



**Hochschule für Technik
und Wirtschaft Berlin**

University of Applied Sciences

Untersuchung von Image Colorization Methoden anhand
Convolutional Neuronal Networks

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft Berlin

Fachbereich IV: Informatik, Kommunikation und Wirtschaft

Studiengang Angewandte Informatik

1. Prüfer: Prof. Dr. Christin Schmidt
2. Prüfer: M.Sc. Patrick Baumann

Eingereicht von: Adrian Saiz Ferri
Immatrikulationsnummer: s0554249
Eingereicht am: XX.XX.2020

Abstract

Die vorliegende Arbeit beschäftigt sich mit der Einfärbung von Graustufenbilder durch Convolutional Neuronal Networks. Das Ziel ist es ein Model zu trainieren, dass selbstständig und ohne menschlichen Einfluss, aus einem Graustufenbild ein plausibles Farbbild erzeugen kann. Um das Model zu trainieren werden Farbbilder genommen und in Graustufenbilder umgewandelt. Die Graustufenbilder werden in das Netzwerk eingespeist und daraus werden die Farbkanäle erzeugt. Am ende wird das Graustufenbild mit den Farbkanälen konkateniert um das Farbbild zu generieren. Da Objekte auf einem Bild mehrere Farben haben können, werden mit Hilfe von Hyperparameter realistische anstatt "richtige" Farben bevorzugt.

Es werden verschiedene Netzwerk Architekturen exploriert und verglichen. Zunächst werden mehrere Experimente mit verschiedene Hyperparameter durchgeführt. Anschließend werden die Experimente ausgewertet und die Ergebnisse untersucht.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Zielsetzung	1
1.3	Vorgehensweise und Aufbau der Arbeit	1
2	Grundlagen	2
2.1	<i>Lab</i> -Farbraum	2
2.2	Neuronale Netze	2
2.2.1	Feedforward Neural Network	3
2.2.2	Fully-connected Neural Network	4
2.2.3	Aktivierungsfunktionen	6
2.2.4	Convolutional Neural Network	9
2.2.5	Kostenfunktionen	9
2.2.6	Backpropagation	9
2.2.7	Optimierungsalgorithmen	9
2.2.8	Max Pooling Layer	9
2.2.9	Transposed Convolution	9
2.3	Verwandte Arbeiten	9
3	Konzeption	10
3.1	Vorherige Arbeiten	10
3.2	Datensatz	10
3.3	Netzwerkarchitekturen	10
3.4	Framework	10
4	Implementierung	11
5	Evaluation	12
5.1	Vergleich der Modelle	12

6 Fazit	13
6.1 Zusammenfassung	13
6.2 Kritischer Rückblick	13
6.3 Ausblick	13
Abbildungsverzeichnis	I
Tabellenverzeichnis	II
Source Code Content	III
Glossar	IV
Literaturverzeichnis	V
Onlinereferenzen	VI
Bildreferenzen	VII
Anhang A	VIII
Eigenständigkeitserklärung	IX

Kapitel 1

Einleitung

1.1 Motivation

TODO

1.2 Zielsetzung

TODO

1.3 Vorgehensweise und Aufbau der Arbeit

TODO

Kapitel 2

Grundlagen

Dieses Kapitel verschafft einen Überblick über die benötigten theoretische Grundlagen, um die Methoden dieser Arbeit zu verstehen. Als erstes wird der “Lab-Farbraum” kurz erklärt. Als nächstes wird eine Einführung in Neuronale Netzwerke gegeben, anschließend werden einzelne Bestandteile und Varianten von Neuronalen Netzwerken erklärt. Abschließend wird einen Überblick über verwandte Arbeiten gegeben.

2.1 *Lab*-Farbraum

Der *Lab*-Farbraum (auch CIELAB-Farbraum genannt) ist ein Farbraum definiert bei der Internationale Beleuchtungskommission (CIE) in 1976. Farben werden mit drei Werte beschrieben. “*L*” (Lightness) definiert die Helligkeit. Die Werte liegen zwischen 0 und 100. “*a*” gibt die Farbart und Farbtintensität zwischen Grün und Rot und “*b*” gibt die Farbart und Farbtintensität zwischen Blau und Gelb. Die Werte für “*a*” und “*b*” liegen zwischen -128 und 127.

TODO: image

2.2 Neuronale Netze

Künstliche Neuronale Netze sind inspiriert durch das Menschliche Gehirn und werden für Künstliche Intelligenz und Maschinelles Lernen angewendet. Sie werden für überwachtes und unüberwachtes lernen verwendet. In der vorliegende Arbeit werden nur Methoden des überwachtes lernen angewendet. Bei überwachtes lernen sind die Datensätze gelabelt

sodass den Output von dem Neuronales Netz mit den richtigen Ergebnissen verglichen werden kann.

Neuronale Netze bestehen aus Neuronen oder auch “Units” genannt, die Schichtenweise in “Layers” (Schichten) angeordnet sind. Beginnend mit der Eingabeschicht (Input Layer) fließen Informationen über eine oder mehrere Zwischenschichten (Hidden Layer) bis hin zur Ausgabeschicht (Output Layer). Dabei ist der Output des einen Neurons der Input des nächsten. [Moe18]

2.2.1 Feedforward Neural Network

Das Ziel von einem Feedforward Neural Network ist die Annäherung an irgendeine Funktion f^* . Ein Feedforward Neural Network definiert eine Abbildung $y = f(x; \theta)$ wo x den Input ist und θ die lernbare Parameter sind (auch Weights genannt). [GBC16, S. 164-223]

Diese Netzwerkarchitektur heißt “feedforward” weil der Informationsfluss von der Input Layer über die Hidden Layers bis zur Output Layer in einer Richtung weitergereicht wird.

Feedforward Neural Networks werden als eine Kette von Funktionen repräsentiert. Als Beispiel, kann man die Funktionen $f^{(1)}, f^{(2)}, f^{(3)}$ in Form einer Kette verbinden um $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$ zu bekommen. Diese Kettenstrukturen sind die am häufigsten genutzte Struktur bei Neuronale Netzwerke. In diesem Fall, $f^{(1)}$ ist das erste Layer, $f^{(2)}$ das zweite und $f^{(3)}$ der Output Layer von diesem Netzwerk. Die Länge dieser Kette definiert die Tiefe von einem Netzwerk. Je tiefer ein Netzwerk ist desto mehr lernbare Parameter hat es und somit eine erhöhte Rechenleistung braucht um trainiert zu werden. In der Praxis werden die Netzwerke sehr tief, daher der Begriff Deep Learning.

Während dem Training werden die Weights von $f(x)$ verstellt, um $f^*(x)$ zu erhalten. Jedes Trainingsbeispiel x ist mit einem Label $y = f^*(x)$ versehen. Die Trainingsbeispiele legen genau fest, was der Output Layer generieren soll. Der Output Layer soll Werte generieren, die nah an y liegen. Das Verhalten von den Hidden Layers wird nicht durch die Trainingsbeispiele festgelegt, sondern der Lernalgorithmus soll definieren, wie diese Layers verwendet werden, um die beste Annäherung von $f^*(x)$ zu generieren. [GBC16, S. 164-223]

2.2.2 Fully-connected Neural Network

Fully-connected Neural Networks sind die am häufigsten vorkommende Art von Neuronale Netze. In dieser Netzwerkarchitektur sind alle Neuronen von einem Layer mit alle Neuronen von der vorherige und nächsten Layer verbunden. Neuronen in dem gleichen Layer sind aber nicht miteinander verbunden. [Fei17]

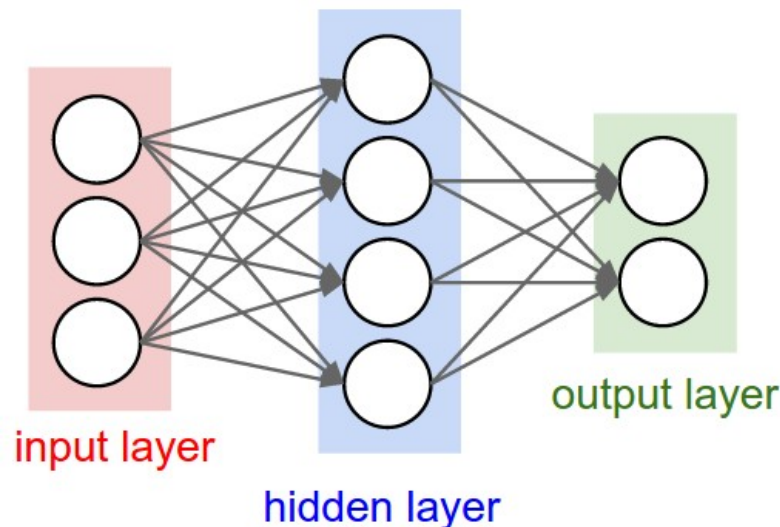


Abbildung 2.1: Fully-connected Neural Network mit 2 Layers (ein Hidden Layer mit 4 Neuronen) und ein Output Layer mit 2 Neuronen [Fei20a]

Eine der wichtigsten Gründe für die Anordnung von Neuronale Netze in Layers ist dass so eine Struktur anhand von Matrix Multiplikationen berechnet werden kann. Das obere Bild 2.1 stellt ein Netzwerk mit 3 Inputs x , eine Hidden Layer mit 4 Neuronen und eine Output Layer mit 2 Neuronen dar. Die Kreisen repräsentieren die Neuronen und einem Bias Wert b , die Pfeilen stellen die Weights w dar.

$$f(x) = w * x + b \quad (2.1)$$

Nach jeden Hidden Layer läuft den Output durch eine Aktivierungsfunktion σ die in 2.2.3 erklärt wird. Daraus wird die vorherige Formel um σ erweitert:

$$f(x) = \sigma(w * x + b) \quad (2.2)$$

Forward Pass

Den Forward Pass von einem Neuronalem Netz wird anhand von Matrizen Multiplikationen berechnet. Um das zu veranschaulichen wird es anhand eines Beispiels erklärt.

Ausgehend von einem Netzwerk mit 3 Inputs, eine Hidden Layer mit 2 Neuronen und einem Output Neuron, ergeben sich folgende Beispielwerte:

$$X = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \quad W = \begin{pmatrix} 10 & 20 \\ -20 & -40 \\ 20 & 0 \\ -40 & 0 \end{pmatrix} \quad W_{out} = \begin{pmatrix} 20 \\ 40 \\ -40 \end{pmatrix} \quad (2.3)$$

Die erste Spalte in der Input Matrix X und die erste Zeile in beide Gewichtsmatrizen W und W_{out} sind die Werte für den Bias. Diese Anordnung von den Bias Werte ermöglicht die Berechnung durch eine einzigen Matrix Multiplikation. Als Aktivierungsfunktion wird ReLU [NH10] verwendet:

$$\sigma(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \quad (2.4)$$

Im ersten Schritt durchläuft den Input durch das Hidden Layer $\sigma(X \times W)$:

$$\sigma \left(\begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 10 & 20 \\ -20 & -40 \\ 20 & 0 \\ -40 & 0 \end{pmatrix} \right) = \begin{pmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{pmatrix} \quad (2.5)$$

Im zweiten Schritt wird den Output von der vorherige Multiplikation mal die Gewichte von dem Output Layer multipliziert:

$$\sigma \left(\begin{pmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 20 \\ 40 \\ -40 \end{pmatrix} \right) = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \quad (2.6)$$

2.2.3 Aktivierungsfunktionen

Eine Aktivierungsfunktion definiert die Aktivierungsrate von einem Neuron. Es gibt verschiedene Aktivierungsfunktionen:

Sigmoid

Sigmoid ist eine nicht lineare Funktion $\sigma(x) = 1/(1+e^{-x})$ welche die Werte in einem Wertebereich von $[0, 1]$ bringt. Große negative Werte werden 0 und große positive Werte werden 1. Sigmoid hat verschiedene Nachteile, es neigt dazu den Gradienten zu verschwinden und die Outputs sind nicht Null zentriert.

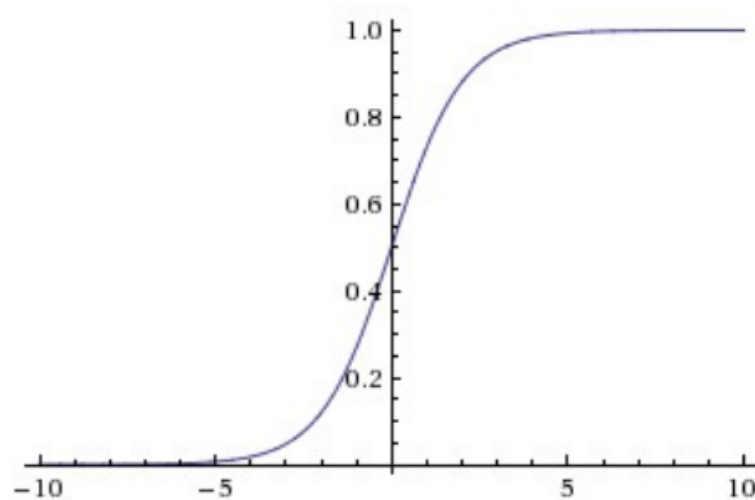


Abbildung 2.2: Sigmoid Aktivierungsfunktion [Fei20b]

Tanh

Die Tanh Aktivierungsfunktion bringt Werte in einem Wertebereich von $[-1, 1]$. Es ist eine skalierte Sigmoid (σ) Funktion, $\tanh(x) = 2\sigma(2x) - 1$. Die Nachteile von Tanh sind ähnlich zu Sigmoid aber der Output ist Null zentriert.

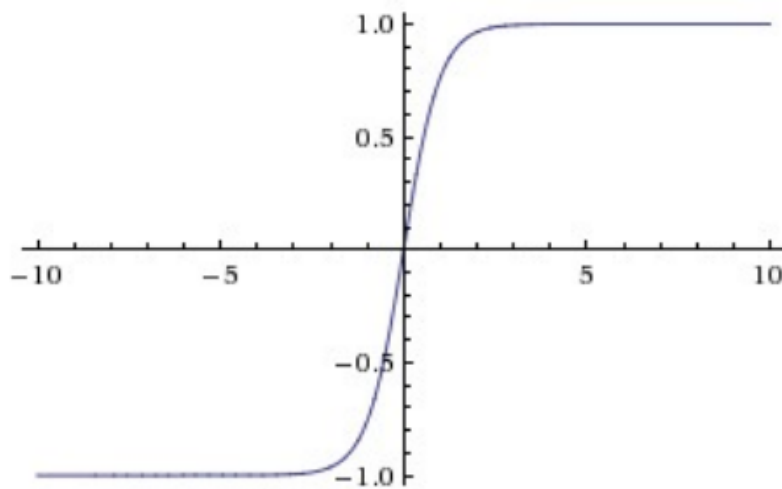


Abbildung 2.3: Tanh Aktivierungsfunktion [Fei20b]

ReLU

Die Rectified Linear Unit berechnet $f(x) = \max(0, x)$, alle negative Werte werden 0 und alle positive Werte bleiben unverändert. Diese Aktivierungsfunktion wurde für die Netzwerke in dieser Arbeit benutzt da es Vorteile gegenüber Sigmoid und Tanh zeigt. Einer der Vorteile ist dass die Mathematische Auswertung der Funktion unkompliziert ist. Außerdem beschleunigt es die Konvergenz von Stochastisches Gradientenabstiegsverfahren im Vergleich zu Sigmoid und Tanh.

Neuronen die ReLU als Aktivierungsfunktion verwenden, können während des Trainings “sterben”. Zum Beispiel, wenn der Gradient in einem Neuron zu groß ist, kann dieser zu einem update der Gewichte führen, wo das Neuron nie wieder aktiviert werden kann. Mit einer korrekter Einstellung der Lernrate kann das vermieden werden. [Fei17]

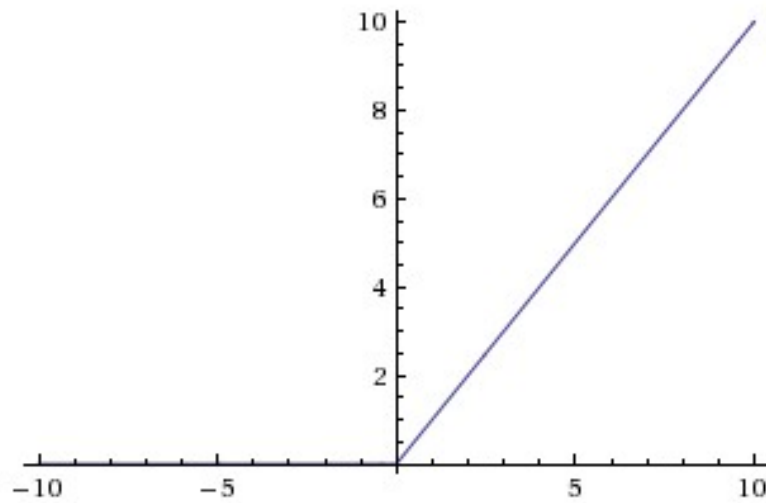


Abbildung 2.4: Rectified Linear Unit (ReLU) [Fei20b]

Leaky ReLU

Leaky ReLU ist eine Variante von ReLU, die versucht, das Problem mit den “sterbenden” Neuronen zu minimieren. Anstatt alle negative Werte in Null zu konvertieren, werden die Werte mit einer Konstante multipliziert. Die Funktion wird zu $f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$, wobei α eine kleine Konstante ist, zum Beispiel 0.001.

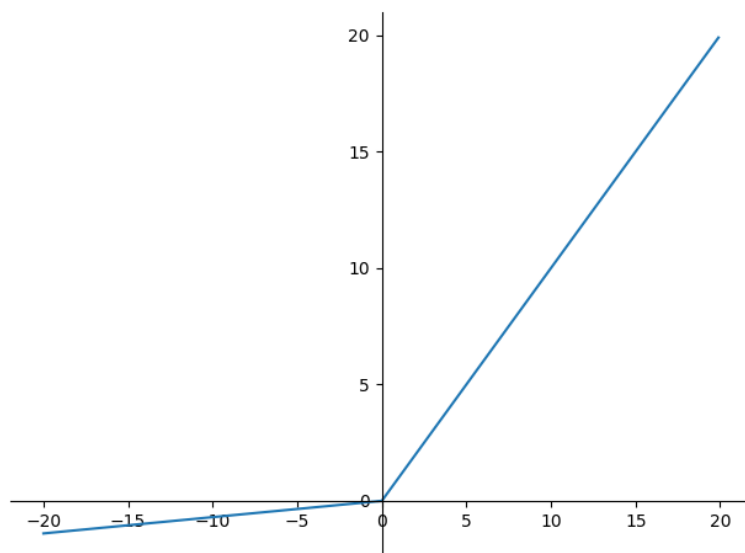


Abbildung 2.5: Leaky ReLU [ccs20]

2.2.4 Convolutional Neural Network

2.2.5 Kostenfunktionen

2.2.6 Backpropagation

2.2.7 Optimierungsalgorithmen

2.2.8 Max Pooling Layer

2.2.9 Transposed Convolution

2.3 Verwandte Arbeiten

TODO

Kapitel 3

Konzeption

TODO

3.1 Vorherige Arbeiten

TODO

3.2 Datensatz

TODO

3.3 Netzwerkarchitekturen

TODO

3.4 Framework

Kapitel 4

Implementierung

TODO

Kapitel 5

Evaluation

TODO

5.1 Vergleich der Modelle

TODO

Kapitel 6

Fazit

TODO

6.1 Zusammenfassung

TODO

6.2 Kritischer Rückblick

TODO (Reflexion und Bewertung der Zielsetzung gegenüber erreichtem Ergebnis)

6.3 Ausblick

TODO

Abbildungsverzeichnis

2.1	Fully-connected Neural Network mit 2 Layers (ein Hidden Layer mit 4 Neuronen) und ein Output Layer mit 2 Neuronen [Fei20a]	4
2.2	Sigmoid Aktivierungsfunktion [Fei20b]	6
2.3	Tanh Aktivierungsfunktion [Fei20b]	7
2.4	Rectified Linear Unit (ReLU) [Fei20b]	8
2.5	Leaky ReLU [ccs20]	8

Tabellenverzeichnis

Source Code Content

Glossar

CIE Internationale Beleuchtungskommission. 2

Layer Schicht. 2, 3

Literaturverzeichnis

- [GBC16] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. MIT Press, 2016.
- [NH10] Vinod Nair und Geoffrey E. Hinton. „Rectified Linear Units Improve Restricted Boltzmann Machines“. In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML’10. Haifa, Israel: Omnipress, 2010, S. 807–814. ISBN: 9781605589077. (Besucht am 27.07.2020).

Onlinereferenzen

- [Fei17] Serena Yeung Fei-Fei Li Justin Johnson. *Neural Networks 1*. 2017. URL: <https://cs231n.github.io/neural-networks-1/> (besucht am 12.07.2020).
- [Moe18] Julian Moeser. *Funktionsweise und Aufbau künstlicher neuronaler Netze*. 2018. URL: <https://jaai.de/kuenstliche-neuronale-netze-aufbau-funktion-291/> (besucht am 10.07.2020).

Bildreferenzen

- [ccs20] ccs96307. 2020. URL: <https://clay-atlas.com/us/blog/2020/02/03/machine-learning-english-note-relu-function/> (besucht am 31.07.2020).
- [Fei20a] Serena Yeung Fei-Fei Li Justin Johnson. 2020. URL: <https://cs231n.github.io/neural-networks-1/> (besucht am 10.07.2020).
- [Fei20b] Serena Yeung Fei-Fei Li Justin Johnson. 2020. URL: <https://cs231n.github.io/neural-networks-1/> (besucht am 12.07.2020).

Anhang A

TODO

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 XX.XX.2018

Adrian Saiz Ferri