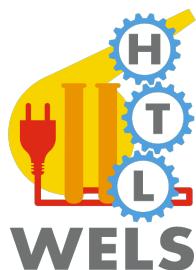


Pixorize

Bildkolorierung mittels Machine Learning

**Stefan Haas, Stefan Etzelstorfer,
Manuel Riedl**

Diplomarbeit Informationstechnologie
2019/2020



HTL Wels
Fischergasse 30
Wels 4600

Diplomanden

Stefan Etzelstorfer

Geburtsdatum: 02.10.2000

Adresse:

E-Mail: etzelstorfer.stefan@gmail.com

Stefan Haas

Geburtsdatum: 01.12.2000

Adresse: Hofwiesenstraße 7, Schlierbach an der Krems 4553

E-Mail: stefan.haas.privat@gmail.com

Manuel Riedl

Geburtsdatum: 29.11.2000

Adresse: Thal 1, Vöcklamarkt 4870

E-Mail: riedl.manuel.privat@gmail.com

Betreuer

Prof. DI Dr. Erich Gams

Prof. OstR Mag. Franz Reitinger

Danksagung

An dieser Stelle wollen wir uns bei allen Menschen in unserem Umfeld bedanken, welche uns bei dieser Diplomarbeit unterstützt haben.

Wir möchten uns besonders bei unseren Familien bedanken. Dafür, dass sie uns stets Zuversicht und seelische Unterstützung während schwierigen Phasen schenkten und uns somit Kraft gaben, um das Ziel konsequent weiterzuverfolgen.

Unser Hauptbetreuer Prof. DI Dr. Erich Gams leistete einen wichtigen Beitrag, indem er uns motivierte und uns dabei half, das große Ganze im Blick zu behalten. Des Weiteren unterstützte er uns mit professionellen Ratschlägen bei der Anmeldung für den Jugend Innovativ Wettbewerb.

Das gilt auch für unseren Zweitbetreuer, Prof. OstR DI Mag. Franz Reitinger, der uns mit seiner technischen Expertise unterstützte. Er stellte uns die notwendigen Ressourcen bereit, ohne welche das Ziel nicht erreicht hätte werden können.

Besonderer Dank gebührt auch Hubert Ramsauer MSc, vom Institut für Machine Learning an der JKU, denn er bot uns selbstlos seine fachliche Expertise an und versorgte uns dabei mit wertvollen Ratschlägen.

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbstständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Wels am _____ Name _____

Unterschrift _____

Wels am _____ Name _____

Unterschrift _____

Wels am _____ Name _____

Unterschrift _____

Gender Erklärung

Aus Gründen der besseren Lesbarkeit wird in dieser Diplomarbeit die Sprachform des generischen Maskulinums angewendet. Es wird an dieser Stelle darauf hingewiesen, dass die ausschließliche Verwendung der männlichen Form geschlechtsunabhängig verstanden werden soll.

Abstract

The goal of Pixorize is the implementation of a new machine learning approach for the colorization of black and white images.

In contrast to other approaches, which handle image colorization completely automatically, Pixorize integrates the user by manually setting color-pixels on the black and white image. This allows more accurate results to be achieved. As a result, different filiations of the same image may be possible.

Kurzfassung

Jeder besitzt alte Familien-Fotos, welche meist nur im Schwarz-Weiß-Format vorliegen. Oft kann man sich unter den Schwarz-Weiß-Fotos wenig vorstellen. Es fehlt der für den Menschen wichtigste Teil: Farbe.

Schon des Öfteren kam der Gedanke, wie beispielsweise die Hochzeits-Fotos, Porträts oder sonstige Schwarz-Weiß-Fotos der Familie in Farbe aussehen würden. Leider stellte sich die Kolorierung von Eigenhand in Programmen wie Photoshop, aber auch analog, am eigentlich Schwarz-Weiß-Foto, als eine große Herausforderung heraus. Auch die Alternative sich die Schwarz-Weiß-Fotos von professionellen Anbietern färben zu lassen, stellt keine Lösung dar: Hierbei muss man mit hohen Kosten rechnen. Des Weiteren ist nicht gewährleistet, dass das Ergebnis den eigenen Vorstellungen schlussendlich entspricht.

Die Diplomarbeit Pixorize beschäftigt sich mit einem neuen Ansatz, Schwarz-Weiß-Fotos zu kolorieren - sprich beispielsweise alte Familien-Fotos und Porträts von großen Persönlichkeiten der Geschichte, welche alle in Form von Schwarz-Weiß-Bildern vorliegen, in Farbe und somit in neuem Glanz erscheinen zu lassen. Dabei bestimmt der Benutzer selbst, wie das Ergebnis – also das zu entstehende Farbfoto - aussehen soll. Dies wird durch gezieltes Setzen von Farb-Punkten ermöglicht, welche aus ein und demselben Schwarz-Weiß-Bild verschiedene Resultate hervorbringen. So ist es beispielsweise dem Benutzer offen und frei, wie er die Kleidung, der im Bild vorhandenen Personen färbt oder welche Farbe die Landschaft im Hintergrund erhält.

Das Hauptaugenmerk von Pixorize: Der Gestaltungskraft des Benutzers freien Lauf zu lassen und demzufolge die Mannigfaltigkeit der entstehenden Resultat-Bilder zu gewährleisten.

Inhaltsverzeichnis

1 Projektmanagement	1
1.1 Projekthandbuch	1
1.1.1 Ziele	1
1.1.2 Nicht-Ziele	1
1.1.3 Projektmeilensteinplan	1
1.1.4 Projektlaufplan	2
1.1.5 Aufteilung der Arbeitspakete	3
1.2 Pflichtenheft	3
1.2.1 Funktionale Anforderungen	3
1.2.2 Nichtfunktionale Anforderungen	4
1.2.3 Verwendete Hardware	4
1.2.4 Use-Case-Diagramm	4
2 Datenaufbereitung - Stefan Etzelstorfer	5
2.1 Theoretische Grundlagen der Datenaufbereitung	5
2.1.1 Einleitung	5
2.1.2 Begriffe und deren Zusammenhänge	6
2.1.3 Die fünf Schritte der Datenaufbereitung	10
2.2 Theoretische Grundlagen der Bildverarbeitung	12
2.2.1 Einleitung	12
2.2.2 Abgrenzungen	12
2.2.3 Farbe	14
2.3 Implementierung eines Datapreparers	18
2.3.1 Einleitung	18
2.3.2 Sammeln der Daten	19
2.3.3 Bewerten der Daten	20
2.3.4 Bereinigen der Daten	22
2.3.5 Transformieren der Daten	24
2.3.6 Persistieren der Daten	36
3 Machine Learning - Stefan Haas	40
3.1 Definitionen	40
3.2 Machine Learning vs Traditional Programming	40
3.3 Supervised Learning	41
3.4 Künstliche neuronale Netze	42
3.5 Convolutional Neural Network (CNN)	44
3.5.1 Filter und Features	45
3.5.2 Pooling und Unpooling	46
3.5.3 Fully Convolutional Neural Network (FCNN)	47
3.6 Netzwerkarchitektur	48
3.7 Training und Evaluierung	53

4	Grafische Benutzer Oberfläche - Manuel Riedl	58
4.1	Einleitung	58
4.1.1	UI und UX	58
4.2	Konzept	59
4.2.1	Workflow	59
4.2.2	Technologienstack	62
4.3	Aufbau der Architektur	70
4.3.1	Kommunikation	70
4.3.2	Übersicht über die Files	71
4.4	Implementierung	74
4.4.1	Bild laden	75
4.4.2	Farbpunkt setzen	79
4.4.3	Bildposition ändern	83
4.4.4	Bild kolorieren	85
4.4.5	Bild anzeigen	87

1 Projektmanagement

1.1 Projekthandbuch

Folgender Abschnitt behandelt die in Pixorize definierten Ziele sowie die Beschreibung des Projektblaufes.

1.1.1 Ziele

- Implementierung eines neuartigen Machine Learning Algorithmus, welcher alte Schwarz-Weiß-Porträts automatisiert oder anhand von gesetzten Farbpunkten koloriert.
- Entwicklung einer Desktopanwendung, welche eine einfache und intuitive Bearbeitung der Bilder ermöglicht.

1.1.2 Nicht-Ziele

- Es soll keine reine Webanwendung entwickelt werden, da das künstliche neuronale Netz eine hohe Rechenleistung benötigt, welche in diesem Fall nur durch einen leistungsstarken Server garantiert werden kann, wodurch zusätzliche Kosten und Wartungsarbeiten anfallen würden.
- Die zu kolorierenden Schwarz-Weiß-Bilder sollen in der Domäne Portät- und Familienfotos liegen. Bilder außerhalb dieser Domäne - zum Beispiel Landschaftsfotos, Architekturfotos, etc - werden nicht behandelt.

1.1.3 Projektmeilensteinplan

Folgende Tabelle beinhaltet die festgelegten Meilensteine von Pixorize:

Datum	Meilenstein
01.10.2019	Recherche abgeschlossen
01.11.2019	Daten normalisiert
01.11.2019	KI-Prototyp entwickelt
01.11.2019	GUI-Prototyp entwickelt
20.12.2019	Funktionsprototyp 1 entwickelt
04.03.2020	Funktionsprototyp 2 entwickelt

Tabelle 1: Meilensteine

1.1.4 Projektablaufplan

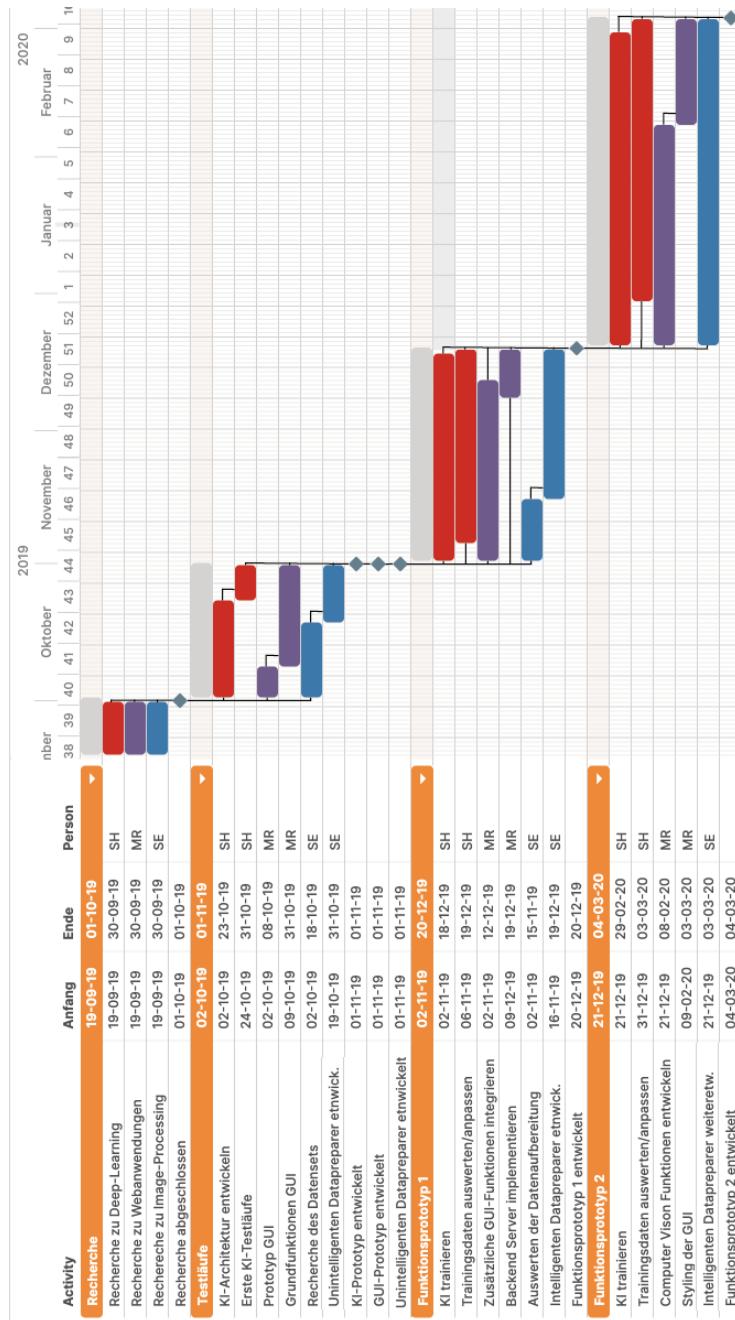


Abbildung 1: Projektablaufplan

1.1.5 Aufteilung der Arbeitspakete

Stefan Haas 275 Stunden			
Recherche zu Deep-Learning	19-09-19	30-09-19	
KI-Architektur entwickeln	02-10-19	23-10-19	
Erste KI-Testläufe	24-10-19	31-10-19	
KI trainieren	02-11-19	18-12-19	
Trainingsdaten auswerten/anpassen	06-11-19	19-12-19	
KI trainieren	21-12-19	29-02-20	
Trainingsdaten auswerten/anpassen	31-12-19	03-03-20	

Manuel Riedl 249 Stunden			
Recherche zu Webanwendungen	19-09-19	30-09-19	
Prototyp GUI	02-10-19	08-10-19	
Grundfunktionen GUI	09-10-19	31-10-19	
Zusätzliche GUI-Funktionen integrieren	16-11-19	19-12-19	
Backend Server implementieren	09-12-19	19-12-19	
Computer Vison Funktionen entwickeln	21-12-19	08-03-20	
Styling der GUI	09-02-20	03-03-20	

Stefan Etzelstorfer 250 Stunden			
Recherche zu Image-Processing	19-09-19	30-09-19	
Recherche des Datensets	02-10-19	18-10-19	
Unintelligenten Datapreparer entwick.	19-10-19	31-10-19	
Auswerten der Datenaufbereitung	02-11-19	15-11-19	
Intelligenten Datapreparer entwick.	16-11-19	19-12-19	
Intelligenten Datapreparer weiterentw.	21-12-19	03-03-20	

1.2 Pflichtenheft

Folgender Abschnitt enthält die an Pixorize gestellten funktionalen- sowie nicht-funktionalen Anforderungen. Des Weiteren erfolgt eine Beschreibung der Hardware Anforderungen.

1.2.1 Funktionale Anforderungen

- Der Benutzer soll durch Auswählen einer Farbe und Setzen eines Farbpunktes Kleidungsstücke beliebig einfärben können.
- Reguläre Bearbeitungsfunktionen, wie Zoomen oder Verschieben des Bildes, sollen dem Benutzer das Bearbeiten des Bildes erleichtern.
- Das kolorierte Bild soll durch Auswählen eines Nachbearbeitung-Algorithmus verbessert werden.

- Das kolorierte Bild soll - nach Kolorierung - lokal im Verzeichnissystem in den gängigen Grafikformaten exportiert werden können.
- Die Anwendung soll plattformunabhängig auf den gängigen Betriebssystemen - wie Windows, MacOs, und Linux - ausführbar sein.
- Die Kolorierung der Bilder soll lokal auf einem Endgerät erfolgen.

1.2.2 Nichtfunktionale Anforderungen

- Das Bild soll mithilfe des künstlichen neuronalen Netzwerks binnen Sekunden koloriert werden.

1.2.3 Verwendete Hardware

Bei Pixorize wurde folgende Hardware zum Trainieren des künstlichen neuronalen Netzes verwendet:

- Ubuntu Server
 - RAM: 16GB
 - Festplatte: 256GB SSD
 - Grafikkarte: NVIDIA GTX 1070

1.2.4 Use-Case-Diagramm

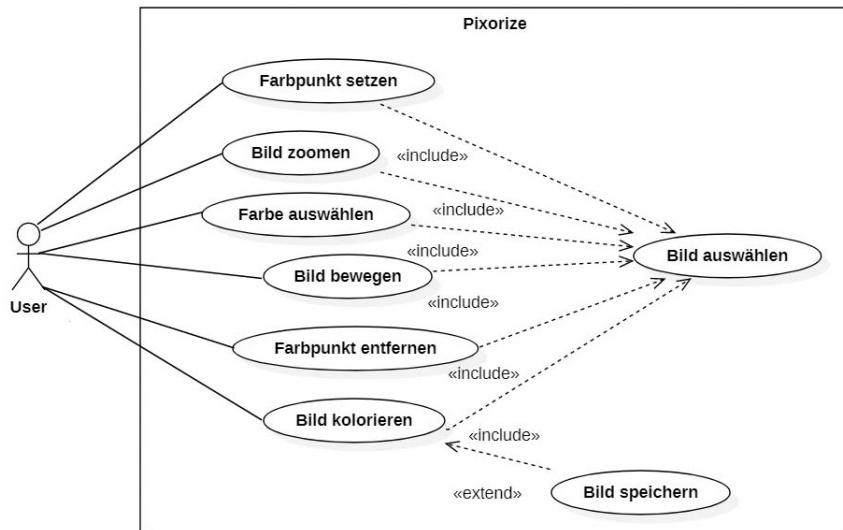


Abbildung 2: Use-Case-Diagramm

2 Datenaufbereitung - Stefan Etzelstorfer

Der folgende Teil beschreibt die für Pixorize notwendige Datenaufbereitung, um aus inkonsistenten, fehlerhaften Datensätzen (in Form von Bildern), einen Datensatz zu generieren, welcher für das Anlernen des künstlichen neuronalen Netzwerkes Anwendung findet.

Es erfolgt eine Unterteilung zwischen den theoretischen Grundlagen und der konkreten Implementierung.

2.1 Theoretische Grundlagen der Datenaufbereitung

2.1.1 Einleitung

Die Datenaufbereitung, oder auch Data Preparation im Englischen genannt[43], umfasst das Sammeln, Transformieren, Bereinigen und Bereitstellen von Rohdaten. Dabei sind die verwendeten Rohdaten in ihrer Ursprungsform oft inkonsistent, fehlerhaft und nicht standardisiert. Dies ist vor allem der Fall, wenn die Daten aus mehreren beziehungsweise verschiedenen Quellen stammen[17]. Im Fall von Pixorize handelt es sich bei den aufzubereitenden Rohdaten um Farbbilder, welche eine Reihe von Transformationen und Schritte der Aufbereitung durchlaufen. Die genannte Aufbereitung erfolgt dabei mittels der Bildverarbeitung, auf die im Abschnitt “Implementierung eines Datapreparers“ 18 näher eingegangen wird.

Das Ziel der Datenaufbereitung ist die Erzeugung eines qualitativ hochwertigen Datensatzes. Ein solcher ist vor allem im Bereich des Machine Learnings unentbehrlich. Denn die meisten Algorithmen erfordern eine sehr spezifische Formatierung der Daten, bevor sie nützliche Erkenntnisse beziehungsweise Informationen liefern können. Oft haben Datensätze Werte, die fehlen, die ungültig oder anderweitig für einen Algorithmus schwierig zu verarbeiten sind. Dabei könnten folgende Probleme auftreten[17]:

- wenn Daten fehlen, kann der Algorithmus sie nicht verwenden
- wenn Daten ungültig oder fehlerhaft sind, erzeugt der Algorithmus weniger genaue oder sogar irreführende Ergebnisse

All diese Gründe machen eine gute Datenaufbereitung für Machine Learning unentbehrlich.

2.1.2 Begriffe und deren Zusammenhänge

Um in die Thematik der Datenaufbereitung und deren konkrete Anwendung im Bereich des Machine Learnings einzusteigen, ist es notwendig, einige Begriffe und deren Zusammenhänge zu erläutern. Zur Visualisierung der Kausalitäten zwischen Zeichen, Daten, Information, Wissen und Machine Learning kommt die sogenannte Wissenspyramide zum Einsatz. Folgendes erläutert die in diesem Modell vorkommenden Schichten[27]:

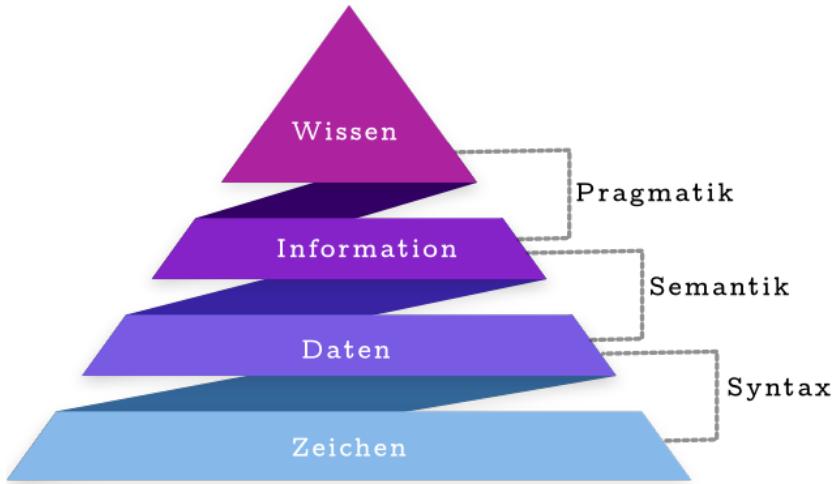


Abbildung 3: Wissenspyramide

Zeichen Auf der untersten Ebene der Wissenspyramide befinden sich die Zeichen. Sie stellen die Basis dar, auf welche die weiteren Schichten aufbauen. Sie können unterschieden werden in[27]:

- Numerische Zeichen
 - {0-9, “+“, “,” “.“}
- Alphanumerische Zeichen
 - {A-Z, a-z, 0-9}
- Sonderzeichen
 - {"*“, “/“, “\$“}

Daten Werden zu den Zeichen Syntax-Regeln hinzugefügt, so spricht man von Daten. Daten entstehen durch Sammeln und Messen von Sachverhalten. Sie können unterschieden werden in[27]:

- Unstrukturierte Daten

- *Unstrukturierte Daten sind Daten, die keine formale Struktur aufweisen*[20].
- Sind weitere Auswertungen zu den Daten notwendig, so müssen unstrukturierte Daten zuvor aufbereitet und strukturiert werden (Datenaufbereitung).

- Strukturierte Daten

- *Strukturierte Daten sind Daten, die in einem vorgegebenen, eindeutigen Format vorliegen*[20].
- Sie weisen eine gewisse Struktur auf und liegen in einer normalisierten Form vor.
- Sie stellen in gewisser Weise Informationen dar (siehe Information 2.1.2).
- Dort, wo Daten maschinell verarbeitet werden, speziell auch im Bereich des Machine Learnings, bringen strukturierte Daten große Vorteile - durch sie ist eine einfache Verarbeitung möglich.

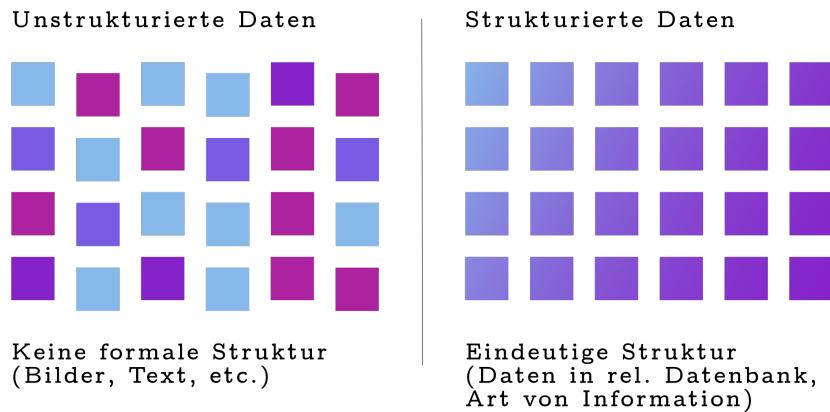


Abbildung 4: Unstrukturierte und strukturierte Daten

Information Wird den Daten Semantik - also ein Kontext - hinzugefügt, so spricht man von Informationen[27]. Informationen stellen Kenntnisse zu einem gewissen Sachverhalt oder Bezugssystem dar. Wie schon in “Daten“ 2.1.2 erwähnt, können aus unstrukturierten (Roh-)Daten strukturierte Daten - also Informationen - entstehen. Dabei durchlaufen die (Roh-)Daten eine gewisse Form der Aufbereitung und Verarbeitung. Diese Schritte werden unter dem Begriff “Datenaufbereitung“ zusammengefasst. Folgende Grafik soll nur grob die Entstehung von Information wiedergeben. Im Abschnitt “Implementierung eines Datapreparers“ 2.3 erfolgt eine genauere Erläuterung.

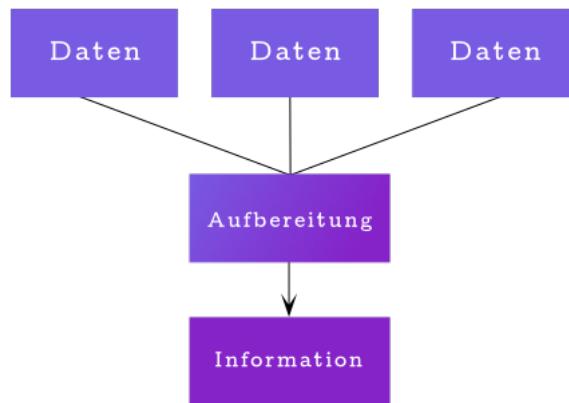


Abbildung 5: Erzeugung von Information

Wissen Durch die Verknüpfung von verschiedenen Informationen entsteht Wissen[27]. Wissen kann somit auch als strukturierte oder organisierte Information definiert und betrachtet werden. Erst mittels Wissen lassen sich Entscheidungen treffen und gewisse Handlungen eingeleitet werden. In der Wissenspyramide befindet sich Wissen an der obersten Ebene. Es bedient sich somit den unteren Ebenen (Zeichen, Daten und Information) und nur durch das Vorhandensein dieser kann es entstehen.

Machine Learning Nun, da alle Schichten der Wissenspyramide erläutert wurden, stellt sich die Frage, in welcher Ebene der Wissenspyramide Machine Learning Algorithmen agieren.

Wird die Wissenspyramide aus menschlicher Sicht betrachtet, so ist der Übergang der einzelnen Schichten fließend. Auch der Weg von Daten zu Wissen ist kein direkter Weg. Viel mehr ist es ein sich immer wieder wiederholender Ablauf: beginnend beim Sammeln der Daten, das Generieren der Informationen und das daraus folgende Aneignen des Wissens.

Betrachtet man diesen Prozess aus der Sicht eines Machine Learning Algorithmus, so kann ein Rekursionprozess betrachtet werden. Während Menschen die Hierarchie in der Wissenspyramide aufsteigen, so flachen Machine Learning Algorithmen sie ab. Egal, wie komplex und fortschrittlich ein Computersystem ist, in Wirklichkeit arbeitet ein solches immer nur mit Daten: Die Inputs eines Machine Learning Algorithmus sind Daten, die Verarbeitung dieser Inputs ist datenzentrisch orientiert und auch die Outputs sind aus der Sicht des Algorithmus erneut Daten[1].

Erst der Mensch kann die aus dem Machine Learning Algorithmus entstandenen Daten (Output) interpretieren und den jeweiligen Schichten der Wissenspyramide, durch Verknüpfen von weiteren Informationen und Daten, zuordnen[1]. Folgende Illustration soll diesen Zusammenhang veranschaulichen:

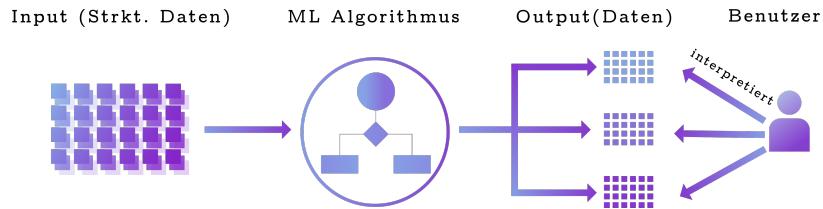


Abbildung 6: Einordnung von ML in der Wissenspyramide

Die genaue Definition zum Begriff “Machine Learning“ und die damit verwandten Begriffe “Artificial Intelligence“, etc. werden im Abschnitt “Machine Learning“ 2.1.2 behandelt.

2.1.3 Die fünf Schritte der Datenaufbereitung

Die Datenaufbereitung umfasst insgesamt fünf Schritte, welche hier erläutert werden[30]. Diese können je nach Branche oder Einsatzgebiet variieren. Die folgenden Schritte können im Bereich des Machine Learnings Anwendung finden. Auf deren konkrete Implementierung in Pixorize wird im Abschnitt “Implementierung der fünf Schritte der Datenaufbereitung“ 2.3.1 eingegangen (dieser Abschnitt soll lediglich die theoretischen Grundlagen vermitteln und ist dementsprechend sehr allgemein formuliert).



Abbildung 7: Schritte der Datenaufbereitung

1. Sammeln der Daten

Der wichtigste Schritt in der Datenaufbereitung stellt das Sammeln der Rohdaten dar[30]. Denn nur durch das Vorhandensein dieser können weitere Schritte zur Aufbereitung getätigten werden. Noch davor ist es notwendig, grob festzulegen, welche Art von Daten und für welchen Zweck die Daten benötigt werden.

Das eigentliche Sammeln umfasst das Selektieren der Daten von verschiedenen Datenquellen (Datenbanken, das “World-Wide-Web“, analoge Daten, etc.).

2. Bewerten der Daten

Nach der Datenerfassung folgt ein ebenso wichtiger Schritt: das Bewerten der Daten[30]. In diesem geht es vor allem um das Verstehen und das Auditieren/Bewerten der Daten. Wichtig hierbei ist eine genaue Überlegung, welche Ergebnisse mit den Daten beziehungsweise dem Datensatz erzielt werden sollen. Dementsprechend werden die dafür nötigen Schritte und Maßnahmen geplant, damit die Daten für einen bestimmten Kontext brauchbar werden.

Durchaus kann dieser als schwerster Schritt betrachtet werden, denn hierbei dürfen keine Fehler passieren. Werden dennoch Fehler gemacht, so kann es durchaus vorkommen, dass von vorne begonnen werden muss.

3. Bereinigen und Prüfen der Daten

Nun, da nach dem Schritt “Bewerten der Daten“ feststeht, welche Schritte getätigten werden müssen, damit die Daten für einen konkreten Verwendungszweck brauchbar werden, folgt die erste Maßnahme hierzu: das “Bereinigen und Prüfen der Daten“[30].

Folgende Aufgaben werden im Schritt “Bereinigen und Prüfen der Daten“ realisiert:

- Entfernung von irrelevanten, inkonsistenten Daten und statistischen „Ausreißern“
- So gut wie möglich: Ergänzung von fehlenden Werten

4. Transformieren und Anreichern der Daten

Es liegt nun ein fehlerfreier, konsistenter Datensatz vor, welcher einer Transformation in eine standardisierte Form bedarf[30]. Allgemein verfolgt dieser Schritt das Ziel, die Daten so zu verändern, damit mit ihnen ein gewisses Ziel oder Ergebnis, welches im Schritt „Bewerten der Daten“ zuvor festgelegt wurde, erreicht werden kann.

Des Weiteren erfolgt in diesem Schritt eine Anreicherung der Daten, das heißt die Daten werden um weitere Daten ergänzt. Es entstehen qualitativ hochwertigere Daten. Wie genau diese Anreicherung der Daten aussehen kann, hängt von dem gewünschten Ergebnis ab.

5. Persistieren der Daten

Der letzte Schritt der Datenaufbereitung beinhaltet die Persistierung der bewerteten, bereinigten, transformierten und angereicherten Daten[30]. Es wäre beispielsweise eine Persistierung in einer Datenbank, in Cloud-Diensten aber auch lokal auf einem Rechner denkbar. Wichtig ist hierbei nur, dass die Daten für die weitere Verwendung leicht und schnell zugänglich sind.

Durch die Durchführung dieser Schritte entsteht ein fehlerfreier, konsistenter, qualitativ hochwertiger Datensatz, auf dessen Basis die Entwicklung des Machine Learning Modells (Abschnitt „Machine Learning“ 2.1.2) erfolgt. Folgende Illustration veranschaulicht, welche Schritte nach der Aufbereitung der Daten folgen.



Abbildung 8: Schritte nach der Datenaufbereitung

2.2 Theoretische Grundlagen der Bildverarbeitung

Bei Pixorize handelt es sich bei den aufzubereitenden Daten um Farb-Bilder, welche in ein sehr spezifisches Format transformiert werden müssen. Dementsprechend erfolgt die Transformation der Daten, also Schritt vier der Datenaufbereitung, mittels der Bildverarbeitung.

Folgender Abschnitt beinhaltet die theoretischen Grundlagen der Bildverarbeitung, sodass im Abschnitt "Implementierung eines Datapreparers" 2.3.1 ein schneller Einstieg möglich ist.

2.2.1 Einleitung

Die Bildverarbeitung, im Englischen Image Processing genannt, umfasst Methoden zur Verarbeitung von digitalen Bildern, welche in der Informatik Signalen entsprechen. Prinzipiell wird ein Bild hierbei als zweidimensionales Signal oder zweidimensionale Matrix betrachtet, sodass übliche Methoden der Signalverarbeitung angewendet werden können[38].

Allgemein versucht die Bildverarbeitung, Bilder so zu transformieren, sodass diese für eine bestimmte Aufgabenstellung besser geeignet sind. Das Ergebnis der Bildverarbeitung stellt wiederum ein Bild dar, jedoch in veränderter Form.

2.2.2 Abgrenzungen

Folgender Abschnitt grenzt den Begriff der Bildverarbeitung zu verwandten Begrifflichkeiten ab:

Visual Computing "Visual Computing" kann als ein Überbegriff für die Teilbereiche Computergrafik, Computer Vision, und Bildverarbeitung betrachtet werden[59].

Alle drei Teilbereiche arbeiten mit Bildern und somit indirekt oder auch direkt mit Daten:

- Daten
 - Die Bildverarbeitung verarbeitet Bilder in Form von Daten (entspricht der Datenverarbeitung)
- Bilder
 - Bilder sind die visuelle Darstellung von numerischen (Bild-)Daten

Computergrafik (Bild) Eine Computergrafik visualisiert numerische Daten und erzeugt aus diesen ein Bild[40]. Die daraus entstandenen Grafiken lassen sich zweidimensional aber auch dreidimensional in der Ebene beschreiben.

Computer Vision Computer Vision ist die Extrahierung von Semantik aus realen Bildern[39]. Allgemein orientiert sich Computer Vision an den visuellen Fähigkeiten des Menschen und versucht, diese maschinell nachzubilden. Dazu gehören die räumliche Erfassung von Gegenständen und Objekten und unter anderem die Interpretation von Bewegungen. Computer Vision entstand aus einem Teilbereich der künstlichen Intelligenz, dessen Entwicklung auf zahlreiche visuelle Problemstellungen zurückzuführen ist. Eine der Kernaufgaben stellt hierbei die Erkennung von Mustern dar, also die Unterscheidung von Texturen, Erkennung von Text und beispielsweise auch menschlichen Gesichtern.

Zusammenhänge zwischen den Begriffen Folgende Grafik veranschaulicht, wie die einzelnen Teilbereiche miteinander agieren und zusammenarbeiten:

- Die **Datenverarbeitung** umfasst das Transformieren und Verändern von **Daten**
- Eine **Computergrafik** (Bild) visualisiert (Bild-)**Daten** und erzeugt daraus **Bilder**
- Von **Bildverarbeitung** spricht man, wenn Bilder (indirekt Daten) so verändert werden, dass sie für eine bestimmte Aufgabenstellung besser verwendet werden können
- **Computer Vision** interpretiert Bilder anhand von (Bild-)**Daten** und versucht anhand von dieser, Muster zu erkennen

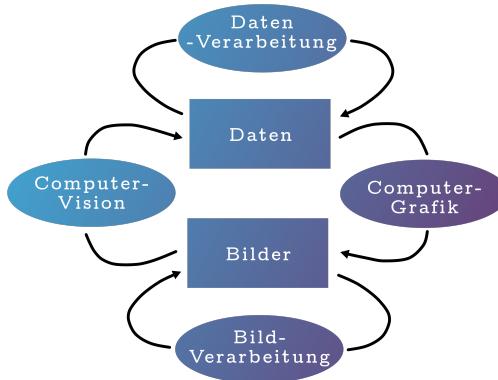


Abbildung 9: Zusammenhang: Daten und Bilder

2.2.3 Farbe

Farbe ist die Grundessenz einer jeden Computergrafik. Auch im alltäglichen Leben spielt sie eine wesentliche Rolle. Der Mensch assoziiert mit ihr Gefühle und Emotionen. Gerade deswegen ist es von Pixorize das Ziel, die fehlende Farbe von Schwarz-Weiß-Bildern durch gezieltes Kolorieren wiederherzustellen, ergo: Die mit dem Bild verbundenen Emotionen wieder erlebbar zu machen.

Farbe ist nicht gleich Farbe. Bei Pixorize wurden verschiedene Farbräume angewendet (warum diese angewendet wurden, wird im Abschnitt "Netzwerkarchitektur" 3.6 beschrieben). Folgendes soll einen einfachen Einstieg in die angewendete Kolorimetrie und den eingesetzten Farbräumen in Pixorize ermöglichen.

Kolorimetrie Die Kolorimetrie ist die Wissenschaft der technischen Beschreibung von Farbe. Ziel dieser ist die exakte Repräsentation von Farbe in einem numerischen Modell (Farbmodell)[45].

Der Mensch sieht ein Objekt in Form von elektromagnetischer Strahlung. Diese trifft auf das menschliche Auge und wird als Nervensignal an das Gehirn weitergeleitet. Diese elektromagnetischen Strahlen sind physikalisch messbar, wenn jedoch die subjektive menschliche Wahrnehmung zu tragen kommt (wenn das Nervensignal das Gehirn erreicht), so ist keine eindeutige Beschreibung von Farbe mehr möglich.

Lediglich der visuelle Stimulus des Menschen, also die subjektive Wahrnehmung, kann numerisch datiert werden (anhand von Experimenten mit einer großen Anzahl von Personen). Das Ergebnis dieser Experimente ist ein einheitlich numerisches Farbmodell.

Farbsymbolik Die Farbsymbolik erforscht die Bedeutung und Symbolik von Farbe[46].

Wie schon erwähnt, spielen Farben im Leben eines Menschen eine wesentliche Rolle. Die Verwendung und Bedeutung von ihr divergiert dabei zwischen den verschiedenen Kulturen. Aber auch jeder Mensch assoziiert mit Farbe andere Gefühle und Emotionen. Es bestehen jedoch auch allgemeine Assoziationen zu ihr: Die Farbe des Himmels ist blau (steht für das Weite), Rot entspricht der Farbe des Blutes (bedeutet Krieg und Tod, aber auch Lebenskraft und Liebe).

Farbmodelle und Farbräume Ein Farbraum, oder auch Farbmodell genannt, ist ein Teilbereich des gesamten Farbbereichs, welcher numerisch in einem Modell (Farbmodell) zusammengefasst wird[47]. In Pixorize wurden primär folgende Farbräume verwendet:

- RGB

- Der RGB-Farbraum stellt die enthaltenen Farben nach dem Prinzip der additiven Farbmischung dar[55]. Hierbei werden die drei Primärfarben: Rot, Grün und Blau miteinander vermischt. Das menschliche Auge interpretiert diese Mischung der drei Grundfarben als einen einzelnen Farbton.
- RGB wurde für die Farbwiedergabe auf Computer-Monitoren entwickelt.
- Jeder Farbpixel auf einem Monitor wird aus den drei Farbkanälen: Rot, Grün, Blau zusammengesetzt. Diese werden jeweils mit 8 Bit (0 bis 255) angegeben.
- Folgende Grafik illustriert den RGB-Farbraum in der dreidimensionalen Ebene und beschreibt, wie Farben darin numerisch dargestellt werden:

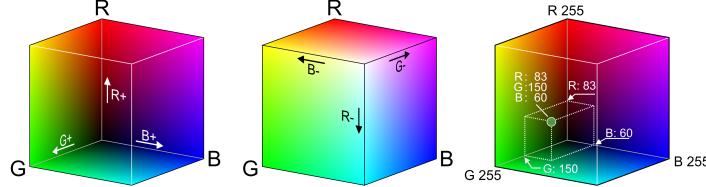


Abbildung 10: RGB-Farbraum[55]

- LAB (CIELAB)

- Der CIELAB-Farbraum, oft auch als “LAB“-Farbraum abgekürzt, beschreibt alle Farben als drei numerischen Werte (Farb-Kanäle)[51]:
 - * L: Die Helligkeits-Achse - Werte von 0 (Schwarz) bis +100 (Weiß)
 - * A: Die Grün-Magenta-Achse - Werte von -179 bis +100
 - * B: Die Gelb-Blau-Achse - Werte von -100 bis +150
- Somit erfolgt eine Trennung zwischen Farbe und Helligkeit: Der Helligkeits-Kanal (L) bestimmt nur die Helligkeit, die beiden anderen Kanälen liefern die dazugehörigen Farbinformationen.
- Im Gegensatz zu RGB können hierbei alle Farben, auch imaginäre, welche für den Menschen nicht sichtbar sind, dargestellt werden.
- Der LAB-Farbraum kann als Kugel im dreidimensionalen Raum betrachtet werden.

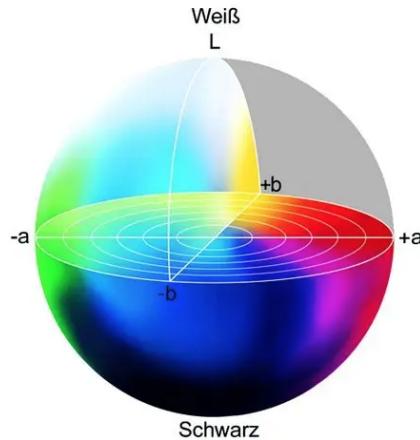


Abbildung 11: LAB-Farbraum[16]

- Graustufen (Greyscale)

- Bei Graustufen, Greyscale im Englischen genannt, handelt es sich um keinen Farbraum im eigentlichen Sinne. Viel mehr entsprechen Graustufen einem Farbraum (sei es RGB oder auch LAB) ohne das Vorhandensein von Farbinformationen. Lediglich die Lichtintensität bleibt dabei erhalten[49].
- Graustufen können folgende Werte annehmen:
 - * Schwarz - der dunkelste Farbton
 - * Weiß - der hellste Farbton
 - * Zwischengrautöne
- Jeder Farbraum kann in den Graustufen-Bereich konvertiert werden:
 - * LAB
 - . Hierbei ist keine Konvertierung notwendig, da die Lichtintensität schon in einem eigenen Kanal (L-Kanal) vorliegt. Dementsprechend gilt Folgendes:
 - . $I = L$
 - . I ... Lichtintensität (Graustufe)
 - . L ... L-Kanal (Lichtintensität) des LAB-Farbraums
 - * RGB
 - . Da im RGB-Farbraum die Lichtintensität in den drei Farbkanälen (RGB) enthalten ist, erfolgt folgende Berechnung, um diese zu extrahieren:
 - . $I = \frac{R+G+B}{3}$
 - . I... Lichtintensität (Graustufe)
 - . R, G, B ... Farbkanäle des RGB-Farbraums

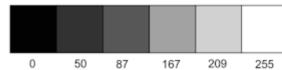


Abbildung 12: Graustufenkonvertierung im RGB-Farbraum

2.3 Implementierung eines Dataprepares

Nun, da die theoretische Basis erläutert wurde, folgt die konkrete Implementierung des Dataprepares.

2.3.1 Einleitung

Der Datapreparer repräsentiert die Gesamtheit aller Funktionalitäten, welche sich der Datenaufbereitung und die damit verbundene Bildverarbeitung widmen. Der genannte Datapreparer implementiert alle fünf Schritte der Datenaufbereitung:

1. Sammeln der Daten
2. Bewerten der Daten
3. Bereinigen der Daten
4. Transformieren der Daten
5. Persistieren der Daten

Zur Entwicklung wurde die Programmiersprache Python herangezogen, bei der auf eine Einführung verzichtet wird. Folgende Code-Stücke, auch als Code-Snippets bezeichnet, sind so dokumentiert, dass auch ohne Vorkenntnisse über die Programmiersprache Python ein einfacher Einstieg in diese gewährleistet ist. Als Resultat des Dataprepares entsteht ein konsistenter, normalisierter Datensatz, welcher anschließend an das künstlich neuronale Netzwerk weitergegeben wird.

2.3.2 Sammeln der Daten

Prinzipiell wäre es mit Pixorize möglich, alle Arten von Schwarz-Weiß-Bildern zu kolorieren (Landschafts-Bilder, Architektur-Bilder, etc). Aufgrund zeitlicher Vorgaben beschränkt sich Pixorize auf die Kolorierung von Portät- und Familien-Bilder, welche im Schwarz-Weiß-Format vorliegen. Daraus ergaben sich folgende Anforderungen an den Roh-Datensatz (an die Bilder):

- Bilder mit unterschiedlichen Personen
- Bilder, welche vorzugsweise im Hochformat vorliegen
- Bilder, welche eine hohe Auflösung aufweisen
- Bilder mit Personen im Vordergrund und minimalistischem Hintergrund (z.B. weiß)

Genau diesen Anforderungen entspricht “Flickr-Faces-HQ“ - ein qualitativ hochwertiger Datensatz, welcher aus insgesamt 70.000 PNG-Bildern mit einer Auflösung von 1024×1024 Pixeln besteht. Des Weiteren weisen die darin enthalten Bilder Unterschiede in Bezug auf Alter, ethnische Zugehörigkeit der Personen auf - ideal für das Trainieren des künstlich neuronalen Netzwerks.

Der Datensatz kann unter dem Link <https://github.com/NVlabs/ffhq-dataset> aufgerufen und heruntergeladen werden[34]. Hier ein paar Ausschnitte:



Abbildung 13: Ausschnitte des Datensatzes[34]

2.3.3 Bewerten der Daten

Nach dem Sammeln der Daten erfolgt das Bewerten dieser. Hierbei wird festgelegt, welche Maßnahmen getätigten werden müssen, um die Daten für einen bestimmten Kontext brauchbar zu machen, damit konkrete Ergebnisse mit dem Datensatz erbracht werden können.

Folgende Überlegungen wurden - unter Beachtung der vorher definierten Anforderungen - getätigten: Das künstlich neuronale Netzwerk muss sich die Fähigkeit aneignen, Bereiche in einem Schwarz-Weiß-Bild zu erkennen (beispielsweise ein Gesicht oder eine Jacke) und diese dementsprechend mit Farbe kolorieren. Zusätzlich muss der Benutzer so in die Kolorierung eingebunden werden, dass dieser mithilfe von gesetzten Farbpunkten direkt am Ergebnis (also dem zu entstehenden Farb-Bild) mitwirken kann. Dadurch kann bestimmt werden, welche Farbe beispielsweise eine im Bild enthaltene Jacke oder ein Pullover erhalten soll. Aufgrund dieser Überlegungen entstand folgender Entwurf des Datensatzes, mit grober Einbeziehung der Implementierung in Python:

Original Farb-Bilder

Damit das neuronale Netzwerk einen Vergleich hat, wie ein koloriertes Schwarz-Weiß-Bild aussehen soll, benötigt es aus Ausgang ein Farb-Bild. Aufgrund der speziellen Anforderungen - auf denen anschließend detailliert im Abschnitt "Entwurf" 3.6 eingegangen wird - müssen diese Bilder im LAB-Farbraum vorliegen. Dabei werden nur die Kanäle "A" und "B" verwendet, also die Farbkanäle. Die Lichtintensität (Kanal "A") wird hierbei entfernt. Diese Bilder werden in einem Array, also einer Datenstruktur zur Speicherung und Verwaltung von Daten, gehalten ("Original Farb-Bilder").

Manipulierte Graustufen-Bilder

Wie schon erwähnt, muss auch das direkte Mitwirken des Benutzers - durch Setzen von gezielten Farbpunkten - miteinbezogen werden. Dies erfolgt folgendermaßen: Die Ausgangs-Bilder, welche im LAB-Farbraum vorliegen, werden in den Graustufen-Bereich umgewandelt. Zusätzlich werden die Bilder mit zufällig gesetzten Farbpunkten versehen, welche die Inputs, also die gesetzten Farbpunkte des Benutzers, simulieren. So lernt das künstlich neuronale Netzwerk, wie es mit Schwarz-Weiß-Bildern und zusätzlich mit Benutzer-Inputs umzugehen hat. Wiederum werden diese Bilder in einem Array zusammengefasst ("Manipulierte Graustufen Bilder").

Die beiden Arrays werden in einer Liste gehalten, die sowohl die Ausgangs-Bilder und die dazugehörigen manipulierten Graustufen-Bilder beinhaltet. So ist jedem Ausgangsbild ("label") ein dazugehöriges manipuliertes Graustufen-Bild ("manipulated_image") zugeordnet - man spricht hier von einem einzelnen Tupel ("tupel"), also einem einzelnen Element aus dem Datensatz. Wie aus obigen zu entnehmen ist, wurden zusätzlich grundlegende Überlegungen getätigten, wie der Datensatz für das künstlich neuronale Netzwerk zugänglich gemacht werden kann. Dies erfolgt durch die schon erwähnte Liste, bestehend aus den Arrays.

Folgende Illustrationen veranschaulichen den groben Entwurf des Datensatzes:

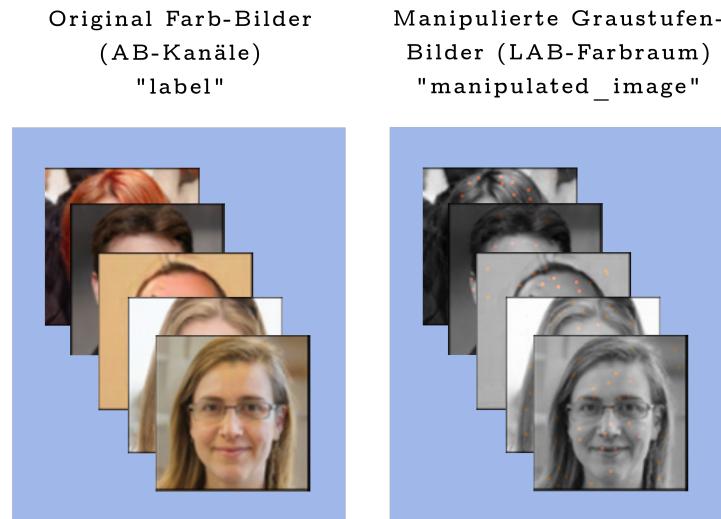


Abbildung 14: Aufbau des Datensatzes[34]

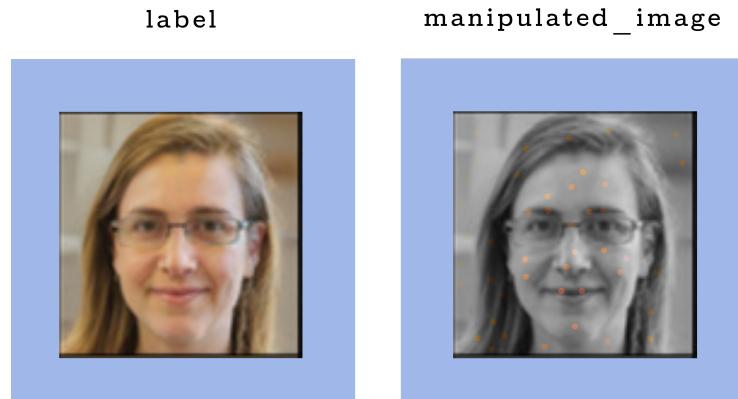


Abbildung 15: Aufbau eines Tupels[34]

2.3.4 Bereinigen der Daten

Nach Schritt zwei, dem Bewerten der Daten, steht fest, wie der Datensatz aufgebaut sein muss und welche Maßnahmen notwendig sind, um den Datensatz für einen bestimmten Kontext brauchbar zu machen. Nun folgt hierzu die erste Maßnahme: das Bereinigen und Prüfen der Daten.

Bei der Bewertung der Daten stellte sich heraus, dass der Datensatz korrupte Bild-Dateien und auch Nicht-Bild-Dateien (wie beispielsweise Text-Dateien, unsichtbare Dateien) enthielt. Für das Bereinigen beziehungsweise Entfernen dieser kam folgendes Python-Script zum Einsatz:

```
import os
from PIL import Image
```

Durch Importieren der Bibliothek “os“ können betriebssystem-interne Funktionen, wie beispielsweise das Erstellen oder Löschen von Dateien, getätigter werden[12].

Die Bibliothek “PIL“ bietet allgemeine Funktionen der Bildverarbeitung und hilft im Folgenden, korrupte Bild-Dateien zu erkennen[7].

```
# Path to the dataset
dataset_dir = "PFAD_ZUM_DATENSATZ"

# List of corrupted and non image files
bad_files = list()

# Generate list of all files (images and non image files)
list_of_files = list()
for (dirpath, dirnames, filenames) in os.walk(dataset_dir):
    list_of_files += [os.path.join(dirpath, file) for file in
                     filenames]
```

In der Variable “dataset_dir“ wird der Pfad zum Verzeichnis des Datensatzes angegeben. Des Weiteren wird eine Liste angelegt, welche alle korrupten Dateien und Nicht-Bild-Dateien enthält (“bad_files“). Anschließend wird das Datensatz-Verzeichnis rekursiv durchlaufen und alle absoluten Dateipfade (aller Dateien) in der Liste “list_of_files“ gehalten.

```

# Creates a list containing all corrupted image files
# - other files (not image files, like .txt etc.) are also
#   included
for filename in list_of_files:
    if filename.endswith('.png') or filename.endswith('.jpg') or
       filename.endswith('.jpeg'):
        try:
            # Opens the image file
            img = Image.open(filename)

            # Verifys, that the image is not corrupted
            img.verify()

            print("Korrekt Bild: ", filename)

        except (IOError, SyntaxError) as e:
            # Print the corrupted images
            print("Fehlerhaftes Bild: ", filename)

            # Add the corrupted file to the list
            bad_files.append(filename)
    else:
        # Print the non image file
        print("Keine Bilddatei: ", filename)

        # Add the non image file to the list
        bad_files.append(filename)

```

Nun wird die Liste "list_of_files" durchlaufen. Wenn eine Datei darin vorhanden ist, welche keine gängige Bild-Dateiendung wie "png", "jpg" oder "jpeg" enthält oder die Bild-Datei korrupt ist, wird diese in die Liste "bad_files" hinzugefügt.

```

# Remove the corrupted and non image files
for filename in bad_files:
    # Checks if this is a file - not a directory, then removes
    # the corrupted image / non image file
    if os.path.isfile(filename):
        print("Lösche: ", filename)
        os.remove(filename)

```

Anschließend wird die Liste "bad_files" durchlaufen und alle Nicht-Bild-Dateien, beziehungsweise korrupte Bild-Dateien aus dem Dateisystem entfernt. Durch dieses Vorgehen liegt nun ein - zumindest syntaktisch - fehlerfreier und konsistenter Datensatz vor, welcher an die folgenden Schritte zur Weiterverarbeitung gegeben werden kann.

2.3.5 Transformieren der Daten

Nun folgt der - im Falle von Pixorize - aufwendigste Schritt. Die Transformation und Anreicherung der Daten. Hierbei werden die Daten (Bilder) in eine standardisierte Form gebracht, welche im Schritt “Bewerten der Daten“ zuvor manifestiert wurde (in Form eines Entwurfs).

Bei Pixorize kann die Transformation der Daten in mehrere Teil-Prozesse gegliedert werden:

- Initialisierung
- Laden der Bilder
- Konvertierung zu Graustufen-Bilder
- Manipulation der Graustufen-Bilder

Zusammengefasst werden diese Teil-Prozesse in Form von Methoden in der Klasse “DataPreparation“. Hier der Aufbau der genannten Klasse mit den zu den Teilprozessen gehörigen Methoden:

```
class DataPreparation:  
  
    # Teil-Prozess: Initialisierung  
    def __init__(self, source_images_dir):  
        def create_directories(self):  
            def list_files(self):  
  
                # Teil-Prozess: Laden der Bilder  
                def get_color_images(self, list_of_files):  
  
                    # Teil-Prozess: Konvertierung zu Graustufen-Bilder  
                    def get_gray_images(self):  
  
                        # Teil-Prozess: Manipulation der Graustufen-Bilder  
                        def get_manipulated_gray_images(self):  
  
                            # Weitere Methoden des Schrittes "Persistieren der Daten"  
                            .  
                            .
```

Hinweis: Im Schritt “Persistieren der Daten“ 2.3.6 erfolgen die eigentlichen Methoden-Aufrufe von “get_color_images“, “get_gray_images“ und “get_manipulated_images“.

Initialisierung Der Teil-Prozess “Initialisierung“ stellt die notwendige Umgebung und Ressourcen für die nächsten Teil-Prozesse zur Verfügung. Folgendes wird in “Initialisierung“ umgesetzt:

Import aller benötigten Bibliotheken und Module:

```
import cv2
from PIL import Image
from skimage.util import random_noise
from torchvision import transforms
import random, os
import numpy as np
```

- cv2 und PIL[23]
 - Die schon erwähnte Bibliotheken “cv2“ und “PIL“ stellen grundlegende Funktionen der Bildverarbeitung zur Verfügung, mit welchen unter anderem Bilder geladen, konvertiert, transformiert und gespeichert werden können.
 - Da “PIL“ eine Vielzahl von Funktionen zur Verfügung stellt, die nicht alle benötigt werden, wird lediglich das Modul “Image“ eingebunden.
- skimage.util[28]
 - Auch bei “skimage.util“ handelt es sich um eine Bibliothek zur Bildverarbeitung - diese wird jedoch ausschließlich, durch Importieren von “random_noise“, zur Anwendung eines Gauß-Filters verwendet.
 - Die Erläuterung, warum ein Gauß-Filter angewendet wurde, erfolgt anschließend in der konkreten Implementierung.
- torchvision[26]
 - Durch den Import von “torchvision“ werden Funktionen zur Transformation an Bilder zur Verfügung gestellt.
 - Dabei behandelt “torchvision“ Bilder in Form von Tensoren, eine spezielle Form von Matrizen.
- random[25]
 - Mit “random“ können Zufallszahlen erzeugt werden.
- os[24]
 - Die schon erwähnte Bibliothek “os“ wird im Prozess “Initialisierung“ zur Erzeugung und zum Durchlaufen von Verzeichnissen genutzt.
- numpy[21]
 - “numpy“ findet Anwendung, um die Bilder in Matrizen (Tensoren) zu konvertieren.

Generierung und Initialisierung aller benötigten Ressourcen (“`__init__`“ Methode):

```
# __init__ function for initialisation
def __init__(self, source_images_dir):
    # Needed dataset directory
    self.source_images_dir = source_images_dir

    # Creates the dataset directory
    self.create_directories()

    # Get list of files
    self.list_of_files = self.list_files()

    # Needed arrays, containing images (color, gray, manipulated)
    self.color_images_array = []
    self.gray_images_array = []
    self.gray_manipulated_images_array = []

    # Parts
    self.min_part = 0
    self.max_part = 50
    self.range_value = 50
```

Die Methode “`__init__`“ stellt den Konstruktor der Klasse “DataPreparation“ dar. Der Methode wird hierbei der Pfad zu dem schon von fehlerhaften und korrupten Bildern befreiten Datensatz übergeben (“`source_images_dir`“). Falls dieses Verzeichnis noch nicht existiert, wird es mithilfe des Aufrufes der Methode “`create_directories`“ erzeugt:

```
# Creates the dataset directory
def create_directories(self):
    # Creates directories, if not exists
    folder = self.source_images_dir
    if not os.path.exists(folder):
        os.mkdir(os.path.join(folder))
```

Die in dem Modul “os“ enthaltene Methode “`path.exists`“ überprüft hierbei, ob das Verzeichnis (“`source_images_dir`“) schon existiert. Wenn dies nicht der Fall ist, wird mit der Methode “`mkdir`“ das gewünschte Verzeichnis erzeugt. Nun, da das Verzeichnis des Datensatzes vorhanden ist (oder schon vorhanden war), werden alle darin enthaltenen Pfade der Bild-Dateien in einer Liste aufgenommen.

Dies erfolgt in der Methode “list_files“:

```
# Get a list with all files - including non image-files - of the
→ directory "source_image_dir"
def list_files(self):
    list_of_files = list()
    for (dirpath, dirnames, filenames) in
    → os.walk(self.source_images_dir):
        list_of_files += [os.path.join(dirpath, file) for file in
    ← filenames]

    return list_of_files
```

Hierbei durchläuft die Methode “walk“, welche in der Bibliothek “os“ inkludiert ist, rekursiv das Verzeichnis des angegebenen Datensatzes (“source_image_dir“). Dabei werden alle Pfade der Bild-Dateien in die Liste “list_of_files“ aufgenommen. Die tatsächlichen Inhalte der Bild-Dateien werden im Teilprozess “Laden der Bilder“ in folgenden Arrays eingefügt, welche ebenfalls in der Methode “__init__“ deklariert werden:

```
# Arrays, containing image data
self.color_images_array = []
self.gray_images_array = []
self.gray_manipulated_images_array = []
```

Das Array “color_images_array“ enthält die gesamten Bild-Daten der Farb-Bilder. Analog dazu verwaltet “gray_images_array“ die gesamten Bild-Daten der in den Graustufenbereich konvertierten Farb-Bilder. “gray_manipulated_images_array“ hält parallel alle manipulierten Graustufen-Bilder mit den zufällig gesetzten Farb-Punkten. Aufgrund begrenzter Ressourcen war es notwendig, nur eine bestimmte Anzahl von Bildern zu laden (anstatt des Ladens aller Bilder zu einem Zeitpunkt). Dies wurde folgendermaßen realisiert:

```
# Parts
self.min_part = 0
self.max_part = 50
self.range_value = 50
```

Durch die Angabe von “min_part“ wird festgelegt, bei welcher unteren Grenze (im Regelfall 0) eine gewisse Anzahl an Bildern (“range_value“) bis zur maximalen Anzahl an Bildern (“max_part“) geladen werden soll. Die genaue Funktionsweise wird im letzten Schritt der Datenaufbereitung (“Persistieren der Daten“ 2.3.6) erläutert.

Laden der Bilder Nun, da alle Ressourcen (Verzeichnisse, Datenstrukturen) erzeugt und Initialisierung-Parameter festgelegt wurden, folgt der Teil-Prozess “Laden der Bilder“. In diesem werden die Bilder aus dem Verzeichnis “source_images.dir“ geladen. Den funktionellen Rahmen bietet die Methode “get_color_images“:

```
# Get an array including all color images (RGB-colorspace)
def get_color_images(self, list_of_files):
    for image_file in list_of_files:
        filex_extension =
            → os.path.splitext(image_file)[-1].lower()

        # Testing, if the file ends with ".jpg", ".jpeg" and
        → ".png"
        if filex_extension == ".jpg" or filex_extension ==
            → ".jpeg" or filex_extension == ".png":
            # Loads the image (as Array) and convert to RGB
            # (cv2 loads in BGR-colorspace - RGB is needed!)
            np_img = cv2.imread(image_file)
            if np_img is None:
                break
        color_image = cv2.cvtColor(np_img, cv2.COLOR_BGR2RGB)
```

Der Methode “get_color_images“ wird die Liste “list_of_files“ übergeben, in der sich die Pfade der Bild-Dateien befinden. Diese wird im Schritt “Persistieren der Daten“ erzeugt und enthält nur eine gewisse Anzahl (“range_value“) an Datei-Pfaden. “list_of_files“ wird anschließend durchlaufen, wobei das aktuelle Elemente temporär in der Variable “image_file“ gehalten wird. Der Vollständigkeit halber wird erneut - wie in “Bereinigen der Daten“ 2.3.4 - überprüft, ob die einzulesende Bild-Datei tatsächlich ein gängiges Bild-Datei-Format aufweist. Wenn dies der Fall ist, wird das Bild mittels der Funktion “imread“ im BGR-Farbraum eingelesen. Dabei liefert “imread“ eine numpy-Matrix “np_image“, bestehend aus den Bild-Daten. Da, wie schon erwähnt, das Bild im BGR-Farbraum eingelesen wird, jedoch der RGB-Farbraum benötigt wird, erfolgt eine dementsprechende Konvertierung mit der Methode “cvtColor“.

Hinweis: Wie jede Bibliothek hat auch “cv2“ ihre Besonderheiten. Eine davon ist, dass Bilder regulär im BGR-Farbraum eingelesen werden. Dieser Farbraum gleicht dem RGB-Farbraum, jedoch mit dem minimalen Unterschied, dass der Blau-Kanal und der Rot-Kanal vertauscht sind.

Anschließend erfolgt - ebenfalls in der Methode “get_color_images“ - eine Überprüfung, ob es sich bei dem eingelesenen Bild tatsächlich um ein korrektes und nicht korruptes Bild handelt. Dies wird folgendermaßen bewerkstelligt:

```
# Test if image is valid and not corrupted (np_img is empty)
if np_img is not None:
    # Insert image into the "color_images_array"
    self.color_images_array.append(color_image)
```

Wenn dies sichergestellt ist, wird das jeweilige Bild in dem Array “color_images_array“ aufgenommen und kann von weiteren Teil-Prozessen verwendet werden.

Hinweis: Alle Bilder werden im Schritt “Transformieren der Daten“ - der Einfachheit halber - im RGB-Farbraum verwaltet. Die Konvertierung in den LAB-Farbraum, den das künstlich neuronale Netzwerk vorschreibt, erfolgt im Schritt “Persistieren der Daten“ 2.3.6.

Konvertierung zu Graustufen-Bilder Im Teil-Prozess “Konvertierung zu Graustufen-Bilder“ werden - nomen est omen - die sich in dem Array “color_images_array“ befindenden Farb-Bilder zu Graustufen-Bilder umgewandelt und in dem Array “gray_images_array“ aufgenommen.

Zusätzlich wird durch Hinzufügen eines Gauß-Filters, also einem Filter zur Glättung beziehungsweise Weichzeichnung von Bildern, ein verschwommenes Graustufen-Bild erzeugt. Dadurch soll das künstlich neuronale Netzwerk sich die Fähigkeit aneignen, auch mit Bildern schlechterer Qualität umgehen zu können. Umgesetzt wird dies in der Methode “get_gray_images“:

```
# Get an array including all gray images (RGB-colorspace)
def get_gray_images(self):
    for np_img in self.color_images_array:
        # Converting the images into grayscale
        gray_image = cv2.cvtColor(np_img, cv2.COLOR_RGB2GRAY)
```

Das Array “color_images_array“ wird durchlaufen. Dabei wird jedes darin enthaltene Farb-Bild (“np_img“), welches in diesem Array in Form einer numpy-Matrix enthalten ist, mithilfe der Methode “cvtColor“ in ein Graustufen-Bild umgewandelt (“gray_image“). Der Gauß-Filter wird folgendermaßen angewendet:

```
# random gaussian blur and noise
if random.randrange(1) == 1:
    # Add salt-and-pepper noise to the image.
    noise_img = random_noise(gray_image, mode='s&p',
                             amount=0.3)
    gray_image = np.array(255 * noise_img, dtype='uint8')

if random.randrange(1) == 1:
    gray_image = cv2.blur(gray_image, (5, 5))
```

Anschließend erfolgt eine erneute Konvertierung in den RGB-Farbraum und das Graustufen-Bild wird in dem Array “gray_images_array“ aufgenommen.

```
# Converting the grayscale image into the RGB-colorspace
# and inserting it into "gray_images_array"
gray_image_rgb = cv2.cvtColor(gray_image,
                             cv2.COLOR_GRAY2RGB)
self.gray_images_array.append(gray_image_rgb)
```

Hinweis: Wie schon erwähnt, behandelt “cv2“ Bilder im BGR-Farbraum. Nach dem Aufruf der Funktion “cvtColor“ wird das Bild intern im BGR-Farbraum verwaltet. Deswegen erfolgt eine erneute Konvertierung in den RGB-Farbraum. Trotzdem handelt es sich immer noch um Graustufen-Bilder - lediglich das mathematische Modell zur Speicherung der in den Bildern enthaltene Farbe ändert sich.

Manipulation der Graustufen-Bilder Der Teil-Prozess “Manipulation der Graustufen-Bilder“ nutzt die Arrays “color_images_array“ und “gray_images_array“. Als Ergebnis dieses Teil-Prozesses entstehen Graustufen-Bilder, welche zufällig gesetzte Farb-Punkte enthalten. Wie schon erwähnt, erlernt so das künstlich neuronale Netzwerk, wie es mit Benutzer-Inputs umzugehen hat. Die Implementierung erfolgte in der Methode “get_manipulated_gray_images“:

```
# Get an array including all manipulated gray images (with
→ RGB-pixels)
def get_manipulated_gray_images(self):
    # Inserts random RGB-pixels into the grayscale image
    for manipulated_np_gray_image, np_color_image in
        → zip(self.gray_images_array, self.color_images_array):
            # Get the width and height of the image
            width, height = Image.fromarray(np_color_image).size
```

Die Arrays “gray_images_array“ und “color_images_array“ werden durchlaufen. Jedes Graustufen-Bild enthält ein dazugehöriges Farb-Bild - man spricht von einem Tupel. Von dem jeweiligen (aktuellen) Tupel wird die Breite (width) und Höhe (height) ausgelesen. Dies erfolgt mit dem in “PIL“ enthaltene Modul “Image“ und der Methode “size“, welche die Dimensionen eines Bildes liefert.

```
# Based on height and width, 10 - 20 random RGB-pixels
→ are inserted
amount_of_random_pixels = random.randint(10, 20)
```

Abschließend wird eine zufällige Anzahl von Farb-Punkten ermittelt (“amount_of_random_pixels“), dabei wird auf die Methode “randint“ von der Bibliothek “random“ zurückgegriffen. In jedem Bild werden 10 bis 20 Farb-Pixel eingefügt.

```
for a in range(0, amount_of_random_pixels):
    random_x_coordinate = random.randint(10, width - 10)
    random_y_coordinate = random.randint(10, height - 10)
```

Nun folgt das konkrete Einfügen der Farb-Punkte in das Graustufen-Bild: Für jeden Farb-Punkt wird eine zufällige Bild-Koordinate gebildet (“random_x_coordinate“ und die dazugehörige “random_y_coordinate“), die je nach Breite und Höhe des Bildes zufällig entsteht. Die Ermittlung der zufälligen Bild-Koordinaten wurde dabei so gewählt, dass kein Farb-Punkt in einem vom Rand ausgehenden Rahmen von zehn Pixel eingefügt wird.

```

# RGB-values from the original image
r, g, b =
Image.fromarray(np_color_image).
getpixel((random_x_coordinate, random_y_coordinate))

# Now, the RGB-values are read and inserted into the
↪ gray-image
cv2.circle(manipulated_np_gray_image,
↪ (random_x_coordinate, random_y_coordinate), 1,
↪ (r, g, b), -1)

# gray images containing color-pixels are inserted into
↪ the array "gray_manipulated_images_array"
self.gray_manipulated_images_array.
append(manipulated_np_gray_image)

```

Die RGB-Farb-Werte (“r“, “g“, “b“) werden von dem zum Graustufen-Bild dazugehörigen Farb-Bild ausgelesen. Dies erfolgt mit der Methode “getpixel“, die in dem Modul “Image“ von “PIL“ vorzufinden ist. Durch die Methode “circle“ von “cv2“ werden die Farb-Punkte in Form von ein Pixel großen Kreisen anschließend in das Graustufen-Bild gezeichnet (zu den zufälligen Farb-Punkt-Koordinaten). Das nun mit Farb-Punkten versehene Graustufen-Bild wird in das Array “gray_manipulated_images_array“ aufgenommen.

Zusätzlich wurde eine alternative Methode zur Manipulation der Graustufen-Bilder entwickelt, welche versucht, in den Bildern Gesichter zu erkennen. Werden Gesichter erkannt, wird in diesen eine höhere Anzahl von Farb-Punkten gesetzt. Dadurch soll das künstlich neuronale Netzwerk erkennen, dass gerade im Gesichtsbereich eine Vielzahl von verschiedenen Farben zusammenspielen. Zum Beispiel die Augenfarbe, die Hautfarbe oder auch die Haarfarbe.

Folgender Ausschnitt der Methode “get_manipulated_gray_images_v2“ gleicht “get_manipulated_gray_images“. Mit dem Unterschied, dass die Farb-Punkte vermehrt im Gesichtsbereich gesetzt werden:

```
# Get an array including all manipulated gray images (with
→   RGB-pixels)
def get_manipulated_gray_images_v2(self):
    # Inserts random RGB-pixels into the grayscale image
    for manipulated_np_gray_image, np_color_image in
        →   zip(self.gray_images_array, self.color_images_array):
```

Wie auch in “get_manipulated_gray_images“ werden die in den Arrays “gray_images_array“ und “color_images_array“ enthaltenen Bilder Element für Element manipuliert.

```
# Check, if there's a face in the image
faces = self.facecascade.
detectMultiScale(self.gray_images_array[index],
    →   scaleFactor=1.2, minNeighbors=5)

if len(faces) != 0:
    # If there's a face in the image - set more
    →   color-pixel in the face area
```

Die Bibliothek “cv2“ bietet Module beziehungsweise Methoden für die einfache Lösung von Computer-Vision-Problemstellungen. Hierbei wird auf den Klassifizierer “facecascade“ zurückgegriffen, welcher in Schwarz-Weiß-Bildern versucht, Gesichter zu erkennen. Mit “len(faces)“ wird überprüft, ob sich Gesichter in dem aktuellen Bild befinden. Wenn ja, werden in dem Gesichtsbereich verhäuft Farb-Punkte zu zufälligen Koordinaten, welchen sich in dem dementsprechenden Gesichtsbereich befinden, eingefügt:

```

for (x, y, w, h) in faces:
    # Set 6-10 random color-pixel in the face (between x and y)
    amount_of_random_pixels = randint(6, 10)

    for a in range(0, amount_of_random_pixels):
        random_x_coordinate = randint(x, x + w)
        random_y_coordinate = randint(y, y + h)

        # RGB-values from the original image
        r, g, b = Image.fromarray(np_color_image).
        getpixel((random_x_coordinate, random_y_coordinate))

        # Now, the RGB-values are read and inserted into the
        # gray-image
        cv2.circle(manipulated_np_gray_image,
        ↪ (random_x_coordinate, random_y_coordinate), 2, (r, g,
        ↪ b), -1)

# Same procedure as for "get_manipulated_gray_images_array"
.
.
.
```

“cv2“ ermittelt für jedes Gesicht die Koordinaten eines jenen Rechtecks, welches das Gesicht einschließt. Anschließend wird eine zufällige Anzahl von Farbpunkten “amount_of_random_pixels“ ermittelt und in das genannte Rechteck eingefügt. Die weitere Vorgehensweise gleicht “get_manipualted_gray_images“ und wird hier nicht weiter erläutert.

Hinweis: Die Anwendung solcher Computer-Vision-Funktionen ist sehr rechenintensiv. Aufgrund mangelnder Ressourcen kam die Methode “get_manipulated_gray_images_array_v2“ nicht zum Einsatz. Auch ohne diese konnten passable Ergebnisse beim Trainieren des künstlich neuronalen Netzwerks geliefert werden.

Der Schritt “Transformieren der Daten“ ist abgeschlossen. Das Ergebnis dieses Schrittes sind zwei Arrays, die dem festgelegten Entwurf im Abschnitt “Bewerten der Daten“ 2.3.3 entsprechen:

- color_images_array (Original Farb-Bilder)
 - Enthält Farb-Bilder
- gray_manipulated_images_array (Manipulierte Graustufen-Bilder)
 - Enthält die zu den Farb-Bildern dazugehörigen, mit Farb-Punkten versehene Graustufen-Bilder

Hinweis: Das Array “gray_images_array“ diente lediglich als Hilfs-Array und stellt somit kein konkretes Ergebnis des Schrittes “Transformieren der Daten“ dar.

2.3.6 Persistieren der Daten

Der abschließende Schritt der Datenaufbereitung umfasst das Persistieren der nun konsistenten, normalisierten und transformierten Daten.

Hierbei werden die im vorherigen Schritt entstandenen Arrays (und die darin enthaltenen Bilder) stückweise an das künstlich neuronale Netzwerk weitergegeben - man spricht von "Parts". Prinzipiell wäre es auch möglich, zu einem Zeitpunkt alle Farb- und manipulierten Graustufen-Bilder an das künstlich neuronale Netzwerk und weiterzugeben. Python hält Bild-Daten jedoch temporär im RAM, wodurch dieser bei einer hohen Anzahl von Bildern überlastet werden würde. Demnach muss - abgestimmt auf den jeweiligen Rechner/Server und dessen Hardware - die Anzahl der zu ladenden und liefernden Bilder angepasst werden (Methode "`__init__`", siehe "Transformieren der Daten" 2.3.5). Wie auch schon im vorherigen Schritt sind auch die Methoden bei "Persistieren der Daten" in der Klasse "DataPreparation" enthalten:

```
class DataPreparation:  
    # Methoden des Schrittes "Transformieren der Daten"  
    .  
    .  
  
    # Loads, transforms the images in parts and returns them  
    def load_images_in_parts(self):  
  
        # Converts the images into a specific format  
        def getAll(self):  
  
            # Method to release the RAM memory  
            def release(self, x):
```

Bilder stückweise laden und übergeben Die Methode “load_images_in_parts“ ruft die in “Transformieren der Daten“ beschriebenen Methoden “get_color_images“, “get_gray_images“ und “get_manipulated_gray_images“ auf. Dabei wird nur die durch die Methode “_init_“ vorgegebene Anzahl an Bildern verarbeitet und anschließend an das künstlich neuronale Netzwerk weitergegeben:

```
# Loads, transforms the images in parts and returns them
def load_images_in_parts(self):
    # Only handle n images per function call of
    # → "load_images_in_parts"
    ranged_list = self.list_of_files[self.min_part:self.max_part]
```

Es wird auf die Liste “list_of_files“ zugegriffen, welche die Dateipfade der Bild-Dateien enthält. Von dieser Liste - wird in Abhängigkeit von “min_part“ und “max_part“ - eine gewisse Anzahl (“max_part“ - “min_part“) von Bild-Dateipfaden in die Liste “ranged_list“ aufgenommen.

```
self.get_color_images(ranged_list)
self.get_gray_images()
self.get_manipulated_gray_images()
```

Anschließend wird diese an die Methode “get_color_images“ übergeben, wo eine Anzahl n von Farb-Bildern aus dem Datensatz geladen werden. Durch den Methoden-Aufruf von “get_gray_images“ und “get_manipulated_gray_images“ werden die dazugehörigen Graustufen- und manipulierte Graustufen-Bilder erzeugt.

```
# Change the parts for the next function call
# Save old minimum
old_min_part = self.min_part
old_max_part = self.max_part

self.min_part = (old_min_part + self.range_value + 1)
self.max_part = (old_max_part + self.range_value + 1)
```

Nun erfolgt eine erneute Initialisierung der Variablen “min_part“ und “max_part“. Dadurch wird gewährleistet, dass der Datensatz bei jedem Funktionsaufruf von “load_images_in_parts“ stückweise abgearbeitet wird. Das bedeutet: n Bilder werden geladen, n Bilder werden manipuliert und n Bilder werden an das künstlich neuronale Netzwerk weitergegeben.

Die Weitergabe an das künstlich neuronale Netzwerk erfolgt folgendermaßen:

```
# If the arrays are empty -> return an empty list
if not (self.color_images_array and self.gray_images_array
→ and self.gray_manipulated_images_array):
    return []
# If the arrays are NOT empty -> return the output of the
→ getAll()-method
else:
    return self.getAll()
```

Aus Gründen der Stabilität wird überprüft, ob die Arrays “color_images_array“ und “gray_manipulated_images_array“ leer sind. Wenn ja, wird eine leere Liste zurückgegeben. Wenn nein, erfolgt der Aufruf der Methode “getAll“, welche die Arrays in eine für das künstlich neuronale Netzwerk passende Form transformiert und deren Rückgabewert (Liste) anschließend an das künstlich neuronale Netzwerk weitergegeben wird.:

```
# Converts the images into a specific format
def getAll(self):
    list = []
```

Es wird eine Liste “list“ angelegt, welche anschließend n Tupel - bestehend aus “label“ und “manipulated.image“ - aufnimmt.

```
for i in range(0, len(self.gray_manipulated_images_array)):
    # mean std between [-1, 1]
    transform1 = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5], std=[0.5])
    ])

    transform2 = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5], std=[0.5, 0.5])
    ])
```

Es werden nun alle Bilder in den Arrays “color_images_array“ und “gray_manipulated_images_array“ Bild für Bild durchlaufen und transformiert. Die in der Bibliothek “pytorch“ enthaltenen Methode “transforms.Compose“ passt die Bild-Daten für die Aktivierungsfunktion des künstlich neuronalen Netzwerks an (für eine Erläuterung siehe “Machine Learning“ 3).

```

# Label - color image
lab_colour_img = cv2.cvtColor(self.color_images_array[i] ,
                             cv2.COLOR_RGB2LAB)

l, a, b = cv2.split(lab_colour_img)

merged_label = cv2.merge((a, b))
label = transform2(merged_label).float()

```

Nun folgt die Konvertierung eines jeden Farb-Bildes (enthalten in “color_images_array“) in den LAB-Farbraum. Anschließend wird das Bild in drei Kanäle gesplittet (“l“, “a“ und “b“). Dies erfolgt mit der Methode “split“ von “cv2“. Durch die Methode “merge“ werden die Farb-Kanäle “a“ und “b“ wieder zu einem Bild zusammengesetzt. Die Lichintensität (“l“) wird hierbei vernachlässigt. Abschließend werden die Bild-Daten für die Aktivierungsfunktion des künstlich neuronalen Netzwerks angepasst und transformiert (“transform2“). Daraus entsteht das schon mehrfach erwähnte “label“.

```

# Manipulated image
lab_manipulated_img =
    cv2.cvtColor(self.gray_manipulated_images_array[i] ,
    cv2.COLOR_RGB2LAB)

l, a, b = cv2.split(lab_manipulated_img)

merged_manipulated_image = cv2.merge((l, a, b))

manipulated_image =
    transform1(merged_manipulated_image).float()

```

In analoger Vorgehensweise entsteht so auch das zum Farb-Bild (“label“) dazugehörige, manipulierte Graustufen-Bild (“manipulated_images“). Mit dem Unterschied, dass auch die Lichtintensität - “l“ - in dem Bild aufgenommen wird.

```

# Tupel (manipulated_image + label)
tupel = [manipulated_image, label]
list.append(tupel)

```

Zusammengefasst wird “manipulated_image“ und “label“ in einem Tupel (“tupel“), welches in der oben angelegten Liste “list“ eingefügt wird.

```
return list
```

Das künstlich neuronale Netzwerk wird nun - durch Aufrufen der Methode “load_images_in_parts“ - kontinuierlich mit Bild-Daten in Form einer Liste versorgt. Der Schritt “Persistieren der Daten“ und die Datenaufbereitung im Allgemeinen ist nun abgeschlossen. Einem erfolgreichen Trainieren des künstlich neuronalen Netzwerks steht nichts mehr im Weg.

3 Machine Learning - Stefan Haas

3.1 Definitionen

Artificial Intelligence[37] ist ein Überbegriff und beinhaltet jegliche Art von Algorithmus, welcher versucht Intelligenz zu erzeugen, aber auch zu imitieren. Dazu zählen auch ganz simple IF-ELSE-Regeln und Entscheidungsbäume. Dieses Gebiet inkludiert das Maschinelle Lernen, als auch das Deep Learning.

Machine Learning[52] ist ein Subgebiet der Künstlichen Intelligenz, welches statistische Methoden verwendet, um spezifische Aufgaben zu lösen. Beim Machine Learning wird versucht, einem Algorithmus das Erkennen eines Musters beizubringen und mit dem Gelernten versuchen Prädiktionen zu treffen. Dieses Gebiet inkludiert auch das Deep Learning.

Deep Learning[44] ist ein Subgebiet des Machine Learnings und bezieht sich ganz speziell nur auf mehrschichtige Neuronale Netze. Dieses Gebiet ist am modernsten und wird aktuell am häufigsten verwendet – zum Beispiel zur Objekterkennung in Bildern. Jedoch ist Deep Learning an sich kein neuer Begriff. Schon im 20. Jahrhundert stellte man die These auf, dass mehrschichtig verknüpfte künstliche Neuronen ein effektives Lernmodell darstellen müsste. Jedoch war die Rechenleistung zu diesem Zeitpunkt, als auch die geringe Menge an Daten, nicht ausreichend, damit Deep Learning zu dieser Zeit erfolgreich hätte sein können. 2012, ein Jahr in welchem man weit bessere Rechenleistungen hatte als zuvor und außerdem über viel mehr Daten verfügte, war die Neubelebung des Deep Learnings. Diese alte These wurde einfach neu aufgegriffen und war durch die technischen Verbesserungen, welche im Lauf der Jahre passierten, in der Lage enorm erfolgreiche Ergebnisse zu erzielen.

3.2 Machine Learning vs Traditional Programming

Herkömmliche Computer wurden erfunden, um zu kalkulieren. Demnach ist ein Computer hervorragend in berechenbaren Aufgaben, wie zum Beispiel das Berechnen von Pi. Ein Mensch würde bei derselben Aufgabe jedoch scheitern, oder zumindest viel mehr Zeit benötigen. Würde man jedoch daraus schließen, dass ein Computer bei jeder Aufgabe besser als ein Mensch sei, so wäre das falsch. Denn es gibt Aufgaben, welche menschliche Kompetenzen erfordern, wie zum Beispiel das Erkennen einer Katze in einem Bild. Schließlich können sich Katzen massiv voneinander unterscheiden und um wirklich effektiv zu entscheiden, ob sich eine Katze in einem Bild befindet, müssten endlos viele Regeln programmiert werden, welche alle Fälle abdecken müssten. Ganz klar, ist hier der Mensch im Vorteil im Gegensatz zum herkömmlichen Computer. Aus diesem Grund stellt sich schon lange die Frage, ob eine Maschine einem Menschen in seinem Verhalten und in seiner Entscheidungsfindung ähneln kann. Im Laufe der Zeit entwickelte man neue Algorithmen, welche auf statistischen Methoden basieren, welche tatsächlich gute Ergebnisse erzielten. Diese Algorithmen sind

unter dem Namen Machine Learning bekannt. Der Schlüssel zum Erfolg: Daten. Daten waren der ausschlaggebende Grund dafür, dass statistische Methoden überhaupt angewandt werden konnten und deshalb stellen Daten für Machine Learning einen essenziellen Grundpfiler dar.

Bei der **traditionellen Programmierung** werden für ein Problem, oder eine Aufgabe, Regeln, in Form von einer spezifischen Folge von Kontrollstrukturen, definiert. Diese Methode ist sehr effektiv und kontrollierbar bei einfachen Aufgaben, ist aber sehr aufwändig für komplexe und dynamische Aufgaben.

Jedoch eignet sich **Machine Learning** hervorragend für solch komplexe und dynamische Aufgaben, wie das Erkennen einer Katze. Anstatt Regeln zu definieren, wird das Programm mit Daten gefüttert. Auf Grund dieser Daten entsteht ein Algorithmus.

3.3 Supervised Learning

Beim Machine Learning unterscheidet man zwischen zwei grundlegenden Lernmethoden, dem Supervised Learning[56] und dem Unsupervised Learning[58]. Beim Supervised Learning erhält ein neuronales Netzwerk zusätzlich zu den Input Daten auch die erwarteten Output Daten. Das kann man sich so vorstellen, als hätte der Algorithmus einen Trainer, welcher ihm Fehler anzeigt und die richtige Lösung verrät. Durch diese Art von Feedback wird dem neuronalen Netz ermöglicht schnell zu lernen. Speziell bei sogenannten Klassifizierungs- und Regressions-Algorithmen kommt diese Lernmethode sehr häufig vor. Will man zum Beispiel in einem Bild ein Objekt identifizieren, so wird einer der beiden Algorithmen verwendet. Unterscheidet man lediglich zwischen zwei verschiedenen Klassen, zum Beispiel zwischen Hund und Katze, so verwendet man einen Klassifizierungs-Algorithmus. Falls man jedoch zwischen mehreren verschiedenen Klassen unterscheidet, so verwendet man einen Regressions-Algorithmus.

Bei Pixorize ist der Output des neuronalen Netzes ein Bild mit Pixelwerten zwischen 0 und 255. Demnach existieren hierbei 256 verschiedene Klassen pro Farb-Channel und somit wird bei Pixorize ein Regressions-Algorithmus verwendet. Wobei man erwähnen muss, dass die Häufigkeitsverteilung dieser Klassen nicht gleichmäßig verteilt ist, da der Datensatz hauptsächlich Hautfarben und Beige-Töne beinhaltet. Demnach ist es wahrscheinlicher, dass ein Pixel hautfarbig koloriert wird, anstatt dass es blau koloriert wird.

3.4 Künstliche neuronale Netze

Das künstliche Neuron Von der Neurobiologie inspiriert, ist das künstliche Neuron[50] eine mathematische und algorithmische Darstellung des natürlichen Neurons. Ein einfaches Neuron hat die Eigenschaft, dass es Synapsen besitzt, welche als Verbindungen zwischen anderen Neuronen dienen. Über diese Synapsen werden Signale gefeuert, welche ein anderes Neuron stimulieren können. Bei einer hohen Stimulation feuert das Neuron und gibt somit ein eigenes Signal weiter. Genau dies wird algorithmisch beim künstlichen Neuron umgesetzt. Bei einem künstlichen neuronalen Netz wird der Begriff Synapse, durch die Begriffe Input/Output ersetzt. Als Input versteht man Daten, welche in das künstliche Neuron einfließen und als Output werden die Daten, welche das künstliche Neuron generiert, bezeichnet. Der sogenannte Feed-Forward-Prozess beschreibt den

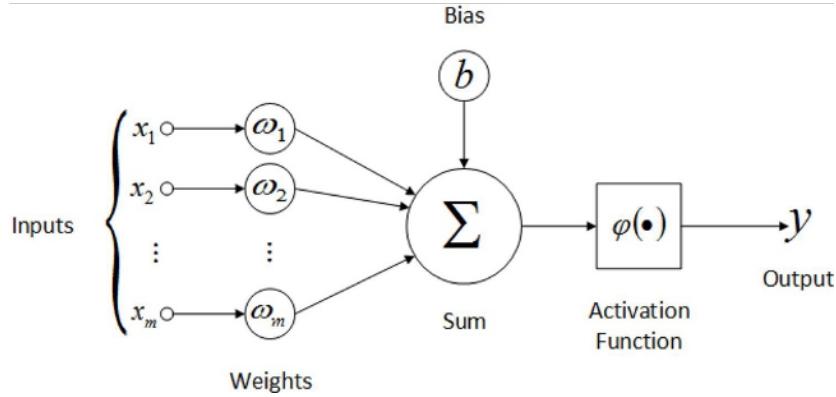


Abbildung 16: Grafische Darstellung eines künstlichen Neurons[15]

Prozess des Durchlaufes eines künstlichen Neurons. Jeder Input wird mit einer jeweiligen Gewichtung[48] multipliziert; diese Inputs werden summiert und anschließend an eine sogenannte Aktivierungsfunktion[36], welche entscheidet, ob das Neuron feuert, übergeben. Wenn man davon spricht, dass ein künstliches Neuron feuert, dann meint man, dass der Output einen hohen Wert besitzt.

Wenn man davon spricht, dass ein künstliches Neuron lernt, so meint man eigentlich lediglich, dass sich die Gewichte ändern. Mit der sogenannten Back-propagation werden die Gewichte aktualisiert. Diese Aktualisierung ist stark von der Fehlerabweichung des Outputs abhängig. Falls zum Beispiel der Output sehr stark vom erwarteten Ergebnis, der sogenannten Ground Truth, abweicht, so werden die Gewichte radikal geändert.

Aktivierungsfunktion Wie bereits erwähnt werden Aktivierungsfunktionen genutzt, um für das Neuron zu entscheiden, ob die Summe der gewichteten Inputs groß genug ist, um das Neuron zu aktivieren – daher stammt auch der Name. Ohne Aktivierungsfunktionen wäre der Output eines Neurons die Summe

aller gewichteten Inputs und das würde eine gewisse Linearität darstellen, wodurch ein künstliches neuronales Netz nicht in der Lage wäre, komplexe Zusammenhänge zu lernen. Durch das verwenden einer nicht-linearen Funktion wird das Lernen komplexer Zusammenhänge ermöglicht. Solche Funktionen müssen folgende Eigenschaften aufweisen:

- Streng monotone Steigung
- Werte zwischen -1 und 1, oder 0 und 1
- Wendepunkte bei $x = 0$

Der **Hyperbolische Tangens (Tanh)**[57] besitzt eine auf null zentrierte Ausgabe, da der Wertebereich hier zwischen -1 und 1 liegt. Diese Funktion wird häufig bei Klassifizierungsproblemen verwendet und bietet den Vorteil gegenüber Funktionen mit Werten zwischen 0 und 1, dass die Gewichte aussagekräftiger sind, da hier mehr Werte möglich sind und die Wertedistribution somit breiter ist.

Die **Rektifizierte Lineare Einheit (ReLU)**[54] ist eine sehr häufig eingesetzte Aktivierungsfunktion, welche zwar einen Wertebereich zwischen 0 und 1 hat, jedoch sehr effizient arbeitet. Hier ist die Ausgabe nicht zentriert und somit eignet sich diese Funktion nicht so sehr für ein Klassifizierungsproblem.

Mehrschichtige künstliche neuronale Netze Einzelne künstliche Neuronen werden lediglich bei simplen Anwendungen verwendet, wie das Annähern einer bestimmten Funktion. Will man jedoch komplexere Anwendungen erschaffen, so müssen mehrere Neuronen verwendet werden. Dabei werden Schichten gebildet, welche eine spezielle Anzahl an Neuronen besitzen, die mit allen Neuronen der darauffolgenden Schicht verbunden sind. Man unterscheidet zwischen sogenannten Input-Schichten, Hidden-Schichten und Output-Schichten.

Die Input-Schicht erhält Daten direkt aus dem Datensatz als Input, verarbeitet diese, erzeugt die Outputs und gibt diese anschließend an die Hidden-Schicht weiter. Die Outputs der Input-Schicht dienen nun den Inputs der aktuellen Hidden-Schicht und auch hier werden diese verarbeitet und dessen Outputs werden auch wieder an die nächste Schicht weitergeleitet. Anders als bei der Input- und Output-Schicht können beliebig viele Hidden-Schichten existieren. Am Ende werden die Outputs der letzten Hidden-Schicht wiederum an die Output-Schicht weitergeleitet. Der Output der Output-Schicht stellt eine Prädiktion, beziehungsweise eine Erwartung dar, welche im Zusammenhang mit Supervised Learning mit den Labels verglichen wird.

Da der Durchlauf solcher künstlichen neuronalen Netze linear von Schicht zu Schicht verläuft, werden diese Netze oftmals als Feed-Forward-Netze bezeichnet.

3.5 Convolutional Neural Network (CNN)

Einleitung Schon lange Zeit versucht man Computern kognitive Fähigkeiten, wie das Sehen, beizubringen. Das Gebiet Computer Vision beschäftigt sich mit digitalen Bildern und Videos und versucht diese zu analysieren und verstehen. Auch in diesem Gebiet sorgte Deep Learning für einen riesigen Durchbruch, denn mit sogenannten Convolutional Neural Networks[41], eine spezielle Netzwerkarchitektur, wurde es möglich Bilder wie ein Mensch zu sehen. Dabei werden häufig Klassifizierungs- und Detektionsalgorithmen verwendet, um zum Beispiel zwischen Hunde- und Katzenbildern zu unterscheiden oder Gesichter in einem Bild zu lokalisieren.

Bilder in Form von Tensoren Im Kontext eines Convolutional Neural Networks ist ein Bild ein vier-dimensionaler Tensor[14], ein Begriff für eine Matrix oder einen Array, welcher im Zusammenhang mit Machine Learning häufig verwendet wird. Beziehungsweise ein Array, verschachtelt in einem anderen Array, wiederum verschachtelt in einem anderen Array. Ganz klar hat ein Bild eine

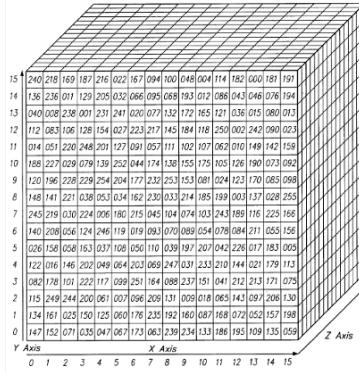


Abbildung 17: 4-D Tensor[4]

Länge und eine Breite bestehend aus einer Anzahl an Pixel, worauf schon mal auf einen zwei-dimensionalen Tensor schließen lässt. Jedoch darf hierbei nicht die Tiefe eines Bildes vergessen werden. Denn, wenn man einen zwei-dimensionalen Tensor hätte, würde jede Pixelposition nur einen Wert besitzen, was wiederum bedeuten würde, dass das Bild keine Farben besitzt, lediglich Graustufen. Denn eine Pixelposition in einem Bild aus dem RGB-Farbraum besitzt drei Werte, welche den Rot-Wert, den Grün-Wert und den Blau-Wert darstellen. Man kann sich das visuell vorstellen als gäbe es drei Bilder, welche übereinanderliegen, wobei eines nur die Rot-Werte, ein anderes nur die Grün-Werte und ein anderes nur die Blau-Werte anzeigt. Diese in Farben aufgeteilten Bilder nennt man in der Fachsprache Farb-Channels. Demnach hat ein RGB-Bild drei Farb-Channels, wiederum hat ein Grayscale Bild, bzw. ein Schwarz-Weiß-Bild lediglich einen Farb-Channel, welcher die Lichtintensität darstellt.

Bedenkt man nun, dass ein Bild mit 512*512 Pixeln, also ein kleines bis mittelgroßes Bild, einen Tensor mit der Form 3*512*512 darstellt, muss man feststellen, dass dem 786 432 Werten entspricht. Um ein Feed-Forward-Netzwerk für solch ein Bild zu entwerfen, würde die erste Schicht somit 786 432 Neuronen besitzen, was eine enorme Menge an Neuronen darstellt, welche lediglich ein Supercomputer verarbeiten kann. Hinzu würden aber auch noch die nächsten Schichten kommen, welche in der Regel eine noch viel größere Menge an Neuronen besitzen. Daraus kann man schließen, dass Bilder zu verarbeiten eine undenkbare Rechenleistung erfordern würde und demnach ein Feed-Forward-Netzwerk keine Lösung für das intelligente Verarbeiten von Bildern darstellt. Demnach müsste ein Algorithmus existieren, welcher weniger Neuronen benötigt.

3.5.1 Filter und Features

Ein CNN verfolgt einen etwas anderen Ansatz, welcher außerdem weniger Neuronen benötigt, Daten zu verarbeiten und versucht dabei die eingehenden Daten in eine Form zu bringen, mit welcher das Netzwerk die Daten fundamental verstehen kann. Anstatt dass jedes Pixel eine Gewichtung hat, gibt es sogenannte Filter, welche zum einen Gewichtungen besitzen und sich somit im Rahmen eines Trainings verändern und zum anderen über das Bild pixelweise iterieren. Da eben nicht für jedes Pixel eines Bildes eine Gewichtung benötigt wird, sondern lediglich für die Filter, ist keine enorme Rechenleistung vorausgesetzt. Solch ein Filter wandert stufenweise über das Bild, beziehungsweise einen Tensor und erzeugt dabei einen neuen Tensor, welcher als Feature bezeichnet wird, wobei der Filter wichtige Muster extrahiert. Somit spiegeln sich die wichtigen Informationen eines Bildes in einem Feature wieder. Diese Filter sind nicht universell und

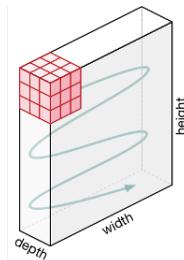


Abbildung 18: 3-D Filter[32]

haben demnach einige Parameter, welche sehr individuelle Filter, als auch Filterbewegungen, erlauben. Es gibt die sogenannte Filter-Größe, welche die Form des Tensors darstellt und zumeist ein 3x3 Tensor ist und zum anderen kann man den Parameter Stride definieren, welcher die Sprungweite pro Iteration angibt. Standardmäßig ist dieser Parameter auf den Wert 1 gesetzt und lässt den Filter somit jeweils um ein Pixel wandern. Des Weiteren kann man auch den Parameter Padding angeben, welcher gegebenenfalls einen leeren Rahmen um ein Bild

zeichnet. Dieser Parameter wird oftmals auf den Wert 1 gesetzt, um keine Pixel bei Bildern mit einer ungeraden Pixelanzahl zu verlieren.

Ein CNN verwendet jedoch nicht nur einen Filter, sondern viele verschiedene und hat auch, wie das gewöhnliche Feed-Forward-Netzwerk, Schichten. In diesen Schichten befinden sich nun verschiedene Filter, wobei jeder Filter ein dazugehöriges Feature erzeugt. Diese Features werden dann an die nächste Schicht weitergeleitet und von den Filtern dieser Schicht verarbeitet. Nun könnte man sich aber vorstellen, dass durch die vervielfachenden Features doch wieder eine hohe Rechenleistung benötigt werden würde. Dies ist grundsätzlich richtig, denn würden die Tensoren ihre ursprüngliche Größe beibehalten, so wäre das in der Tat sehr rechenintensiv. Deshalb musste eine weitere Operation eingeführt werden, welche die Tensoren regelmäßig verkleinert.

3.5.2 Pooling und Unpooling

Die sogenannte Pooling-Operation ist in der Lage einen Tensor zu schrumpfen. In der Regel wird ein Tensor halbiert und das geschieht indem man einen Pooling-Filter der Form 2×2 mit einem Stride von 2 verwendet. Bei einem Pooling-Filter kann es, je nachdem welche Pooling-Operation angewendet wird, Unterschiede zu dem herkömmlichen Filter geben. Zum einen gibt es das Max Pooling und zum anderen das Average Pooling.

- Beim Max Pooling wird immer der höchste Wert innerhalb eines Pooling-Filters, welche übrigens keine Gewichtungen besitzen, ausgewählt und in ein neues Feature extrahiert.
- Beim Average Pooling wird, wie der Name schon sagt, das Mittel aller Werte innerhalb eines Pooling-Filters erzeugt und in ein neues Feature extrahiert.

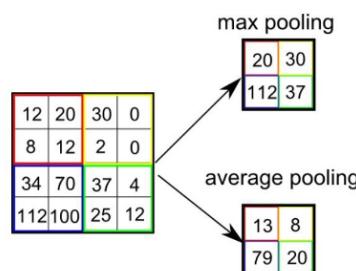


Abbildung 19: Pooling Operation[32]

Verfolgt man einen Ansatz, bei welchem am Ende ein Bild in der ursprünglichen Größe benötigt wird, so muss man zusätzlich Unpooling Operationen verwenden. Auch bei diesen muss beachtet werden, ob Max Unpooling oder Average

Unpooling verwendet werden soll. Denn beide haben Unterschiede, welche denen des Poolings ähnlich sind.

- Beim Average Unpooling werden die Werte eines Features vervielfacht und in ein größeres Feature übertragen. Die Besonderheit hierbei ist, dass jede Position des neuen Features den ursprünglichen Wert des kleineren Features erhält.
- Beim Max Unpooling wird ein Wert in eine Position innerhalb eines größeren Features übertragen und die restlichen Positionen werden mit dem Wert 0 befüllt. Das Auswählen dieser Position erfolgt jedoch nicht wahllos, sondern mit sogenannten Indices, welche bei der Pooling-Operation entstehen. Denn bei dieser kann mithilfe von Indices vermerkt werden, an welcher Position der Wert ursprünglich war.

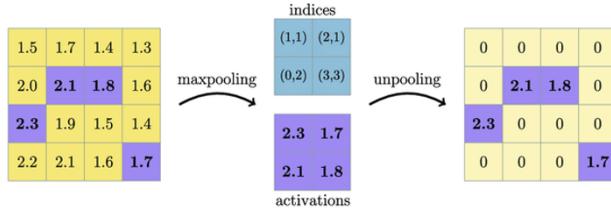


Abbildung 20: Unpooling Operation[32]

3.5.3 Fully Convolutional Neural Network (FCNN)

Eine Spezialform des CNN ist ein Fully Convolutional Neural Network[31], welches verwendet wird, wenn ein Bild nicht als Eingabe verwendet wird, sondern auch als Ausgabe des Netzwerks erwartet wird. Grundsätzlich spielt die Größe eines Bildes, beziehungsweise die Länge und Breite eines Bildes, keine Rolle, da bei dem Entwurf eines CNN nur die Filter relevant sind, welche völlig unabhängig von der Größe eines Tensors sind. Jedoch werden oftmals, speziell bei Klassifizierungs- und Detektionsalgorithmen, Netzwerkearchitekturen entwickelt, welche eine oder mehrere Feed-Forward-Schichten implementieren. Da bei einer Feed-Forward-Schicht eine fixierte Anzahl an Neuronen definiert ist, muss demnach jeglicher Input auf die passende Form gebracht werden, wobei die ursprüngliche Größe verloren geht. Natürlich macht dieser Ansatz Sinn bei einem Klassifizierungsalgorithmus, welcher eine fixierte Anzahl an Output-Neuronen besitzen muss. Jedoch gibt es Anwendungsfälle, welche ein Bild in der ursprünglichen Größe benötigt. Hierbei dürfen deshalb keine Feed-Forward-Schichten verwendet werden und somit bleibt die Größe der Tensoren im gesamten Netzwerk irrelevant. Das heißt, wenn man ausschließlich Convolutional Operationen verwendet, ist es grundsätzlich möglich die ursprüngliche Größe beizubehalten, jedoch muss es bei einem Auftreten einer Pooling Operation eine dazugehörige Unpooling Operation geben. Somit wird es möglich ein Bild mit der ursprünglichen Größe zu erzeugen.

3.6 Netzwerkarchitektur

CieLAB statt RGB Da Schwarz-Weiß-Bilder koloriert werden sollen, ist es nicht notwendig die Lichtintensität zu verändern, denn diese sollte dem Original gleichen. Somit ändert sich die grundlegende Struktur von Schatten, Konturen und Helligkeit nicht. Außerdem möchte man verhindern, dass das neuronale Netz diese Lichtintensität verändern kann, sonst müsste es zuerst lernen die Lichtintensität nicht zu manipulieren, jedoch die Farben schon. Deshalb sollte das neuronale Netz lediglich die Farben, unabhängig von der Lichtintensität, erzeugen.

Anstatt also den RGB-Farbraum zu verwenden, haben wir uns zum CieLAB-Farbraum gewandt, denn dieser bietet klare Vorteile in unserem Anwendungsfäll. Denn in RGB wird die Lichtintensität in jedem Farb-Channel interpretiert und hier können die Farben somit nicht von der Lichtintensität getrennt werden. Dazu benötigt man andere Farbräume, wie zum Beispiel der CieLAB-Farbraum, welcher zwei gesonderte Farb-Channels und einen Lichtintensitäts-Channel bietet. Hinzukommend existieren in diesem Farbraum mehr Farben, was aber in unserem Anwendungsfäll von keiner Bedeutung ist. CieLAB ermöglicht uns also, dass das Netzwerk lediglich die Farb-Channels erzeugt. Um am Ende ein ganzes Bild zu erzeugen, werden einfach die erzeugten Farb-Channels mit dem ursprünglichen Lichtintensitäts-Channel zusammengefügt. Deshalb ist es Pixorize möglich natürlich aussehende Bilder, welche die ursprüngliche Lichtintensität beibehalten, zu erzeugen. Die technische Zusammensetzung beider Modelle wurde im Kapitel 2.2.3 bereits genau beschrieben.

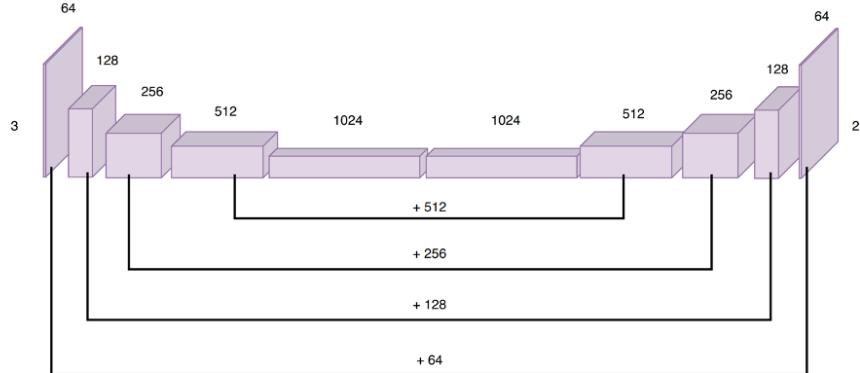


Abbildung 21: Grafische Darstellung der Netzwerkarchitektur

Entwurf Da wir ein Bild kolorieren wollen, ist zum einen klar, dass ein CNN verwendet werden muss, aber da wir ja nicht nur ein Bild für die Eingabe nutzen, sondern auch die Ausgabe ein Bild sein soll und die originale Größe eines Bildes erhalten bleiben soll, muss demnach ein FCNN in einer symmetrischen

Form verwendet werden. Symmetrisch bedeutet, dass das Netzwerk in der ersten Hälfte tiefer wird und in der zweiten Hälfte diese Tiefe wieder verliert. Dabei muss auch darauf geachtet werden, dass zu jeder Pooling Operation eine dazugehörige Unpooling Operation stattfindet. Denn beim Pooling schrumpft das Bild und um zur ursprünglichen Größe zurückzukommen, muss Unpooling ebenso verwendet werden, damit das Bild wieder größer wird.

Als Input werden 3 Channels verwendet, der Lichtintensitäts-Channel und die beiden Farb-Channel, welche die per Zufall gesetzten Farbpunkte beinhalten. Da empfohlen wird, dass man die Anzahl an Neuronen, beziehungsweise die Anzahl an Filtern, nach einer Zweierpotenz orientiert, beinhaltet die erste Schicht 64 Filter. Die darauffolgenden Schichten der ersten Hälfte des Netzes haben immer die doppelte Anzahl der vorherigen Schicht, wodurch eine starke Tiefe entsteht, welche für professionelle Anwendungen benötigt wird. Am Ende jeder Schicht der ersten Hälfte des Netzwerks folgt Max Pooling, wodurch die Features kleiner werden und eine gewisse Performance garantiert wird. In der zweiten Hälfte des Netzwerks wird die Anzahl an Filtern pro Schicht halbiert und da in der ersten Hälfte Max Pooling verwendet wurde, muss jede Schicht der zweiten Hälfte Max Unpooling implementieren. In der letzten Schicht, der Output-Schicht, gibt es nur 2 Filter, denn somit werden lediglich die Farb-Channels ausgegeben.

Skip Connections[6] und Checkerboard Artifacts Leider hat diese Architektur eine Schwachstelle, denn sie erzeugt künstliche Muster im Output-Bild. Diese Muster ähneln dem Muster eines Schachbrettes, weshalb man dies Checkerboard Artifacts nennt. Diese Musterung entsteht, wenn die Größe des Strides und des Filters nicht perfekt abgestimmt sind und sich dadurch Pixel überschneiden. Hat man einen Filter der Größe 3x3 und ein Stride von 2, so würden sich jeder zweite Pixel überschneiden wodurch diese Pixel intensiver verarbeitet werden, wodurch ein gerastertes Muster entsteht.[5] Um diese Mus-

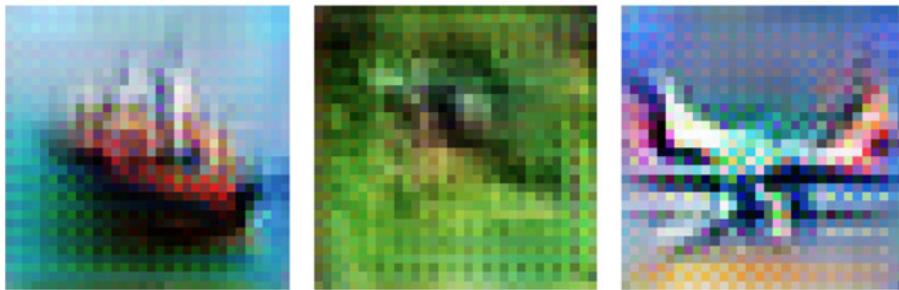


Abbildung 22: Checkerboard Artifacts[5]

ter zu verhindern wird für gewöhnlich empfohlen eine bikubische Interpolation statt einer Deconvolutional Schicht, eine Schicht, welche verwendet wird, um die Convolutional Operation umzukehren, weshalb Deconvolutional Schichten nur in der zweiten Hälfte des Netzwerks verwendet werden, zu verwenden. Jedoch kann man dies nur ersetzen, wenn keine Bilder unterschiedlicher Größen

verarbeitet werden würden, denn mit einer bikubischen Interpolation könnte es zu inkonsistenten Änderungen der Größe eines Tensors kommen. Dennoch gibt es eine weitere Möglichkeit, welche zwar nicht so effektiv wie eine bikubische Interpolation ist, jedoch eine Alternative darstellt, welche das Verarbeiten verschieden großer Bilder ermöglicht. Skip Connections. Zusätzlich bieten Skip Connections den Vorteil, dass sie bei einer U-förmigen Netzwerkarchitektur, wie die bei Pixorize, verhindert, dass grobe Details übersehen werden. Schließlich untersucht das neuronale Netz die Details eines Bildes auf einer sehr kleinen Ebene und setzt am Ende diese Details wieder zusammen, jedoch kann es dabei passieren, dass die Übergänge der Details unnatürlich wirken. So würde man ein Puzzle bauen, wobei versucht wird, dass ein klares Bild entsteht, jedoch wird am Ende jedes einzelne Puzzle-Teil klar erkennbar sein. Skip Connections können diese Granularitäten feiner zusammensetzen, indem bei einer Skip Connection mehrere Schichten übersprungen werden. Dadurch erhält die zweite Hälfte des Netzwerks zu sehen, wie der Tensor früher aussah und kann dabei das große Ganze im Blick behalten.

Implementierung in PyTorch[11] PyTorch ist ein Machine Learning Framework, welches von Facebook entwickelt wurde, um für Python einen Machine Learning Bausatz anzubieten. PyTorch ist ein Konkurrenzprodukt von Googles Tensorflow[13], bietet jedoch klare Vorteile, wie zum Beispiel, dass dynamischere neuronale Netze erzeugt werden können, weshalb PyTorch sehr häufig in der Forschung eingesetzt wird. Außerdem kann man zwischen einzelnen Schichten eines neuronalen Netzes debuggen, was von großem Vorteil ist, da neuronale Netze, wie Black Boxes sind und in der Regel nur schwer nachvollziehbar sind.

```
import torch.nn as nn
```

Mithilfe dieses Imports kann das Modul für neuronale Netze eingebunden werden. Indem man eine Klasse implementiert, welche von nn.Module erbt, kann man eine Architektur und den Feed-Forward-Prozess entwickeln.

```
class AE(nn.Module):
    def __init__(self):
        super(AE, self).__init__()

    def forward(self, x):
        return x
```

Im Konstruktor können verschiedene Attribute definiert werden, welche wie Bausteine agieren und in der forward-Methode eingesetzt werden. Die Reihenfolge im Konstruktor spielt keine Rolle, da hier wirklich nur die Attribute erzeugt werden, jedoch nicht verwendet werden. Wiederum spielt die Reihenfolge in der forward-Methode sehr wohl eine Rolle, schließlich werden die Attribute hier in einer bestimmten Reihenfolge zusammengesetzt.

```

def __init__(self):
    super(AE, self).__init__()
    self.leaky_reLU = nn.LeakyReLU(0.2)
    self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=1,
                           return_indices=True)
    self.unpool = nn.MaxUnpool2d(kernel_size=2, stride=2,
                                 padding=1)
    self.tanh = nn.Tanh()

    self.conv1 = nn.Conv2d(in_channels=3, out_channels=64,
                         kernel_size=3, stride=1, padding=1)
    self.conv2 = nn.Conv2d(in_channels=64, out_channels=128,
                         kernel_size=3, stride=1, padding=1)
    self.conv3 = nn.Conv2d(in_channels=128, out_channels=256,
                         kernel_size=3, stride=1, padding=1)
    self.conv4 = nn.Conv2d(in_channels=256, out_channels=512,
                         kernel_size=3, stride=1, padding=1)
    self.conv5 = nn.Conv2d(in_channels=512, out_channels=1024,
                         kernel_size=3, stride=1, padding=1)
    self.conv6 = nn.ConvTranspose2d(in_channels=1024,
                                  out_channels=512, kernel_size=3, stride=1, padding=1)
    self.conv7 = nn.ConvTranspose2d(in_channels=1024,
                                  out_channels=256, kernel_size=3, stride=1, padding=1)
    self.conv8 = nn.ConvTranspose2d(in_channels=512,
                                  out_channels=128, kernel_size=3, stride=1, padding=1)
    self.conv9 = nn.ConvTranspose2d(in_channels=256,
                                  out_channels=64, kernel_size=3, stride=1, padding=1)
    self.conv10 = nn.ConvTranspose2d(in_channels=128,
                                   out_channels=2, kernel_size=3, stride=1, padding=1)

```

Bei Pixorize wurden im Konstruktor die Aktivierungsfunktionen, Pooling- und Unpooling-Operationen, die Convolutional und Deconvolutional Schichten deklariert.

Bei den Convolutional Schichten ist zu erkennen, dass ein Filter der Form 3x3, ein Stride von 1 und ein Padding von 1 verwendet wird, wodurch die Tensoren keine Verluste in der Größe erleiden.

nn.Conv2d stellt eine Convolutional Schicht dar und nn.ConvTranspose2d stellt eine Deconvolutional Schicht dar, wobei beide Arten dieselben Parameter besitzen. out_channels gibt an, wie viele Features eine Schicht erzeugt, demnach beschreibt in_channels wie viele Features in einer Schicht fließen. Wie man erkennen kann besitzen die Deconvolutional Schichten mehr in_channels, als man erwarten würde. Um genau zu sein doppelt so viele, denn in die Deconvolutional Schichten fließen nicht nur die Features der vorherigen Schicht, sondern auch die Features der Skip Connections.

```

def forward(self, x):
    x = self.leaky_relu(self.conv1(x))
    out1 = x
    size1 = x.size()
    x, indices1 = self.pool(x)
    x = self.leaky_relu(self.conv2(x))
    out2 = x
    size2 = x.size()
    x, indices2 = self.pool(x)
    x = self.leaky_relu(self.conv3(x))
    out3 = x
    size3 = x.size()
    x, indices3 = self.pool(x)
    x = self.leaky_relu(self.conv4(x))
    out4 = x
    size4 = x.size()
    x, indices4 = self.pool(x)
    x = self.leaky_relu(self.conv5(x))
    x = self.leaky_relu(self.conv6(x))
    x = self.unpool(x, indices4, output_size=size4)
    x = self.leaky_relu(self.conv7(torch.cat((x, out4), 1)))
    x = self.unpool(x, indices3, output_size=size3)
    x = self.leaky_relu(self.conv8(torch.cat((x, out3), 1)))
    x = self.unpool(x, indices2, output_size=size2)
    x = self.leaky_relu(self.conv9(torch.cat((x, out2), 1)))
    x = self.unpool(x, indices1, output_size=size1)
    x = self.tanh(self.conv10(torch.cat((x, out1), 1)))

    return x

```

Wie gesagt wird in der forward-Methode der Feed-Forward-Prozess abgebildet und die im Konstruktor angelegten Attribute werden in einer Reihenfolge gebracht, wobei die Variable x den Tensor zur Laufzeit darstellt. Wie man erkennen kann werden für fast alle Schichten die Aktivierungsfunktion LeakyReLU verwendet, welche der ReLU-Funktion ähnelt. Die Besonderheit ist, dass der Wertebereich leicht unter null erweitert wird, wodurch das sogenannte Absterben von ReLU-Neuronen verhindert wird. Denn es kann vorkommen, dass manche ReLU-Neuronen immer schwächer werden und schließlich immer null als Wert ausgeben, wobei man dies als totes Neuron bezeichnet. Die Tanh-Funktion wird in der Output-Schicht verwendet, da diese zum einen für eine Regression benötigt wird und zum anderen, weil diese Funktion einen Wertebereich von -1 bis 1 und somit aussagekräftige Ausgabe-Werte besitzt.

3.7 Training und Evaluierung

CUDA[42] Da der Datensatz aus zirka 70.000 Bildern mit jeweils 1024*1024 Pixel besteht, musste eine große Rechenleistung gegeben sein, um ein Training in einer absehbaren Zeit durchzuführen. Durch die Verwendung eines Servers mit einer leistungsfähigen NVIDIA-Grafikkarte wurde es möglich, die Rechenleistung, im Vergleich zur Verwendung eines Prozessors, um ein Vielfaches zu steigern, wodurch die durchschnittliche Trainingszeit auf zirka fünf Stunden verkürzt wurde. Um das Berechnen also auf die Grafikkarte auszulagern, wurde die Bibliothek CUDA von NVIDIA verwendet, welche es sehr einfach macht, einzelne Variablen in den Grafikspeicher zu übertragen. Die Berechnung auf der Grafikkarte ist besonders effizient bei Matrixrechnungen, welche bei künstlichen neuronalen Netzen gegeben sind, da die gewichteten Inputs und Outputs als Matrizen innerhalb von PyTorch umgesetzt werden.

```
device = torch.device("cuda" if torch.cuda.is_available() else
                     "cpu")
model = AE().to(device=device)
```

Wie hier gut erkennbar ist, wird die Variable device deklariert und gegebenenfalls als CUDA-Variable initialisiert, falls eine NVIDIA-Grafikkarte vorhanden ist. Mit der Funktion .to(device=device) ist es möglich, dass man eine Variable auf der Grafikkarte initialisiert.

Hyperparameter Beim Training gibt es eine Schleife, in welcher über die Daten iteriert wird und dabei die Gewichte des neuronalen Netzes aktualisiert werden. Diese Aktualisierung der Gewichte ist stark vom Fehler abhängig. Um zu bestimmen wie hoch der Fehler, beziehungsweise die Abweichung zwischen der Ground Truth und dem Output des Netzes, ist, muss ein Kriterium, beziehungsweise eine mathematische Formel, ausgewählt werden.

```
criterion = nn.MSELoss(reduction="sum")
```

Bei Pixorize wurde MSELoss, also der Mean Squared Error[53], verwendet, da dies eine häufig verwendete Fehlerfunktion im Zusammenhang mit Regressionen ist.

Eine weitere wichtige Rolle für das Aktualisieren der Gewichte spielt der sogenannte Optimizer. Durch das Kriterium wird festgestellt, wie groß der Fehler ist, wobei sich der Optimizer darum kümmert, den Fehler zu verringern, indem er bestimmt wie sich die einzelnen Gewichte ändern.

```
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

Hier wird das Verfahren Adam[9] verwendet, da dieses der beste Optimizer ist und demnach sehr verbreitet ist, wobei die Gewichte des Netzes und die sogenannte Learning-Rate übergeben werden müssen.

Die Learning-Rate kann als Schrittweite pro Aktualisierung betrachtet werden, beziehungsweise gibt sie an, wie stark sich ein Gewicht maximal pro Iteration ändern kann. Die Anpassung dieses Parameters macht viel Sinn, schließlich möchte man den minimalsten Fehler erhalten, wobei aber lokale Minima existieren, in welchen Gewichte hängen können, falls die Learning-Rate sehr gering ist. Ist die Learning-Rate groß, ist es wahrscheinlicher solche lokalen Minima zu überspringen, jedoch könnte es passieren, dass sich die Gewichte nicht im globalen Minimum einpendeln, wodurch der Fehler während des Trainings sehr inkonsistent schwankt. Deshalb ist es notwendig, diesen Parameter laufend pro Trainingsdurchlauf zu ändern, um somit einen passenden Wert zu finden, wobei der Wert zwischen 0,1 und 0,000001 liegen sollte. Bei Pixorize beträgt die Learning-Rate 0,0001.

Training Loop

```
while True:
    data = datapreparer.load_images_in_parts()

    if len(data) == 0:
        print("break train")
        break

    dataloader = DataLoader(data, batch_size=1, shuffle=False)
    for batch in dataloader:
        image, label = batch
        image = image.to(device=device)
        label = label.to(device=device) # ground truth

        output = model(image).to(device=device)

        loss = criterion(output, label)

        optimizer.zero_grad()
        loss.backward() # calculate gradients
        optimizer.step() # update weights
```

Hier ist das Grundgerüst der Training Loop zu sehen. Dabei fällt auf, dass die batch_size 1 beträgt, wodurch jedes Bild einzeln in das Netz eingeführt wird und dass nach jedem einzelnen Bild die Gewichte aktualisiert werden. Bei der Verwendung anderer Architekturen wird für gewöhnlich eine größere batch_size verwendet, um mehrere Bilder gepuffert zu verarbeiten, jedoch kann dies nur getan werden, wenn jedes Bild dieselbe Größe besitzt. Da wir die Möglichkeit für verschiedene große Bilder offenlassen wollten, haben wir uns für eine batch_size von 1 entschieden. Der DataLoader returniert eine Liste bestehend aus mehreren Tupel, beziehungsweise Paare bestehend aus Input Bilder und deren Ground Truth. Darauffolgend wird das Input Bild in das Netz gefüttert und ein

Output generiert. Dieser Output wird anschließend mithilfe des Kriteriums mit der Ground Truth verglichen, wodurch ein Fehlerwert entsteht. Da der Fehler bestimmt wurde, ist es nun möglich, dass die Gewichte aktualisiert werden.

Matplotlib[33] Nach dem Training wäre es nun natürlich sehr relevant zu wissen, wie gut das Netz abgeschnitten hat. Deshalb werden während des Trainings die Fehler mitprotokolliert, um diese nach dem Training in einer Statistik anzuzeigen. Man erhofft natürlich zu sehen, dass sich der resultierende Graph degressiv verhält, beziehungsweise, dass der Fehler geringer wird. Natürlich muss jedoch beachtet werden, dass es völlig normal ist, wenn der Fehler auch mal in die Höhe schwanken kann. Dennoch sollte der Fehler im großen Ganzen kleiner werden, wobei der Fehler in der Regel am Anfang am meisten abnimmt. Mithilfe

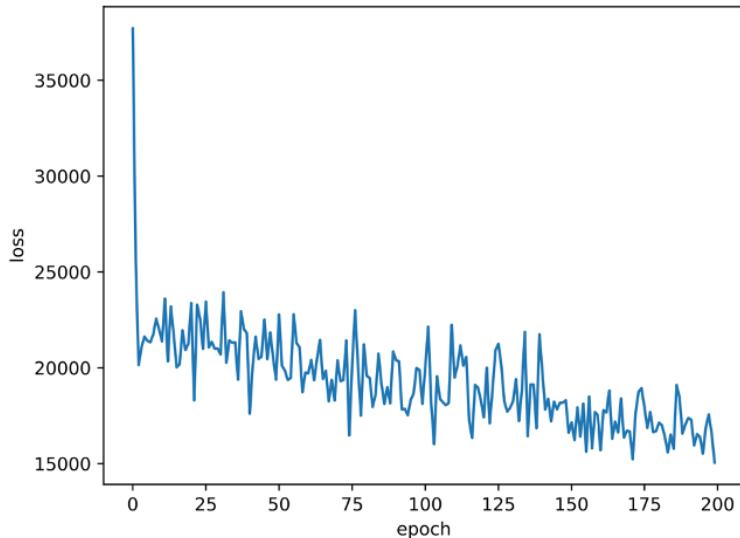


Abbildung 23: Verlauf des Fehlers

der Bibliothek Matplotlib ist es möglich Daten zu visualisieren und Statistiken aller Art zu erzeugen. In dieser Statistik wurde jeder vierzigste Fehler mitprotokolliert, wodurch man ein Bild mit nur geringen lokalen Schwankungen erhält.

Resultate Durch die Verwendung von CieLAB, statt RGB, ist es uns möglich gewesen, die ursprünglich Lichtintensität beizubehalten, wodurch die Ergebnisse natürlicher, als die der Konkurrenz (ColouriseSG[8]) sind. Jedoch ist unser künstliches neuronales Netz nicht nur wegen des Beibehaltens der Lichtintensität, sondern auch wegen der Spezialisierung auf Porträt-Bilder besser.

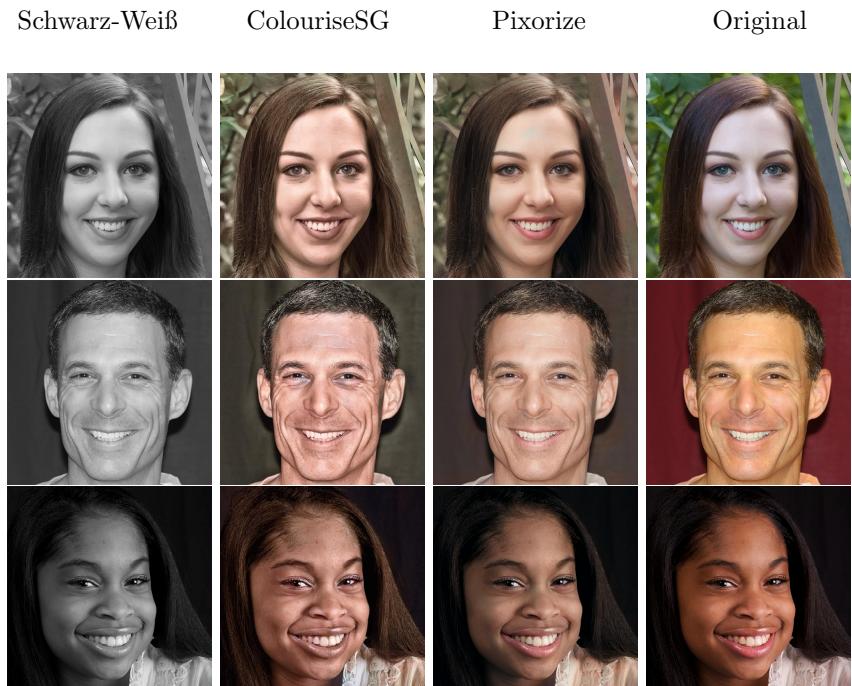


Abbildung 24: Vergleich zwischen der Konkurrenz (ColouriseSG) und Pixorize



Abbildung 25: Vollautomatisch koloriertes Porträt von Richard Feynman

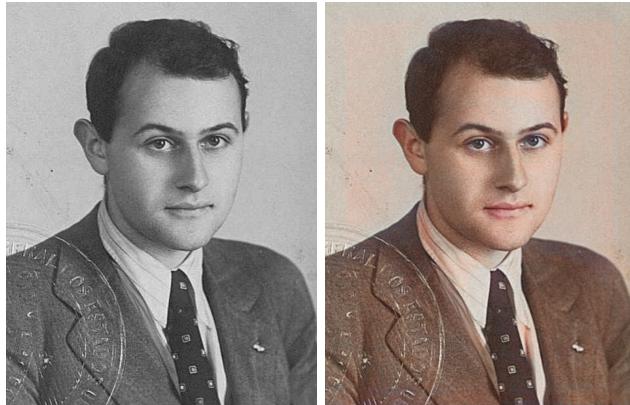


Abbildung 26: Vollautomatische Kolorierung

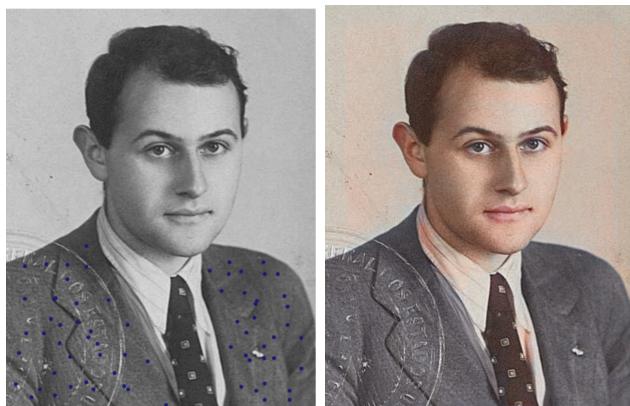


Abbildung 27: Benutzerinteraktive Kolorierung mit blauen Farb-Punkten

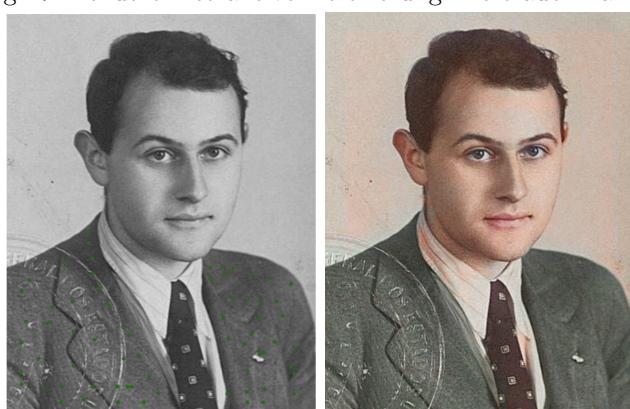


Abbildung 28: Benutzerinteraktive Kolorierung mit grünen Farb-Punkten

4 Grafische Benutzer Oberfläche - Manuel Riedl

4.1 Einleitung

Als grafische Benutzeroberfläche, oft UI abgekürzt, bezeichnet man die Oberfläche, womit Menschen mit Maschinen – Computern interagieren. UIs können schlichte Kommandozeilen-Fenster (CLI) sein, worüber man nur mit textbasierten Befehlen mit dem Computer interagiert, aber auch komplexe grafische Benutzeroberflächen, kurz GUI. Über die GUI haben Menschen die Möglichkeit, der Maschine, Aufgaben zu erteilen, ohne die komplexen Abläufe im Hintergrund zu verstehen. GUIs beinhalten grafische Steuerelemente, wie Buttons, Icons, Slidern und Bilder worüber der Benutzer durch simples Bedienen mit Maus und Tastatur dem Programm Befehle erteilt. Die GUI ist heutzutage die meist verwendete Art eines UI. Egal ob Sie auf ihren Smartphone Apps verwenden oder am Computer Programme benutzen, man muss fast nirgends mehr nur mittels Kommandozeile mit der Anwendung interagieren.

4.1.1 UI und UX

Um den Unterschied zwischen User Interface und User Experience zu verstehen, muss zuerst geklärt werden, was Webdesign eigentlich ist. UI und UX, aber auch IA (Informationsarchitektur, befasst sich mit Site-Mapping und Navigation) und CRO (Conversion Rate Optimierung, Feinabstimmung des Seitendesigns) zählen zu den Kategorien des Webdesigns. Im Allgemeinen befasst sich Webdesign mit dem Visuellen und den Funktionalitäten einer Webseite.

Das User Interface stellt die Oberfläche, auf der die Interaktion zwischen Computer und Mensch stattfindet, dar. Ziel des UI ist es, die Benutzerschnittstelle übersichtlich, nutzbar und sinnvoll zu gestalten, damit diese einfach bedient werden kann. Für den Benutzer ist das UI der wichtigste Aspekt der Anwendung. Die einzelnen Bedienelemente müssen bedacht an die Zielgruppe und aufeinander abgestimmt sein. Sei es die Farbe des Layouts oder die Größe der Buttons, alles muss stimmen um den User an die Anwendung zu binden.

Die User Experience umfasst alle Interaktionen eines Nutzers mit der Webseite. UX beschreibt alle Wahrnehmungen und Reaktionen die vor, während und nach der Nutzung auftreten. Außerdem behandelt UX die Strukturierung und die Organisation von Webseiten. Aktionen, wie der Benutzer von Webseite zu Webseite kommt oder wie Funktionalitäten angewendet werden können sind weitere Teile der UX. Weiters ist die Joy of Use ein wichtiger Aspekt. Sie entsteht, wenn der Nutzer mit Hilfe der Webseite seine Nutzungsziele einfach und angenehm erreichen kann.

4.2 Konzept

Im folgenden Kapitel werden die Funktionalitäten von Pixorize sowie die Bedienung des Programms vorgestellt. Weiters werden alle verwendeten Technologien beschrieben und die Kommunikation mit dem Server wird erläutert.

4.2.1 Workflow

Dieser Abschnitt präsentiert die Benutzeroberfläche und beschäftigt sich mit den Funktionalitäten die Pixorize zu bieten hat.

Bild laden Dieses Fenster erscheint beim Start von Pixorize. Es beinhaltet das Logo, eine kurze Auflistung der Funktionen und die Möglichkeit ein Bild zu laden.

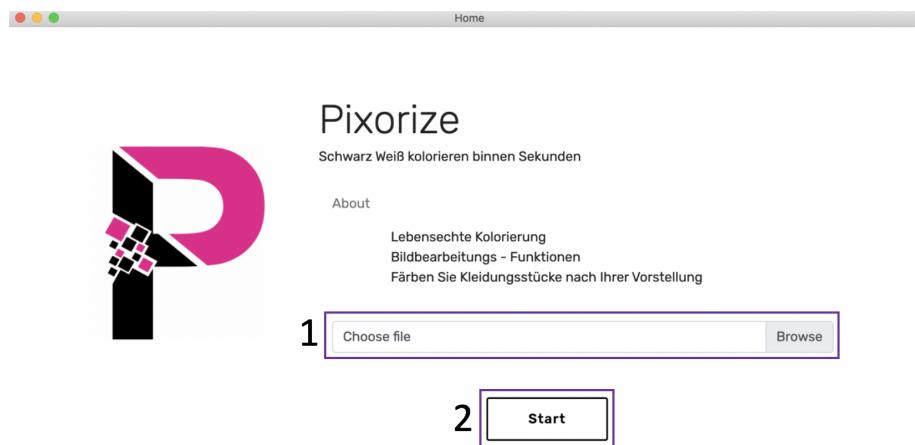


Abbildung 29: Ladefenster

1. Durch den File-Manger wählt der Benutzer sein Bild aus [4.4.1]
2. Durch klicken auf Start öffnet sich das Bearbeitungsfenster [4.4.1]

Bild bearbeiten Das Bearbeitungsfenster erscheint nach erfolgreichem Laden des Bildes. Dieses Fenster ist der Kern der Grafischen Benutzeroberfläche. Es beinhaltet alle Funktionen, die für die Bearbeitung notwendig sind. Das Fenster ist in drei Sektionen gegliedert: Werkzeugeiste – Bearbeitungsfläche – Farbauswahl.

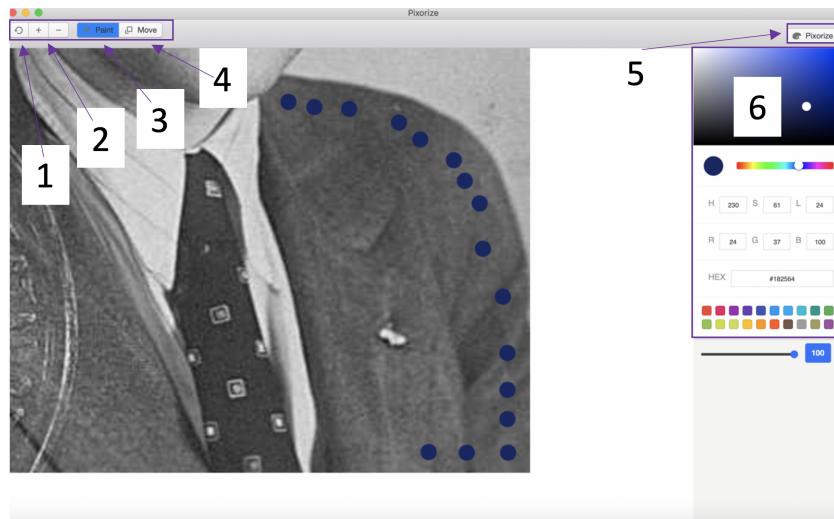


Abbildung 30: Bearbeitungsfenster

Dem Benutzer wird durch die Zurück-Funktion die Möglichkeit geboten, fehlerhaft gesetzte Farbpunkte wieder zu entfernen [4.4.2]

Durch Aufrufen der Zoom-Funktionen, kann das Bild skaliert werden [4.4.3]

Beim Auswählen der Paint-Funktion, kann man Farbpunkte auf dem Bild setzen [4.4.2]

Durch Auswählen der Move-Funktion, kann der Benutzer das Bild bewegen [4.4.3]

Beim Klicken auf Pixelize, wird das Bild mithilfe des künstlichen neuronalen Netzes koloriert. Für die Dauer des Vorgangs erscheint eine Progress Bar [4.4.4]

Der Color-Picker ermöglicht es dem Benutzer, eine Farbe auszuwählen [4.4.2]

Processing Image



Abbildung 31: Progress Bar

Bild anzeigen Nachdem das Bild mithilfe des künstlichen neuronalen Netzes koloriert wurde, erscheint das Präsentationsfenster. Dieses Fenster beinhaltet zusätzliche Funktionen die der Nachbearbeitung des Bildes dienen. Außerdem ist eine Möglichkeit vorhanden, das Bild zu speichern.

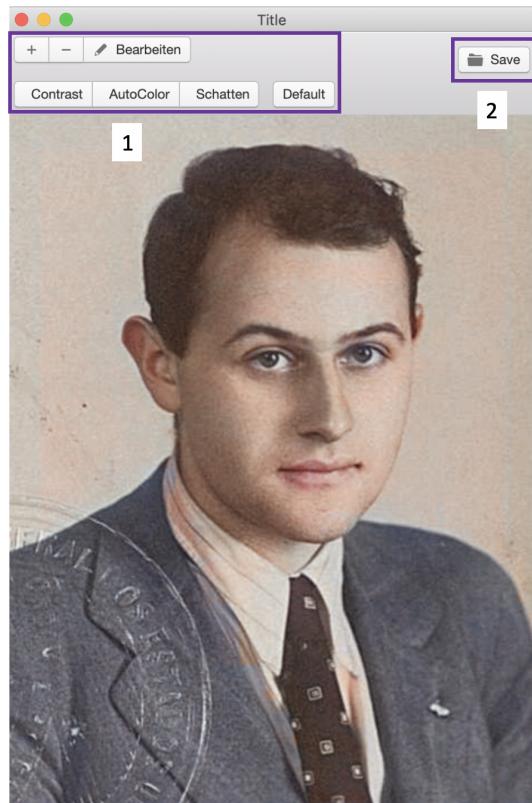


Abbildung 32: Präsentationsfenster

1. Der Benutzer kann mit den Nachbearbeitungs-Algorithmen den Kontrast verbessern, die Farben und Schatten verbessern. [4.4.5]
2. Mittels der Save-Funktion kann der Benutzer sein Bild speichern und erhält, falls dies erfolgreich verlaufen ist, eine Benachrichtigung [4.4.5]

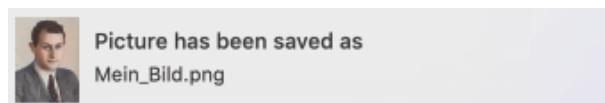


Abbildung 33: Save-Benachrichtigung

4.2.2 Technologienstack

Im Folgenden werden alle verwendeten Programme, Technologien und Bibliotheken vorgestellt. Das Programm wurde mithilfe von HTML, CSS und JavaScript entwickelt. Um eine Desktop-Anwendung zu erstellen wurde zusätzlich noch das Framework Electron verwendet. Um eine Kommunikation mit Python zu etablieren wurde der Python-Server “Flask“ verwendet.

HTML Der Begriff HTML steht für “Hypertext Markup Language“, auf deutsch “Hypertext Auszeichnungssprache“[3]. Im Allgemeinen gibt HTML die Struktur und den Aufbau einer jeden Webanwendung vor. Browser wie Chrome oder Firefox übersetzen diesen Text binnen Millisekunden und zeigen den Inhalt als Webseite an.



CSS Die Abkürzung CSS steht für “Cascading Style Sheets“, was übersetzt so viel bedeutet wie “gestufte Gestaltungsbögen“[2]. CSS wird für die grafische Gestaltung einer Webseite verwendet. Farbe, Seitenabstand oder Schriftgröße ändern, sind nur einige von unzähligen Funktionen die CSS bereitstellt. Als Hauptgrundlage für die Gestaltung der GUI wurde das für CSS-Layout “photonkit“ verwendet[29]. Photonkit wurde speziell für Desktop-Anwendungen entwickelt und ist im simplen “Apple“ Design gehalten. Photonkit stellt eine ganze Menge an vordefinierten Klassen bereit, die die Strukturierung und das Aussehen der Webseite bestimmen.



Klasse	Beschreibung
window	Umschließt die gesamte Anwendung
window-content	Beinhaltet den gesamten Inhalt
pane-sm sidebar	Erstellt eine Sidebar
toolbar toolbar-header	Erstellt eine Menüleiste
toolbar-actions	Erstellt eine Reihe für Buttons
btn-group	Erstellt eine Button Reihe
btn btn-default	Erstellt einen Button
icon icon-name	Fügt ein Icon im Button ein

Tabelle 2: Photonkit-Klassen

```

<div class="window">
  <header class="toolbar toolbar-header">
    <h1 class="title">Header with actions</h1>
    <div class="toolbar-actions">
      <div class="btn-group">
        <button class="btn btn-default">
          <span class="icon icon-home"></span>
        </button>
        ...
      </div>
      ...
    </header>
    <div class="window-content">
      <div class="pane-group">
        <div class="pane-sm sidebar"></div>
        ...
      </div>
    </div>
    <footer class="toolbar toolbar-footer">
      ...
    </footer>
  </div>

```

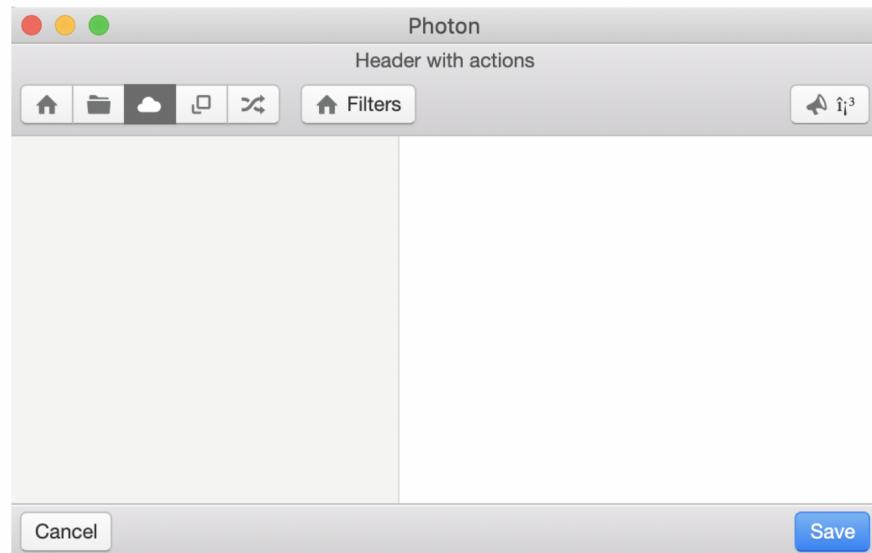


Abbildung 34: Beispiel einer Desktop-Anwendung mit Photon

JavaScript JavaScript, kurz JS, ist eine Programmiersprache, mit der man interaktiv auf Funktionen durch den Benutzer reagieren kann [19]. Durch eine Interaktion mit der Webseite z.B: Button klicken wird ein Event ausgelöst. Dieses Event ist im JS-File definiert und kann entsprechend behandelt werden.



Ebenfalls wurde die JavaScript-Bibliothek jQuery verwendet [18]. jQuery ist eine Bibliothek, die das Arbeiten mit Javascript übersichtlicher und effizienter gestaltet. Durch den \$ Operator wird die Bibliothek angesprochen.

Mit `$(‘id’)` wird auf ein Element zugegriffen. Dies ist um einiges kürzer als mit JavaScript.

```
//mit jQuery
let element = $('#element-id');
//mit reinem JavaScript
let element = document.getElementById("element-id");
```

Um auf Eigenschaften der HTML-Elementen, wie Höhe oder Breite eines Canvas zuzugreifen, muss mit jQuery folgendes beachtet werden: jQuery liefert ein Array zurück und keine simple Variable wie JavaScript.

```
//mit jQuery
const canvas_referenz = $('#canvasID');
let height = canvas_referenz[0].height;
//mit reinem JavaScript
const canvas_referenz = document.getElementById("canvasID");
let height = canvas_referenz.height;
```

Mit jQuery können Module erstellt werden. Diese Module können als Art “Klasse“ angesehen werden. Module beinhalten meistens mehrere Funktionen, die denselben Kontext behandeln. Durch das Zusammenfassen mehrere Funktionen zu einem Modul, wird die Übersicht des JS-Files gewährleistet.

Übergabeparameter des Moduls werden direkt bei der Definition im Funktions-Teil angegeben. Das Modul kann ohne diese Übergabeparameter, zB: `canvas_ref` im unteren Beispiel, nicht erstellt werden. Variablen, die im jeweiligen Modul angelegt werden, sind nur in diesem Modul sichtbar. Funktionen werden mit `function` angelegt. Im `return`-Bereich des Moduls werden alle Funktionen aufgelistet, auf die außerhalb des Moduls zugegriffen werden muss.

```
const CanvasEvents = (function (canvas_ref) {
    let var1;
    const canvas = $(canvas_ref);

    function addCanvasEvents() {
        canvas.on('mousedown', setMouseDown);
    }
}
```

```

        function delCanvasEvents() {
            canvas.off('mousedown');
        }
        function setMouseDown() {
            mouseDown = true;
        }
        return {
            addCanvasEventsMover: addCanvasEventsMover,
            delCanvasEventsMover: delCanvasEventsMover
        }
    );
}

```

Um Funktionen aus den definierten Modulen zu verwenden, muss ein Typ dieses Moduls angelegt werden. Falls das Modul einen Übergabeparameter benötigt, muss dieser angegeben werden. Funktionen innerhalb des Moduls, werden über **Modulreferenz.Funktionsname** aufgerufen.

```

let canvasE = null;
canvasE = CanvasEvents(canvas);
canvasE.addCanvasEvents();

```

jQuery stellt außerdem eine Methode der asynchronen Datenübertragung zur Verfügung. Asynchronous JavaScript and XML, kurz AJAX ermöglicht es HTTP Anfragen durchzuführen und die HTML Seite zu verändern, ohne diese komplett neu zu laden[35]. Daten werden in Ajax im JavaScript Object Notation-Format übertragen. JSON ist ein standardisiertes, für Menschen einfach lesbbares, Datenformat zum Austausch von Daten zwischen Anwendungen.

Die asynchrone Datenübertragung ist ein Übertragungsverfahren, bei dem Daten asynchron übertragen werden, das heißt, die Übertragung erfolgt zu beliebigen Zeiten wenn gerade Kapazitäten am Server frei sind. Die Ajax-Engine managt die Kommunikation mit dem Server. Sie entscheidet wann Daten gesendet werden und wann das User-Interface aktualisiert wird.

Bei der synchronen Datenübertragung werden die Daten sofort vom User-Interface an den Server weitergeleitet. Die Übertragung wird mittels eines Taktsignals zeitlich synchronisiert. Der große Nachteil der synchronen Datenübertragung ist, dass während das User-Interface auf eine Antwort des Servers wartet, der User keine anderen Aktionen auf der Anwendung durchführen kann - die Anwendung freezed.

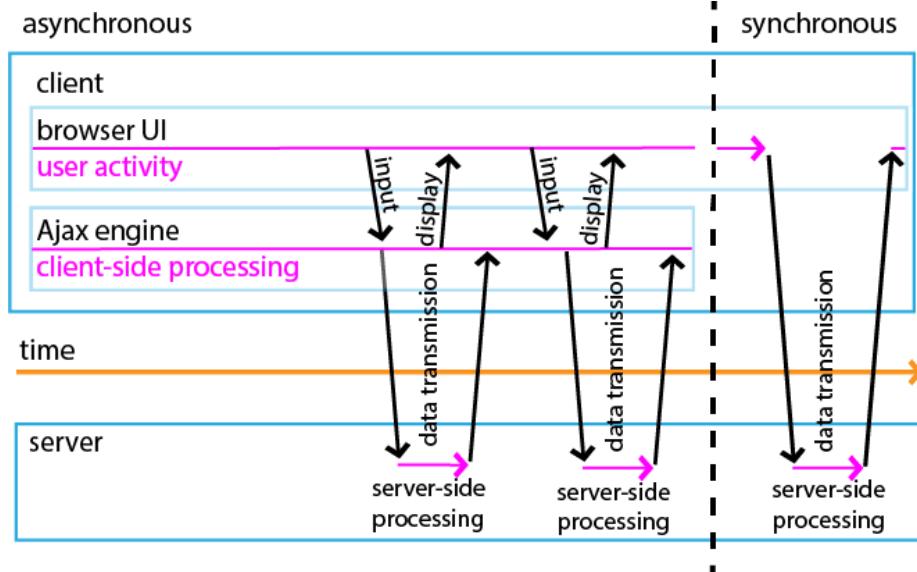


Abbildung 35: Synchrone und Asynchrone Datenübertragung

Mit `$.ajax` wird ein neuer Ajax-Call eingeleitet. Im `url`-Bereich wird die Adresse des Servers angegeben. Da es sich bei Pixorize um eine Desktop-Anwendung handelt, wird `localhost` verwendet. Jede einzelne Funktion, die am Server aufgerufen wird, hat ihre eigene Domäne. In diesem Fall wird die Funktion 1 unter der Domäne `/function1/` aufgerufen. Im `type`-Feld wird die Übertragungsart POST oder GET angegeben. Die Daten werden im JSON-Format im `data`-Bereich angegeben. Über den Namen, in diesem Fall `base64image`, können diese am Server wieder ausgelesen werden. Der `return`-Wert vom Server, also die zurück gesendeten Daten werden im `success`-Bereich ausgelesen.

```
function ajaxCall(transfer_data) {
  $.ajax({
    url: "http://127.0.0.1:5000/_function1/",
    type: "POST",
    data: {
      base64image: transfer_data
    },
    success: function (returnstring) {
      ipcRenderer.send('input-image', returnstring);
    }
});
```

Electron Electron ist ein Framework, das die Programmierung einer Plattformübergreifenden Desktop-Anwendungen ermöglicht[10]. Das Framework bedient sich bei den gängigen Wertentwicklungs-Sprachen HTML5, CSS, JavaScript und NodeJS. Unter der Verwendung von Chromium, einer Open Source Version des Chrom Browsers, werden diese Technologien ausgeführt. Electron ist nichts anderes als ein minimalistischer Browser, mit der zusätzlichen Funktion, mit dem Betriebssystem zu interagieren. Das Framework managt alle Interaktionen mit dem Betriebssystem. Es bedarf keiner speziellen Programmierung für unterschiedliche Systeme.

Electron Anwendungen bestehen aus zwei Prozessen: Einem Main-Prozess und keinem oder mehreren Renderer-Prozessen. Der Main-Prozess ist für den Life-Cyklus der Anwendung zuständig. Er erstellt und beendet Renderer-Prozesse. Außerdem ist es dem Main-Prozess möglich, mit den APIs des Betriebssystems zu kommunizieren. Der Renderer-Prozess ist für die Darstellung der Fenster zuständig. Der Prozess lädt die Webseite und stellt sie dar. Jeder der Prozesse verwendet die Multiprozess-Architektur von Chromium. Somit können der Main-Prozess und die Rednerer-Prozesse untereinander kommunizieren.

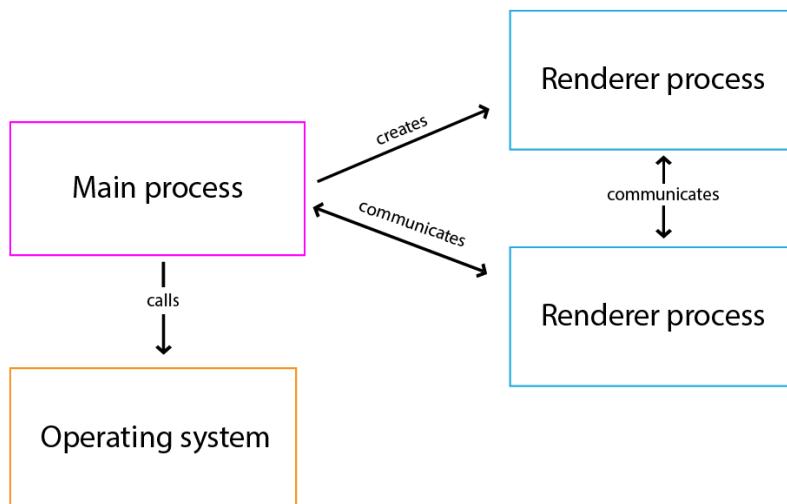


Abbildung 36: Main und Renderer Prozesse

Beim Starten der Anwendung wird als erstes die `package.json` ausgeführt. In der `json`-Datei sind der `Name`, die `Version`, die `main.js` und alle benötigten `node-module` definiert.

```
{  
  "name": "Pixorize",  
  "version": "1.0.0",  
  "main": "js/main.js"  
}
```

Die definierte `main.js` wird ausgeführt und der Main-Prozess erstellt die Renderer-Prozesse.

Ein neues Fenster wird immer mit `BrowserWindow` erzeugt. Darin definiert ist die Höhe und die Breite. Die `nodeIntegration` ist auf `true` gesetzt um Kommunikation zwischen diesem Fenster und dem Main-Prozess bzw. anderen Fenstern zu ermöglichen. Mit `.loadfile` wird das Html-File angegeben, welches im Fenster angezeigt werden soll. Das Html-File führt die JavaScript und CSS Dateien aus.

```
const {app, BrowserWindow} = require('electron');  
let imageLoadWindow = new BrowserWindow({  
  width: 1100,  
  height: 570,  
  webPreferences:{  
    nodeIntegration: true  
  },  
});  
imageLoadWindow.loadFile('load.html');
```

Mithilfe des ipc-Moduls wird zwischen den Prozessen kommuniziert. Dabei werden zwei Arten unterschieden: Das ipcMain-Modul und das ipcRenderer-Modul. Im JavaScript-File wird das ipcRenderer-Modul und im main-File das ipcMain-Modul verwendet. Mit `ipcRenderer.send()` wird eine Benachrichtigung an das ipcMain-Modul gesendet. Dieses empfängt die Nachricht durch den EventListener `ipcMain.on()`. Mithilfe von `.show()` wird das neue Fenster angezeigt und mit `.close()` das vorherige geschlossen.

```
//renderer.js  
const {ipcRenderer} = require('electron');  
function openNewWindow() {  
  ipcRenderer.send('open-new-window');  
}  
//main.js  
const {app, ipcMain} = require('electron');  
ipcMain.on('open-new-window', () => {  
  applicationWindow.show();  
  imageLoadWindow.close();  
});
```

Es kann nicht nur ein Event ausgelöst werden, es können auch Daten mitgesendet werden. Beispiel: Fenster 1 möchte an Fenster 2 Daten schicken. Fenster 1 schickt die Daten an die Main und diese leitet sie mithilfe von `window2.webcontents.send()` an Fenster 2 weiter. In Fenster 2 empfängt die Daten ein EventListener.

```
ipcMain.on('input-image', (evt, data)=> {
    applicationWindow.webContents.send('input-received', data);
});
```

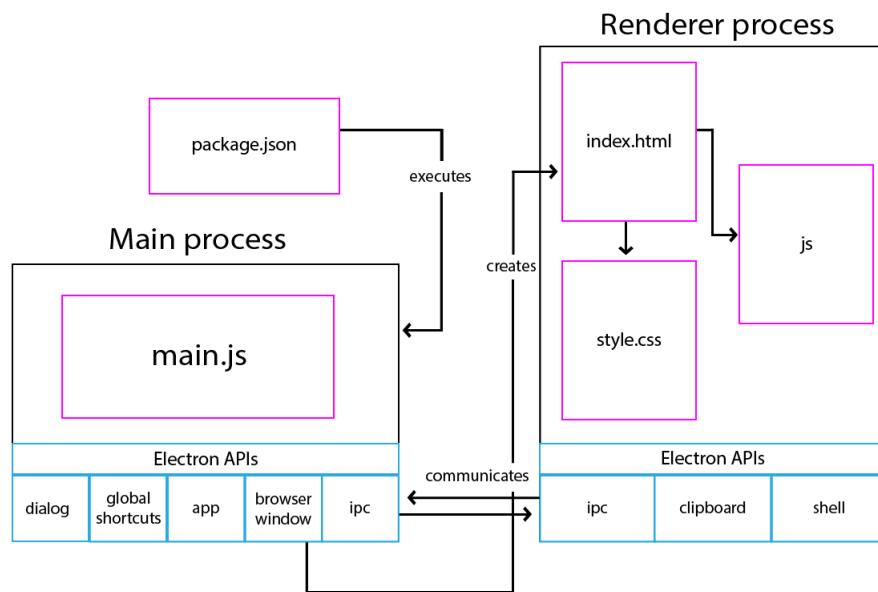


Abbildung 37: Electron lifecycle

4.3 Aufbau der Architektur

In diesem Abschnitt wird auf den Aufbau der Anwendung und der Kommunikation zwischen Client-Server eingegangen. Zusätzlich folgt zur besseren Verständlichkeit eine Übersicht über alle verwendeten Files.

4.3.1 Kommunikation

Das künstlich neuronale Netz läuft nur in einer Python-Umgebung. Um mit dem neuronalen Netz zu kommunizieren, muss eine Verbindung zwischen Javascript und Python entstehen. Bei Pixorize wurde das Webframework Flask[22], geschrieben in Python, verwendet. Bei Flask handelt es sich um einen Webserver im Python-Umfeld, das hat den Vorteil, dass dieser direkt auf das neuronale Netz zugreifen kann. Mithilfe von Ajax-Calls wird mit dem Flask-Server kommuniziert.

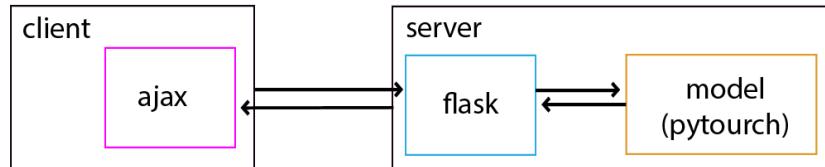


Abbildung 38: Kommunikation Client-Server

Über die Klassen `Flask` und `CORS` wird der Server erstellt. Durch `@app.route()` wird eine neue Domäne zum Server hinzugefügt. Zur Domäne gehört eine namensgleiche Funktion. Diese wird automatisch aufgerufen, sobald sich ein Client auf die Domäne verbindet. In dieser Funktion können alle typischen Python-Operation, wie Auslesen der Empfangen Daten oder das Aufrufen einer weiteren Funktion durchgeführt werden. Über den `return`-Wert liefert der Server das Ergebnis an den Client zurück.

```
app = Flask(__name__)
cors = CORS(app)

@app.route('/_get_data/', methods=['POST'])
def _get_data():
    base64image = request.form['base64image']
    string = run(base64image)
    return string
```

4.3.2 Übersicht über die Files

Es folgt eine Übersicht über alle verwendeten Files, deren Module sowie Funktionen.

Jedem Anwendungsfenster ist ein HTML-File und ein JavaScript-File zugeordnet. Dieses JavaScript-File kommuniziert mit dem main-File von Electron und dem Python-Server.

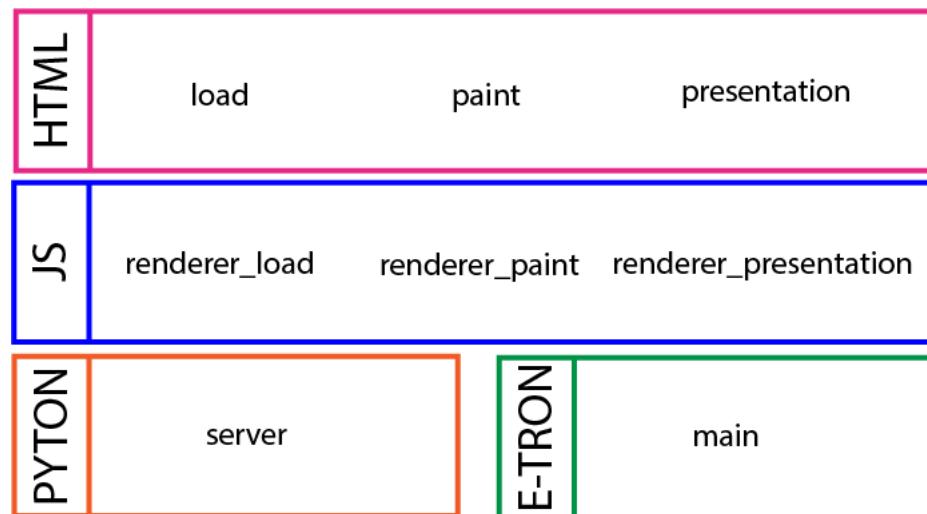


Abbildung 39: Abbildung aller Files

File: renderer.load

Modul: Load

load()	loadTheImage()
readTheFile()	

File: renderer.paint

Modul: Mover

addCanvasEventsMover()	delCanvasEventsMover()
setMouseDown()	setMouseUp()
move()	

Modul: Painter

delCanvasEventsPaint()	addCanvasEventsPaint()
delCanvasEventsProgressbar()	init()
getimages_guitartCords()	getPictureCoordsFromClick()
getPointCoordsForPicture()	drawOriginalImage()
isDrawablePoint()	point()
draw()	drawOnFakeCanvas()
colorchange()	
<i>Modul: Zoom</i>	
load()	loadTheImage()
readTheFile()	

Modul: Pixorize

saveRealCanvas()	goPython()
------------------	------------

Modul: Events

addEvents()	unbindEvents()
-------------	----------------

File: renderer_presentation

Modul: Actions

drawImageLoader()	addEvents()
contrast()	color()
shadow()	addButtons()
removeImage()	removeButtons()
save()	sendPath()
calculateWindowSize()	drawImage()

File: main.js

Funktionen

createWindow()	createPresntWindow()
----------------	----------------------

EventListender: ipcMain.on

open-new-Window	input-image
final-image	show-progressbar
resize	resize-buttonsAdd
set-progressbar-completet	save

File: server.py

Funktionen

autocolor()	autocontrast()
save()	into_sw()
run()	

Server-Domäne

_get_data	_contrast
_autocolor	_sw
_save	

Tabelle 3: Auflistung der verwendeten Files, deren Module sowie Funktionen

4.4 Implementierung

In diesem Abschnitt wird die Implementierung der Funktionen beschrieben. Zur Veranschaulichung der Funktionen wurden Grafiken hinzugezogen.

Canvas In der Anwendung werden alle Bild-Interaktionen mithilfe von Canvas umgesetzt. Canvas ist ein mit Höhen und Breiten-Angaben beschriebener Bereich, in den per JavaScript gezeichnet werden kann.

```
<canvas height="665" width="1070" id="canvasid"></canvas>
```

Um auf dem Canvas zu zeichnen, benötigt man eine Referenz auf den Context. Die `drawImage` Methode besitzt eine Unmenge von Parametern, die den Ausschnitt und die Positionierung des Bildes im Canvas bestimmen.

```
let canvas = $('#canvas');
let ctx = canvas[0].getContext("2d");
ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy,
              dWidth, dHeight);
```

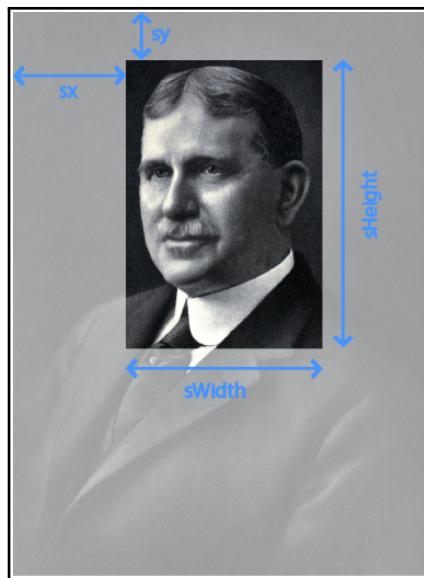


Abbildung 40: Ausgangs-Bild

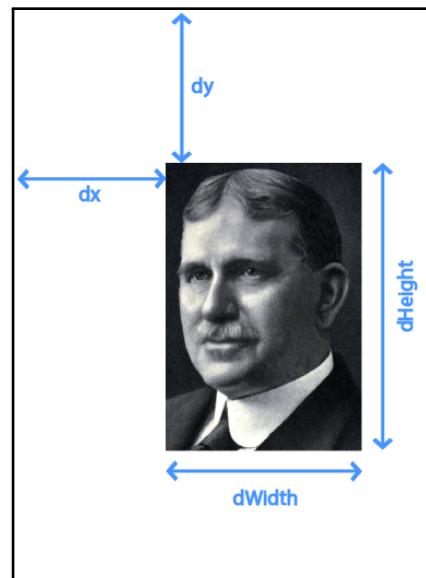


Abbildung 41: Bild im Canvas

4.4.1 Bild laden

Damit der Benutzer sein eigenes Bild verwenden kann, muss eine Lade-Funktionalität implementiert werden.

Um ein Bild aus dem File-System zu laden, eignet sich der `input`-Tag von Html. Durch die Spezifikation `type="file"` öffnet sich der File-Dialog des Benutzers. Um sicherzustellen, dass nur Bilder geladen werden können, werden im `accept`-Bereich die akzeptierten Dateiendungen angegeben.

```
<input type="file" accept="image/jpg, image/png" id="load" >
```

Um das ausgewählte Bild zu übernehmen, wird mittels jQuery ein EventListener auf den `input`-Tag gesetzt. Das Modul Load wird initialisiert und die `init`-Funktion aufgerufen.

```
('#load').on('change', function () {
    loader = Load();
});
```

In der `init`-Funktion wird ein neues File angelegt und die Funktion `readTheFile` aufgerufen. In `readTheFile` wird ein `FileReader` erzeugt und mithilfe dessen das Bild als `base64`-Codierung gelesen. Bei erfolgreichem Lesen wird die Funktion `loadTheImage` mit dem codierten Bild als Übergabeparameter aufgerufen.

```
function init() {
    const file = [...event.target.files].pop();
    readTheaFile(file).then((image) => loadTheImage(image))
}
```

Da es öfters vorkommt, dass alte Bilder einen Braunstich haben und um sicherzustellen, dass der Benutzer ein Schwarz-Weiß-Bild geladen hat, wird das Bild in Grey-Scale konvertiert. Mithilfe von Ajax wird der eingelesene `base64`-String an die Adresse `127.0.0.1:5000/_sw/` gesendet. Im `success`-Abschnitt des Ajax-Calls wird mithilfe des `ipcRenderer`-Moduls das konvertierte Bild, dass nun im Grey-Scale vorliegt, an das `ipcMain`-Modul von Electron gesendet.

```
function loadTheImage(image) {
    $.ajax({
        url: "http://127.0.0.1:5000/_sw/",
        type: "POST",
        data: {
            base64image: image
        },
        success: function (returnstring) {
            ipcRenderer.send('input-image', returnstring);
        }
});
```

Das ipcMain-Modul wird benachrichtigt und leitet den base64-String an das applicationWindow weiter.

```
ipcMain.on('input-image', (evt, data) => {
    applicationWindow.webContents.send('input-received', data);
});
```

Das ipcRenderer-Modul empfängt den **base64-String** und erstellt ein neues Image-Objekt. Das Bild wird zuerst in ein Dummy-Image geladen um mithilfe der **onload** Methode sicherzustellen, dass das Bild vollständig geladen ist. Erst wenn das Image-Objekt vollständig initialisiert ist, wird das Dummy-Bild auf das Original-Bild geladen und das Modul Events erstellt. Es wird die Funktion **addevents** aufgerufen, die im Events-Modul enthalten ist. In dieser werden alle User-Interaktionen (Button click, Color Picker..) hinzugefügt. Außerdem werden die Module Painter, Mover, Zoomer und Pixorize initialisiert.

```
ipcRenderer.on('input-received', (evt, data) => {
    const preloadedImage = new Image();
    $(preloadedImage).on('load', () => {
        originalImage = preloadedImage;
        events = Events();
        events.addevents();
        painter = Painter(canvas, context);
        mover = Mover(canvas);
        zoomer = Zoom(canvas);
        pixorize = Pixorize(canvas);
    });
    preloadedImage.src = data;
});
```

Nach der Initialisierung des Painter-Moduls wird die **init**-Funktion des Moduls aufgerufen. Innerhalb dieser wird der Mittelpunkt des Canvas berechnet, um das Bild mittig zu laden. Um Canvas Interaktionen zu ermöglichen, werden CanvasEvents hinzugefügt. In der **drawOriginal**-Funktion wird das Bild gezeichnet.

```
function init() {
    $('#paint1').addClass('activeButton');
    imageCenterX = canvas[0].width / 2;
    imageCenterY = canvas[0].height / 2;
    context.strokeStyle = 'black';
    addCanvasEvents();
    drawOriginalImage();
}
```

Die `drawOriginal`-Funktion wird immer dann aufgerufen, wenn der Benutzer das Bild verschiebt, zoomt oder einen Farbpunkt setzt. Die Methode löscht das alte Bild und zeichnet das Bild neu. Zu Beginn wird die Höhe und die Breite des Bildes berechnet. Da der User die Möglichkeit hat das Bild zu zoomen, verändert sich die Größe des Bildes. Um die korrekte Darstellung zu gewährleisten, wird jedes Mal aufs Neue, mithilfe des aktuellen Scalings die Höhe und Breite berechnet. Das alte Bild wird vom Canvas gelöscht, die aktuellen Start-Koordinaten des Bildes werden abgefragt und das Bild wird anhand der Werte gezeichnet. Anzumerken ist hierbei, dass das Originalbild zu keinem Zeitpunkt im Programm verändert wird, nur immer neu berechnet, sodass stets das Original erhalten ist.

```
function drawOriginalImage() {  
    let w = originalImage.width * scale;  
    let h = originalImage.height * scale;  
    context.clearRect(0, 0, canvas[0].width, canvas[0].height);  
    const start = getimages_guitartCords();  
    context.globalAlpha = opacity;  
    context.drawImage(originalImage, start[0], start[1], w, h);  
    context.globalAlpha = 1.0;  
}
```

Mithilfe der `getImageStartCords`-Funktion wird die aktuelle Position des Bildes im Canvas berechnet. In den `ImageCenter`-Variablen ist immer der aktuelle Mittelpunkt des Bildes in Abhängigkeit vom Canvas gespeichert. Durch das aktuelle Scaling des Bildes lassen sich die Start-Koordinaten berechnen.

```
function getImageStartCords() {  
    const startX = (imageCenterX - originalImage.width / 2 * scale);  
    const startY = (imageCenterY - originalImage.height / 2 * scale);  
    return [startX, startY]  
}
```

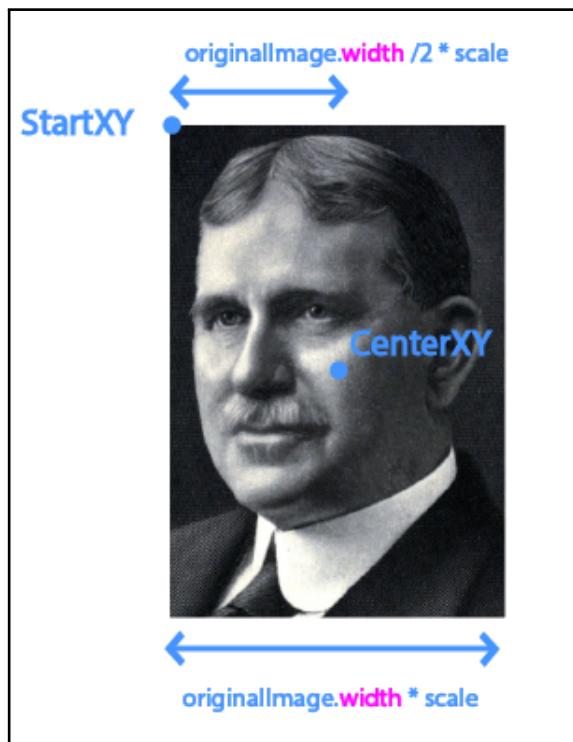


Abbildung 42: Function: `getImageStartCords()`

4.4.2 Farbpunkt setzen

Manuell gesetzte Farbpunkte dienen dem Neuronalen Netz als Information, wie gewisse Bereiche eingefärbt werden sollen.

Um einen Farbpunkt auf dem Bild zu setzen ist zuerst eine Farbpalette zur Farbauswahl notwendig. Bei Pixorize wurde das npm-Modul `a-color-picker` verwendet. Dieser Color-picker ermöglicht es dem Benutzer Farben selbst auszuwählen oder vordefinierte Farben zu verwenden. Um die ausgewählte Farbe abzurufen, wird eine Referenz auf den im HTML-File angelegten Color-picker über die Klasse `picker` hergestellt.

```
<!-- HTML-File -->
<div acp-color="#EFE9E7" acp-palette="#f44336|#e91e63|..." class="picker">

// Js-File
const colorPicker = AColorPicker.from('.picker');
```

Um auf Änderungen in der Farbpalette zu reagieren, wird ein `change`-Listener auf die Referenz der Farbpalette gesetzt. Dieser ruft eine Funktion auf, die die aktuell zu verwendende Farbe aktualisiert.

Alle Farbpunkte auf dem Bild, werden in einem Array von der Klasse Point gespeichert. In der Klasse Point sind alle notwendige Information (x/y-Kordinate / Farbe) des Farbpunkts enthalten. Jedes Mal wenn ein neuer Farbpunkt gesetzt wird, wird ein neues Point Objekt erzeugt.

```
const points = [];
const Point = (function (x, y, color) {
    return {
        x: x,
        y: y,
        color: color
    };
});
```

Beim Klicken auf das Canvas wird ein `click`-Event ausgelöst. Dieses `click`-Event ruft die Funktion `point` auf. In `point` werden die Kordinaten des Mauszeigers zum Zeitpunkt der Canvas-Interaktion ausgelesen und der Funktion `isDrawAblePoint` übergeben. Falls diese true liefert, wird `getPictureCoordsFromClick` aufgerufen. Die Funktion liefert als Ergebnis ein Point-Objekt das zum Array hinzugefügt wird. Zum Abschluss wird die `draw`-Funktion aufgerufen. Diese aktualisiert die Darstellung des Bildes auf der Benutzeroberfläche.

```

function point(evt) {
    const x = evt.pageX - canvas_coords.left;
    const y = evt.pageY - canvas_coords.top;
    if (isDrawablePoint(x, y)) {
        points.push(getPictureCoordsFromClick(x, y));
    }
    draw();
}

```

Um sicherzustellen, dass der Benutzer auf das Bild geklickt hat und nicht auf den Rest der Zeichenfläche, ist die Funktion `isDrawablePoint` zuständig. Mithilfe der Startkoordinaten, dem aktuellen Scaling und der Höhe sowie Breite des Bildes, werden die Endkoordinaten berechnet. Dadurch, dass sowohl Start-als auch Endkoordinaten des Bildes bekannt sind, weis man genau wo im Canvas sich das Bild befindet. Abhängig von einer simplen Überprüfung, ob sich die Koordinaten des Mausklicks innerhalb der Koordinaten des Bildes befinden, wird entweder `true` oder `false` zurückgegeben.

```

function isDrawablePoint(x, y) {
    const start = getimages_guitartCords();
    let imageEndX = (originalImage.width * scale + start[0]);
    let imageEndY = (originalImage.height * scale + start[1]);

    if (x > imageEndX || x < start[0] || y > imageEndY
        || y < start[1]) {
        return false;
    }
    else {
        return true;
    }
}

```

Die Farbpunkte werden relativ zum Originalbild im Array gespeichert. Da die Koordinaten nur relativ zum Canvas ausgelesen werden, müssen diese umgerechnet werden. Durch Abzug der Startkoordinaten und dividieren durch das aktuelle Scaling, erhält man die Koordinaten des Farbpunkts auf dem Originalbild. Die Koordinaten mit der zugehörigen Farbe des Punktes werden als Point Klasse formatiert und zurückgegeben.

```
function getPictureCoordsFromClick(x_click, y_click) {
    const start = getimages_guitartCords();

    const x_in_image = (x_click - start[0]) / scale;
    const y_in_image = (y_click - start[1]) / scale;

    return Point(x_in_image, y_in_image, context.strokeStyle)
}
```

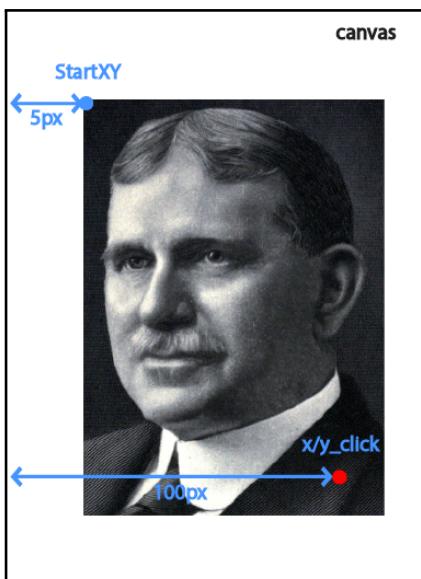


Abbildung 43: Relativ zum Canvas

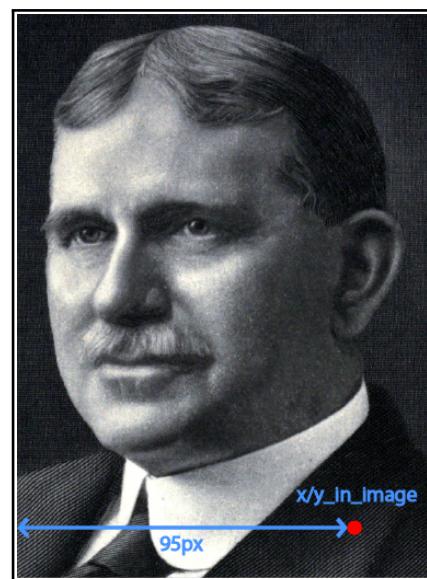


Abbildung 44: Relativ zum Bild

Innerhalb der `draw`-Funktion wird die Anzeige des Benutzers aktualisiert. Das Bild wird neu gezeichnet und die Farbpunkte werden neu gesetzt. Mit `forEach` wird das Array der vorhandenen Farbpunkte durchlaufen. Da die Farbpunkte im Array relativ zum Originalbild gespeichert sind, skaliert die Funktion `getPointsCoordsForPicture` die Farbpunkte relativ zum Canvas. Um einen Punkt zu zeichnen, wird die Methode `arc` verwendet. In dieser gibt man den Mittelpunkt des Kreises auf der x und y-Achse an, sowie den Radius und den Winkel. Der Radius ist abhängig vom aktuellen Scaling des Bildes.

```
function draw() {
    drawOriginalImage();
    points.forEach(p => {
        const drawCoords = getPointCoordsForPicture(p)
        context.beginPath();
        context.arc(drawCoords[0], drawCoords[1], 5 *
        scale, 0, 2 * Math.PI);
        context.fillStyle = p.color;
        context.strokeStyle = p.color;
        context.fill();
        context.stroke();
    });
    scaledifference = [0, 0];
    scalediffXY = [0, 0];
}
```

Durch die `undo`-Funktion hat der Benutzer die Möglichkeit, falsch gesetzte Farbpunkte zu löschen. Mit `pop()` wird das zuletzt hinzugefügte Element im Array gelöscht, und somit der Farbpunkt entfernt. Die `draw`-Funktion wird aufgerufen, um die Anzeige des Benutzers zu aktualisieren.

```
function undo() {
    points.pop();
    draw();
}
```

4.4.3 Bildposition ändern

Es gibt zwei Arten die Position des Bilds zu ändern: Zoomen und Bewegen

Zoomen Dadurch, dass in der `draw`-Funktion das aktuelle Scaling schon berücksichtigt wird und somit immer dynamisch die angezeigte Größe des Bildes berechnet wird, muss beim Zoomen nur das aktuelle Scaling geändert werden. Beim Zoomen wird das Maus-Event `mousewheel` ausgelöst, das die Funktion `zoom` aufruft. Über `wheelDelta` wird ermittelt, ob der User herauszoomt oder hineinzoomt.

```
function zoom(event) {
    if (event.originalEvent.wheelDelta > 0) {
        zoomIn();
    } else {
        zoomOut();
    }
}
```

Das aktuelle Scaling wird um den festgelegten Zoom-Step erhöht und die Anzeige des Benutzers wird aktualisiert.

```
function zoomIn() {
    if (scale < MAX_ZOOM) {
        scale += ZOOM_STEP;
        painter.draw();
    }
}
```

Bewegen Um zu bestimmen, wie weit der Benutzer das Bild verschieben will, müssen drei Maus-Events abgefragt werden: `mousedown`, `mouseup` und `mousemove`. Beim ersten Klicken auf das Canvas, werden die aktuellen Mauskoordinaten ausgelesen und relativ zum Canvas berechnet.

```
function setMouseDown(event) {
    mouseDown = true;
    oldmousePos = [event.pageX - canvas_coords.left,
                  event.pageY - canvas_coords.top];
}
```

Während der Benutzer das Bild bewegt, ist die Funktion `move` aktiv. In dieser wird die Verschiebung des Bildes anhand der Startkoordinaten und der aktuellen Position des Cursors berechnet. Die Differenz dieser beiden Werte wird zum Mittelpunkt des Bildes addiert, um die neue Position des Bildes festzulegen. Die aktuelle Position wird als `oldmousePos` angenommen und die Benutzeroberfläche wird aktualisiert, sodass die Verschiebung des Bildes sichtbar wird.

```

function move(event)
{
    if(mouseDown)
    {
        delta = [event.pageX - canvas_coords.left - oldmousePos[0],
                 event.pageY - canvas_coords.top - oldmousePos[1]];

        imageCenterX = imageCenterX + delta[0];
        imageCenterY = imageCenterY + delta[1];

        oldmousePos = [event.pageX - canvas_coords.left, event.pageY -
                      canvas_coords.top];
        painter.draw();
    }
}

```

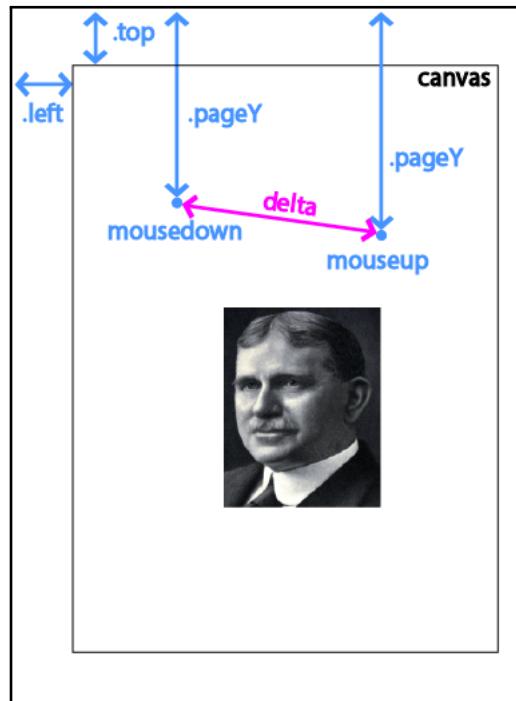


Abbildung 45: Bewegung des Bildes

Um zwischen “Farbpunkt setzen“ und “Bild bewegen“ zu wechseln, wurde ein Radio-Button verwendet, der die nicht/notwendigen Canvas-Events hinzufügt oder entfernt.

4.4.4 Bild kolorieren

Das bearbeitete Bild des Benutzers muss nun mithilfe des neuronalen Netzes koloriert werden.

Da das Kolorieren des Bildes einige Zeit in Anspruch nimmt, werden während dieses Vorgangs alle Interaktionen mit der Benutzeroberfläche deaktiviert. Um das aktuelle Bild zum Server zu schicken wird die Funktion `saveRealCanvas` aufgerufen und das ipcMain-Modul benachrichtigt, damit ein Ladebalken erscheint. Im ipcMain-Modul wird ein neues Fenster mit dem Ladebalken erzeugt, das für die Dauer der Kolorierung angezeigt wird.

```
$('#pixorize').on('click', function () {
    events.unbindevents();
    painter.delCanvasEventsonProgressbar();
    pixorize.saverealcanvas(canvas);
    ipcRenderer.send('show-progressbar');
});
```

Mithilfe der Methode `toDataURL`, wird eine Repräsentation des Canvas als `png/base64`-String zurückgegeben. Um das Bild vollständig und in Originalauflösung zu übermitteln wird ein neues Canvas, das für den Benutzer unsichtbar ist, in der Größe des Bildes erzeugt. Mit `drawOnfakeCanvas` werden die gesetzten Farbpunkte dem neuen Canvas hinzugefügt.

```
function saverealcanvas(canvas1) {
    let fakecanvas = document.createElement('canvas');
    //Invisible canvas is created...
    painter.drawOnfakeCanvas(contextfromfakecanvas);
    result = fakecanvas.toDataURL();
    goPython();
}
```

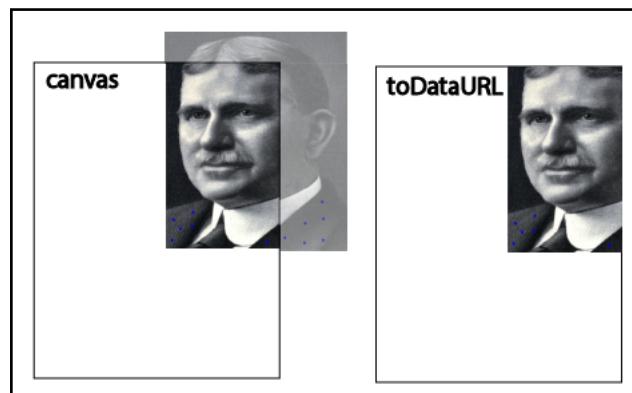


Abbildung 46: Ohne neu Zeichnen des Bildes

Der `base64`-String wird im `data`-Bereich des Ajax-Calls übermittelt. Wenn der Flask-Server das kolorierte Bild zurückliefert, wird dieses an das ipcMain-Modul weitergeleitet. Das Electron-Main-Modul sendet das Bild an das Präsentations-Fenster weiter. Der Ladebalken wird ausgeblendet und alle Canvas-Interaktionen werden wieder aktiviert.

```
function goPython() {
    $.ajax({
        url: "http://127.0.0.1:5000/_get_data/",
        type: "POST",
        data: {
            base64image: result
        },
        success: function (returnstring) {
            ipcRenderer.send('set-progressbar-completed');
            ipcRenderer.send('final-image', returnstring);
            //addAllCanvasEvents....
        }
    });
}
```

Der Flask-Server empfängt den `base64`-String und gibt diesen an die `run`-Funktion weiter. Die Funktion gibt das fertig kolorierte Bild, als `base64`-String, an den Ajax-Call zurück.

```
@app.route('/_get_data/', methods=['POST'])
def _get_data():
    base64image = request.form['base64image']
    string = run(base64image)
    return string
```

Die `run`-Funktion fügt das neuronale Netz und das Bild des Benutzers endgültig zusammen. Der `base64`-String wird in ein Numpy Array umgewandelt welches als `png`-Bild, in einer programminternen Ordnerstruktur gespeichert wird. Das neuronale Netz wird mit `colorize\image.main` aufgerufen. Es liest das gespeicherte Bild aus, koloriert dieses und speichert das fertige Bild wieder in einem internen Ordner ab. Das eingefärbte Bild wird wieder ausgelesen und in einen `base64`-String umgewandelt. Abschließend wird dieser an den Aufrufer zurückgegeben.

```
def run(base64image):
    image64 = base64.b64decode(base64image.partition(',') [2])
    image = Image.open(io.BytesIO(image64))
    image_np = np.array(image)

    Image.fromarray(image_np).save("./images_gui/input/default"
                                  +" .png")
    colorize_image.main()
```

```

with open("./images_gui/output/default.png", "rb") as img_file:
    img_data = img_file.read()
    img_str = base64.b64encode(img_data).decode('ascii')
    img_str = "data:image/png;base64," + img_str

return img_str

```

4.4.5 Bild anzeigen

Das kolorierte Bild wird in einem eigenen Fenster angezeigt, das nach beenden des Kolorierungsvorgangs aufpopt.

Dieses Fenster passt sich der Größe des Bildes an. Um jedoch zu verhindern, dass nicht das gesamte Bild am Bildschirm dargestellt werden kann (das Bild zu größer ist als der Bildschirm), wird das Scaling des Bildes bei Bedarf angepasst. Die Höhe des Bildschirms wird über `screen.height` ermittelt und es wird eruiert, ob die Höhe des Bildes kleiner ist als die des Bildschirm. Ist dies der Fall, wird die Höhe und die Breite des Bildes an das ipcMain-Modul weitergeleitet. Das Main-File ändert mit `presentationWindow.setSize` die Größe des Fenseters. Die Funktion `drawImage` wird aufgerufen, um das Bild mit den berechneten Größen zu zeichnen. Falls das Bild nicht in voller Größe am Bildschirm dargestellt werden kann, ist eine schrittweise Verminderung des Scalings notwendig, bis die maximal mögliche Darstellungsgröße erreicht ist.

```

function calculateWindowSize() {
    let windowHeight = screen.height;
    if (imageWindowHeight + 250 < windowHeight) {
        size = imageWindowWidth + ":" + imageWindowHeight;
        ipcRenderer.send('resize', size);
        drawImage(image);
    } else {
        while (imageWindowHeight + 250 > windowHeight) {
            scaleFirstDraw = scaleFirstDraw - 0.01;
            imageWindowHeight = imageWindowHeight
                * scaleFirstDraw;
            imageWindowWidth = imageWindowWidth
                * scaleFirstDraw;
        }
        size = imageWindowWidth + ":" + imageWindowHeight;
        ipcRenderer.send('resize', size);
        drawImage(image);
    }
}

```

Bild speichern Die Speicherung des kolorierten Bildes wird mithilfe von Electron, durch Aufrufen eines systemspezifischen Speicherdialogs, ermöglicht. Um das Bild zu speichern, wird das Main-Modul benachrichtigt um den Save-Dialog zu öffnen. Der Save-Dialog öffnet sich als eigenes Fenster. Der Benutzer gibt den Namen des Bildes und den Speicherort an. Der ausgewählte Pfad, mit angegebenen Speichernamen wird an das Präsentationsfenster zurückgeliefert.

```
ipcMain.on('save',(event =>
{
    dialog.showSaveDialog({
        properties: ['openDirectory'],
        filters:[{name: 'images_gui',extensions:['png']}]
    }).then((data) => {
        presentationWindow.send('saved-file',data.filePath.toString())
    });
});
```

Dieser Pfad und das aktuelle Bild wird durch einen Ajax-Call an den Flask-Server weitergeleitet. Das Bild wird über Python in den angegeben Pfad gespeichert. Ist der Speichervorgang abgeschlossen, wird mittels Electron eine systemspezifische Benachrichtigung gesendet. In dieser wird der Name des Bildes und eine Miniaturansicht davon angezeigt.

```
const notification = {
    title: 'Picture has been saved as',
    body: 'placeholder',
    icon: path.join(__dirname, '../assets/images/save.png')
};
```

Bild nachbearbeiten Python stellt einige Bibliotheken, die Bildbearbeitungsfunktionen anbieten, zur Verfügung. Durch diese Bibliotheken können z.B: Farbe, Kontrast oder Schatten automatisch verbessert werden. Um diese Funktionen in Pixorize zu integrieren, wird das Bild mittels eines Ajax-Calls dem Server übermittelt. Dieser speichert das Bild in einen Order und ruft den ausgewählten Bearbeitungsalgorithmus auf. Er liest das Bild aus dem gespeicherten Order aus und bearbeitet dieses. Das bearbeitete Bild wird zurückgeschickt und das alte Bild durch das neue ersetzt.

Abbildungsverzeichnis

1	Projektablaufplan	2
2	Use-Case-Diagramm	4
3	Wissenspyramide	6
4	Unstrukturierte und strukturierte Daten	7
5	Erzeugung von Information	8
6	Einordnung von ML in der Wissenspyramide	9
7	Schritte der Datenaufbereitung	10
8	Schritte nach der Datenaufbereitung	11
9	Zusammenhang: Daten und Bilder	13
10	RGB-Farbraum[55]	15
11	LAB-Farbraum[16]	16
12	Graustufenkonvertierung im RGB-Farbraum	17
13	Ausschnitte des Datensatzes[34]	19
14	Aufbau des Datensatzes[34]	21
15	Aufbau eines Tupels[34]	21
16	Grafische Darstellung eines künstlichen Neurons[15]	42
17	4-D Tensor[4]	44
18	3-D Filter[32]	45
19	Pooling Operation[32]	46
20	Unpooling Operation[32]	47
21	Grafische Darstellung der Netzwerkarchitekturen	48
22	Checkerboard Artifacts[5]	49
23	Verlauf des Fehlers	55
24	Vergleich zwischen der Konkurrenz (ColouriseSG) und Pixorize	56
25	Vollautomatisch koloriertes Porträt von Richard Feynman	56
26	Vollautomatische Kolorierung	57
27	Benutzerinteraktive Kolorierung mit blauen Farb-Punkten	57
28	Benutzerinteraktive Kolorierung mit grünen Farb-Punkten	57
29	Ladefenster	59
30	Bearbeitungsfenster	60
31	Progress Bar	60
32	Präsentationsfenster	61
33	Save-Benachrichtigung	61
34	Beispiel einer Desktop-Anwendung mit Photon	63
35	Synchrone und Asynchrone Datenübertragung	66
36	Main und Renderer Prozesse	67
37	Electron lifecycle	69
38	Kommunikation Client-Server	70
39	Abbildung aller Files	71
40	Ausgangs-Bild	74
41	Bild im Canvas	74
42	Function: getImageStartCords()	78
43	Relativ zum Canvas	81
44	Relativ zum Bild	81

45	Bewegung des Bildes	84
46	Ohne neu Zeichnen des Bildes	85

Tabellenverzeichnis

1	Meilensteine	1
2	Photonkit-Klassen	62
3	Auflistung der verwendeten Files, deren Module sowie Funktionen	73

Literatur

- [1] Adam Michael Wood. The wisdom hierarchy: From signals to artificial intelligence and beyond. <https://www.oreilly.com/content/the-wisdom-hierarchy-from-signals-to-artificial-intelligence-and-beyond/>. Eingesehen am 03.03.2020.
- [2] Tim Aschermann. Was ist C. [https://praxistipps\(chip.de/was-ist-css-einfach-erklaert_46818](https://praxistipps(chip.de/was-ist-css-einfach-erklaert_46818)). Eingesehen am 24.02.2020.
- [3] Tim Aschermann. Was ist HTML? [https://praxistipps\(chip.de/was-ist-html-verstaendlich-erklaert_40979](https://praxistipps(chip.de/was-ist-html-verstaendlich-erklaert_40979)). Eingesehen am 24.02.2020.
- [4] Chris Olah Augustus Odena, Vincent Dumoulin. A Beginner's Guide to CNNs. <https://pathmind.com/wiki/convolutional-network>. Eingesehen am 27.03.2020.
- [5] Chris Olah Augustus Odena, Vincent Dumoulin. Deconvolution and Checkerboard Artifacts. <https://distill.pub/2016/deconv-checkerboard/>. Eingesehen am 27.03.2020.
- [6] Chris Olah Augustus Odena, Vincent Dumoulin. Skip Connections Eliminate Singularities. <https://arxiv.org/abs/1701.09175>. Eingesehen am 27.03.2020.
- [7] Alex Clark. Pillow. <https://pillow.readthedocs.io/en/stable/>. Eingesehen am 26.02.2020.
- [8] ColouriseSG. ColouriseSG. <https://colourise.sg/>. Eingesehen am 27.03.2020.
- [9] Jimmy Ba Diederik P. Kingma. Adam: A Method for Stochastic Optimization. <https://arxiv.org/abs/1412.6980>. Eingesehen am 27.03.2020.
- [10] Electron. Electron. <https://www.electronjs.org/>. Eingesehen am 9.03.2020.
- [11] Facebook. PyTorch. <https://pytorch.org/>. Eingesehen am 27.03.2020.

- [12] The Python Software Foundation. os — Miscellaneous operating system interfaces. <https://docs.python.org/3/library/os.html>. Eingesehen am 26.02.2020.
- [13] Google. Tensorflow. <https://tensorflow.org/>. Eingesehen am 27.03.2020.
- [14] Jason Brown Lee. Tensoren. <https://machinelearningmastery.com/introduction-to-tensors-for-machine-learning/>. Eingesehen am 25.03.2020.
- [15] Jayesh Bapu Ahire. The Artificial Neural Networks Handbook. <https://medium.com/@jayeshbahire/the-artificial-neural-networks-handbook-part-4-d2087d1f583e>. Eingesehen am 27.03.2020.
- [16] kennerblick.net. Lab-Farbraum. <https://www.kennerblick.net/cielab.html>. Eingesehen am 26.02.2020.
- [17] Michael Matzer. Data preparation. <https://www.bigdata-insider.de/datenaufbereitung-ist-ein-unterschaetzer-prozess-a-803469/>. Eingesehen am 28.02.2020.
- [18] Tom McFarlin. Was ist jQuery? <https://code.tutsplus.com/de/tutorials/what-is-jquery--cms-26232>. Eingesehen am 3.03.2020.
- [19] MDN-Mitwirkenden. JavaScript Grundlagen. https://developer.mozilla.org/de/docs/Learn/Getting_started_with_the_web/JavaScript_basis. Eingesehen am 2.03.2020.
- [20] Michaela Tiedemann. Alles über Daten. <https://www.alexanderthamm.com/de/blog/alles ueber daten-grundlagen-formen-und-der-wert-der-daten/>. Eingesehen am 28.02.2020.
- [21] numpy.org. numpy. <https://numpy.org/>. Eingesehen am 26.02.2020.
- [22] Pallets. Quickstart - Flask Documentation. <https://flask.palletsprojects.com/en/1.1.x/quickstart/>. Eingesehen am 30.03.2020.
- [23] pypi.org. opencv-python. <https://pypi.org/project/opencv-python/>. Eingesehen am 26.02.2020.
- [24] python.org. os. <https://docs.python.org/3/library/os.html>. Eingesehen am 26.02.2020.
- [25] python.org. random. <https://docs.python.org/3/library/random.html>. Eingesehen am 26.02.2020.
- [26] pytorch.org. torchvision. <https://pytorch.org/docs/stable/torchvision/index.html>. Eingesehen am 26.02.2020.

- [27] Raffael. Wissenspyramide. <https://derwirtschaftsinformatiker.de/2012/09/12/it-management/wissenspyramide-wiki/>. Eingesehen am 26.02.2020.
- [28] scikit image.org. scikit-image. <https://scikit-image.org/docs/dev/api/skimage.util.html>. Eingesehen am 26.02.2020.
- [29] Connor Sears. Photon. <http://photonkit.com/>. Eingesehen am 28.02.2020.
- [30] Shana Pearlman. Was ist Datenaufbereitung? <https://de.talend.com/resources/what-is-data-preparation/>. Eingesehen am 26.02.2020.
- [31] Sik-Ho Tsang. Fully Convolutional Neural Networks. <https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1>. Eingesehen am 25.03.2020.
- [32] Sumit Saha. A Comprehensive Guide to CNNs. <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. Eingesehen am 27.03.2020.
- [33] The Matplotlib development team. Matplotlib. <https://matplotlib.org>. Eingesehen am 27.03.2020.
- [34] GitHub tkarras. ffhq-dataset. <https://github.com/NVlabs/ffhq-dataset>. Eingesehen am 26.02.2020.
- [35] Wikipedia. Ajax(Programmierung). [https://de.wikipedia.org/wiki/Ajax_\(Programmierung\)](https://de.wikipedia.org/wiki/Ajax_(Programmierung)). Eingesehen am 5.03.2020.
- [36] Wikipedia. Aktivierungsfunktionen. https://en.wikipedia.org/wiki/Activation_function. Eingesehen am 25.03.2020.
- [37] Wikipedia. Artificial Intelligence. https://en.wikipedia.org/wiki/Artificial_intelligence. Eingesehen am 25.03.2020.
- [38] Wikipedia. Bildverarbeitung. <https://de.wikipedia.org/wiki/Bildverarbeitung>. Eingesehen am 26.02.2020.
- [39] Wikipedia. Computer Vision. https://en.wikipedia.org/wiki/Computer_vision. Eingesehen am 26.02.2020.
- [40] Wikipedia. Computergrafik. <https://de.wikipedia.org/wiki/Computergrafik>. Eingesehen am 26.02.2020.
- [41] Wikipedia. Convolutional Neural Network. https://de.wikipedia.org/wiki/Convolutional_Neural_Network. Eingesehen am 25.03.2020.
- [42] Wikipedia. CUDA. <https://de.wikipedia.org/wiki/CUDA>. Eingesehen am 27.03.2020.

- [43] Wikipedia. Data preparation. https://en.wikipedia.org/wiki/Data_preparation. Eingesehen am 28.02.2020.
- [44] Wikipedia. Deep Learning. https://de.wikipedia.org/wiki/Deep_Learning. Eingesehen am 25.03.2020.
- [45] Wikipedia. Farbe. <https://de.wikipedia.org/wiki/Farbe#Farbmodelle>. Eingesehen am 26.02.2020.
- [46] Wikipedia. Farbsymbolik. <https://de.wikipedia.org/wiki/Farbsymbolik>. Eingesehen am 26.02.2020.
- [47] Wikipedia. Farbsymbolik. <https://de.wikipedia.org/wiki/Farbraum>. Eingesehen am 26.02.2020.
- [48] Wikipedia. Gewichtung. <https://de.wikipedia.org/wiki/Gewichtung>. Eingesehen am 25.03.2020.
- [49] Wikipedia. Grau. <https://de.wikipedia.org/wiki/Grau>. Eingesehen am 26.02.2020.
- [50] Wikipedia. Künstliches Neuron. https://en.wikipedia.org/wiki/Artificial_neuron. Eingesehen am 25.03.2020.
- [51] Wikipedia. Lab-Farbraum. <https://de.wikipedia.org/wiki/Lab-Farbraum>. Eingesehen am 26.02.2020.
- [52] Wikipedia. Machine Learning. https://de.wikipedia.org/wiki/Maschinelles_Lernen. Eingesehen am 25.03.2020.
- [53] Wikipedia. Mittlere Quadratische Abweichung. https://de.wikipedia.org/wiki/Mittlere_quadratische_Abweichung. Eingesehen am 10.04.2020.
- [54] Wikipedia. Relu. [https://de.wikipedia.org/wiki/Rectifier_\(neuronale_Netzwerke\)](https://de.wikipedia.org/wiki/Rectifier_(neuronale_Netzwerke)). Eingesehen am 25.03.2020.
- [55] Wikipedia. Rgb-Farbraum. <https://de.wikipedia.org/wiki/RGB-Farbraum>. Eingesehen am 26.02.2020.
- [56] Wikipedia. Supervised Learning. https://en.wikipedia.org/wiki/Supervised_learning. Eingesehen am 25.03.2020.
- [57] Wikipedia. Tangens hyperbolicus. https://de.wikipedia.org/wiki/Tangens_hyperbolicus_und_Kotangens_hyperbolicus. Eingesehen am 25.03.2020.
- [58] Wikipedia. Unsupervised Learning. https://en.wikipedia.org/wiki/Unsupervised_learning. Eingesehen am 25.03.2020.
- [59] Wikipedia. Visual Computing. https://de.wikipedia.org/wiki/Visual_Computing. Eingesehen am 26.02.2020.

Diplomarbeit Informationstechnologie
2019/2020

HTL Wels
Fischergasse 30
Wels 4600