Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Image Restoration von MRT-Bildern mit Convolutional Neural Networks

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang Angewandte Informatik

1. Prüfer:	Prof. Dr. Christian Herta
2. Prüfer:	Jonas Annuscheit

Eingereicht von:	Georg Graf von Westerholt
Immatrikulationsnummer:	s0556752
Eingereicht am:	05.03.2020

Kurzbeschreibung

Bewegungsartefakte in MRT-Bildern stellen seit Jahrzehnten ein schwer zu lösendes Problem für die Medizin da. Sie entstehen durch metallische Fremdkörper, Blutungen oder Bewegung des Patienten. Durch neue Errungenschaften im Bereich des maschinellen Lernens bieten sich auch neue Ansätze zur Lösung dieses Problems. Insbesondere neue Convolutional Neural Network Architekturen liefern dabei immer bessere Ergebnisse. Diese Arbeit hat es sich zum Ziel gemacht Artefakte von MRT-Bildern mittels Convolutional Neural Networks zu entfernen. Für das trainieren der Netzwerke soll dabei ein Generator für künstliche Artefakte erstellt werden.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Image Restoration ist das Problem der Wiederherstellung von Bildern, die durch Unschärfe und Rauschen beeinträchtigt werden. Da bildgebende Geräte niemals perfekt sind, gibt es viele Anwendungen für Bildrekonstruktion zu denen auch die medizinische Bildwiederherstellung gehört. Im Rahmen von Magnetresonanztomographie (MRT) gibt es Artefakte die störend für eine genaue Begutachtung sein können. Ein MRT ist eine medizinische Bildgebungstechnik, die in der Radiologie verwendet wird, um Bilder der Anatomie und der physioligischen Prozesse des Körpers zu erstellen.[Sch05] „Artefakte sind im Bild dargestellte oder fehlende Signale, die nicht mit der räumlichen Verteilung der Gewebe in der gemessenen Schicht korrespondieren“[EF07, S. 29]. Nach Engelbrecht et al. können bei MRT-Aufnahmen Artefakte unter anderem durch metallische Fremdkörper, Blutungen oder Bewegung des Patienten entstehen. Besonders Bewegungsartefakte stellen ein Problem für medizinische Anwendungen dar: „Over 30 years of research have produced numerous methods to mitigate or correct for motion artefacts, but no single method can be applied in all imaging situations. “[ZMH15, S. 1]
Ein konkreter Anwendungsfall von Image Restoration ist die Entfernung solcher Artefakte. Für die Bildrekonstruktion sind Convolutional Neural Networks (CNNs) ein erfolgreiches Werkzeug im Bereich des Deep Learnings. So wird in dem Paper “Image Restoration using Convolutional Auto-encoders with Symmetric Skip Connections“ beispielsweise eine CNN-Architektur beschrieben, die in dem Feld der Bildrekonstruktion hervorragende Ergebnisse erzielt.[MSY16]
Die Motivation dieser Arbeit ist es durch das Einsetzen von CNNs eine Bildwiederherstellung von MRT-Bildern zu ermöglichen. Dafür sollen zwei CNN Architekturen, der

Autoencoder und das RED-Net, auf ihre Performance in der Bildwiederherstellung verglichen werden.

1.2 Zielsetzung und Vorgehensweise

In dieser Abschlussarbeit soll analysiert werden, welche Bildartefakte in MRT-Aufnahmen mit Convolutional Neural Networks verringert werden können und welche Architektur sich dazu eignet. Es soll zunächst eine Software zum generieren von künstlichen MRT-Artefakten geschrieben werden. Dabei soll es sich anfänglich um einfache Artefakte wie ein simples Rauschen handeln. Die Artefakte werden als Trainingsdatensatz genutzt. Als Schwerpunkt soll ein Vergleich zweier Implementierungen von CNNs auf ihre Genauigkeit in der Bildrekonstruktion entstehen. Bei den CNNs handelt es sich um einen Autoencoder und dem RED-Net welche auf die Beseitigung der Artefakte trainiert werden. Dabei kann ein Qualitätskriterium wie der "Mean Squared Error" (MSE), zwischen Original MRT-Bild und dem MRT-Bild mit generierten Artefakten und anschließendem Denoising, indizieren wie gut die Implementierungen sich auf MRT-Artefakte anwenden lassen.[SMS17] Des weiteren soll die Generator-Software um das generieren von komplexen Artefakten erweitert werden. Die CNNs werden erneut mit diesen trainiert und anschließend verglichen. Dabei sollen zum testen der trainierten Modelle real existierende MRT-Artefakte bereinigt und die Ergebnisse analysiert werden.

1.3 Aufbau der Arbeit

Um ein Verständnis für die Problemstellung und der in dieser Arbeit entwickelten Lösungen zu erlangen, werden zunächst die wissenschaftlichen Grundlagen erläutert. Dafür wird zunächst das essentielle zu maschinellen Lernen und darauf aufbauend zu Neuronalen Netzen erklärt. Anschließend werden CNNs und die Bildrekonstruktion mit CNNs beschrieben. Nachdem die Grundlagen geschaffen worden sind wird im Analyse Teil die Problemstellung tiefgehend erläutert. In dem darauf folgenden Kapitel wird die Konzeption des Projektes dargestellt. Das Kapitel Umsetzung beschreibt die Implementierung des Projektes. Dabei wird sowohl die Entwicklung der Convolutional Neural Network Modelle als auch die Implementierung des künstlichen Artefakt Generator beschrieben. Im Kapitel Evaluation werden die Modelle evaluiert und für ein Entfernen von Realen Bewegungsartefakten ausgewählt.

Kapitel 2

Grundlagen

Um die in dieser Arbeit verwendeten Methoden zu verstehen, ist es notwendig mit den theoretischen Grundlagen vertraut zu sein. Dafür werden zunächst Kernpunkte des maschinellen Lernens und neuronaler deutlich gemacht deutlich gemacht. Darauf aufbauend wird die Bildrekonstruktion mit Convolutional Neural Networks erläutert. Dabei werden die beiden in dieser Arbeit verwendeten Modelle vorgestellt.

2.1 Maschinelles Lernen

Das maschinelle Lernen befasst sich weitgehend mit der Aufgabe Modelle und Algorithmen zu erstellen, die aus Daten "lernen". Das maschinelle Lernen hat sich dabei als nützliches Werkzeug erwiesen, um Probleme zu modellieren die auf andere Weise nur schwer zu formulieren wären. Im Gegensatz zu klassischen Computerprogrammen die per Hand programmiert werden um eine Aufgabe zu erfüllen, wird beim maschinellen Lernen ein Teil des menschlichen Beitrags von einem lernenden Algorithmus ersetzt. Durch die steigende Verfügbarkeit von Rechenkapazitäten und Daten findet maschinelles Lernen einem steigenden praktischen Nutzen, sodass man von einen regelrechten Boom sprechen kann. [GBC16, S. 96–97]

2.1.1 Struktur

Algorithmen für maschinelles Lernen sind zwar vielfältig und verwenden unterschiedliche Techniken, ihre Struktur kann allerdings verallgemeinert werden. Die Struktur des Großteils der Lernalgorithmen kann dabei in folgende Komponenten unterteilt werden[Ng18]:

- Modell
- Evaluation
- Optimierung

Die einzelnen Komponenten können sich je nach Lernalgorithmus sehr unterschiedlich gestalten. Die Erklärung der Komponenten ist allerdings besonders wichtig, um eine Intuition für Neuronale Netze aufzubauen, welche im späteren Verlauf erläutert werden.

Daten Vorverarbeitung

Machinelles Lernen erfordert Datensätze mit bestimmten Eigenschaften. Jeder Datensatz enthält eine Menge von n Instanzen, welche aus einen Paar von Input x_i und Label y_i bestehen.
Input:

$$X = [x1, x2, x3, ..., x_n]$$

Dabei ist i der Index der Instanz und n die Dimension des Inputs.
Einzelne Instanzen des Inputs sowie der Labels müssen jeweils vom einheitlichen Typ sein. Im Falle von Bildern als Input Daten sind es Werte für einzelne Pixel (z.B: 0-255). In anderen Fällen können es Reelle Zahlen sein. Generell gilt, dass der Input normalisiert sein sollte. Normalisiert bedeutet, dass alle Werte in einem festgelegten Wertebereich liegen. Diese Regel erfüllt sich bei Bildern in der Regel automatisch, da jeder Pixel bereits seinen festgelegten Wertebereich hat.[Ng18]

Modell

Ein Modell nimmt den Input x_i und verwendet ihn um den Output $\hat{y}_i$ vorherzusagen. Diese Vorhersagewerte $\hat{y}_i$ sollen möglichst nah bei dem Label y_i liegen. Jedes Modell verfügt über Parameter, die durch den Vektor θ dargestellt werden. Durch den Trainingsprozess wird θ angepasst.[Ng18]

Evaluationskriterium

Das Evaluationskriterium ist das Messsystem mit welchem die Ergebnisse bewertet werden. Ein häufig genutztes Evaluationskriterium ist die Fehlerrate die mit der Kostenfunktion berechnet wird.

Die in dieser Arbeit verwendete Kostenfunktion ist die mittlere quadratische Kostenfunktion (Mean Squared Error, MSE):

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}(h_\theta(x^{(i)}) - y^{(i)})^2$$

Die Funktion beschreibt die Abweichung zwischen den vorhersage Werten eines Modells $h_\theta(x^i) = \hat{y}_i$ und dem angestrebten Wert y_i. Die Abweichung wird Kosten genannt.[Ng18]

Optimierung

In der Optimierung sollen die Parameter θ angepasst werden, so dass die Kostenfunktion minimiert wird. Das Gradientenabstiegsverfahren ist ein sehr beliebter Optimierungsalgorithmus der genau dies tut. Der Algorithmus führt wiederholt ein Update der θ aus:

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

Der rechte Teil der Formel ist die partielle Ableitung der Kostenfunktion nach θ:

$$\begin{aligned}\frac{\partial}{\partial\theta_j}J(\theta) &= \frac{\partial}{\partial\theta_j}\frac{1}{2}(h_\theta(x) - y)^2 \\ &= 2\cdot\frac{1}{2}(h_\theta(x) - y)\cdot\frac{\partial}{\partial\theta_j}(h_\theta(x) - y) \\ &= (h_\theta(x) - y)\cdot\frac{\partial}{\partial\theta_j}(\sum_{i=0}^{n}\theta_i x_i - y) \\ &= (h_\theta(x) - y)x_j\end{aligned}$$

Eingesetzt in die Gradienten Formel ergibt das:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

Die Kosten sind durch $h_\theta(x^{(i)})$ abgebildet. Sie haben also einen direkten Einfluss auf die Korrektur der θ. So wird beispielsweise bei hohen Kosten die Korrektur von θ proportional

hoch sein. Allerdings wird die Korrektur um den Parameter α gedämpft um eine schrittweise Annäherung zu ermöglichen und so zu verhindern, dass über das Ziel "geschossen"wird. Das Verfahren wird so oft wiederholt bis die Kostenfunktion $J()\theta$ ihr globales Minimum erreicht.[Ng18]

Generalisierung

Trainingsdaten können nicht alle möglichen Eingabewerte eines Modells abbilden. Daher ist es notwendig, dass der Lernalgorithmus verallgemeinern kann. Die Fähigkeit zu Verallgemeinern wird Generalisierung genannt. Ist ein Model zu simpel kann es dazu führen, dass wichtige Aspekte nicht erfasst werden können.[Bis06, S. 2]
Komplexe Modelle wie Neuronale Netze eignen sich daher besser um komplexe Verhältnisse zu erfassen. Sie können sich allerdings zu stark an die Trainingsdaten anpassen, was Overfitting genannt wird. Overfitting tritt in der Regel bei einem komplexen Problem und gleichzeitiger Verwendung von zu wenig oder ungenügenden Trainingsdaten auf. Das Modell lernt die bekannten Beispiele abzubilden, versteht Zusammenhänge aber nicht.[Mit97, S. 66–68] Es ist also mit einen "reinen Auswendig lernen"vergleichbar.

2.2 Neuronale Netze

Neuronale Netze sind ein beliebtes Modell des maschinellen Lernens. Auf einem besonderen Modell Namens Convolutional Neural Network (CNN) liegt dabei das Hauptaugenmerk dieser These. Bevor näher auf das CNN eingegangen wird ist es allerdings Notwendig ein Verständnis für das reguläre Neuronale Netz aufzubauen.

2.2.1 Ursprung

Neuronale Netze (Neural Networks, NNs) wurden ursprünglich künstliche neuronale Netze genannt, weil sie entwickelt wurden, um die neuronale Funktion des menschlichen Gehirns nachzuahmen.[Mit97, S. 82] Als Pionierarbeit dieser Forschung gelten das Neuron von Warren McCulloch und Walter Pitts aus dem Jahr 1943 und das Perceptron von Frank Rosenblatt aus dem Jahr 1957[Roj, S. 55]. Obwohl künstliche neuronale Netze also kein

neues Konzept sind, erleben Sie erst seit den letzten Jahren einen Boom. Dafür werden zwei Hauptgründe genannt:

- Rechenleistung moderner Computer
- Verfügbarkeit von großen Mengen an Daten (Big Data)

Begünstigt durch die Rechenleistung moderner Systeme werden NNs immer vielschichtiger und damit tiefer. Das Lernen mit neuronalen Netzen ist daher auch unter dem Begriff Deep Learning geläufig.[GBC16, S. 1–2]

2.2.2 Struktur

Der Begriff neuronales Netz ist sehr allgemein und beschreibt eine breite Familie von Modellen. Ein NN ist ein verteiltes, paralleles Modell, das komplexe Nichtlinearität approximieren kann. Das Netzwerk setzt sich aus mehreren Recheneinheiten genannt Neuronen zusammen, die in einer bestimmten Topologie zusammen gesetzt sind. Wie andere maschinelle Lernalgorithmen setzt sich auch das NN aus Modell, Evaluations und Optimierungsverfahren zusammen. Evaluation und Optimierungsverfahren unterscheiden sich dabei nicht von anderen maschinellen Lernalgorithmen. Der große Unterschied liegt darin, dass das Modell eines NNs wesentlich komplexer ist. Das Modell eines NNs ergibt sich aus seinen Teilen - den Neuronen, und seiner Topologie - der Art und Weise wie die Neuronen miteinander verbunden sind.[BH18]

Neuron

Wie bereits erwähnt wurde das Modell des Neurons von der Biologie inspiriert. Mit fortschreitender Forschung war die Nachahmung allerdings nicht mehr so wichtig und moderne NN-Modelle entsprechen ihren biologischen Gegenstücken nur oberflächlich.

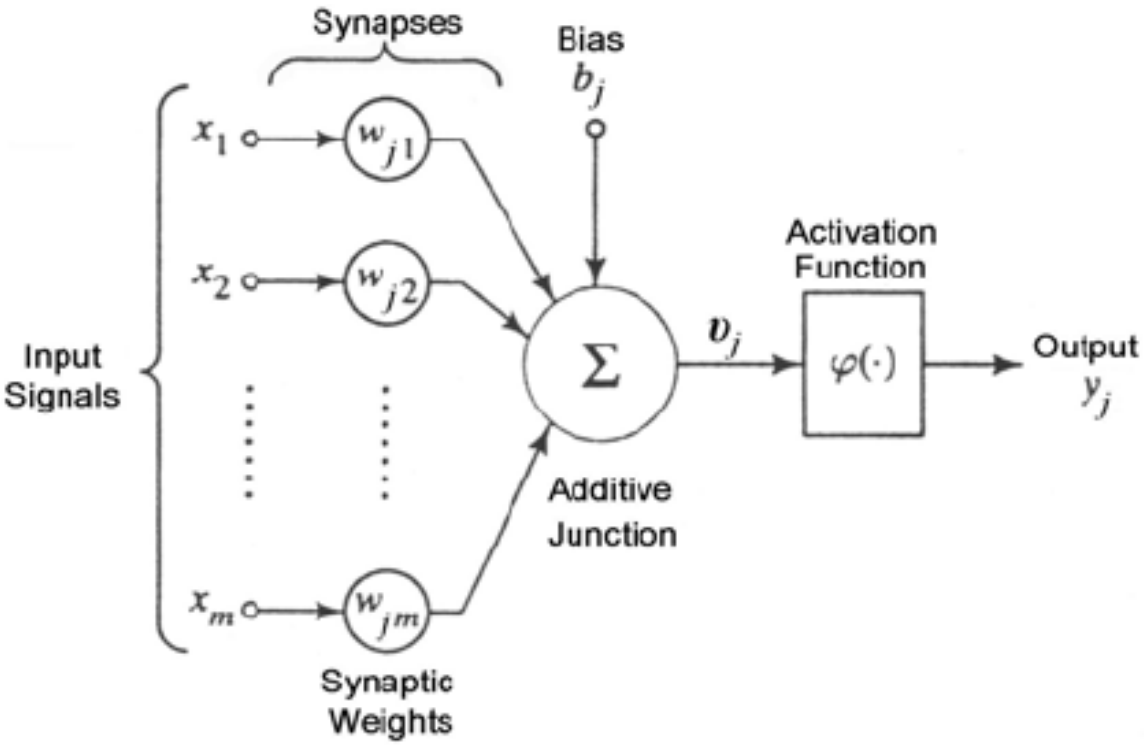


Abbildung 2.1: künstliches Neuron[Per06]

Das Neuron ist eine Recheneinheit die eine nichtlineare Transformation ihres Inputs durchführt. Ein Neuron besteht aus:

- Inputs
- Weights
- Bias
- Aktivierungsfunktion

Jedes Neuron hat mehrere Eingangswerte (Inputs) x und jedes x hat ein zugewiesenes Gewicht (Weight) w. Die Inputs des Neurons werden mit den Weight Parametern w gewichtet. Die Weights werden während des Lernens optimiert. Sie entsprechen damit dem Parameter θ der in der Beschreibung des machinellen Lernmodells vorgestellt wurde. Der Bias d ist ebenfalls ein optimierbarer Parameter. Jedes Neuron hat nur einen Bias, welcher das Neuron als Ganzes beeinflusst.[Alp10, S. 237–238]

$$z = \sum_{i=1}^{n} x_{ji} w_i + b_j$$

Zunächst wird jeder Input x mit seinem Weight multipliziert und der Bias dazu addiert. Die Ergebnisse werden summiert und in eine Aktivierungsfunktion (Activation Function) $f(z)$ eingesetzt.[Bis06, S. 227–228]

Aktivierungsfunktion

Damit sich ein NN einer nichtlinearen Funktion annähern kann, muss der summierte Input eines Neurons z in eine Nichtlinearität gebracht werden. Zu diesem Zweck wird z in eine nichtlineare Aktivierungsfunktion $f(z)$ eingesetzt.[Alp10, S. 246] Ihre Verwendung hängt ab von der Art des Netzwerks und Schicht in der sie sich befindet. Was eine Schicht ist wird im späteren Verlauf erläutert.
Eine der ältesten und historisch am häufigsten verwendeten Aktivierungsfunktionen ist die Sigmoid-Funktion:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Der Output der Sigmoid-Funktion liegt zwischen $0 - 1$, was sich besonders für einfache Klassifizierungen eignet.[Alp10, S. 238] Problem der Sigmoid-Funktion ist, dass ihre Gradienten sehr klein werden (Vanishing Gradient Problem) bei sehr hohen und kleinen Werten. Dadurch kann sich der Lern-Prozess verlangsamen oder komplett zum stoppen kommen.[See19]
Eine andere häufig genutzte Funktion ist die Hpyerbelfunktion Tangens Hyperbolicus (tanh).

$$t(z) = tanh(-z)$$

Die tanh - Funktion hat einen Wertebereich von -1 bis 1. Der Vorteil gegenüber der Sigmoid Funktion ist, dass der Output der tanh null zentriert ist was das konvergieren beschleunigt. Die tanh - Funktion hat ebenfalls das vanishing Gradient Problem.[Hay09, S. 136–137]

Die derzeit am häufigsten verwendete Aktivierungsfunktion ist die Rectified Linear Unit (RELU):

$$r(z) = max[0, z]$$

Der Nachteil der Funktion ist, dass sie für $z <= 0$ nicht differenzierbar ist. Dies ist aber vernachlässigbar, da sie kein vanishing Gradient Problem hat und aufgrund ihrer Einfachheit ein schnelles Lernen ermöglicht.[Aga18]

Neben der RELU findet auch die Softmax-Funktion eine hohe Verwendung.

$$\sigma(z_j) = \frac{e^{z_j}}{\sum_{k=i}^{n} e^{z_k}}$$

Sie wird vor allem in der Outputschicht eines Netzwerks eingesetzt. Die Funktion beschreibt dabei bei Mehrklassenproblemen die Wahrscheinlichkeit des Inputs zu einer Klasse j zu gehören.[Bis06, S. 228]

Topologie

Es gibt verschieden häufig verwendete Topologien von neuronalen Netzen. Dabei ist das Feed-Forward Netzwerk die Grundlage für die meisten anderen Topologien. Es Zeichnet sich dadurch aus, dass die Informationen nur Vorwärts, also von den Inputs zu den Outputs, fließen. Es gibt hierbei also keine Zyklen oder Schleifen in dem Netzwerk wie in etwa bei dem Recurrent Neural Network (RNN) [LB96, S. 287–288].
Ein Kriterium der Topologie ist, wie einzelne Neuronen im Netzwerk verbunden sind. Am häufigsten sind neuronale Netze in Schichten (Layer) angeordnet. In jeder Schicht kann es von $1 - n$ Neuronen geben. Die erste Schicht wird Input-Layer genannt, die letzte Output-Layer. Die Schichten dazwischen werden Hidden-Layer genannt.
Die Beschreibung des Netzwerks beruht auf den Verbindungen zwischen einzelnen Schichten. Das gängigste Schema heißt "fully Connected". Hierbei hat jedes Neuron in den verborgenen Schichten Input-Verbindungen mit allen Neuronen der vorherigen Schicht und der Output ist mit allen Neuronen der folgenden Schicht verbunden. Wird in dieser Arbeit von einen NN gesprochen ist ein Fully Connected Feed Forward Neural Network gemeint.[He16, S. 164]

2.2.3 Optimierung

Alle Optimierungsverfahren eines neuronalen Netzes basieren auf dem Gradientenabstiegsverfahren. Es ist also ein iterativer Prozess der darauf abzielt den Trainingsfehler zu verringern, indem er die Parameter des Netzes anpasst. Dafür muss man durch alle Teile des Netzwerkes gehen und ihren Einfluss auf die Kosten bestimmen. Die Technik, die zur Lösung des Problems verwendet wird, wird als Backpropagation bezeichnet.[Bis06, S. 240–241].

Backpropagation

Der Schlüssel zu Backpropagation ist die Kettenregel aus der Differentialrechnung. Zunächst werden die partiellen Ableitungen der Kostenfunktion im Hinblick auf die Weights w und des *bias* der letzten Schicht gebildet. Für die Vorletzte Schicht werden die Ableitungen der einzelnen Parameter im Hinblick auf die Ableitungen der letzten Schicht gebildet. Dieser Vorgang wird wiederholt, bis man die erste Schicht erreicht und somit alle Gradienten berechnet hat. Der Name Backpropagation rührt durch die Übergabe der Ableitungen durch das Netzwerk.[Bis06, S. 241–244]

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) sind spezielle Typen von neuronalen Netzen die ursprünglich in bildverarbeitenden Anwendungen verwendet wurden. Seit ihren Erfolg in der Bildverarbeitung, finden sie auch Anwendung in der Verarbeitung von Videos und der natürlichen Sprache.

2.3.1 Convolutional Layer

Convolution heißt ins Deutsche übersetzt Faltung. Ein Convolutional Layer zeichnet sich also, wie der Name schon andeutet, durch eine Faltungsoperation aus. Die Faltungsoperation wird auf den Input mittels eines Filters namens Kernel ausgeführt. Die Ausgabe der Faltungsoperation wird Feature Map genannt. Sie heißt so, da die Operation wichtige Merkmale (Features) des Inputs in die Feature Map extrahiert. Mit der Convolution gelingt mit anderen Worten eine Berücksichtigung der Position eines Pixel und seiner Nachbarpixel. Der Input eines convolutional Layers ist in der Regel ein als Matrix konvertiertes Bild oder die Feature Maps aus einem vorherigen Layer. Der Kernel hat typischerweise eine quadratische Form und seine Breite kann von 3 bis n Pixel reichen.[Bis06, S. 267–268]

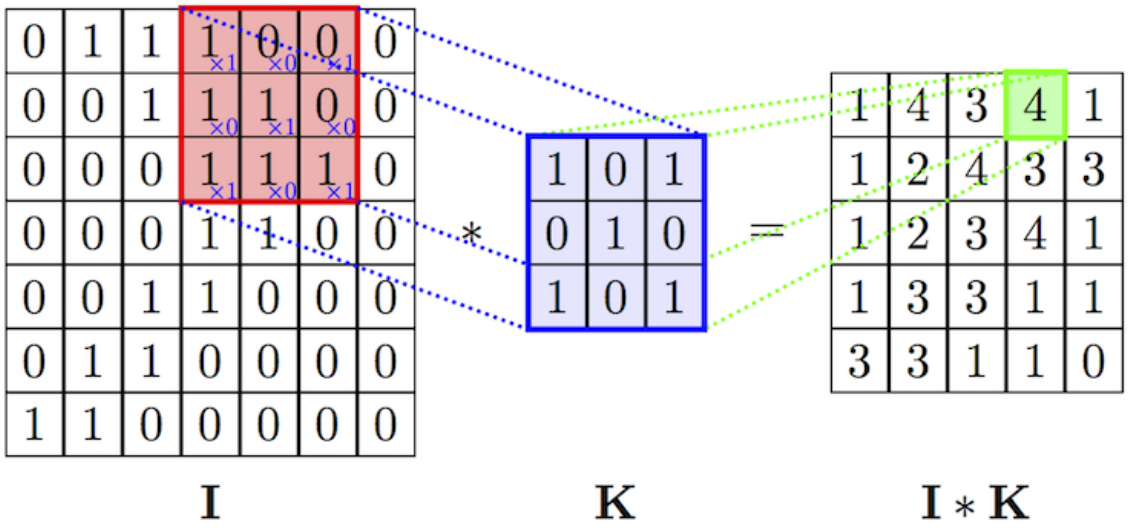


Abbildung 2.2: Feature Extraction, hier ohne Bias.[Lau19]

Der Kernel K wandert mit einer festgelegten Schrittgröße (Stride) den Input I entlang und führt für den erfassten Bereich eine Convolution, also eine mathematische Operation aus. Die vom Kernel erfassten Werte werden bei jedem Schritt jeweils mit den sich auf gleicher Höhe befindenden Werten (Weights) des Kernels multipliziert und anschließend aufsummiert. Die Summe wird mit dem Bias des Kernels addiert und anschließend als Feature in der Feature-Map abgebildet. Die Weights und der Bias eines Kernels sind trainierbare Parameter. In der Regel werden mehrere Kernel verwendet und damit auch mehrere Feature-Maps erstellt, die zusammen eine Schicht ergeben. Die Features stellen die verschiedenen Merkmale des Eingabebildes dar. In den ersten Schichten sind diese Merkmale in der Regel Kanten. Je höher die Schicht, desto komplexere Features werden erfasst.[Bis06, S. 269]

extraction.png

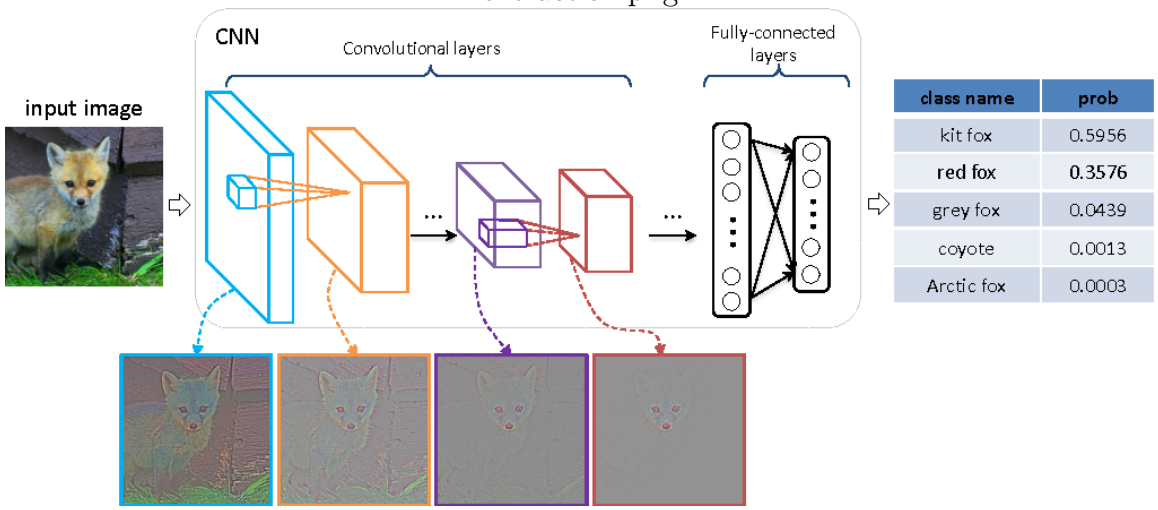


Abbildung 2.3: Convolutional Neural Network[Yu14]

Die Abbildung 2.3 zeigt eine Architektur in der zunächst die Features des Bildes durch die Convolutional Layer extrahiert werden. Anschließend kann ein Fully Connected Layer anhand derer eine Klassifizierung vornehmen.

Abhängig von der Größe des Kernels und seiner Schrittgröße kann die Convolution eine Feature-Map mit einer anderen Größe als dem Input erzeugen. Wenn die Größe erhalten bleiben soll, müssen die Ränder mit Nullen aufgefüllt werden (Zero Padding). Um die Breite der Feature-Map zu berechnen gibt es folgende Formel:

$$output_w = \frac{W - F_w + 2P}{S} + 1$$

Wobei W die Breite des Inputs ist, F_w die Breite des Kernels, P die Größe des Zero Paddings und S die Größe des Strides. Entsprechend lässt sich auch die Höhe des Outputs berechnen:

$$output_h = \frac{H - F_h + 2P}{S} + 1$$

Jedes Convolutional Layer hat eine Nichtlinearität als Output. Dies entspricht der Aktivierungsfunktion in einem Feed Forward Neural Network. Die Aktivierungsfunktion ist üblicherweise die ReLu.[Kar19]

Pooling Layer

Diese Ebene stellt in der Regel keinen Lernprozess dar, sondern wird zum Verringern der Größe des Inputs verwendet, um die Anzahl an Parametern und Berechnungen in Netzwerk zu verringern. Der Input des Pooling Layers ist normalerweise der Output der Aktivierungsfunktion. Der Input wird dabei von einem Kernel mit einem Stride s, der der Breite des Kernels entspricht, in mehrere rechteckige Felder unterteilt. Mit den Einheiten in jedem Feld wird eine Einheit des Outputs generiert. Dadurch wird die Größe des Input Layers verringert, während die wichtigsten Informationen erhalten bleiben. Mit anderen Worten werden die Informationen der Input-Schicht durch ein Pooling Layer komprimiert. Die Art der Operation, die für jedes Element ausgeführt wird, wird von der Art des Pooling Layers bestimmt. Max-Pooling ist eine typische Pooling-Operation. Hierbei wird aus jedem Feld der maximale Wert in den Output übernommen.[Kar19]

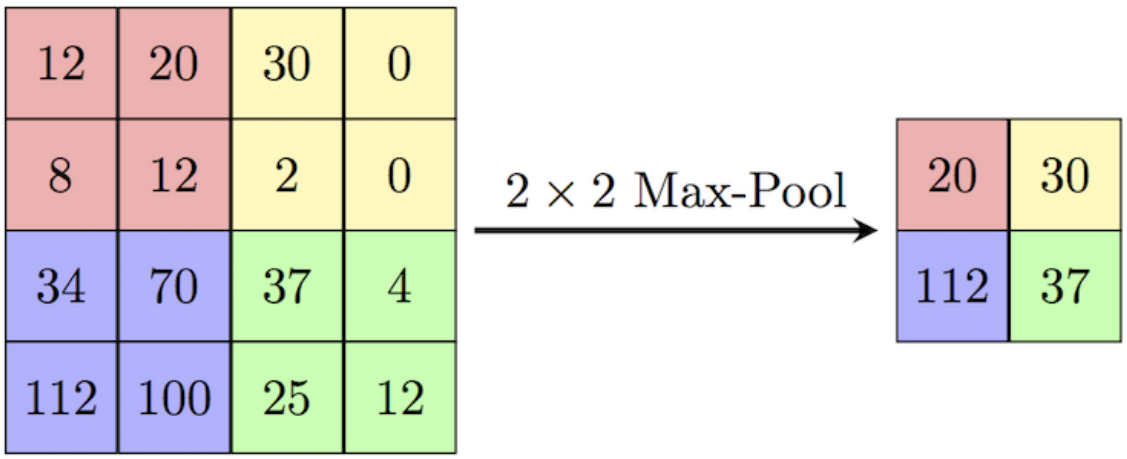


Abbildung 2.4: Max Pooling[Wik20]

2.3.2 Optimierung

Der Optimierungsprozess eines CNNs ist ähnelt dem eines FCNN. Er basiert also ebenfalls auf dem Gradientenabstiegsverfahren. Der Prozess ist allerdings komplizierter, da das CNN aus verschiedenen Arten von Schichten besteht. Das bedeutet, dass bei der Backpropagation die jeweiligen Regeln der einzelnen Schichten beachtet werden müssen.[Kar19]

2.4 Bildrekonstruktion mit CNNs

Bildrekonstruktion (Image Denoising) ist ein Feld des maschinellen Sehens (Computer Vision) und wird seit längerem erforscht. Bei der Bildrekonstruktion geht es darum Rauschen (Noise) von einem Bild zu entfernen. Es gibt dabei verschiedene Arten von Rauschen die von einem einfachen gaußschen Rauschen bis zu einem Bewegungsartefakt eines MRT-Bildes reichen können. Bei der Bildrekonstruktion verlieren Bilder üblicher Weise an Detail. Das Entfernen von Rauschen bei minimalem Detail-Verlust ist das bedeutendste Qualitätskriterium einer Denoising Methode.[Fan19]
Im Bereich der Computer Vision werden CNNs aufgrund ihrer Behandlung von Merkmalen eines Bildes öfter genutzt als reguläre FCNNs. Eine früh entwickelte Denoising Methode mittels CNNs ist der Autoencoder.[Gon16]

2.4.1 Autoencoder

Ein regulärer CNN Autoencoder ist eine CNN-Netzwerk, das versucht eine Annäherung an die Identitätsfunktion zu erlernen.

$$y^i \approx x^i$$

Mit anderen Worten heißt das, dass der Output möglichst dem Input entsprechen soll.

Architektur

Der Input wird von durch einen Encoder abwechselnd von Convolutional- und Max-Pooling Layer codiert. Bei der Codierung handelt es sich um eine komprimierte Repräsentation des Inputs. Der Decoder besteht aus Convolutional- und Un-Pooling Layer. Durch die Un-Pooling Layer verdoppelt sich die Dimensionen des Inputs. Es handelt sich dabei um ein Upsampling der Codierung, um die originale Repräsentation wieder abzubilden.[Gon16]
Wie die Archtiektur im Detail aussieht wird im späteren Teil über die Implementierung genauer Erläutert.

Denoising mittels Autoencoder

Autoencoder werden größtenteils wegen ihrer Fähigkeit der Dimensionsreduktion durch den Encoder angewendet. Allerdings kann ein Autoencoder auch für das Denoising eingesetzt werden. Dazu muss sich an der Struktur nichts ändern, es reicht wenn als Trainingsinput Bilder mit Rauschen und als Ground Truth die jeweiligen Bilder ohne Rauschen genutzt werden. Bei einem trainierten Denoising Autoencoder soll das Rauschen beim codieren "verloren"gehen, indem es nicht als wesentliches Merkmal extrahiert wird. Der Decoder setzt dann die original Repräsentation ohne Rauschen zusammen.[Gon16]
Der einfache Autoencoder ist einer der ersten CNN Denoising Architekturen und dient für moderne Architekturen wie dem RED-Net.

2.4.2 RED-Net

Das RED-Net steht für very deep Residual Encoder-Decoder Networks und wird im Paper „Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections“von Xiao-Jiao Mao, Chunhua Shen und Yu-Bin Yang erstmals vorgestellt. Da sich in dieser Arbeit häufig auf dieses Paper bezogen wird, wird das Paper fortan "RED-Net Paper"genannt. Wie der Name andeutet handelt es sich hierbei um einen erweiterten Autoencoder. Es gibt ebenfalls einen Decoder und einen Encoder, allerdings wird hierbei komplett auf Pooling-Layer verzichtet. Der Encoder besteht rein aus Convolutional Layer. Das decoding wird von Deconvolutional Layer gehandhabt, welche eine transponierte Convolution ausführen. Durch das Verzichten auf Pooling- und Unpooling-Layer wird das "wegwerfen"von wesentlichen Details verhindert. Ein Padding sorgt dafür, dass alle Layer die gleiche Größe behalten.[MSY16]

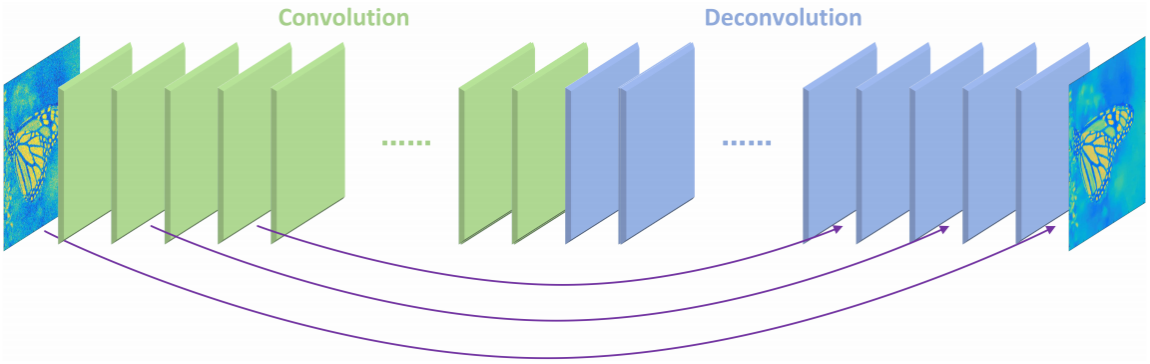


Abbildung 2.5: RED-Net[MSY16]

Das RED-Net verfügt über Skip-Connections die in der Abbildung als lilane Pfeile dargestellt werden. Skip-Connections wurden von K. He, X. Zhang, S. Ren und J. Sun das erste mal in einer CNN Architektur namens ResNet (Residual Networks) eingesetzt. Im Paper „Deep Residual Learning for Image Recognition“wird die Architektur und ihre hervorrangenden Ergebnisse beschrieben.[**he2016dee**]

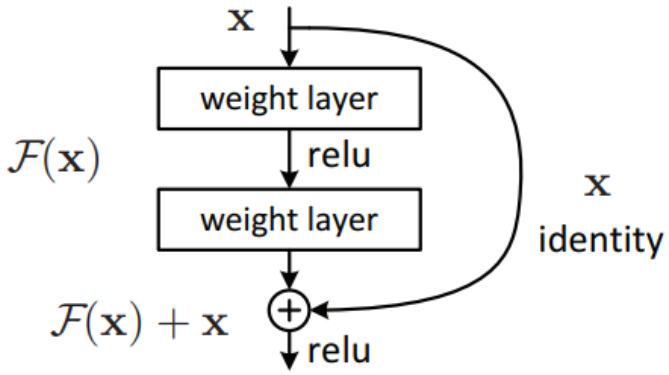


Abbildung 2.6: Skip-Connection[He16]

Skip Connections leiten zum einen Details direkt an hintere Schichten weiter. Bei dem RESNet bedeutet das konkret, dass die die Eingangswerte einer Schicht auf Ausgangswerte der folgenden Schicht addiert werden, wie in Abbildung 2.6 sichtbar. Zum anderen können die Gradienten von den hinteren Schichten nach vorne gereicht werden. Diese beiden Eigenschaften machen das Training des „end-to-end mappings“ vom korrumpierten zum bereinigten Image effektiver und genauer.[He16]
Die Besonderheit des RED-Nets ist allederings, dass die Skip-Connections nicht das vordere mit dem hinteren Layer verbinden, sondern Sie symetrisch die Layer des Encoder mit den gegenüberliegenden Layer des Decoder verbinden, wie in der Abbildung 2.5 sichtbar. Diese Änderung ermöglicht eine „elementweise Korrespondenz"“[MSY16, S. 5], die wichtig bei pixelweisen Vorhersageproblemen ist und die Leistung des Netzwerkes durch Tests nachgewiesen verbessert hat.[MSY16]

2.4.3 Kostenfunktion und PSNR

Die Kostenfunktion für das Denoising mit CNNs ist die bereits vorgestellte Mean Squarred Error (MSE) Funktion.[MSY16]. Die Funktion berechnet hierbei durchschnittliche

Abweichung der Pixel zwischen Ground Truth Bild und dem Output des Modells. Neben dem MSE wird auch die Peak signal-to-noise ratio berechnet. Die PSNR bestimmt das Signal-Rausch-Verhältnis und ist eine oft genutzte Metrik in der Bildrekonstruktion mit NNs. Dabei beruht ihr Wert auf den MSE und den Maximalen Pixel Wert. Im Gegensatz zur MSE ist bei der PSNR ein hoher Wert positiv. Sie wird mittels folgender Formel ermittelt:

$$10 * log_{10}(\frac{{I_{max}}^2}{MSE})$$

Was umgeschrieben werden kann zu:

$$20 * log_{10}(I_{max}) - 10 * log_{10}(MSE)$$

I_{max} ist dabei der Maximale Pixel Wert, der entweder immer 1.0 oder 255 ist. Der PSNR hat den Vorteil gegenüber dem MSE ,dass er die Maximalen Werte in Betracht zieht und daher der PSNR immer im Bereich $0 - 100$ liegt. Wodurch er sich letztendlich besser als Metrik eignet. [HZ10]

Kapitel 3

Analyse

Nachdem eine Basis der theoretischen Grundlagen geschaffen worden ist, werden in diesem Teil der Arbeit zunächst der zu bereinigenden MRT-Bilder, also der verwendete Datensatz, und die daraus resultierende Problemstellung erläutert.

3.1 Datensatz

Die für dieser Arbeit zur Verfügung gestellten Daten stammen aus der "The Healthy Brain Network Biobank", die in Rahmen einer Open-Resource transdiagnostischen Forschung entstanden ist.[Ale17] Hierbei dürfen Nutzer der Website http://mindcontrol-hbn.herokuapp.com/ nach einer Richtlinie die Bild-Qualität von MRT Gehirn-Bildern bewerten. Die Bewertungen kann jeder vornehmen es bedarf also keiner Qualifikation um Bewertungen für Bilder abgeben zu dürfen. In dieser Arbeit davon ausgegangen dass die Bewertungen unter Einhaltung der Richtlinien vorgenommen wurden.

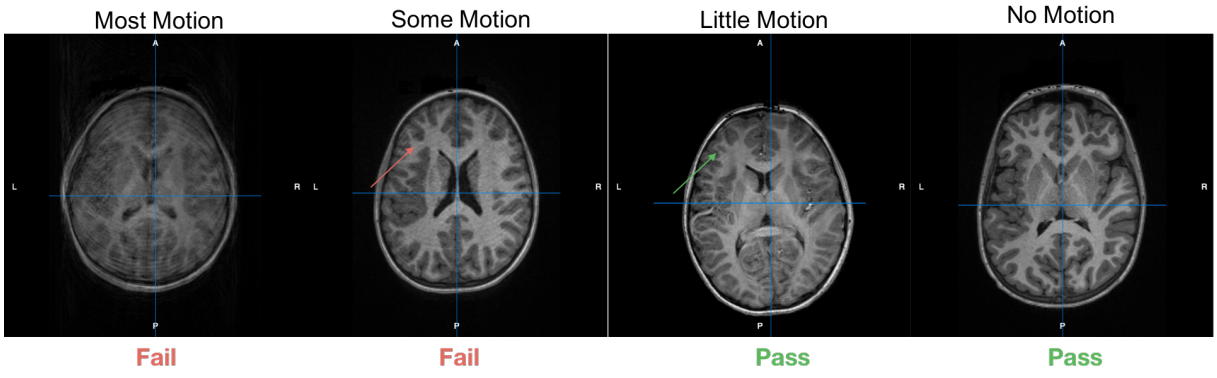


Abbildung 3.1: Richtlinie für Bewertung von Bewegungsartefakten

Der Datensatz umfasst zum einen 83145 Qualitätsbewertungen zu MRT-Bildern und zum anderen die dazugehörigen 3609 MRT-Bilder. Diese haben mindestens 19 Bewertungen. Die Bewertungen reichen von 0 (fail) bis 1 (pass). Die Bilder lassen sich so nach einer durchschnittlichen Benotung sortieren.

3.2 Problemstellung

Bei einer Anschauung der Bewertungen und Bilder wird deutlich, dass mittlere bis starke Bewegungsartefakte eine häufige Ursache für eine schlechte Bewertung sind. Im Gegensatz zu anderen medizinischen Artefakten wie metallische Fremdkörper oder Blutungen sind Bewegungsartefakte, wie in der Abbildung **??** dargestellt, auch für einen Laien erkennbar. Des weiteren stellen Bewegungsartefakte wie in 1.1 Motivation bereits erwähnt ein besonderes Problem für medizinische Anwendungen dar und ihre Entfernung somit ein lohnendes Ziel.
Zusammenfassend wurde aufgrund des hohen Vorkommens, der Erkennbarkeit und der medizinischen Problematik, das Entfernen von Bewegungsartefakten mittels CNNs als der zentrale Punkt der Arbeit gewählt. Da der Artefaktgenerator hauptsächlich im Rahmen dieser Arbeit funktional sein soll, wird davon abgesehen ihn ausführlich zu testen.

Kapitel 4

Konzeption

In diesen Kapitel wird ein Bild von der Idee und der Struktur des Projektes geschaffen bevor die Implementierung im Detail erläutert wird. Dafür wird zunächst die Wahl der Denoising CNNs begründet. Anschließend wird erläutert warum ein Bewegungsartefakt-Generator für das Trainieren dieser Notwendig ist.

4.1 Wahl der CNN-Architekturen

Als Basis für das Projekt wurde der im Theorieteil erläuterte Autoencoder gewählt. Dieser lässt sich aufgrund seiner Einfachheit ohne großen Aufwand implementieren und schafft ein tieferes Verständnis für das Denoising mit CNNs. Das Prinzip des Autoencoder dient außerdem als Basis für moderne Denoising-Architekturen und schafft damit eine Brücke zu dem RED-Net. Das RED-Net wurde gewählt da es in Image Denoising sehr gute Ergebnisse erzielt hat und das das Prinzip eines Autoencoders besitzt. Das Anwenden des RED-Nets bietet außerdem einen persönlichen Wissensgewinn. So lassen sich beispielsweise Erkenntnisse über die Vorteile von Deconvolution gegenüber dem Upsampling des Basic-Autoencoders gewinnen.

4.2 Generierung von Artefakten

Das Trainieren der beiden CNN-Modellen benötigt wie im Theorieteil bereits beschrieben, zu jedem Datensatz x einen gelabelten Datensatz $\hat{y}$. Aus den MRT-Bildern des Datensatzes werden die Bilder mit keinem oder sehr geringem Bewegungsrauschen genommen. Sie

entsprechen dann $\hat{y}$, werden also mit den Output y der CNNs verglichen, um den Loss zu bestimmen. Da allerdings keine Input Daten x, also Bilder die identisch zu $\hat{y}$ aber mit einem Rauschen bzw. Artefakt versehen vorhanden sind, müssen diese künstlich generiert werden. Für den Fall, dass generieren von künstlichen Bewegungsartefakten nicht gelingt, sollen zusätzlich Trainingsdaten mit einem einfachen gaußschen Rauschen generiert und die Modelle mit diesen trainiert und evaluiert werden.

4.3 Projektstruktur

Das Projekt besteht aus folgenden Klassen mit angegebenen Rollen:

preprocessing.ipynb
: Daten -Vorverarbeitung und -Aufteilung

autoencoder.py
: Basic Autoencoder-Modell

rednet.py
: Red-Net-Modell

artefactgenerator.py
: Generieren von Bewegungsartefakten und gaußschen Rauschen

utils.py
: Bieten von Hilfsfunktionen

train.py
: Training der Modelle mit Artefaktgenerator

test.ipynb
: Auswertung des Trainings und dertrainierten Modelle

4.4 Randbedingungen

Um die benötigten Rechenkapazitäten für das Trainieren der Modelle zu erhalten wird der DT-Cluster der HTW- Berlin genutzt. Damit kann maximal 20 GB Arbeitsspeicher und eine Grafikkarte mit 11GB Grafikspeicher genutzt werden.

Nach der Beschreibung der Projektstruktur wird wird im nächsten Kapitel die Umsetzung des Projektes beschrieben.

Kapitel 5

Umsetzung

In diesem Kapitel wird die Umsetzung des Projektes beschrieben. Dafür werden in den einzelnen Abschnitten die im vorherigen Kapitel aufgelisteten Klassen jeweils mit ihren Funktionen beschrieben. Dabei sollen auch Gedankengänge geschildert werden. Das Projekt wurde komplett in der Programmiersprache Python geschrieben. Das Projekt findet sich auf dem beiläufigen Datenträger in dem Ordner project. Dort findet sich auch eine Readme Datei mit den zu installierenden Bibliotheken und einer Erklärung zur Ausführung des Projektes.
Zu beginn dieses Kapitlels wird ein kurzer Überblick über die in diesem Projekt essenziellen Python-Bibliotheken gegeben.

5.1 Bibliotheken

Numpy

Numpy ist eine grundlegende Bibliothek für wisssenschaftliche Berechnungen in Python.[Oli06] In dieser Arbeit werden alle Bilderdaten innerhalb von Numpy-Arrays bearbeitet.

TensorFlow und Keras

TensorFlow ist eine kostenlose Open-Source Softwarebibliothek für maschinelles Lernen und bietet eine Reihe an Werkzeugen um maschinelle Lernanwendungen zu erstel-

len. Keras ist TensorFlows API für das Erstellen und Trainieren von neuronalen Netz-Modellen.[Mar15]

Scikit-image

Scikit-image ist eine Sammlung von Algorithmen für die Bildverarbeitung und der Computer Vision.[Wal14]

5.1.1 Datenvorverarbeitung

Wie bereits in Abschnitt 4.2 beschrieben brauchen wir für das Generieren von MRT-Bewegungsartefakten Bilder ohne Artefakte. Durch Betrachtung der Bilder und ihrer durchschnittlichen Bewertungen lässt sich feststellen, dass von den 1500 bestbewerteten Bildern die meisten ohne oder sehr geringe Bewegungsartefakte haben und die Bewegungsartefakte mit schlechter werdenden Bewertungen häufiger und stärker werden.
Da eine händische Auswahl der Bilder langwierig ist, wurden zunächst aus dem vorhandenen Datensatz die bestbewerteten 1000 Bilder genommen. Dies funktioniert durch ein Mapping der Bildernamen aus der CSV entnommenen Rangliste mit denen des Datensatzes. Die Bilder des Datensatzes haben eine Größe von 256 Pixel Breite und Höhe. Nach Training der Modelle hat sich allerdings herausgestellt, dass die Größe zu gering ist, um eine Analyse durch Anschauung der Testresultate zu ermöglichen, da viele Details nicht erkennbar sind.
Mittels Url-Request lassen sich die die Bilder auch von der Website downloaden. Bei einem Download aller Bilder die im Datensatz enthalten sind zeigt sich, dass 1894 der Bilder auch in einer Höhe und Breite von 320 Pixel vorhanden sind und damit über 56.25% mehr Pixel verfügen. In Anbetracht dessen wurde sich für den Schritt entschieden die Datenbasis auf die 320 Pixel Bilder zu reduzieren und damit in etwa zu halbieren und dafür eine höhere Datenqualität zu erhalten.
Um genügend Bilder für das Generieren von Bewegungsartefakten zu erhalten wurden aus der neuen Datenbasis die Bilder genommen die zu den bestbewerteten 1200 Bildern gehören. Dies macht 516 Bilder, die nach Durchmischung in einem 80 zu 20 Verhältnis in Trainings und Evaluationsdaten aufgeteilt sind. Dabei wurde bei einer Anschauung einer Stichprobe von 50 Bildern ein Bild mit mittleren und eins mit sehr leichtem Rauschen entdeckt, was für vertretbar gehalten wird.
Für das Generieren von gaußschem Rauschen werden alle 1894 Bilder zu 60 20 20 Trainings-

Evaluation- und Testdaten aufgeteilt , da in der Datenbasis keine realen gaußschen Artefakte vorhanden sind.

5.2 Autoencoder-Modell

Der Basic Autoencoder wurde mittels Tensorflow.Keras selbst programmiert. Das Model besteht aus den in Kapitel 2.4.1 definierten Komponenten und wurde dem Paper „Medical image denoising using convolutional denoisingautoencoders“ nachempfunden.[Gon16] Die Architektur mit der nach ersten Tests die besten Ergebnissen erzielt wurde sieht wie folgt aus:

Layer (type)	Output Shape	Param #
inputs (InputLayer)	[(None, 320, 320, 1)]	0
encoder_conv1 (Conv2D)	(None, 320, 320, 64)	640
encoder_pool1 (MaxPooling2D)	(None, 160, 160, 64)	0
encoder_conv2 (Conv2D)	(None, 160, 160, 32)	18464
encoder_pool2 (MaxPooling2D)	(None, 80, 80, 32)	0
decoder_conv1 (Conv2D)	(None, 80, 80, 32)	9248
decoder_upsamp1 (UpSampling2	(None, 160, 160, 32)	0
decoder_conv2 (Conv2D)	(None, 160, 160, 64)	18496
decoder_upsamp2 (UpSampling2	(None, 320, 320, 64)	0
outputs (Conv2D)	(None, 320, 320, 1)	577

Total params: 47,425
Trainable params: 47,425
Non-trainable params: 0

Abbildung 5.1: Modell-Struktur des Autoencoders

5.3 RED-Net-Modell

Die Implementierung des RED-Net wurde hauptsächlich von den Github User *pmcbride* übernommen. Der Link zum Repository findet sich auf dem beigelegten Datenträger der Klasse rednet.py.
Die Implementierung orientiert sich dabei an dem RED-Net Paper. In dem Paper wird die Architektur mit 10 (RED10), 20 (RED20) und 30 (RED30) Layern getestet. Für eine Erhöhung der Layer Anzahl müssen hierbei nur Parameter angepasst werden. An der Implementierung musste nur die Dimension der Kernel an dem Input und Output Größe

der Bilder angepasst werden. Die Architektur des RED10 mit 32er Kernelgröße schaut wie folgt aus:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
encoder_inputs (InputLayer)  [(None, 320, 320, 1)]     0
_________________________________________________________________
encoder_conv1 (Conv2D)       (None, 320, 320, 32)      320
_________________________________________________________________
encoder_conv2 (Conv2D)       (None, 320, 320, 32)      9248
_________________________________________________________________
encoder_conv3 (Conv2D)       (None, 320, 320, 32)      9248
_________________________________________________________________
encoder_conv4 (Conv2D)       (None, 320, 320, 32)      9248
_________________________________________________________________
encoder_conv5 (Conv2D)       (None, 320, 320, 32)      9248
_________________________________________________________________
decoder_deconv1 (Conv2DTrans (None, 320, 320, 32)      9248
_________________________________________________________________
decoder_deconv2 (Conv2DTrans (None, 320, 320, 32)      9248
_________________________________________________________________
decoder_deconv3 (Conv2DTrans (None, 320, 320, 32)      9248
_________________________________________________________________
decoder_deconv4 (Conv2DTrans (None, 320, 320, 32)      9248
_________________________________________________________________
decoder_deconv5 (Conv2DTrans (None, 320, 320, 1)       289
=================================================================
Total params: 74,593
Trainable params: 74,593
Non-trainable params: 0
_________________________________________________________________
```

Abbildung 5.2: Modell-Struktur des RED-Nets

Die Darstellung beinhaltet der Übersichtlichkeit halber nicht die Skip-Connections. Die Skip-Connections haben eine Schrittgröße von 2. Im Fall des RED10 bedeutet, dass es zwei Connections gibt. Eine verbindet den Input des ersten Convolution Layers mit dem Output des vierten Deconvolution Layers. Die andere verbindet den Input des dritten Convolution Layers mit dem Output des zweiten Deconvolution Layers.

5.4 Artefact Generator

Die Klasse *artefactgenerator.py* verfügt über Funktionen sowohl zur Generierung von künstlichen Bewegungsartefakten, als auch gaußschen Rauschens.

Gaußsches Rauschen

Gaußsches Rauschen wird mit der Funktion $skimage.util.random_noise$ der Scikit-image Bibliothek generiert. Dabei bestimmt ein zufällig gewählter Varianz-Wert die Stärke des

Rauschens.

5.4.1 Bewegungsartefakte

Für das Generieren der Bewegungsartefakte muss zunächst eine Basis für das Aussehen von Bewegungsartefakten geschaffen werden.

Abbildung 5.3: Beispiel von Bewegungsartefakten

In der Abbildung 5.3 sieht man auf der linken Seite ein Bild mit starken Bewegungsartefakten, auf der rechten Seite ein Bild ohne Bewegungsartefakten. Bei Betrachtung von Abbildung wird deutlich, dass sich Bewegungsartefakte wellenartig äußern. Die Form der Wellen entspricht der des Randes des Objektes bzw. des Gehirns. Die Stärke der "Wellen"nimmt von außen nach innen ab.
Der erste Ansatz war zunächst ein Bild mit sich selbst zu überlappen. Die Überlappung zweier Bilder gelingt dabei mit der Addition ihrer Werte. Das zu überlappende Bild wird dabei vorher mit einen Faktor um ein vielfaches abgeschwächt. Außerdem findet jede Überlappung um eine Position definiert durch x und y versetzt statt. Das Resultat dieses Ansatzes entsprach einem weißem Blur-Effekt, da die mehrfachen Additionen die Werte über das ganze Objekt hinweg erhöhen.
Da wie bereits erwähnt die Form der Wellen denen des Randes des Gehirns ähneln, sollte es ausreichen ein Objekt in Form des Randes mit dem Bild zu überlappen. Dies würde dafür Sorgen, dass nur an wenigen Stellen, also da wo sich der Rand mit den Bild überlappt, die Werte addiert werden. Die Form des Randes erhält man mit der Anwendung eines Sobelfilters, oder auch Kantendetektionsfilter. Der Sobelfilter erkennt Kanten in einem Wert als Kante, indem er auch seine Nachbarwerte betrachtet.[GM13] Er ähnelt vom Vorgehen also einem Convolutional Filter, der Kanten extrahiert. Eine Sobelfilter-Funktion

wird ebenfalls von Scikit-image bereitgestellt.
Die Überlappung des Sobel-Bildes auf dem Original-Bild wird im groben ausgeführt wie im ersten Ansatz beschrieben. Das Objekt im Sobelbild wird durch Kürzung des Sobelbildes um die Größe der Positionsänderung verschoben. Für eine Überlappung wird anschließend der gleiche Abschnitt des ungekürzten Original-Bildes mit dem Sobel Bild addiert.

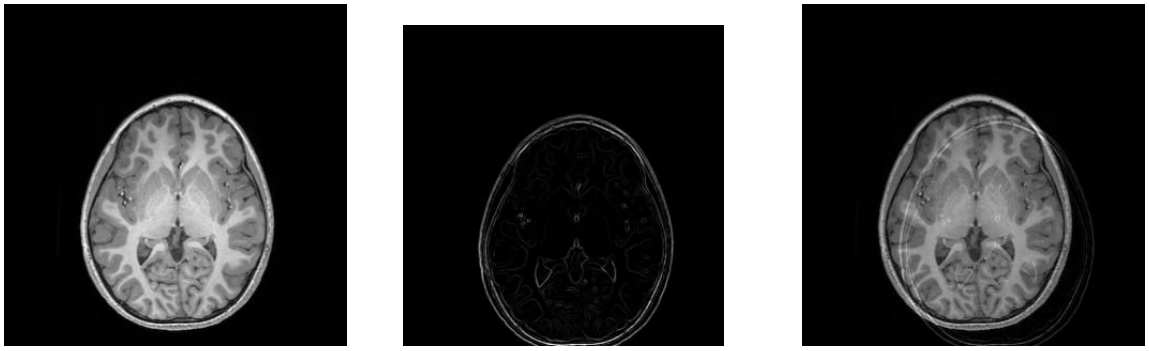


Abbildung 5.4: Überlappung eines Sobels

Für Abbildung 5.5 wurde aus veranschaulichungs Gründen ein sehr hoher Überlappungsfaktor verwendet was dazu führt, dass das Bild verdunkelt. Realer wirkende Überlappungen werden durch einen niedrigeren Faktor erzeugt.
Die Überlappung findet in der Funktion overlap_image statt. Diese wird von zwei weiteren Funktionen aufgerufen die verschiedene Ansätze verfolgen.

Symmetrische Überlappungen

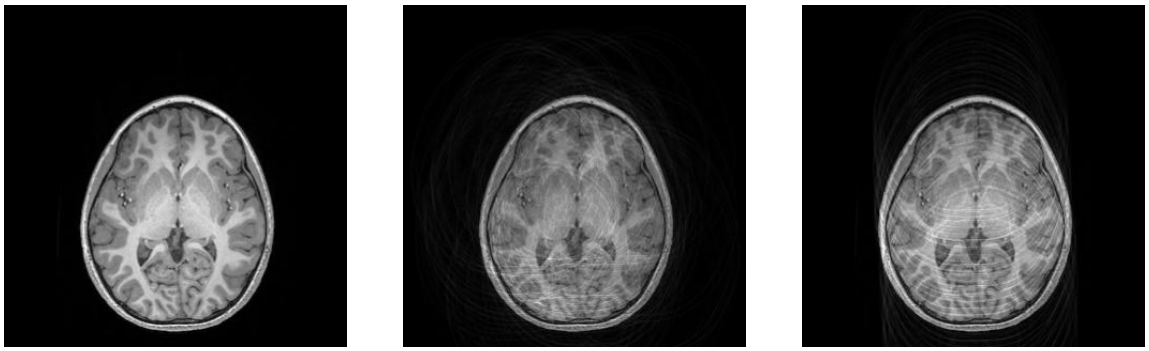
In der Funktion symmetric_overlapping wird versucht ein möglichst reales Bewegungsrauschen zu erzeugen, indem jede Überlappung symmetrisch nach oben und nach unten versetzt stattfindet. Ein Überlappungsfaktor wird bei jeder Überlappung zufällig gewählt, allerdings wird dieser nach jeder Überlappung leicht abgeschwächt. Die Anzahl der Überlappungen werden zufällig zwischen 4 und 15 gewählt. Ebenso der Abstand zwischen einer Überlappung zur nächsten. Außerdem gibt es einen zufällig gewählten Faktor Multiplikator der die generelle Rauschstärke eines Bildes bestimmt. Diesen Faktor gibt es auch bei den zufälligen Überlappungen.

Zufällige Überlappungen

In dieser random_overlapping soll ein eher zufälliges Rauschen generiert werden, um eine bessere Generalisierung zu ermöglichen. Dabei gibt es einen zufälligen Wert der die Anzahl und einen der das Streuungsmaß der Überlappungen festlegt. Das Streuungsmaß legt den Rahmen Fest indem die Überlappungen stattfinden können. Je kleiner das Steuungsmaß desto näher liegen die Überlappungen beieinander. Eine Verdunklung des Bildes wird hervorgerufen bei mehrmaligen Überlappungen an den gleichen Positionen. Dies liegt daran, dass sich eine Überlappung am selben Wert, dieser durch die mehrfache Addition stark erhöht wird und sich so ein sehr starker Maximalwert ergibt. Der Kontrast wird mit der Differenz des Maximalwertes und des Minimalwertes berechnet, welcher hierbei immer 0 ist. Steigt also der Maximalwert wird das Bild dunkler. Um die Verdunklung zu reduzieren wird bei der Stärke des Überlappungsfaktors das Streuungsmaß und die Anzahl der Überlappungen berücksichtigt.

Für die folgende Abbildung **??**sieht man links das Original-Bild, in der Mitte ein Bild mit zufälligen- und rechts ein Bild mit symmetrischen Überlappungen. Für beide Bilder wurden durchschnittliche Parameter Werte gewählt. Sie stellen also ein mittelstarkes Rauschen dar.

Abbildung 5.5: Künstliche Bewegungsartefakte



Wie in den Abbildungen ersichtlich gelingt keine komplette Verhinderung der Verdunklung durch Überlappung. Allerdings schafft es gerade die symmetrische Variante das wellenartige Struktur von Bewegungsartefakten zu simulieren. Es gelingt auch eine Sobel Abbildung an Stellen außerhalb des Gehirns zu verringern: Die schwarzen Flächen haben geringere Werte als der Durchschnitt des Bildes. An diesen Stellen wird der Sobel-Faktor um einen Multiplikator verringert.

5.5 Training

Die Klasse *train.py*läd die Daten und Modelle und startet anschließend das Training. Außerdem speicherte sie den Trainingsverlauf und das trainierte Modell für spätere Tests. Durch das Ausführen mit Angabe von Parametern lässt sich beispielsweise aussuchen welches Modell man trainieren will und auf welche Art von Rauschen trainiert werden soll. Dazu gibt es mehr Informationen in der Readme-Datei im Anhang. Als Optimierungsverfahren wird Adam mit einer Lernrate α von 0.001 verwendet. Dieser wurde in dem "RED-Net Paperëmpfohlen und ermöglicht auch bei dem Autoencoder ein vergleichsweise schnelles Lernen. Kostenfunktion ist der MSE. Die Funktion zur Berechnung des PSNR befindet sich in der *utils.py* Klasse. Die Klasse hat eine Funktion zum generieren von Trainings- und Evaluationsdaten zur Laufzeit des Trainings. Dafür werden die Funktionen des $artefactgenerator.py$ genutzt.

5.6 Evaluation

Die Klasse *test.ipynb* visualisiert und berechnet die Ergebnisse die im nächsten Kapitel vorgestellt werden. Dabei nutzt die Klasse Funktionen der $utils.py$.

Kapitel 6

Evaluation

In diesem Kapitel werden die Ergebnisse zusammengetragen und analysiert. Zunächst werden die Trainings- und Evaluations- Ergebnisse der beiden Modelle im Beseitigen von gaußschen Rauschen und künstlichen Bewegungsartefakten verglichen. Anschließend wird das Beseitigen von realen Artefakten mit dem RED-Net getestet.

6.1 Entfernung von gaußschem Rauschen

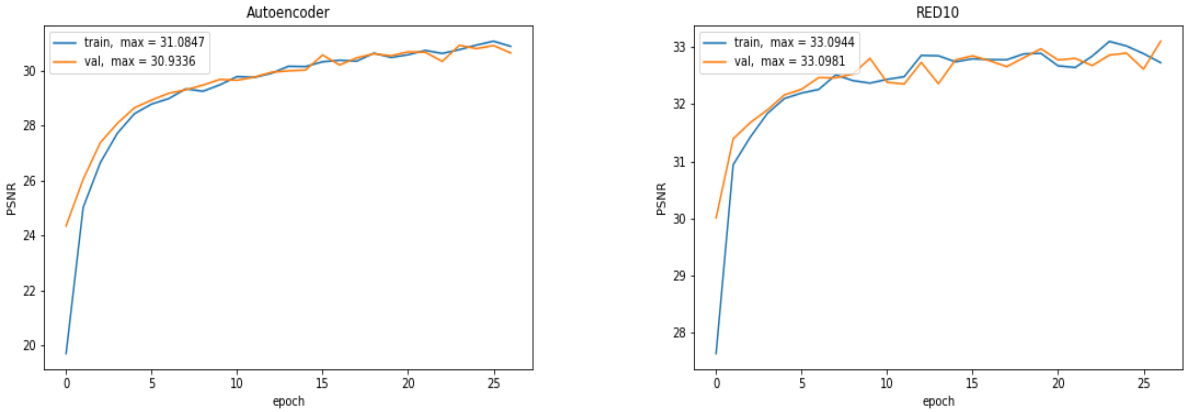


Abbildung 6.1: Trainingsverlauf des RED-Nets und des Autoencoders im Entfernen von gaußschem Rauschen

Für das Trainieren mit Gaußschem Rauschen wurden die beiden in der Konzeption dargestellten Modelle verwendet. Da sich die PSNR aus dem MSE ergibt, wird nur der PSNR dargestellt. Ein hoher Wert ist hierbei positiv. In Abbildung 6.1 zeigt sich dass das RED10

bei der Beseitigung von gaußschen Rauschen bessere Resultate erzielt. Es hat sich außerdem gezeigt, dass der Autoencoder bei hohen Gaußschem Rauschen nicht lernen kann. Daher wurde für das Training beider Modelle die maximale Varianz auf 0.2 reduziert.

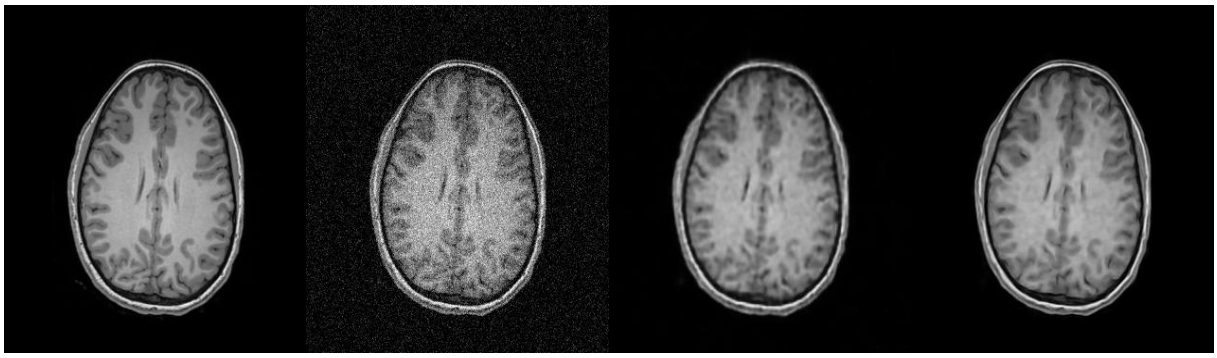


Abbildung 6.2: Resultate der Modelle im Entfernen von Gaußschem Rauschen

In der Abbildung 6.2 ist auf der linken Seite das Ground Truth Bild zu sehen. Rechts davon das mit gaußschem Rauschen versetzte. Davon wieder rechts das bereinigte Bild durch den Autoencoder. Auf der Rechten Seite das bereinigte Bild durch das RED-Net. Es zeigt sich dass es beiden Modellen das Rauschen zu entfernen. Allerdings gelingt es dem RED-Net einen höheren Detailgrad zu bewahren. Dies lässt sich durch den Verzicht von einfachen Down- und Upsampling mittels Pooling-Layer erklären, durch denen Details verloren gehen. Für die Entfernung der künstlichen Bewegungsartefakte wurde daher nur das RED-Net verwendet, da die Bewahrung von Details gerade für medizinischen Anwendungen eine wichtige Anforderung ist.

6.2 Entfernung von künstlichen Bewegungsartefakten

Es wurden Modelle mit den beiden verschiedenen Simulationen von Bewegungsrauschen trainiert. Außerdem Modelle mit einer Kombination beider Rauschen trainiert, indem für jeden Input zufällig aus eins der beiden Rauschen ausgewählt wird. Anhand der Trainingshistorien lassen sich die die Resultate der verschiedenen Parameter ablesen.

Die Tabelle 6.1 zeigt alle getesteten Varianten des RED-Net mit den aufgezeigten Parametern. Die PSNR ist der maximal erreichte Durchschnittswert aller Epochen. Das Modell wurde nach der jeweiligen Epoche gespeichert. Die Nummern im Modell-Namen geben die Anzahl der Schichten an. RED10 bedeutet, dass es 5 Convolutional und 5 Deconvolutional Layer gibt. Die Anzahl der Kernel bestimmt die Tiefe der Layer. Bei der Auswertung zeigt

Entfernung von symmetrischen Artefakten			
Modell	Anzahl Kernel	Batchgröße	PSNR
RED10	**64**	**1**	**29.926**
RED10	32	1	29.683
RED10	32	5	29.456
RED10	64	5	29.340
RED10	64	10	28.748
RED20	32	1	28.487
RED20	64	1	28.274
RED10	32	10	27.981
RED20	64	10	27.635
Entfernung von zufälligen Artefakten			
Anzahl Layer	Anzahl Kernel	Batch-Größe	PSNR
RED10	**64**	**1**	**29.355**
RED10	32	1	29.245
RED10	64	5	28.686
RED10	64	10	28.202
RED10	32	10	27.994
RED10	32	5	27.763
Entfernung von kombinierten Artefakten			
Anzahl Layer	Anzahl Kernel	Batchgröße	PSNR
RED10	**32**	**1**	**29.513**
RED10	64	1	29.355
RED10	128	1	29.138
RED10	64	5	29.007
RED10	32	5	27.877

Tabelle 6.1: RED-Net: Evaluation von Parametern und Trainingsdaten

sich, dass ein Modell mit 10 Layern die besten Ergebnisse erzielt. Dies entspricht nicht den Ergebnissen des bezüglichen Papers indem eine Erhöhung der Layeranzahl in einer höheren PSNR resultiert. Der SGD ohne Mini-Batch die besten Ergebnisse und Bestätigt hingegen das Paper. Je größer der Batch desto schlechter sind die Ergebnisse ausgefallen. Die Modelle die für das weitere Testen ausgewählt wurden sind hervorgehoben.

Vor dem Testen mit realen Bewegungsartefakten, wird geprüft ob der durchschnittliche Output eines Modells eine bessere PSNR zu den Groundtruth Bilder hat als die mit künstlichen Bewgungsartefakten versehenen Bilder. Dies soll testen ob durch die Bereinigung der Bilder eine messbare Verbesserung gegeben ist. Dafür wird das mit der Kombination beider Rauschen trainierte Modell verwendet, dabei ist davon auszugehen, dass die anderen Modelle in ihrem jeweiligen Spezialgebiet bessere Ergebnisse erzielen. Hierfür wird der Durchschnitt aus 25 Bildern verwendet.

Der PSNR zwischen Groundtruth Bildern und mit symmetrischem Rauschen versetzten Bildern liegt bei 28.272. Der PSNR zwischen Groundtruth und mit zufälligem Rauschen versetzten Bildern bei 26.098. Die von dem Modell bereinigten Bilder mit symmetrischen Rauschen haben eine PSNR zu den Groundtruth von 29.699. Die mit zufälligem Rauschen bereinigten Bilder haben eine PSNR von 28.272. Die Ergebnisse sprechen dafür, dass das Modell durch das Bereinigen der Artefakte die Bilder wieder näher ans Original bringen kann. Dies soll nun durch das Entfernen von realen Artefakten geprüft werden. Da es dabei aber keine Ground Truth Bilder gibt, kann der PSNR dafür allerdings keine Metrik sein.

6.3 Entfernung von realen Bewegungsartefakten

Für das Testen wurden 16 Bilder mit leichten, mittleren und starken Artefakten handlich ausgewählt. Die drei ausgewählten Modelle haben diese jeweils bereinigt. Dabei hat der Output des symmetrisch trainierten Modells mit einem durchschnittlichen PSNR von 34.827 den gerginsten von allen Modellen. Das zufällig trainierte Modell weißt einen PSNR von 32.152 auf und das Kombinations-Modell einen von 33.480. Dies lässt darauf schließen, dass das symmetrische Modell den Detailgrad des Inputs am wenigsten verändert. Bei visueller Betrachtung der Resultate konnte keins der Modelle präferiert werden. Die Leistung im Entfernen der Artefakte wirkt in etwa gleich. Aufgrund der besseren PSNR steht im folgenden die visuelle Analyse der Resultate des symmetrisch trainierten Modells im Fokus. Diese kann Aufgrund ihrer Beschaffenheit etwas Subjektiv ausfallen. Für eine

genauere Betrachtung aller Ergebnisse empfiehlt sich ein Blick in die *test.ipynb* Datei auf dem beigelegten Datenträger.

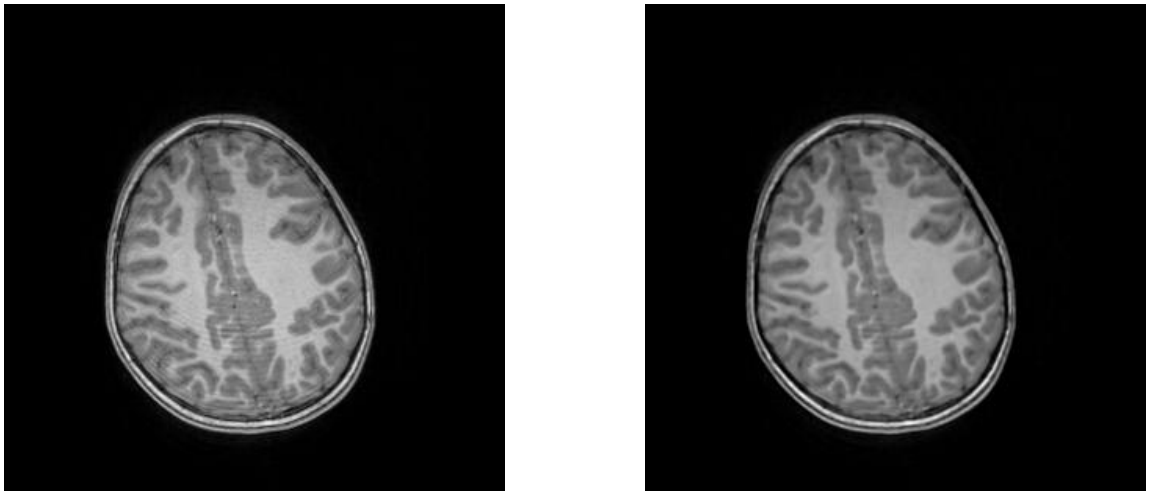


Abbildung 6.3: Entfernen von leichten Bewegungsartefakten

In der Abbildung 6.3 sieht man auf der linken Seite ein MRT-Bild mit Bewegungsartefakten und auf der rechten Seite das bereinigte Bild durch das Modell. Es zeigt sich, dass es dem Modell möglich ist leichte Bewegungsartefakte zu entfernen. Es sind nur noch vereinzelnd wellenartige Artefakte sichtbar. Der Detailgrad des Bildes bleibt dabei gut erhalten.

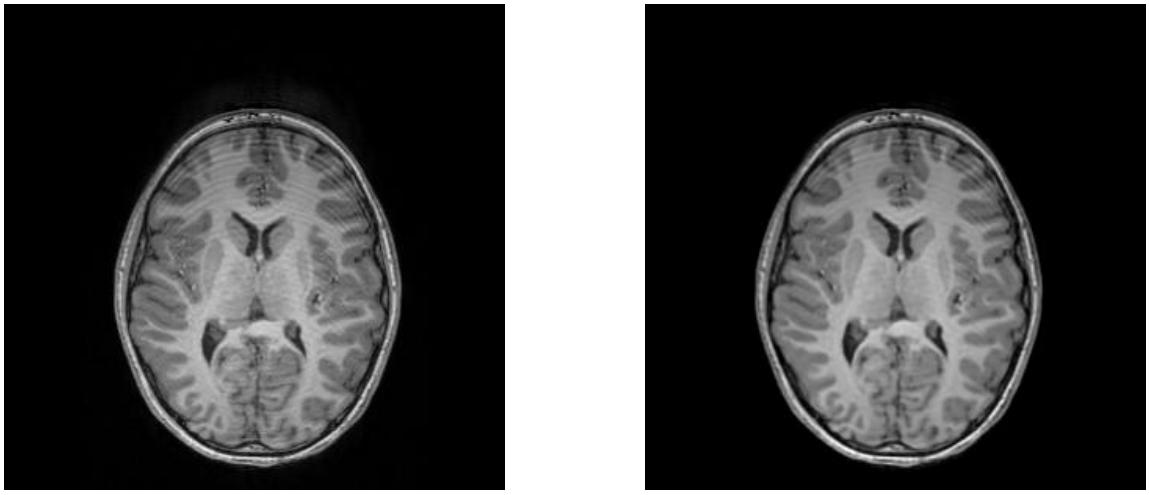


Abbildung 6.4: Entfernen von mittleren Bewegungsartefakten

Das Entfernen von mittleren Bewegungsartefakten gelingt zum Teil. Es zeichnet sich ab, dass die dünneren Wellen von dem Modell erkannt und entfernt werden. Dickere Wellen im oberen Teil des Gehirns werden ignoriert. Dies kann daran liegen, dass die beim Generieren überlappten Sobel eine ebenfalls dünnere Form haben.

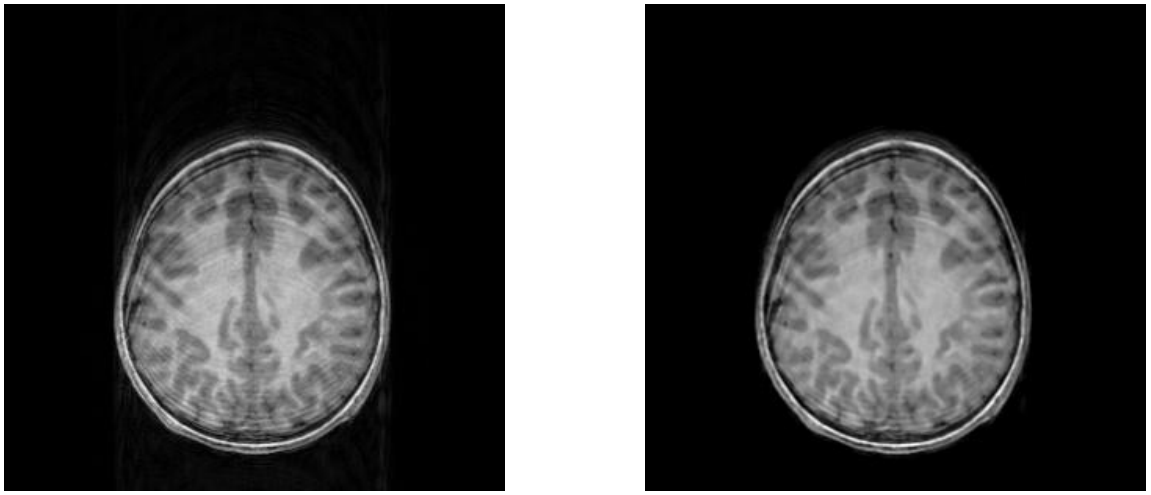


Abbildung 6.5: Entfernen von starken Bewegungsartefakten

Das Entfernen von Starken Bewegungsartefakten gelingt dem Modell dementsprechend kaum.
Insgesamt zeigen die Ergebnisse jedoch, dass ein Entfernen von Bewegungsartefakten auf Basis eines Generators möglich ist. Der erhaltene Detailgrad der bereinigten Bilder stellt ein zufriedenstellendes Ergebnis dar. Auch eine Veränderung des Kontrastes durch die Bereinigung fällt wenn nur sehr gering aus.

Kapitel 7

Fazit

7.1 Zusammenfassung

Die Bildrekonstruktion von MRT-Bildern mittels Convolutional Neural Networks ist die zentrale Problemstellung dieser Arbeit. Dabei konnte zunächst erfolgreich ein einfaches gaußsches Rauschen künstlich generiert und mittels zweier Modelle, dem Autoencoder und dem RED-Net entfernt werden. Es gelingt ebenfalls künstliche Bewegungsartefakten für ein Training auf das bereinigen dieser zu generieren. Das RED-Net wurde in verschiedenen Varianten anschließend erfolgreich mit unterschiedlich generierten Artefakten trainiert. Anhand einer Analyse der Ergebnisse wurden die besten Modelle für ein entfernen von realen Bewegungsartefakten ausgewählt. Diesen Modellen ist das Entfernen von realen Bewegungsartefakten erfolgreich möglich. Dabei lassen sich leichte bis mittlere Bewegungsartefakte mit gutem Ergebnis entnehmen. Das Problem des Entfernens von starken Artefakten bleibt hingegen ungelöst. In Anbetracht der Resultate kann gesagt werden, dass es in dieser Arbeit gelungen ist die Zielsetzung zu erreichen. Neben des implementierten Projektes konnte auch Wissen erzielt werden. Dabei konnten im Bereich der Bildverarbeitung im Rahmen der Simulierung von Artefakten Erkenntnisse gewonnen werden. Die Implementierung, das Training und das Testen der CNN Modelle konnten vorhandene Kenntnisse im Bereich des maschinelles Lernen erweitern. Obwohl die Zielsetzung erfüllt wurde gibt es noch Raum für Kritik.

7.2 Kritischer Rückblick

Die Auswahl des Autoencoders als CNN Modell hat einen guten Einstieg in das Denoising mit CNNs geschaffen. Es war allerdings davon auszugehen, dass das RED-Net erheblich bessere Resultate liefert. Man hätte also die Auswahl auf eine ebenfalls moderne Architektur treffen können. Dies würde die Resultate vergleichbarer machen und sich dadurch evtl. weitere Erkenntnisse erschließen. Aus Zeitgründen wurde das Implementieren und testen eines weiteren Modells allerdings abgelehnt.
Ein weiterer Kritikpunkt ist die vorhandene Aufteilung von Bild-Arrays in Patches vor dem Training mit einem CNN. Dieses Vorgehen wird im RED-Net Paper empfohlen. Patches beinhalten die Werte von benachbarten Pixeln. Dafür wird ein Bild von einem Kernel mit einer festgelegten Größe und einem Stride unterteilt. Durch entstehende Überlappungen wird die gesamt Anzahl von zu lernenden Pixeln erhöht.[MSY16] An diese Umsetzung wurde sich in dieser Arbeit versucht allerdings ist sie nicht gelungen und wurde aus zeitlichen Gründen schließlich gestrichen. Eine erfolgreiche Umsetzung hätte die Resultate nochmal verbessern können.
Da die Arbeit sowohl versucht die Modelle als auch die Generatoren zu optimieren, wurden Ressourcen nicht immer optimal eingesetzt. Es wäre möglich, dass bessere Ergebnisse erzielt worden wären bei einem größeren Fokus auf die Optimierung der Artefakt-Generierung.
Ein weiteres Problem ist die mangelnde Messbarkeit der Qualität der Bereinigten MRT-Bilder. Diese wurden in dieser Arbeit per Hand und damit auch aus subjektiver Wahrnehmung bewertet. Dazu mehr im Ausblick.

7.3 Ausblick

Für eine Fortführung der Arbeit müsste zunächst ein Qualitätskriterium der Artefaktentfernung geschaffen werden. Ein Vorschlag wäre es ein CNN zu nutzen, was darauf trainiert wird reale Artefakte zu erkennen. Ein solches CNN könnte ebenfalls für die Verbesserung der Generierung von künstlichen Artefakten genutzt werden. Es wird dabei darauf trainiert zwischen realen und simulierten Artefakten zu unterscheiden. Je schwerer dies dem CNN fällt desto besser ist die Simulation. Dies würde dem Diskriminator in einem Generative Adversial Network entsprechen.[RMC15]
In dem Paper "Retrospective correction of motion artifact affected structural MRI images using deep learning of simulated motionäus dem Jahr 2018 wird genau dieses Vorhaben

mit guten Ergebnissen umgesetzt. [Duf18] Dies bestätigt das Vorgehen in dieser Arbeit als auch die Relevanz und Aktualität des Themas. Abschließend lässt sich sagen dass Bewegungsartefakte im MRT ein reales Problem für die Medizin dar stellen, was für die Relevanz dieser Arbeit spricht.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[Aga18] Abien Fred Agarap. „Deep learning using rectified linear units (relu)“. In: *arXiv preprint arXiv:1803.08375* (2018). URL: https://arxiv.org/pdf/1803.08375.pdf (besucht am 20. 02. 2020).

[Ale17] Lindsay M Alexander u. a. „The Healthy Brain Network Biobank: An open resource for transdiagnostic research in pediatric mental health and learning disorders“. In: (2017). URL: https://www.biorxiv.org/content/10.1101/149369v1 (besucht am 02. 03. 2020).

[Alp10] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2010. URL: https://kkpatel7.files.wordpress.com/2015/04/alppaydin_machinelearning_2010.pdf (besucht am 20. 02. 2020).

[BH18] Tal Ben-Nun und Torsten Hoefler. „Demystifying parallel and distributed deep learning: An in-depth concurrency analysis“. In: *arXiv preprint arXiv:1802.09941* (2018). URL: https://arxiv.org/pdf/1802.09941.pdf (besucht am 25. 02. 2020).

[Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[Duf18] Ben A Duffy u. a. „Retrospective correction of motion artifact affected structural MRI images using deep learning of simulated motion“. In: (2018). URL: https://pdfs.semanticscholar.org/f489/dc8ef6b3d9770280c1279f62d5b30f025508.pdf?_ga=2.98059231.1505468273.1583258973-1769629937.1577551726 (besucht am 02. 03. 2020).

[EF07] Volkher Engelbrecht und Michael Forsting. *MRT und MRA des Kopfes: Indikationsstellung-Wahl der Untersuchungsparameter-Befundinterpretation*. Georg Thieme Verlag, 2007.

[Fan19] Linwei Fan u. a. „Brief review of image denoising techniques“. In: *Visual Computing for Industry, Biomedicine, and Art* 2.1 (2019), S. 7.

[GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[GM13] Samta Gupta und Susmita Ghosh Mazumdar. „Sobel edge detection algorithm". In: *International journal of computer science and management Research* 2.2 (2013), S. 1578–1583. URL: https://pdfs.semanticscholar.org/6bca/fdf33445585966ee6fb3371dd1ce15241a62.pdf (besucht am 02. 03. 2020).

[Gon16] Lovedeep Gondara. „Medical image denoising using convolutional denoising autoencoders". In: (2016), S. 241–246. URL: https://arxiv.org/pdf/1608.04667.pdf (besucht am 02. 03. 2020).

[Hay09] Simon Haykin. *Neural Networks and Learning Machines, 3/E*. Pearson Education India, 2009. URL: http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf (besucht am 20. 02. 2020).

[He16] Kaiming He u. a. „Deep residual learning for image recognition". In: (2016), S. 770–778. URL: https://arxiv.org/pdf/1512.03385.pdf (besucht am 02. 03. 2020).

[HZ10] Alain Hore und Djemel Ziou. „Image quality metrics: PSNR vs. SSIM". In: *2010 20th International Conference on Pattern Recognition*. IEEE. 2010, S. 2366–2369. URL: https://ieeexplore.ieee.org/abstract/document/5596999 (besucht am 02. 03. 2020).

[LB96] P Liang und NK Bose. *Neural network fundamentals with graphs, algorithms and applications*. 1996.

[Mar15] Martın Abadi u. a. „TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems". In: (2015). Software available from tensorflow.org. URL: http://tensorflow.org/ (besucht am 02. 03. 2020).

[Mit97] Tom M Mitchell u. a. *Machine learning*. McGraw-hill New York, 1997.

[MSY16] Xiao-Jiao Mao, Chunhua Shen und Yu-Bin Yang. „Image restoration using convolutional auto-encoders with symmetric skip connections". In: *arXiv preprint arXiv:1606.08921* (2016). URL: https://arxiv.org/pdf/1603.09056.pdf (besucht am 20. 02. 2020).

[Ng18] Andrew Ng. „CS229 Lecture notes". In: *CS229 Lecture notes* 1.1 (2018). URL: http://cs229.stanford.edu/notes/cs229-notes1.pdf (besucht am 27. 01. 2020).

[Oli06] Travis Oliphant. „NumPy: A guide to NumPy". In: (2006). URL: http://www.numpy.org/ (besucht am 02. 03. 2020).

[RMC15] Alec Radford, Luke Metz und Soumith Chintala. „Unsupervised representation learning with deep convolutional generative adversarial networks“. In: *arXiv preprint arXiv:1511.06434* (2015). URL: https://arxiv.org/pdf/1511.06434.pdf (besucht am 02. 03. 2020).

[Roj] Raul Rojas. *Neural Networks: A Systematic Introduction, 1996*. Springer-Verlag, Berlin. URL: https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf (besucht am 20. 01. 2020).

[Sch05] F Schick. „Grundlagen der Magnetresonanztomographie (MRT)“. In: *Der Radiologe* 45.1 (2005), S. 69–88. URL: https://link.springer.com/article/10.1007/s00117-004-1146-1 (besucht am 02. 03. 2020).

[See19] Abigail See. „Natural language processing with deep learning“. In: *Lecture Notes Stanford University School of Engineering* (2019). URL: https://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture07-fancy-rnn.pdf (besucht am 20. 02. 2020).

[SMS17] Jishnu Sadasivan, Subhadip Mukherjee und Chandra Sekhar Seelamantula. „Signal Denoising Using the Minimum-Probability-of-Error Criterion“. In: *arXiv preprint arXiv:1702.07869* (2017). URL: https://arxiv.org/abs/1702.07869 (besucht am 20. 02. 2020).

[Wal14] Stéfan van der Walt u. a. „scikit-image: image processing in Python“. In: *PeerJ* 2 (Juni 2014), e453. ISSN: 2167-8359. DOI: 10.7717/peerj.453. URL: https://doi.org/10.7717/peerj.453 (besucht am 02. 03. 2020).

[Yu14] Wei Yu u. a. „Visualizing and comparing convolutional neural networks“. In: *arXiv preprint arXiv:1412.6631* (2014). URL: https://arxiv.org/pdf/1412.6631.pdf (besucht am 20. 02. 2020).

[ZMH15] Maxim Zaitsev, Julian Maclaren und Michael Herbst. „Motion artifacts in MRI: a complex problem with many partial solutions“. In: *Journal of Magnetic Resonance Imaging* 42.4 (2015). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4517972/ (besucht am 20. 02. 2020).

Bildreferenzen

[Kar19] Andrej Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. 2019. URL: http://cs231n.github.io/convolutional-networks/ (besucht am 06. 02. 2020).

[Lau19] Laufer, Julian. *Image aesthetics quantification with a convolutional neural network (CNN)*. 2019. URL: https://jenslaufer.com/machine/learning/image-aesthetics-quantification-with-a-convolutional-neural-network.html (besucht am 20. 02. 2020).

[Per06] Enio Bueno Pereira. *Artificial Neuron models and its parts*. 2006. URL: https://www.researchgate.net/figure/Artificial-Neuron-models-and-its-parts-Source-Adapted-from-Haykin-1994_fig2_229036664 (besucht am 20. 01. 2020).

[Wik20] Wikipedia. *Convolutional neural network*. 2020. URL: https://en.wikipedia.org/wiki/Convolutional_neural_network (besucht am 06. 02. 2020).

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, den 03.03.2020

Georg Graf von Westerholt