Entwurf und Evaluation einer Schicht zur Kontrastnormalisierung in Convolutional Neural Networks

Design and Evaluation of a contrast normalizing Layer for Convolutional Neural Networks

MASTERARBEIT

eingereicht am
Masterstudiengang

Systems Engineering

an der
Universität Bremen

am 9. November 2021

Stephan Hopfmüller
Matrikelnummer: 4136404

Erstprüfer: Prof. Dr. Ing. Udo Frese
Zweitprüfer: Dr. Felix Putze

Inhaltsverzeichnis

Kurzfassung

Objekterkennung mittles Convolutional Neural Networks kann bereits geometrisch unregelmäßige Objekte erkennen. Jetzige Ansätze haben immer noch Probleme mit kontrastreichen Bildern, die vergleichsweise kontrastarme Objekte enthalten. Der unmodifizierte U-Net- / ResNet-Ansatz, welcher hier als Vergleichs-Netz verwendet wird, nutzt keine Kontrastnormalisierung. Um zu überprüfen, ob das neuronale Netz durch eine Kontrastnormalisierungslayer verbessert wird, wurde die besagte Ebene implementiert, um zu probieren, die Objekterkennung in schwierigen Beleuchtungsverhältnissen zu verbessern. Ich generiere und verwende einen eigenen Datensatz, da bereits existierende Datensätze nicht für das Erkennen von schwierigen Kontrastverhältnissen optimiert sind. Mithilfe von weichem Licht werden kontrastarme Bilder generiert, während mit hartem Licht kontrastreiche Bilder generiert werden. ResNet wird dann mit der Kontrastnormalisierungsebene erweitert und in Kombination mit U-Net zum Testen verwendet. Der Ansatz der in dieser Arbeit genutzt wurde war das trainieren des Netzes mit kontrastarmen Bildern. Nachdem beide Netzwerke trainiert sind, werden sie auf dem kontrastreichen Datensatz getestet, um zu sehen, wie die Netze den anderen Datensatz ohne spezifisches Training handhaben. Falls die Kontrastnormalisierung so funtioniert wie vorgesehen, sollte das modifizierte Netz eine bessere Leistung haben als das unmodifizierte Netz. Außerdem sollte das modifizierte Netz genauso gut wie das unmodifizierte Netz auf dem Trainingsdatensatz funktionieren. Leider zeigt das U-Net mit dem modifizierten ResNet keine Verbesserung gegenüber dem unmodifizierten ResNet. Dies zeigt, dass moderne neuronale Netze wahrscheinlich keine Kontrastnormalisierung brauchen, um zu funktionieren.

Abstract

Object detection using Convolutional Neural Networks already enables the detection of geometrically irregular objects. Current approaches still struggle with high contrast images which contain comparatively low contrast objects. The unmodified U-Net with ResNet approach, used here as a comparison network, lacks a contrast normalization. To check whether the neural network improves with a contrast normalization layer, said layer has been implemented here to try and improve object detection in difficult lighting conditions. I create and use a custom generated training data set since existing training data sets are not optimized for difficult contrast situations. Using soft lighting a set of low contrast images are created, similarly using hard lighting high contrast images are created. ResNet is then expanded with a contrast normalization layer and further used in conjunction with U-Net to test. The approach used here is training the network using low contrast images. After both networks are trained they are fed a custom generated high contrast data set to see how both neural networks cope with the other data set without specific training. If the contrast normalization works as intended the modified network should perform better than the unmodified network. Also the modified network should perform as well as the unmodified network on the training data set. Unfortunately the U-Net with the modified ResNet shows no improvement over the unmodified ResNet configuration. This suggests that modern neural networks probably do not require contrast normalization to work.

Kapitel 1

Einleitung

1.1 Motivation

Neuronale Netze ermöglichen Bildverarbeitung in Fällen, die mit klassischen Mitteln nur schwer möglich sind. Dazu zählt beispielsweise die zuverlässige Erkennung von nicht stark geometrischen Objekten sowie die Verarbeitung von Bildern mit natürlichen, detailreichen Hintergründen, sodass die Möglichkeit der Datenverarbeitung durch ihre Nutzung stark anwächst und oft kaum Vorverarbeitung nötig ist. Ebenso können mit neuronalen Netzen auch die Anforderungen für die Nutzung von Bildverarbeitung in Produktionsumgebungen gesenkt werden, sodass diese leichter angewandt werden und somit auch mehr Überwachung in zunehmend autonom funktionierender Produktion erfolgen kann. Schwierigkeiten gibt es bisher unter Anderem noch mit der Verarbeitung von kontrastreichen Bildern, bei denen interessante Bereiche, wie zu erkennende Objekte, nur schwachen Kontrast zur Umgebung und in sich aufweisen.

1.2 Problemstellung

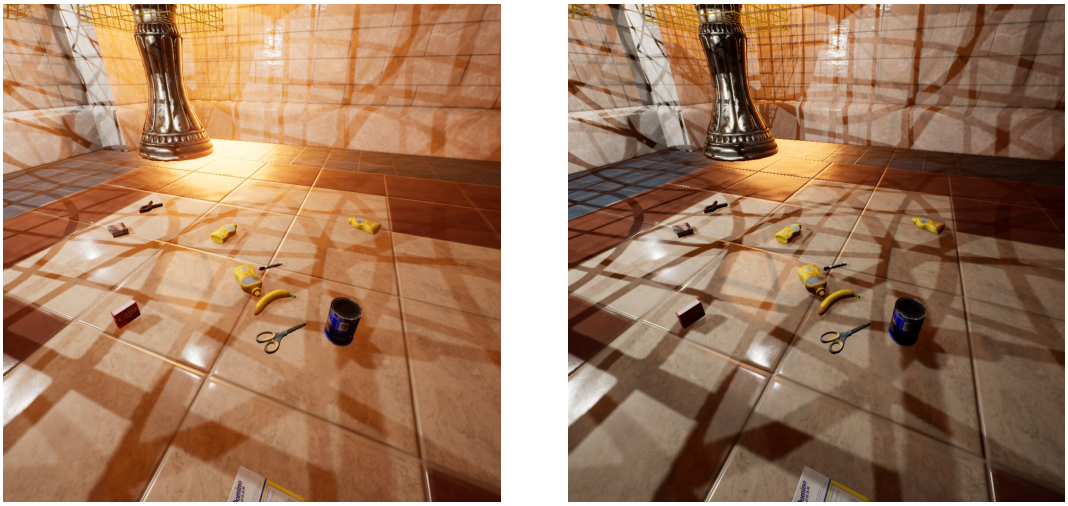
Um Objekterkennung von neuronalen Netzen im allgemeinen Fall zu verbessern, müsste unter anderem besser mit unterschiedlichen Beleuchtungssituationen in einem jeweiligen Bild umgegangen werden können - diese führen zu starken Kontrasten, welche die Erkennung von Objekten stören können. Dabei existieren verschiedene Ansätze zur Lösung dieses Problems, wie die Vorverarbeitung von Bildern, um Kontraste etwas zu normalisieren und somit die Verarbeitung mit existierenden Methoden und Netzen zu verbessern. Zu Anfangszeiten des Aufschwungs neuronaler Netze gab es Elemente in den verbreiteten neuronalen Netzen, welche in begrenztem Rahmen dazu fähig sind, Kontraste zu normalisieren und damit den Umgang mit Bildern mit schwieriger Beleuchtung zu verbessern. Bei den von mir betrachteten aktuellen, erfolgreichen Netzen sind jedoch keine derartigen Methoden implementiert, daher ist mein Ziel in dieser Arbeit, ein modernes neuronales Netz mit einer Schicht zur Kontrastnormalisierung zu verbinden, um die guten Ergebnisse, soweit möglich, auch auf Bilder mit schwierigerer Beleuchtung zu übertragen.

1.3 Zielsetzung

Gängige Feature Detectors wie HOG[7], SIFT[25] und SURF[2] enthalten eine Kontrastnormalisierung. Moderne Netzarchitekturen, wie beispielsweise ResNet, DenseNet und MobileNet, scheinen keine Kontrastinvarianz mehr zu enthalten [6] [23] und die in AlexNet beinhaltete Kontrastnormalisierung ist recht stark eingeschränkt. Aktuell erreichen Convolutional Neural Networks gute Ergebnisse, daher versuche ich, exemplarisch eine moderne Netzarchitektur in Form eines Convolutional Neural Networks um eine lokale Kontrastinvarianz zu ergänzen. Da übliche neuronale Netze aus einer Verkettung aus mehreren Schichten bestehen, ist es mein Ziel die Ergänzung in Form einer eigenen Schicht umzusetzen, sodass die Möglichkeit der Verwendung in anderen Netzen offen steht. Meine Idee ist dabei, dass die Schicht dem Netz im Lernprozess mehr Freiheiten zur Anpassung in Bezug auf Kontraste gibt, als dies in AlexNet der Fall ist. Den Entwurf führe ich auf Basis einer alternativen Betrachtung von Kontrasten durch, wie sie in [13] beschrieben ist. Um eine Aussage über eine mögliche Verbesserung der Bildverarbeitung durch die neue Schicht treffen zu können, führe ich anschließend anhand des praktischen Anwendungsfalls der semantischen Segmentierung von Objekten ein Vergleich des neuronalen Netzes mit und ohne diese Schicht durch. Dabei liegt mein Fokus auf dem Anwendungsfall der Semantic Segmentation, da ich bei diesem durch die lokale Klassifizierung von Bildern am ehesten einen Vorteil erwarte.

1.4 Vorgehen

Um das Netz optimal trainieren sowie am Ende eine sinnvolle Aussage über die Leistungsfähigkeit treffen zu können, generiere ich als ersten Teil der praktischen Umsetzung einen Datensatz für Semantic Segmentation, der jeweils ein kontrastreiches und ein kontrastarmes Bild für jeden Szeneaufbau enthält. Anschließend entwickle ich die Schicht für die lokale Kontrastnormalisierung. Darauf folgend ist es mein Ziel, die Performanz eines Convolutional Neural Networks durch Experimente mit und ohne diese Schicht zu evaluieren. Infolgedessen beurteile ich abschließend darauf aufbauend den praktischen Nutzen der neuen Schicht, soweit möglich.



(a) Weiche Beleuchtung für weiche Kontraste

(b) Harte Beleuchtung für harte Kontraste

Abbildung 1.1: Der Datensatz für das Training zeigt Objekte in zwei Beleuchtungseinstellungen je Konstellation

Kapitel 2

Grundlagen

Datenaufteilung

Die Trainingsdaten werden in drei Teile aufgeteilt. Der Trainingsdatensatz ist der Teil, auf dem das Netz trainiert wird. Der Validierungsdatensatz ist der Teil, mit dem während des Trainings durch frische Daten überprüft wird, ob die Parameter gut gewählt sind. Der Testdatensatz ist der Teil, mit dem auf eine Änderung zwischen dem geänderten und dem ursprünglichen Netz getestet wird.

Tensorflow

Tensorflow ist eine ursprünglich von Google entwickelte quelloffene Bibliothek für das Trainieren von neuronalen Netzen.

Keras

Keras ist eine Python Schnittstelle zu Tensorflow.

ImageNet

ImageNet ist ein Datensatz für das trainieren von Neuronalen Netzen.

ResNet

ResNet steht für Residual Neural Network. Dies ist eine Kategorie von neuronalen Netzen, welche Verbindungen von alten Ebenen zu neuen Ebenen hat. Dadurch werden die namensgebenden Residuen, die vorherige Ebene, für das Training verwendet.

Kontrastnormalisierung

Kontrastnormalisierung ist eine Bildbearbeitungstechnik, in der die Enden des Histogrammes beschnitten und anschließend wieder expandiert werden, um die Kontrastverhältnisse zu verbessern.

Kapitel 3

Stand der Technik

Es gibt bereits diverse Ansätze dazu, die Verarbeitung von kontrastreichen Bildern mit neuronalen Netzen zu verbessern. Diese betrachte ich hier hier aufgeteilt in die Vorverarbeitung der zu verarbeitenden Bildern und Anpassungen an den neuronalen Netzen an sich.

3.1 Vorverarbeitung

Zum Einen können die Bilder vorverarbeitet werden, sodass die Kontraste vor der Verarbeitung durch die neuronalen Netze normalisiert werden, dazu zählt beispielsweise die Kontrastnormalisierung über einen Histogrammausgleich. Durch die Verzerrung des Histogramms kann die Erkennung allerdings auch verschlechtert werden, beispielsweise können bei seitlicher Beleuchtung unnatürlich scharfe Kontraste entstehen [27]. Nutzt man eine logarithmische Betrachtung der Gradienten, kann dieses Problem verringert werden, es verbleibt jedoch, dass die Kanten des Eigenschattens eines Objekts leicht als Kanten des Objekts selbst fehlinterpretiert werden können [37].

3.1.1 Anpassung der Netze

Neben der Vorbearbeitung gibt es in der klassischen Bildverarbeitung auch Feature Detectors (Algorithmen zur Extraktion von Merkmalen), die eine Kontrastnormalisierung beinhalten und damit relativ unabhängig von Intensitäten von Kontrasten mit diesen umgehen können. Dazu zählen „Histogram of Oriented Gradients“[7] (HOG), „Scale-Invariant Feature Transform“[25] (SIFT) und „Speeded Up Robust Features“[2] (SURF). Beide Ansätze geben die Normalisierung fest kodiert vor, der Vorteil von neuronalen Netzen ist im Allgemeinen jedoch ihre Anpassungsfähigkeit an natürliche Problemstellungen, daher ist für mich vorstellbar, dass eine lokale Kontrastnormalisierung im Netz selbst besser funktioniert. Da die oben erwähnten Feature Detectors verbreitet Anwendung finden, steht dazu im Kontrast bei neuronalen Netzen eine scheinbare Abwesenheit von Kontrastnormalisierung. Im neuronalen Netz AlexNet ist mit „Local Response Normalization“ eine Kontrastnormalisierung über ein Intervall von benachbarten Kanälen enthalten, damit ist diese prinzipiell im Lernprozess veränderbar, jedoch ist über die Beschränkung auf die benachbarten Intervalle die Anpassungsfähigkeit stark eingeschränkt [1]. In darauf folgenden, noch erfolgreicheren Netzen wie MobileNet, Res-

Net und DenseNet findet keine Kontrastnormalisierung mehr statt.

3.2 Auswirkung auf die Zielsetzung

Gängige Feature Detectors wie HOG, SIFT und SURF enthalten eine Art Kontrastnormalisierung. Dabei liegt ein breites Anwendungsspektrum für diese Algorithmen vor, beispielsweise Objekterkennung, die Verbindung von mehreren Fotos zu einem Panorama oder in der Fotogrammetrie zu einem 3D-Objekt.
Moderne Netzarchitekturen, wie beispielsweise die Convolutional Neural Networks ResNet, DenseNet und MobileNet, scheinen keine Kontrastinvarianz mehr zu enthalten [6] [23] und die in AlexNet beinhaltete Kontrastnormalisierung ist recht stark eingeschränkt.

1989)

Aktuell erreichen Convolutional Neural Networks gute Ergebnisse,wie beispielsweise an den oberen Platzierungen der letzten Jahre in der „ImageNet Large Scale Visual Recognition Challenge“ (ILSVRC) zu sehen ist. Mein Ansatz ist es, mit der lokalen Kontrastnormalisierung eine von mir als einen Kernaspekt der Feature Detectors der klassischen Bildverarbeitung erachtete Komponente mit einem neuronalen Netz zu kombinieren. Dabei halte ich es für denkbar, mit dem resultierenden Convolutional Neural Network bei einem Test auf einen Datensatz mit harten Kontrasten anschließend bessere Ergebnisse erhalten zu können als ohne meine Erweiterung.

Kapitel 4

Datensatzerzeugung

4.1 Motivation

Die neue Schicht soll Kontraste etwas normalisieren, um den Umgang des neuronalen Netzes mit harten Kontrasten zu verbessern. Damit ich eine qualitative Aussage über das Ergebnis machen kann, benötige ich einen Datensatz, mit dem ich die Leistung des Netzes unter verschiedenen Kontrastbedingungen testen kann. Existierende Datensätze wie beispielsweise MS COCO[24] oder Falling Things [34] haben andere Schwerpunkte und erfüllen damit diese Anforderungen nicht. Daraus ergibt sich als Zwischenziel die Erzeugung eines Programms, mit dem ich einen Datensatz synthetisieren kann, welcher verschiedene Beleuchtungseinstellungen und damit Stärken an Kontrasten enthält. Hierbei plane ich ein, die einzelnen Bilder des Datensatzes mit verschiedene Kontraststärken zu erzeugen, sodass bei der Verwendung zwischen diesen ausgewählt werden kann.
Zum Testen der Schicht im Netz im Anwendungsfall der Semantic Segmentation mit Supervised Learning wird dabei als Ground Truth (Musterlösung) ein Bild mit pixelweiser Klassifizierung benötigt. Dadurch kann im Trainingsprozess durch den Vergleich der Predicted Truth (Vorhersage des Netzes) mit der Ground Truth über Backpropagation das Netz trainiert werden.

Inspiriert vom Datensatz „Falling Things“ [34] ist es mein Ziel in diesem Abschnitt, einen ähnlichen Datensatz mit dem „NVidia Deep learning Dataset Synthesizer“[33] (NDDS) zu konstruieren. Eine synthetische Erzeugung eines Datensatzes hat den Vorteil, dass eine große Menge an Daten mit klar bestimmbaren Rahmenbedingungen erzeugt werden kann. Andererseits wäre ein Datensatz auf Basis echter Bilder im Rahmen dieser Arbeit nur sehr schwer umzusetzen, da explizit verschiedene Beleuchtungen je Szene benötigt werden und gleichzeitig die Varianz in den Daten zum Training eines kompletten Netzes groß genug sein muss. Die Zusammenstellung einer entsprechend großen Menge an existierenden Bildern mit jeweils verschiedenen Beleuchtungen derselben Szene erscheint mir dabei genauso unrealistisch wie die Aufnahme eigener Bilder mit entsprechenden Beleuchtungsbedingungen, wenn diese eine genügend große Varianz aufweisen sollen. Des Weiteren wird für die Semantic Segmentation eine pixelweise Klassifizierung der Bilder benötigt, welche einen darüber hinaus gehenden großen manuellen Arbeitsaufwand bedeuten würde und deutlich ungenauere Ergebnisse liefern dürfte. Da das Ziel dieser Arbeit die grundlegende Beurteilung meiner Schicht beinhaltet, jedoch

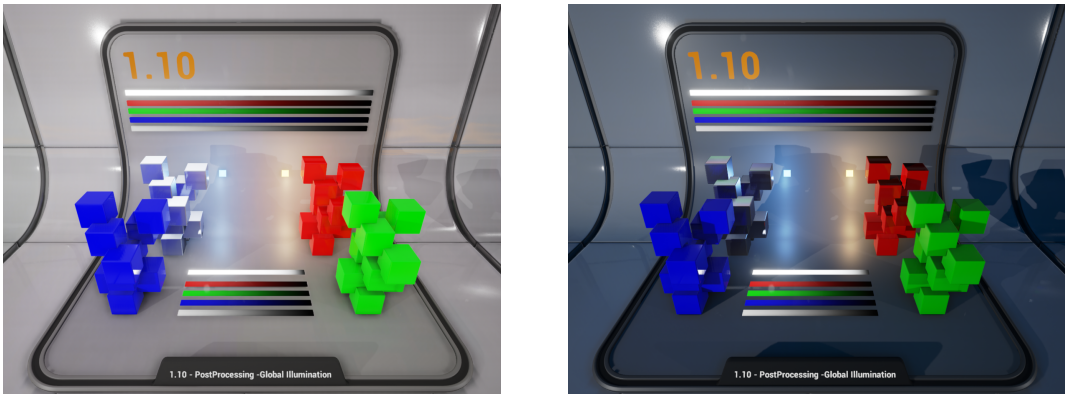


Abbildung 4.1: Beispiel der mit Global Illumination als Post-Processing-Effekt möglichen Resultate aus der Dokumentation der Unreal Engine [14]

keine direkte Anwendbarkeit auf echte Bilder vorgesehen ist, erzeuge ich meinen Datensatz daher synthetisch. Dabei ist davon auszugehen, dass der Grad des möglichen Fotorealismus mit der für NDDS zu verwendenden Unreal Engine[11] weiter fortschreiten wird. Dementsprechend wird der Nachteil gegenüber echten Bildern bei geplanter Anwendung eines damit trainierten Netzes auf echte Bilder in Zukunft voraussichtlich weiter kleiner werden.
Da NDDS zwar die dafür nötigen Werkzeuge an sich für die Erstellung eines Datensatzes wie Falling Things bereitstellt, jedoch nicht direkt den Aufbau mitliefert, ist es mit Teil dieser Arbeit, einen ähnlichen Aufbau zu erzeugen. Dazu modifiziere ich NDDS entsprechend, sodass die jeweilige Szene mit zwei verschiedenen Beleuchtungen aufgenommen wird und ein Ergebnis mit unterschiedlich starken Kontrasten vorliegt.

Wie unter Abbildung 4.1 erkenntlich ist, kann mit verschiedenen Stärken der Global Illumination effektiv verschieden starke Kontraste erzeugt werden. Damit ist diese Rendering-Option für das gewünschte Resultat geeignet und ich verwende sie mit unterschiedlichen Stärken zur Erzeugung der sich in den Kontrasten unterscheidenden Bildern.

Als Hintergrundumgebung verwende ich das Beispiellevel „Sun Temple“[15] der Unreal Engine. Dies ist eines der im Falling-Things-Datensatz verwendeten Levels und gestattet freie Verwendung unter Creative Commons CC BY-NC-SA 4.0 mit dem Ziel, die grafischen Fähigkeiten der Unreal Engine für den Einsatz in Anwendungen auf mobilen Endgeräten zu demonstrieren. Dementsprechend betrachte ich es als gut geeignet, um damit einen Datensatz mit dem Schwerpunkt auf Kontrasten und damit der Beleuchtung zu erzeugen. Als Zielobjekte für die Semantic Segmentation des Netzes verwende ich das gleiche Subset des YCB-Datensatzes[4] [1], wie es in Falling Things verwendet wird.

[1] zum Download erhältlich unter http://www.ycbbenchmarks.com/object-models

4.2 Aufbau der Umgebung

Ähnlich zum Falling-Things-Datensatz ist mein grundlegendes Ziel hier, in einem kleinen Bereich Objekte erscheinen und fallen zu lassen, sodass sie in einer zufälligen Anordnung neben- und übereinander vorliegen, wodurch sie in Bezug auf die Beleuchtung Einfluss aufeinander nehmen. Anschließend soll eine Kamera die Objekte aufnehmen und dabei verschiedene Arten von Bildern erzeugen: Sowohl solche, die das normal Sichtbare abbilden, als auch unter anderem Bilder einer Class Segmentation, in denen nur die Objekte abhängig von ihrer Objektart in unterschiedlichen Farben angezeigt werden. Dabei möchte ich das normale Bild aufteilen in zwei verschiedene mit unterschiedlich starker Global Illumination.

Da der Sun Temple ein Gebäude mit detaillierter Innenarchitektur ist und die Objekte sowie die Kamera im freien Raum platziert werden müssen, stecke ich Teile des Levels ab, wie unter der Abbildung 4.2 zu sehen ist. Diese abgesteckten Bereiche verwende ich als die Regionen, in denen die Objekte und die Kamera jeweils platziert werden können. Dabei liegt mein Fokus auf den Teilen des Levels, die besonders abwechslungsreiche Beleuchtung beinhalten.

Abbildung 4.2: Abgesteckte Bereiche im Sun Temple (gelb)

Um mehr Schlagschatten für weitere harte Kontraste zu erzeugen, welche insbesondere auch die Objekte selbst schneiden, modifiziere ich das Level derartig, dass vor allen Beleuchtungsquellen sich eine Art simples Gitter befindet. Dies beinhaltet dabei flache Gitter vor den Fenstern und würfelförmige um die Feuerschalen.

Weitere Details finden sich in Kapitel 6.1.1

4.3 Ablauf

- detaillierter im entsprechenden Unterkapitel in „Implementierung“

4.3.1 Dynamischer Szeneaufbau

Für jede Szene wird einer der abgesteckten Bereiche zufällig ausgewählt, gewichtet über die Grundfläche. Eine Kamera wird auf einen Punkt im ausgewählten Bereich gerichtet und dabei zufällig um diesen platziert. Die Umgebung dieses Punktes wird mit dynamisch platzierten, mit Gittern verdeckten Lichtquellen zusätzlich ausgeleuchtet. Anschließend erscheinen Objekte über dem Punkt und fallen herunter.

Weitere Details finden sich in Kapitel 6.1.1

4.3.2 Aufnahme der Bilder

Auslöser für die Aufnahme der Bilder ist ein Stillstand der Objekte oder ein Überschreiten der maximalen Szenenlaufzeit. Die Szene wird anschließend in sechs Variationen aufgenommen, diese sind:

- Ein unmodifiziertes Bild
- Ein Tiefenbild
- Eine Class Segmentation
- Eine Instance Segmentation
- Die Szene mit hoher Global Illumination
- Die Szene mit niedriger Global Illumination

Eine hohe Global Illumination sorgt dabei für weiche Beleuchtung und damit niedrige Kontraste. Umgekehrt sorgt eine niedrige Global Illumination für eine harte Beleuchtung und damit hohe Kontraste.

Weitere Details finden sich in Kapitel 6.1.1

4.3.3 Beispiele

Die Abbildung 4.3 und 4.4 zeigen exemplarische Ausschnitte des Datensatzes, den ich wie hier beschrieben erzeuge.

Abbildung 4.3: Beispiele des erzeugten Datensatzes, links jeweils mit hoher und rechts mit niedriger Global Illumination

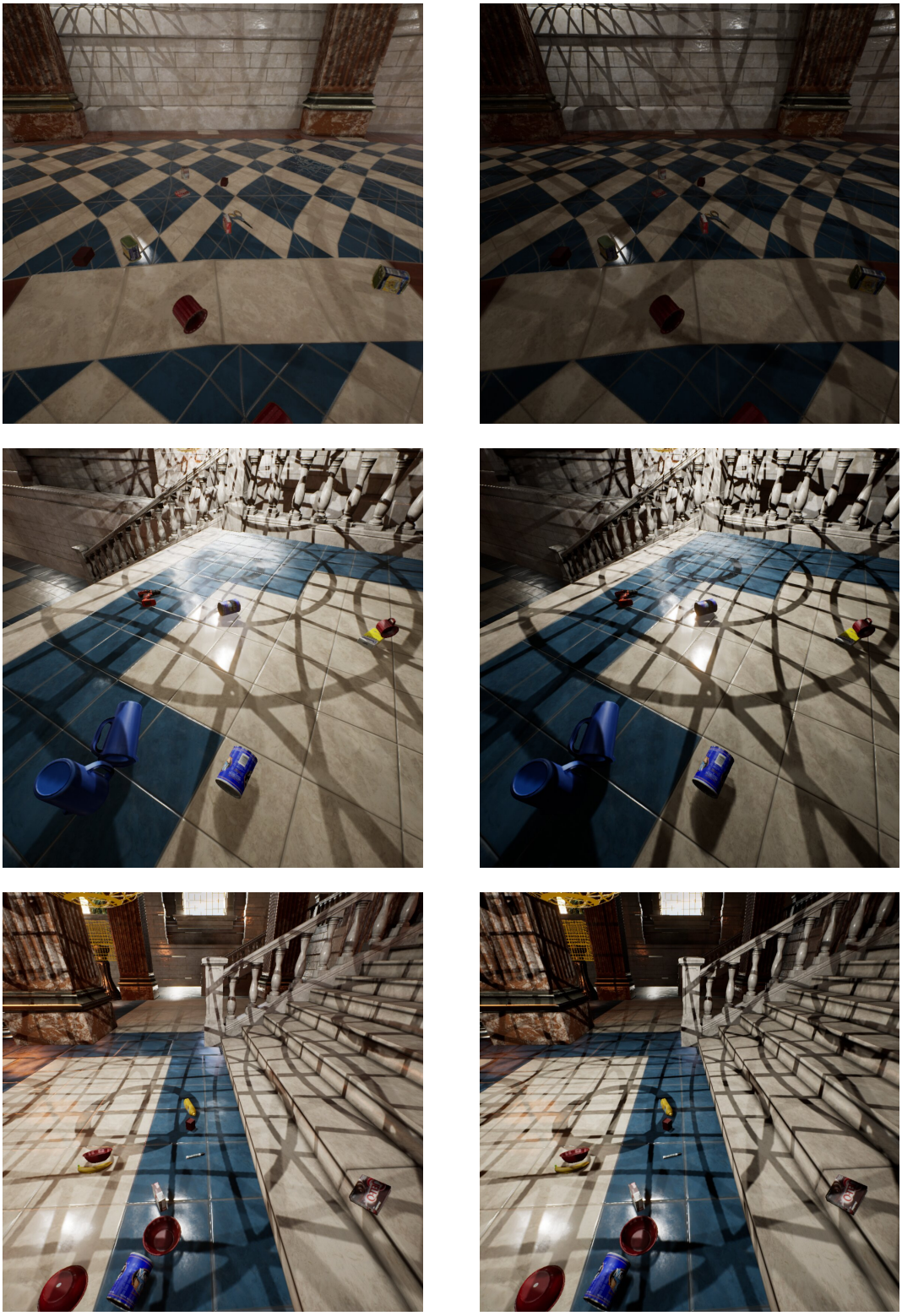


Abbildung 4.4: Weitere Beispiele des erzeugten Datensatzes, links jeweils mit hoher und rechts mit niedriger Global Illumination

Kapitel 5

Algorithmus, Methodik, Ansatz

5.1 Allgemeiner Ansatz

Ausgangspunkt der Idee für den Algorithmus meiner eigenen Schicht ist eine kontrastnormalisierte Gradientenerkennung zur Kreisdetektion [13]. Dabei übernehme ich Kernaspekte aus dem mathematischen Beweis und modelliere diese analog in meiner Schicht, um kontrastnormalisierte Gradienten über alle Kanäle des jeweiligen Eingabebilds erhalten zu können. Für den Anwendungsfall der Semantic Segmentation bietet sich eine Encoder-Decoder-Struktur an, die sich zentral durch Reduktion der Auflösung bei gleichzeitiger Erhöhung der Kanäle im Encoder und einer Erhöhung der Auflösung im Decoder auszeichnet. Dazu verwende ich das U-Net[31] in Kombination mit einem ResNet50 als Encoder, da ich ResNet als eine Art Standard für aktuelle neuronale Netze empfinde und mich bereits damit auseinandergesetzt habe. Mein Ziel ist hier, durch Einbau meiner eigenen Schicht in das ResNet im Encoder ein Beispielnetz zu schaffen, um zu evaluieren, ob dieses Bilder mit harter Beleuchtung besser verarbeiten kann als das Ausgangsnetz. Dabei ist mein Ansatz, eine der bestehenden Schichten durch meine eigene zu ersetzen, um die grundlegende Struktur beizubehalten. Um die beiden Netze vergleichen zu können, verwende ich einen eigenen Datensatz, den ich wie unter 4 beschrieben erzeuge. Dieses Kapitel befasst sich dabei mit dem Algorithmus meiner Schicht, Details zur Implementation befinden sich in Kapitel 6.2 und die Experimente um das Training und die Evaluation des Netzes befinden sich in Kapitel 7.

5.1.1 Anwendung auf einen Kanal

Im Folgenden nehme ich einen Mittelwert der Eingabedaten von 0 an. In einem Netz kann er beispielsweise durch eine Batch Normalization vor dieser Schicht erreicht werden, da die Daten dadurch auf einen Mittelwert von 0 und einer Varianz von 1 skaliert werden.

Analog zum kontrastnormalisierten Gradientenbild mit einem Kanal unter [13, (22)] und für mehrere Kanäle unter [13, (44)] erstelle ich eine kontrastnormalisierende Schicht. Dazu ergänze ich eine zweidimensionale Convolution um einen Divisor.

Beginnend mit der Betrachtung eines Bild mit einem Kanal gehe ich aus von einer Art gewichteten Varianz des Bildes, wie sie in [13, 4.3 Interpretation] für den Divisor

beschrieben ist. Dabei steht X für das Eingabebild:

$$X \cdot X - (X \cdot I)^2 \tag{5.1}$$

Dem Beweis folgend ergibt sich über die Substitution von $X \cdot I$ mit $X * w$ und $X \cdot X$ mit $X^2 * w$ in [13, 4.4 Contrast Normalized Gradient Vector Image] eine lokale Betrachtung in Form des kontrastnormalisierenden Gradienten cng in [13, (22)][1].

$$cng = \frac{\begin{pmatrix} X * (-\eta x w) \\ X * (-\eta y w) \end{pmatrix}}{\sqrt{X^2 * w' - (X * w')^2 + \sigma^2}} \tag{5.2}$$

Diesen übertrage ich in eine Schicht für ein Convolutional Neural Network: Der Zähler kann bei Zusammenfassung von $-\eta x w$ zu einem neuen w als eine Convolution[2] (Faltung) mit dem Bild aufgefasst werden, wie sie als zentrale Elemente von Convolutional Neural Networks verwendet werden.

Die Gleichung 5.2 setzt voraus, dass die Gewichte w' auf Summe 1 normalisiert sind. In der Übertragung soll die Summe der Gewichte beliebig sein und in Verbindung mit dem σ^2 ausdrücken, wie stark die Kontrastnormalisierung aktiv sein soll. Deshalb ist am 1. Summanden ein zusätzlicher Term $\sum_{di,dj,k} w'_{di,dj,k,q}$ nötig.

Die Terme im Nenner lassen sich dann ebenso übertragen, wenn man w' als einen Filter betrachtet. Dabei stelle ich $(X * w')^2$ als eine Convolution da und ebenso $X^2 * w'$, wobei die Pixelwerte des Bildes dazu vorab quadriert werden.

Dabei gilt zu beachten, dass beide Convolutions sich hier einen gemeinsamen Filter teilen. Basierend auf der Funktion der Convolution ergibt sich damit:

$$output_{b,i,j,q} = \frac{\sum_{di,dj} \left(w_{di,dj,q} * X_{i+di,j+dj,k} \right)}{\sqrt{a - b + \sigma_q^2}} \tag{5.3}$$

$$\text{mit} \qquad a = \sum_{di,dj} w'_{di,dj,q} \cdot \underbrace{\sum_{di,dj} w'_{di,dj,q} * X^2_{b,i+di,j+dj}}_{\text{Convolution}} \tag{5.4}$$

$$\text{und} \qquad b = \left(\underbrace{\sum_{di,dj} w'_{di,dj,q} * X_{i+di,j+dj}}_{\text{Convolution}} \right)^2 \tag{5.5}$$

$$\tag{5.6}$$

Das σ^2 fungiert hier als eine Art Bias pro Ausgabekanal[3] und kann damit als zu den Ergebnissen der Convolutions zu addierender Vektor übernommen werden. An den For-

[1] Im Original[13, (22)] steht auch im Zähler ein w statt eines w', es handelt sich auf Nachfrage aber um verschiedene Filter

[2] Bei Grundlage der Funktion von https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

[3] im Original σ_q^2, da als Quantisierungsrauschen vorgesehen

meln der normalen und der Depthwise Convolution orientiert, wie sie in der Tensorflow-Dokumentation verwendet werden[4], nutze ich bei Verweis auf einzelnen Werten den Index q für die Ausgabekanäle, ohne diesen meine ich den gesamten Vektor.

5.1.2 Anwendung auf mehrere Kanäle

Zur Übertragung auf den Fall eines mehrkanaligen Eingabebilds gehe ich ähnlich mit dem kontrastnormalisierenden Gradienten cng^* in [13, (44)][5][3][5] vor:

$$cng^* = \frac{\sum_j \begin{pmatrix} X_j * (-\eta x w) \\ X_j * (-\eta y w) \end{pmatrix} \begin{pmatrix} X_j * (-\eta x w) \\ X_j * (-\eta y w) \end{pmatrix}^T}{\sum_i \left(X_i^2 * w' - (X_i * w')^2 + \sigma^2 \right)} \tag{5.7}$$

Zur Anwendbarkeit stelle ich den Zähler bei meiner Übertragung auch hier durch eine Convolution dar. In der Formel in 5.7 ist der Zähler ein quadratischer Term (ein äußeres Produkt) einer Convolution, aus Gründen, die im Beweis [13] liegen. Das ist so in die Architektur eines Convolutional Neural Networks nicht zu übertragen. Deshalb verwende ich in Analogie zur Struktur des Einkanal-Falls eine Wurzel im Nenner, aber mit dem Term des Nenners in 5.7. Dadurch übertrage ich $\sum_i$ in 5.7 als eine Summierung über die Kanäle, umgekehrt erhalte ich dadurch im Falle nur eines Kanals wieder die Gleichung 5.3.

Damit erstelle ich die folgende Umsetzung pro einzelne Batch, wobei ich zur Vereinfachung der Darstellung hier kein Striding und keine Dilation Rate verwende:

$$output^*_{b,i,j,q} = \frac{\sum_{di,dj,k} \left(w_{di,dj,k,q} * X_{i+di,j+dj,k} \right)}{\sqrt{a - b + \sigma_q^2}} \tag{5.8}$$

$$\text{mit} \qquad a = \sum_{di,dj,k} w'_{di,dj,k,q} \cdot \underbrace{\sum_{di,dj,k} w'_{di,dj,k,q} * X^2_{b,i+di,j+dj,k}}_{\text{Convolution}} \tag{5.9}$$

$$\text{und} \qquad b = \sum_k \left(\underbrace{\sum_{di,dj} w'_{di,dj,k,q} * X_{i+di,j+dj,k}}_{\text{Depthwise Convolution}} \right)^2 \tag{5.10}$$

$$\tag{5.11}$$

Die Depthwise Convolution ist hier erforderlich, um die Quadrierung vor der Summe über die Eingangskanäle k vorzunehmen[6].

[4] siehe https://www.tensorflow.org/api_docs/python/tf/nn/depthwise_conv2d und https://www.tensorflow.org/api_docs/python/tf/nn/conv2d

[5] Auch hier liegt eine Korrektur des w im Zähler zu einem w' gegenüber dem Original[13, (44)] vor

[6] Zwar könnten beide Terme als Depthwise Convolution dargestellt werden, allerdings ist die Depthwise Convolution in der aktuell in Tensorflow genutzten Implementation allerdings signifikant langsamer beim Training.

Dadurch, dass der Zähler bei einer Convolution bleibt, ist die Schicht als eine Convolution um eine lokale Skalierung ergänzt zu verstehen. Dabei ist diese Skalierung als abhängig von der Varianz des Bildes zu verstehen, wobei der Bereich der lokalen Betrachtung vom verwendeten Filter w' abhängt. Für den vorgesehenen Einsatz muss dieser dabei die gleichen Eigenschaften wie der Filter w aufweisen, dazu zählen seine Größe und Anwendungsdetails wie Striding (Schrittgröße bei der Convolution).

Durch das Verhältnis von w' und σ^2 zueinander bestimmt sich die Stärke der Skalierung und damit der Kontrastnormalisierung. Eine Vergrößerung von σ^2 verringert den Grad der Normalisierung, eine Verkleinerung verstärkt sie, da sich der durch die Convolutions vom Bild abhängige relative Anteil entsprechend ändert. Eine proportionale Änderung von w' und σ führt zu einer Skalierung insgesamt, die bei einem Einsatz in einem Convolutional Neural Network durch eine anschließende Batch Normalization kompensiert werden kann.

Um eine Division durch 0 zu verhindern, beschränke ich $w' > 0$ und $\sigma^2 > 0$. Dabei verwende ich zur Verhinderung eines verschwindenden Gradienten im ungültigen Bereich einen Regularizer, der das Loss vergrößert, um die Werte im Prozess des Trainings im unbeschränkten Bereich zu halten. Für einen minimalen Wert min lässt sich dies allgemein wie folgt darstellen:

$$l_{limit}(x) = \begin{cases} 0 & |x > min \\ min - x & |x \leq min \end{cases} \tag{5.12}$$

$$loss_{limit} = \sum_{di,dj,k,q} l_{limit}\left(w'_{di,dj,k,q}\right) + \sum_{q} l_{limit}\left(\sigma_q\right) \tag{5.13}$$

Dabei beschränkt 5.12 die Betrachtung effektiv auf alle Werte, die den minimal akzeptablen Wert min unterschreiten. Mit 5.13 erhalte ich den jeweiligen Abstand zu min, der im beschränkten Bereich als linearer Faktor differenzierbar ist und damit sinnvoll als Erweiterung des Loss agieren kann. Durch eine unterschiedliche Wahl von min für w' und σ^2 kann dabei ein minimale Stärke der Kontrastnormalisierung festgelegt werden.

5.1.3 Stärke der Kontrastnormalisierung

Um die Stärke der Kontrastnormalisierung quantifizieren zu können, verwende ich als eine Art Maß

$$strength = \frac{w'^2}{w'^2 + \sigma^2} \tag{5.14}$$

Da der Divisor durch die Convolution des Bildes in 5.7 mit w' von diesem abhängt, gibt die Größe von w'^2 im Verhältnis zu $sigma^2$ den Grad der Einbeziehung des Bildes an. Liegt $w' = 0$ und $\sigma^2 \neq 0$ vor, so wird $strength = 0$ und es gibt keinen Einfluss des Bildes auf den Divisor; folglich findet keine Normalisierung statt. Falls dabei $\sigma^2 \neq 1$ gilt, so erfolgt eine Skalierung, die durch eine anschließende Batch Normalization aufgefangen werden kann. Gilt anders herum $\sigma^2 = 0$ und $w' \neq 0$, so wird $strength = 1$ und die Kontrastnormalisierung ist maximal. Der Fall, dass $w' = 0$ und σ^2 vorliegt, muss wegen

der Division durch 0 verhindert werden. Dies ist möglich durch setzen von $min > 0$, wodurch allerdings die Grenzfälle der Stärke der Kontrastnormalisierung ausgeschlossen werden.

Ermöglichung der Grenzfälle

Damit die Grenzfälle mit verwendet werden können, kann alternativ für den gesamten Divisor die Bedingung $\sum_i(\sqrt{X_i^2 * w' - (X_i * w')^2 + \sigma^2}) > 0$ gesetzt werden. Dadurch kann bei gleichem min für w' und σ^2 dieses auf $min = 0$ gesetzt werden, sodass insbesondere auch keine Kontrastnormalisierung möglich ist. Damit kein „explodierender" Wertebereich entsteht, wenn der Divisor gegen 0 geht, sollte in alle Fällen entweder min oder die Limitierung des Divisors weit genug über 0 gesetzt sein. Dabei hat $min = 0$ zu verwenden den Nachteil, dass eine Begrenzung des Divisors über 0 in dem entsprechenden Bereich darunter einen fehlenden Gradienten zur Folge haben kann. Daher untersuche ich bei den Experimenten entsprechend darauf.

5.1.4 Verwendung von vortrainierten Gewichten

Bei der Anwendung eines Convolutional Neural Networks auf neue Problemstellungen sind normalerweise große Mengen an Daten zum Training erforderlich. Da diese nicht immer zur Verfügung stehen, kann mit Transfer Learning aus einer anderen Aufgabe antrainiertes Grundwissen auf eine neue Aufgabe versucht werden, dieses Wissen in Teilen zu übernehmen. Dieses Vorwissen ist in der Regel mit intensivem Training auf einem großen Datensatz wie beispielsweise dem ImageNet entstanden. Daher ist seine Erzeugung mit einem größeren Aufwand verbunden. Um die kontrastnormalisierende Schicht in einem bereits vortrainierten Netz zu verwenden, kann über den unteren Grenzfall der Stärke der Kontrastnormalisierung bei Ersetzung einer zweidimensionalen Convolution ein identisches Ergebnis erreicht werden. Damit liegt ein Ausgangspunkt vor, von dem ich in Kapitel 7 als Teil meiner Experimente versuche, ein besseres Ergebnis zu erreichen. Durch den Einsatz von Gewichten aus einem Trainingsprozess des unmodifizierten Netzes, der bereits ein Plateau erreicht hat, kann ich damit eine Aussage machen, ob ausgehend von diesen noch eine Verbesserung mit der kontrastnormalisierenden Schicht möglich ist.

5.2 Optimierungen des Trainings

Im Folgenden nenne ich zentrale Optimierungen des Trainingsprozesses, die über implementierungsspezifische Details hinaus gehen, wie ich sie in 6.2 beschreibe.

5.2.1 Initialisierung der Softmax-Schicht

Um den Prozess des Trainings zu beschleunigen, verwende ich eine Initialisierung der Softmax-Schicht, wie sie unter [22] beschrieben ist. Diese Schicht liefert als letzte des unmodifizierten und aller in dieser Arbeit betrachteten modifizierten Netze als Ausgabe die jeweiligen Ergebnisse der Semantic Segmentation. Durch die Initialisierung ist nach [22] möglich, dem Netz einen Startpunkt aufbauend auf der zu erwartenden Verteilung

der Klassen im Bild zu geben. Wie dort beschrieben erstelle ich über den Datensatz, den ich im Rahmen dieser Arbeit erzeuge eine Summe der Pixel pro Klasse und setze den Logarithmus dieses Wertes als Initialisierung der Gewichte der Softmax-Schicht ein. Da der Datensatz eine große Anzahl an Bildern enthält, benötige ich das in [22] weiter beschriebene Laplace Smoothing nicht. Durch die große Anzahl ist von einer breiten Verteilung der Häufigkeit an Pixeln auszugehen. Sollte eine Klasse in allen Trainingsbildern durch keinen einzigen Pixel repräsentiert sein, liegt damit ein methodischer Fehler vor, der zu beheben ist. Anhand von eigenen Versuchen habe ich die Beobachtung aufgestellt, dass der Fortschritt des Trainings für Semantic Segmentation mit dieser Art der Initialisierung um mehrere Epochen beschleunigt wird. Daher verwende ich diese Initialisierung in den Experimenten im Kapitel 7.

5.2.2 Augmentations

Augmentations bezeichnen im Kontext des Trainings von neuronalen Netzen zufällige Veränderungen der Eingabedaten. Dies dient der effektiven Vergrößerung eines Datensatzes, um ein besseres Trainingsergebnis erhalten zu können. Dazu führen sie üblicherweise für das Training zufällige Veränderungen nach ihrem jeweiligen Schema an dem Trainingsdatensatz durch, bevor dieser jeweils für das Training der aktuellen Epoche verwendet wird.

Je kleiner ein Trainingsdatensatz ist, desto größer ist die Gefahr des Overfittings, bei dem ein Netz zu spezifische Details des Trainingsdatensatzes lernt und dadurch auf einem Validierungsdatensatz signifikant schlechtere Ergebnisse liefert. Um dieses Problem abzuwenden, verwende ich ein Set an diversen Augmentations, wie ich sie in Kapitel **??** beschreibe.

5.3 Methodik der Evaluation

Um eine Evaluation der kontrastnormalisierenden Schicht durchzuführen, untersuche ich anhand des Umgangs des betrachteten Netzes mit starken Kontrasten in Bildern mit und ohne den Einbau der Schicht, die ich im Rahmen dieser Arbeit vorstelle. Den konkreten Vergleich führe ich auf Basis eines Trainings beider Netzvarianten mit dem Teil des Datensatzes mit weicher Beleuchtung durch. Dabei teile ich den Datensatz zur Vergleichbarkeit vorher so auf, dass immer dieselben Szenen zum Training, der Validation beim Training und für den Test hier verwendet werden. Durch den Vergleich der Vorhersagen des Netzes auf Teile des Datensatzes mit sowohl weicher als auch harter Beleuchtung erhalte ich eine Aussage darüber, welches der Netze besser für diese Aufgabe geeignet ist. Daraus schließe ich, inwieweit die vorgestellte Schicht hier eine Verbesserung darstellt.

Ein mindestens erreichbares Loss lässt sich dabei über die Entropie der Klassenverteilung bestimmen:

$$p_i = \frac{n_i}{\sum_j n_j} \tag{5.15}$$

$$E = -\sum_i p_i \cdot ln(p_i) \tag{5.16}$$

Dabei sind n_i die Anzahl der Pixel einer Klasse i in allen betrachteten Bildern und $\sum_j n_j$ die Anzahl der Pixel aller Klassen zusammen. Mit der in Abschnitt 5.2.1 beschriebenen Initialisierung sollte dieses Ergebnis bereits in etwa als Ausgangslage des Trainings vorliegen.

5.4 Erweiterung auf alle Stufen des Encoders

Wie ich weiter in Kapitel 6.2 beschreibe, ist ein zentraler Punkt meiner Evaluation der Einbau meiner Schicht in ein U-Net mit einem ResNet50. Darüber hinausgehend schlage ich eine Struktur als Erweiterung vor, wie die kontrastnormalisierende Schicht auf alle Auflösungsstufen des Encoders angewendet werden kann. Dabei ist für mich ein zentraler Aspekt die Kontrastnormalisierung von Bilddaten und nicht die von Zwischenergebnissen. Innerhalb des Netzes selbst könnte eine Normalisierung der Kontraste ein Erlernen von Strukturen behindern, darum erachte ich nur einen Einsatz direkt am Anfang eines Convolutional Neural Networks für sinnvoll. Die Struktur ergänzt die Encoder-Decoder-Struktur des U-Nets dabei um einen weiteren Pfad.Diesen bezeichne ich im Folgenden als Seitenarm des Netzes. Meine Idee ist es, die Eingabedaten für jeden Schritt der Auflösungsreduktion des Encoders entsprechend zu skalieren, durch die kontrastnormalisierende Schicht verarbeiten zu lassen und anschließend in das eigentliche Netz einzuspeisen. Diese Einspeisung soll, jeweils in der ersten Convolution nach der Reduktion der Auflösung erfolgen. Um die Erhöhung der Anzahl der Gewichte zu begrenzen, schlage ich vor, dass alle kontrastnormalisierenden Schichten geteilte Gewichte verwenden. Dazu müssen alle in ihnen verwendeten Convolutions die gleichen Eigenschaften wie Anzahl der Filter und unter anderem Striding aufweisen. Da für jede Schicht das σ^2 als kombinierter Bias der beiden Convolutions mit w' im Divisor agiert, besitzen dabei über ein gleichsetzen der Filteranzahl der w' aller kontrastnormalisierenden Schichten auch alle σ^2 die gleiche Form.

5.4.1 Verwendung von vortrainierten Gewichten

Um wie in Abschnitt 5.1.4 vortrainierte Gewichte des Ausgangsnetzes weiterverwenden zu können, kann auch hier der Grenzfall einer Kontrastnormalisierung der Stärke 0 als Initialisierung verwendet werden. Des Weiteren sollten für die von mir vorgeschlagene Struktur des Seitenarms die Teile der Gewichte der entsprechenden Elemente angepasst werden, die mit den kontrastnormalisierenden Schichten aus dem Seitenarm verrechnet werden. Indem auch diese mit 0 initialisiert werden, erreicht das Netz wieder einen Ausgangspunkt identisch zu dem unmodifizierten Netz. Dadurch können etwaige Verbesserungen, die in einem derartig modifizierten Netz erreicht werden und bei gleichwertigem weiteren Training des unmodifizierten Netzes nicht erreicht werden, auf die neue Struktur zurückgeführt werden.

Kapitel 6

Implementierung und Umsetzung

In diesem Kapitel ergänze ich die Umsetzung der Programmierung auf technischer Ebene gegenüber 4. Mein Ziel ist es dabei vor allem auf die Punkte einzugehen, die bei der Umsetzung eines vergleichbaren Projektes wichtig sein dürften.

6.1 Datensatzerzeugung

Im Folgenden beschreibe ich relevante Teile der Erzeugung des Datensatzes auf Ebene der Programmierung mit der Unreal Engine. Dabei baue ich auf der Beschreibung des Aufbaus in 4.2 und des Ablaufs in 4.3 auf.

6.1.1 Funktionsweise

Die Unreal Engine als Rendering und Physics Engine wird in vielen Szenarien verwendet, beispielsweise in der Erstellung des Falling-Things-Datensatzes[34], von dem die Erstellung des hier beschriebenen Datensatzes inspiriert ist. Aufgrund der Vorteile synthetischer Datensatzerzeugung, wie unter Abschnitt 4.1 ausgeführt, ist diese hier für vergleichbare Projekte sinnvoll. Es können, aufbauend auf der im Rahmen dieser Arbeit erstellten Software, mit moderatem Aufwand weitere große Datensätze erzeugt werden, die für vergleichbare Projekte genutzt werden können. Dabei läuft die Generierung der einzelnen Bilder wie folgt ab:

Für jede einzelne Szene lasse ich einen der abgesteckten Bereiche jeweils zufällig auswählen, gewichtet mit dessen Grundfläche, sodass, auf alle Szenen betrachtet, auf der Bodenflächen aller Bereiche im Schnitt gleich viele Objekte landen.Damit die Objekte später an einem räumlich beschränkten Ort liegen bleiben, platziere ich nun als Ziel der später erscheinenden Objekte einen großen Zylinder so, dass er seitlich nicht aus dem Bereich hinaus ragt.Dieser Zylinder funktioniert dabei als durchsichtiges, aber physikalisch blockierendes Objekt, sodass sich in diesem bewegende Objekte auf den innen liegenden Bereich beschränkt sind.Um den Zylinder richte ich die Kamera zur Aufnahme der Bilder auf den Mittelpunkt des Bodens des Zylinders aus. Entfernung und Winkel zum Zylinder in senkrechter und waagerechter Richtung lasse ich dabei zufällig innerhalb eines festgelegten Intervalls wählen, dies erhöht die Varianz der Bilder. Das Intervall wähle ich dabei so, dass die später im Zylinder liegenden Objekte gut sichtbar sind.Da an den meisten Orten im Sun Temple auch mit den Gittern nur wenige Schlag-

schatten entstehen, platziere ich über der Höhe der Kamera in der Nähe des Zylinders weitere Beleuchtungsquellen, wobei ich die genaue Position zufällig ermitteln lasse.Um diese platziere ich jeweils auch ein Gitterobjekt, im Kontrast zu den statischen sind diese Gitter jedoch kugelförmig, sodass gebogene Schatten entstehen. Da diese meist näher als die anderen Beleuchtungsquellen an dem Zylinder sind, ergeben sich dadurch schärfere Schatten.Anschließend werden die Objekte innerhalb des Zylinders in der Luft in zufälliger Lage mit aktiver Physik erzeugt. In der Folge fallen diese auf den Boden.

Sobald die Objekte zur Ruhe gekommen sind oder ein Zähler abgelaufen ist, werden die Bilder mit der zuvor platzierten Kamera aufgenommen. Das Kameraobjekt aus NDDS kann dabei verschiedenen Arten von Bildern aufnehmen. Dazu zählen ein normal gerendertes Bild, ein Tiefenbild, ein Bild der Class und eines der Instance Segmentation. Für die Erstellung dieses Datensatzes füge ich zwei weitere Optionen hinzu, mit der hier verwendeten modifizierten Version von NDDS kann damit auch ein Bild mit hoher und ein Bild mit niedriger Global Illumination aufgenommen werden.

Dazu lasse ich vor der Aufnahme dieser beiden Arten von Bildern die Stärke der Global Illumination auf den entsprechenden Wert setzen. Um die Möglichkeiten der späteren Weiterverwendung des erzeugten Datensatzes zu erhöhen, lasse ich dabei auch die in dieser Arbeit nicht benötigten normalen Bilder, Tiefenbilder und Bilder der Instance Segmentation erzeugen.

Nachdem alle Bilder aufgenommen sind, ist die Szene fertig bearbeitet und der Aufbau der nächsten Szene wird begonnen, wofür sich der Ablauf ab 4.3.1 wiederholt. Dies setzt sich fort, bis eine vorab eingestellte Anzahl an Szenen aufgenommen ist.

Da eine Änderung der Auflösung der exportierten Bilder keine direkt erkennbare Änderung der Simulationsgeschwindigkeit bewirkt, verwende ich eine Auflösung von 1120×1120, die über der zu erwarteten benötigten Auflösung liegt. Durch die deutlich größere Größe können die Bilder anschließend auf die benötigte Auflösung herunter skaliert und diese Zielgröße auch etwas nach oben hin angepasst werden, ohne dazu einen neuen Datensatz erstellen zu müssen. Dabei erachte ich die Auflösung von 224×224 als die minimal benötigte Größe, da sie für das Training der auf dem ImageNet-Datensatz vortrainiert verfügbaren Gewichte verwendet wurde[20]. Als Minimum erachte ich diese deshalb, weil der ImageNet-Datensatz seinen Fokus auf größere Bildstrukturen setzt: Das Ziel ist die Klassifizierung des gesamten Bildes, bei der Semantic Segmentation in dieser Arbeit sind die zu erkennbaren Strukturen jedoch kleiner.

Zur Verwendung als Datensatz mit der „tf.data“ API von Tensorflow sortiere ich die Dateien anschließend wie folgt:

Bezeichnung	Bedeutung
normal	Unmodifiziert gerendertes Bild
cs	Class Segmentation
is	Instance Segmentation
depth	Tiefenbild
gi	Bilder mit erhöhter Global Illumination für weiche Beleuchtung und schwache Kontraste
gi_low	Bilder mit geringer Global Illumination für harte Beleuchtung und damit scharfe Kontraste

Anschließend teile ich die Bilder entsprechend auf in einen Teil zum Trainieren. Insgesamt erstelle ich 24000 Bilder, diese teile ich in Unterordnern weiter auf: 20000 Bilder zum Training, 2000 zur Validierung und 2000 für den Vergleichstest. Dies hat den Zweck, durch die fixe Aufteilung für alle Experimente die gleichen Bedingungen zu schaffen. Dabei müssen die Dateinamen angepasst werden, da Tensorflow in der genutzten „tf.data“ API eine Nummerierung beginnend mit 0 erwartet. Hierzu bietet sich die Verwendung eines Werkzeugs zur Dateiumbenennung wie beispielsweise GPRename [30].

6.1.2 Optimierungen

Mit den von mir gewählten Parametern erweist sich eine Wartezeit von 5 Sekunden nach dem Beginn des Falls der Objekte als ausreichend, damit die Objekte anschließend still liegen. Da ich für den hier zu erzeugenden Datensatz insgesamt 24.000 Bilder erzeuge, würde dies eine Wartezeit $33,\overline{3}$ Stunden bedeuten. Wenn Anpassungen und damit neue Durchläufe notwendig sein sollten, könnte dies schnell zu vielen Tagen Wartezeit führen. Um die Zeit zu verkürzen, lasse ich die Simulationsgeschwindigkeit auf den maximalen Faktor von 20 setzen, während die Objekte am Fallen sind. Da bei hoher Geschwindigkeit öfters Fehler wie beispielsweise zappelnde Objekte aufzutreten scheinen, lasse ich die Geschwindigkeit kurz vor Ende aller Bewegungen auf normal zurücksetzen. Da dennoch aufgrund der Objektdichte derartige Fehler immer wieder auftreten können, sollte die maximale Wartezeit auf Stillstand aller Objekte nicht zu hoch liegen.

Die Erzeugung von Objekten in der Unreal Engine ist im Vergleich zu ihrer Bewegung relativ rechenintensiv. Neben der in NDDS enthaltenen automatischen Wiederverwendung von Rohformen der zu fotografierenden Objekte verwende ich daher auch die neu hinzugefügten Hilfsobjekte wie den Zylinder und den beweglichen Beleuchtungsquellen weiter, um die Gesamtrechendauer zu senken.

6.1.3 Hinweise zur Verwendung

Zur Erstellung des Datensatzes verwende ich eine modifizierte Version von NDDS, weshalb die Installation grundlegend ähnlich wie in der zugehörigen Anleitung [33] beschrieben abläuft. Die nennenswerten Unterschiede sind hier, dass die Unreal Engine in der Version 4.24.3 statt 4.22 verwendet wird. Neuere Versionen werden voraussichtlich kleinere Anpassungen im Code benötigen. Des Weiteren liegt die modifizierte Version von NDDS auf dem beiliegendem Speichermedium vor, sodass das originale NDDS nicht mit dem Programm Git heruntergeladen werden muss. Das für die Anwendung hier fertig aufgebaute, modifizierte Level „Sun Temple“ ist ebenso auf dem Speichermedium enthalten und kann direkt in der Unreal Engine geöffnet werden.

Damit realistische Schatten um die Objekte entstehen, stelle ich alle Beleuchtungsquellen auf dynamisch - vorberechnete Schatten würden nur für statische Objekte auftauchen. Die Tiefe, in der die Schatten fokussiert sind, sollte abhängig vom gewählten Entfernungsbereich der Kamera gewählt werden.

6.2 Kontrastnormalisierende Schicht

Die Entwicklung führe ich mit Python und den folgenden externen Bibliotheken durch:

Bibliothek	Version	Verwendungszweck
Tensorflow[26]	2.6.0	Implementationsgrundlage, stellt die Elemente der Schichten und die Struktur zum Laden des Datensatzes bereit
Segmentation Models[36]	1.0.1, modifiziert	Erstellung des neuronalen Netzes zur Segmentierung
Classification Models[35]	1.0.0, modifiziert	Für Segmentation Models genutzte Implementation des ResNet, die ich um meine eigene Schicht erweitere
OpenCV[3]	4.5.2.54	Grafische Operationen außerhalb von Tensorflow wie die Visualisierung und die Analyse der Klassenverteilung
NumPy[19]	1.19.5	Grundlegende mathematische Operationen, insbesondere auf Matrizen
Matplotlib[21]	3.4.3	Erstellung von Diagrammen

Folgende externe Bibliotheken nutze ich dabei rein zu Zwecken der detaillierteren Einsicht während der Entwicklung zur Problemsuche, sie sind nicht von Bedeutung für die eigentliche Ausführung:

Bibliothek	Version	Verwendungszweck
guppy3[28]	3.1.2	Anzeige der Speicherauslastung durch einzelne Python Objekte
timebudget[10]	0.7.1	Anzeige der Ausführungsdauer von Abschnitten des Codes

Die Routine zur Visualisierung basiert auf dem Beispiel für die Semantic Segmentation mit mehreren Klassen in [36].

Kapitel 7

Experimente und Diskussion

In den folgenden Abschnitten stelle ich diverse Experimente vor, die ich als Teil dieser Arbeit durchführe. Dabei sind von zentraler Betrachtung jeweils das Loss, welches der Wert ist, nach dem das jeweilige Netz durch das Training optimiert wird. Als Metriken verwende ich Intersection over Union[29] (IoU) und F1[5], wie vorgestellt unter [32]. Da keine direkte Optimierung nach diesen möglich ist, verwende ich als Loss die Categorical Crossentropy. Dabei nutze ich den Optimizer Adam, da ich diesen als Standard empfinde und er deutlich schneller als Stochastic Gradient Descent ist, meist abgekürzt zu SGD. Mit der Geschwindigkeit des Trainings der verwendeten Netze beziehe ich mich im Folgendem auf den Abfall des Validation Loss pro Epoche.

7.1 Allgemeine Einstellungen der Trainingsumgebung

Sofern nicht anders vermerkt, verwende ich für alle Experimente die folgenden Rahmenbedingungen: Ich nutze eine Batch Size von 20, da dies das Maximum ist, das mit allen Konstellationen im Rahmen des mir zur Verfügung stehenden Videospeichers bleibt und so eine Vergleichbarkeit der einzelnen Experimente sichergestellt ist. Die größte gemeinsame Batch Size verwende ich, da in der möglichen Größenordnung in der Regel durch Bildung des Gradienten über mehrere Bilder geringere Schwankungen der Gewichte und ein schnelleres Training insgesamt erreichbar sind. Eine Ausführung auf Google Colab[1] ist grundsätzlich möglich, je nach verwendeter Auflösung und zugewiesener Hardware kann allerdings eine Reduktion der Batch Size vonnöten sein. Angegebene Ausführungsdauern erreiche ich unter Verwendung einer NVidia GeForce GTX 3090 in Kombination mit einem AMD Ryzen 5900X. Als Optimierer nutze ich Adam, da dieser relativ stabile Resultate zu bewirken und eine sehr breite Verwendung aufzuweisen scheint und daher gute Vergleichswerte liefern dürfte. Angelehnt an das Beispiel der Semantic Segmentation für mehrere Klassen von [36] verwende ich dabei vorerst eine Learning Rate von 0.001.

[1] https://colab.research.google.com/

7.2 Mindestens zu erwartendes Loss

Aufbauend auf meinem Ansatz in 5.3 kann über eine Analyse der Verteilung der Pixel auf die Klassen eine Aussage darüber gemacht werden, was für ein Loss als zu erreichendes Minimum angesehen werden kann.

ID im Datensatz	Objekt des YCB-Datensatzes[4]	Anteil p_i
0	Hintergrund	0.961096
1	002_master_chef_can	0.002355
2	003_cracker_box	0.004751
3	005_tomato_soup_can	0.001085
4	006_mustard_bottle	0.001924
5	010_potted_meat_can	0.001266
6	011_banana	0.000904
7	019_pitcher_base	0.004869
8	025_mug	0.001291
9	036_wood_block	0.003445
10	040_large_marker	0.000278
11	061_foam_brick	0.000760
12	004_sugar_box	0.002171
13	009_gelatin_box	0.000857
14	021_bleach_cleanser	0.002692
15	051_large_clamp	0.001121
16	007_tuna_fish_can	0.000776
17	008_pudding_box	0.001363
18	024_bowl	0.002392
19	035_power_drill	0.002364
20	037_scissors	0.000680
21	052_extra_large_clamp	0.001548

Aus den Werten dieser Tabelle folgt für die Entropie E nach 5.16:

$$E = -\sum_i p_i \cdot ln(p_i) \ = 136.720$$

Da dies ein sehr hoher Wert ist, kann damit nur sehr begrenzt eine Aussage über Trainingsverläufe gemacht werden. Zu erklären ist dies durch den sehr hohen Anteil des Hintergrunds in Relation zu den Objektklassen, da diese über den Logarithmus am Ende größere Werte produzieren.

Betrachtet man nur den Hintergrund und vereint alle weiteren Klassen zu einer gemeinsamen, erhält man:

$$E \ = 3.286$$

Dieser Wert erscheint damit als plausiblerer Vergleichswert, bei Verwendung einer Softmax-Initialisierung wie unter Kapitel 5.2.1.

7.3 Auflösung und Format des Datensatzes

Um zu prüfen, ob eine Implementation des Algorithmus für einen Kanal sinnvoll ist, wie ich ihn unter 5.1.1 beschreibe, teste ich ein unmodifiziertes U-Net mit ResNet auf einen schwarz-weißen Datensatz. Dazu nutze ich einen Datensatz, den ich wie in Kapitel 4 beschrieben erzeuge, und beim Laden in das neuronale Netz in schwarz-weiße Bilder umwandle. Dabei können die durch ein Training mit ImageNet zur Verfügung gestellten Gewichte[2] durch die abweichende Kanalzahl der Eingabebilder nicht für Transfer Learning verwendet werden. Ein Training mit dem Teil des Datensatzes mit weicher Beleuchtung resultiert in dem Verlauf in Abbildung 7.1.

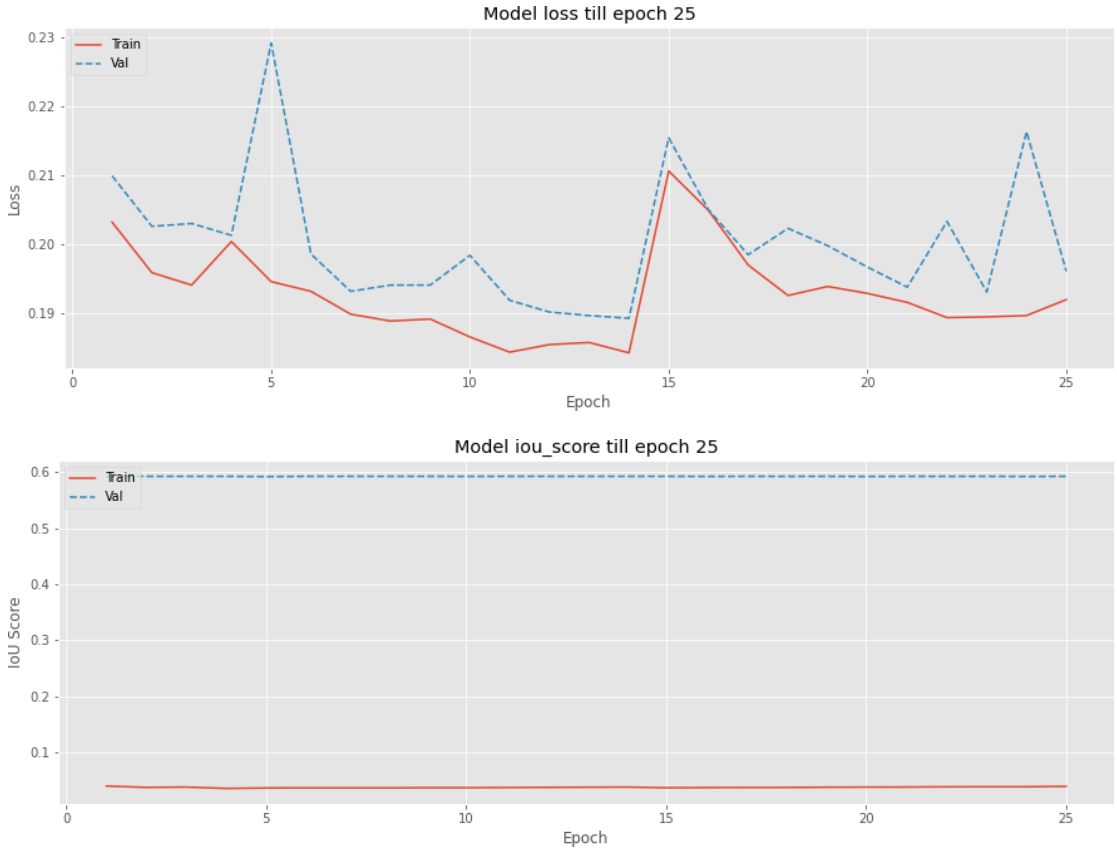


Abbildung 7.1: Trainingsverlauf mit einem Datensatz der Kantenlänge 224 bei Konvertierung in schwarz-weiß

Testweise Vorhersagen des Netzes auf sowohl den Test- als auch den Trainingsdatensatz resultieren in einem Bild fast nur aus dem Hintergrund bestehend, wie exemplarisch für ein Bild des Trainingsdatensatzes in Abbildung 7.2 zu sehen ist. Dabei ist links das Eingangsbild aus dem Datensatz, in der Mitte die Ground Truth (Musterlösung) und rechts die Predicted Truth, die Vorhersage des Netzes abgebildet.

[2] Für die Bibliothek Classification Models zu finden unter https://github.com/qubvel/classification_models/releases/tag/0.0.1

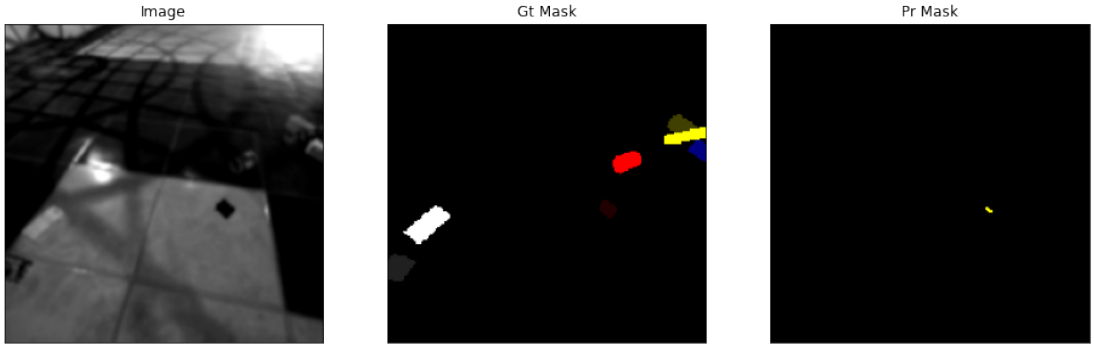


Abbildung 7.2: Vorhersage des Netzes für ein schwarz-weißes Bild der Kantenlänge 224 aus dem Trainingsdatensatz

Aus diesem Verlauf schließe ich, dass ein Training wahrscheinlich nur schwer möglich ist. Dabei wird es im Folgenden deutlicher durch den Vergleich mit weiteren Trainingsverläufen im mehrkanaligen Fall.

Anschließend teste ich den Datensatz mit allen verfügbaren Kanälen im unmodifizierten Netz, dabei ist hier möglich Transfer Learning einzusetzen. Dies resultiert in dem unter Abbildung 7.3 zu findenden Trainingsverlauf.

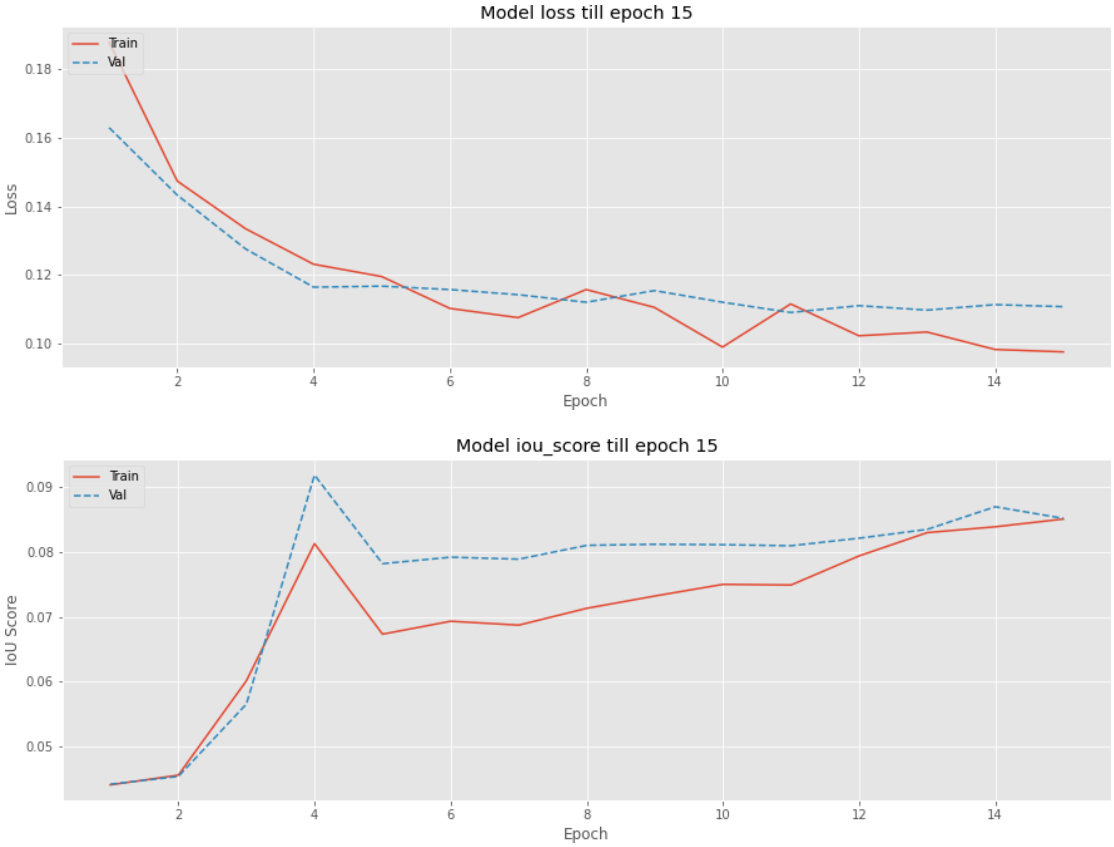


Abbildung 7.3: Trainingsverlauf mit einem Datensatz der Kantenlänge 224 bei Verwendung der Farbkanäle

Abbildung 7.4: Vorhersage des Netzes für ein farbiges Bild der Kantenlänge 224 aus dem Trainingsdatensatz

Zwar zeigt dieser Graph einen besseren Trainingsverlauf, jedoch sind in den unter beispielhaften Vorhersagen des Netzes nur Ansätze von Objekten zu erkennen.

Deshalb orientiere ich mich im Folgenden näher am ImageNet-Datensatz, mit dem

das ResNet50 für die ImageNet Large Scale Visual Recognition Challenge erfolgreich trainiert wurde. Ein Vergleich mit Beispielen aus dem ImageNet-Datensatz[8], wie sie unter Abbildung 7.5 zu sehen ist, zeigt relativ zu den Bildausschnitten deutlich größere Objekte.

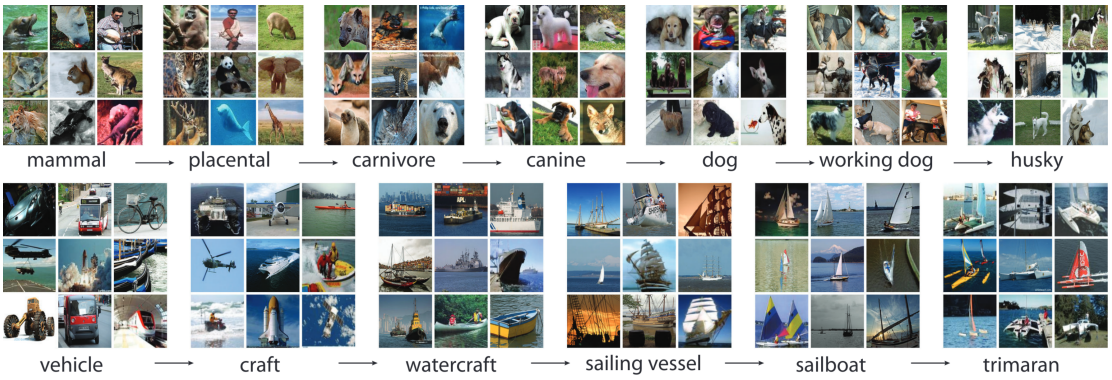


Abbildung 7.5: Beispiele aus dem ImageNet-Datensatz [8, Figure 1]

Da die vortrainierten Gewichte des ResNet durch Training mit Bildern aus dem ImageNet-Datensatz in Größe von 224×224 Pixeln erzeugt wurden, halte ich es für denkbar, dass diese Gewichte auf größere Strukturen ausgelegtes Wissen enthalten.
Durch ein Abschätzen der durchschnittlichen Objektgröße in den betrachteten Beispielbildern gehe ich davon aus, dass ein Datensatz mit Bildern in der Auflösung von 498×498 zu einer ähnlichen effektiven Größe der Objekte führen dürfte. Um flexibel bei der verwendeten Größe zu sein, generiere ich den Datensatz in der Auflösung 1120×1120, um ihn anschließend entsprechend herunter zu skalieren.
Als Kompromiss zwischen großen Bildern und damit steigendem Rechenaufwand nutze ich im Folgendem als Ziel der Skalierung eine Auflösung von 498×498, zugeschnitten auf 448×448, was das Vierfache der zuerst verwendeten Auflösung ist. Inspiriert von der Anwendung beim ImageNet-Datensatz erfolgt der Zuschnitt dabei im Trainingsdatensatz zufällig verschoben. Zwar sind Convolutional Neural Networks grundlegend durch die Convolution (Faltung) positionsinvariant, durch die Skalierung, wie sie im ResNet enthalten ist, ergibt sich jedoch eine gewisse Schrittgröße. Dies bedeutet, dass bei einem festen Ausschnitt immer in bestimmten Schritten Pixel durch die Schichten verbunden werden. Verschiebe ich nun die Eingabedaten, so erhöht sich damit die Varianz der Daten, die diese Schichten als Training erhalten. Um die Reproduzierbarkeit der Validation sicherzustellen, verwende ich dafür einen Ausschnitt nur jeweils auf den mittleren Bereich eines Bildes. Das konstante seitliche Abschneiden in der Validation erachte ich als einen akzeptablen Kompromiss, da mit der gewählten minimalen Einstellung der Kameraentfernung die Objekte meist zentriert sind. Dazu sind die Objekte vor ihrem Fall auf den Boden zufällig untereinander positioniert, sodass ich davon ausgehe, dass keine Objektklassen in einer problematischen Stärke benachteiligt wird. Zwar sind runde Objekte vorhanden, jedoch ist in der Analyse des vergleichbar erzeugten Datensatzes Falling Things [34], der die selben Objekte nutzt, keine zu weite Streuung der Objekte erkennbar. Das Ergebnis des Trainings des unmodifizierten Netzes mit Transfer Learning ist unter Abbildung 7.6 dargestellt. Beispielhafte Vorhersagen des Netzes auf

ein Bild des Trainings- und eines des Testdatensatzes finden sich unter 7.7.

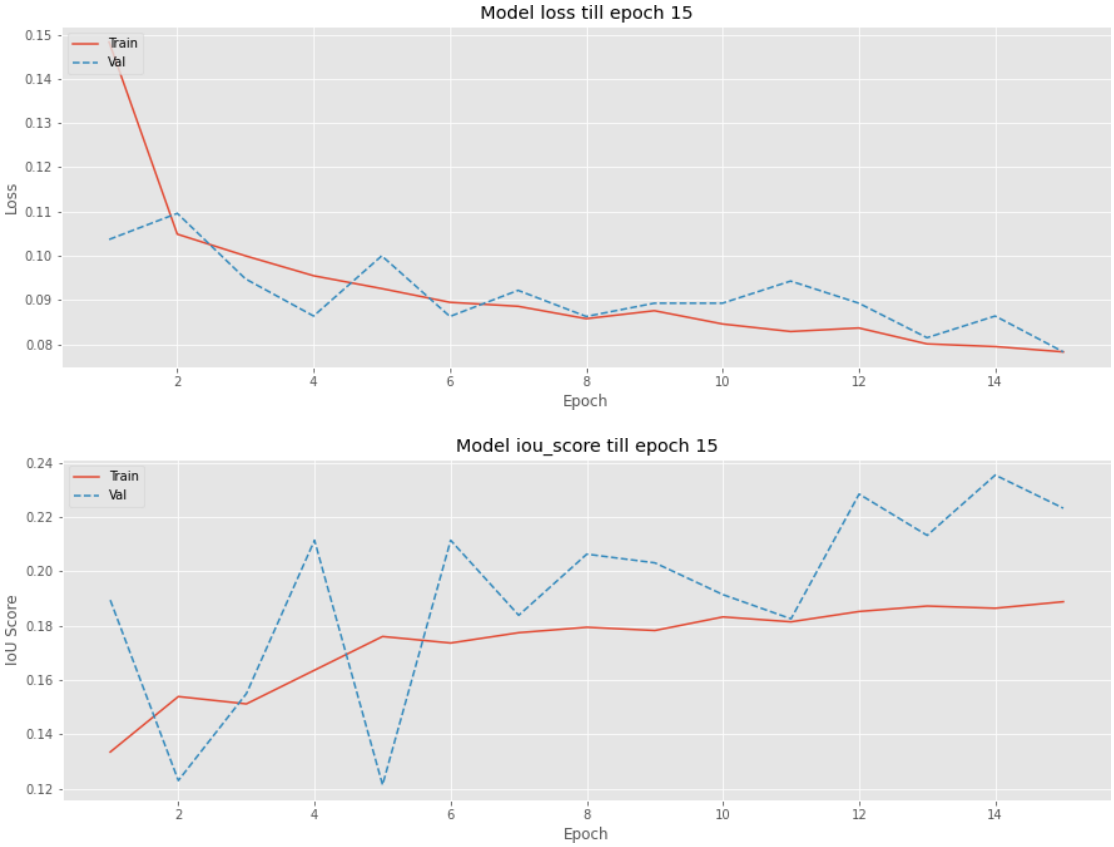


Abbildung 7.6: Trainingsverlauf mit einem Datensatz der Kantenlänge 448 bei Verwendung der Farbkanäle

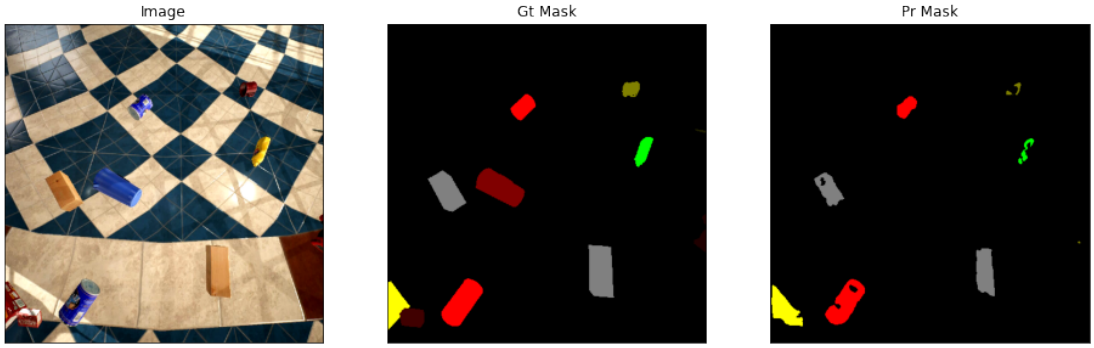
7.3.1 Verwechslung der Klammern

Bei einer größeren manuellen Studie ist auffällig, dass öfters die beiden Klammer-Objekte (051_large_clamp und 052_extra_large_clamp aus dem YCB Datensatz[4]) verwechselt werden. Da die Unterscheidung eher für Anwendungen beispielsweise in der Robotik wichtig sind, lege ich beide Objekte im Folgenden zu einer gemeinsamen Klasse zusammen, um den Fokus des Trainings mehr auf den Aspekt der Kontrastnormalisierung legen zu können.

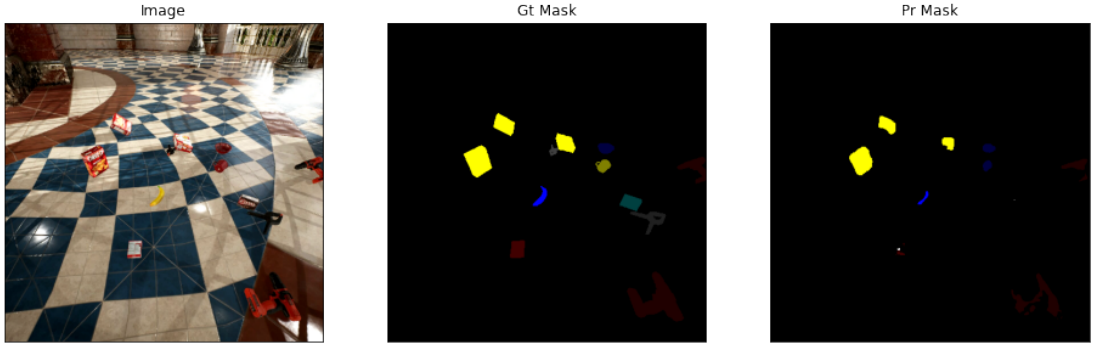
7.4 Augmentations

Um ein Overfitting auf die Trainingsdaten zu vermeiden, sind Augmentations hilfreich. Inspiriert von Beispiel für die Segmentierung von mehreren Klassen in [36] verwende ich eine Kombination aus diversen Augmentations[3]. Neben grundlegenden wie zufälligem

[3] Da die „tf.data“ Schnittstelle von Tensorflow gegenüber der dort verwendeten eigenen Methode zur Einlesung des Datensatzes je nach verwendeter Hardware große Leistungsvorteile durch eine Paralleli-



(a) Beispiel der Vorhersage auf den Trainingsdatensatz



(b) Beispiel der Vorhersage auf den Testdatensatz

Abbildung 7.7: Vorhersage des Netzes für ein farbiges Bild der Kantenlänge 448 aus dem Trainings- und dem Testdatensatz

Spiegeln, leichtem Rotieren, Verändern der Helligkeit, Anpassung von Kontrast, Farbton und Sättigung verwende ich außerdem die Cutout Augmentation[9], da dieses auch Verbesserungen für die Anwendung der Semantic Segmentation liefert[12]. Dabei stellt auch der unter dem Abschnitt 7.3 zufällige Ausschnitt eine Augmentation dar. Zum Vergleich mit dem Validierung- und Testdatensatz werden von mir keine dieser Augmentations verwendet, nur ein zentraler Zuschnitt wird zum Einhalten des Bildformates benötigt.

Ein Beispiel in Abbildung 7.8 zeigt den Verlauf eines Trainings des unmodifizierten Netzes ohne Augmentations, abgesehen von einem zentralen Zuschnitt, wie ich ihn ansonsten für die Validierung nutze. Dabei verwende ich den Teil des Datensatzes mit weicher Beleuchtung. Ich nutze hier für stabilere Ergebnisse bei einem längeren Training eine von 0.0001 auf 0.00001 gesenkte Learning Rate. Dabei verwende ich die in Tensorflow beinhaltete Erkennung eines Plateaus der Gradienten um erkennen zu können, ob noch größere Änderungen durch weiteres Training zu erwarten sind. Eine genauerer Vergleich verschiedener Learning Rates folgt in Abschnitt 7.6.

sierung bietet, nutze ich alternative Implementationen der Augmentations.

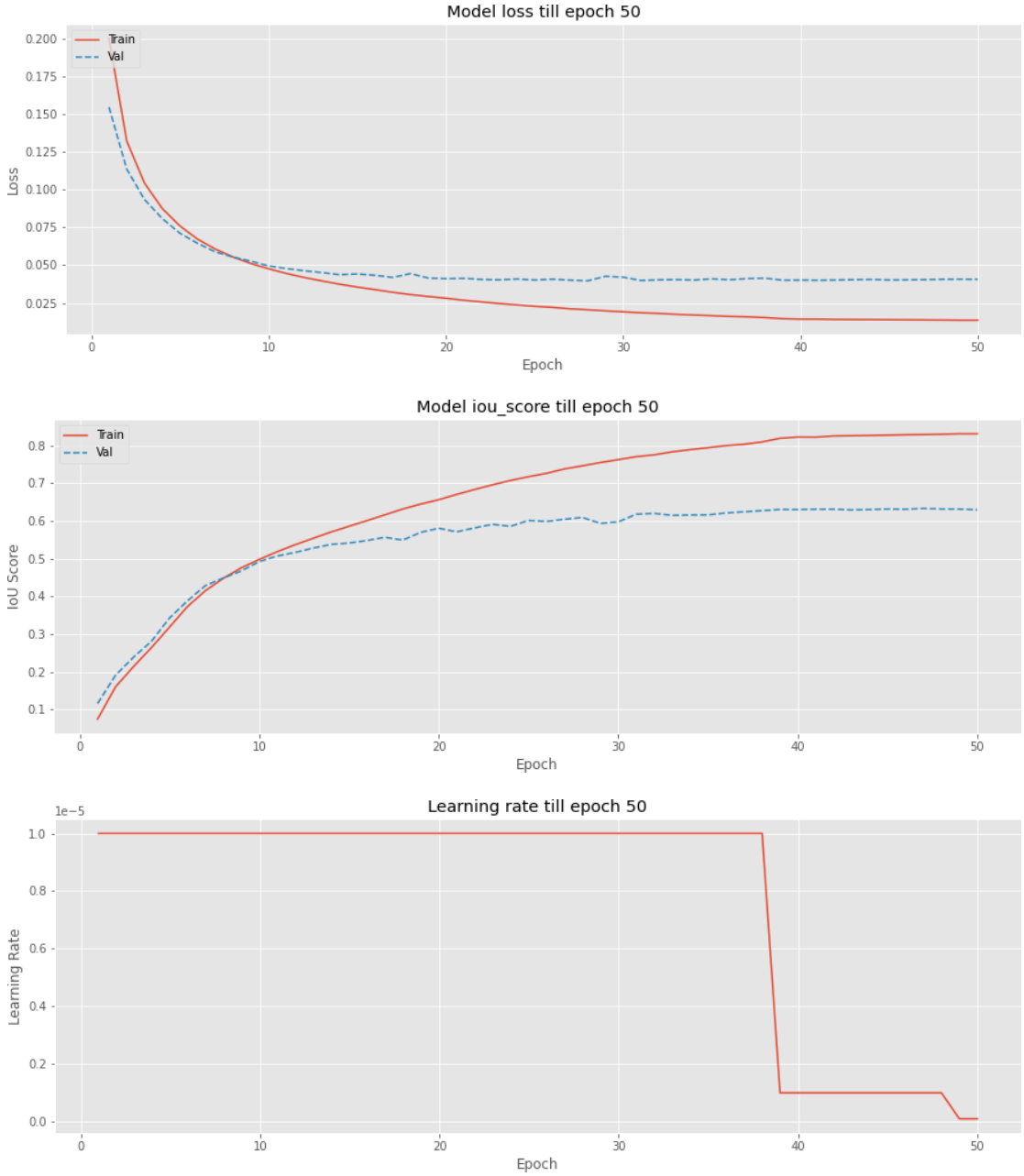


Abbildung 7.8: Training des unmodifizierten Netzes ohne Augmentations, mit weicher Beleuchtung und einer Learning Rate von 0.00001

Zum Vergleich ist in Abbildung 7.9 ein Trainingsverlauf mit aktivierten Augmentations abgebildet.

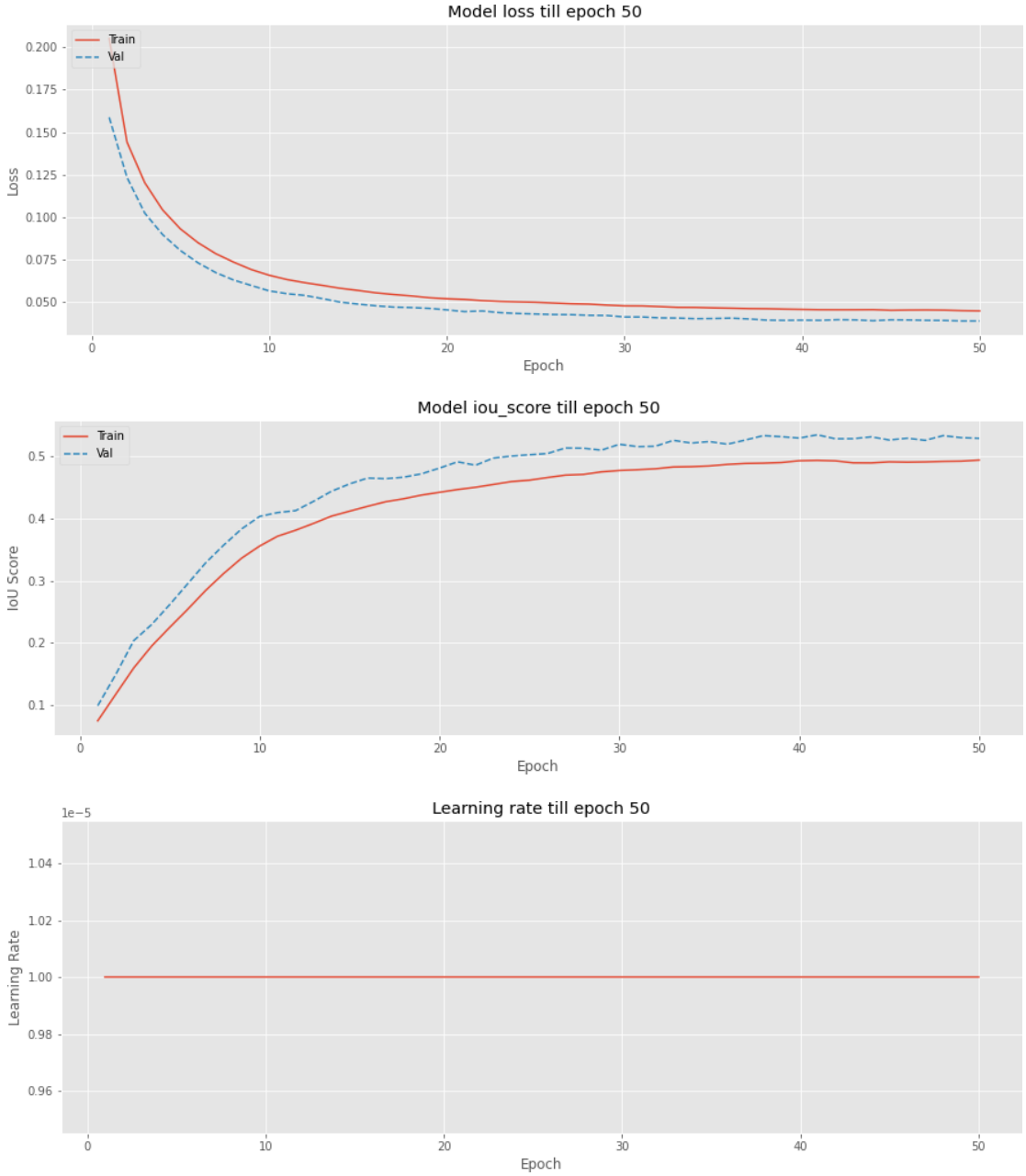


Abbildung 7.9: Training des unmodifizierten Netzes mit Augmentations, mit weicher Beleuchtung und einer Learning Rate von 0.00001

Die Abbildung 7.10 zeigt das selbe Experiment des Trainings ohne Augmentations angewendet auf das modifizierten Netz. Dabei verwende ich als Initialisierung konstante Werte, 0 für alle Werte in w' und 1 für alle Werte in σ^2. Im Abschnitt 7.5 behandele ich weiterführend verschiedene Initialisierungsansätze.

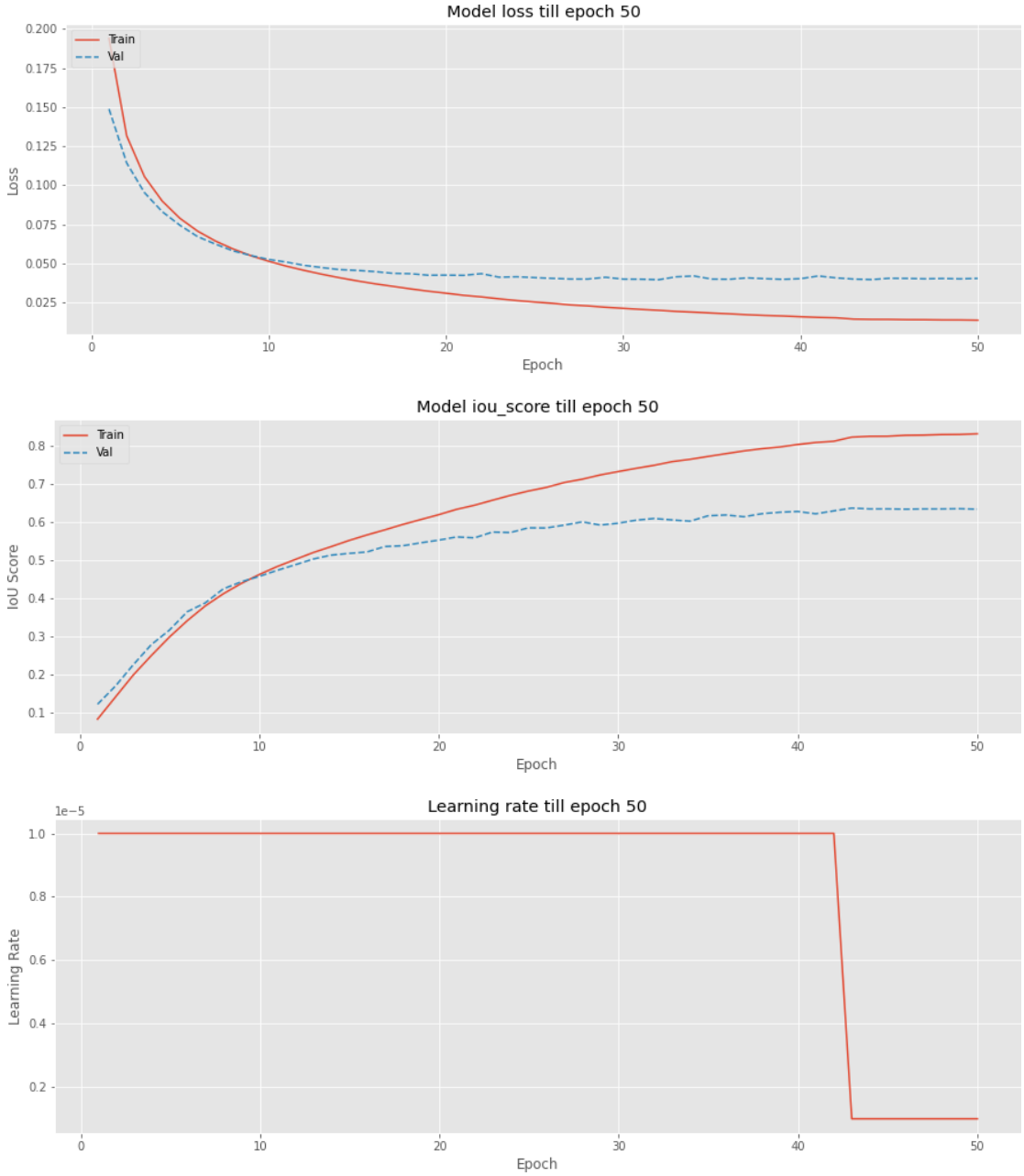


Abbildung 7.10: Training des modifizierten Netzes mit der kontrastnormalisierenden Schicht ohne Augmentations, mit weicher Beleuchtung und einer Learning Rate von 0.00001

Der Vergleich ist möglich über Abbildung 7.11, sie ist das Resultat des Trainings des modifizierten Netzes unter der Verwendung der Augmentations.

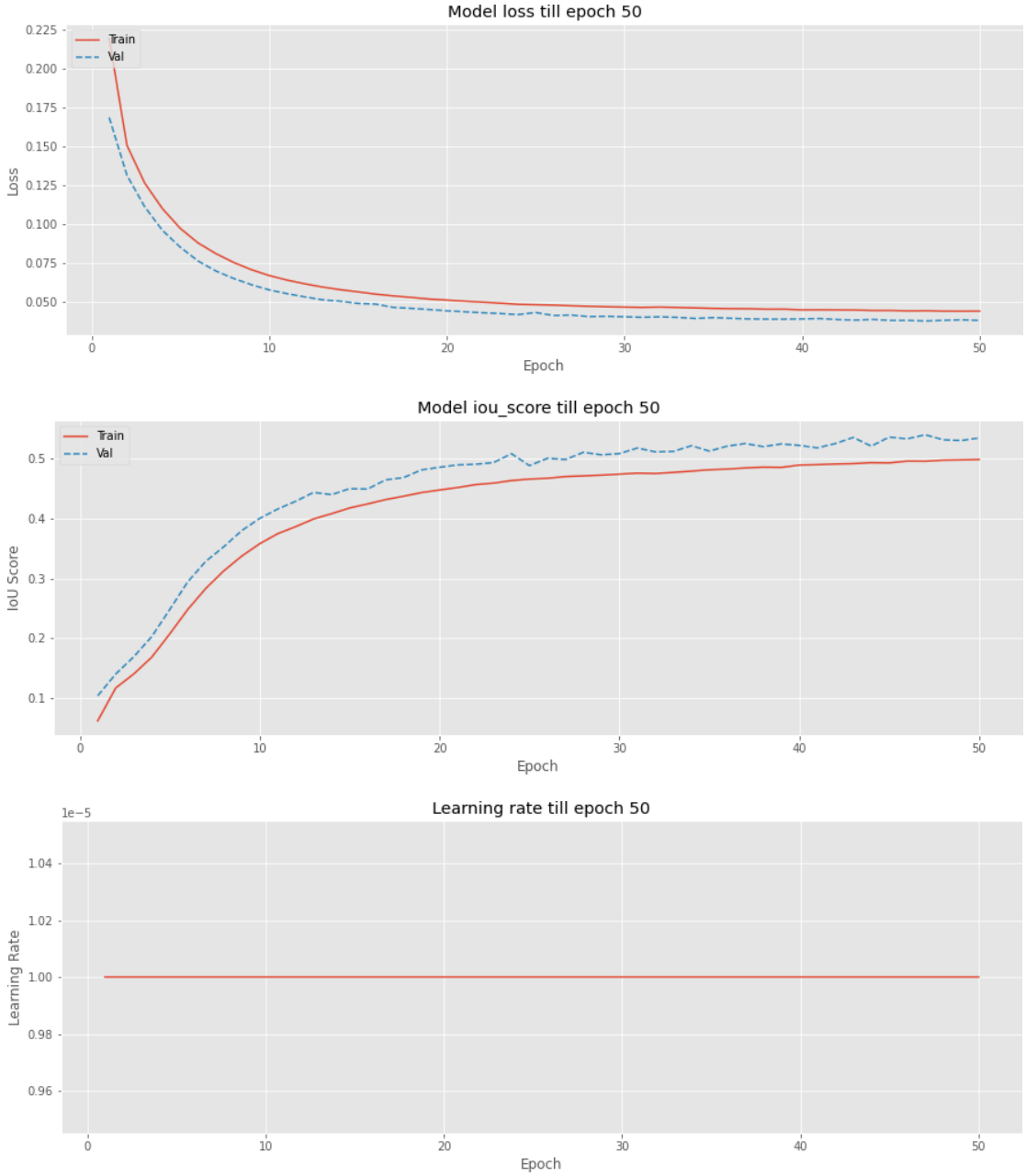


Abbildung 7.11: Training des modifizierten Netzes mit der kontrastnormalisierenden Schicht unter Verwendung der Augmentations, mit weicher Beleuchtung und einer Learning Rate von 0.00001

Da die Learning Rate jeweils durch die Plateauerkennung deutlich vor dem Ende des Trainings gesenkt wird, gehe ich davon aus, dass weiteres Training keine größeren Veränderungen mehr bewirken würde.

Ein Vergleich der Verläufe legt dar, dass ohne die Augmentations relativ bald ein Overfitting eintritt, da das Loss im Training im Verlauf er Epochen weiter sinkt, bei der Validierung jedoch auf einem höheren Wert verbleibt. Umgekehrt bei dem Verlauf

der Metrik Intersection over Union, hier zeigen höhere Werte ein besseres Ergebnis an. Daraus folgere ich, dass bei einem weiteres Training auf dem Trainingsdatensatz keine nennenswerten Verbesserung mehr erreichbar sind. Durch Verwendung der Augmentations ist dies nicht der Fall, durch die Kombination ist hier die Validierung sogar im Vorteil. Daraus schließe ich, dass die Kombination dieser Augmentations sehr erfolgreich ist.

7.5 Initialisierung der kontrastnormalisierenden Schicht

Für die Initialisierung der kontrastnormalisierenden Schicht in dem modifizierten Netz sind für mich verschiedene Ansätze denkbar.

7.5.1 Initialisierung für Transfer Learning

Für die Fortsetzung des Trainings mit vortrainierten Gewichten im Rahmen des Transfer Learnings sehe ich eine Initialisierung als sinnvoll an, welche vor weiterem Training die gleichen Resultate liefert wie das unmodifizierte Netz. Dazu nutze ich wie in Abschnitt 5.1.4 beschriebenen den unteren Grenzfall, bei dem w' mit 0 und σ^2 mit 1 initialisiert werden. Des Weiteren kopiere ich die Gewichte der durch die kontrastnormalisierende Schicht zu ersetzenden Convolution in die Convolution im Zähler. Dadurch die Initialisierung von $2'$ und σ^2 wird der Nenner 1, sodass die Schicht mit diesen Gewichten gleichwertig zur ersetzten Schicht fungiert. Dadurch können Veränderungen im Trainingsverlauf durch einen Vergleich des folgenden Trainingsprozess auf die veränderte Struktur zurückgeführt werden.
Würde man stattdessen die Schicht anderweitig initialisieren, so ist damit zu rechnen, dass umliegende Schichten sich stark verändern und damit vermutlich ihr vortrainiertes Wissen leicht durch größere Änderungen verlieren können.

7.5.2 Initialisierung ohne Transfer Learning

In Betrachtung von [16] kann eine naive Initialisierung von Gewichten in neuronalen Netzen potentiell schlechte Resultate liefern. Ein Anhaltspunkt scheint dabei zu sein, allzu große Änderungen des Wertebereichs zu vermeiden. Am Beispiel der He-Initialisierung orientierend, wie sie unter [17] beschrieben ist, ist ein mögliches Ziel dabei, die Varianz vor und nach des betrachteten Teils eines Convolutional Neural Networks möglichst ähnlich zu halten. Allerdings ist die Berechnung der Varianz meiner eigenen Schicht durch den Bruch nicht ohne Weiteres möglich. Um die Veränderung der Varianz der Schicht zu überprüfen, erstelle ich daher ein Skript, das meine eigene Schicht in einem separaten minimalen Netz betrachtet, welches nur aus dieser Schicht besteht. Dabei versuche ich die Bedingungen wie im ansonsten genutzten Netz nachzustellen. Dazu verwende ich die gleiche Anzahl an Ausgabekanäle und erstelle einen Pseudo-Datensatz, der uniform verteiltes Bildrauschen auf allen drei Farbkanälen (Rot, Grün und Blau) als Eingabe erzeugt. Es ergibt sich durch die Uniformverteilung pro Kanal bei Bildwerten zwischen

0 und 1 jeweils die folgende Varianz:

$$\mathrm{Var}(X_k) = \frac{1}{12} \cdot (1-0)^2 \tag{7.1}$$

$$= \frac{1}{12} \tag{7.2}$$

Mit einer Initialisierung von σ^2 mit $0.5 \leq \sigma^2 \leq 1.0$ und für die einzelnen Werte des Filters w' zwischen 0 und 0.5 ergibt sich eine Varianz im Bereich von 5.395 und 6736.218, mit einem Mittel von 1437.746

Aufgrund der großen Diskrepanz zwischen der Varianz der Eingabe in und der Ausgabe aus dem Netz, gehe ich davon aus, dass dies das Training behindert.

Nach [16] ist es empfehlenswert, die Wertebereiche auszunutzen, wobei durch die im ResNet vielfach verwendeten Batch Normalizations dieser immer wieder im Schnitt auf 0 zentriert von -1 bis 1 verläuft. Um die Varianz ähnlich zu halten, suche ich also eine Lösung, die dabei auf diesen Wertebereich abzielt.

Auf der Suche nach einer Initialisierung, die diese Kriterien erfüllt, lasse ich mit dem Skript im Folgenden empirisch nach einer möglichen Lösung suchen. Konkret richte ich die Suche nach einer uniform verteilten Initialisierung für w' und σ^2, inspiriert vom Vergleich von von uniformen und normalverteilten Initialisierungen in [16]. Dabei sind die beiden über die jeweils konfigurierte Stärke der Kontrastnormalisierung gekoppelt, die wie unter Kapitel 5.1.3 beschrieben berechnet wird. Dadurch wird die Suche vereinfacht und eine Initialisierung für eine gewünschte Stärke der Kontrastnormalisierung als Ausgangspunkt ermittelt.

Durch eine zuerst grobe Suche für eine Initialisierung, wie sie unter Abbildung 7.12 gezeigt ist, kann ich den Bereich eingrenzen, in dem eine detailliertere Suchen von Nöten ist.

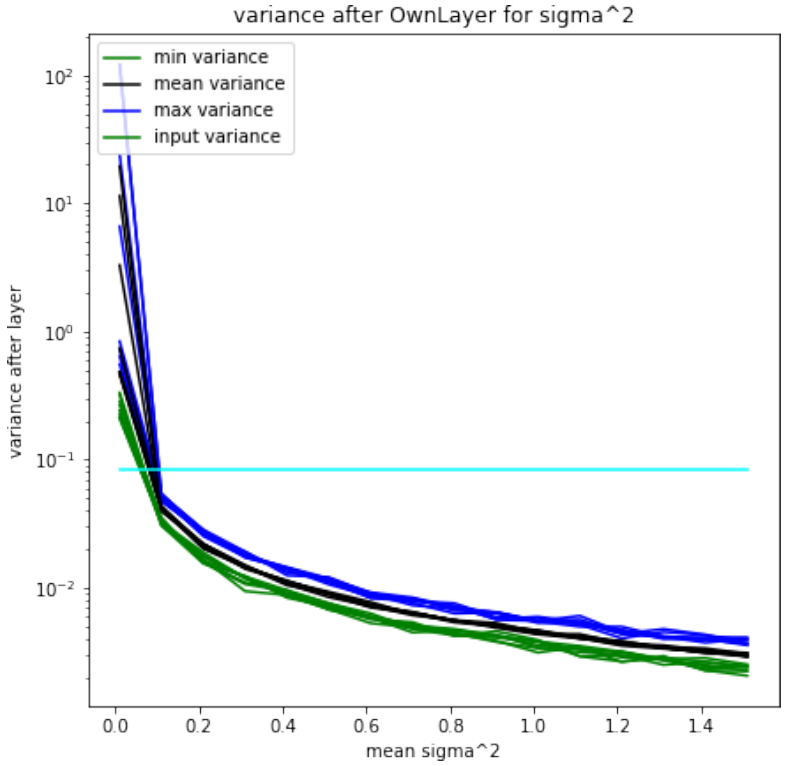


Abbildung 7.12: Analyse der Varianz nach der Schicht bei einem uniformverteilten Rauschen als Eingang

Die Abbildung zeigt dabei die Suche für einen geeigneten Mittelwert und Bereich von σ^2 für eine Stärke der Kontrastnormalisierung von 0.5. Durch eine anschließende detailliertere Suche, dargestellt unter Abbildung 7.13 in einem nun genauer bekannten Umfeld erhalte ich eine Spezifikation für σ^2, welche die Änderung der Varianz durch die kontrastnormalisierende Schicht bei uniformem Rauschen gering hält.

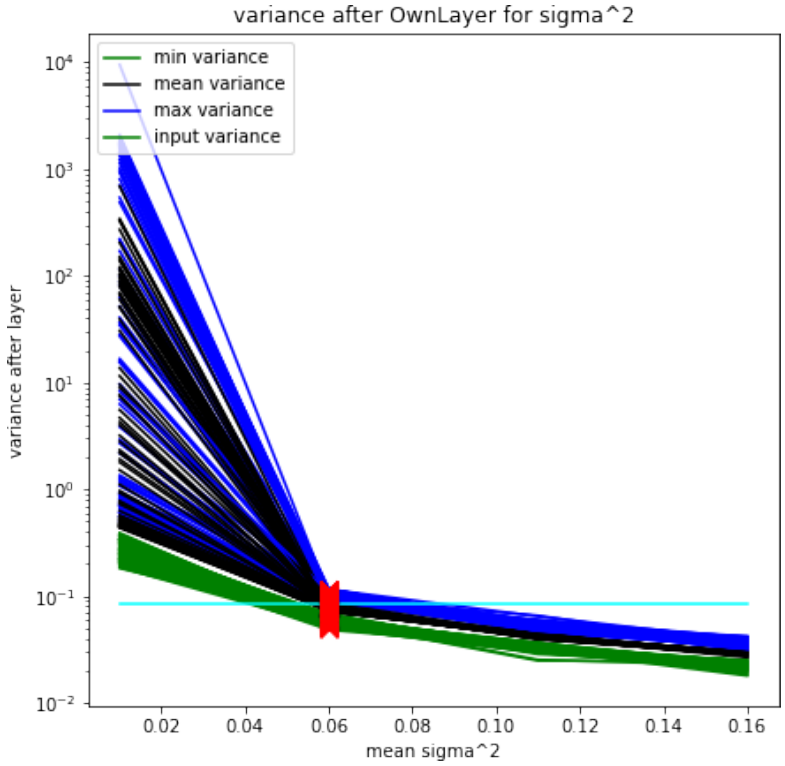


Abbildung 7.13: Detaillierte Analyse der Varianz nach der Schicht bei einem uniformverteilten Rauschen als Eingang

Von den markierten Werten ist die geringste Spreizung der Varianz um den gewünschten Wert zu erreichen mit einem mittleren σ^2 von $0,06$ und einer Spreizung von $0,01$. Damit liegt die kleinste Varianz in den Ausgabekanälen der Schicht bei etwa $0,061$, die mittlere bei $0,076$ und die maximale bei $0,093$. . Diese Werte sind dabei sehr nahe dem gewünschten Wert von $\frac{1}{12} = 0,08\overline{3}$

Das an dieser Stelle thematisierte Problem der Veränderung der Varianz tritt dabei für die unter Abschnitt 7.5.1 beschriebene Initialisierung mit teilweise vortrainierten Gewichten bei der Betrachtung der Schicht als Gesamtes nicht direkt auf, da in diesem Fall die Schicht zu Beginn wie die damit ersetze Convolution wirkt. Die Gradienten bewirken entsprechend keine stark ansteigenden Werte, sodass kein unmittelbarer Anstieg des Loss auftaucht.

7.6 Learning Rate

Die Learning Rate hat Einfluss darauf, wie schnell ein Training eines neuronalen Netzes „gute“ Ergebnisse liefert. Dabei ist ein Abwägen erforderlich, da eine zu große Learning Rate dazu führen kann, dass das Training sehr instabil verläuft und kein Optimum gefunden werden kann.

Im Folgenden vergleiche ich verschiedene Learning Rates auf ihre Eignung für den Einsatz im Rahmen dieser Arbeit.

Der Standardwert für den hier verwendeten Optimizer Adam beträgt 0.001, Abbildung 7.14 zeigt ein Training des unmodifizierten Netzes mit diesem.

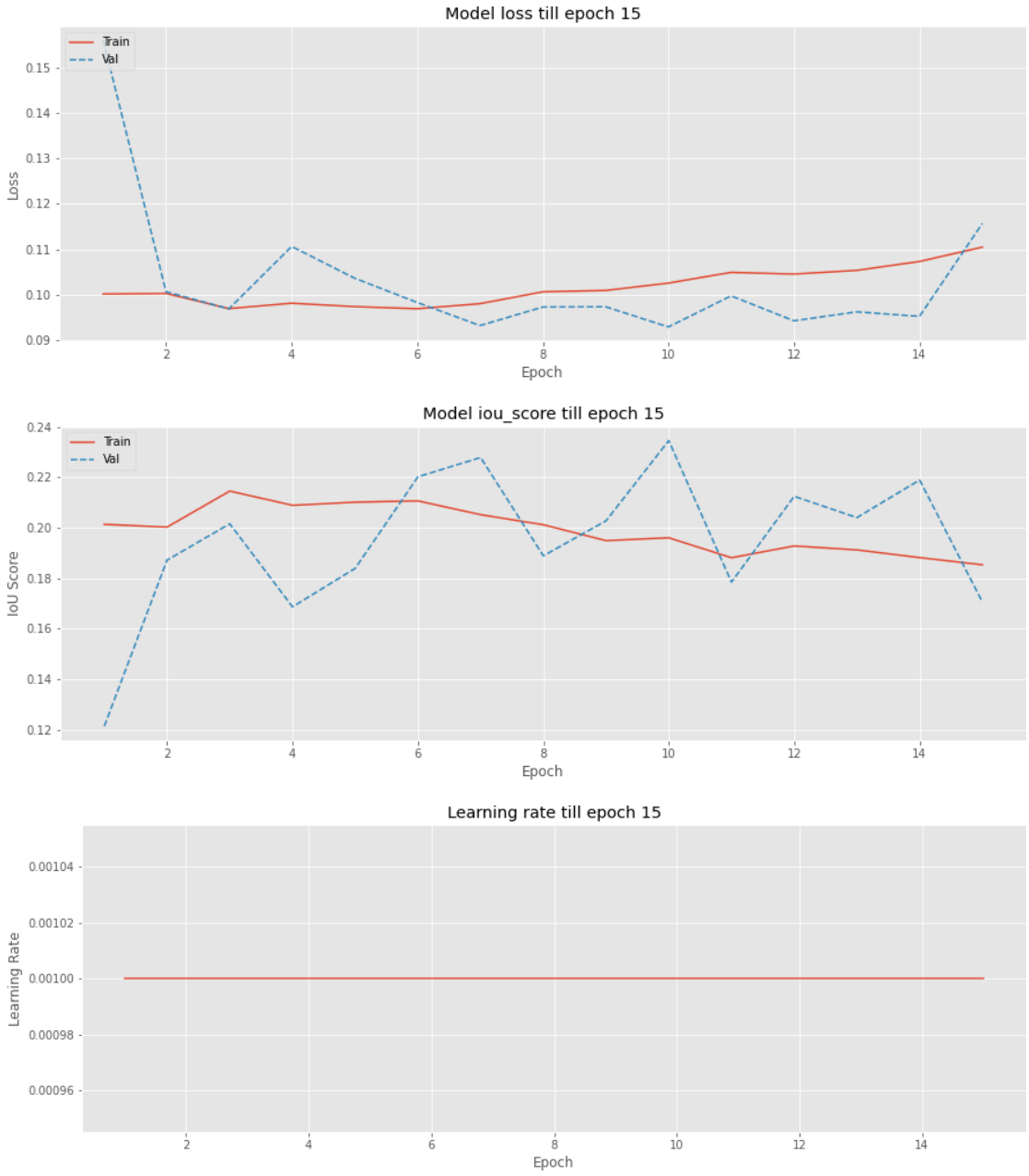


Abbildung 7.14: Training des unmodifizierten Netzes mit der kontrastnormalisierenden Schicht mit weicher Beleuchtung und einer Learning Rate von 0.001

Ein Wert von 0.0001, wie er im Beispiel zur Semantic Segmentation für mehrere Klassen in [36] verwendet ist, führt unter ansonsten gleichen Vorraussetzungen zu dem Verlauf in Abbildung 7.15.

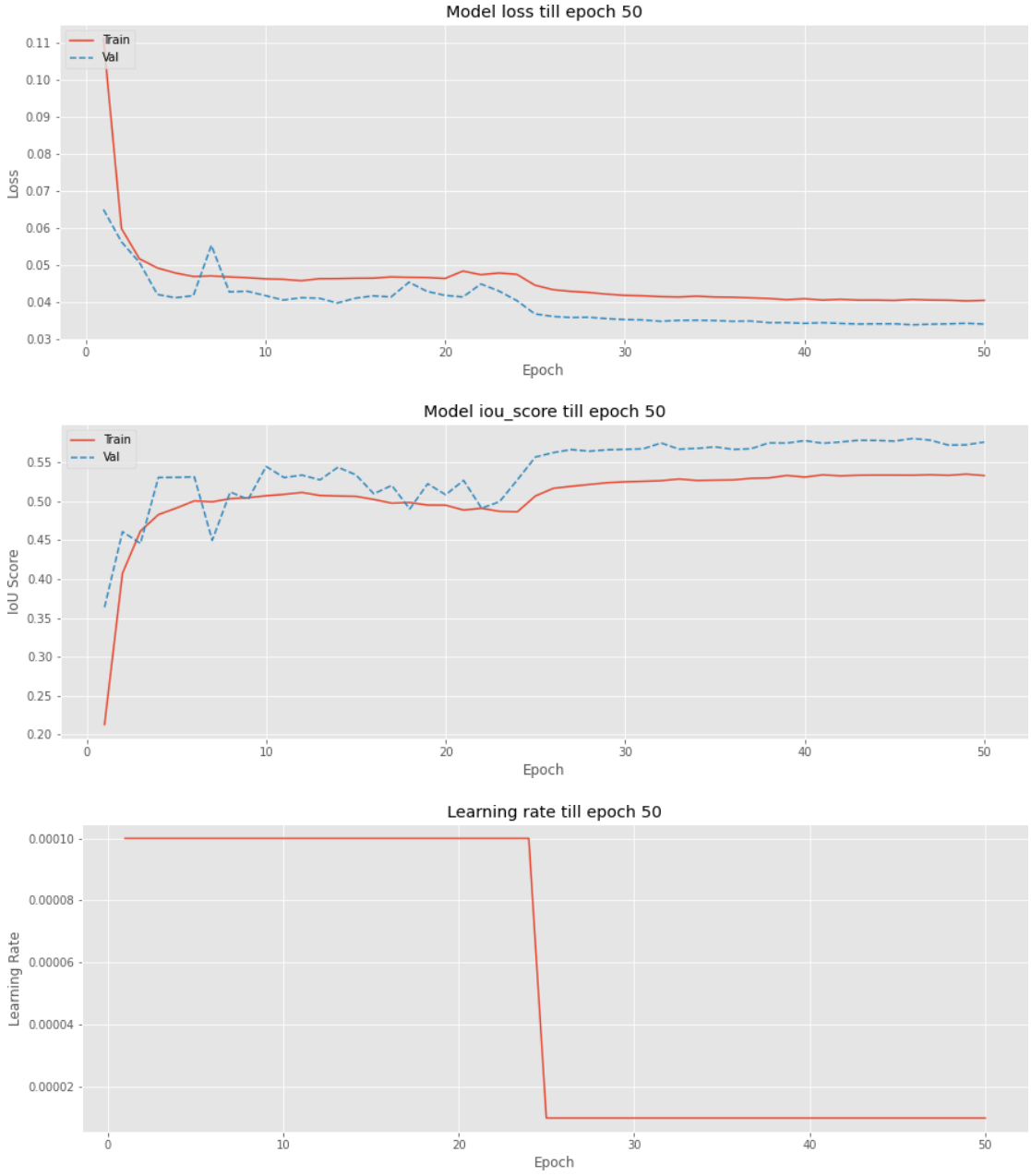


Abbildung 7.15: Training des unmodifizierten Netzes mit der kontrastnormalisierenden Schicht mit weicher Beleuchtung und einer Learning Rate von 0.0001

Ein Training des modifizierten Netzes mit den gleichen Bedingungen resultiert in dem Verlauf unter Abbildung 7.16.

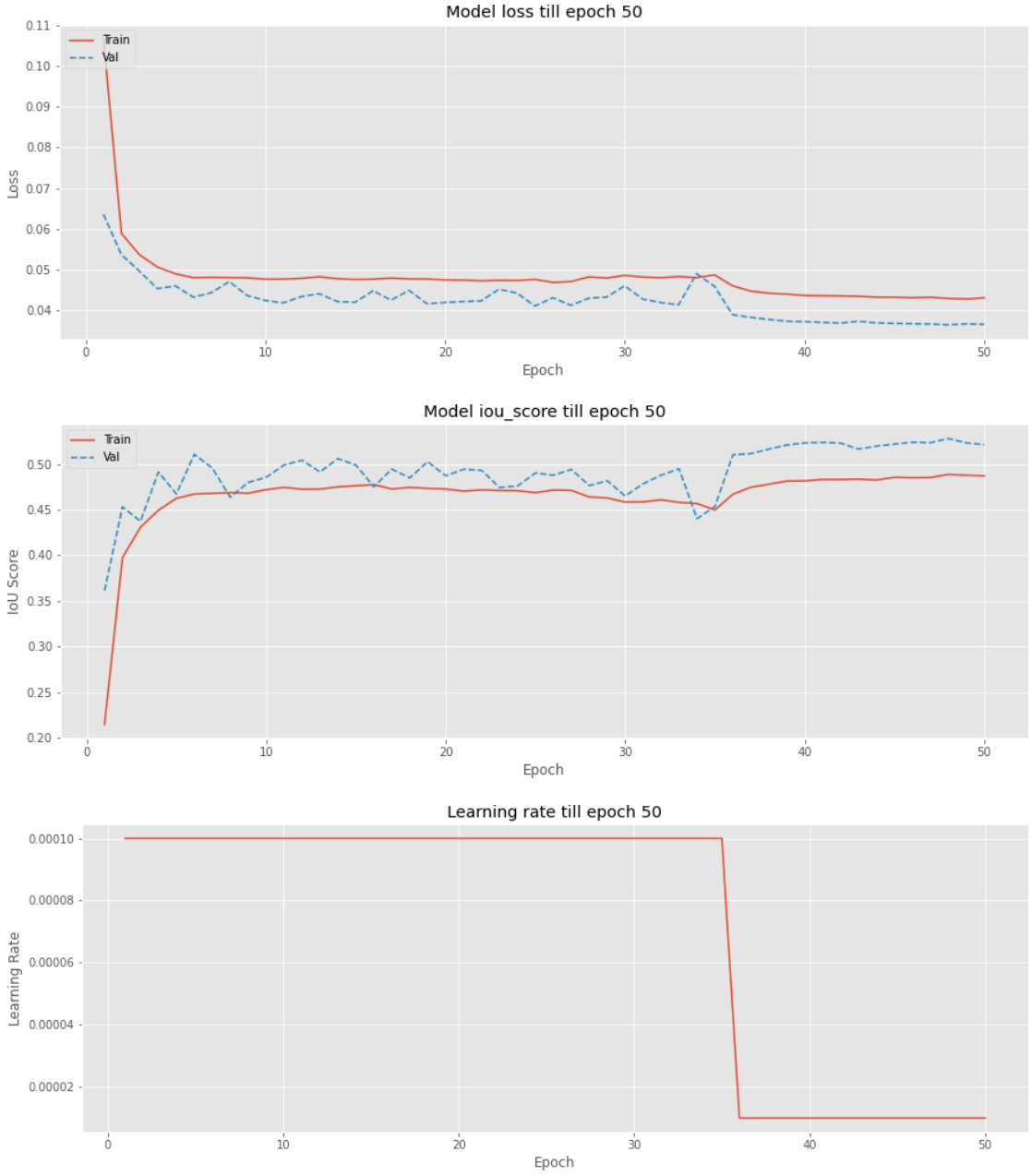


Abbildung 7.16: Training des modifizierten Netzes mit der kontrastnormalisierenden Schicht mit weicher Beleuchtung und einer Learning Rate von 0.0001

Durch weiteres Senken auf eine Learning Rate von 0.00001 ergibt sich für das unmodifizierte Netz der Trainingsverlauf, wie in Abbildung 7.9 (unter Abschnitt 7.5.2) dargestellt. Für das modifizierte Netz entsteht der Ablauf unter Abbildung 7.17

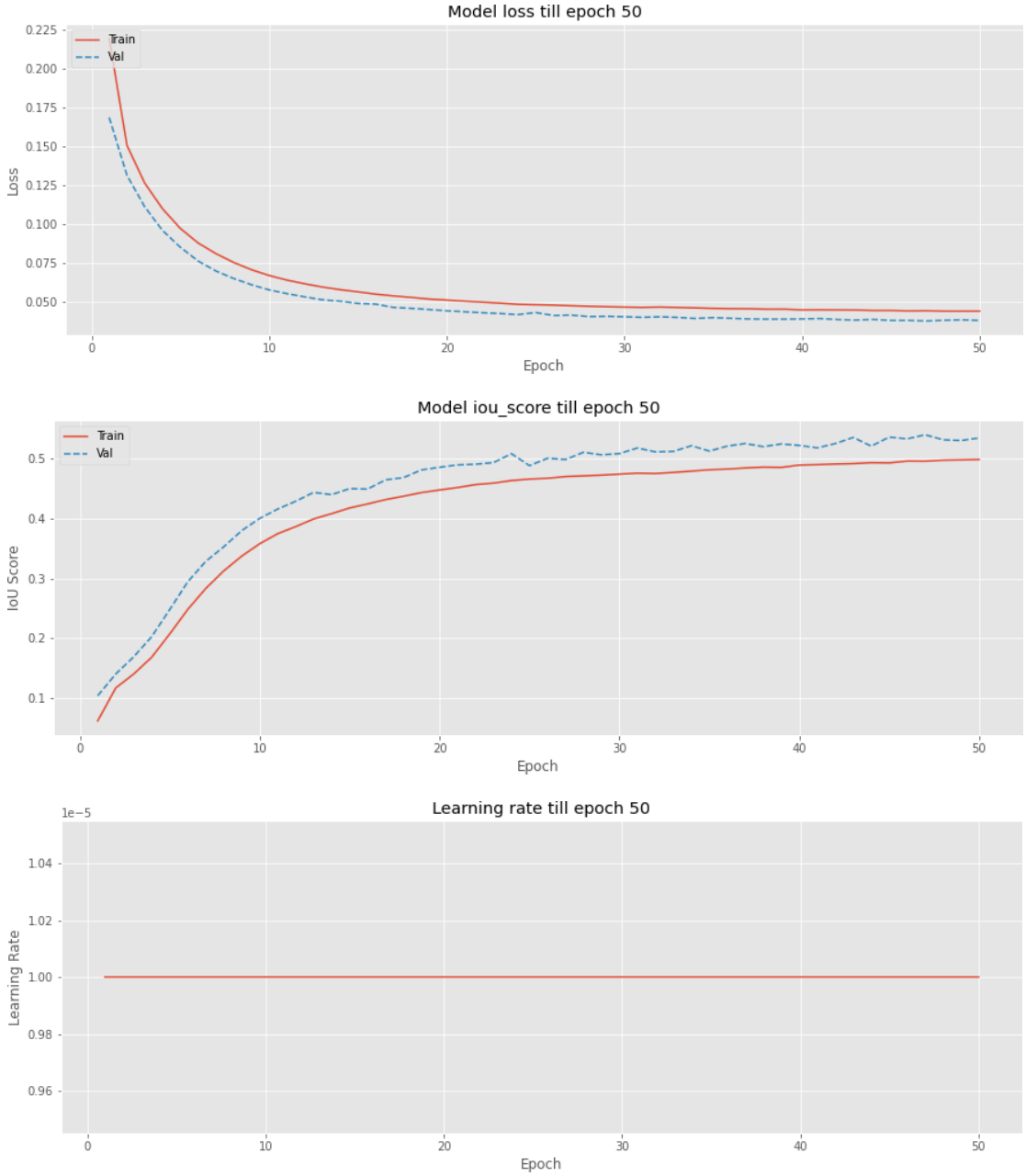


Abbildung 7.17: Training des modifizierten Netzes mit der kontrastnormalisierenden Schicht mit weicher Beleuchtung und einer Learning Rate von 0.0001

Ein Vergleich dieser Verläufe zeigt, dass ein Wert von 0.001 zu hoch für gute Resultate in diesem Rahmen ist.

Durch weitere hier nicht detailliert aufgeführte Experimente gelange ich zu der Beobachtung, dass eine Learning Rate von 0.001, die im Verlauf des Trainings durch die Plateauerkennung gesenkt wird, schlechtere Ergebnisse liefert als von Anfang an eine Learning Rate von 0.001 zu verwenden.

Die Verwendung von 0.0001 als Learning Rate führt durch die Plateauerkennung

bei dem unmodifizierten Netz zum niedrigsten Loss der Validation nach 50 Epochen, jedoch dauert es bei dem modifizierten Netz länger, bis ein Plateau erkannt wird. Dies erschwert die Vergleichbarkeit, da hier verschiedene Sichten möglich sind. Man könnte die Ergebnisse jeweils beim Eintritt eines Plateaus der Gradienten vergleichen, jedoch wären dann auch die in beiden Varianten identischen Schichten im modifizierten Netz länger trainiert. Es ist allerdings dabei zu bemerken, dass an dieser Stelle auch dann keine sichtbaren Verbesserungen im Training auf dem weichen Datensatz zu erkennen sind.

Eine Learning Rate von 0.00001 führt zu einer stabileren Kurve, die damit die Vergleichbarkeit verbessert, auch wenn die Endergebnisse hinter denen mit einer Learning Rate von 0.0001 liegen.

7.6.1 Vergleich mit harter Beleuchtung

Ein Evaluation auf den Testdatensatz mit harter Beleuchtung ergibt folgende Ergebnisse:

Netz-Typ	Learning Rate	Loss	mittlere IoU	mittlere F1
unmodifiziertes Netz	0.00001	0.0690	0.3979	0.5030
modifiziertes Netz	0.00001	0.0887	0.3373	0.4724
unmodifiziertes Netz	0.0001	0.0547	0.4646	0.5821
modifiziertes Netz	0.0001	0.0585	0.4158	0.5270

Aus einem Vergleich der Resultate für je eine Learning Rate miteinander lässt sich erkennen, dass keine Verbesserung vorliegt.

7.6.2 Training des modifizierten Netzes mit Transfer Learning

Ein Training des unmodifizierten Netzes für 15 Epochen ergibt die unter Abbildung 7.18 zu sehenden Verlauf.

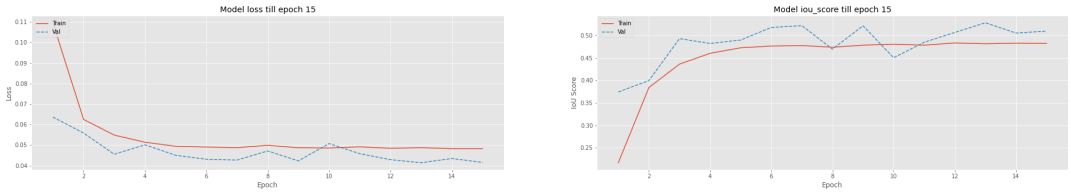


Abbildung 7.18: Trainingsverlauf des unmodifizierten Netzes für 15 Epochen

Aufbauend auf den resultierenden Gewichten folgt ein Training für weitere 15 Minuten, der gesamt resultierende Verbrauch steht unter Abbildung 7.19. Dabei nutze ich eine durch Analyse der Varianz optimierte Initialisierung, mit einer Stärke der Kontrastnormalisierung von $\frac{2}{3}$

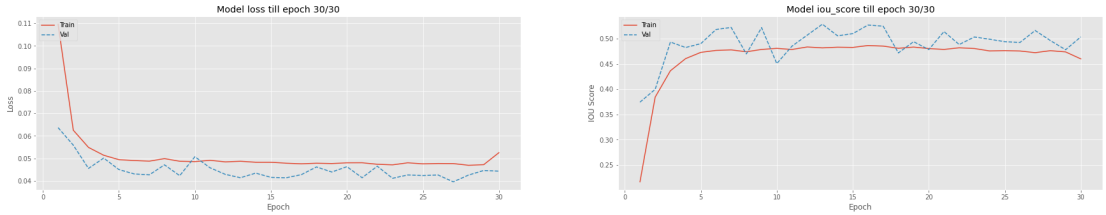


Abbildung 7.19: Trainingsverlauf des modifizierten Netzes aufbauend auf 7.18, verlängert um weitere 15 Epochen

Aus den jeweils besten Gewichten ergeben sich die folgenden Ergebnisse:

Netz-Typ	Anzahl Epochen	Test auf	Loss	IoU	F1
unmodifiziertes	15	weiche Beleuchtung	0.0411	0.5300	0.6347
modifiziertes	extra 15	weiche Beleuchtung	0.0394	0.5162	0.6256
unmodifiziertes	15	harte Beleuchtung	0.0691	0.4076	0.5199
modifiziertes	extra 15	harte Beleuchtung	0.0675	0.3786	0.4920

Daraus ersichtlich ist, das auch mit einem fortgesetzten Training keine Verbesserung erzielt werden kann.

7.7 Verbesserungsansätze

Diverse mögliche Probleme und Verbesserungen sind für mich denkbar, die ich im Folgenden nenne:

7.7.1 Für den Datensatz

Es ist denkbar, dass der Datensatz nicht so gut wie angenommen für das hier betrachtete Problem ist. Die Varianz des Hintergrunds ist durch die im gesamten Sun Temple verwendeten Fliesen eher eingeschränkt, eine Kombination mit weiteren Umgebungen würde diese erhöhen. Da nur eine relative Betrachtung erfolgt, sollten diese Auswirkungen keine grundlegenden Trainingsprobleme verursachen, jedoch scheint auch das normale Netz noch einigermaßen mit der harten Beleuchtung umgehen zu können. Ein Datensatz mit noch härterer Beleuchtung könnte das Problem verschärfen und möglicherweise andere Resultate liefern.

7.7.2 Trainingsoptimierungen

Um das Training zu optimieren, sind sowohl inhaltliche Aspekte denkbar, als auch solche, die auf die Implementierung mit Tensorflow bezogen sind.

Ein besseres Ergebnis der Metriken könnte erreicht werden, wenn ein Weg gefunden wird, direkter nach diesen zu optimieren. Dabei ist das Problem, dass übliche Metriken nicht direkt ableitbar sind und der Optimizer daher für diese keine Gradienten berechnen kann, deshalb müssen seperate Funktionen als Loss verwendet werden, die ableitbar sind.

Am Beispiel der F1 Metrik könnte dies wie unter [18] durch Anpassungen umgangen werden, durch die eine ähnliche, ableitbare Funktion erzeugt wird, die einen direkten

Zusammenhang mit der Metrik besitzt. Da ich in Einzelfällen während meiner Experimente beobachte, dass relative Änderungen am Loss nicht im gleichen Verhältnis Auswirkungen auf die Metriken haben, könnte dies die Zuverlässigkeit der Ergebnisse verbessern.

Beim Training des modifizierten Netzes mit den hier verwendeten Versionen fällt auf, dass die Depthwise Convolution für eine deutliche Verlangsamung sorgt. Dabei tritt dies nur während des Trainings, nicht aber während der Vorhersage auf. Eine Überprüfung der Ressourcenauslastung ergibt, dass der Grad der parallelen Berechnung deutlich eingeschränkt ist gegenüber der Verwendung normaler Convolutions. Eine Verbesserung ist möglich durch die Darstellung der Depthwise Convolution als eine normale Convolution, wie sie unter **??** gezeigt wird. Eine direkte Optimierung des Codes wäre dem gegenüber zu bevorzugen, sofern technisch möglich, da mit der dort beschriebenen Methode für die wiederum unnötige Rechnungen entstehen.

7.7.3 Für die Anwendung auf ein unmodifiziertes Netz

Ein logarithmischer Histogrammausgleich kann verwendet werden, um Bilder mit harter Beleuchtung etwas auszugleichen. Es wäre denkbar, dass entsprechend vorverarbeitete Bilder in einem Convolutional Neural Network, was nicht auf diese trainiert ist, besser verarbeitet werden können. Dazu könnte man, beispielsweise im Rahmen der „tf.data“ API von Tensorflow, die Bilder vor der Verwendung normalisieren. Durch eine Justierung der Stärke des Ausgleichs könnten die Bilder mit harter Beleuchtung an die mit weicher Beleuchtung etwas angeglichen werden, wodurch das Problem limitiert sein könnte. Dabei wäre es für mich interessant zu vergleichen, welche Ergebnisse sich abgängig davon erzielen lassen, ob das Training auf dem Datensatz mit weicher Beleuchtung mit oder ohne einen logarithmischen Histogrammausgleich erfolgt. Darauf aufbauend könnte potentiell eine entsprechende allgemeine Vorverarbeitung Verbesserungen geben. Einer der zentralen Unterschiede zu meinem in dieser Arbeit erwünschten Ziel wäre eine feste Normalisierung - mein Ansatz hier ist, dem neuronalen Netz den Grad der Normalisierung pro Kanal zum freien Training verfügbar zu machen.

Kapitel 8

Fazit

Es war mir nicht möglich, einen Vorteil des modifizierten Netzes nachzuweisen. Dabei habe ich Wert darauf gelegt, eine möglichst gute Vergleichsbasis zu schaffen, um nicht aussagekräftige Ergebnisse zu vermeiden. Daraus ist zu schließen, dass die kontrastnormalisierende Schicht in der hier vorgestellten Struktur für das U-Net in Verbindung mit einem ResNet keine Vorteile liefert. Es ist denkbar, dass moderne Netzarchitekturen insgesamt keinen direkten Vorteil durch Kontrastnormalisierung innerhalb ihrer Schichten erhalten können.

Ausblick

Eine genauere mathematische Analyse einer optimalen Initialisierung der kontrastnormalisierenden Schicht dürfte maßgeblich dabei helfen, das Training mit dieser zu verbessern. Mit den hier vorgestellten Ansätzen verläuft der Prozess des Trainings langsamer, möglicherweise verhindert dies auch bessere Resultate. Eine ungünstige Initialisierung anderer Netze kann auch zu schlechten Ergebnissen führen. Eine Analyse moderner Netze auf ihre Empfindlichkeit gegenüber Stärken der Kontraste in einem Datensatz kann dabei helfen, eine Aussage darüber zu treffen, ob meine oder eine ähnliche Kontrastnormalisierung dort sinnvoll anzuwenden ist.

Eine weitere Überlegung wäre, einen logarithmischen Histogrammausgleich auf dem Datensatz auszuführen, bevor er in das Netz eingespeist wird. Dies könnte dazu führen, dass keine Kontrastnormalisierung innerhalb des neuralen Netzes notwendig ist.

Quellenverzeichnis

[1] Aqeel Anwar. „Difference between Local Response Normalization and Batch Normalization“. *towards data science* (Juni 2019). URL: https://towardsdatascience.com/difference-between-local-response-normalization-and-batch-normalization-272308c034ac?gi=cee2663f8a5d (besucht am 15. 07. 2020) (siehe S. 5).

[2] Herbert Bay, Tinne Tuytelaars und Luc Van Gool. „Surf: Speeded up robust features“. In: *Computer Vision – ECCV 2006*. Springer. Springer Berlin Heidelberg, 2006, S. 404–417 (siehe S. 2, 5).

[3] G. Bradski. „The OpenCV Library“. *Dr. Dobb's Journal of Software Tools* (2000) (siehe S. 23).

[4] Berk Calli u. a. „The ycb object and model set: Towards common benchmarks for manipulation research“. In: *2015 international conference on advanced robotics (ICAR)*. IEEE. 2015, S. 510–517 (siehe S. 8, 25, 30).

[5] N Chinchor. „Proceedings of the 4th Conference on Message Understanding“ (1992) (siehe S. 24).

[6] *CS231n Convolutional Neural Networks for Visual Recognition*. Stanford University. Kap. Normalization Layer. URL: https://cs231n.github.io/convolutional-networks/#norm (besucht am 15. 07. 2020) (siehe S. 2, 6).

[7] Navneet Dalal und Bill Triggs. „Histograms of oriented gradients for human detection“. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Bd. 1. Ieee. 2005, 886–893 vol. 1 (siehe S. 2, 5).

[8] Jia Deng u. a. „ImageNet: A large-scale hierarchical image database“. *2009 IEEE Conference on Computer Vision and Pattern Recognition* (2009), S. 248–255 (siehe S. 29).

[9] Terrance DeVries und Graham W. Taylor. *Improved Regularization of Convolutional Neural Networks with Cutout*. 2017. arXiv: 1708.04552 [cs.CV] (siehe S. 31).

[10] Leo Dirac. *timebudget*. https://github.com/leopd/timebudget. 2019 (siehe S. 23).

[14] Epic Games. *An overview of the Post Processing example level, example 1.10: Global Illumination*. URL: https://docs.unrealengine.com/4.27/en-US/Resources/ContentExamples/PostProcessing/1_10/ (besucht am 03. 11. 2021) (siehe S. 8).

[11] Epic Games. *Unreal Engine*. Version 4.24. URL: https://www.unrealengine.com (siehe S. 8).

[15] Epic Games. *Unreal Engine Sun Temple, Open Research Content Archive (ORCA)*. Okt. 2017. URL: https://developer.nvidia.com/ue4-sun-temple (siehe S. 8).

[12] Geoff French u. a. *Semi-supervised semantic segmentation needs strong, varied perturbations*. 2020. arXiv: 1906.01916 [cs.CV] (siehe S. 31).

[13] U. Frese u. a. „A Contrast Normalized Gradient Detection Measure applied to Circle Detection“. Unpublished manuscript. 2011 (siehe S. 2, 13–15).

[16] Daniel Godoy. *Hyper-parameters in Action! Part II — Weight Initializers*. https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404. Juni 2018 (siehe S. 36, 37).

[17] Shoray Goel. *Kaiming He initialization*. https://medium.com/@shoray.goel/kaiming-he-initialization-a8d9ed0b5899. Juli 2019 (siehe S. 36).

[18] Michal Haltuf. *Best loss function for F1-score metric*. https://www.kaggle.com/rejpalcz/best-loss-function-for-f1-score-metric. 2018 (siehe S. 45).

[19] Charles R. Harris u. a. „Array programming with NumPy“. *Nature* 585.7825 (Sep. 2020), S. 357–362. URL: https://doi.org/10.1038/s41586-020-2649-2 (siehe S. 23).

[20] Kaiming He u. a. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV] (siehe S. 21).

[21] J. D. Hunter. „Matplotlib: A 2D graphics environment“. *Computing in Science & Engineering* 9.3 (2007), S. 90–95 (siehe S. 23).

[22] Liam Kinney, Casey Chu und Maneesh Apte. „Smart Initialization Yields Better Convergence Properties in Deep Abstractive Summarization“ () (siehe S. 17, 18).

[23] *Knet.jl 0.7.2 documentation*. Istanbul, Turkey: Koç University. Kap. Convolutional Neural Networks. URL: https://knet.readthedocs.io/en/latest/cnn.html#normalization (besucht am 15. 07. 2020) (siehe S. 2, 6).

[24] Tsung-Yi Lin u. a. „Microsoft coco: Common objects in context“. In: *European conference on computer vision*. Springer. 2014, S. 740–755 (siehe S. 7).

[25] David G Lowe. „Distinctive image features from scale-invariant keypoints“. *International journal of computer vision* 60.2 (2004), S. 91–110 (siehe S. 2, 5).

[26] Martin Abadi u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (siehe S. 23).

[27] Shashank N. Mathur, Anil K. Ahlawat und Virendra P. Vishwakarma. *Illumination Invariant Face Recognition using Supervised and Unsupervised Learning Algorithms*. Version 11830. Juli 2008. URL: https://doi.org/10.5281/zenodo.1078015 (besucht am 15. 07. 2020) (siehe S. 5).

[28] Sverker Nilsson und YiFei Zhu. *Guppy 3*. https://github.com/zhuyifei1999/guppy3. 2019 (siehe S. 23).

[29] Hamid Rezatofighi u. a. „Generalized intersection over union: A metric and a loss for bounding box regression“. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, S. 658–666 (siehe S. 24).

[30] Albino Riganello u. a. *GPRename*. Version 20210405. 5. Apr. 2021. URL: http://gprename.sourceforge.net (siehe S. 22).

[31] Olaf Ronneberger, Philipp Fischer und Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV] (siehe S. 13).

[32] Yutaka Sasaki u. a. „The truth of the f-measure. 2007“. *URL: https://www. cs. odu. edu/˜ mukka/cs795sum09dm/Lecturenotes/Day3/F-measure-YS-26Oct07. pdf* (2007) (siehe S. 24).

[33] Thang To u. a. *NDDS: NVIDIA Deep Learning Dataset Synthesizer*. https://github.com/NVIDIA/Dataset_Synthesizer . 2018 (siehe S. 7, 22).

[34] Jonathan Tremblay, Thang To und Stan Birchfield. *Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation*. 2018. arXiv: 1804.06534 [cs.CV] (siehe S. 7, 20, 29).

[35] Pavel Yakubovskiy. *Classification Models*. https://github.com/qubvel/classification_models. 2019 (siehe S. 23).

[36] Pavel Yakubovskiy. *Segmentation Models*. https://github.com/qubvel/segmentation_models. 2019 (siehe S. 23, 24, 30, 40).

[37] Jun-Yong Zhu, Wei-Shi Zheng und Jian-Huang Lai. „Logarithm Gradient Histogram: A general illumination invariant descriptor for face recognition“. In: *2013 10th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition, FG 2013*. ISBN: 978-1-4673-5545-2. Apr. 2013, S. 1–8. URL: https://doi.org/10.1109/FG.2013.6553738 (besucht am 15. 07. 2020) (siehe S. 5).