Bachelorarbeit im Rahmen des Studiengangs
Angewandte Mathematik und Informatik

Fachhochschule Aachen, Campus Jülich

Fachbereich 9 – Medizintechnik und Technomathematik

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Super Resolution von mikroskopischen Gehirnaufnahmen via Generative Adversarial Networks

Jülich, September 2024

Oskar Druska

Eigenständigkeitserklärung

Diese Arbeit ist von mir selbstständig angefertigt und verfasst. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden.

(Ort, Datum)	(Unterschrift)

Diese Arbeit wurde betreut von:

1. Prüfer:	Prof. Dr. Jörg Striegnitz	(FH Aachen, Campus Jülich)
2. Prüfer:	Dr. Christian Schiffer	(FZJ, INM-1)

Sie wurde angefertigt am Forschungszentrum Jülich (FZJ) im Institut für Neurowissenschaften und Medizin - Funktionelle und strukturelle Organisation des Gehirns (INM-1).

Diese Arbeit dokumentiert ein Training des *Super Resolution Generative Adversarial Network* (SRGAN) (Ledig et al., 2017) und vergleicht dabei verschiedene Trainingskonfigurationen. Das SRGAN ist ein *Generative Adversarial Network*, das darauf spezialisiert wurde, niedrigaufgelöste (low-resolutionen, LR) Inputbilder hochzuskalieren (Super-Resolution, SR). Dabei wird Wert darauf gelegt, feine Details und Strukuren des hochaufgelösten Referenzoriginals (high-resolution, HR) zu rekonstruieren.

Das SRGAN wurde auf 18.000 monochromen, 256 x 256 Pixel, HR Bildern trainiert und mit 20.000 weiteren dieser Bilder validiert. Eine untersuchte Trainingskonfiguration setzt sich dabei aus Skalierungsfaktor, Epochenanzahl und Generatorverlustfunktion zusammen. Als Baseline wurde das Validierungsset auch per bikubischer Interpolation skaliert.

Die verwendeten Generatorverlustfunktionen unterscheiden sich im Qualitätsmaß, mit dem ein SR Bild bewertet wird und dessen Gewichtung. Dabei wurde untersucht, wie sich der mit dem *Very Deep Convolutional Neural Network* (VGG) (Simonyan und Zisserman, 2015) implementierte *Perceptual Loss* (meschl. wahrnehmbarer Unterschied zweier Bilder) auf das Training des SRGANs und dessen Output auswirkt.

Die vom SRGAN generierten SRGAN Bilder wurden mittels des *Contour Proposal Networks* (CPN) (Upschulte et al., 2021) validiert, indem sowohl für das HR Original und das SR Bild eine Zellsegmentierung durchgeführt wurde. Diese paarweisen Zellsegmentierungen wurden für den gesamten Validierungsdatensatz durchgeführt und anhand des Jaccard-Indizes verglichen. Für jede Konfiguration wurden diese Indizes gemittelt und so ein Jaccard-Score brechnet.

Aus den Jaccard-Scores ging hervor, dass das SRGAN, unabhängig des Skalierungsfaktors, für eine Generatorverlustfunktion, die sowohl einen Perceptual Loss per VGG als auch einen klassischen Mean-Squared-Error (MSE) berücksichtigt, die besten Ergebnisse hervorbringt. Dennoch erreicht eine bikubische Interpolation für eine zweifache Skalierung ähnliche Jaccard-Scores wie das SRGAN (ca. 80%) und bietet, da sie nicht vorher trainiert werden muss, eine brauchbare Alternative. Für eine vierfache Skalierung erreichten sowohl SRGAN als auch die bikubische Interpolation Jaccard-Scores von nur $55\% \pm 10\%pt$ und bieten daher bislang beide keine konsistente Methode zum Hochskalieren von mikroskopischen Gehirnaufnahmen.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation . . . 1
1.2 Vorhaben . . . 2

2 Hintergrund **4**
2.1 Digitale Bildskalierung . . . 4
2.2 Deep Learning . . . 6
2.3 Gradientenabstiegsverfahren . . . 8
2.4 Backpropagation . . . 9
2.5 Convolutional Neural Networks . . . 9
2.5.1 Faltung . . . 10
2.5.2 Pooling . . . 12
2.6 Generative Adversarial Networks (GANs) . . . 13
2.6.0.1 Mathematischer Hintergrund . . . 14
2.7 Super Resolution GAN . . . 15
2.7.1 Netzwerkstruktur . . . 15
2.7.2 Verlustfunktionen . . . 17
2.7.2.1 Content Loss . . . 17
2.7.2.2 Adversarial Loss . . . 18
2.8 Zellsegmentierung . . . 18
2.9 Evaluationsmethoden . . . 19
2.9.1 Metriken . . . 19

3 Datensätze **21**
3.1 Präparatgewinnung . . . 21
3.2 Datenpunkte und Datensätze . . . 21

4 Experimente **23**
4.1 Implementierung . . . 23
4.1.1 Preprocessing . . . 24
4.2 Training . . . 24
4.2.1 Generator Verlustfunktionen . . . 25
4.3 Ausgabe . . . 26

4.4 Validierung 26
4.4.1 PSNR-Score 26
4.4.2 Jaccard-Score 31
4.4.3 Vergleich mit bikubischer Interpolation 31

5 Diskussion **32**

Literatur **35**

Abbildungsverzeichnis

2.1 Bilineare Interpolation 5
2.2 Vergleich der drei Interpolationsmethoden 7
2.3 Faltung eines Bildes 11
2.4 Max-Pooling 13
2.5 SRGAN Generator 16
2.6 SRGAN Diskriminator 16

3.1 Validierungsset (Ausschnitt) 22

4.1 Bikubische Skalierung und CPN Zellmasken 27
4.2 SRGAN Output bei 2-facher Skalierung 28
4.3 2fache Skalierung, *PAPER* Loss 28
4.4 2fache Skalierung, *BOLTS* Loss 28
4.5 2fache Skalierung, *OURS* Loss 28
4.6 SRGAN Output bei 4-facher Skalierung 29
4.7 4fache Skalierung, *PAPER* Loss 29
4.8 4fache Skalierung, *BOLTS* Loss 29
4.9 4fache Skalierung, *BOLTS* Loss 29
4.10 Jaccard-Scores (IoU) 30
4.11 Peak-Signal-Noise-Ratios Scores (PSNR) 30

1 Einleitung

1.1 Motivation

Das menschliche Gehirn, sein Aufbau und die internen Prozesse sind trotz jahrhundertlanger Forschung nicht gänzlich durchdrungen. Zu dessen Aufbau gehören u. A. Zelltypen, -verteilungen, Hirnregionen und Faserstrukturen. Das Institut für Neurowissenschaften und Medizin (INM-1) des Forschungszentrums Jülich (FZJ) hat das Ziel, einen Gehirnatlas, den sogenannten *Julich Brain Atlas* [1] zu entwickeln, der solche Strukturen festhält und veranschaulicht. Er realisert ein 3D-Modell, das strukturelle und funktionelle Daten in ein gemeinsames Koordinatensystem intergiert und so eine anatomische Referenz liefert.

Zu diesem Zwecke wird die Struktur des Gehirns untersucht, in dem in histologischen Verfahren Gehirnschnitte präpariert und mit hohen Auflösungen lichtmikroskopisch eingescannt werden.

Histologie beschreibt die Wissenschaft von und Arbeit mit biologischen Geweben. Histologische Verfahren sind Prozesse, die Gewebeproben konservieren und für weitere Untersuchungen präparieren. Häufig beinhalten solche Methoden ein Schneiden und Färben der Proben. Die Färbung erhöht den Kontrast der untersuchten Gewebeproben und macht feine Strukturen sichtbarer. Oft werden Farbstoffe gewählt, die besser an gewisse Zell- oder Fasertypen binden, um genau diese in den Aufnahmen hervorheben zu können. In der Regel ist dies sogar notwendig, da viele Gewebetypen nicht genug Eigenfärbung aufweisen, um unter Lichtmikroskopen deutlich erkennbar zu sein. Im Bezug auf den Julich Brain Atlas werden so also Zellen und Nervenbahnen hervorgehoben, die daraufhin digitalisiert und in das 3D-Modell integriert werden.

Ein ganzes Gehirn, welches dann aus vielen Tausenden histologischen Schnitten besteht, zu digitalisieren erfordert hohe Präzision und verursacht einen enormen zeitlichen Aufwand. Daher geschieht die Digitalisierung der histologischen Präparate im INM-1 nicht manuell sondern mit *high-throughput light-microscopic* Scannern, die, neben der sehr hohen Auflösung der Optiken ($1\mu m/\text{px}$), auch in der Lage sind, selbstständig Schnittpräparate einzulegen und auszutauschen. Diese können also "unbemannt" – rund um die Uhr – Gehirnaufnahmen machen. Dabei kommen mehrere dieser Hochdurchsatzmikroskope zum Einsatz, die zusammen ca. ein Jahr für die Digitalisierung eines ganzen Gehirns benötigen. Ein Gehirn umfasst

dabei ca. 6000 bis 8000 Schnitte á $20\mu m$ Dicke. Die Datenmenge eines solchen eingescannten Gehirns beträgt ca. 50 Terabyte.

Näheres zu den in dieser Arbeit verwendeten Präparaten und deren Herstellung in Kapitel 3.

Die Herstellung dieser $20\mu m$ dicken Gehirnschnitte verlangt äußerste Präzision und verursacht trotz höchster Sorgfalt unvermeidbare Risse und Verschiebungen im untersuchten Gewebe. Diese Prozessartefakte verursachen Ungenauigkeiten in den aus den Aufnahmen generierten Modellen und sollen daher korrigiert werden. Das *Image Inpainting* [1] (Kropp et al., 2023) [2] ist eine Machine Learning Methode um solche Prozessartefakte auszubessern. Das Image Inpainting verwendet die in Ho, Jain und Abbeel, 2020 [3] untersuchten *Diffusion Probabilistic Models* um Prozessartefakte mit Diffusionsmodellen auf eine iterative Art auszubessern.

Der Laufzeitanspruch dieser Korrekturberechnungen steigt mit der Auflösung der zu korrigierenden Gehirnaufnahmen und nimmt bei einer Auflösung von $1\mu m/\text{px}$, abhängig der Größe der Artefakte (kleine Risse bis Verschiebungen ganzer Sektionen) viele Stunden bis wenige Tage in Anspruch. Um diesem Laufzeitaufwand entgegenzuwirken sollen die Bilder, auf die das Inpainting angewendet werden soll, zunächst in ihrer Auflösung reduziert (engl. *Downsampling*), dann durch das Inpainting ausgebessert und anschließend wieder hochskaliert (engl. *Upsampling*) werden. Ziel ist es, beim Upsampling feine Details und Strukturen, die durch das Downsampling verlorengegangen sind, wiederherzustellen.

Dieser Prozess, ein Bild in seiner Auflösung zu erhöhen, ohne dabei auf klassische Interpolationsmethoden (s. 2.1) zurückzugreifen, nennt sich *Super Resolution*. Der Fokus liegt dabei darauf, die feinen Strukturen des originalen, hochaufgelösten Bildes aus den niedrigaufgelösten Bildern abzuleiten.

1.2 Vorhaben

In dieser Bachelorarbeit werden *Generative Adversarial Networks* (GANs), Bildgenerierungsmodelle des Deep Learnings, erläutert (s. Abschnitt 2.6). Der Fokus liegt dabei auf dem *Super Resolution GAN* (SRGAN) [4], einer Spezialisierung der GANs, welches mit Blick auf Super-Resolution konstruiert wurde (s. Abschnitt 2.7).

Netzwerke im Sinne des Deep Learnings (auch neuronale Netze oder Modelle) sind Algorithmen, die Eingabedaten (häufig Bilder oder Audios) durch eine Vielzahl an Matrix- und Vektormultiplikationen und -operationen auf Ausgabewerte (Klassifikationen, Wahrscheinlichkeitsverteilungen, neue Bilder oder Audios, ...)

[1] In dieser Arbeit werden in vielen Fällen, angeglichen an die überwiegend englischsprachige Literatur, englische Fachbegriffe verwendet, insbesondere dann, wenn es für diese keine eindeutige deutsche Übersetzung gibt.

abbilden. Diese Abbildungen sind dabei von Millionen von Parametern abhängig, welche anhand von gegebenen Datensätzen adjustiert werden. Dieses Adjustieren nennt sich *Training*.

Des Weiteren werden verschiedene Trainingskonfigurationen präsentiert (s. Abschnitt 4.2) und im Rahmen einer *Gittersuche* miteinander verglichen (s. Abschnitt 4.4). Eine Gittersuche prüft mittels eines Validierungsscores (einem numerischen Wert, der die Qualität der Ausgabe eines Neuronalen Netzes quantifiziert) systematisch verschiedene Konfigurationen mit dem Ziel, unter diesen ein Optimum zu finden. Die Validierung dieser geschieht über ein drittes Modell, dem *Contour Proposal Network* (CPN), das für ein Inputbild eine Zellsegmenteirung (s. Abschnitt 2.8) durchführt und welches bereits trainiert gegeben ist. Für das Training des SRGANs wird ein histologischer Datensatz des INM-1 verwendet, der dem Okzipitallappen des *BigBrains* [5] entnommen ist. Näheres dazu in Kapitel 3.

2 Hintergrund

Dieses Kapitel erläutert zunächst klassische Bildskalierungsmethoden sowie grundlegende Konzepte des Machine Learnings und stellt insbesondere die verwendeten SRGAN und CPN Modelle vor. Weiterhin werden die zur Bewertung des SRGAN-Trainings verwendeten Evalutationsmetriken präsentiert.

2.1 Digitale Bildskalierung

Ein Bild im Sinne der digitalen Bildverarbeitung ist eine 2- oder 3-dimensionale Matrix an Integerwerten. Jeder Eintrag der Matrix repräsentiert ein Pixel, der Integerwert stellt i. d. R. den Helligkeitswert des Pixels dar. Graustufenbilder (allgemein monochrome Bilder) werden durch eine 2-D Matrix dargestellt. In digitalen Farbbildern wird jedes Pixel durch ein 3-Tupel an Helligkeitswerten dargestellt. Die Einträge des 3-Tupel repräsentieren die Helligkeiten verschiedener Farbkanäle, häufig Rot, Grün und Blau. Die in dieser Arbeit verwendeten Bilder sind monochrom und verwenden eine Pixelauflösung von 8 bit. Jedes Pixel nimmt also einen von $2^8 = 256$ Helligkeitswerten an. 0 liest sich dann als schwarz, 255 als weiß. Bei der Darstellung auf einem Computerbildschirm werden diese Farbintensitäten auf die Leuchteinheiten des Bildschirms abgebildet.

Digitale Bildskalierung beschreibt den Prozess, ein digitales Bild in seiner Größe zu verändern. Dies ist z. B. nötig, um einen Benutzer an einem Computerbildschirm zum Beispiel in das Bild rein- und rauszoomen lassen zu können, da die zugrundeliegende Bildmatrix auf eine größere oder kleinere Pixelfläche abgebildet werden muss. Skalierung wirkt sich auch auf den Speicherbedarf des Bildes aus. Hochaufgelöste Bilder sind speicherintensiver und beinhalten mehr Information als niedrigaufgelöste Bilder.

Beim *Hochskalieren* werden die Einträge eines Bild auf ein größeres Bild abgebildet. Dabei werden die Einträge des originalen Bildes auf die des neuen, größeren Bildes gestreckt. Die Bestimmung der neuen, jetzt undefinierten Bildpixel ist von der gewählten Skalierungsmethode abhängig. Die Erscheinung, insbesondere auch die menschliche Wahrnehmung des größeren Bildes ist von der verwendeten Skalierungsmethode abhängig. *Herunterskalieren* beschreibt das Verfahren, die Bildpixel auf ein kleineres Bild abzubilden. Das Bild wird kleiner und verliert dabei an Information.

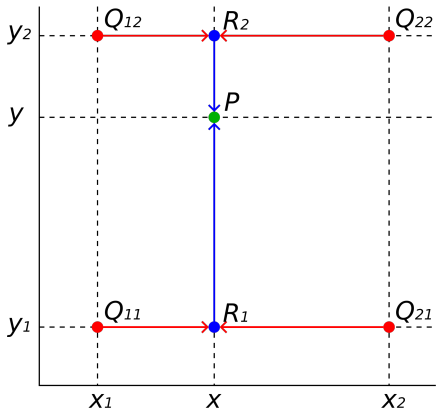


Abbildung 2.1: Bilineare Interpolation; P ist das neue bislang unbekannte Pixel. Zunächst werden die umliegenden, bekannten Pixelwerte Q_{ij} gewichtet mit Abstand zu P in X-Richtung gemittelt. Die so bestimmten Zwischenergebnisse werden erneut, diesmal in Y-Richtung gemittelt. Die Reihenfolge der Achsenmittelung ist kommutativ.

[6]

Im Folgenden werden verschiedene Methoden zum Hoch- und Herunterskalieren vorgestellt. Die drei bekanntesten Methoden zur digitalen Bildskalierung sind die *Pixelwiederholung*, die *bilineare Interpolation* und die *bikubische Interpolation*.

Die *Pixelwiederholung* (auch als Nearest-Neighbour-Interpolation bekannt) ist die simpelste der zweidimensionalen Interpolationsmethoden. Ein zu interpolierendes Bildpixel erhält den Wert seines nächsten, bekannten Pixels. Andere umliegende Bildpixel werden dabei nicht näher betrachtet

Bei der *bilineare Interpolation* werden die neu berechneten Pixelwerte anhand ihres Abstand zu anderen, aus dem Original bekannten Bildpixeln berechnet. Die bekannten benachbarten Pixelwerte werden ihrem Abstand zum neuen Pixel entsprechend gewichtet, aufsummiert und gemittelt. Das Ergebnis ist ein Bild, das unschärfer, aber auch “weniger kantig” wirkt. Abb. 2.1 zeigt eine grafische Darstellung des Interpolationsprozesses.

Die *bikubische Interpolation* ist einer Erweiterung der kubischen Splines und modifiziert die bilineare Interpolation dahingehend, dass nicht nur die direkten umliegenden Nachbarn in die Berechnung eines neuen Pixels einfließen.

Kubische Splines sind eine Interpolationsmethode, welche einen Funktionszug

aus Polynomen 3. Grades durch einen Datensatz legt. Besonders an den kubischen Splines ist, dass die Teilpolynome an den Abschnittsgrenzen nicht nur die gleichen Funktionswerte, sondern auch gleiche erste und zweite Ableitungen aufweisen. Sie sind also zweifach stetig differenzierbar. Sie haben den Vorteil, dass der aus der Interpolation resultierende Polynomgrad unabhängig von der Anzahl der bekannten Funktionswerte immer 3 sein wird.

Die bikubische Interpolation konstruiert also eine Fläche auf Basis eines gegebenen Gitternetzes an bekannten Funktionswerte, welche an jedem Punkt erneut zweifach differenzierbar sein wird. Die Ergebnisse gelten als glatter als die der bilinearen Interpolation.

Die vorausgegangen Skalierungsmethoden unterscheiden sich in Laufzeitaufwand und Glätte der Ergenisse. Im Allgemeinen gilt: je glatter die Ergebnisse desto höher der Laufzeitaufwand. Zum Vergleich der visuellen Ergebnisse, siehe Abb. 2.2.

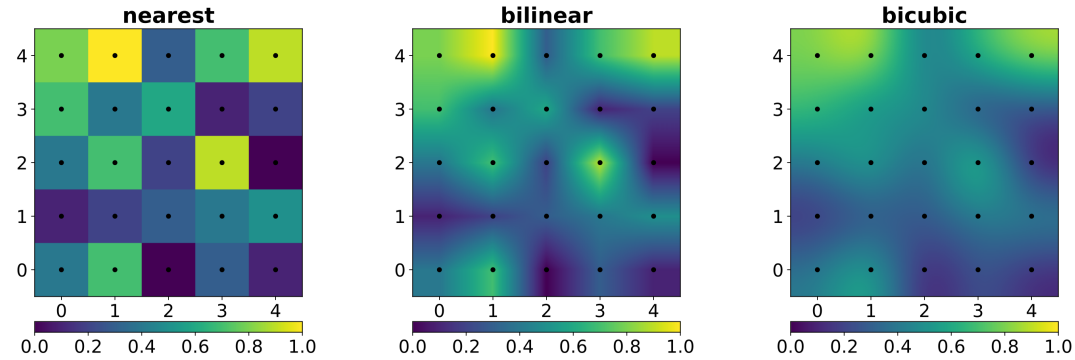
Super Resolution (SR) beschreibt eine Klasse an Deep Learning Algorithmen, welche, genau wie die zuvor gezeigten Interpolationsmethoden, Bilder hochskalieren. Die Algorithmen der Super Resolution grenzen sich zu denen der Interpolationen aber insofern ab, dass sie beabsichtigen, Information, die im gegebenen Bild nicht vorhanden ist, zu rekonstruieren. Information meint z. B. feine Details oder Texturen, die durch ein voriges Herunterskalieren, oder durch limitierte Auflösung der Photosensoren, verloren gegangen sind.

Dabei wird zwischen *Single Image Super Resolution* (SISR) und *Multi-frame Super Resolution* (MFSR) unterschieden. Die SISR betrachtet das hochzuskalierende Bild einzeln und berücksichtigt dabei keine Informationen aus evtl. ähnlichen oder "benachbarten" Bildern (z. B. bei Videos). Konträr dazu arbeitet die MFSR mit mehrerern Bildern, sowohl als Input als auch beim Output.

Diese Arbeit beschäftigt sich mit SISR, in der Absicht ein Neuronales Netzwerk zu trainieren, um ein einzelnes Bild in seiner Auflösung zu verbessern.

2.2 Deep Learning

Deep Learning, als Teilgebiet des Machine Learnings, umfasst eine Reihe an Algorithmen und Methoden und wird häufig dann angewendet, wenn Datenpunkte eines potenziell großen Eingaberaums (Bilder, Audio) verarbeitet werden. Charakteristisch ist, dass Eingabedaten im Vorhinein so wenig wie möglich vorverarbeitet werden sollen. Häufig werden Bilder oder Audios als Ganzes gelabelt (z. B. was abgebildet ist oder gehört werden kann) ohne dabei näher auf Lokalität oder Zeitpunkt des Targets auf dem Datenpunkt einzugehen. Die Bezeichnung als "tiefes" Lernen reicht daher, dass die für diese Anwendungen verwendeten neuronalen Net-



[7] [6] [8]

Abbildung 2.2: Vergleich drei verschiedenener Bildskalierungsmethoden. Die schwarzen Punkte sind die Bildpixel, die aus dem Original bekannt waren. Die farbigen Flächen wurden mittels verschiedener Interpolationsmethoden berechnet.

ze in der Regel viele Schichten und umso mehr trainierbare Parameter aufweisen. [9]

Ähnlich wie beim klassischen Machine Learning werden Inputparameter auf Vorhersagen abgebildet. Blumentypen könnte man z. B. anhand von Farbe, Form oder Blattgröße klassifizieren. Der Bumentypus ist dann ein *Target*, die Ausprägungen sind *Features*. Beim Deep Learning stellt dann jedes Pixel/Bit des Inputs ein Eingabefeature dar. Methoden des Deep Learnings müssen in der Lage sein, in diesen "rohen" Inputvektoren, Muster und Zusammenhänge zu erkennen, um daraus stärker abstrahierte Features ableiten und weiterverarbeiten zu können.

Die Fähigkeit, höhere Features zu abstrahieren, wird in die Netzwerkstruktur eines Models eingearbeitet und muss aufgrunddessen während des Trainings auch adjustiert werden. Häufig werden hochdimensionale Inputdaten zunächst verkleinert und die Informationen aus verschiedenen Bildregionen in sogenannten *Feature Maps* zusammen gefasst (s. Faltungsnetzwerke Unterabschnitt 2.5.1). Diese sind einfacher zu verarbeiten. Diese Reduktion der Inputdimension ist parametrisiert und häufig Teil des Trainings eines Deep Learning Modells. Dies erfordert im Allgemeinen ein ausgedehnteres (mehr Daten, mehr Epochen) Training und im Zuge dessen eine größere Menge an Trainings- und Validierungsdaten, als simplere, niedrig-dimensioniertere Modelle.

2.3 Gradientenabstiegsverfahren

Das Gradientenabstiegsverfahren (auch simpel: der Gradientenabstieg) ist eine Methode der Numerik um Optimierungsprobleme, insbesondere die Parameterfindung in neuronalen Netzen, zu lösen. $E(\theta)$ ist dabei der vom mit θ parametrisierten Modell verursachte Fehler, also der Unterschied zwischen der Vorhersage des Modells und dem gewünschten Ergebnis. Die Parameter θ des Modells (auch "Gewichte" genannt) sollen so gewählt werden, dass der Fehler E minimal wird. Dazu werden die Modellparameter θ entlang des Gradienten $\nabla E(\theta)$ verschoben:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla E(\theta) \tag{2.3.1}$$

Der Parameter $\eta > 0$ nennt sich *Lernrate* und bestimmt damit die Schrittweite des Gradientenabstiegs. [10]

Da beim Training eines Modells mehrere ($|\mathcal{N}|$) Datenpunkte zum Einsatz kommen, und das Netzwerk auf all Jene hin optimiert werden soll, hat die Fehlerfunktion des Netzwerks die folgende Form:

$$E(\theta) = \sum_{n \in \mathcal{N}} E_n(\theta) \tag{2.3.2}$$

Dabei ist $E_n(\theta)$ die Diskrepanz zwischen der Vorhersage des Netzwerks auf dem Datenpunk n und dem gewünschten Ergebnis. Die Auswertung des Gradienten erweist sich hier als rechenaufwändiger, da dieser für jeden Datenpunkt n bestimmt werden müsste. Besonders für große $|\mathcal{N}|$ ist dies zeitintensiv. Es gibt allerdings eine Variante des Gradientenabstiegs, welche den Gradienten pro Iteration für nur einen Datenpunkt $k \in \mathcal{N}$ bestimmt und anhand dessen ein Update des Inputs vornimmt:

$$x^{(t+1)} = x^{(t)} \pm \eta \nabla E_k(x) \tag{2.3.3}$$

Dieser Variante wird *stochastischer Gradientenabstieg* genannt, weil der Datenpunkt k dem Datensatz zufällig entnommen wird. Der stochastische Gradientenabstieg nähert sich dem Minimum i. A. nicht auf direktem Wege, ist aber pro Iteration schneller zu berechnen und weniger anfällig für Duplikate im Datensatz.

Ein Kompromiss der beiden Varianten ist der *Mini-Batch Gradientenabstieg*, der das Trainingsset zufällig in b-große Subsets (*Batches*) (häufig 32, 64 ode 128 Datenpunkte pro Batch) aufteilt und den Gradientenabstieg für je einen dieser Batches durchführt. Das verringert den Rechenaufwand, den ein "großer", allumfassender Gradientenabstieg hätte ohne dabei zu kleinschrittig zu sein, wie es beim stochastischen Gradientenabstieg häufig der Fall ist. Der Mini-Batch Gradientenabsteig konvergiert im Allgemeinen schneller als der stochastische Gradientenabsteig und ist pro Gradientenberechnung schneller, da dieser nicht für den gesamten Trainingsdatensatz berechnet werden muss.

2.4 Backpropagation

Die *Backpropagation* ist ein Algorithmus zur schrittweisen Berechnung des Gradienten einer Netzwerkfehlerfunktion, der zur Optimierung der Netzwerkparameter benötigt wird.[10] Die iterative Auswertung von Fehlern und die dahingehende Anpassung der Netzwerkparameter verhält sich analog zur *Forwardpropagation* von Inputparametern, in dem Fehlerinformationen (Output vs. Target) in einer Rückwärtsbewegung durch das Netzwerk getragen werden. Forwardpropagation beschreibt den Prozess, einen Datenpunkt in ein Modell zu geben und auf Basis dessen eine Vorhersage zu berechnen.

Die Backpropagation allein ist keine Methode zur Minimierung einer Netzwerkfehlerfunktion E_n. Sie bietet aber eine Möglichkeit der effizienten Auswertung aller partiellen Ableitungen bzgl. der Parameter w_{ji} und damit des Gradienten ∇E_n.

Die ursprünglich von Rumelhart et al. (1986) konstruierte Backpropagation funktioniert wie folgt:

1. Berechne für einen Inputvektor $\mathbf{x}_n$ die Vorhersage des Modells.

2. Bestimme die δ_k als $\delta_k = y_k - t_k$.
   Dabei ist y_k die k-te Komponente der Modellvorhersage und t_k die k-te Komponente des gewollten Ergebnisses.

3. Bestimme alle δ_j als $\delta_j = \frac{d}{dx}(h(a_j)) \sum_k w_{kj}\delta_k$

4. Bestimme alle als $\frac{\partial E_n}{\partial w_{ji}}$ als $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$

2.5 Convolutional Neural Networks

Convolutional Neural Networks (dt. Faltungsnetzwerke; folgend: CNNs) sind eine Unterart der Neuronalen Netze und darauf spezialisiert, Bilder zu verarbeiten.

Datenpunkte wie z. B. Bilder haben die Eigenschaft, dass ihre *Features* (Pixel) eine hohe Lokalität aufweisen. Das heißt, dass Features, die räumlich nah beieinander liegen, stark zueinander korrelieren. Diese Eigenschaft machen sich CNNs zu Nutze, in dem sie die Abbildung der *Faltung* anwenden, die immer gleich ein Pixel und seine Umgebung in Betracht zieht.

Faltungen bilden mehrere Pixel auf Eines ab und können den Effekt haben, dass das Outputbild kleiner als das Inputbild wird. Dies erweist sich deshalb als nützlich, da Bilder aufgrund ihrer oft (annährend) quadratischen Anzahl an Inputpixeln, große Modellstrukturen und in Folge dessen viele Parameter fordern, um angemessen verarbeitet werden zu können. Ein Bild der Größe 32x32 weist schon 1024 Inputparameter auf; 3072 wenn das Bild farbig ist und aus je 3 Farbkanälen

pro Pixel besteht. Schon ein simple gewichtete Summe hätte für ein Bild dieser Größe 1024 (bzw. 3072) adjustierbare Parameter.

Das Ergebnis einer Faltung in einem neuronalen Netzwerk wird *Feature Map* genannt.

2.5.1 Faltung

In der Analysis ist die *Faltung* (auch *Konvolution*) ein Operator, der zwei Funktionen/Folgen (respektive kontinuierlich/diskret) auf eine Dritte abbildet. [9]

Für den kontinuierlichen Fall, also die Faltung zweier Funktionen f und g, ist diese wie folgt definiert:

$$(f * g)(x) := \int_{R^n} f(\tau)g(x-\tau)d\tau \tag{2.5.1}$$

Die Faltung ist der Flächeninhalt der sich deckenden Funktionen f und g, wenn g der x-Achse entlang "über" f geschoben wird. Die Faltung ist kommutativ.

Der diskrete Fall (das Falten zweier Folgen a, b) funktioniert analog, in dem man Folge a entlang Folge b "schiebt", die sich jeweils deckenden Folgenelemente multipliziert und alle Produkte aufsummiert. Anschließend wird Folge b "weiter geschoben".[11]

Die Faltung lässt sich auf höhere Dimensionen erweitern; Die feste Folge a ist dann das *Bild*, die bewegte Folge b nennt sich *Kernel*.

Das Verschieben des Kernels K über das Bild I ist in Abb. 2.3 visualisiert. Die Faltung im diskreten Fall lässt sich anschaulich mit dem Konzept eines *Sliding Windows* vergleichen.

Faltungen auf Bildmatrizen "scannen" einen Bereich nach einem über den Kernel definierten Feature ab. Ein Scan meint dabei das Ergebnis der Multiplikation von Bild I und Kernel K an einer festen Position des Kernels. Dabei definieren *Maße*, *Einträge* und *Schrittweite* des Kernels, welches Muster gesucht und wie deutlich es vorhanden sein muss, damit es auf der Feature Map zu sehen sein wird. Die Werte der Feature Map C geben Aufschluss darüber, ob ein Feature im Bild gegeben ist und wenn ja, wo und wie ausgeprägt.

Die *Einträge* des Kernels charakterisieren die Transformation, die auf dem Bild ausgeführt wird.

Beispiele: Die Faltung eines Bildes mit einem Kernel der Form NxN, dessen Einträge alle $\frac{1}{N^2}$ sind, resultiert in einer Art lokalem Mittelwert. Die Feature Map C ist eine Modifikation des Bildes I welche unschärfer wirkt.

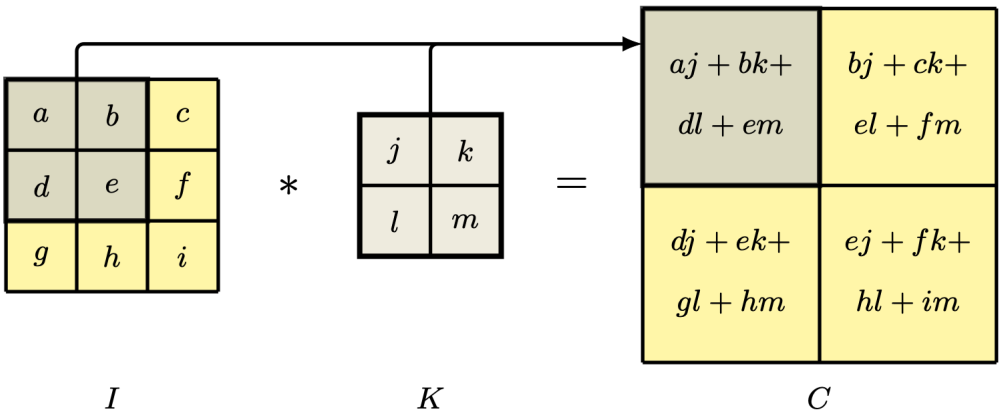


Abbildung 2.3: Faltung $I * K$ von Bildmatrix I und Kernel K [9]

Faltungen können ebenso verwendet werden, um bestimmte Strukturen in Bildern ausfindig machen zu können. Die Wahl der Einträge des Kernels K charakterisiert dabei die Struktur, nach der ein Bild I "gescannt" wird.

Ein Kernel K mit

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{2.5.2}$$

detektiert horizontale Änderung in Pixelwerten, also Kanten, wenn gefaltet mit einem Bild I.[9]

Die *Maße* W, H des Kernels bestimmen das "Sichtfeld" der Faltung. Je Größer der Kernel, desto mehr Information des Gesamtbildes sind in einem Pixel des Outputbildes enthalten. Gleichbleibend große Strukturen werden durch größere Kernels weniger deutlich erkannt.

Die *Schrittweite* S bestimmt die Abtastrate der Faltung. Die eigentliche Faltung setzt diese auf 1, der Kernel wird also nach jedem Scan um genau ein Pixel verschoben. Setzt man diese $S > 1$ so wird die Faltung "flüchtiger". Features müssen also stärker ausgeprägt sein, um im gleichen Maße auf der Feature Map erkennbar zu sein. Dies hat den Effekt, dass die resultierende Feature Map auch kleiner wird. Eine Schrittweite S gleich der Kernelbreite/-länge W/H hätte zur Folge, das jedes Bildpixel nur einmal gefaltet wird.

Die *Kerneleinträge* der Faltung sind lernbar und Teil des Trainings eines CNNs. Sie machen einen Großteil der CNN Trainingsparameter aus und stellen so die Muster und Strukturen dar, die das CNN zu erkennen lernt. *Kernelmaße* und *Schrittweite* sind i. d. R. fest.

2.5.2 Pooling

Faltungsoperationen auf Bildmatrizen sind stark von der Lokalität der gesuchten Features abhängig. Verändert das Feature (z. B. ein Gesicht) seine Position auf dem Bild, so wird sich auch die entsprechende Aktivierung auf der Feature Map bewegen.

Für Anwendungen wie z. B. der Klassifikation (*Ist ein Gesicht auf dem Bild oder nicht?*) ist es wünschenswert, wenn solche Änderungen der Lokalität nur geringe bis keine Auswirkungen auf die Feature Map haben.[9]

Pooling ist eine der Faltung ähnliche Operation, welche diese Eigenschaft zum Ziel hat. Das Pooling behält das *Sliding Window* der Faltung bei und ersetzt die Einträge des Kernels durch eine Funktion. Diese und ihre Parameter sind fest und werden während des Trainings auch nicht modifiziert. Ein übliches Beispiel ist das *Max-Pooling*, dessen Kernel den größten Wert seines gescannten Bereiches zurück gibt (s. Abb. 2.4).

Auch wird der Pooling-Kernel beim "scannen" so über das Bild geschoben, dass die gescannten Bereiche sich nicht überlappen. Die Schrittweite des Poolings entspricht also der Kernelweite.

Pooling reduziert die Auflösung des Inputs, da die resultierende Feature Map kleiner ist, als das gefaltete Bild.

Angenommen, eine vorangegangene Faltung produziert eine Feature Map, deren Aktivierungen angeben, wie stark ein gemessenes Feature im gefalteten Bild vorhanden war. Die Anwendung des Max-Poolings auf diese Featuremap erhält die Information über stark präsente Features, verwirft aber (in Teilen) deren Lokalität. Pooling erlaubt weiterhin, Inputs variabler Größe auf eine feste Outputgröße zu reduzieren, in dem die Weite des Poolingkernels in Abhängigkeit der Form des Inputbildes gewählt wird.

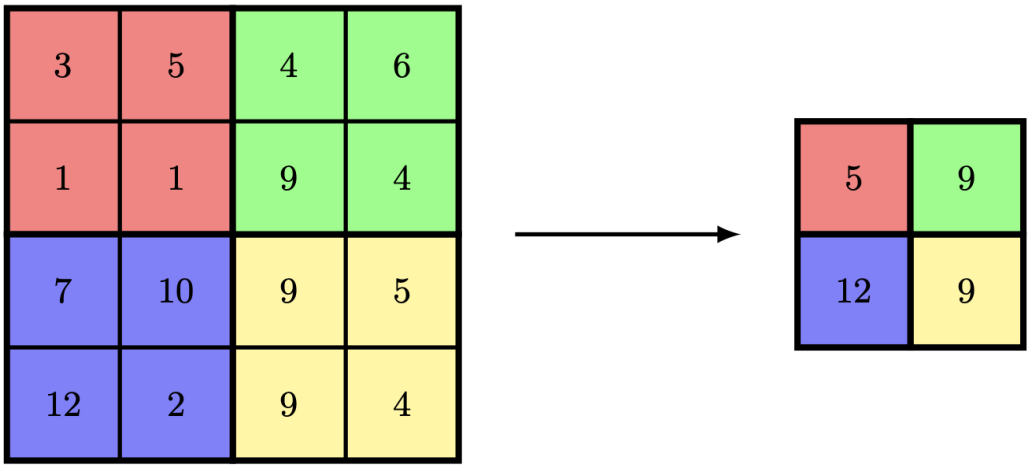


Abbildung 2.4: Max-Pooling
Der Kernel gibt den jeweils größten Wert seiner Komponenten zurück (farbig veranschaulicht)
[9]

2.6 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (adversarial - dt. gegnerisch; GANs) sind eine Spezialisierung der Faltungsnetze, die darauf optimiert sind, Bilder, die einem Inputtypus entsprechen, zu generieren. Sie wurden 2014 von Goodfellow et al. konzipiert. [12]

In einem GAN werden ein *Generator* und ein *Diskriminator* in einem *Nullsummenspiel* gegenübergestellt. Sowohl der Generator als auch der Diskriminator sind neuronale Netze und können unabhängig voneinander betrieben werden. Sie haben ihre eigenen Parametersets und können ggf. getrennt voneinander trainiert werden.

Ein *Nullsummenspiel* ist ein Begriff der *Spieltheorie* und beschreibt ein Spiel, in dem sich die Gewinne und Verluste der einzelnen Spieler gegenseitig wegheben. Ein Spiel im Sinne der Spieltheorie ist ein Szenario in dem sich mind. 2 Parteien (Spieler) in einer Konfliktsituation gegenüber stehen und deren Handlungen nicht nur vom eigenen Zustand sondern auch von denen der Mitspieler beeinflusst werden.

Die Motivation des Generators ist, Datenpunkte zu synthetisieren, welche denen des Trainingssets “ähnlich” sind; semantisch gleichen, um damit einen Diskriminator zu täuschen. Aufgabe des Diskriminators ist also, für jeden Datenpunkt, der ihm präsentiert wird, die Entscheidung zu fällen, ob dieser dem Trainingsset entstammt, also *echt* ist, oder vom Generator erstellt, also *falsch* ist. Er soll lernen,

echte von falschen Datenpunkten unterscheiden zu können.

Insofern Generator und Diskriminator als zwei miteinander verbundene Modelle realisiert sind, können beide mittels *Backpropagation* (s. Abschnitt 2.4) anhand des Diskriminatoroutputs trainiert werden.

Der folgende Abschnitt fasst die Erkenntnisse Goodfellows et al. [12] zusammen.

2.6.0.1 Mathematischer Hintergrund

Im Machine Learning wird angenommen, dass alle Eingabedaten einer unbekannten Wahrscheinlichkeitsverteilung entspringen. Ziel des GAN-Trainingsprozesses ist es, diese Verteilung zu approximieren.

Der Generator $G(z;\theta_g)$ wird als stetige Funktion modelliert, welche aus einem Rauschen z und den Gewichten θ_g einen Datenpunkt generiert. z wird dabei einer Rauschverteilung $p_z(z)$ entnommen. Häufig wird dafür eine Normal- oder Gleichverteilung verwendet.

Weiterhin modelliert $D(\mathbf{x};\theta_d)$ einen Diskriminator, der anhand des Inputs $\mathbf{x}$ und den Gewichten θ_d den gegebenen Datenpunkt als *echt* oder *falsch* klassifiziert. $D(\mathbf{x})$ repräsentiert die Wahrscheinlichkeit, dass $\mathbf{x}$ dem Trainingsset entstammt, also *echt* ist. $D(G(z))$ ist also die Wahrscheinlichkeit, dass ein *falscher* Datenpunkt als *echt* klassifiziert wird.

Sowohl Generator G als auch Diskriminator D werden als CNNs realisiert. Aufgrunddessen ist Backpropagation von der Diskriminatorklassifikation bis zum Inputlayer des Generators möglich.

Aus der obigen Modellierung ergibt sich folgende Verlustfunktion:

$$V(D,G) = E_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}))] + E_{\mathbf{z}\sim p_z(\mathbf{z})}[log(1 - D(G(\mathbf{z})))] \tag{2.6.1}$$

G versucht diesen Verlust zu minimieren; D zu maximieren.

V wird minimal, wenn $D(\mathbf{x}) = 0$, also *echte* Datenpunkte $\mathbf{x}$ als *falsch* klassifiziert werden und vice-versa wenn $D(G(\mathbf{z})) = 1$, also *falsche* Datepunkte $G(\mathbf{z})$ immer als *richtig* klassifiziert werden. V wird maximal, wenn $D(\mathbf{x}) = 1$ und $D(G(\mathbf{z})) = 0$, Datenpunkte also immer entsprechend ihres Ursprungs korrekt klassifiziert werden.

Das Erreichen des Maximums würde bedeuten, dass der Generator sich anhand des Feedbacks des Diskriminators nicht mehr verbessern kann, das dieser immer richtig klassifizieren wird. Das Erreichen des Minimums hätte den gleichen Effekt zu Folge, da jeder Output des Generators stets gegeläufig vom Diskriminator klassifiziert wird.

Der Generator wird verschiedene Strategien zur Bildgenerierung testen und diese anhand der Verlustfunktion bewerten. Reicht dieser Verlust nun zu früh im Training an einen Extremwert, kann der Generator diesen nicht mehr zur akkuraten Bewertung seiner Strategien nutzen. Das Training kollabiert.

Aufgrunddessen wird eine Balance von $D(\mathbf{x}) = \frac{1}{2}$ angestrebt, bei der der Diskriminator nur willkürlich über die Authentizität des Inputs entscheiden kann. Dies bedeutet, dass der Diskriminator die Authentizität der Bilder "raten" muss. Dies garantiert aber nicht, dass der Generator nach menschlichem Urteilsvermögen "gute" Bilder erzeugt.

2.7 Super Resolution GAN

Das *SRGAN* (Super Resolution Generative Adversarial Network) ist ein GAN (Abschnitt 2.6) welches für die Aufgabe spezialisiert wurde, Super-Resolution durchzuführen, also ein ein niedrig aufgelöstes Bild (*Low-Resolution, LR*) hochzuskalieren (*High-Resolution*, HR). Es wurde 2017 von Ledig et al. konzipiert. [4]

Zum Training wird dabei ein Datensatz an HR Bildern in Originalqualität verwendet und während der Trainingsschritte in seiner Auflösung reduziert (s. Abschnitt 2.1).

Übliche Bildkskalierungsmodelle sind darauf abgestimmt, eine Funktion des mittleren Fehlers (*Mean-Squared-Error*, MSE) pixelweise zu minimieren. Bei Anwendung im Bereich der Bildverarbeitung erwirken solche Minimierungen allerdings häufig sehr glatte Ergebnisse. Wünschenswert in der Bildskalierung sind Ergebnisse, die den wahrnehmbaren Inhalt und den Detailgrad des Originals erhalten bzw. rekonstruieren.

Ledig et al. [4] stellen ein GAN vor, welches von der gebräuchlichen MSE Fehlerfunktion abweicht und stattdessen einen *wahrnehmbaren Fehler* (Perceptual Loss) implementiert. Dieser Perceptual Loss, anstelle das SR Bild objektiv mit Metriken, wie dem MSE, zu bewerten, qualifiziert das SR Bild anhand seiner menschlich wahrgenommenen "Güte" oder Ähnlichkeit.

Dieser wird durch ein als gegeben vorrausgesetztes CNN *Very Deep Convolutional Neural Network* (VGG) realisiert. Das VGG ist ein von Simonyan und Zisserman [13] aufgestelltes CNN und wird im SRGAN in seiner 19-Schichten Variante (VGG19) verwendet. Näheres zur Verwendung als Fehlerfunktion in Unterabschnitt 2.7.2.

2.7.1 Netzwerkstruktur

Ziel ist es, einen Generator G zu trainieren, der in der Lage ist, für einen LR Input I^{LR} einen SR Output I^{SR} zu erzeugen. Statt einem Rauschen z wird hier also ein Bild in den Generator gegeben. Dieses Bild wird *as is* verwendet, also auch nicht weiter verrauscht. Der Generator ist als CNN implementiert. Die Netzwerkarchitektur ist in Abb. 2.5 und Abb. 2.6 veranschaulicht.

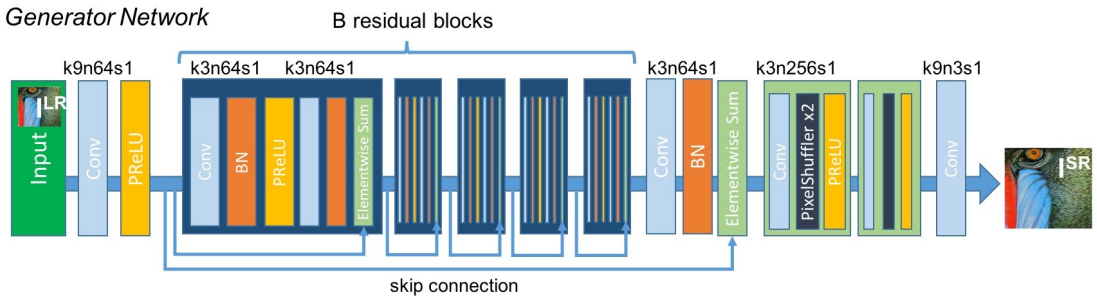


Abbildung 2.5: Generator Netzwerk

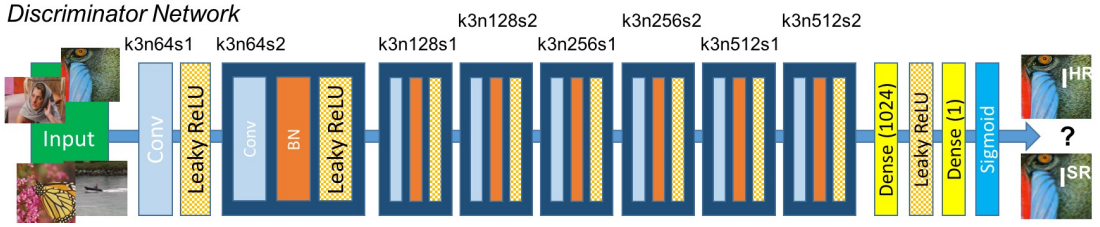


Abbildung 2.6: Diskriminator Netzwerk

Für die gegebenen HR Bilder I^{HR} werden die Parameter θ_G des Generators wie folgt bestimmt.

$$\hat{\theta_G} = arg\min_{\theta_G} \frac{1}{N} \sum_{n=1}^{N} l^{SR}(G_{\theta_G}(I^{LR}), I^{HR}) \tag{2.7.1}$$

Dabei gilt $N = |\mathcal{N}|$ und gibt die Anzahl an Bildern des Trainingssets an. Der Index n identifiziert ein Bild des Trainingssets. [1]

$G_{\theta_G}(I^{LR})$ ist das vom Generator auf Basis des gegebenen LR Bildes I^{LR} unter Verwendung der Parameter θ_G erzeugte hochaufgelöste Bild I^{SR}.

Die Wahl der Verlustfunktion l^{SR} ist hierbei entscheidend und stellt den charakteristischen Unterschied des SRGANs zu klassisschen GANs dar.

Hinzu kommt ein Diskriminator D, dessen Aufgabe ist, ein Inputbild als *echt* (I^{HR}) oder *falsch* (I^{SR}) zu klassifizieren (s. Abschnitt 2.6).

Die Verwendung der GAN Struktur (Generator vs. Diskriminator) reduziert die Mengen an potentiellen Outputbildern auf jene, die denen des Trainingsdatensatzen ähneln.

[1] Der Lesbarkeit wegen wird der Index n ab hier an nur an der Summe verwendet, nicht jedoch an den Bildvariablen I_n^X.

Das SRGAN löst also folgendes min-max Problem (analog zu Abschnitt 2.6):

$$\min_{\theta_G} \max_{\theta_D} \mathbb{E}_{I^{HR} \sim p_{\text{train}}(I^{HR})}[\log D_{\theta_D}(I^{HR})] + \mathbb{E}_{I^{LR} \sim p_G(I^{LR})}[\log(1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))] \tag{2.7.2}$$

2.7.2 Verlustfunktionen

Das SRGAN definiert eine Verlustfunktion im Bezug auf die "wahrnehmbaren" Unterschiede zwischen einem generierten SR Bild und dem zugehörigen HR Original (*Perceptual Loss function*).

Diese steht im Kontrast zum überlichweise verwendeten MSE, welcher den quadratischen euklidischen Abstand zwischen I^{SR} und I^{HR} minimieren soll, mit der Intention, so die Details und Texturen des zugrundeliegenden HR Originalbildes zu erhalten.

Dieser *Perceptual Loss* l^{SR} setzt sich aus einem *Content Loss* l^{SR}_X und einem *Adversarial Loss* l^{SR}_{Gen} wie folgt zusammen:

$$l^{SR} = l^{SR}_X + 10^{-3} l^{SR}_{Gen} \tag{2.7.3}$$

2.7.2.1 Content Loss

Für den Content Loss präsentieren Ledig et al. [4] zwei Varianten:

Der *MSE Loss* wird als quadratischer Abstand des SR und HR Bildes berechnet:

$$l^{SR}_{MSE} = \frac{1}{r^2 W H} \sum_{x=1}^{rW} \sum_{y=1}^{rH} (I^{HR}_{x,y} - G_{\theta_G}(I^{LR})_{x,y})^2 \tag{2.7.4}$$

Dabei sind W, H die Breite und Höhe der Bilder und r der Skalierungsfaktor, auf dem das SRGAN trainiert werden soll. Die Indizes x, y geben die jeweiligen Pixel der Bilder an.

Der *VGG19 Loss* nutzt ebenfalls einen MSE, berechnet diesen aber auf den vom VGG19 Netzwerk [13] berechneten Feature Maps:

$$l^{SR}_{VGG/i.j} = \frac{1}{W_{i,j} H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(I^{HR})_{x,y} - \phi_{i,j}(G_{\theta_G}(I^{LR}))_{x,y})^2 \tag{2.7.5}$$

Dabei ist $\phi_{i,j}$ die Feature Map der j-ten Faltung vor dem i-ten Maxpooling. $W_{i,j} H_{i,j}$ sind die Breite und Höhe der Feature Maps. Die Feature Maps eines

CNN enthalten die abstrahierten Informationen über Strukturen und Muster eines Bildes sowie deren Ausprägung und Lokalität (s. Abschnitt 2.5). Bilder die sich "ähneln" (in Inhalt, Stil, Komposition, ...) weisen also auch ähnliche (euklidisch nahe) Feature Maps auf.

2.7.2.2 Adversarial Loss

Der *Adversarial Loss* integriert die Leistung des Diskriminators in die Verlustfunktion:

$$l_{Gen}^{SR} = \sum_{n=1}^{N} -\log D_{\theta_D}(G_{\theta_G}(I^{LR})) \tag{2.7.6}$$

Dabei ist $D_{\theta_D}(G_{\theta_G}(I^{LR}))$ die Wahrscheinlichkeit, dass ein SR Bild vom Diskriminator als HR, also als echt, klassifiziert wird.

Ziel des Generators ist es, diese Wahrscheinichkeit zu maximieren, Gleichung (2.7.6) also zu minimieren.

Ledig et al. [4] minimieren Gleichung (2.7.6) statt $\log[1 - D_{\theta_D}(G_{\theta_G}(I^{LR}))]$ (s. Gleichung (2.7.2)) aus Gründen eines besseren Gradientenabstiegs.

2.8 Zellsegmentierung

Zellsegmentierung ist ein Verfahren, dass für ein Inputbild bestimmt, welche Pixel des Bildes Teil einer Zelle sind. Spezialisierung unterscheiden dabei weiter zwischen verschiedenen Zell- oder Gewebetypen. Das *Contour Proposal Network* (CPN) ist ein von Upschulte et al. [14] entwickeltes Modell, dass für ein Inputbild eine *Zellsegmentierung* durchführt. Es trifft dabei für jedes Inputpixel die binäre Entscheidung, ob es Teil einer Zelle ist oder nicht.

Das CPN ist wie folgt aufgebaut: Zunächst generiert ein frei wählbares CNN zwei Feature Maps P_1 (high-resolution) und P_2 (low-resolution). Auf P_2 werden dann gleichzeitig und unabhängig voneinander Pixel klassifiziert und Konturen definiert. Ein Klassifikationsnetzwerk ordnet jedes Pixel der Feature Map P_2 einer Klasse zu; im Falle des CPNs, ob das Pixel teil einer Zelle ist, oder nicht. Ein paralleles Regressionsnetzwerk (ein neuronales Netz, dass kontinuierliche Outputwerte liefert) generiert Konturen für die auf dem Bild dargestellten Objekte. Dies können Zellen, Blutgefäße, Fetteinlagerungen, etc. sein. Die Konturen sind dabei per *Fourier Deskriptoren* definiert; das Regressionsnetzwerk bestimmt die zugehörigen Fourierkoeffizienten.

Anhand der Klassifikationen und der Konturen wird eine Liste der Konturen erstellt, die Zellen markieren. Aus dieser Liste kann per Rasterisation eine Bitmaske berechnet werden, deren Einträge angeben, ob das entsprechende Pixel Teil einer Zelle ist oder nicht.

2.9 Evaluationsmethoden

Da der Output des SRGANs auch in anderen Netzwerken verwendet werden soll, ist es wichtig, die Qualität dessen auch unabhängig der menschlichen Wahrnehmung zu beurteilen. Dazu wird in dieser Arbeit für sowohl die HR Bilder I^{HR} als auch die vom SRGAN generierten SR Bilder I^{SR} eine *Zellsegmentierung* (s. Abschnitt 2.8) berechnet. Diese Zellsegmentierungsmasken der HR und SR Version eines Bildes werden verglichen, um eine Abschätzung zu geben, wie ("gut") die SR Bilder gegenüber ihren HR Originalen in anderen Modellen verarbeitet werden. Auch wird das Peak-Signal-Noise-Ratio für jedes HR-SR Bildpaar berechnet um einen mathematischen Indikator für die Qualität der SR Rekonstruktion zu geben; unabhängig der Performance in dritten Netzwerken.

Im Folgenden werden Metriken vorgestellt, die zur Evaluation der SR Bilder herangezogen wurden:

2.9.1 Metriken

Der *Jaccard Index* (auch Jaccard-Koeffizient oder *Intersection over Union* (IoU)) quantifiziert die Ähnlichkeit zweier Mengen. Er ist definiert als das Verhältnis von Schnittmenge zu Vereinigung:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2.9.1}$$

Vergleicht man Listen von binären Argumenten (True/False), so wird Gleichung (2.9.1) zu:

$$IoU(P, Q) = \frac{|P \wedge Q|}{|P \vee Q|} \tag{2.9.2}$$

Die Logischen Verknüpfungen AND und OR werden paarweise durchgeführt; der Betragsoperator $|\circ|$ gibt dann die Anzahl an True-Elementen der Liste zurück.

Für zwei Mengen/Listen mit identischen Elementen/Argumenten gilt $J(A, B) = IoU(P, Q) = 1$. Umgekehrt gilt für komplett gegenläufige Mengen/Listen $J(A, B) = IoU(P, Q) = 0$.

Der *Mean-Squared-Error* (MSE) ist eine verbreitete Methode, um die Ähnlichkeit zweier Datenpunkte P, Q mit N Features zu bewerten. Dazu wird der elementweise euklidische Abstand berechnet, quadriert, aufsummiert und über die Anzahl der Elemente gemittelt. Für den Mean-Squared-Error MSE gilt:

$$MSE = \frac{1}{N} \sum_{n=0}^{N-1} [P(n) - Q(n)]^2 \tag{2.9.3}$$

n kann ein mehr-dimensionaler Index sein. Dann läuft die Summe über alle Subindizes.

Das *Peak-Noise-Signal-Ratio* (PSNR) ist ein Maß, das häufig verwendet wird, um die Qualität von rekonstruierten Bildern zu bewerten und ist ein Indikator dafür, wie menschliche Wahrnehmung eine Rekonstruktion bewerten würde. Dabei gilt I als das rauschfreie Original und K als die Approximation. Es wird häufig in Dezibel gemessen und ist definiert wie folgt:

$$PSNR(I, K) = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE(I, K)} \right) \tag{2.9.4}$$

MAX_I ist die Spannweite des Wertebereichs, den die Pixel des Originals I annehmen können; Im Falle eines monochromen 8-bit Bildes also 255. Im Allgemeinen gilt für ein Bild mit B Bits pro Pixel: $MAX = 2^B - 1$

Für das PSNR gilt, je höher desto besser. Bei perfekter Rekonstruktion wären I und K identisch und der MSE daher 0. Je nach Definition wäre das PSNR dann ∞ oder undefiniert. Je größer das Rauschen, desto höher der MSE und desto kleiner das PSNR. Umgekehrt gilt: $\lim_{MSE \to \infty} PSNR = 0$

3 Datensätze

3.1 Präparatgewinnung

Die in dieser Arbeit verwendeten Aufnahmen wurden im Rahmen des *BigBrain* Projektes erstellt. [5] Das *BigBrain* ist ein 3D-Modell eines Menschenhirns mit einer Auflösung von $20\mu m$/vx.

Das für das *BigBrain* verwendete Spenderhirn entstammt einem 65 Jahre alten, männlichen Spender und wog bei seiner Entnahme 1392 Gramm. Die Autopsie wurde innerhalb 24 Stunden nach dem Tod des Spenders durchgeführt. Das Präparat wurde mit Formalin fixiert, in Paraffin eingebettet und durch eine Silberfärbung (Merker, 1983) kontrastiert. [1] Das Spenderhirn wurde dann in 7404 histologische Schnitte mit einer Dicke von je $20\mu m$ coronal geschnitten, also parallel zum Gesicht. Die Schnitte sind posterior nach anterior (vom Hinterkopf zum Gesicht hin) aufsteigend nummeriert. Anschließend wurden die Schnitte bei einer Auflösung von $1\mu m$/px unter Verwendung eines *High-throughput lightmicrospic Scanner* (Modell *Tissue Scope HS* des Herstellers Huron Digital Pathology Inc.) eingescannt. Die Schnitte weisen eine mittlere Größe von 77.000 x 105.000 Pixeln auf. [15]

3.2 Datenpunkte und Datensätze

Sowohl die Aufnahmen des Trainigssets als auch die des Validierungssets entstammen dem *Okzipitallappen*, dem hintersten Anteil des Großhirns. Ein Großteil dessen wird vom *visuellen Cortex* eingenommen, der die visuelle Wahrnehmung des Menschen möglich macht.

Das *Trainingsset* enthält 18.000 monochrome, randomisierte Bildausschnitte der gescannten Schicht 429 mit Abmessung 1 x 256 x 256; d. h. einem Farbkanal und je 256 Pixel Länge und Breite. Alle Bildausschnitte wurden demselben Gehirnschnitt entnommen. Das *Validierungsset* enthält 20.000 ebenfalls monochrome, randomisierte Bildausschnitte der Schicht 430 und weist die gleichen Maße wie das Trainingsset auf (1 x 256 x 256 Pixel). Die Validierungsaufnahmen wurden also einer benachbarten Schnittprobe entnommen und entstammen daher derselben

[1] Das genaue Verarbeitungsprotokoll ist in Amunts et al. (2000) gegeben.

Hirnregion wie die Bilder des Trainingssets.

Diese randomisierten Bildausschnitte des BigBrains sind derzeit nur intern verfügbar. Der von Schiffer et al. [16] zusammengestellte Datensatz *Selected 1 micron scans of BigBrain histological sections* bietet ähnliche Datenpunkte. Es ist davon auszugehen, dass dieser ähnliche Ergebnisse produzieren würde.

Abb. 3.1 zeigt 32 dem Validierungsset entnommene Bildausschnitte in Originalauflösung (*High Resolution*, 1 x 256 x 256 Pixel).

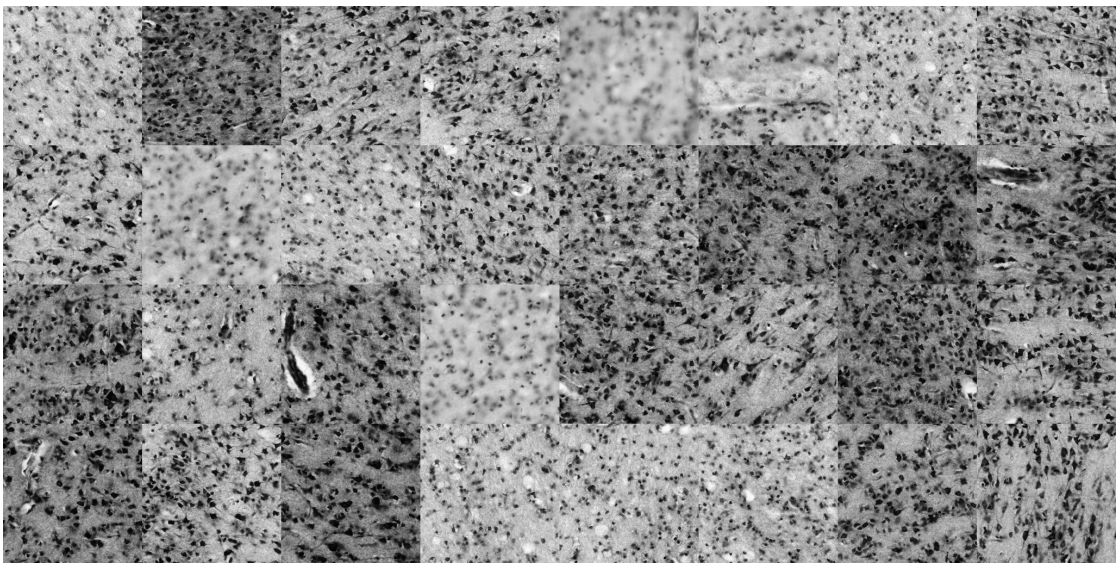


Abbildung 3.1: Rasterdarstellung von 32 Bildausschnitten des verwendeten Validierungssets

4 Experimente

Das Vorhaben dieser Arbeit ist es, ein SRGAN (s. Abschnitt 2.7) mit einem Datensatz des INM-1 (s. Kapitel 3) zu trainieren, um Ausschnitte von histologischen Gehirnaufnahmen der Größe 1 x 256 x 256 Pixel in ihrer Auflösung um das je Zwei- und Vierfache zu vergrößern. Die verwendete Implementation der SRGAN Architektur basiert auf PyTorch Lightning (s. Abschnitt 4.1) und wurde der Modellbibliothek *PyTorch Lightning Bolts* (Bolts) (s. Abschnitt 4.1) entnommen.

Die in Bolts implemenierte Variante des SRGAN [17] divergiert von der Architektur aus dem SRGAN-Paper [4] mit Blick auf die verwendeten Generatorverlustfunktionen. Dieses Kapitel wird die Unterschiede der beiden Verlustfunktionen erläutern, eine Dritte präsentieren und das SRGAN im Bezug auf die verwendete Verlustfunktion und die Anzahl der Trainingsepochen optimieren. Die Optimierung findet per Gittersuche statt.

Um die Qualität einer SRGAN Konfiguration zu bewerten, werden verschiedene, in Abschnitt 2.9 erläuterte, Metriken verwendet.

4.1 Implementierung

Die im Rahmen dieser Arbeit durchgeführten Trainings des SRGANs nutzen allesamt die Implementierung der PyTorch Lightning Bolts Bibliothek (s. Abschnitt 4.1). [18] Lediglich die Verlustfunktion wurde verändert, um das SRGAN auf mikroskopische Gehirnaufnahmen hin zu optimieren.

PyTorch Lightning (Lightning, PL) ist eine Programmbibliothek des Entwicklers PyTorch Lightning AI [19] und basiert auf dem weit verbreiteten Python Framework *PyTorch*. [20]

PyTorch bietet Algorithmen und Datenstrukturen um Modelle zu bauen und zu trainieren, Datensätze zu verwalten und trainierte Modelle in Form von *Checkpoints* zu exportieren. Ein Checkpoint ist eine Datei, die den aktuellen Zustand eines Modells speichert. Diese beinhaltet neben der Modellarchitektur und den -parametern auch Einstellungen wie z. B. den Skalierungsfaktor und die verwendete Lernrate (s. Abschnitt 2.3). PyTorch ist in der Lage, einen solchen Checkpoint zu laden und das darin definierte Modell zu rekonstruieren.

Lightning erweitert PyTorch um Datenstrukturen wie *LightningModules* (Netzwerke und Modelle), *DataLoadern* (Lesen und Laden, Mischen von Datasets), *DataModules* (Separierung von Train-, Validierungs- und Testsets), *Trainern* (Trainieren, Validieren und Testen von Modellen; auch parallel) und *Logging*. Viele dieser Datenstrukturen kapseln von PyTorch gegebene Funktionen und machen Machine Learning Workflows so modularer.

PyTorch Lightning Bolts (Bolts) ist eine Sammlung an Lightning Modulen, DataModules, Callbacks, etc., welche häufig als konkrete Implementierungen verschiedener Paper dienen; so auch das in dieser Arbeit untersuchte SRGAN. [17]

Das im SRGAN-Paper [4] konstruierte Netzwerk entspricht nicht exakt der in Bolts implementierten Variante. Die Versionen unterscheiden sich dabei in der Komposition der Generatorverlustfunktionen. So erweitert die Bolts Implementation den Generatorverlust um eine MSE-Komponente zwischen HR und SR Bild. Die Bolts Variante verwendet also den im SRGAN-Paper diskutierten Perceptual Loss und nimmt weiterhin einen MSE hinzu. Die verschiedenen Varianten werden in Abschnitt 4.2 präsentiert und verglichen.

4.1.1 Preprocessing

Das in *Bolts* implementierte SRGAN erwartet ein *DataModule* welches ein HR und das zugehörige LR Bild zurückgibt. Der in dieser Arbeit verwendete DataLoader lädt und gibt nur die HR Versionen des Traningssets zurück (also die Originale). Das SRGAN wurde dahingehend modifiziert, dass es das ihm übergebene HR Bild per bikubischer Interpolation (s. Abschnitt 2.1) in seiner Auflösung reduziert und daraufhin den Trainigsschritt startet. Dazu wurde die von PyTorch gegebene Funktion `resize` verwendet, die, neben der Zielgröße, auch eine Interpolationsmethode entgegennimmt.

Abhängig von der untersuchten Konfiguration des SRGANS wurde die Bildgröße halbiert oder geviertelt.

4.2 Training

Das SRGAN wurde mit verschiedenen Konfigurationen trainiert. Eine Konfiguration setzt sich dabei aus einem Skalierungsfaktor S, der Epochenanzahl E und der verwendeten Generatorverlustfunktion *Loss* zusammen. Alle Konfigurationen wurden für insgesamt 200 Epochen trainiert. Die Epochenanzahl gibt an, wie oft ein Datensatz für das Training verwendet wird. 200 Epochen heißt also, dass jedes Bild des Trainingssets insgesamt 200 Mal für das Training des SRGANs verwendet wurde. Dabei wurde nach je 10 Epochen ein *Checkpoint* erstellt. Für jede Kombina-

tion von Skalierungsfaktoren und Generatorverlustfunktionen sind also insgesamt 20 Checkpoints verfügbar.

Tab. 4.1 listet die trainierten Kombinationen an Skalierungsfaktoren und Generatorverlustfunktionen. Die unter *Loss* genannten Namen für die Generatorverlustfunktionen sind im Unterabschnitt 4.2.1 aufgeschlüsselt.

Tabelle 4.1: Trainingskonfigurationen

ID	*S*	*Loss*	max *E*	*Batch Size b*
B2	2	BOLTS	200	32
B4	4	BOLTS	200	32
P2	2	PAPER	200	32
P4	4	PAPER	200	32
O2	2	OURS	200	32
O4	4	OURS	200	32

4.2.1 Generator Verlustfunktionen

Das SRGAN wird auf seine Performance mit Hinblick auf die verwendete Generatorverlustfunktion getestet. Dieser Abschnitt definiert die in Tab. 4.1 genannten Kürzel der Verlustfunktionen.

PAPER meint die ursprünglich vom SRGAN-Paper vorgeschlagene Generatorverlustfunktion und definiert sich wie folgt:

$$l_{PAPER} = l^{SR} = l^{SR}_{VGG} + 10^{-3} \cdot l^{SR}_{Gen} \tag{4.2.1}$$

mit l^{SR}_{VGG} wie in Gleichung (2.7.5) und l^{SR}_{Gen} wie in Gleichung (2.7.6)

BOLTS ist die in der Bolts-Implementation um den MSE erweiterte *PAPER* Verlustfunktion:

$$l_{BOLTS} = 0.006 \cdot l^{SR}_{VGG} + 10^{-4} \cdot l^{SR}_{Gen} + l^{SR}_{MSE} \tag{4.2.2}$$

mit l^{SR}_{MSE} wie in Gleichung (2.7.4) und l^{SR}_{VGG}, l^{SR}_{Gen} wie in *PAPER*.

OURS ist eine in dieser Arbeit neu vorgeschlagene Alternative. Sie ersetzt den in l_{PAPER} verwendeten Perceptual Loss l^{SR}_{VGG} durch einen klassichen MSE Loss l^{SR}_{MSE}:

$$L_{OURS} = 0.01 \cdot l^{SR}_{Gen} + l^{SR}_{MSE} \tag{4.2.3}$$

mit l^{SR}_{Gen}, l^{SR}_{MSE} wie bereits erwähnt.

4.3 Ausgabe

Die Bilder in Abb. 4.2 und Abb. 4.6 wurden während des Trainings dokumentiert. Die Abbildungen zeigen das jeweils erste Bild des Validierungsdatensatzes (s. Kapitel 3) im HR Original (I^{HR}) (1. von links) und den zugehörigen Generatoroutput I^{SR} nach je 10, 50, 100, 150 und 200 Epochen (2. bis 6. von links) mit angegebenem Skalierungsfaktor S unter Verwendung der Generatorverlustfunktion *Loss*. Darunter je die für das darüberliegende Bild vom CPN generierten Zellmasken. Weiterhin zeigt Abb. 4.1 die Ergebnisse der bikubischen Interpolation und respektive die vom CPN generierten Zellmasken.

Sie sind nicht validiert und dienen lediglich der Orientierung und einer intuitiven Einordnung der Trainingsergebnisse.

4.4 Validierung

Alle gegebenen SRGAN Konfigurationen, bestehend aus Skalierungsfaktor S, Epochenanzahl E und Verlustfunktion *Loss* (s. Unterabschnitt 4.2.1) werden unter Verwendung eines Jaccard-Koeffizienten und des Peak-Signal-Noise-Ratios (PSNR) validiert.

Für jede Konfiguration wird das SRGAN auf alle Bilder des Validierungssets (Anzahl der Bilder $|\mathcal{N}|$ = 20.000) angewendet. Für alle generierten SR Validierungsbilder werden dann Jaccard-Koeffizienten und PSNRs berechnet. Das genaue Vorgehen und die Eingabetypen für sowohl den Jaccard-Koeffizienten als auch das Peak-Signal-Noise-Ratio sind in Abschnitt 2.9 beschrieben. Die Jaccard- und PSNR-Scores einer Konfiguration werden als gemittelte Jaccard-Koeffizienten und PSNRs über alle 20.000 Validierungsbilder hinweg berechnet.

Abb. 4.10 und Abb. 4.11 zeigen die Jaccard- und PSNR-Scores für jede Trainingskonfiguration in Abhängigkeit der Epochenanzahl E und vergleichen diese mit den Scores, die weiterhin für die bikubische Interpolation berechnet wurden.

4.4.1 PSNR-Score

Der PSNR-Score (Abb. 4.11) bietet keine verlässliche Aussage, da sein arithmetisches Mittel über alle Verlustfunktionen und Skalierungsfaktoren hinweg bei 5 ± 10^{-3} liegt. Auch die Standardabweichungen liegen allesamt in Größenordnungen von $1,511 \pm 10^{-4}$. Bemerkenswerterweise erreichen die bikubischen Interpolationen für Faktor 2 und 4 ein PSNR von $\approx 28,5$ und $\approx 23,9$ respektive.

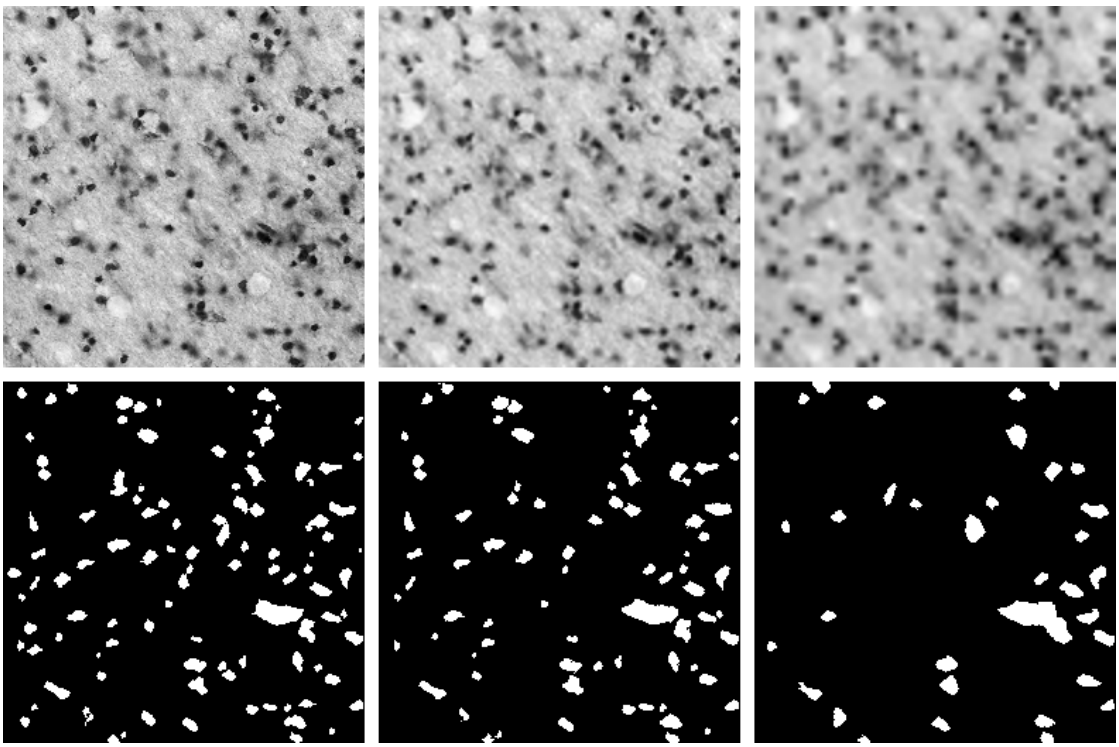


Abbildung 4.1: Oben: HR, $S = 2$, $S = 4$; bikub. interpoliert (v.l.n.r)
Unten: CPN Zellmasken der ebengleichen Skalierungen

Abbildung 4.2: Vegleich der SR Bilder bei 2-facher Skalierung unter Verwendung versch. Generatorerlustfunktionen im HR Original (1. v. l.) und nach je 10, 50, 100, 150 und 200 Trainingsepochen (2. v. l. n. r.). Darunter je die vom CPN generierten Zellmasken

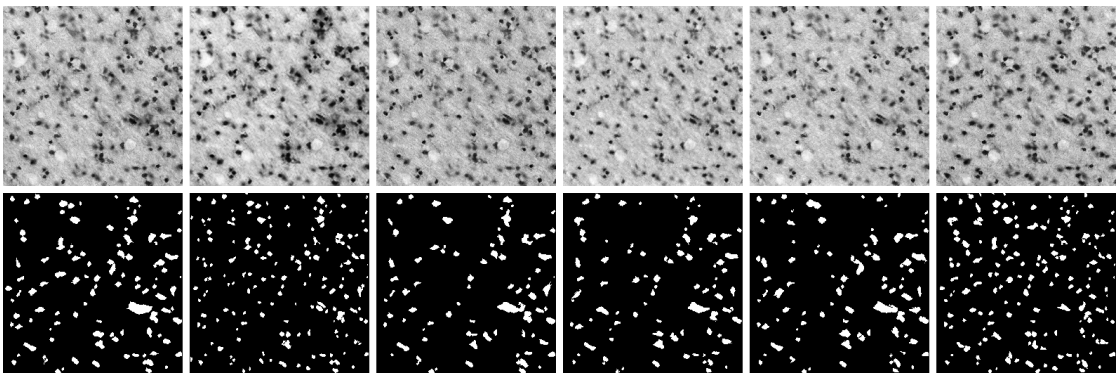


Abbildung 4.3: $S = 2$, *Loss PAPER*

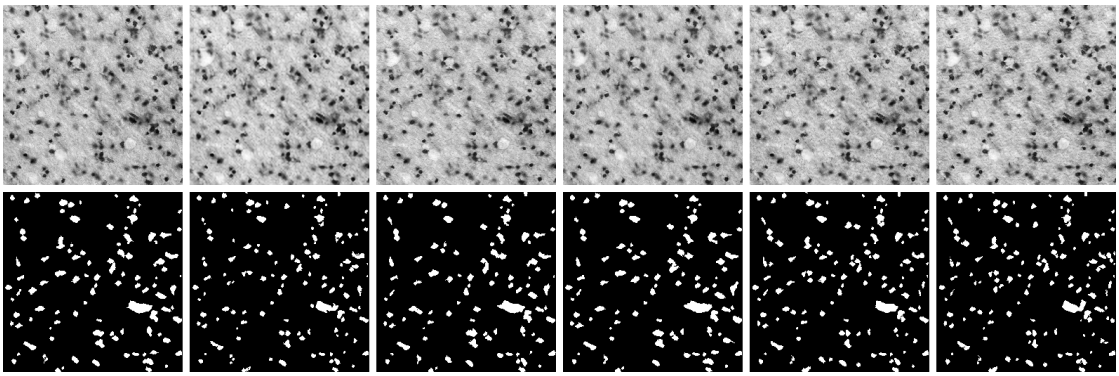


Abbildung 4.4: $S = 2$, *Loss BOLTS*

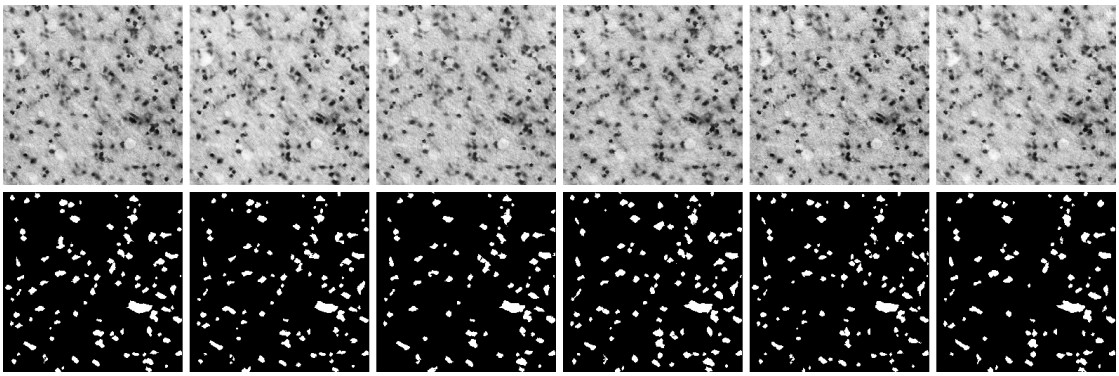


Abbildung 4.5: $S = 2$, *Loss OURS*

Abbildung 4.6: Vegleich der SRGAN Bilder bei 4-facher Skalierung unter Verwendung versch. Generatorerlustfunktionen im HR Original (1. v. l.) nach je 10, 50, 100, 150 und 200 Trainingsepochen (2. v. l. n. r.). Darunter je die vom CPN generierten Zellmasken

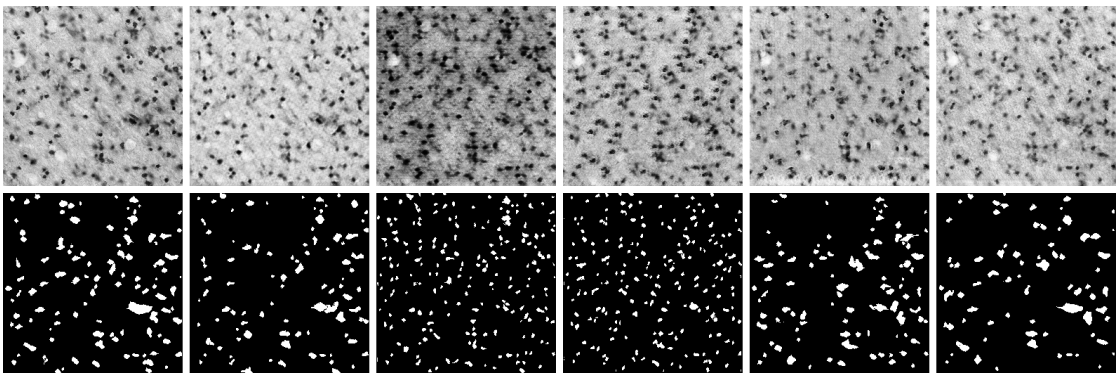


Abbildung 4.7: $S = 4$, *Loss PAPER*

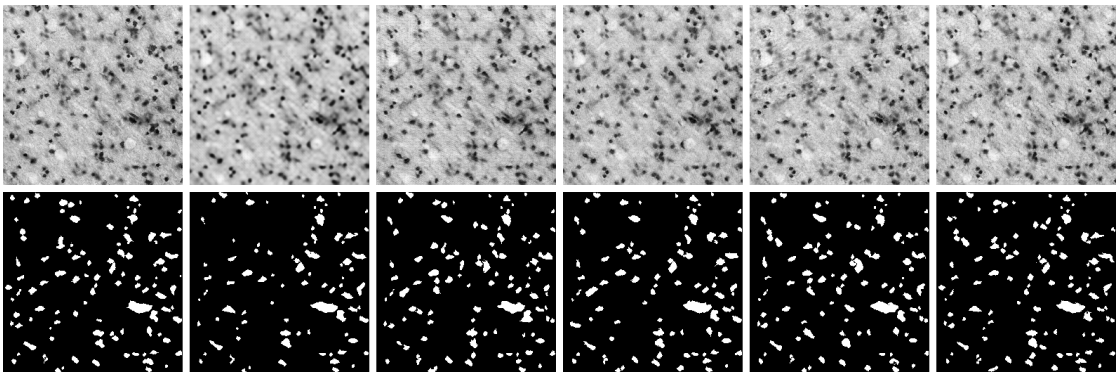


Abbildung 4.8: $S = 4$, *Loss BOLTS*

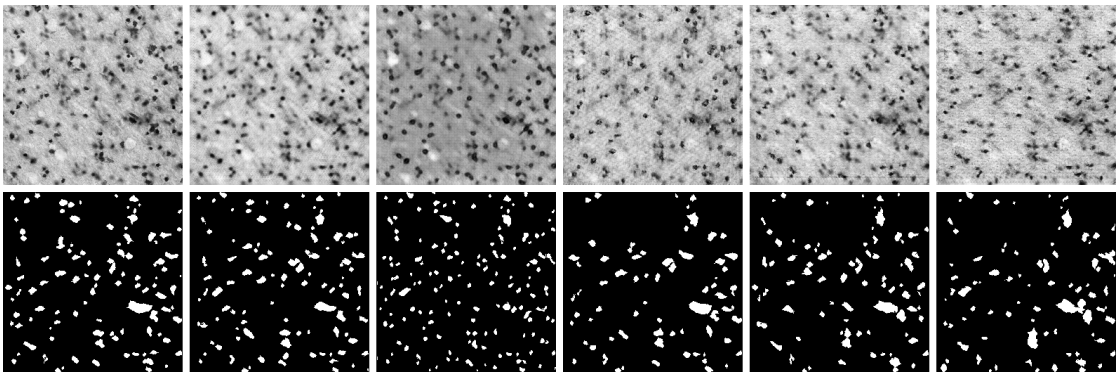


Abbildung 4.9: $S = 4$, *Loss OURS*

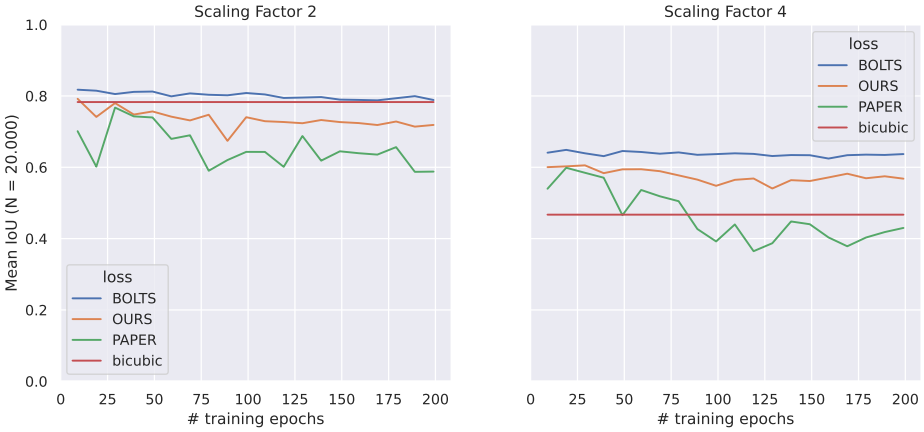


Abbildung 4.10: Jaccard-Scores (IoU)

Abbildung 4.11: Peak-Signal-Noise-Ratios Scores (PSNR)

4.4.2 Jaccard-Score

Mit Blick auf Abb. 4.10 lässt sich erkennen, dass die *BOLTS* Variante über alle Trainingsepochen hinweg die höchsten Jaccard-Scores aufweist; gefolgt von *OURS* und zuletzt *PAPER*.

Die *OURS* Variante erreicht ohne VGG zwar höhere Jaccard-Scores und ist über die Epochen hinweg konsistenter, als *PAPER*, aber konsquent schlechter als *BOLTS*. Beide Varianten *BOLTS* und *PAPER* verwenden den *Perceptual Loss*. Der Unterschied dieser beiden Varianten liegt in der Hinzunahme eines MSE bei *BOLTS*.

4.4.3 Vergleich mit bikubischer Interpolation

Die bikubische Interpolation erreicht ein deutlich höheres PSNR als das SRGAN mit den präsentierten Generatorverlustfunktionen. Dieses liefert aber mit Blick auf die PSNR-Werte für das SRGAN keine verlässliche Aussage über die Qualität der Bilder und steht auch nicht näher im Zusammenhang mit der Performance des CPN. Der Jaccard-Score der bikubischen Interpolation ordnet sich für die 2-fache Skalierung an zweiter Stelle, gerade unter der *BOLTS* Variante, ein und liefert für die 4-fache Skalierung, mit der *PAPER* Variante vergleichbare Ergebnisse.

5 Diskussion

Das Training des SRGANs mit Absicht der Super Resoltion von mikroskopischen Gehirnaufnahmen hat funktioniert und liefert visuell schlüssige Ergebnisse. Dabei liefert das SRGAN, wie zu erwarten, bei einer zweifachen Skalierung bessere Ergebnisse als bei einer vierfachen Skalierung, da von vorneherein noch mehr Information im LR Bild erhalten bleiben.

Allerdings zeigt die Datenlage (s. Abschnitt 4.4), dass sich das SRGAN vorerst auch mit zweifachem Skalierungsfaktor nicht gegen die bikubische Interpolation durchsetzten kann. Denn für die zweifache Skalierung weist die bikubische Interpolation zwar etwas schlechtere Jaccard-Scores auf als die der besten SRGAN-Konfiguration *BOLTS*, hat aber keinerlei Trainingsaufwand und gewinnt dadurch an Vorteil.

Für eine vierfache Skalierung weisen sowohl SRGAN als auch bikubische Interpolation schlechte Jaccard-Scores auf; das CPN erkennt in beiden Fällen maximal 60% der Zellen des HR Originals. Unabhängig der Weiterverarbeitung einer CPN Zellsegmentierung steht dieser Genauigkeitsverlust nicht im Verhältnis zum vorher betriebenen Trainingsaufwand.

Anhand der Abb. 4.6 ist zu erkennen, dass die Zellsegmentierung des CPNs durch die SRGAN-Prozessartefakte beeinflusst wird. Das, besonders in Epoche 50 und 100 (3. und 4. v. l.), durch das SRGAN eingefügte Rauschen schmählert die Zellsegmentierung des CPNs und lässt dieses Zellen erkennen, wo schon im HR Original keine erkannt wurden.

Die im Unterabschnitt 4.2.1 präsentierte Variante *OURS* (ohne Perceptual Loss) wurde mit der Vermutung implementiert, dass das VGG, welches auf ImageNet [1] trainiert wurde, die SRGAN Bilder verzerren könnte. Die Bilddomäne von ImageNet ist deutlich größer als die des verwendeten Trainingssets. Das SRGAN soll in dieser Arbeit aber nur für Super-Resolution von mikroskopischen Gehirnaufnahmen trainiert werden. Dies rechtfertig den Einwand, dass das VGG dadurch den Trainingsprozess stören könnte.

[1] ImageNet [21] ist ein Datensatz an natürlichen Bildern (Pflanzen, Tiere, Gegenstände, ...), der entsprechend des WordNets strukturiert ist. Das WordNet [22] dokumentiert Synonyme und semantische Verbindungen von knapp 100.000 Wörtern der englischen Sprache.

Die Jaccard-Scores der SRGAN-Konfigurationen lassen aber keine Eindeutige Interpretation dessen zu, weil die Varianten mit Perceptual Loss nicht konsequent bessere/schlechtere Jaccard-Scores aufweisen: Denn die *OURS* Variante (ohne Perceptual Loss), ordnet sich im Bezug auf die Jaccard-Scores, zwischen den anderen Varianten *BOLTS* und *PAPER* ein (beide verwenden das VGG für einen Perceptual Loss).

Es sei aber anzumerken, dass die *BOLTS* Variante (mit den höchsten Jaccard-Scores) den Perceptual Loss auch nur mit einem Faktor von 0.006 gewichtet. Die *PAPER* Variante, die den Perceptual Loss 1-fach gewichtet, weist weiterhin die größten Schwankungen der Jaccard-Scores pro Epoche auf. Auch wenn sich der Effekt, den das VGG auf die Performance des SRGANs hat, nicht eindeutig feststellen lässt, deuten diese starken Jaccard-Score Schwankungen der *PAPER* Variante auf ein durch das VGG verzerrtes SRGAN-Training hin. Andererseits weist auch die *OURS* Variante, komplett ohne Perceptual Loss, stärkere Jaccard-Score Schwankungen als die *BOLTS* Variante auf, auch wenn diese schwächer sind, als die Schwankungen der *PAPER* Variante. Es ist zu Vermuten, dass geringe Gewichtungen des VGG Perceptual-Loss das Training stabilisieren können.

Die Rangordnung der Verlustfunktionen bzgl. der Jaccard-Scores zeigt, dass eine alleinige Verwendung des Perceptual Loss (*PAPER* Variante) keine konsequent guten Ergebnisse bringt. Sie zeigt aber, dass auch die alleinige Verwendung des MSEs (*OURS* Variante) nur bedingt bessere Zellsegmentierungen ermöglicht. Den Jaccard-Scores zur Folge bewährt sich eine Kombination aus einem "bisschen" Perceptual Loss und einer vollen Gewichtung des MSE (*BOLTS* Variante).

Bei einer so hohen Auflösung von $1\mu m/\text{px}$ und zudem noch (im Vergleich zum gesamten histologischen Schnitt) kleinen Bildausschnitten von $256^2\mu m^2$ erzeugen "glatte" Bilder (per MSE) bessere Zellsegmentierungen, vermutlich weil es in dieser Größenordnung wichtiger ist, die Zelle vom Hintergrund unterscheiden zu können, als dass tatsächlich die feinen Details der Gewebeaufnahme erhalten bleiben. Ein MSE alleine genügt aber dennoch nicht (s. Jaccard-Scores von *OURS*), evtl. weil die SR Bilder dadurch zu "weich" werden und Zellkonturen mit dem Hintergrund verschmelzen könnten (ähnlich der klassischen Interpolationsmethoden).

Diese Arbeit hat gezeigt, dass das SRGAN eine gute Grundlage für Super-Resolution von mikroskopischen Gehirnaufnahmen bietet, mit der Absicht, Zellsegmentierungen für die erzeugten SR Bilder durchzuführen. Für einen zweifachen Skalierungsfaktor performt das SRGAN vergleichbar gut zur bikubische Interpolation; für eine vierfache Skalierung sogar allgemein besser. Dennoch sind die Ergebnisse bei einer vierfachen Skalierung bislang zu schlecht um das SRGAN in solch einer Konfiguration als SR Methode zu empfehlen. Es ist derzeit davon aus-

zugehen, dass die Jaccard-Scores für größere Skalierungsfaktoren noch schlechter werden.

Eine Möglickeit, um bessere Jaccard-Scores zu erreichen wäre, das CPN auf die SR Bilder der SRGANs auszurichten, anstatt das SRGAN auf das CPN abzustimmen. Die bislang für das CPN genutzten Trainingsdatensätze könnten S-fach herunterskaliert und mit den derzeit verfügbar Konturinformationen an das CPN gegeben werden. Es müsste als nur weiter trainiert, nicht jedoch neu konturiert werden. Wünschenswert wäre aber eine SR-Methode (sei das ein anderes Modell oder eine andere Konfiguration), die, unabhängig der Weiterverarbeitung dieser SR Bilder, konsequent gute Ergebnisse liefert.

Die Jaccard-Scores der SRGAN-Konfigurationen haben gezeigt, dass sich die Gewichtung des Perceptual Loss deutlich auf sowohl die visuelle Qualität der Bilder als auch die darauf ausgeführten Zellsegmentierungen auswirkt. Es gilt weitere Gewichtungen des Perceptual Loss zu implementieren und zu Untersuchen, inwiefern sich diese besser oder schlechter auf die CPN Zellsegmentierung auswirken.

Es bleibt unklar, ob andere Deep Learning Modelle ähnlich dem CPN auf die SRGAN Bilder reagieren. Diese Arbeit hat jedoch gezeigt, dass das SRGAN durchaus brauchbare Bilder für eine Zellsegmentierung generieren kann; wenn auch noch nicht gleich gut der HR Originalbilder. Es ist nicht ausgeschlossen, dass auch andere Deep Learning Modelle gut auf den Super-Resolutionen des SRGANs performen könnten.

Literatur

[1] Katrin Amunts. URL: https://julich-brain-atlas.de.

[2] Jan-Oliver Kropp u. a. „Denoising Diffusion Probabilistic Models for Image Inpainting of Cell Distributions in the Human Brain“. In: (2023). arXiv: 2311.16821 [eess.IV]. URL: https://arxiv.org/abs/2311.16821.

[3] Jonathan Ho, Ajay Jain und Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: 2006.11239 [cs.LG]. URL: https://arxiv.org/abs/2006.11239.

[4] Christian Ledig u. a. „Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network“. In: (2017). arXiv: 1609.04802 [cs.CV]. URL: https://arxiv.org/abs/1609.04802.

[5] Katrin Amunts u. a. „BigBrain: An Ultrahigh-Resolution 3D Human Brain Model“. In: *Science* 340.6139 (2013), S. 1472–1475. DOI: 10.1126/science.1235381. eprint: https://www.science.org/doi/pdf/10.1126/science.1235381. URL: https://www.science.org/doi/abs/10.1126/science.1235381.

[6] URL: https://commons.wikimedia.org/wiki/File:Interpolation-bilinear.svg.

[7] URL: https://commons.wikimedia.org/wiki/File:Interpolation-nearest.svg.

[8] URL: https://commons.wikimedia.org/wiki/File:Interpolation-bicubic.svg.

[9] Christopher M. Bishop. *Deep Learning [E-Book] : Foundations and Concepts*. Hrsg. von Hugh Bishop. 1st edition 2024. englisch. Cham: Springer, 2024, S. 649. URL: https://doi.org/10.1007/978-3-031-45468-4.

[10] Christopher M. Bishop. *Pattern recognition and machine learning*. Hrsg. von Christopher M. Bishop. Information science and statistics. englisch. New York, NY: Springer, 2006. URL: http://wwwzb.fz-juelich.de/contentenrichment/inhaltsverzeichnisse/bis2009/ISBN-0-387-31073-8.pdf.

[11] Grant Sanderson. „But what is a convolution?“ In: (2022). URL: https://www.youtube.com/watch?v=KuXjwB4LzSA&t=311s.

[12] Ian J. Goodfellow u. a. „Generative Adversarial Networks“. In: (2014). arXiv: 1406.2661 [stat.ML].

[13] Karen Simonyan und Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition“. In: (2015). arXiv: 1409.1556 [cs.CV]. URL: https://arxiv.org/abs/1409.1556.

[14] Eric Upschulte u. a. *Contour Proposal Networks for Biomedical Instance Segmentation*. 2021. arXiv: 2104.03393 [cs.CV]. URL: https://arxiv.org/abs/2104.03393.

[15] Christian Schiffer. „Deep Neural Networks for Large-Scale Cytoarchitectonic Mapping of the Human Brain“. Diss. Heinrich-Heine-Universität Düsseldorf, 2022.

[16] Christian Schiffer, Claude Lepage und Mona et al. Omidyeganeh. *Selected 1 micron scans of BigBrain histological sections*. Version 1.0. 10. Juni 2022. DOI: 10.25493/JWTF-PAB. URL: https://search.kg.ebrains.eu/instances/73c1fa55-d099-4854-8cda-c9a403c6080a.

[17] Christoph Clement. *SRGAN on PyTorch Lightning Bolts*. Version 0.7.0. 12. Juli 2023. URL: https://github.com/Lightning-Universe/lightning-bolts/tree/master/src/pl_bolts/models/gans/srgan.

[18] William Falcon. *PyTorch Lightning Bolts*. Version 0.7.0. URL: https://github.com/Lightning-Universe/lightning-bolts/tree/master?tab=readme-ov-file.

[19] PyTorch Lightning AI. *PyTorch Lightning*. Version 2.1.3. URL: https://lightning.ai/pytorch-lightning.

[20] PyTorch. *PyTorch*. Version 2.4.1. URL: https://pytorch.org.

[21] J. Deng u. a. „ImageNet: A Large-Scale Hierarchical Image Database“. In: *CVPR09*. 2009.

[22] Princton University. *About WordNet*. 2010. URL: https://wordnet.princeton.edu.