Modellimplementierung zu trennen. So bringt eine Vorhersage Laufzeitvorteile mit sich, da diese unabhängig von einer ressourcenintensiven Optimierung geschehen kann.

Für die quantitative Bewertung der Kolorierung wird dabei zunächst der Testsatz herangezogen, welcher dem Modell unbekannt verblieben ist. Eine Evaluierung seiner Daten ist demnach unvoreingenommen und spiegelt die Anforderungen an die reale Verwendung des Modells am ehesten wider. Eine qualitative Bewertung folgt daraufhin auf Basis realer Bildinhalte, die es zu kolorieren gilt. Weitere Verwendungskonzepte, wie bspw. die Integration der Funktionalität in eine eigenständige Anwendung, sind für diese Arbeit nicht vorgesehen.

4 Implementierung

Nach der Konzipierung der Projektarbeit folgt in diesem Abschnitt deren konkrete Implementierung. Jene Implementierung erfolgt mittels *TensorFlow*, ein quelloffenes Framework für maschinelles Lernen, welches als anwendungs- und entwicklungsorientiertes Hilfsmittel alle drei Domänen des maschinellen Lernens abdeckt. Auf Basis eines hauseigenen Tutorials [28] wurden Programmierabläufe erstellt, die neben der Datenbeschaffung und -aufbereitung, das Modell schichtweise implementieren, es kompilieren, sowie optimieren und letztendlich über Hyperparameter und Metriken validieren. Jene Programmierabläufe lassen sich im *Jupyter Notebook* unter *github.com/tobiasvossen/B.Sc* nachvollziehen, dessen Programmierzellen die komplette Implementierung beinhalten. Der weitere Softwarerahmen inkludiert *Colaboratory* als Entwicklungsumgebung, *TensorBoard* als Analysehilfe, *Python* als Programmiersprache und *Keras* als High-Level-Programmierschnittstelle. Eine Versionsübersicht der Softwarekomponenten, sowie die Struktur des *Jupyter Notebook* lassen sich dem Anhang entnehmen. Verweise auf die konkrete Implementierung sind den folgenden Abschnitten *kursiv* angefügt.

4.1 Datenimplementierung

Zu Beginn der Implementierung stand die Wahl eines geeigneten Datensatzes im Raum, der in referenzierten Systemen nicht selten auf die Datenbank *ImageNet* fiel. Da jene Datengrundlage speziell auf die Objekterkennung zugeschnitten ist, fehlt ihr ein gewisser Realitätsbezug für die vorliegende Aufgabenstellung. Denn formatfüllend abgebildete, einzelne Objekte bilden selten das Motiv einer zu kolorierenden Fotografie. An dieser Stelle wird eine Datengrundlage benötigt, die anstatt der Erkennung von Objekten, sich derer von Szenen annimmt. Jene Datengrundlage bildet eine Umgebung ganzheitlich ab und beschränkt sich daher nicht auf gesonderte Objekte. Vielmehr ist die Möglichkeit gegeben, dass eine Umgebung mehrere Objekte beinhaltet, was die Quantität an Objekten im Datensatz erhöht, während diese im *ImageNet* Datensatz auf die Menge an Klassen beschränkt ist.

Während *Pix2Pix* und *ChromaGAN* auf Basis von *ImageNet* trainiert sind, setzten Nazeri et al. [24] stattdessen auf *Places365* [29], ein ursprünglich aus 365 unterschiedlichen Umgebungen bestehender Datensatz, der mittlerweile auf mehr als 400 Szenenklassen gewachsen ist. Ein weiteres Merkmal ist die teilweise Unterscheidung einer Oberkategorie in mehrere Unterkategorien. So werden Stadionszenen bspw. nach der Sportart aufgeteilt, oder Kulturstätten nach abgebildetem Innen- bzw. Außenbereich unterschieden. Aktuell zählt der Datensatz über 10 Millionen Datenexemplare, welche sich lizenzfrei in Trainings-, Validierungs- und Testsätzen akquirieren lassen [29, 3].

Datenakquise – *Collect data*

Die Akquise jener Datensätze geschieht problemlos über Server des Herausgebers *MIT* (Massachusets Institute of Technology). Da die verwendete Entwicklungsumgebung begrenzte Speicherkapazitäten besitzt, wurde lediglich der aus 36500 Beispielen bestehende Validierungssatz, sowie der aus 328500 Beispielen bestehende Testsatz bezogen. Ersterer

wurde dabei simultan für das Training, sowie die Validierung benutzt, während letzterer als Testsatz seinen ursprünglichen Zweck erfüllte.

Datenaufbereitung – *Preprocess data*

Auf die Akquise der Datengrundlage folgt ihre Aufbereitung, da es sich um komprimierte Bilddaten nach dem JPEG-Standard handelt. Dabei wird jedes Bild nach der folgenden Kette prozessiert.

1. Dekodierung der Rohdaten von JPEG- in Tensor-Format
2. Normalisierung von Ganzzahlen auf Gleitkommazahlen
3. Skalierung auf 256×256 Pixel
4. Farbraumtransformation in HSV bzw. Lab-Farben
5. Normalisierung von vorzeichenlose auf -behaftete Gleitkommazahlen

Letzterer Schritt umfasst eine zweite notwendige Normalisierung, um das Modell mit einem vorzeichenbehafteten Wertebereich $\mathcal{R} \in \{-1, 1\}$ zu füttern, da die letzte Aktivierungsschicht des abbildenden Generators als hyperbolischer Tangens modelliert ist, anstatt auf vorzeichenlose Aktivierungsfunktionen zu setzen.

Die Reihenfolge spielt dabei eine übergeordnete Rolle, da bspw. die Farbraumtransformation an vierter Stelle auf einem beschränkten Zahlenintervall beruht, wonach ihr eine Normalisierung vorangestellt werden muss. Weitere Aufbereitungen, die weniger formale Aspekte der Datengrundlage behandeln als vielmehr dessen inhaltliche Repräsentation betrachten, sind nicht zu berücksichtigen. Es ist zudem nicht notwendig die Verteilung der Beispielklassen anzugleichen, da diese in der Datengrundlage bereits repräsentativ enthalten ist. Weiterhin wendet die fehlende Nutzung der eigentlichen Szenenkategorien, speziell negative Einflüsse durch schiefe Klassenverteilungen ab. Besondere Aufmerksamkeit bedürfen lediglich farblose Beispiele im Datensatz, da diese mitunter als einkanaliges Grauwertbild dekodiert werden.

Datensatz – *Build dataset*

Der letzte Schritt der Datenimplementierung erzeugt auf Basis vorangegangener Schritte einen sofort nutzbaren Datensatz, welcher bereits gemischt und in Stapel aufgeteilt ist. Weitere Augmentationsschritte sind nicht notwendig, da das Projektziel die Kolorierung auf reale Szenen beschränkt, welche keiner geometrischen Verzerrung ausgesetzt sind und das Modell somit auch nicht translationsinvariant kolorieren muss, wie es bspw. in der Objekterkennung gefordert ist. Eine Erweiterung der Datenmenge ist ebenfalls nicht vonnöten, da ihre Quantität bereits für tiefes Lernen ausreicht, was über die Szenenerkennung mittels eines CNN bereits betrachtet wurde [29, 5].

4.2 Modellimplementierung

Auf die Beschaffung und Vorbereitung der Datengrundlage folgt die Implementierung des Modells auf Basis seiner Architektur. Weitere Abschnitte behandeln kompilierende, trainierende und validierende Schritte, die es vor einer letztendlichen Verwendung des Modells zu realisieren gilt.

4.2.1 Architektur des Modells

Die Modellarchitektur versprach den Zusammenschluss einer rekonstruktiven Encoder-Decoder-Struktur als Generator und einer differenzierenden Decoder-Struktur als Diskriminator. Jene Subnetze, die in Kombination das mehrfach erwähnte GAN bilden, werden im weiteren Verlauf dieses Abschnittes implementiert, indem ihre Architektur mitsamt transformierenden Schichten konkretisiert wird.

Generator – *Generator*

Da der Generator als Encoder-Decoder-Struktur konzipiert wurde, besitzt er in seiner Mitte eine Art Flaschenhals, welcher zwar ein hohes Maß an Repräsentation beinhaltet, diese aber in ihrer Auflösung beschränkt ist (siehe Abbildung 18). Betrachtet man die vielzitierte Objekterkennung, stellt solch eine Auflösungsminderung kein Problem dar. Dem dort verwendeten Encoder reichen die verschiedenen Repräsentationsformen, um den ursprünglichen Bildinhalt zu klassifizieren. Ähnliches gilt für den bereits konzipierten Diskriminator, welcher lediglich zwischen Original und Fälschung unterscheiden muss, den primären Bildinhalt dabei aber verwirft. Jener Verlust der Struktur wäre für den Generator keine Alternative, da er als Pixel-zu-Pixel-Transformation den Bildinhalt adäquat rekonstruieren muss. An dieser Stelle kommt eine Netzwerk-Architektur namens *U-Net* [30] ins Spiel.

Jene Architektur wurde ursprünglich für die Segmentierung medizinischer Bildinhalte entwickelt, wodurch eine strukturwahrende Abbildung an erster Stelle stand, da es sich um regionsbasierte Segmentierungsverfahren handelt, die auf lokale Strukturen angewiesen sind. Diese Anforderung wurde im referenzierten *U-Net* über Verbindungen zwischen dem zusammenziehenden (engl. contracting path) und dem expandierenden (engl. expansive path) Teil des Netzes sichergestellt. Ersterer verringert gemäß des Encoder-Konzeptes mit voranschreitender Netzwerktiefe die Bildauflösung, während zweiterer als Decoder die ursprüngliche Auflösung wiederherstellt. Dabei stellt der zusammenziehende dem expandierenden Teil seine verarbeiteten Daten über symmetrische Verbindungen zur Verfügung, die als Verknüpfungen (engl. concatenating path) zwischen Encoder und Decoder geschaltet sind (Abbildung 20).

Die konkrete Implementierung der *U-Net* Architektur erfordert zunächst eine Betrachtung beteiligter Schichten im En- bzw. Decoder, welche für die Verringerung bzw. Erhöhung der Auflösung verantwortlich sind. Jene Auflösungsanpassung wird im Folgenden über unterabtastende bzw. überabtastende Blöcke realisiert.

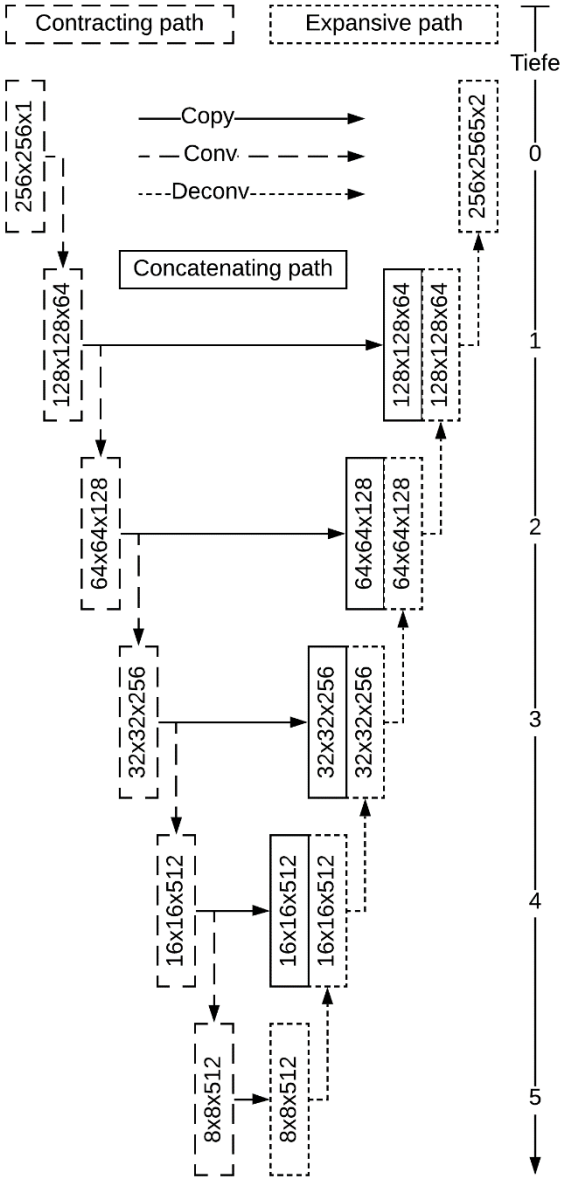


Abbildung 20: Implementierung des Generators als U-Net in Anlehnung an [30, Abb. 1]

Unterabtastung – *Downsampling*

Unterabtastende Blöcke bestehen aus einer Faltungsschicht, welche die Auflösung des Eingangs halbiert, aber dessen Repräsentationen verdoppelt. Aus einem dreikanaligen 8×8 Eingangsbild wird bspw. ein sechskanaliges 4×4 Ausgangsbild mit halber Auflösung. Die Anzahl an Kanälen, welche die Zahl der Repräsentationen verbirgt, wird über die Menge an verschiedenen Faltungsmasken bestimmt. Soll die Zahl der Repräsentationen und damit bewusst die Aussagekraft dieser erhöht werden, muss die Anzahl an Faltungsmasken wachsen. Damit die Datenmenge nach dem Encoder-Prinzip abnimmt, wird im Umkehrschluss die Bildauflösung verringert. Im ursprünglichen *U-Net* verringert sich die Auflösung im zusammenziehenden Teil über eine Art Mittelwertbildung benachbarter Pixelwerte, nur das anstatt des Mittelwertes das Maximum beteiligter Pixelwerte verwendet und der Rest verworfen wird (engl. Max-Pooling-Layer). Die Bevorzugung der Maximalwerte hätte neben der Dilatation von Strukturen, eine Erhöhung der Pixelwerte zur Folge, deren verfälschender

Effekt für strukturwahrende Abbildungen, wie im Falle des Generators gefordert, unbrauchbar wäre. Stattdessen behilft sich diese Implementierung einer schrittweisen Faltung. Anstatt wie üblich jedes Pixel einzeln zu falten, wird bei einer Schrittweite von 2×2 lediglich jedes zweite Pixel verarbeitet, was zu einer halbierten Auflösung führt. Durch die quadratische Größe der Faltungsmaske, die ähnlich zur Max-Pooling-Layer üblicherweise die 3×3 Nachbarschaft verarbeitet, wird dennoch das gesamte Bild einbezogen. Damit lässt sich zusammenfassen, dass ein unterabtastender Block im Encoder die Auflösung halbiert, während die Repräsentationstiefe verdoppelt wird. Neben der Faltungsschicht beheimatet jener Block zwei weitere Transformationen, eine Normalisierungs- und eine Aktivierungsschicht, wie die folgende Abbildung verdeutlicht:

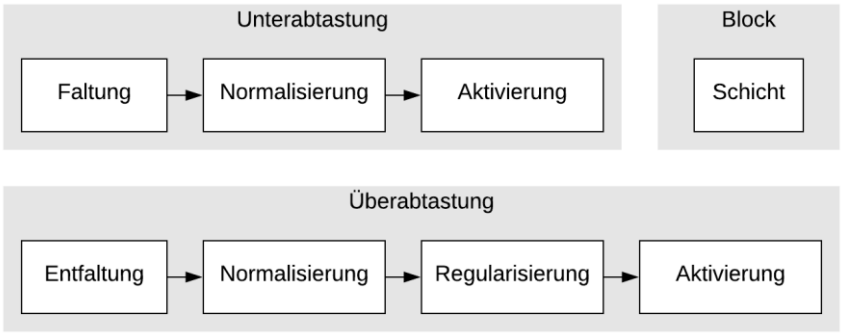


Abbildung 21: Unterabtastender (ob.) und überabtastender Block (un.)

Überabtastung – *Upsampling*

Im Gegensatz zur bereits betrachteten Unterabtastung verdoppelt ein überabtastender Block im Decoder die Auflösung des Eingangsbildes und halbiert gleichzeitig dessen Repräsentationstiefe. Die ursprüngliche Faltung wird über den Inverse-Faltungsoperator rückgängig gemacht, wodurch sich der Bildinhalt wiederherstellt. Analog zum unterabtastenden Block wird der Entfaltung eine Normalisierungsschicht angefügt, die im Folgenden näher betrachtet wird.

Normalisierung – *Normalization*

Normalisierung hält bereits innerhalb der Aufbereitung der Eingangsdaten Einzug, wo sie die weiteren Verarbeitungsschritte begünstigt. Ein weiterer Grund liegt zudem in der späteren Verwendung des Modells, dessen eingängliche Bilddaten natürlich von unbekannter Natur sind. Damit das Modell im Zuge einer späteren Vorhersage besser generalisiert, werden alle Eingänge, egal ob bekannt während des Trainings, oder unbekannt innerhalb der Verwendung, angeglichen bzw. normalisiert. Dies hat zur Folge, dass zwei identische Bildsignale trotz abweichender Signalstärke durch bspw. unterschiedliche Bittiefe dieselbe Kolorierung erreichen. Eine ähnliche Intention verfolgt die Normalisierungs-Schicht, die lediglich Neuronensignale der vorherigen Schicht über einen Mittelwertabzug im Verhältnis zur Standardabweichung glättet (9). Jene Glättung gleicht Signale, ähnlich wie in der Datenaufbereitung, einander an, was den Trainingsprozess unterstützt, da er mit weniger schwankenden Fehlerfunktionen zu kämpfen hat. Weiterhin erfolgt die Normierung nicht pixel-, sondern kanalweise über das gesamte Bild, wodurch sich normalisierende Schichten lediglich über die Berechnungsgrundlage von Mittelwert μ und Standardabweichung σ un-

terscheiden. Dabei normiert die BN (engl. Batch Normalization) [31] auf dem gesamten Stapel, sodass jede Normalisierung lediglich auf einer Repräsentation bzw. einem Kanal agiert. Zerlegt man zusätzlich den Stapel in seine ursprünglichen Beispiele, normiert die IN (engl. Instance Normalization) [32] jedes Beispiel aus einem Stapel unabhängig vom restlichen Stapel. Zum besseren Verständnis benötigt die IN bei einem dreikanaligen RGB-Eingangsbild und acht Beispielen pro Stapel in Summe also 24 Normalisierungen, während die BN lediglich drei berechnet. Daneben stellt jede Normierung zwei weitere Parameter γ, β bereit, die sich optimieren lassen und dem Netz als trainierbare Gewichte bereitstehen.

(9) $y = \frac{\gamma(x-\mu)}{\sigma} + \beta$

Weiterhin fügt die Normierung dem Signal einen geringen Rauschanteil hinzu, da die Schätzung von Mittelwert und Standardabweichung immer mit einem Fehler belastet ist. Jene Rauschzugabe begünstigt jedoch die Verallgemeinerung des Modells, was unter Regularisierung bekannt ist und im Folgenden erläutert wird.

Regularisierung – *Regularization*

Zusätzlich zur Normalisierungs- und Aktivierungsschicht wird dem dekodierenden Teil des Generators eine zufällige Komponente beigefügt, die als Aussetzer-Schicht (engl. Dropout) [33] unbestimmte Verbindungen zwischen Neuronen der vorherigen und nachfolgenden Schicht im jeweiligen Block aussetzt. Dies hat zur Folge, dass jene Neuronen und damit auch ihre Transformationen punktuell keinen Einfluss auf die gemeinsame Verarbeitung des Blocks haben, was letztendlich eine deterministische Rekonstruktion des Generators verhindern soll. Da die Rauscheingabe der ursprünglichen Implementierung eines GAN verworfen wurde, könnten zwei ähnliche Eingaben zu ein und derselben Ausgabe führen. Abhilfe schaffen die gerade behandelten aussetzenden Schichten, aber auch der zugesetzte Rauschanteil normalisierender Schichten aus dem vorherigen Abschnitt. Beide ermöglichen eine Regularisierung des Modells, indem sie deterministische Vorhersagen verhindern bzw. variierende Kolorierungen begünstigen.

Aktivierung – *Activation*

Die abschließende Schicht eines unter- bzw. überabtastenden Blocks entscheidet letztendlich darüber, ob das Signal eines einzelnen Neurons stark genug ist, um in den nächsten Block transferiert zu werden. Anschaulich sind Neuronen entweder aktiviert oder deaktiviert, wobei letztere das Eingangssignal aus der vorherigen Schicht stoppen und verfallen lassen. Diese Entscheidung trifft die Aktivierungsschicht, welche im gängigsten Fall als ReLU (engl. Rectifier Linear Unit) [34] implementiert ist. Die Aktivierungsfunktion dieser Schicht sperrt den negativen Anteil des Signals und lässt einzig seinen Positivanteil durch (10). Da sich negative Eingangssignale auflösen, während positive unverändert bleiben, würde in einem zufällig initialisierten Netz lediglich die Hälfte der Neuronen aktiviert werden, was für den darauffolgenden Lernprozess von Vorteil ist.

(10) $ReLU(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$

Probleme bereitet jene Aktivierungsfunktion im Falle eines negativ belasteten Netzes. Springen aus den vorherigen Schichten größtenteils negative Werte hervor, würde die Akti-

vierungsschicht eine Weiterleitung verhindern. Jene inaktiven Neuronen könnten sich infolgedessen nicht mehr an der Repräsentation des Netzes beteiligen, was zwar einer Regularisierung ähnlich sieht, in diesem Fall aber durch die deterministische Deaktivierung der Neuronen nicht zielführend ist. Abhilfe schafft eine spärliche Aktivierung negativer Eingangswerte, was durch eine LeakyReLU erreicht wird, die im Folgenden beziffert ist.

(11) $LeakyReLU(x) = \begin{cases} x, & x > 0 \\ 0.1x, & x \leq 0 \end{cases}$

Nach Umsetzung des Generators auf Basis faltender, normalisierender, regulierender und aktivierender Schichten, wird im folgenden Abschnitt sein Gegenspieler, der Diskriminator, implementiert.

Diskriminator – *Discriminator*

Nach der Konzeption des Diskriminators (siehe 3.2.2) soll dieser als Encoder implementiert werden, welcher das hochauflösende Eingangsbild ausgangsseitig als Original bzw. Fälschung klassifiziert. Da jene binäre Klassifizierung mitunter zu allgemein unterscheidet, wird das Bild in Ausschnitte (engl. Patches) zerlegt, die jeweils eine eigene Realitätsfrage gestellt bekommen. Der Ausgang des Diskriminators ${Y_{dis}}^{k \times k \times 1}$ fasst demnach anstatt eines einzelnen Ausgangsneurons ein ganzes Raster an $k \times k$ binären Klassifikationen, was als *PatchGAN* [35] bekannt wurde und ebenfalls in allen referenzierten Systemen implementiert ist. Die Anzahl an Klassifikatoren ist experimentell auf 30×30 festgelegt worden, wonach jeder klassifizierte Bildausschnitt 70×70 Pixel misst. Da die Zielfunktion der Fehlerrückführung lediglich eine Einschätzung erwartet, werden alle 30×30 binären Klassifikatoren gemittelt, was im Vergleich zum ursprünglichen Encoder-Konzept die Genauigkeit der Einschätzung erhöhen soll und zudem hochfrequente Strukturen unverfälschter einbezieht. Die Architektur des Diskriminators ist im Folgenden verdeutlicht, wobei jede Auflösungsverringerung analog zum Generator über unterabtastende Blöcke aus Faltungs-, Normalisierungs- und Aktivierungsschicht realisiert wird, welche kaskadiert den Diskriminator bilden:

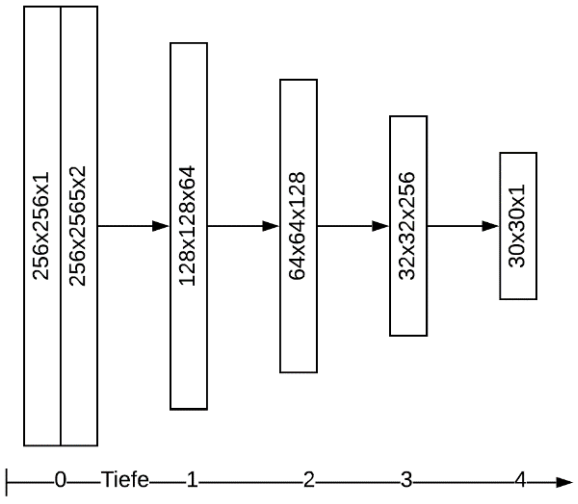


Abbildung 22: Implementierung des Diskriminators als PatchGAN [35]

4.2.2 Kompilierung des Modells

Nach der vorangegangenen Implementierung der Netzarchitektur auf Block- bzw. Schichtebene folgt die Kompilierung des Modells, welche sich neben der Wahl einer geeigneten Zielfunktion, mit der Optimierung der Modellparameter auf Basis jener ausgewerteten Zielfunktion beschäftigt. Dabei unterscheiden sich Generator und Diskriminator nicht nur in ihrer Architektur, sondern auch in der Fehlerrückführung auf Basis ihrer jeweiligen Zielfunktion.

Zielfunktion des Generators – *Generator loss*

Um ein stabiles Training zu fördern und die plausible Vorhersage des Generators zu gewährleisten wurden im Abschnitt 3.2.4 Maßnahmen vorgestellt, die auf einer angepassten Verlustfunktion basieren. Jener Rekonstruktionsverlust auf Basis der absoluten Abweichung (7) wird nun der ursprünglichen Zielfunktion des cGANs (4) zur Seite gestellt, was ebenfalls in [23] und [24] geschieht und zur folgenden Zielfunktion des Generators führt.

(12) $G^* = \min_G \max_D \mathcal{L}_{cGAN}(D, G) + \lambda \cdot \mathcal{L}_{L1}(G)$

Dabei sei erwähnt, dass mit fortschreitendem Training einzig der Rekonstruktionsverlust $\mathcal{L}_{L1}$ gegen ein Minimum streben sollte, da der GAN-Verlust $\mathcal{L}_{cGAN}$ ein numerischer Indikator für das Gleichgewicht zwischen Generator und Diskriminator darstellt. Fällt er stark ab, handelt es sich womöglich um schwindende Gradienten, welche das Training gänzlich zum Erliegen bringen (siehe 3.2.3).

Zielfunktion des Diskriminators – *Discriminator loss*

Die Zielfunktion des Diskriminators verbleibt in ursprünglicher Form, da es hier nicht auf die Rekonstruktion der realen Datenverteilung ankommt. Auch eine vom Original abweichende, aber adäquate Kolorierung, wie bspw. ein blau anstatt rot gefärbtes Auto, kann für den menschlichen Betrachter als plausibel gelten und sollte damit keine fehlerhafte Einschätzung des Diskriminators hervorrufen, dessen Zielfunktion im Folgenden aufgeführt ist.

(13) $D^* = \min_G \max_D \mathcal{L}_{cGAN}(D, G)$

Optimierung – *Optimiser*

Die Nutzung der Zielfunktion erfolgt nun nach Abbildung 10 über die Optimierung – vielmehr über das Verfahren, das hinter der Optimierung steckt. Hierbei hat sich in den vergangenen Jahren Adam (engl. Adaptive Moment Estimation) [36] durchgesetzt, welcher im Vergleich zu herkömmlichen Gradientenverfahren, wie bspw. SGD (engl. Stochastic Gradient Descent), auf Basis des ersten und zweiten zentralen Moments (mittlere Abweichung und Varianz), seine Parameter adaptiv anpasst. Dies hat zur Folge, dass ein gesuchtes Minimum aufgrund des höheren Moments bei Näherung an dieses Minimum schneller erreicht wird, als es über Gradientenverfahren ohne adaptive Parametrierung der Fall wäre. Anschaulich lässt sich jenes Moment mit dem Rollen einer Kugel vergleichen, die je nach Beschaffung des Gefälles an Geschwindigkeit zunimmt, bis sie den Fußpunkt des Gefälles erreicht. Weitere Begriffe, wie die *Lernrate* der Optimierung, die mit der Schrittweite der Näherung an ein gesuchtes Minimum, korreliert, werden ebenfalls adaptiv angepasst und begünstigen daher eine erfolgreiche Optimierung. Jene erfolgreiche Optimierung führt ten-

denziell zu einer Konvergenz des Trainingsprozesses, welcher im Folgenden implementiert wird.

4.2.3 Training des Modells

Das Training basiert, wie in Abschnitt 3.1.3 definiert, auf einer stapelweisen Sichtung des Trainingssatzes. Netzparameter werden demnach erst nach Berücksichtigung einer bestimmten Stapelgröße (engl. Batch Size) angepasst. Gleichzeitig durchläuft das Training einmalig alle Stapel pro Trainingsepoche, sodass jedes Datenexemplar mehrfach vom Modell gesehen wird, bevor das Training abgeschlossen ist. Jeder Stapel an Trainingsdaten wird nun für die konsekutive Optimierung von Generator und Diskriminator genutzt, was im Vergleich zur ursprünglichen Implementierung des GAN die Reihenfolge invertiert.

Verlustmetriken – *Loss metrics*

Da die Zielfunktion des Generators (siehe 4.2.2) aus zwei Teilen besteht, können beide Summanden inklusive ihrer Fehlersumme als *Verlustmetriken* herangezogen werden. Jene trainingsbezogenen Metriken sind in der folgenden Tabelle aufgeführt und sollen Aufschluss darüber geben, wie stabil bzw. erfolgreich das Training verläuft.

Subnetz	**Metrik**	**Ziel**
Generator	Kreuzentropie-Verlust $\mathcal{L}_{cGAN}(G)$	Gleichgewicht
Generator	Rekonstruktionsverlust $\mathcal{L}_{L1}(G)$	Minimum
Generator	Summierter Fehler G^*	Minimum
Diskriminator	Kreuzentropie-Verlust $\mathcal{L}_{cGAN}(D)$	Gleichgewicht
Diskriminator	Summierter Fehler D^*	Minimum

Tabelle 4: Verlustmetriken

Dabei streben die Kreuzentropie-Verluste von Generator bzw. Diskriminator ein Gleichgewicht an, das im bekannten Nash-Equilibrium zwischen beiden Gegenspielern resultiert. Sollte eines der beiden Subnetze die Überhand gewinnen, wird dies mit schwindendem Kreuzentropie-Verlust indiziert, was bspw. aufgrund von schwindenden Gradienten zu einem stagnierenden Trainingsprozess führt. Der Rekonstruktionsverlust hingegen strebt im Laufe des Trainingsprozesses ein Minimum an, da dieser die Abweichung von realer zu imaginärer Vorhersage beinhaltet und somit bereits während des Trainings einen guten Indikator für eine erfolgreiche Kolorierung darstellt.

Trainingsprozess – *Training process*

Doch wie verläuft der Trainingsprozess im Detail? Dafür wird zunächst das synthetisierte Bild auf Basis der Luminanz-Komponente vom Generator vorhergesagt. Jene Kopie füttert daraufhin den Diskriminator, dessen Einschätzung über den Kreuzentropie-Verlust $\mathcal{L}_{cGAN}$ ausgewertet wird, welcher sich genau dann verringert, wenn der Diskriminator eine generierte Kopie als Original klassifiziert. Gleichzeitig bestimmt die pixelweise Abweichung von Kopie und Original den Rekonstruktionsverlust $\mathcal{L}_{L1}$. Die Summe beider Verluste fließt anschließend als zu verringernder Fehler in die Optimierung, welche nach Abarbeitung des

Stapelsatzes den Generator mitsamt seiner Parameter dahingehend anpasst, wie die folgende Abbildung verdeutlicht.

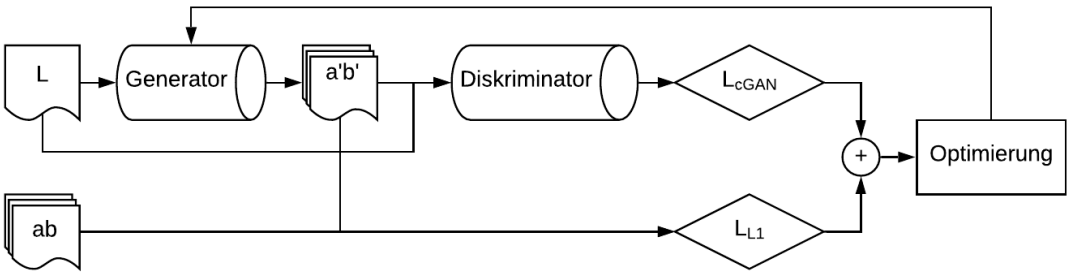


Abbildung 23: Training des Generators in Anlehnung an [28]

Vergleichend dazu sichtet der Diskriminator sowohl das Original als auch die generierte Kopie, um zwei Einschätzungen abzugeben. Eine als Original deklarierte Kopie lässt dabei den Kreuzentropie-Verlust ebenso ansteigen, wie ein als Fälschung entlarvtes reales Bild. In beiden Fällen liegt der Diskriminator mit seiner Einschätzung falsch und seine Fehlersumme wächst, die es daraufhin über die Optimierung des Diskriminators zu verringern gilt, wie die folgende Abbildung darstellt.

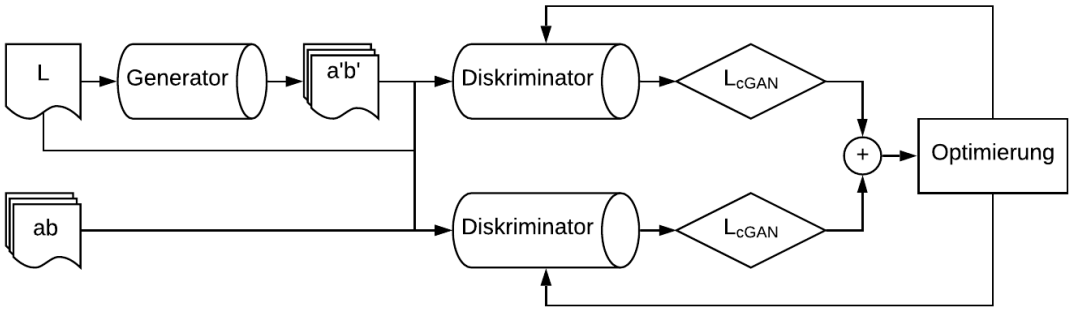


Abbildung 24: Training des Diskriminators in Anlehnung an [28]

4.2.4 Validierung des Modells

Nach Implementierung, Kompilierung und Optimierung des Modells folgt zugleich seine Validierung. Wie bereits in Abschnitt 3.2.5 angeschnitten, lassen sich dazu quantitative und qualitative Indikatoren heranziehen, die im Folgenden als Metriken konkretisiert werden sollen.

Quantitative Metriken – *Quantitative metrics*

Dabei besitzt die neuronale KI (siehe Abbildung 8) den Vorteil, dass ihr Training bereits auf bekannten Frage-Antwort-Paar fußt. Die Validierung eines Modells kann daher mit erstrebenswerten Zielen abgeglichen werden, die in diesem Falle als realer Farbanteil eines Bildes bekannt und verfügbar sind. Folgende numerische Metriken benutzen dabei allesamt jene Zielwerte zur Validierung des Modells.

Der mittlere quadratische Abstand (engl. Mean Squared Error, MSE) bildet den Mittelwert der Abweichungsquadrate zwischen einem bekannten Istwert Y und einem geschätzten

Sollwert Y' (14) und definiert darüber die Qualität jenes Schätzers. Da das vorliegende Modell ebenfalls eine Schätzung über eine plausible Kolorierung abgibt, lässt sich diese Metrik problemlos für die Modellvalidierung nutzen.

(14) $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - {Y'}_i)^2$

Ähnliches gilt für das Spitzen-Signal-Rausch-Verhältnis (engl. Peak Signal-to-Noise Ratio, PSNR) als absolute Metrik im Bereich der quantitativen Auswertung. Jenes logarithmierte Verhältnis zwischen Nutzsignal I_{signal} und Rauschsignal I_{noise} (15) wird in der Bildverarbeitung als Indikator für eine hochwertige Rekonstruktion gesehen. Dabei entspricht das Nutzsignal dem originalen Bild und das Rauschsignal der Abweichung zwischen originalem und rekonstruiertem Bild, wonach eine verringerte Abweichung zu einem erhöhten Signal-Rasch-Verhältnis und damit tendenziell zu einer hochwertigeren Rekonstruktion führt.

(15) $PSNR = \frac{I_{signal}}{I_{noise}} = 20 \cdot \log_{10}\left(\frac{MAX}{\sqrt{MSE}}\right)[dB]$

Neben statistischen Metriken lässt sich auf Basis der Farbraumkomponenten (L, a, b) der Farbabstand (16) als wahrnehmungsbezogene Größe modellieren, die innerhalb des Lab-Farbraumes auf Basis von visuellen Empfindungen des menschlichen Sehsystems berechnet wird. Dabei lassen gewisse Farbabstände auf die Unterscheidbarkeit zweier Farben schließen. Ein Farbabstand kleiner 0,5 attestiert bspw. der Reproduktion einen nahezu unmerklichen Farbunterschied, der selbst vom geübten Betrachter nicht zu unterscheiden ist [8, 5.6].

(16) $\Delta E = \sqrt{(L - L')^2 + (a - a')^2 + (b - b')^2}$

Dennoch ist eine quantitative Auswertung mit Vorsicht zu genießen, da eine abweichende Vorhersage, die dennoch nahe an der Realität liegen kann, zu hohen Abweichungen führt. Dies verschlechtert die jeweilige Metrik und damit scheinbar auch die Kolorierung des Modells. Jene Problematik, die bspw. ein blau gefärbtes Auto im Vergleich zum rot gefärbten Original – trotz vorhandener Plausibilität – sanktioniert, haben alle vorgestellten Metriken gemeinsam.

Qualitative Metriken – *Qualitative metrics*

Abhilfe schaffen qualitative Auswertungen, welche sich gänzlich vom Zielwert der realen Kolorierung entfernen. Vielmehr lässt sich der überwachte Lernprozess des Diskriminators als Vorlage heranziehen. Analog zur modellbasierten Einschätzung zwischen realem und synthetisiertem Bildinhalt versuchen menschliche Probanden im *Turing-Test* die Vorhersage des Generators von der originalen Kolorierung zu unterscheiden und als maschinell erstellt zu entlarven. Je häufiger das synthetische Bild dabei den Probanden täuscht, desto plausibler scheint die Kolorierung des Modells.

Eine ähnlich qualitative Auswertung fußt auf der ursprünglichen Verwendung des Datensatzes als überwachte Objekterkennung und ist als *Inception Score* bekannt. Jenes als *Inception [37]* betitelte Modell feierte in der Vergangenheit als tiefes faltendes neuronales Netz bahnbrechende Erfolge in der Klassifizierung von Objekten. Lässt man nun anstatt der realen Daten die vom Generator synthetisierten Daten klassifizieren, so ergibt sich eine Klassenverteilung anhand der Häufigkeit an erkannten Objekten, welche daraufhin zweigleisig

analysiert wird. Zum einen sollen identische Objekte zu einer normalverteilten Klassifikation führen, da abweichende Objekte nicht erkannt werden und zum anderen sollen variierende Objekte zu einer gleichverteilten Klassifikation führen, da verschiedene Objekte erkannt werden. In diesem Optimalfall wäre der *Inception Score* hoch und die synthetisierten Bilder des Generators würden das Modell überzeugen. Nähern sich aber beide Verteilungen einander an, ist mindestens eines der beiden Ziele nicht erfüllt, sodass der *Inception Score* abfällt. Jener Abfall wird über die KLD (Kullback-Leibler-Divergenz) beziffert, welche sich mit wachsender Deckungsgleichheit zweier Verteilungen verringert (17).

(17) $\mathrm{KLD}(\mathrm{P},\mathrm{Q}) = \sum_x P(x) \cdot \log_{10} \frac{P(x)}{Q(x)}$

Da die qualitative Auswertung über den *Inception Score* auf einer bestimmten Daten-Modell-Grundlage fußt, ist er für diese Arbeit aktuell nicht von Relevanz. Vielmehr wird über die zu Grunde liegende KLD die Kohärenz zwischen realer Datenverteilung P und generierter Datenverteilung Q untersucht. Jenes Abweichungsmaß ist zwar nicht symmetrisch, strebt aber für sich annähernde Datenverteilungen gegen ein Minimum.

Die folgende Tabelle fasst die quantitativen und qualitativen Metriken zum Abschluss der Modellvalidierung zusammen:

Auswertung	**Metrik**	**Ziel**
Quantitativ	Mittlerer quadratischer Abstand MSE	Minimum
Quantitativ	Signal-Rausch-Verhältnis $PSNR$	Maximum
Quantitativ	Farbabstand ΔE	Minimum
Quantitativ	Kullback-Leibler-Divergenz KLD	Minimum
Qualitativ	Turing-Test	Maximum
Qualitativ	Inception-Score	Maximum

Tabelle 5: Quantitative und qualitative Metriken der Modellvalidierung

4.3 Verwendungsimplementierung

Nach erfolgreicher Optimierung des Modells wird der Generator als abbildendes Subnetz, nun dafür verwendet, realitätsnahe Vorhersagen zu treffen und unbekannte Inhalte zu kolorieren. Dafür lässt sich neben der quantitativen Auswertung des Testsatzes wiederum die qualitative Auswertung nach menschlicher Betrachtung implementieren.

Modell testen – *Test model*

Die abschließende numerische Auswertung, welche sich an den quantitativen Metriken der Modellvalidierung orientiert, wird auf dem Testsatz vollzogen, damit diese vergleichbar bleiben.

Kolorierung – *Kolorize examples*

Die abschließende visuelle Auswertung wird ebenfalls auf dem Testsatz vollzogen, dessen grauwertige Eingangsbilder dem realen, sowie kolorierten Ausgangsbild gegenübergestellt werden.

Modell protokollieren – *Log model*

Zum Abschluss der Implementierung wird das Modell für eine spätere Verwendung aufbereitet. Es werden folgende Funktionalitäten bereitgestellt, welche sich vollständig unter *github.com/tobiasvossen/B.Sc* einsehen lassen:

- Visualisierung und Bereitstellung der Modellarchitekturen zur Reproduktion des Systems
- Exportierung des Modellfortschritts zur Verwendung der Kolorierung
- Visualisierung und Bereitstellung des Trainingsfortschrittes zur Wiederaufnahme der Modelloptimierung

Nach Implementierung aller daten-, modell- und verwendungsspezifischen Schritte des Systems, wird dieses im abschließenden Praxiskapitel analysiert und evaluiert.

5 Evaluierung

Nach der konkreten Implementierung des Konzeptes, folgt mit der Evaluierung das letzte praxisbezogene Kapitel dieser Projektarbeit. Die folgenden Abschnitte analysieren dafür die Repräsentation des Farbraumes innerhalb der Datenevaluierung, hinterfragen Architekturen über die Modellevaluierung und bewerten kolorierte Resultate anhand der Verwendung des Systems. Dabei werden experimentelle Hyperparameter definiert, welche im Zuge dieses Kapitels anhand verschiedener Metriken evaluiert werden. Jene Evaluation geschieht auf Basis dreier Trainingsläufe, die im Folgenden aufgezählt sind:

(i) Erster Trainingslauf mit beschränkter Datenmenge und verringerter Epochenzahl: Ziel dieses Trainingslaufes ist die Abbildung des Trainingssatzes, was sich in einem verringerten Rekonstruktionsverlust äußern sollte. Dafür wird eine Vielzahl unterschiedlicher Hyperparameter verwendet, die es für nachfolgende Läufe einzugrenzen gilt.
(ii) Zweiter Trainingslauf mit repräsentativer Datenmenge und erhöhter Epochenzahl: Ziel dieses Trainingslaufes ist die Abbildung des Validierungssatzes, was sich in einem verringerten Farbabstand äußern sollte. Dafür wird sich an der Menge verbliebener Hyperparameter bedient, die im vorherigen Lauf eingegrenzt wurde.
(iii) Dritter Trainingslauf mit repräsentativer Datenmenge und erhöhter Epochenzahl: Ziel dieses Trainingslaufes ist die Abbildung des Testsatzes, was sich in einem erhöhten Signal-Rauschabstand äußern sollte. Dafür werden feste Hyperparameter verwendet, die sich in vorherigen Läufen als ideal herausgestellt haben.

In der Theorie wird (i) bereits den Trainingssatz abbilden, täte sich aber mit der Rekonstruktion des Validierungs- bzw. Testsatzes schwer, was als Überanpassung (engl. Overfitting) bekannt ist. Ein begrenzt repräsentativer Trainingssatz lässt das Modell zwar optimieren, aber nicht generalisieren, sondern lediglich spezialisieren.

Im Gegenzug sollte (ii) zusätzlich den Validierungssatz abbilden und (iii) letztendlich den Testsatz akkurat kolorieren, was über eine Erhöhung der Datenmenge bzw. einer Verlängerung des Trainingsprozesses sichergestellt wird. Das Eingrenzen optimaler Hyperparameter trägt weiterhin seinen Teil dazu bei, dennoch beruht der Erfolg des maschinellen Lernens, speziell der von tiefen neuronalen Netzen, auf hohen Datenmengen und vermehrten Trainingsiterationen. Um beides auf ein angemessenes Maß zu reduzieren, wurde die Stapelgröße von 64 Beispielen pro Stapel auf verringerte 32 in (i) bzw. 16 in (ii) und (iii) herabgesetzt, was dem Konzept des *Mini-Batch-Learning* entspricht. Diese Entscheidung führt erstens zu einer Beschleunigung des Trainingsprozesses, da die Fehlerrückführung aufgrund verringerter Stapelgröße häufiger eingreift und zweitens zu einer Verbesserung des Optimierungsprozesses, der durch erhöhte Normalisierung stabilisiert wird. Die folgende Tabelle beziffert dabei die verwendete Epochen- und Datenanzahl pro Trainingslauf:

Lauf	**Epochen**	**Beispiele**	**Stapelgröße**	**Trainingssatz**	**Validierungssatz**	**Testsatz**
(i)	64	512	32	384	64	64
(ii) + (iii)	128	8192	16	8000	96	96

Tabelle 6: Epochen- und Datenanzahl pro Trainingslauf

5.1 Datenevaluierung

Der erste Trainingslauf diente dem Eingrenzen genereller Variationen und wurde daher aufgrund einer Vielzahl möglicher Kombinationen mit begrenzter Datenmenge bestritten. Die Dauer einer Epoche war neben der Stapelmenge abhängig vom verwendeten Farbraum, der verfügbaren GPU, sowie der quantisierten Bittiefe. Letztere lässt sich speziell während des laufzeitintensiven Trainingsprozesses verringern, sodass dieser beschleunigt wird. Man spricht von der sog. *Mixed Precision* [38]*,* da mit Ausnahme der Optimierung alle signalverarbeitenden Schritte bei voller Bittiefe verbleiben, sodass die namensgebende Vermischung der Genauigkeiten entsteht. In diesem Fall wird die gängige Auflösung von 32-bit innerhalb des Optimierungsprozesses auf 16-bit beschränkt, was die Dauer des Trainings halbiert. Weitere Zeitersparnis bringt der Verzicht auf die rechenintensive nichtlineare Konvertierung in Lab-Farben mit sich, dessen Laufzeit vom HSV-Farbraum deutlich unterboten wird. Neben der Laufzeit lassen sich weitere Unterschiede auf Basis des Farbraumes feststellen, die im folgenden Abschnitt betrachtet werden.

5.1.1 Hyperparameter auf Datenebene

Die erste Parameterentscheidung bezieht sich auf die Wahl des Farbraumes, welcher auf Datenebene großen Einfluss auf weitere Komponenten des Systems besitzt. Da die Datengrundlage über RGB-Farben repräsentiert wird, muss diese zunächst über eine Farbraumtransformation in HSV- bzw. Lab-Farben konvertiert werden. Ersterer Fall bringt als lineare Transformation konstante Laufzeit-Vorteile mit sich, die unabhängig von weiteren Parametern alle Rücktransformationen für die nachträgliche Evaluation in RGB-Farben beschleunigen.

Im Gegensatz dazu begünstigen Lab-Farben eine frühzeitige Trainingskonvergenz, was auf den gesamten Optimierungsprozess gesehen ebenfalls Laufzeit-Vorteile mit sich bringt. In der Theorie konvergiert das Training auf Lab-Farben schneller, da die konvertierten Farbebenen aufgrund der Mehrheit ungesättigter Bildinhalte eine normalverteilte Struktur aufweisen. Jene Werteverteilung ähnelt der initialisierten Gewichtsverteilung der (ent)faltenden Schichten des Modells, sodass die Abweichung von imaginärer zu realer Datenverteilung zu Trainingsbeginn geringer sein sollte und dieses schneller konvergieren ließe. Farbton und -sättigung hingegen verteilen sich homogener, wie die folgenden zwei Abbildungen verdeutlichen:

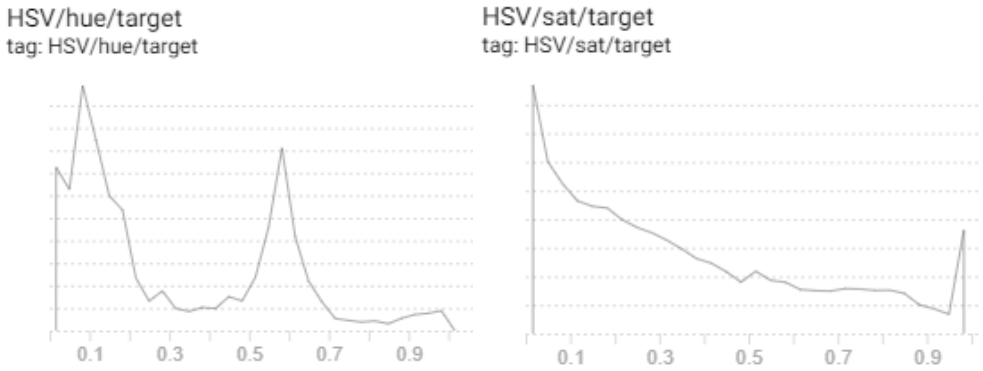


Abbildung 25: Chrominanz-Komponenten der realen Datenverteilung im HSV-Farbraum

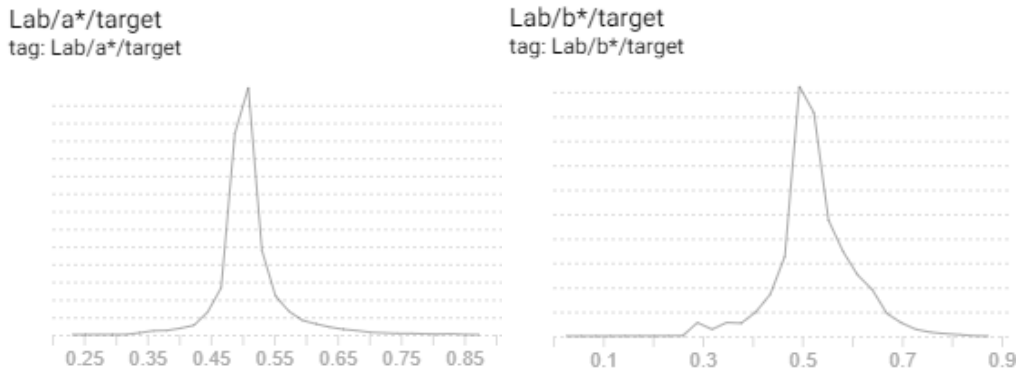


Abbildung 26: Chrominanz-Komponenten der realen Datenverteilung im Lab-Farbraum

Der Vergleich beider Farbkanäle, die als Zielwert der Generator-Abbildung gelten, stellt die Machbarkeit einer simultanen Vorhersage beider Chrominanz-Komponenten in Frage. Wo referenzierte Systeme ausschließlich auf Lab-Farben setzen und somit zwei ähnliche Verteilungen abbilden müssen, tut sich der Decoder im generativen Modell mit der hohen Divergenz innerhalb des HSV-Farbraumes schwer. Ein Lösungsansatz, welcher sich dieser Problemstellung annimmt, indem er die Chrominanz-Komponente kanalweise vorhersagt, wird auf Modellebene evaluiert.

5.1.2 Evaluierung der Datenparameter

Wie die folgenden Diagramme vergleichend darstellen, liegen die anfänglichen Abbildungsqualitäten des HSV-Farbraumes deutlich unter denen des Lab-Farbraumes (Abbildung 27, li.). Mit verlängerter Trainingsdauer lässt sich nach 64 Epochen des ersten Trainingslaufes ein Fazit ziehen. Der anfängliche Unterschied zwischen beiden Farbräumen bleibt auch zu Trainingsende bestehen, was sich an einer abweichenden Divergenz zwischen HSV- und Lab-Farben auf dem Validierungssatz verdeutlicht (Abbildung 27, re.). Die weiteren Trainingsläufe prozessieren daher ihre Datengrundlagen – analog zu referenzierten Systemen – auf Lab-Farben.

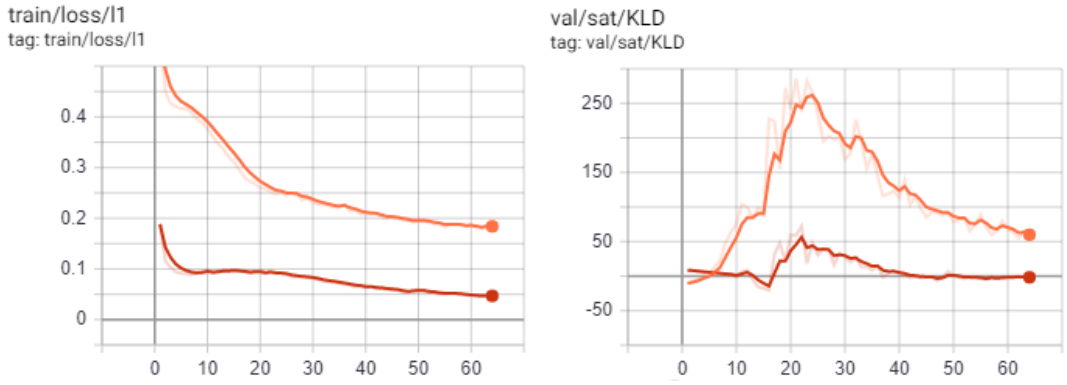


Abbildung 27: Rekonstruktionsverlust (li.) und KL-Divergenz (re.) für HSV- (orange) und Lab-Farben (rot)

5.2 Modellevaluierung

Der zweite Teil dieses Kapitels beschäftigt sich mit der Evaluation modellspezifischer Hyperparameter, die in der folgenden Tabelle aufgeführt sind. Zu beachten ist, dass sich nicht

jede Steuergröße auf das gesamte Netz auswirkt, da manche Parameter lediglich das generative bzw. diskriminierende Modell beeinflussen.

Modell	Parameter	Beschreibung
Generator	Netztiefe	Anzahl verwendeter Blöcke
	Lernrate	Lernrate der Optimierung
	Regularisierung	Regularisierungsschicht
	Kanaltrennung	Decoder-Trennung
Beide	Normalisierung	Normalisierungsschicht
	Aktivierung	Aktivierungsfunktion
Diskriminator	Netztiefe	Anzahl verwendeter Blöcke
	Lernrate	Lernrate der Optimierung

Tabelle 7: Hyperparameter auf Modellebene

5.2.1 Hyperparameter auf Modellebene

Im Folgenden werden jene Hyperparameter vorgestellt, welche sowohl die komplette Modellarchitektur betreffen, aber auch spezielle Schichten variieren, sodass mögliche Optimierungen allein durch eine Parametrisierung des Modells erreicht werden können.

Netztiefe

Betrachtet man zunächst ein Modell anhand seiner Repräsentationen, lassen diese auf die Komplexität des Modells schließen, welche sich anhand der (ent)faltenden Schichten abschätzen lässt. Dabei gilt: je mehr Faltungsmasken eingesetzt werden, desto größer ist die Zahl an Repräsentationen, welche das Modell abdeckt und umso höher wird seine Komplexität eingeschätzt. Betrachtet man die Generator-Architektur des referenzierten *U-Net,* erhöht sich die Zahl verwendeter Faltungsmasken ab einer bestimmten Netztiefe nicht mehr, sodass die Komplexität des Modells unverändert bleibt. Dies hängt mit endlichen Trainingskapazitäten zusammen, die ansonsten durch weitere Faltungsmasken schneller erschöpft wären. Vergleichend zählt die Diskriminator-Architektur weitaus weniger unterabtastende Blöcke, sodass seine Netztiefe geringer ausfällt als die des Generators. Um letztendlich die Balance zwischen Generator und Diskriminator zu wahren, wurde im Vergleich zu referenzierten Systemen die Netztiefe des Generators und damit die Zahl an verwendeten Repräsentationen angepasst. Generell sollte der Generator als abbildendes Subnetz aber weiterhin eine höhere Komplexität aufweisen als sein diskriminierender Gegenspieler, da zweiterer lediglich als Kontrollstruktur dient. Schwindende Kreuzentropie-Verluste, die als Anzeichen eines stagnierenden Trainingsprozesses gelten, können mit einer Annäherung der Komplexitäten zwischen Generator und Diskriminator verringert werden, ohne dass auf weitere stabilisierende Maßnahmen, wie bspw. geglättete Zielwerte, zurückgegriffen werden muss.

Kanaltrennung

Während *Pix2Pix* seitens ihrer Generator-Architektur einen einzelnen Decoder vorsieht, an dessen Ende die Farbinformation als zweikanaliger Ausgang bereitsteht, lässt sich jene Farbvorhersage aufspalten, was bereits konzeptionell betrachtet wurde (siehe 3.2.1). Über den Hyperparameter der Kanaltrennung ließe sich nun der Vergleich zwischen einer simultanen und einer getrennten Abbildung der Farbebenen ziehen, wobei zweitere Variante trotz unveränderter maximaler Repräsentationstiefe die Komplexität des Decoders verdoppelt, was zu einem verlängerten Trainingsprozess führt. Bewusst wird sich in diesem Fall für eine gemeinsame Optimierung entschieden, welche ihre Anpassung über das gesamte generative Modell vollzieht. Es bleibt also trotz Kanaltrennung im dekodierenden Teil des Generators bei jeweils einer Fehlerrückführung des Generators bzw. Diskriminators.

Normalisierung

Normalisierende Schichten finden jeweils im Generator sowie Diskriminator Einzug (siehe Abbildung 21), sodass jede (ent)faltende Transformation normalisiert wird, bevor ihre Signale durch das Netz traversiert werden. Neben der stapelweisen Normalisierung (BN) referenzierter Systeme prüft diese Arbeit zusätzlich, ob eine beispielweise Normalisierung (IN), die bessere Wahl wäre. Für den Verzicht auf normalisierende Schichten gab es in der Literatur keinen Ansatz und so wurde diese Alternative verworfen.

Regularisierung

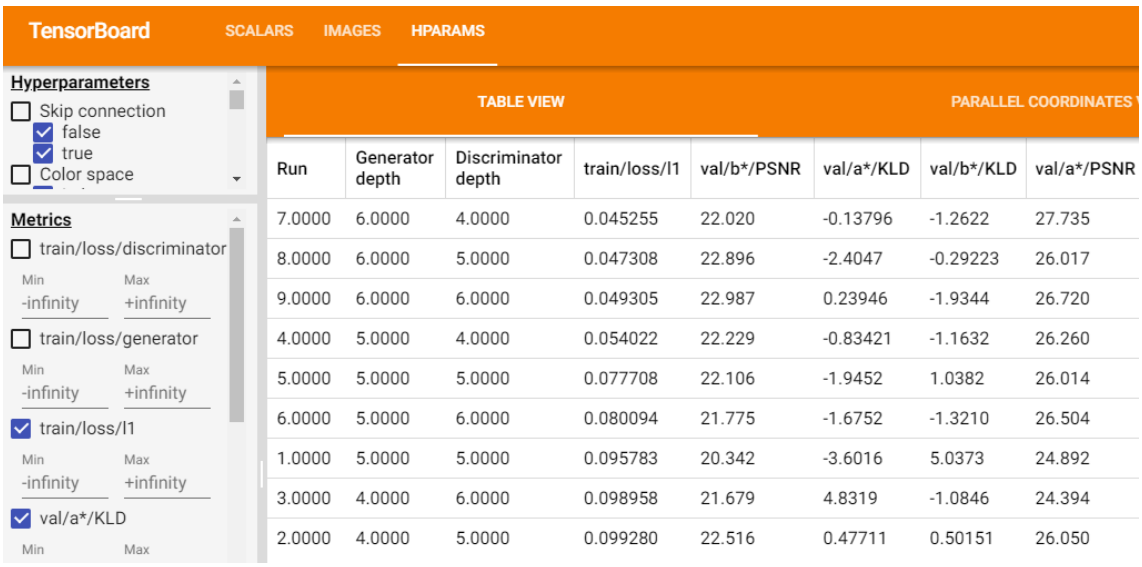
Regulierende Schichten sind auf Basis aussetzender Schichten (Dropout) lediglich dem dekodierenden Teil des Generators vorbehalten, wonach unterabtastende Blöcke keine Regularisierung erfahren. Um gleiche Voraussetzungen zu schaffen, ließe sich die ursprünglich verworfene Rauschzugabe des Generators wiederholen. Im Gegensatz zum niedrigauflösenden Eingangsrauschen stellt dieses System der Luminanz-Komponente aber ein auflösungsidentisches normalverteiltes Rauschen zur Verfügung, welches den Eingang des Generators auf zwei Repräsentation erweitert. Gemäß der Intention einer stabilen Klassifikation wurde auf jedwede Regularisierung im Diskriminator verzichtet.

5.2.2 Evaluierung der Modellparameter

Während das Training über die Verlustmetriken (siehe Tabelle 4) evaluiert wird, kommt vor Ende jeder Trainingsepoche der Validierungssatz zum Einsatz, welcher weitere quantitative Metriken (siehe Tabelle 5) bereitstellt, auf Basis derer sich in (i) folgende Beobachtungen aufstellen lassen.

Netztiefe

Der Einfluss der Netzarchitektur wurde anhand verschiedener Netztiefen bewertet. Dafür wurde jeweils der Generator bzw. Diskriminator aus vier bis sechs unter- bzw. überabtastenden Blöcken erzeugt, sodass in Summe neun mögliche Kombinationen durchlaufen wurden. Betrachtet man die folgende tabellarische Übersicht, lässt sich festhalten, dass der Generator als abbildendes Modell eine tiefere Architektur aufweisen sollte als der kontrollierende Diskriminator:



Run	Generator depth	Discriminator depth	train/loss/l1	val/b*/PSNR	val/a*/KLD	val/b*/KLD	val/a*/PSNR
7.0000	6.0000	4.0000	0.045255	22.020	-0.13796	-1.2622	27.735
8.0000	6.0000	5.0000	0.047308	22.896	-2.4047	-0.29223	26.017
9.0000	6.0000	6.0000	0.049305	22.987	0.23946	-1.9344	26.720
4.0000	5.0000	4.0000	0.054022	22.229	-0.83421	-1.1632	26.260
5.0000	5.0000	5.0000	0.077708	22.106	-1.9452	1.0382	26.014
6.0000	5.0000	6.0000	0.080094	21.775	-1.6752	-1.3210	26.504
1.0000	5.0000	5.0000	0.095783	20.342	-3.6016	5.0373	24.892
3.0000	4.0000	6.0000	0.098958	21.679	4.8319	-1.0846	24.394
2.0000	4.0000	5.0000	0.099280	22.516	0.47711	0.50151	26.050

Abbildung 28: Evaluierung der Netztiefe

Neben dieser Übersicht lässt sich anhand eines Streudiagramms die Architektur des größten und kleinsten Rekonstruktionsverlustes auffinden. Trotz abweichender Verluste auf dem Trainingssatz erreichen beide Architekturen einen vergleichbaren Signal-Rauschabstand auf dem Validierungssatz. Dies lässt auf eine fehlende Generalisierung des Modells schließen, wie bereits für den ersten Trainingslauf aufgrund von begrenzter Datenmenge vorhergesagt wurde (siehe Abbildung 29).

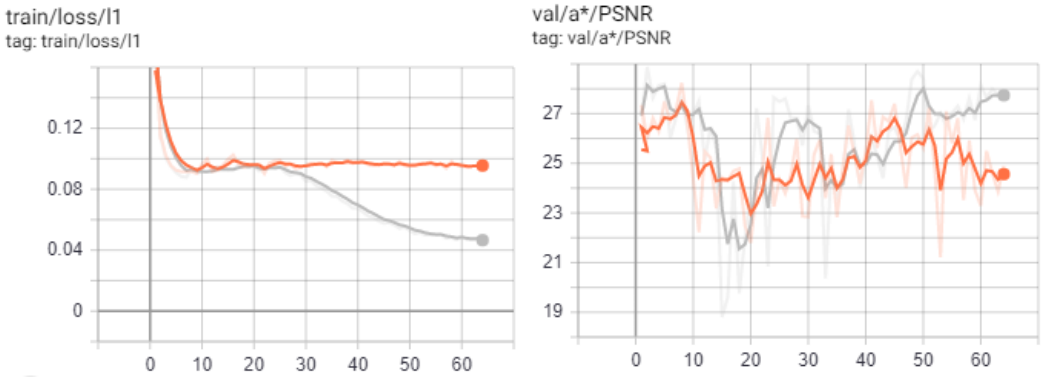


Abbildung 29: Rekonstruktionsverlust (li.) und Signal-Rauschabstand (re.) zweier unterschiedlicher Netzarchitekturen

Kanaltrennung

Zieht man die duplizierte Decoder-Architektur einer simultanen Rekonstruktion des Chrominanzanteil vor, bildet der Generator sowohl auf dem Trainings- als auch auf dem Validierungssatz realitätsnäher ab, sobald seine Vorhersage kanalweise getätigt wird. Weitere Trainingsläufe werden daher mit eben dieser Kanaltrennung vollzogen.

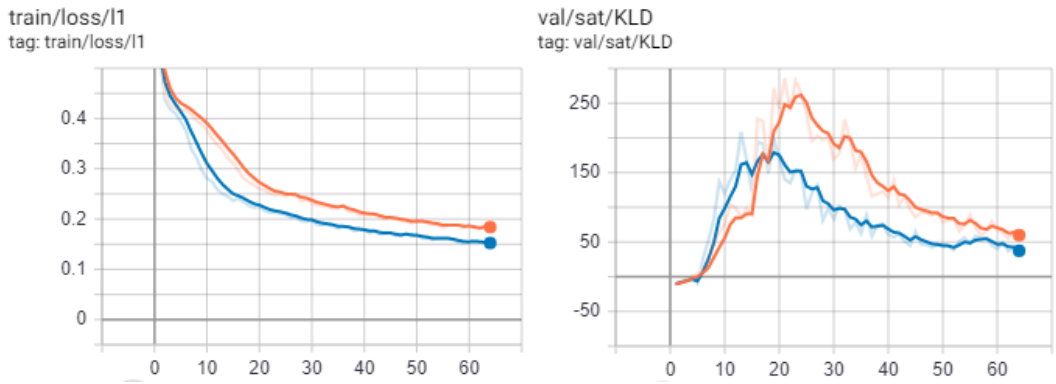


Abbildung 30: Rekonstruktionsverlust (li.) und KL-Divergenz (re.) ohne (orange) und mit Kanaltrennung (blau) auf HSV-Farben

Normalisierung

Neben der Betrachtung des Rekonstruktionsverlustes, der in erster Linie die Abbildungsqualität der realen Datenverteilung bewertet, beziffert der Kreuzentropie-Verlust das Zusammenspiel beider Subnetze auf Trainingsebene. Dabei lässt sich festhalten, dass eine stapelweise Normalisierung den Generator besser abbilden lässt und im Umkehrschluss eine beispielweise Normalisierung den Diskriminator zu fehlerfreieren Einschätzungen verhilft. Dies deckt sich mit der Theorie, wonach die stapelweise Normalisierung den Generator besser generalisieren lässt, während der Diskriminator anhand von einzeln normierten Beispielen seine Einschätzung stabilisieren kann, wie folgende Diagramme vergleichend belegen:

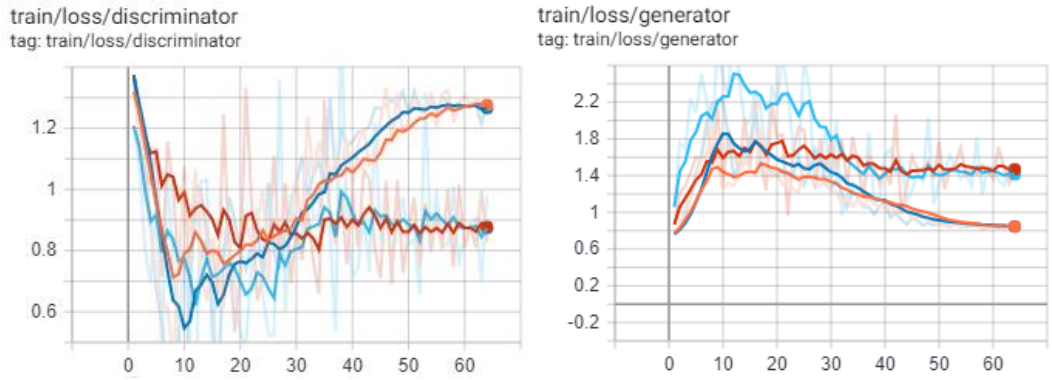


Abbildung 31: Kreuzentropie-Verlust von Diskriminator (li.) und Generator (re.) für stapelweise (orange/blau) bzw. beispielweise Normalisierung (rot/blau)

Zurück zum Rekonstruktionsverlust lässt sich festhalten, dass das Training mithilfe der stapelweisen Normierung schneller konvergieren sollte, da sein Rekonstruktionsverlust sich mit voranschreitender Dauer minimiert (Abbildung 32, li.). Die Betrachtung der Divergenz zwischen imaginärer und realer Datenverteilung bescheinigt beiden Normalisierungsmethoden aber wiederum ähnliche Abbildungsqualitäten (Abbildung 32, re.).

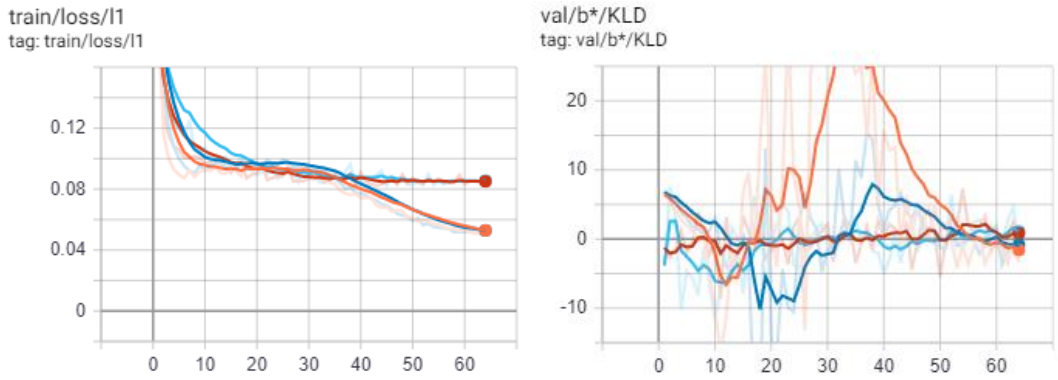


Abbildung 32: Rekonstruktionsverlust (li.) und KL-Divergenz (re.)

Ein Kompromiss aus beiden Normalisierungsmethoden lässt sich über die Verkleinerung der Stapelgröße erreichen, wie es in (ii) und (iii) auf Basis von *Mini-Batch-Learning* geschah. Jedes Beispiel wird damit zwar weiterhin auf dem gesamten Stapel normalisiert, dessen regulierender Effekt sich aber durch die verkleinerte Stapelgröße einer beispielweisen Normalisierung annähert. Demnach bedienen sich folgende Trainingsläufe einer stapelweisen Normalisierung in Kombination mit reduzierter Stapelgröße.

5.3 Verwendungsevaluierung

Nach Abschluss des ersten Trainingslaufes und der Eingrenzung der Daten- und Modellparameter wurde die Datenmenge signifikant erhöht, sodass nachfolgende Trainingsläufe das Modell besser generalisieren lassen sollten.

Quantitative Evaluierung

Der zweite Trainingslauf konnte dabei nach 128 Epochen zwar den Trainingssatz zu 97 % abbilden, was einem Rekonstruktionsverlust von 0,03 entspricht, verfehlte eine Generalisierung auf dem Validierungssatz aber trotz erhöhter Datenmenge. Dabei entpuppte sich der Diskriminator als zu mächtig, da er mit steigender Epochenanzahl immer häufiger die synthetisierte Kolorierung des Generators entlarvte. Schwindende Gradienten führten dazu, dass sich die Verlustfunktion des Generators erhöhte, während seine Abbildungsqualität abnahm, was sich in einem stagnierenden Signal-Rauschabstand auf dem Validierungssatz äußerte. Eine sofortige Abhilfe schaffte in (iii) die Anpassung der Diskriminator-Lernrate, die darauf folgend lediglich ein Zehntel der des Generators betrug. Dies entspricht einer Relation, die sich so nur in [24] wiederfand, wohingegen *Pix2Pix* und *ChromaGAN* auf Basis identischer Lernraten optimieren. Letztendlich lässt sich diese Relation mit der abweichenden Komplexität beider Gegenspieler erklären. Während das generative Modell der Netztiefe fünf auf 26 Millionen Parameter zurückgreifen kann, besitzt der Diskriminator mit Netztiefe vier nur $1/40$ dieser Komplexität, was durch die Lernrate kompensiert werden muss. Durch die reduzierte Lernrate des Diskriminators, stabilisieren sich in (iii) die Kreuzentropie-Verluste und damit das Verhältnis beider Gegenspieler, was zu einer akkurateren Abbildung der realen Datenverteilung geführt hat. Die folgenden Diagramme zeigen dabei die Kreuzentropie-Verluste aus (ii) und (iii) untereinander auf:

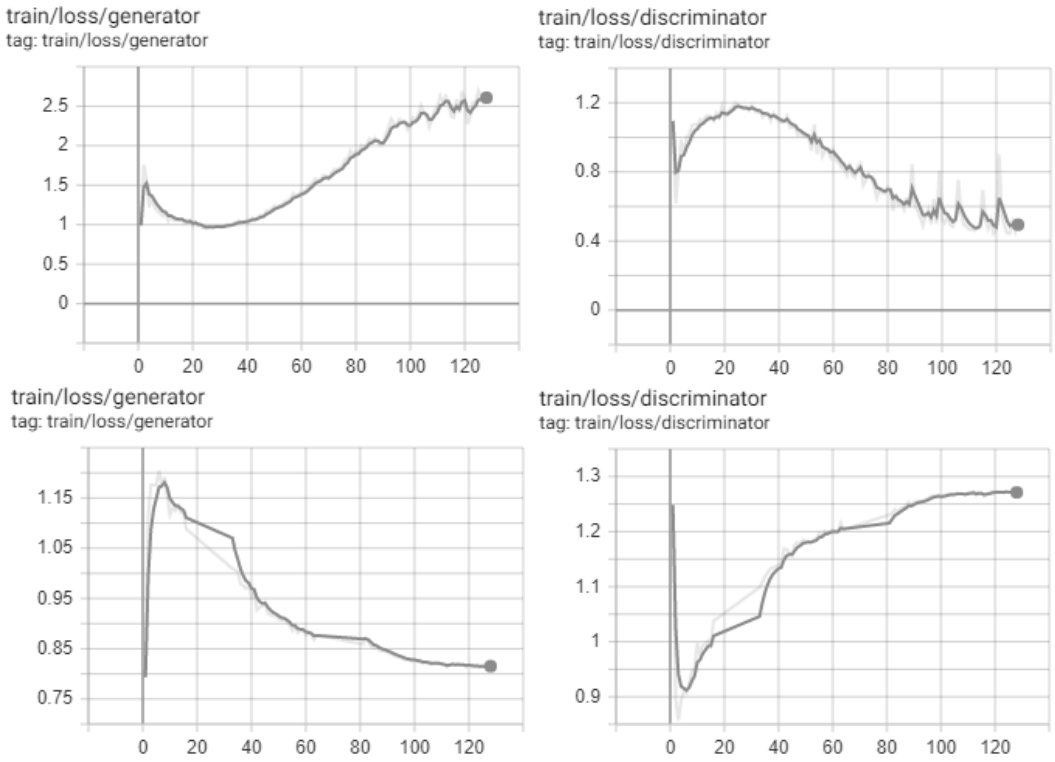


Abbildung 33: Kreuzentropie-Verluste aus (ii) [ob.] und (iii) [un.]

Auf dem Validierungssatz, der teilbekannt war, da er aus derselben Datengrundlage wie der Trainingssatz stammte, wurde mit erhöhter Epochenzahl ein fallender Farbabstand zwischen imaginärer Kolorierung und realer Vorlage erreicht. Dieser lag aber letztendlich über dem Grenzwert eines tolerierbaren Farbunterschiedes, wie das folgende Diagramm verdeutlicht:

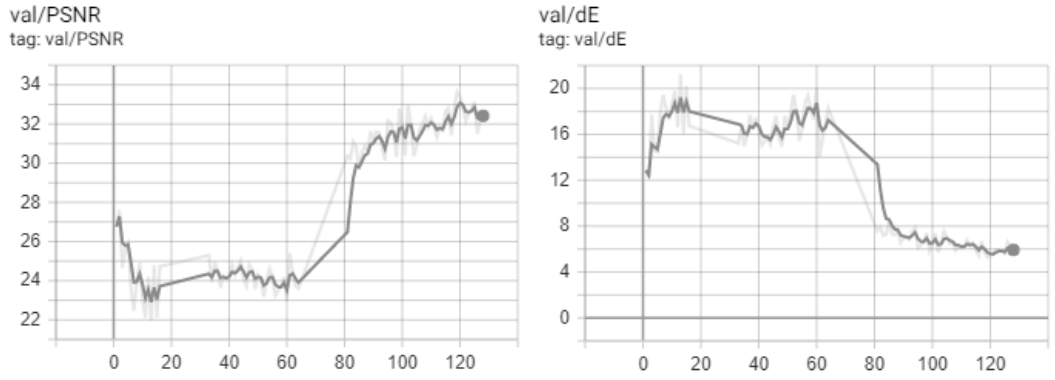


Abbildung 34: Signal-Rauschabstand (li.) und Farbabstand (re.) auf dem Validierungssatz in (iii)

Nach 64 Epochen wurde der Trainings- und Validierungssatz mit neuen Beispielen gefüllt, sodass die Kolorierung akkurater wurde, was sich in einem sprunghaften Anstieg des Signal-Rauschabstandes äußerte, der nach 128 Epochen letztendlich bei 33 dB lag.

Generell ließen sich die numerischen Resultate anhand folgender Punkte weiter verbessern:

- Verlängerung des Trainingsprozesses, da dieser in (iii) basierend auf den Kreuzentropie-Verlusten stabil erscheint
- Erweiterung des Trainingssatzes, da dieser in (iii) basierend auf dem Farbabstand nicht tolerierbar koloriert
- Erweiterung des Validierungs- und Testsatzes, da diese in (iii) zu komprimiert für eine repräsentative Auswertung erscheinen
- Ausgleich der Komplexität beider Subnetze, da diese in (iii) basierend auf den Kreuzentropie-Verlusten noch kein Gleichgewicht anstreben

Speziell letzterer Punkt ist in dieser Projektarbeit nicht gegeben, wodurch der Diskriminator neben einer verringerten Netztiefe auch weniger Komplexität mit sich bringt. Seine Verluste fallen dabei konsequent höher aus als die des Generators, was mitunter an seiner Architektur liegt. Diese beruht lediglich auf einem Decoder, anstatt analog zum Generator, als Encoder-Decoder-Struktur implementiert zu sein. Die Angleichung beider Subnetze würde daher auf Basis des Nash-Equilibriums eine Trainingskonvergenz begünstigen. Bedauerlicherweise generiert das GAN aktuell noch besser als es diskriminiert, wonach realitätsferne Kolorierungen keine Sanktionierung vom Diskriminator erfahren. Dies zeigt im kommenden Abschnitt auch die Auswertung auf dem Testsatz, dessen Vorhersage im Vergleich zum Validierungssatz den Signal-Rauschabstand um weitere 6 dB auf 27 dB reduziert. Dies impliziert eine fallende Signalqualität, für die in der folgenden qualitativen Auswertung mögliche Gründe offenbart werden.

Qualitative Evaluierung

Jene Auswertung konnte eine nahezu perfekte Kolorierung des Trainings- bzw. Validierungssatzes offenlegen, zeigte für vergleichbare Exemplare des Testsatzes aber eine zurückhaltende Kolorierung. Dies deckt sich mit dem fallenden Signal-Rauschabstand des vorherigen Abschnittes, welcher speziell durch ungesättigte Vorhersagen abfällt, da sich diese in geringen Signalstärken äußern. Jene Beobachtung zieht sich durch den gesamten Testsatz, sodass letztendlich das folgende gemischtes Fazit gezogen werden muss.

(a) Das System bildet bekannte Trainings- und teils bekannte Validierungsdaten quantitativ und qualitativ akkurat ab. Sowohl eine numerische als auch eine visuelle Auswertung bescheinigen dem System eine realitätsnahe Kolorierung.
(b) Das System bildet unbekannte Testdaten quantitativ und qualitativ inakkurat ab. Daher bescheinigt eine visuelle Auswertung dem System im Vergleich zur numerischen Betrachtung lediglich eine begrenzt plausible Kolorierung.

Beiden Resümees ist gemein, dass ihre Abbildung die ursprüngliche Struktur des Bildinhaltes rekonstruieren, die im Falle von (a) allerdings selten vom realen Farbton abweicht bzw. im Falle von (b) die reale Farbsättigung unterrepräsentiert. Ein ähnliches Resümee ziehen Nazeri et al. in [24, 5] auf Basis selbiger Testdaten. Ihr generatives Modell koloriert dabei größtenteils in Sepiafarben, was auch in der vorliegenden Arbeit teilweise zu beobachten ist. Weitere Betrachtungen schließen die bereits erwähnte Abbildung einer ungesättigten Farbwahrnehmung ein, wie in Abbildung 35 aufgezeigt wird. Neben dieser abschließenden

Darstellung kolorierter Resultate aus der Datengrundlage, lassen sich unter *github.com/tobiasvossen/B.Sc* weitere Ergebnisse begutachten.

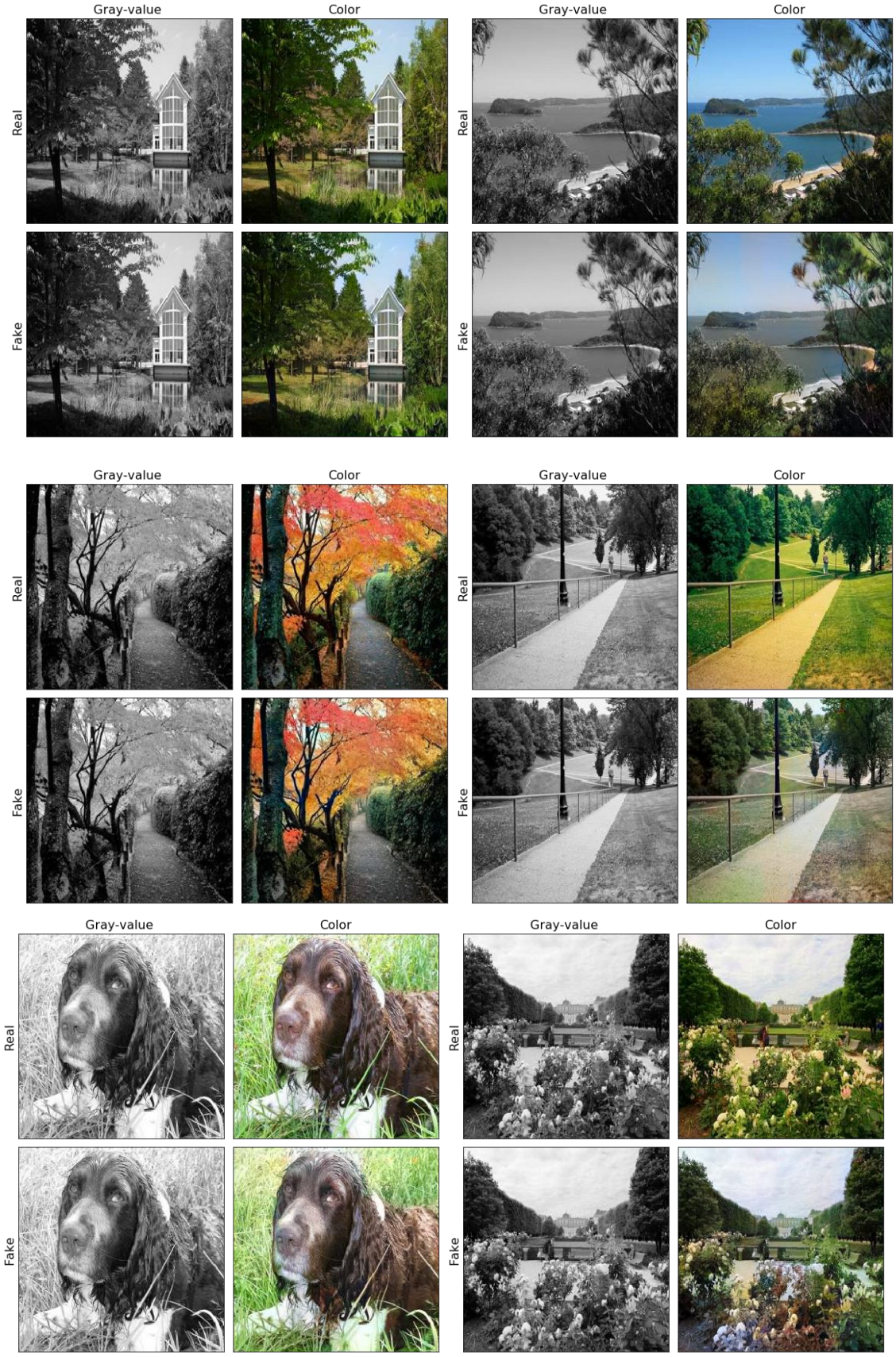


Abbildung 35 – Kolorierung von Validierungs- (li.) und Testexemplaren (re.)

6 Fazit

Die vorliegende Arbeit versuchte sich an der Kolorierung farbloser Inhalte, wie sie bspw. in der Restaurierung historischer Schwarz-Weiß-Aufnahmen benötigt wird. Dabei wurde die Vorhersage der Farbe über den Verbund zweier neuronaler Netze realisiert. Ein generatives Modell, genannt Generator, welches den monochromen Eingang auf einen polychromen Ausgang abbildet und sein differenzierender Gegenspieler, der Diskriminator, dessen Unterscheidung zwischen realem Original und generierter Kopie das Zusammenspiel beider Netze optimiert. Da jenes Zusammenspiel auf dem Min-Max-Theorem basiert, welches für gegensätzliche Intentionen seiner Mitspieler sorgt, hat sich die Bezeichnung GAN (engl. Generative Adversarial Network) durchgesetzt. Dem Generator wird dabei die Luminanz-Komponente eines realen Bildes zur Seite gestellt, die als Hilfskondition für eine deterministische Kolorierung sorgt und als cGAN (engl. Conditional Generative Adversarial Network) bekannt ist.

Als Datengrundlage standen 365 verschiedene Kategorien aus dem *Places365* Datensatz bereit, dessen Beispiele ganze Szenen beinhalten, anstatt lediglich einzelne Objekte zu repräsentieren. Über die Trennung von Luminanz- und Chrominanzanteil eines Datenexemplars wurde dem Generator ein bekanntes Frage-Antwort-Paar bereitgestellt, dessen Beziehung zueinander über ein tiefes faltendes neuronales Netz erlernt wurde. Dieses Netz wurde als Encoder-Decoder-Architektur implementiert, deren Flaschenhals über verknüpfende Stränge zwischen unterabtastendem Encoder und überabtastendem Decoder kompensiert wurde. Dahingegen wurde der Diskriminator als klassifizierender Encoder konzipiert, dessen binäre Einschätzung über den Ursprung eines zu differenzierenden Farbbildes urteilt, welche daraufhin zur Optimierung des Netzes beiträgt. Dabei pocht er lediglich auf eine fehlerfreie Klassifizierung, während sein Gegenspieler neben einer akkuraten Rekonstruktion der Chrominanz die Intention besitzt, den Diskriminator mit seiner imitierten Vorhersage zu täuschen. Anhand dieses Gegenspiels lassen sich abweichende Ziele erkennen, die es über den Optimierungsprozess auszubalancieren gilt, da ansonsten einer der Gegenspieler die Überhand gewinnt. Dafür wurden regulierende und normalisierende Maßnahmen getroffen, die letztendlich das Training stabilisieren sowie eine erfolgreiche Kolorierung sicherstellen sollen.

Jene Maßnahmen wurden mithilfe von *TensorFlow* implementiert, dessen bereitgestellte Funktionalität die volle Bandbreite an Anforderungen des maschinellen Lernens abdeckt. Von der Beschaffung einer Datengrundlage und ihrer Aufbereitung, über die Konzeption des Modells, seiner Kompilierung mittels geeigneter Zielfunktion, bis hin zu seiner Optimierung, Validierung und Verwendung, hält *TensorFlow* für jede Baustelle das passende Werkzeug bereit. Neben dem softwareseitigen Framework wurde das System in der verwandten Entwicklungsumgebung *CoLaboratory* umgesetzt, deren Funktionsumfang neben uneingeschränkter Kompatibilität zu *TensorFlow*, rechenstarke Hardware in Form von Grafikprozessoren bereitstellt, welche die intensiven Trainingsprozesse meistern. Aufgrund der quelloffenen Natur des Frameworks lassen erweiterte Funktionalitäten nicht lange auf sich warten, wodurch aktuelle Normalisierungsmethoden, dort ebenso ermöglicht werden, wie das intensive Suchen und Finden geeigneter Hyperparameter für das komplexe Modell.

Neben einer erfolgreichen Implementierung wurde das Modell abschließend analysiert und evaluiert. Dabei wurde über eine Hyperparameter-Suche eine passende Datenrepräsentation, eine komplex abbildende Modellarchitektur, sowie ein erfolgreich trainierender Optimierungsprozess gesucht und gefunden. Im Speziellen geht die Evaluierung auf den verwendeten Farbraum ein, betrachtet die Netztiefe, sowie Blöcke beteiligter Modelle und parametrisiert deren transformierende Schichten. Letztendlich trug die Optimierung des Netzes Früchte und verhalf dem generativen Modell zu einer akkuraten Rekonstruktion vertrauter Trainingsdaten, wohingegen die Abbildung unbekannter Datengrundlagen nur eingeschränkt gelang. Die plausible Kolorierung fremder Inhalte ist als oberstes Projektziel daher mit diversen Defiziten belastet, für die im folgenden Ausblick Verbesserungsansätze vorgestellt werden, die eine Kolorierung in Zukunft noch realistischer zu gestalten vermögen.

Betrachtet man zunächst die Datenebene, ließe sich der originalen Datengrundlage ein Satz kolorierter Kopien zur Seite stellen, die sich für einen alternativen Lernprozess verwenden ließen. Durch Anhängen des Datenursprungs, der entweder ein reales Original oder eine kolorierte Kopie indiziert, ließe sich die Zielfunktion des diskriminierenden Modells erweitern. Folglich würde der Diskriminator für eine fehlerhafte Einschätzung einer aus der kopierten Datengrundlage stammenden, erzeugten Kopie höher sanktioniert. Aktuell basiert jede erzeugte Kopie noch auf der realen Datengrundlage, wonach das generative Modell lediglich auf realen Daten konditioniert wird.

Ein ähnlicher Ansatz erweitert die Zielfunktion nicht innerhalb der Daten-, sondern auf Verwendungsebene. Findet der *Inception Score* bereits im Trainingsprozess Einzug, könnte dieser innerhalb der Modelloptimierung als Indikator für die Leistung des Generators herangezogen werden. Jene qualitative Auswertung wird dabei der quantitativen Zielfunktion zur Seite gestellt, was dessen Rückmeldung für die Fehlerrückführung verbessern könnte. Abweichende Vorhersagen vom Original würden so eine geringere Anpassung nach sich ziehen, sollten sie vom *Inception Score* als plausibel gewertet werden.

Auf Modellebene ließe sich zudem über Methoden des *Transfer learning* der Diskriminator durch ein bereits trainiertes Netz ersetzen, welches als Encoder weiterhin den hochdimensionalen Eingang auf einen klassifizierenden Ausgang abbildet. Jene Netze finden bspw. Anwendung in der Szenenerkennung des ursprünglich für diesen Zweck entworfenen *Places365* Datensatz. Analog ließe sich der enkodierende Teil des Generators durch ein Modell zur Objekterkennung ersetzen, dessen Gewichte bereits auf unbegrenzter Zeit- und Datengrundlage optimiert wurden.

Neben dem Ersetzen von Netzkomponenten könnte der Unterabtastung des Generators – wie in [25] – ein differenzierendes Subnetz zur Seite gestellt werden, dessen resultierende Klassenverteilung als Zusatz zum enkodierten Signal dem Decoder angehängt wird. Jener Ansatz ist vergleichbar mit der Verwendung des *Inception Score* für die Optimierung, wobei dessen Klassifizierung nach Verlassen des Generators ansetzt, während in [25] bereits im unterabtastenden Zweig das parallel klassifizierende Subnetz Verwendung findet.

In Anlehnung an das betrachtete *Mini-Batch-Learning* ließe sich der Optimierungsprozess des Diskriminators über die sog. *Mini-Batch-Discrimination* verbessern. Da diese sich schrittweise nach jedem Datenstapel optimiert, müsste der Generator im nächsten Schritt lediglich die Optimierung des Diskriminators ausgleichen, was zum bekannten *Mode collap-*

se führt, dem über eine verringerte Stapelgröße und damit durch eine häufigere Optimierung des Diskriminators im Vergleich zum Generator entgegen gewirkt werden kann [27, 3.2].

Neben konzeptionellen Schritten zur Verbesserung, ließen sich mittels *TensorFlow* weitere Funktionalitäten implementieren, welche sich speziell auf die Architektur eines GAN fokussieren. Dazu stellt *TF-GAN* als Erweiterung von *TensorFlow* aktuelle Methoden zur Verfügung, die dabei helfen, ein GAN abseits der grundlegenden Konzepte zu optimieren. Alternative Ziel- und Verlustfunktionen, wie der in [25] anteilig verwendete Wasserstein-Verlust, sind ebenso Teil dieser Erweiterung wie qualitative Auswertungen über den *Inception Score* und verwandte Metriken, die weit über die Aussagekraft numerischer Indikatoren, wie die des Signal-Rauschabstandes hinausgehen. Alle vorgestellten Verbesserungen unseres Systems ließen sich mithilfe von *TF-GAN* implementieren, sodass einer Weiterentwicklung des Systems, die in einer realistischeren Kolorierung mündet, in Zukunft nichts im Wege steht.

Anhang

A1. Grundlagen

Veröffentlichungen

System	Daten	Farbraum	Generator	Diskriminator	Zielfunktion
[24]	Places365	Lab	U-Net	PatchGAN	cGAN, L1
Pix2Pix [23]	ILSVRC	Lab	U-Net	PatchGAN	cGAN, L1
ChromaGAN [25]	ILSVRC	Lab	DCGAN Classification	PatchGAN	WGAN, L2, Classification

Tabelle 8: Grundlagen-Übersicht

Datensatz

github.com/tobiasvossen/B.Sc

A2. Implementierung

Jupyter Notebook

github.com/tobiasvossen/B.Sc

1. First steps
   i. Program parameters
   ii. Data parameters
   iii. Model parameters
   iv. Import libraries
   v. GPU acceleration
2. Data
   i. Collect data
   ii. Preprocess data
   iii. Build dataset
3. Model
   i. Building blocks
      a. Downsampling
      b. Upsampling
      c. Normalization
      d. Regularization
      e. Activation
   ii. Build model
      a. Generator
      b. Discriminator
   iii. Compile model
      a. Generator loss
      b. Discriminator loss
      c. Optimiser
   iv. Train model
      a. Loss metrics
      b. Training process
   v. Validate model
      a. Quantitative metrics
      b. Qualitative metrics
   vi. Program execution
      a. Hyperparameters
      b. Train model
4. Deployment
   i. Test model
   ii. Kolorize examples
   iii. Log model
5. Last steps

Software

Software	Version	Funktion
CoLaboratory	1.0.0	Entwicklungsumgebung
Keras	2.2.4	High-level API
Python	3.6.9	Programmiersprache
TensorBoard	2.1.0	Analysetool
TensorFlow	2.1.0	Low-level API
TensorFlow Addons	0.6.0	Hyperparameter, Normalisierung

Tabelle 9: Software-Übersicht

Generator

github.com/tobiasvossen/B.Sc

Schicht	Auflösung	Tiefe
Eingang	256 x 256	1
Unterabtastung	128 x 128	64
Unterabtastung	64 x 64	128
Unterabtastung	32 x 32	256
Unterabtastung	16 x 16	512
Unterabtastung	8 x 8	512
Verknüpfung	16 x 16	512
Überabtastung	32 x 32	256
Verknüpfung	32 x 32	256
Überabtastung	64 x 64	128
Verknüpfung	64 x 64	128
Überabtastung	128 x 128	64
Verknüpfung	128 x 128	64
Überabtastung	256 x 256	2

Tabelle 10: Generator-Übersicht

Diskriminator

github.com/tobiasvossen/B.Sc

Schicht	Auflösung	Tiefe
Eingang	256 x 256	1
Eingang	256 x 256	2
Verknüpfung	256 x 256	3
Unterabtastung	128 x 128	64
Unterabtastung	64 x 64	128
Unterabtastung	32 x 32	256
Faltung	30 x 30	1

Tabelle 11: Diskriminator-Übersicht

Abkürzungsverzeichnis

Abb. Abbildung
Adam Adaptive Moment Estimation
BN Batch Normalization
cGAN Conditional Generative Adversarial Network
CNN Convolutional Neural Network
CV Computer Vision
DCNN Deep Convolutional Neural Network
DL Deep Learning
GAN Generative Adversarial Network
GPU Graphics Processing Unit
IN Instance Normalization
JSD Jensen-Shannon-Divergenz
KLD Kullback-Leibler-Divergenz
li. links
LSGAN Least Square Generative Adversarial Network
MIT Massachusets Institute of Technology
MSE Mean Squared Error
ob. oben
PSNR Peak Signal-to-Noise Ratio
re. rechts
ReLU Rectifier Linear Unit
SGD Stochastic Gradient Descent
un. unten

Abbildungsverzeichnis

Tabellenverzeichnis

Quellenverzeichnis

[1] Bundesministerium für Bildung und Forschung, *Wissenschaftsjahr 2019 - Künstliche Intelligenz.* [Online] Verfügbar unter: https://www.wissenschaftsjahr.de/2019/. Zugriff am: 31. Januar 2020.

[2] G. E. Moore, *Cramming more components onto integrated circuits*. McGraw-Hill New York, NY, USA: McGraw-Hill New York, NY, USA.

[3] J. Dean, D. Patterson und C. Young, „A new golden age in computer architecture: Empowering the machine-learning revolution“, *IEEE Micro*, Jg. 38, Nr. 2, S. 21–29, 2018.

[4] arXiv, *Advanced Search | arXiv e-print repository.* [Online] Verfügbar unter: https://arxiv.org/search/advanced?advanced=&terms-0-operator=AND&terms-0-term=cs.CV&terms-0-field=cross_list_category&terms-1-operator=OR&terms-1-term=cs.CL&terms-1-field=cross_list_category&terms-2-operator=OR&terms-2-term=cs.LG&terms-2-field=cross_list_category&terms-3-operator=OR&terms-3-term=cs.AI&terms-3-field=cross_list_category&terms-4-operator=OR&terms-4-term=cs.NE&terms-4-field=cross_list_category&terms-5-operator=OR&terms-5-term=stat.ML&terms-5-field=cross_list_category&classification-physics_archives=all&classification-include_cross_list=include&date-filter_by=specific_year&date-year=2019&date-from_date=&date-to_date=&date-date_type=submitted_date&abstracts=show&size=50&order=-announced_date_first. Zugriff am: 31. Januar 2020.

[5] L. Huawei Technologies Co., *Huawei präsentiert die „Unvollendete“.* [Online] Verfügbar unter: https://consumer.huawei.com/de/campaign/unfinishedsymphony/. Zugriff am: 31. Januar 2020.

[6] FaceApp Inc, *FaceApp.* [Online] Verfügbar unter: https://www.faceapp.com/. Zugriff am: 31. Januar 2020.

[7] Mike Dorrance, *The creative coder adding color to machine learning.* [Online] Verfügbar unter: https://www.blog.google/technology/ai/creative-coder-adding-color-machine-learning/. Zugriff am: 31. Januar 2020.

[8] A. Ford und A. Roberts, „Colour space conversions“, *Westminster University, London*, Jg. 1998, S. 1–31, 1998.

[9] F. Chollet, *Deep Learning mit Python und Keras: Das Praxis-Handbuch vom Entwickler der Keras-Bibliothek*. MITP-Verlags GmbH & Co. KG, 2018.

[10] J. McCarthy, M. L. Minsky, N. Rochester und C. E. Shannon, „A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955“, *AI magazine*, Jg. 27, Nr. 4, S. 12, 2006.

[11] Oxford University Press, *Artificial Intelligence.* [Online] Verfügbar unter: https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095426960.

[12] Yufeng Guo, *What is Machine Learning? (AI Adventures) - YouTube.* [Online] Verfügbar unter:

https://www.youtube.com/watch?v=HcqpanDadyQ&list=PLIivdWyY5sqJxnwJhe3etaK7utrBiPBQ2. Zugriff am: 31. Januar 2020.

[13] Google Developers, *Background: What is a Generative Model?* [Online] Verfügbar unter: https://developers.google.com/machine-learning/gan/generative. Zugriff am: 31. Januar 2020.

[14] I. J. Goodfellow *et al.,* „Generative Adversarial Networks“, Jun. 2014. [Online] Verfügbar unter: http://arxiv.org/pdf/1406.2661v1.

[15] Y. Hong, U. Hwang, J. Yoo und S. Yoon, „How Generative Adversarial Networks and Their Variants Work: An Overview“, *ACM Comput. Surv.*, Jg. 52, Nr. 1, S. 1–43, 2019, http://arxiv.org/pdf/1711.05914v10.

[16] R. Rubinstein, „The cross-entropy method for combinatorial and continuous optimization“, *Methodology and computing in applied probability*, Jg. 1, Nr. 2, S. 127–190, 1999.

[17] M. Mirza und S. Osindero, „Conditional Generative Adversarial Nets“, Nov. 2014. [Online] Verfügbar unter: http://arxiv.org/pdf/1411.1784v1.

[18] X. Mao *et al.,* „Least squares generative adversarial networks“ in *Proceedings of the IEEE International Conference on Computer Vision*, S. 2794–2802.

[19] A. Levin, D. Lischinski und Y. Weiss, „Colorization using optimization“, *ACM Trans. Graph.*, Jg. 23, Nr. 3, S. 689, 2004.

[20] T. Welsh, M. Ashikhmin und K. Mueller, „Transferring color to greyscale images“ in *ACM transactions on graphics (TOG)*, S. 277–280.

[21] Z. Cheng, Q. Yang und B. Sheng, „Deep colorization“ in *Proceedings of the IEEE International Conference on Computer Vision*, S. 415–423.

[22] R. Zhang, P. Isola und A. A. Efros, „Colorful image colorization“ in *European conference on computer vision*, S. 649–666.

[23] P. Isola, J.-Y. Zhu, T. Zhou und A. A. Efros, „Image-to-image translation with conditional adversarial networks“ in *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 1125–1134.

[24] K. Nazeri, E. Ng und M. Ebrahimi, „Image colorization using generative adversarial networks“ in *International Conference on Articulated Motion and Deformable Objects*, S. 85–94.

[25] P. Vitoria, L. Raad und C. Ballester, „ChromaGAN: An Adversarial Approach for Picture Colorization“, Jul. 2019. [Online] Verfügbar unter: http://arxiv.org/pdf/1907.09837v1.

[26] J. Deng *et al.,* „Imagenet: A large-scale hierarchical image database“ in *2009 IEEE conference on computer vision and pattern recognition*, S. 248–255.

[27] Alphabet Inc., *TensorFlow Tutorials: Pix2Pix.* [Online] Verfügbar unter: https://www.tensorflow.org/tutorials/generative/pix2pix. Zugriff am: 14. Januar 2020.

[28] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva und A. Torralba, „Places: A 10 million image database for scene recognition“, *IEEE transactions on pattern analysis and machine intelligence*, Jg. 40, Nr. 6, S. 1452–1464, 2017.

[29] O. Ronneberger, P. Fischer und T. Brox, „U-net: Convolutional networks for biomedical image segmentation“ in *International Conference on Medical image computing and computer-assisted intervention*, S. 234–241.

[30] S. Ioffe und C. Szegedy, „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“, Feb. 2015. [Online] Verfügbar unter: http://arxiv.org/pdf/1502.03167v3.

[31] D. Ulyanov, A. Vedaldi und V. Lempitsky, „Instance Normalization: The Missing Ingredient for Fast Stylization“, Jul. 2016. [Online] Verfügbar unter: http://arxiv.org/pdf/1607.08022v3.

[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov, „Dropout: a simple way to prevent neural networks from overfitting“, *The journal of machine learning research*, Jg. 15, Nr. 1, S. 1929–1958, 2014.

[33] A. L. Maas, A. Y. Hannun und A. Y. Ng, „Rectifier nonlinearities improve neural network acoustic models“ in *Proc. icml*, S. 3.

[34] C. Li und M. Wand, „Precomputed real-time texture synthesis with markovian generative adversarial networks“ in *European Conference on Computer Vision*, S. 702–716.

[35] D. P. Kingma und J. Ba, „Adam: A Method for Stochastic Optimization“, Dez. 2014. [Online] Verfügbar unter: http://arxiv.org/pdf/1412.6980v9.

[36] C. Szegedy *et al.,* „Going deeper with convolutions“ in *Proceedings of the IEEE conference on computer vision and pattern recognition*, S. 1–9.

[37] P. Micikevicius *et al.,* „Mixed precision training“, *arXiv preprint arXiv:1710.03740*, 2017.

[38] T. Salimans *et al.,* „Improved techniques for training gans“ in *Advances in neural information processing systems*, S. 2234–2242.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorgelegte Abschlussarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Düsseldorf, den 3. Februar 2020

Unterschrift

(Tobias Vossen)

TH Köln
Gustav-Heinemann-Ufer 54
50968 Köln
www.th-koeln.de

Technology
Arts Sciences
TH Köln